Algorithms and Techniques for Efficient and Effective Nearest Neighbours Classification

Stefanos Ougiaroglou

Ph.D. Dissertation

Supervised by
Georgios Evangelidis

Supported by
State Scholarships Foundation of Greece (I.K.Y.)

Submitted to
Department of Applied Informatics
School of Information Sciences
University of Macedonia

Thessaloniki, Greece
June 2014

Advisory committee

Georgios Evangelidis (supervisor), Professor,
Department of Applied Informatics, University of Macedonia, Greece

Dimitiris A. Dervos, Professor,
Department of Information Technology, Alexander T.E.I. of Thessaloniki, Greece

Jose Aldana F. Montes, Professor,
Departamento de Lenguajes y Ciencias de la Computación, University of Malaga, Spain

Examination committee

Dimitiris A. Dervos, Professor,
Department of Information Technology, Alexander T.E.I. of Thessaloniki, Greece

Georgios Evangelidis, Professor,
Department of Applied Informatics, University of Macedonia, Greece

Georgia Koloniari, Lecturer,
Department of Applied Informatics, University of Macedonia, Greece

Konstantinos G. Margaritis, Professor,
Department of Applied Informatics, University of Macedonia, Greece

Jose Aldana F. Montes, Professor,
Departamento de Lenguajes y Ciencias de la Computación, University of Malaga, Spain

Apostolos N. Papadopoulos, Assistant Professor,
Department of Computer Science, Aristotle University, Greece

Nikolaos Samaras, Associate Professor,
Department of Applied Informatics, University of Macedonia, Greece

Abstract

Although the k-NN classifier is considered to be an effective classification algorithm, it has some major weaknesses that may render its use inappropriate for some application domains and / or datasets. The first one is the high computational cost involved (all distances between each unclassified item and all training data must be computed). Although nowadays systems are equipped with powerful processors, in cases of large datasets, this drawback renders the classification a time-consuming and in some cases a prohibitive procedure. Another weakness is the high storage requirements for maintaining the training data. Eager classifiers (e.g., decision tress, neural networks) can discard the training data after the construction of the classification model in order to save space. In contrast, the k-NN classifier must have all the training data always available. Moreover, the classification accuracy achieved by the classifier depends on the quality of the available training data. Noisy and mislabelled data, as well as outliers and overlaps between data regions of different classes may mislead the algorithm and affect the classification accuracy.

The aforementioned weaknesses constitute an active research problem. The dissertation is motivated by these weaknesses and tries to remedy the problem. Therefore, it contributes novel algorithms and techniques that can effectively deal with the aforementioned weaknesses. In other words, it proposes algorithms and techniques for efficient and effective k-NN classification. The contributions are distinguished into three main categories: (i) new data reduction techniques that deal with all the weak points of the classifier and avoid the limitations and disadvantages of existing data reduction techniques, (ii) novel hybrid algorithms that combine different types of speed-up techniques and that can effectively reduce the computational cost of the classifier, and, (iii) improvements and experimentations for existing algorithms.

The proposed algorithms, techniques and improvements are evaluated on several datasets and experimentally compared to state-of-the-art methods. The experimental measurements are validated by statistical tests of significance. The results illustrate that the proposed methods satisfy the goals for which they were developed and lead to improved classification, in terms of accuracy, preprocessing and computational cost.

keywords: *k nearest neighbours, classification, clustering, data reduction, prototypes, prototype selection and abstraction, condensing, editing, cluster-based methods, hybrid algorithms, streaming / dynamic environments, time-series, preprocessing, computational cost, accuracy.*

Περίληψη

Ο κατηγοριοποιητής εγγύτερων γειτόνων είναι ένας αποτελεσματικός αλγόριθμος κατηγοριοποίησης. Ωστόσο, περιλαμβάνει μειονεκτήματα και αδυναμίες που τον καθιστούν ακατάλληλο σε συγκεκριμένα πεδία εφαρμογής ή/και σύνολα δεδομένων. Το πρώτο μειονέκτημα είναι το υψηλό κόστος κατηγοριοποίησης ως αποτέλεσμα του υπολογισμού των αποστάσεων μεταξύ κάθε αντικείμενου προς κατηγοριοποίηση και όλων των αντικειμένων που ανήκουν στο σύνολο εκπαίδευσης. Αν και τα σημερινά υπολογιστικά συστήματα είναι εφοδιασμένα με ισχυρούς επεξεργαστές, σε περιπτώσεις μεγάλων συνόλων δεδομένων, το συγκεκριμένο μειονέκτημα καθιστά την κατηγοριοποίηση μια ιδιαίτερα χρονοβόρα διαδικασία, η εκτέλεση της οποίας μπορεί να είναι απαγορευτική. Το δεύτερο μειονέκτημα αφορά τις μεγάλες απαιτήσεις σε αποθηκευτικό χώρο. Κατηγοριοποιητές που βασίζονται σε μοντέλα κατηγοριοποίησης (π.χ., δένδρα απόφασης, νευρωνικά δίκτυα) μπορούν μετά την κατασκευή του μοντέλου να διαγράψουν τα δεδομένα εκπαίδευσης ώστε να εξοικονομήσουν χώρο. Αντίθετα, ο κατηγοριοποιητής εγγύτερων γειτόνων πρέπει να έχει πάντα όλα τα δεδομένα εκπαίδευσης διαθέσιμα. Έτσι δεν είναι δυνατή η εξοικονόμηση αποθηκευτικού χώρου. Τέλος, η ακρίβεια που επιτυγχάνει ο κατηγοριοποιητής εγγύτερων γειτόνων εξαρτάται από την ποιότητα των δεδομένων εκπαίδευσης. Δεδομένα με θόρυβο, αντικείμενα χωρίς ετικέτα κλάσης, ακραία σημεία και επικαλύψεις στις περιοχές διαφορετικών κλάσεων αποπροσανατολίζουν τον κατηγοριοποιητή με αποτέλεσμα τη μείωση της ακρίβειας.

Τα μειονεκτήματα αυτά αποτελούν μια ενεργή περιοχή έρευνας. Η διδακτορική διατριβή έχει ως κίνητρο την αντιμετώπιση των συγκεκριμένων μειονεκτημάτων. Ως εκ τούτου, η διατριβή συνεισφέρει καινοτόμους αλγόριθμους που αντιμετωπίζουν με αποτελεσματικό τρόπο τα μειονεκτήματα αυτά. Με άλλα λόγια, η διατριβή προτείνει αλγόριθμους και τεχνικές αποτελεσματικής κατηγοριοποίησης εγγύτερων γειτόνων. Η συνεισφορά έχει χωριστεί σε τρεις κατηγορίες: (i) νέες τεχνικές μείωσης όγκου των δεδομένων εκπαίδευσης που αντιμετωπίζουν όλα τα μειονεκτήματα και δεν παρουσιάζουν τις αδυναμίες υπαρχουσών τεχνικών, (ii) υβριδικούς αλγορίθμους που συνδυάζουν διαφορετικού τύπου μεθόδους επιτάχυνσης με στόχο την μείωση του υπολογιστικού κόστους της κατηγοριοποίησης (iii) βελτιώσεις σε υπάρχουσες τεχνικές και πειραματικές μελέτες.

Η απόδοση των προτεινόμενων αλγόριθμων, τεχνικών και βελτιώσεων ελέγχθηκε πειραματικά και συγκρίθηκε με γνωστές στη βιβλιογραφία μεθόδους χρησιμοποιώντας διάφορα σύνολα δεδομένων. Οι πειραματικές μετρήσεις επικυρώθηκαν με το μη παραμετρικό στατιστικό τεστ του Wilcoxon. Τα αποτελέσματα υποδεικνύουν ότι οι αλγόριθμοι, οι τεχνικές και οι βελτιώσεις επιτυγχάνουν τον σκοπό για τον οποίο αναπτύχθηκαν και ότι οδηγούν σε αποτελεσματική κατηγοριοποίηση σε ότι αφορά την ακρίβεια, το κόστος κατηγοριοποίησης και το κόστος προ-επεξεργασίας.

Λέξεις κλειδιά: *κ εγγύτεροι γείτονες, κατηγοριοποίηση, συσταδοποίηση, μείωση όγκου δεδομένων, αντιπρόσωποι, επιλογή αντιπροσώπων, δημιουργία αντιπροσώπων, συμπύκνωση, επεξεργασία με σκοπό τη μείωση θορύβου, μέθοδοι βασισμένοι στη συσταδοποίηση, Υβριδικοί αλγόριθμοι, ροές δεδομένων και δυναμικά περιβάλλοντα, χρονοσειρές, προ-επεξεργασία δεδομένων, υπολογιστικό κόστος, ακρίβεια*

Acknowledgements

My PhD studies were a unique experience in my life. Apart from the scientific knowledge that undoubtedly provided me with, it expanded my research concerns and strengthened my critical perception and creative thinking. These will be my “inheritance” for the future. For these reasons, I would like to thank the people who contributed to the successful completion of the dissertation.

The essential supervision of my studies facilitated their successful completion. Without this, my work would be difficult and might not get completed. Therefore, I would like to thank my supervisor, Georgios Evangelidis, for the scientific guidance, the essential supervision and for the unlimited time that he spent. For the same reasons, I would like to thank the other two members of the advisory committee, Dimitris A. Dervos, Professor at the ATEI of Thessaloniki and Jose Aldana F. Montes, Professor at University of Malaga. Last but not least, I thank Mr. Leonidas Karamitopoulos for the help that offered me in time-series and statistics related issues.

Of course, the rest faculty members of the examination committee contributed to the successful completion of the dissertation. Consequently, I would like to thank Georgia Koloniari, Konstantinos Margaritis, Apostolos Papadopoulos, and Nikolaos Samaras.

I would also like to make a special reference to the institutes that supported my Ph.D. studies. The State Scholarships Foundation of Greece (I.K.Y.), which not only supported me financially, but its scholarship allowed me to be on a sabbatical leave throughout my Ph.D. studies from the Secondary Education Institution where I work. The Regional Directorate of Primary and Secondary Education of Western Greece that granted the sabbatical leave. Actually, without the sabbatical leave, my studies would be impossible to complete. Finally, the Research Committee of the University of Macedonia for the financial support that provided to me for covering conference-related expenses.

I should also mention the Head of the Lyceum of Thermo, Ioannis Botsaris. Our long and very interesting discussions encouraged me to continue my studies by conducting Ph.D. research. I also thank my good friends Kimon Stavrou and Evripidis Pityrigkas for their help and support.

Finally, I would like to express my gratitude to my parents, Anestis and Anthi. I thank them for the support, the unconditional love and the constant encouragement to continue my studies at this level. I am really proud of them and I consider a large part of my research to belong to them. Finally, I would like to say a huge thank you to the woman who beautifies my life. She is my companion in life, my wife, Elena. She contributed to the completion of the dissertation with her love, patience and understanding. Consequently, the dissertation is devoted to her.

Ευχαριστίες

Η διαδικασία εκπόνησης της διδακτορικής μου διατριβής αποτελεί μια ξεχωριστή εμπειρία στη ζωή μου. Εκτός από την επιστημονική γνώση που αναμφίβολα μου παρείχε, με έφερε αντιμέτωπο με προκλήσεις που διευρύνανε τις ερευνητικές μου ανησυχίες και ενίσχυσαν την κριτική μου αντίληψη και τη δημιουργική μου σκέψη. Τα χαρακτηριστικά αυτά αποτελούν "κληρονομιά" για την μετέπειτα πορεία της ζωής μου. Για αυτούς τους λόγους θα ήθελα να ευχαριστήσω τους ανθρώπους που συνετέλεσαν στην επιτυχή ολοκλήρωση της διατριβής.

Η ουσιαστική επίβλεψη της διδακτορικής διατριβής διευκόλυνε την εκπόνηση της. Χωρίς αυτή, το έργο μου θα ήταν δύσκολο και ίσως να μην το έφερνα εις πέρας. Συνεπώς, οφείλω ένα μεγάλο ευχαριστώ στον επιβλέποντα της διατριβής, καθηγητή κ. Γεώργιο Ευαγγελίδη, για την επιστημονική καθοδήγηση και την ουσιαστική επίβλεψη της διατριβής καθώς και για τον απεριόριστο χρόνο που αφιέρωσε. Για τους ίδιους λόγους, εξίσου θερμά ευχαριστώ τα άλλα δύο μέλη της συμβουλευτικής επιτροπής, κ. Δημήτρη Δέρβο, καθηγητή του Αλεξάνδρειου ΤΕΙ Θεσσαλονίκης και κ. Jose Aldana F. Montes, καθηγητή του Πανεπιστημίου της Μάλαγα. Επίσης, ευχαριστώ τον κ. Λεωνίδα Καραμητόπουλο, διδάκτορα του Πανεπιστημίου Μακεδονίας, για την πολύτιμη βοήθεια που μου προσέφερε σε θέματα χρονοσειρών και στατιστικής.

Ασφαλώς, τα μέλη ΔΕΠ, που μαζί με τα μέλη της συμβουλευτικής επιτροπής, συγκρότησαν την επταμελή εξεταστική επιτροπή της διατριβής συνετέλεσαν στην επιτυχή ολοκλήρωση της. Έτσι, ευχαριστώ την κ. Γεωργία Κολωνιάρη, λέκτορα του Παν. Μακεδονίας, τον κ. Κωνσταντίνο Μαργαρίτη, καθηγητή του Παν. Μακεδονίας, τον κ. Απόστολο Παπαδόπουλο, επίκουρο καθηγητή του Α.Π.Θ. και τον κ. Νικόλαο Σαμαρά, αναπληρωτή καθηγητή του Παν. Μακεδονίας.

Επίσης, θα ήθελα να κάνω ειδική μνεία στους αρμόδιους των δύο φορέων που υποστήριξαν την εκπόνηση της διατριβής. Ο πρώτος φορές είναι το Ίδρυμα Κρατικών Υποτροφιών (Ι.Κ.Υ.), που όχι μόνο με υποστήριξε οικονομικά καθ' όλη τη διάρκεια των διδακτορικών σπουδών, αλλά η υποτροφία που μου παρείχε μου έδωσε τη δυνατότητα να βρίσκομαι σε εκπαιδευτική άδεια με αποδοχές από τη Δευτεροβάθμια Εκπαίδευση όπου εργάζομαι και για τα τέσσερα χρόνια εκπόνησης της διατριβής. Ο δεύτερος φορέας είναι η Περιφερειακή Διεύθυνση Πρωτοβάθμιας και Δευτεροβάθμιας Εκπαίδευσης Δυτικής Ελλάδας που μου χορήγησε την εκπαιδευτική άδεια. Είναι γεγονός ότι χωρίς τη χορήγηση της εκπαιδευτικής άδειας, η έναρξη των διδακτορικών μου σπουδών θα ήταν ανέφικτη. Επιπρόσθετα, ευχαριστώ την επι-

τροπή ερευνών του πανεπιστημίου Μακεδονίας για τη χρηματοδότηση που μου παρείχε για τη συμμετοχή μου σε επιστημονικά συνέδρια.

Θα ήταν παράλειψη αν δεν έκανα αναφορά στον προϊστάμενο μου, αλλά πάνω από όλα φίλο μου, διευθυντή του Λυκείου Θέρμου Αιτωλοακαρνανίας, Ιωάννη Μπότσαρη. Οι πολύωρες και γεμάτες ενδιαφέρον συζητήσεις μας με παρότρυναν να συνεχίσω τις σπουδές μου διεξάγοντας διδακτορική έρευνα. Επίσης ευχαριστώ τους καλούς μου φίλους Ευριπίδη Πιτυρίγκα και Κίμων Σταύρου για τη βοήθεια που μου παρείχαν όπου και όποτε τους τη ζήτησα.

Κλείνοντας, θα ήθελα να εκφράσω την ευγνωμοσύνη μου στους γονείς μου, Ανέστη και Ανθή. Τους ευχαριστώ για την υποστήριξη που με αμέριστη αγάπη μου παρείχαν όλα αυτά τα χρόνια καθώς και για τη συνεχή παρότρυνση για συνέχιση των σπουδών μου σε αυτό το επίπεδο. Είμαι πραγματικά περήφανος για αυτούς. Δε θα ήταν υπερβολή αν πω ότι ένα πολύ μεγάλο κομμάτι της διατριβής τους ανήκει και τους αξίζουν συγχαρητήρια. Τέλος, μεγάλο ευχαριστώ οφείλω στον άνθρωπο που μου ομορφαίνει τη ζωή. Δε θα μπορούσε να είναι άλλος εκτός από τη συνοδοιπόρο μου στη ζωή, σύζυγο μου, Έλενα. Με αγάπη, υπομονή και κατανόηση συνέβαλε ουσιαστικά στην ολοκλήρωση της διατριβής και για αυτό της αφιερώνεται.

Στην Έλενα

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Classification

The efficiency and the effectiveness of data mining algorithms is an important research problem that has attracted the attention of both the academia and the industry [56, 109]. Classification (or supervised learning according to machine learning terminology) is a crucial data mining task. Classification algorithms (or classifiers) [64] try to assign new, unclassified data items to a set of predefined classes, on the basis of the available training data, i.e., a set of already classified instances (or items). A typical classification example is to assign an email to either class “spam” or class “non-spam”.

Classifiers can be divided into two main algorithm categories [64]: (i) eager classifiers, and, (ii) lazy (or instance based) classifiers. Both share the same motivation, that is, accurate class prediction. However, they differ on how they work. Essential role for the effectiveness of the algorithms of both categories plays the available training set. An eager classifier pre-processes the available training data and builds a classification model that is then used to classify new, unclassified items. On the other hand, lazy classifiers do not build any classification model. In effect, they consider the training dataset as the classification model. A lazy algorithm classifies a new item by scanning the training set at the time it arrives.

Since eager classifiers build a classification model before the arrival of any new item, the classification process is very fast. Although lazy classifiers do not spend any time to build classification models, their classification process is more time-consuming than that of eager classifiers. A drawback of eager classifiers is that they have to generate a single hypothesis that covers the entire training set. This is not always feasible, may affect the classification accuracy and may render the construction of the classification model an extremely time-consuming and complicated pre-processing task. On the other hand, lazy classifiers use the entire training set and, thus, can adopt more complex hypothesises about the data. Consequently, they may improve the classification accuracy. A disadvantage of lazy classifiers is that they require all the training data to be always available, which leads to high storage requirements. In contrast, in eager classification, after the construction of the classification model, the training data can be removed in order to save space .

During the past decades, the problem of classification has attracted the interest of many researchers from different research fields of computer science. Therefore, various eager and lazy classifiers have been proposed and are available in the literature.

Classification decision trees [110] constitute a well-known subcategory of eager classifiers. Based on the available training data, these classifiers build a tree structure that is used to classify new items. Other eager classifiers are based on artificial neural networks [59, 141]. A neural network is first trained by the training items and then it performs classifications. Probabilistic classifiers belong also to the eager classifiers category. They build a classification model that is based on probabilities. One characteristic example of a probabilistic classifier is the naive Bayes classifier [37, 142]. Another subcategory of eager algorithms includes the classifiers based on association rules mining [119]. They discover association rules within the available training data. These rules are used for classification purposes.

On the other hand, the category of the lazy classifiers includes the well-known k Nearest Neighbours classifier [28, 27] and the case-based reasoning classification methods [74]. The k Nearest Neighbours classifier is the research subject of study of the this dissertation.

1.2 The *k*-Nearest Neighbours classifier

The k-Nearest Neighbours (k-NN) classifier [28, 27] is an effective and extensively used lazy classification algorithm. It is a simple and easy to implement classifier, can be exploited in many application domains and can be easily integrated in many systems. Moreover the k-NN classifier is analytically tractable and for $k = 1$ and unlimited items the error rate is asymptotically never worse than twice the minimum possible, which is the Bayes' rate [27].

Since the k-NN classifier is a lazy classifier, it does not build any classification model. The algorithm uses the training data whenever a new item needs to be classified. In particular, it

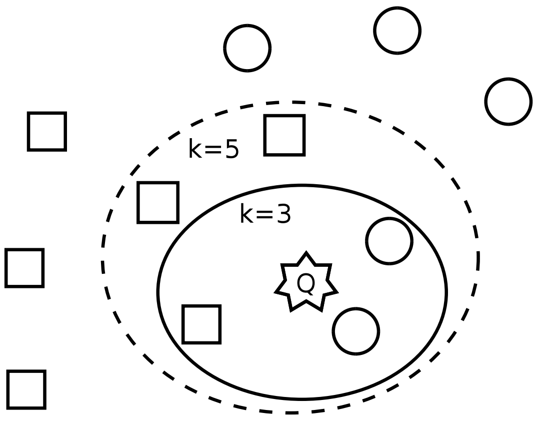


Figure 1.1: k Nearest Neighbours classification process with $k = 3$ and $k = 5$

classifies an item x by searching in the available training data and retrieving the k nearest items (neighbours) to x according to a distance metric. Then, x is assigned to the most common class among the classes of the retrieved k nearest neighbours. This class is often called the major class and is determined via a procedure known as the nearest neighbours voting. Note that when $k = 1$, the algorithm is also known as nearest neighbour classifier (or 1-NN rule).

Figure 1.1 illustrates a two-dimensional example of the classification process. More specifically, it shows a dataset with two classes, squares and circles, and a query item (Q) that needs to be classified to one of these classes. If we set k to be equal to three (solid line circle in Figure 1.1), Q is classified to the class circle because two of the three nearest neighbours are circles. On the other hand, if k is set to be equal to five (dashed line circle in Figure 1.1), the query item is assigned to the class square because three of the five neighbours belong to class square.

The classification performance certainly depends on the selection of the value of parameter k. The value of k that achieves the highest classification accuracy depends on the dataset used and its determination usually implies tuning via costly trial-and-error preprocessing tasks. Although the determination of k can not follow any general rule and the "best" k may be completely different for different datasets, larger k values are appropriate for datasets with noise since they examine larger neighbourhoods. However, they do not clearly define the boundaries between distinct classes. In contrast, small parameter values render the classifier more noise-sensitive. Therefore, in cases of training data that contains noise, classification is probably less accurate. It is worth mentioning that even the best k value may not be optimal. This happens because the k-NN classifier uses a unique k value. Different k values may be optimal for different regions of the data space. Consequently, heuristics for dynamic determination of k [105] can be adopted that can achieve higher accuracy than the k-NN classifier with its "best" k value determination.

In cases of binary classification problems (datasets with two classes), k should have an odd value to avoid ties (both classes are the most common) during nearest neighbours voting. In cases of non-binary problems, k can have any value. Here, possible ties during voting are resolved by selecting either a random "most common" class or the class of the nearest neighbour. The popular Weka software [53] and many other data mining / machine learning software tools resolve ties randomly. In the experimental studies of this dissertation, we adopt the single nearest neighbour classifier to resolve ties.

Another important issue that should be addressed is the selection of the metric used to compute the distance between items. Certainly, this decision should take into consideration the data types of the dataset attributes (variables). In cases of real and / or integer attributes, the Euclidean distance is the commonly-used distance metric. However, other distance metrics can be adopted (e.g., Mahalanobis, Manhatan, Minkowski, Chebyshev) [35]. Many other similarity measures have been proposed to handle nominal attributes (non-metric spaces). However, all experimentations in this dissertation are conducted on datasets with real and / or integer attributes. Therefore, we adopt the Euclidean distance as the distance metric. Consequently, data items described by n attributes are considered as data points (or vectors) in the n-dimensional Euclidean metric space, and the Euclidean distance between points p and q is given by:

$$d(p,q) = d(q,p) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \ldots + (q_n - p_n)^2} = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

Different attribute ranges can affect the distance value. Even in cases where all attributes have the same significance, attributes with wide ranges have higher impact on the distance value than attributes with narrower ranges. Suppose that attribute "salary" ranges from 800 to 5000 and attribute "number of children" takes values from 0 to 6. With both attributes having the same significance, "salary" has higher impact in the distance computation than "number of children". Therefore, the range of the attributes should be normalized to a particular interval range (e.g., $[0, 1]$). Assume that a dataset contains n items and an attribute e should be normalized to $[0, 1]$. The attribute value of the i-th item, $i = 1, \ldots, n$ is normalized as follows:

$$normalized(e_i) = \frac{e_i - E_{min}}{E_{max} - E_{min}}$$

where E_{min} and E_{max} are the minimum and maximum values for attribute e, respectively. Data normalization is a common preprocessing procedure in many data mining tasks. Some data mining software suites apply it by default.

Several variations of the k-NN classifier have been proposed. The most important one is the distance-weighted k-NN rule [39] that uses a distance-weight function to weight more heavily the closer neighbours than the further ones. The nearest neighbour is weighted by one while the furthest of neighbour is weighted by zero. The weights of all other neighbours are scaled to this interval. A new item is classified by a majority weighted vote: it is assigned to the class with the largest sum of weights.

1.3 Motivation / Weaknesses of the *k*-NN classifier

Although the k-NN classifier is considered to be an effective method, it has some weaknesses that may render its use inefficient. The first weak point is the high computational cost involved. The k-NN classifier must compute all distances between each unclassified item and all items stored in the training set. In cases of large datasets, this drawback renders its use a time-consuming and in some cases a prohibitive procedure. For instance, suppose that a classification system stores 100,000 training items. In addition, suppose that the system should classify about 50,000 unclassified items by executing the k-NN classifier over the training data. This means that the system must compute five billion distances. Although nowadays systems are equipped with powerful processors, these computations are time-consuming and unacceptable in cases of time-constraint environments. We should mention that, apart from the size of the training set, the computational cost of the classification task also depends on the data dimensionality. The higher the data dimensionality, the more the computations performed for a distance computation.

Another weakness of the k-NN classifier is the large storage requirements for the training data. Contrary to eager classifiers that can discard the training data after the construction of the classification model, the k-NN classifier needs the training data to be always available. Consequently, the k-NN classifier must be executed on computer systems equipped with enough main memory to store the training data.

The last weakness is that the k-NN classifier, like many other classification methods, is a noise-sensitive method. In particular, the classification accuracy highly depends on the quality of the training data. Noise and mislabelled data, as well as outliers and overlaps between data regions of different classes, lead to less accurate classification. Usage of high k values extends the examined neighbourhood and, thus, can partially remedy this drawback. However, it implies a high number of trial-and-error executions to determine the appropriate k value and the noise is uniformly distributed in the training set.

These weaknesses constitute an active research problem and have attracted the interest of the data mining research community. This dissertation is also motivated by these weaknesses and contributes novel algorithms and techniques to address them.

1.4 Method categories for efficient and effective *k*-NN classification

Although most of the recent research efforts focus on the reduction of the computational cost of the k-NN classifier, numerous algorithms and techniques have been proposed that can deal with the other weak points of the classifier. A possible categorization of methods for improved k-NN classification is: (i) Multi-attribute Indexes, (ii) Data Reduction Techniques, and, (iii) Cluster-Based Methods.

Multi-attribute indexes [112, 139, 22] can accelerate the nearest neighbours searches and, as a consequence, they can speed-up the k-NN classifier. However, storage requirements increase, since, in addition to the training data, the index must be stored, too. Moreover, indexes can be applied only on datasets with moderate dimensionality (e.g., 2-10). In higher dimensions, the phenomenon of the dimensionality curse makes sequential scans more effective than indexes [129]. Thus, it is essential to first apply a dimensionality reduction technique, such as the Principal Component Analysis (PCA) [67]. Unfortunately, PCA may lead to significant information loss. In addition, each unclassified item should be transformed before searching for its nearest neighbours. Therefore, classification may become less efficient.

Data Reduction Techniques (DRTs) have two points of view: (i) item reduction, and, (ii) dimensionality reduction. The dissertation considers them from the first point of view. DRTs can effectively cope with all weaknesses. They can be grouped into two main algorithm categories: (i) prototype selection algorithms [48], and, (ii) prototype abstraction [123] algorithms. Prototype selection algorithms select representative items (or prototypes) from the initial training set, whereas prototype abstraction algorithms generate items by summarizing on similar training items and use them as prototypes. In effect, each prototype represents a specific data area of the multidimensional space.

Prototype selection algorithms are divided into two subcategories. They can be either condensing or editing algorithms. Prototype abstraction and condensing algorithms have the same motivation. They aim to built a small representative set of the initial training data. This set is usually called the condensing set. Usage of a condensing set has the benefits of low computational cost and storage requirements, while classification accuracy is not negatively affected. On the other hand, editing algorithms aim to improve accuracy rather than achieve high reduction rates. To achieve this, they try to improve the quality of the training data by removing noise, outliers and mislabelled items and by smoothing the decision boundaries between classes (see Figure 2.2). Ideally, an editing algorithm builds an edited training set without overlaps between the classes. Figure 1.2 summarizes the aforementioned categories in a hierarchical taxonomy. It is worth mentioning that some condensing algorithms integrate the idea of editing. In [48], they are called hybrid algorithms.

Prototype selection and abstraction algorithms are reviewed and compared to each other in [48] and [123], respectively. Also, both of these papers present interesting taxonomies. Other relevant reviews can be found in [122, 131, 65, 51, 17, 84, 76]. Furthermore, Chapter 2 presents a short review of the most significant DRTs.

Cluster-based methods can speed-up the k-NN classification process. Like multi-attribute indexes and contrary to DRTs, they do not reduce the size of the training set. Moreover, they do not improve the quality of the training data. Their unique goal is the reduction of the computational cost. To achieve this, cluster-based methods pre-process the training data and group it into clusters. For each new item, they dynamically form an appropriate training subset of the initial training set, which then is used to classify the new item. This subset is usually called reference set and is a union of some clusters.

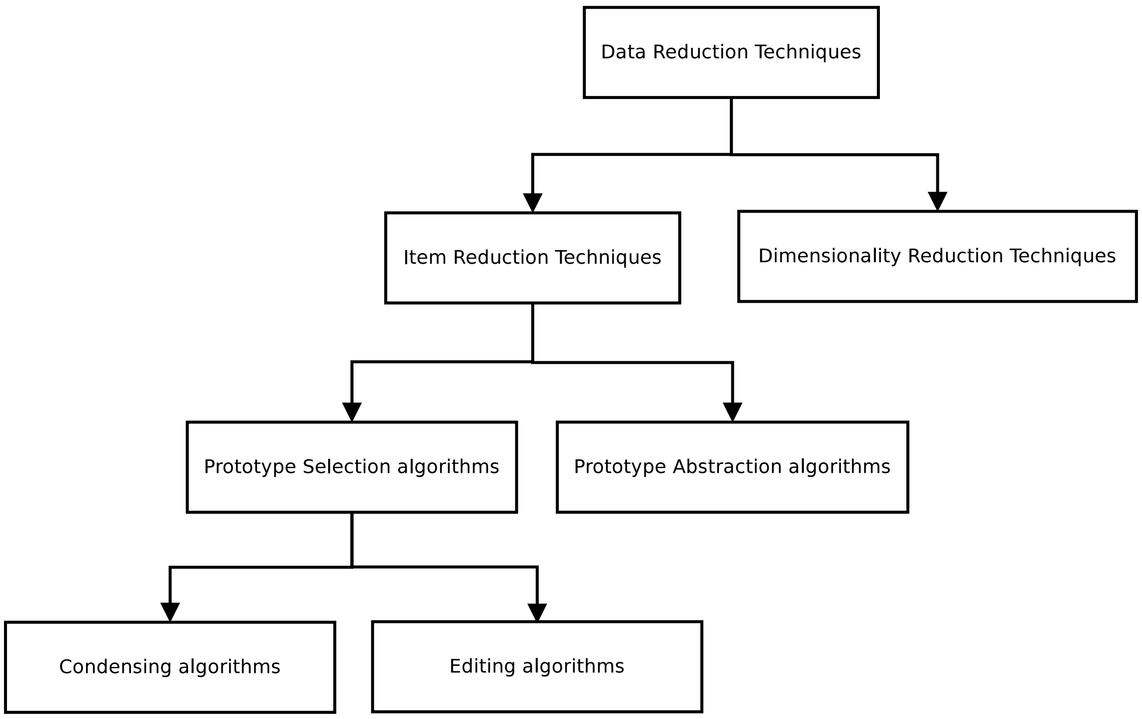

Data Reduction Techniques
Item Reduction Techniques
Dimensionality Reduction Techniques
Prototype Selection algorithms
Prototype Abstraction algorithms
Condensing algorithms
Editing algorithms

Figure 1.2: Categories of data reduction techniques: An hierarchical taxonomy

1.5 Contribution / Dissertation organization

This dissertation focuses on DRTs and cluster-based methods. It contributes new, novel DRTs and proposes improvements for existing methods. Moreover, it proposes hybrid classification schemas that combine DRTs and cluster-based methods. In this section, we summarize the main findings and the contribution and, at the same time, we present the organization of the dissertation.

Chapter 2 presents a survey of existing methods for efficient and effective k-NN classification. Prototype selection and abstraction algorithms as well as cluster-based methods and multi-attribute indexes are reviewed. Their advantages and drawbacks are discussed. Although Chapter 2 emphasizes on the methods implemented in the context of the dissertation and used for comparison purposes in the experimental studies, it briefly reviews all the state-of-the-art methods. In addition, Chapter 2 discusses k-means clustering that is the base of the methods introduced in the following chapters.

Chapter 3 contributes new DRTs based on forming clusters containing items of a specific class only (homogeneous clusters). More specifically, the chapter introduces a family of four novel fast and non-parametric (independent of tuning parameters) algorithms. These are (i) the Reduction through Homogeneous Clusters (RHC) algorithm [92, 91], (ii) the dynamic version of RHC (dRHC) [91], (iii) the Editing through Homogeneous Clusters (EHC) algorithm [98] and (iv) the Editing and Reduction through Homogeneous Clusters (ERHC) algorithm [89]. Although they are based on the same idea, each one aims to a different goal. RHC is a prototype abstraction algorithm that achieves high reduction rates with low preprocessing cost while maintaining k-NN classification accuracy at high levels. dRHC is a dynamic prototype abstraction algorithm that incrementally builds its condensing set and, therefore, is appropriate for dynamic / streaming environments [1] where new training data is gradually available and for very large datasets that can not fit in main memory. EHC is an editing algorithm that improves the quality of the training data by removing noise, outliers and mislabelled data as well as by smoothing the decision boundaries between distinct classes. ERHC combines the idea of RHC with that of EHC. In effect, it is a variation of RHC that can effectively deal with datasets with noise.

Chapter 4 also focuses on data reduction. It proposes two additional prototype abstraction algorithms, namely, (i) the Abstraction IB2 (AIB2) algorithm [96, 88], and, (ii) the Reduction through k Means (RkM) algorithm [95]. AIB2 is a prototype abstraction version of the well-known IB2 algorithm. It inherits all the good properties of IB2 but it performs better. RkM is a simple noise-tolerant prototype abstraction algorithm that is based on k-means clustering [79, 133].

Chapter 5 proposes novel methods that combine different speed-up strategies for fast k-NN classification in hybrid classification schemas. Although they do not reduce the storage requirements, they accelerate the classification process. The chapter introduces three hybrid classification methods and variations. First, a fast, hybrid and model-free classification algorithm and two variations [101] are proposed. The reduction of the computational cost is achieved by the combination of the conventional k-NN and the minimum distance [38] classifiers. Then, a classification method is proposed that combines the idea of DRTs with that of cluster-based methods to achieve the desirable performance [102, 90]. The particular method can be used either to improve accuracy at a lower cost, or to reduce cost at a minimum level without sacrificing accuracy. This is achieved by appropriately adjusting a set of input parameters. The last section of Chapter 5 extends the aforementioned idea and proposes a hybrid classification method that also combines the strategy of data reduction with that of cluster-based methods [97, 94]. However, the proposed method is non-parametric (independent of tuning parameters). A variation of the aforementioned method is also proposed that can improve the performance of state-of-the-art DRTs.

Chapter 6 presents some additional research tasks and experimentations conducted in the context of the dissertation. Initially, it elaborates on fast time series classification through general-purposes DRTs [104, 103]. Then, it deals with the recently proposed Prototype Selection by Clustering (PSC) algorithm [86, 85, 87] that is a prototype abstraction algorithm based on k-means clustering. In particular, Chapter 6 demonstrates that the reduction rate and the accuracy of PSC can be improved by generating a large number of clusters [93]. Finally, the chapter focuses on the cluster-based method proposed by Hwang and Cho [62] by presenting an extensive experimental study. The results show that if a set of parameters is carefully defined, one can achieve improved classification performance.

Chapter 7 concludes the dissertation by summarizing on its findings and contribution. In addition, it gives some research directions for future work. Last but not least, Appendix A presents WebDR [99], a web-based application developed during the PhD research and offers all DRTs coded in the context of the dissertation available on-line. In effect, WebDR is a workbench that allows the user to plan and run experiments as well as to evaluate the performance of DRTs over several know datasets through the web.

Chapter 2

Background knowledge

2.1 Data Reduction Techniques

2.1.1 Introduction

As already mentioned in Section 1.4, there are two categories of Data Reduction Techniques (DRTs) [123, 48, 76, 122, 131, 65, 51, 17, 84]: prototype selection and prototype abstraction algorithms. In addition, prototype selection algorithms are distinguished into condensing and editing algorithms. Prototype abstraction and condensing algorithms can cope with the weaknesses of the k-NN classifier regarding the high classification computational cost and storage requirements. This is achieved by building a small representative set of the initial training data. This set is called the condensing set and contains only essential items. Applying the k-NN classifier using the condensing set, one has the benefits of much lower computational cost and storage requirements, whereas accuracy remains high or does not degrade significantly. On the other hand, editing algorithms attempt to improve the accuracy by removing training data that is noise and smoothing the decision boundaries between classes.

DRTs can be evaluated using three criteria. The first one is the reduction rate that indicates how much smaller the size of the condensing set is with regards to the size of initial training set. Practically, it is the ratio of the number of discarded items over the number of initial items of the training set. Obviously, the higher the reduction rate, the faster the k-NN classification. Another major criterion is the classification accuracy achieved by the k-NN classifier when it runs over the condensing set. The third criterion is the preprocessing computational cost, that is, the cost required to build the condensing set. For certain domains, one criterion may be

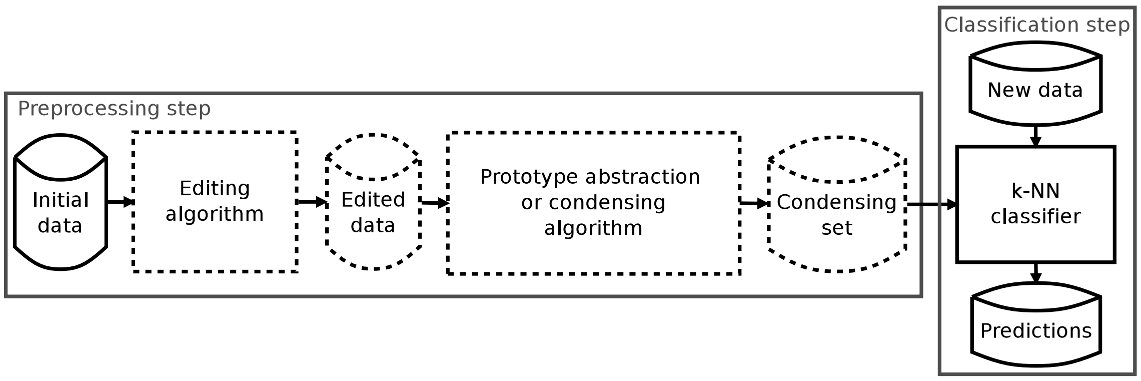


Figure 2.1: k-NN classification through data reduction

more critical than another. However, all of them should satisfy certain minimum requirements. The dissertation considers all criteria as having the same significance.

Although editing has a completely different goal than the other DRTs, it can be used to improve their performance by increasing their reduction rates and / or accuracy levels. More specifically, the reduction rates of many prototype abstraction and condensing algorithms depend on the level of noise in the training data. High levels of noise in the training set prevent many condensing or prototype abstraction algorithms from achieving high reduction rates. In effect, the higher the level of noise, the lower the reduction rates achieved. Therefore, effective application of such algorithms implies removal of noise from the data, i.e., application of an editing algorithm beforehand [29, 76]. Hence, an editing algorithm should be used on a training set with noise in order to either improve accuracy or make the application of condensing and prototype abstraction algorithms more effective.

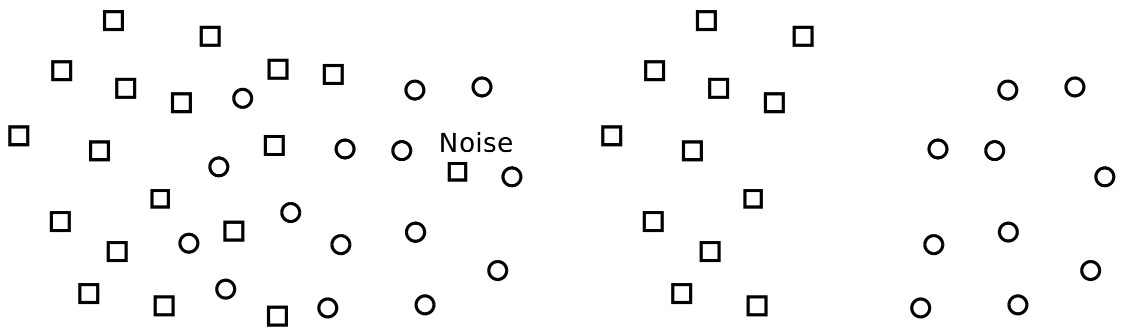
Figure 2.1 summarizes the k-NN classification process through data reduction. The whole process includes two phases, preprocessing and classification. Certainty, the preprocessing phase is optional. In general, there are four possible types of preprocessing: (i) no-preprocessing, (ii) only editing, (iii) only condensing, and (iv) both editing and condensing. If the training set does not contain noise and misleading data and its size is small, no preprocessing is required. When the size of the training set is small, but it contains noise, only an editing algorithm should be executed during preprocessing. On the other hand, in cases of large and noise-free training sets, data reduction without editing should be executed.

Finally, in cases of large and training data with noise, both kinds of preprocessing algorithms must be ran. All four types of preprocessing are available on WebDR[1] (see Appendix A).

The goal of a complete data reduction preprocessing procedure is to build a noise-free condensing set by keeping or generating for each class a sufficient number of prototypes that are essential for the k nearest neighbours classification. We should mention that many DRTs have been implemented under the KEEL software [7], which is an open-source Java-based framework.

2.1.2 Prototype selection algorithms for data editing

Editing algorithms improve the quality of the training data by removing outliers, noise and mislabelled items as well as by smoothing the class decision boundaries. Ideally, an editing procedure tries to create a training set without overlaps between classes. Figure 2.2 presents the type of data that editing algorithms try to remove. In this section, three state-of-the-art editing algorithms are explained in detail. They have been coded in C and are used for comparison purposes in Chapter 3. Moreover, they are available on WebDR and can be executed on-line. In addition, the most popular editing algorithms are briefly surveyed.



(a) Initial training set (b) Edited set

Figure 2.2: Smoothing decision boundaries and removing noise

[1]https://ilust.uom.gr/webdr

The Edited Nearest Neighbor (ENN) rule

The reference editing algorithm is Wilson's Edited Nearest Neighbor (ENN) rule [132]. It constitutes the base of all other editing algorithms. ENN-rule is very simple. Algorithm 1 lists the pseudo-code of ENN-rule. Initially, the edited set (ES) is set to be equal to the training set (TS) (line 1). For each item x of TS, the algorithm scans TS and retrieves its k nearest neighbours (line 3). If x is misclassified by the majority vote of the retrieved nearest neighbours, it is removed from ES (lines 4–7). ENN-rule considers wrongly classified items to be noise or close-border items and, thus, they must be removed. Note that, in each algorithm iteration, ENN-rule searches for nearest neighbours in the original training set and not in the "under construction" edited set.

Algorithm 1 ENN-rule

```
Input: $TS, k$
Output: $ES$
1: $ES \leftarrow TS$
2: for each $x \in TS$ do
3:     $NNs \leftarrow$ find the $k$ nearest to $x$ neighbors in $TS - \{x\}$
4:     $majorClass \leftarrow$ find the most common class of $NNs$
5:     if $x_{class} \neq majorClass$ then
6:         $ES \leftarrow ES - \{x\}$
7:     end if
8: end for
9: return $ES$
```

Obviously, the cost of editing depends on the size of the training set. In cases of large datasets, ENN-rule is a time-consuming algorithm. ENN-rule must compute all distances between the training items. Therefore, $\frac{N \times (N-1)}{2}$ distances must be computed, where N is the number of items in the training set.

A crucial issue that should be addressed is the determination of the value of k that determines the size of the examined neighbourhood. [131, 49, 84] consider $k = 3$ to be a typical value. This is adopted in many papers (e.g. [113]), whereas, other papers use $k = 3$ and additional k values (e.g., [118, 58]). In some cases, researchers determine the value of k that achieves the best performance through trial-and-error procedures (e.g., [126]). In [132], the impact of k is discussed in detail. Furthermore, in [58], a large number of k values are experimentally evaluated. It turns out that the best value of k depends on the dataset at hand

and should be determined by considering the distribution of items in the multidimensional space. Even the best value of k may not be optimal and it may remove items that are not noise (see [49]) or keep items that are noise. This happens because ENN-rule uses a unique k value for the entire training set. Different k values may be optimal for different regions in space.

All *k*-NN

All-kNN [120] is a popular variation of ENN-rule. It iteratively executes ENN-rule with different k values (see Algorithm 2). All-kNN adopts $kmax$ as an upper limit for the value of k. Initially, the edited set (ES) is set to be the whole training set (TS) (line 1). For each item x in TS (line 2), All-kNN applies the k-NN classifier on the items of TS (lines 6–7), initially with $k = 1$, and tries to remove x from ES in a way similar to ENN-rule. If x is misclassified, it is removed and the procedure continues with the next item (lines 8–10). Otherwise, k is incremented by one (line 12) and the algorithm retries to remove x. If the item is not removed after $kmax$ iterations (line 5), x remains in the final ES and All-kNN continues with the next item.

Algorithm 2 All-kNN

```
Input: $TS, kmax$
Output: $ES$
 1: $ES \leftarrow TS$
 2: for each $x \in TS$ do
 3:     $k \leftarrow 1$
 4:     $flag \leftarrow FALSE$
 5:     while ($k \leq kmax$) and ($flag == FALSE$) do
 6:         $NNs \leftarrow$ find the $k$ nearest to $x$ neighbors in $TS - \{x\}$
 7:         $majorClass \leftarrow$ find the most common class in $NNs$
 8:         if $x_{class} \neq majorClass$ then
 9:             $ES \leftarrow ES - \{x\}$
10:             $flag \leftarrow TRUE$
11:         end if
12:         $k \leftarrow k + 1$
13:     end while
14: end for
15: return $ES$
```

Since All-kNN uses more than one values for k, it removes more items than ENN-rule. Although All-kNN is an iterative version of ENN-rule, an efficient implementation of it does not re-compute the same distances again and again. Therefore, All-kNN computes as many distances as ENN-rule and is parametric, too. The value of $kmax$ must be determined by the user. This usually implies tuning via trial-and-error. Garcia-Borroto et al. consider $kmax = 7$ or $kmax = 9$ to be appropriate values [49].

Multiedit

Multiedit [34] is another well-known editing approach. Its pseudo-code is presented in Algorithm 3. Initially, the edited set (ES) is set to be equal to the training set (TS) (line 1). Then, TS is divided into n random subsets, $s_1, s_2, \ldots, s_n$ (line 5). The algorithm continues by applying ENN-rule over each item $x \in s_i$ (line 7) of each subset s_i (line 6), but searching for the single nearest neighbor (1-NN) in the modulo n following subset, i.e., $s_{(i+1) \bmod n}$ (line 8). The misclassified items are removed from ES (line 10). If at least one item is removed, TS is set to be ES (line 20) and the whole process is repeated. Multiedit continues until the last R iterations produce no editing (lines 11,15–16,21).

Here, parameter k is not used since multiedit utilizes the 1-NN classifier. However, parameters n and R influence the resulting edited set. Parameter $n \geq 3$ determines the number of subsets. In many papers (e.g., [49, 118]), $n = 3$ is either adopted or proposed. Parameter R determines the number of non-editing iterations. In [49], $R = 2$ is suggested as an appropriate value. Nevertheless, the best values for these parameters can not be determined without tuning through a trial-end-error procedure.

Multiedit usually achieves higher reduction rates than ENN-rule. It can successfully remove noise, outliers and close-border items. However, it may also remove items that are not noise. If items of two or more classes are close to each other, multiedit may eliminate entire classes [49]. Another drawback of multiedit is that it is based on a random formation of subsets, i.e., repeated applications may build a completely different edited set from the same training set.

Multedit is usually more time-consuming than ENN-rule. However, it may compute even fewer than $\frac{N \times (N-1)}{2}$ distances. An implementation of multiedit that does not compute a distance more than once should have the distances that have been already computed available until the end of the execution. Therefore, such an implementation requires more memory. In case of a simple implementation where each distance may be computed more than once, the computational cost of the algorithm highly depends on the value of R.

Algorithm 3 Multiedit

```
Input: $TS, n, R$
Output: $ES$
 1: $ES \leftarrow TS$
 2: $r \leftarrow 0$
 3: repeat
 4:   $flag \leftarrow$ FALSE
 5:   $S \leftarrow$ set of $n$ random subsets, $s_1, s_2, \ldots, s_n$ of $TS$
 6:   for each $s_i \in S$ do
 7:     for each $x \in s_i$ do
 8:       $nn \leftarrow$ find the nearest neighbor in $s_{(i+1) \bmod n}$
 9:       if $x_{class} \neq nn_{class}$ then
10:         $ES \leftarrow ES - \{x\}$
11:         $flag \leftarrow$ TRUE
12:       end if
13:     end for
14:   end for
15:   if $flag ==$ FALSE then
16:     $r \leftarrow r + 1$
17:   else
18:     $r \leftarrow 0$
19:   end if
20:   $TS \leftarrow ES$
21: until $r == R$ {until none of the last R iterations edit data}
22: return $ES$
```

Other editing algorithms

Three state-of-the-art editing algorithms were previously presented. They are used for comparison purposes in our experimental study in Chapter 4. Many more editing approaches have been proposed in the literature.

EENProb and ENNth [126] are extensions of ENN-rule. Both retrieve the k nearest neighbors, and then perform editing based on probability estimations. Repeated ENN (RENN) rule [120] is also a variation of ENN-rule. Actually, it is quite similar to All-kNN. RENN-rule applies ENN-rule in an iterative way until each item's majority of k nearest items have the same class. In [58], another simple variation of ENN-rule is proposed that places an item in the edited set, only if all its k nearest neighbours have the same class label with it (distance ties increase the value of k).

Sanchez et al. proposed two editing algorithms that are based on geometric information provided by proximity graphs [118]. They are also based on the concept of removal of misclassified items. To the best of our knowledge, they are the only non-parametric editing algorithms. Nevertheless, the type of proximity graphs used influence the resulting edited set. In [118], two types of proximity graphs were used. Consequently, four editing approaches were obtained and evaluated. From this point of view, even these algorithms can be characterized as parametric methods.

k-NCN editing and its iterative version [113] are also based on ENN-rule. Particularly, they use the k nearest centroid neighbourhood classifier [117] instead of the k-NN classifier. Both are based on the following simple idea: the appropriate neighbourhood that should be examined for each item is determined by taking into consideration not only its nearest neighbours but also the symmetrical distribution of neighbours around it.

In [13, 113] a depuration algorithm is proposed for editing training data. In addition to removing some training items, the algorithm also changes the class labels of some items. To achieve this, it uses two input parameters (see [13] or [113] for details). [66] considers and evaluates editing approaches based on the depuration algorithm and proposes the Neural Network Ensemble Editing (NNEE). This method is also parametric. NNEE trains a neural network ensemble that is then used to relabel some items. Last by not least, a recent paper [115] proposes the use of local support vector machines for noise reduction. Like the other methods, its performance depends on parameter tuning.

2.1.3 Prototype selection algorithms for data condensation

This subsection presents in detail three state-of-the art condensing algorithms. These algorithms were coded in C and can be executed on-line through WebDR[2] (see Appendix A). They are used for comparison purposes in the next chapters of the dissertation. Moreover, the section presents a short survey of other well-known condensing algorithms.

Condensing Nearest Neighbour rule

The Condensing Nearest Neighbour (CNN) rule [57] is the earliest and also a reference condensing algorithm. It is used in many papers for comparison purposes. CNN-rule (and many other DRTs) builds its condensing set based on the following simple idea. Items that lie in

[2]`https://ilust.uom.gr/webdr`

the "internal" data area of a class (i.e., far from class decision boundaries) are useless during the classification process. Consequently, they can be removed without loss of accuracy. By adopting this idea, CNN-rule tries to place into the condensing set only the items that lie in the close-class-border data areas. They are the only essential items for the classification process. Figure 2.3 depicts this strategy. The idea is that the k-NN classifier will be able to have similar accuracy using either the training set or the condensing set. However, the usage of the condensing set involves much lower computational cost and storage requirements than the training set.

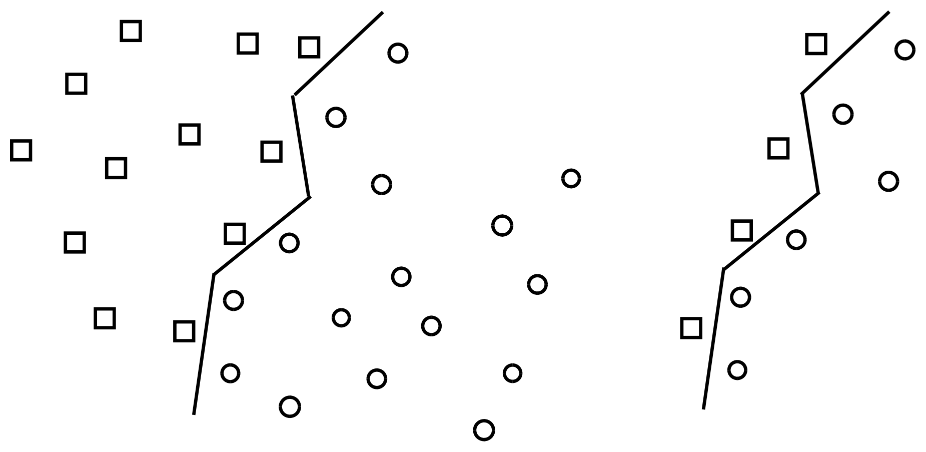

(a) Training set (b) Condensing set

Figure 2.3: Initial training data and close-class-border data

CNN-rule tries to keep the close-class-border items as follows (see Algorithm 4). Initially, an item of the training set (TS) is moved to the condensing set (CS) (line 2). Then, CNN-rule applies the 1-NN rule and classifies the items of TS by scanning the items of CS (line 6). If an item is misclassified, it is moved from TS to CS (lines 7–11). The algorithm continues until there are no moves from TS to CS during a complete pass of TS (line 13). This ensures that the content of TS is correctly classified by the content of CS. The remaining content of TS is discarded (line 14).

CNN-rule considers that misclassified items are probably close to decision boundaries and so they must be placed in the condensing set. Obviously, items that are noise are misleading. CNN-rule wrongly places that kind of items with their neighbourhood (items that are close to them) in the condensing set. Therefore, the reduction rate (and the preprocessing cost) is

Algorithm 4 CNN-rule

```
Input: $TS$ Output: $CS$
 1: $CS \leftarrow \varnothing$
 2: pick an item of $TS$ and move it to $CS$
 3: repeat
 4:     $stop \leftarrow TRUE$
 5:     for each $x \in TS$ do
 6:         $NN \leftarrow$ Nearest Neighbour of $x$ in $CS$
 7:         if $NN_{class} \neq x_{class}$ then
 8:             $CS \leftarrow CS \cup \{x\}$
 9:             $TS \leftarrow TS - \{x\}$
10:             $stop \leftarrow FALSE$
11:         end if
12:     end for
13: until $stop == TRUE$ {no move during a pass of $TS$}
14: discard $TS$
15: return $CS$
```

affected by high levels of noise. Of course, the number of discrete classes may also affect the reduction rate. The more classes, the more boundaries probably exist and as a consequence more prototypes are collected.

An advantage of CNN-rule is that it is a non-parametric approach. It determines the number of the prototypes automatically, without user-defined parameters. Another desired property is that CNN-rule through the multiple passes over the data guarantees that the removed training items can be correctly classified by executing 1-NN rule in the context of the final condensing set. A drawback is that the resulting condensing set depends on the ordering of the training set items. Therefore, CNN-rule builds a different condensing set by examining the same training data in a different order.

Furthermore, CNN-rule cannot handle new training data, i.e., is non-incremental[3]. This means that new training data cannot update an already constructed condensing set. CNN-rule involves multiple passes over the data and it cannot use training data available at a later time to update its condensing set. To construct an updated condensing set, CNN-rule needs to

[3]In the literature [48, 123], DRTs can be incremental or decremental depending on the way they build their condensing set. An incremental DRT begins with an empty condensing set and adds items to it, whereas a decremental DRT uses the whole training set as the initial condensing set and then removes items. In this dissertation, the use of term incremental refers to the ability of the DRT to update an already constructed condensing set

Algorithm 5 IB2

Input: TS **Output**: CS

```
 1: $CS \leftarrow \varnothing$
 2: pick an item of $TS$ and move it to $CS$
 3: for each $x \in TS$ do
 4:     $NN \leftarrow$ Nearest Neighbour of $x$ in $CS$
 5:     if $NN_{class} \neq x_{class}$ then
 6:         $CS \leftarrow CS \cup \{x\}$
 7:     end if
 8:     $TS \leftarrow TS - \{x\}$
 9: end for
10: return $CS$
```

be executed from scratch over the complete training set (new and old training data). Therefore, the training items that do not enter the condensing set should be retained. Hence, CNN-rule is inappropriate for dynamic / streaming environments [1] where new training data is gradually available. In addition, CNN-rule requires that all training items are memory resident.

The IB2 algorithm

IB2 belongs to the well-known family of Instance-Based Learning (IBL) algorithms [5, 4] and is based on CNN-rule. In effect, IB2 constitutes a simple one pass variation of CNN-rule. Algorithm 5 presents IB2 in pseudo-code. Each training item $x \in TS$ is classified using the 1-NN classifier on the current CS (line 4). If x is classified correctly, it is discarded (line 8). Otherwise, x is transferred to CS (line 6).

Similarly to CNN-rule, IB2 is non-parametric and its condensing set highly depends on the order of items in the training set. Contrary to the CNN-rule, IB2 does not ensure that all discarded items can be correctly classified by the final version of the condensing set. However, since it is a one-pass algorithm, it is very fast, i.e., it involves low preprocessing computational cost.

In addition, IB2 builds its condensing set incrementally. New training items can be taken into consideration after the creation of the condensing set. In other words, new training data segments can update an existing condensing set in a simple manner and without considering the “old” (removed) data that had been used for the construction of the condensing set. Each new training item can be examined, be placed or not in the condensing set, and then, removed.

Moreover, IB2 can handle new class labels. Therefore, IB2 is an appropriate DRT for dynamic / streaming environments, whereby new training data may be gradually available. Certainly, IB2 can not deal with data streams with concept drift [125]. IBL-DS [15] adopts the idea of the family of IBL algorithms and can deal with such data.

Last but not least, contrary to the CNN-rule and to many other DRTs, IB2 does not require that all training data reside in main memory. Therefore, it can be applied in devices whose memory is insufficient for storing all the training data.

Prototype selection by clustering

Prototype Selection by Clustering (PSC) [86, 85, 87, 93] is a recently proposed condensing algorithm whose main goal is fast execution of the data reduction procedure rather than high reduction rates. It is also based on the concept that the non-close-class-border items are redundant and can be removed. PSC uses k-means clustering [79, 133] in order to find clusters in the training set. For each homogeneous cluster (i.e., clusters that contain only items of a specific class), it keeps only the training item that is the closest to the cluster mean. For each non-homogeneous cluster, it keeps only the items that define the decision boundaries between different classes in the cluster.

A disadvantage of PSC is that it is parametric. The user has to determine the number of clusters that will be created through a trial-and-error procedure. Although, its condensing set is independent of data order (it builds the same condensing regardless the order of data in the training set), the choice of the initial means for k-means clustering influences the contents of the final condensing set. Different initial means, lead to different clusters, and consequently, different condensing sets are built.

Section 6.3 presents PSC in more detail and proposes improvements.

Other condensing algorithms

There are many other condensing algorithms that either extend CNN-rule or are based on the same idea, that is, the removal of the non-close-border data. Some of these algorithms are the Reduced Nearest Neighbour (RNN) rule [50], the Selective Nearest Neighbour (SNN) rule [107], the Modified CNN rule [33], the Generalized CNN rule [25], the Fast CNN algorithms [9, 10], Tomek's CNN rule [121], the Patterns with Ordered Projection (POP) algorithm [106, 3], the recently proposed Template Reduction for k-NN (TRkNN) [42] and, of course, the IB2 algorithm [5, 4].

Before concluding this short review, we should mention that some condensing algorithms integrate the idea of editing. These algorithms are called hybrid [48]. Hybridization is achieved by combining condensing and editing mechanisms in a data reduction procedure. These prototype selection algorithms try to remove the non-close-border items and, at the same time, smooth border areas and remove noise and mislabelled items. Algorithms that belong to the DROP family [131] (or RT algorithms as called in [130]) are characteristic examples of hybrid algorithms. IB3 [5, 4] and C-Pruner [143] are other typical examples than belong to this category.

For the interested reader, in Garcia et al [48], all types of prototype selection algorithms (editing, condensing and hybrid) are reviewed and compared to each other. Moreover, the paper introduces taxonomies of existing prototype selection algorithms.

2.1.4 Prototype abstraction algorithms

Although prototype abstraction algorithms have the same motivation with condensing approaches, they differ on the way they build the condensing set. Contrary to condensing algorithms that select some "real" training items as prototypes, prototype abstraction algorithms generate new prototypes by summarizing on similar items. Actually, a k-NN classifier that adopts the idea of prototype abstraction runs over an artificial training set. In this subsection, we present some well-known prototype abstraction algorithms.

The algorithm of Chen and Jozwik

Chen and Jozwik have proposed a well-known and effective prototype abstraction algorithm. Chen and Jozwik's algorithm (CJA) [23] initially retrieves the most distant items, x and y in the training set. The distance between x and y determines the diameter of the dataset. Then, based on these two items, CJA divides the training set into two subsets: items that lie closer to x are placed in S_x whereas items that lie closer to y are placed in S_y. CJA proceeds by selecting to divide subsets that contain items of more than one classes (non-homogeneous subsets). The non-homogeneous subset with the largest diameter is divided first. If all subsets are homogeneous, CJA continues by dividing the homogeneous subsets. This procedure continues until the number of subsets becomes equal to a user specified value. In the end, for each created subset S, CJA averages the items in S and creates a mean item that is assigned the label of the majority class in S. The mean items created constitute the final condensing set.

The mean item m of each subset S, is computing by averaging the t attribute values of items x_i, $i = 1, 2, \ldots, |S|$ that belong to S. Therefore, the t average attributes $m.d_j$ of m are computed as follows:

$$m.d_j = \frac{1}{|S|} \sum_{x_i \in S} x_i.d_j, j = 1, 2, \ldots, t$$

Algorithm 6 lists in pseudo-code a possible implementation of CJA. It accepts a training set (TS) and the number of prototypes, n, that will be generated. The algorithm uses a data structure to store the subsets created. Initially, the entire TS constitutes a subset and is stored in TS (lines 1–2). Then, the non-homogeneous subset C with the largest diameter is divided into two subsets (lines 4,8). If all subsets are homogeneous, CJA divides the homogeneous subset C with the highest diameter (lines 5–7). Both subsets are added to S, while C is removed (lines 9–11). The procedure of constructing subsets continues until n subsets have been created (line 3). The last step is the mean computation (or prototype generation) for each subset and its inclusion in the condensing set CS (lines 13-18).

CJA selects the next subset that will be divided by examining its diameter. The idea is that a subset with a large diameter probably contains more training items. Therefore, if this subset is divided first, a higher reduction rate will be achieved. A desirable property is that CJA builds the same condensing set regardless of the ordering of the data in the training set. However, it has two weak points. The first is that the algorithm is parametric. The user has to specify the number of prototypes. This usually involves tuning via a costly trial-end-error procedure. In certain domains, this property may be desirable, since it allows one to control the size of the condensing set. However, it prohibits the automatic determination of the size of the condensing set in accordance with the nature of the available data. The second weakness is that the items that do not belong to the most common class of the subset are not represented in the condensing set. Since the mean item of each subset is labelled by the most common class, items that belong to other classes are practically ignored.

Reduction by Space Partitioning algorithms

The Chen and Jozwik algorithm constitutes the ancestor of the family of Reduction by Space Partitioning (RSP) algorithms [114] that are a popular set of three prototype abstraction algo-

Algorithm 6 CJA

Input: TS, n
Output: CS

```
1: $S \leftarrow \varnothing$
2: add($S, TS$)
3: for $i = 2$ to $n$ do
4:     $C \leftarrow$ select the non-homogeneous subset $\in S$ with the largest diameter
5:     if $C == \varnothing$ {All subsets are homogeneous} then
6:         $C \leftarrow$ select the homogeneous subset $\in S$ with the largest diameter
7:     end if
8:     $(S_x, S_y) \leftarrow$ divide $C$ into two subsets
9:     add($S$, $S_x$)
10:    add($S$, $S_y$)
11:    remove($S$, $C$)
12: end for
13: $CS \leftarrow \varnothing$
14: for each subset $T \in S$ do
15:     $r \leftarrow$ compute the mean item by averaging the items in $T$
16:     $r.label \leftarrow$ find the most common class label in $T$
17:     $CS \leftarrow CS \cup \{r\}$
18: end for
19: return $CS$
```

rithms known as RSP1, RSP2, and RSP3. RSP1 deals with the second drawback of CJA. More specifically, RSP1 computes as many mean items as the number of different classes in each subset. Therefore, it averages the items that belong to each class in the subset. Obviously, RSP1 builds larger condensing sets than CJA. However, it attempts to improve accuracy since it takes into account all training items (it does not ignore items).

RSP1 and RSP2 differ on how they select the next subset to be divided. Similar to CJA, RSP1 uses the subset diameter as the splitting criterion, based on the idea that the subset with the larger diameter may contains more training items, and so, a higher reduction rate could be achieved. In contrast, RSP2 uses as its splitting criterion the highest overlapping degree. This criterion assumes that items that belong to a specific class lie as close to each other as possible while items that belong to different classes lie as far as possible. According to [114], it is better to divide the subset with the highest overlapping degree. The overlapping degree of a subset is the ratio of the average distance between items belonging to different classes and the average distance between items that belong to the same class.

RSP3 adopts the concept of homogeneity. It continues splitting the non-homogeneous subsets and terminates when all of them become homogeneous (i.e., contain items of a specific class only). RSP3 can use either the largest diameter or the highest overlapping degree as spiting criterion. Actually, since all non-homogeneous subsets are divided, the choice of splitting criterion becomes an issue of secondary importance. RSP3 is the only RSP algorithm (CJA included) that automatically determines the size of the condensing set (it does not use any input parameter). Consequently, RSP3 eliminates both weak points of CJA. It is worth mentioning that like CJA, RSP1 and RSP2, the condensing set built by RSP3 does not depend on the data order in the training set.

Algorithm 7 lists the pseudo-code of RSP3. It utilizes a simple data structure S to hold the unprocessed subsets. Initially, the whole training set (TS) is an unprocessed subset and is placed in S (line 2). At each repeat-until iteration, RSP3 selects the subset C with the highest splitting criterion value (line 5) and checks if C is homogeneous or not. If it is homogeneous, the mean item is computed by averaging the items in C and is paced in the condensing set (CS) as prototype (lines 6–9). Otherwise, C is divided into two subsets D_1 and D_2 (line 11) in the CJA fashion. These new subsets are added to S and C is removed from S (lines 12–15). The repeat-until loop continues until S becomes empty (line 16), i.e., all subsets are homogeneous.

Considering RSP3, we observe that it generates few prototypes for representing non close-class-border areas, and many prototypes for representing close-class-border areas. Certainly, the reduction rate achieved by RSP3 highly depends on the level of noise in the data. The higher the level of noise in the data, the smaller subsets created and, as a consequence, the lower reduction rate achieved. It is worth mentioning that finding the most distant items in each subset implies the computations of all distances between the items of the subset. Hence, they are costly procedures which deteriorate the overall preprocessing cost of the algorithm. In cases of large datasets, this drawback may render its execution prohibitive.

Other prototype abstraction algorithms

We have implemented RSP3 in C. It is used for comparison purposes in the dissertation. Furthermore, RSP3 is available on WebDR[4] (see Appendix A). We presented in detail only this algorithm as well as its ancestors. However, there are many more prototype abstraction algorithms.

[4] `https://ilust.uom.gr/webdr`

Algorithm 7 RSP3

```
Input: $TS$
Output: $CS$
 1: $S \leftarrow \varnothing$
 2: add($S, TS$)
 3: $CS \leftarrow \varnothing$
 4: repeat
 5:     $C \leftarrow$ select the subset $\in S$ with the highest splitting criterion value
 6:     if $C$ is homogeneous then
 7:         $r \leftarrow$ calculate the mean item by averaging the items in $C$
 8:         $r.label \leftarrow$ class of items in $C$
 9:         $CS \leftarrow CS \cup \{r\}$
10:     else
11:         $(D_1, D_2) \leftarrow$ divide $C$ into two subsets
12:         add($S$, $D_1$)
13:         add($S$, $D_2$)
14:         remove($S$, $C$)
15:     end if
16: until IsEmpty($S$)
17: return $CS$
```

Algorithms that are based on Learning Vector Quantization (LVQ) are traditional algorithms of that type. Initially, Kohonen proposed a set of four LVQ-based algorithms [72] (Also, see Kohonen et al. [73]). Like Chen and Jozwik's algorithm, RSP1 and RSP2 ,they allow the user to choose the trade-off between accuracy and reduction rate by determining the size of the condensing set via input parameters. To the best of our knowledge, five other LVQ-based prototype abstraction algorithms based on Kohonen's work are available in the literature: VQ [135], LVQ with Training Count (LVQTC) [83], adaptive LVQ [136], hybrid LVQ3 [71] and LVQ with pruning (LVQPRU) [75]. The experimental study presented in [123] proves that the LVQ-based approaches achieve generally worse accuracy than the conventional 1-NN classifier. Moreover, the study notes that LVQ-based approaches do not work well on large datasets [5].

Some other well-known prototype abstraction algorithms are based on clustering preprocessing procedures. The Self Generating Prototypes (SGP) [43], the Symbolic Nearest Mean Classifier (SNMC) [31, 30], and the Generalized Modified Chang's Algorithm (GMCA) [82]

[5]The study considered that a large dataset contains more than 2000 items. The largest dataset had 19020 items

are characteristic examples. The latter algorithm is based on the idea of the earliest and well-known prototype abstraction algorithm introduced by Chang [21].

Interested readers can find a survey of that type of DRTs in Triguero et al. [123]. More specifically, in this study prototype abstraction algorithms are reviewed, categorized and compared to each other.

2.2 Cluster-based methods

Although cluster-based methods use entire training sets, they reduce the computational cost of k-NN classification. Each cluster-based method consists of two parts: (i) a preprocessing algorithm that builds clusters in the training set, and (ii) a classifier which utilizes these clusters in order to perform fast classifications. In the classification phase, the classifier dynamically forms a subset of the training data at the time an unclassified item should be classified. This training subset is called reference set and is used to classify the particular item (using k-NN). Actually, the reference set consists of the items of one or more clusters built by the preprocessing phase. In this way, cluster-based methods compute fewer distances than the conventional k-NN classifier.

Hwang and Cho proposed an effective cluster-based method [62] that uses the k-means clustering algorithm [79, 133] to find clusters in the data. Each cluster is divided into the core and the peripheral sets. Items lying in a certain distance from the cluster mean are characterized as "core" items, whereas the rest are characterized as "peripheral" items. If an unclassified item x lies within the "core area" of the nearest cluster, it is classified by the k-NN classifier executed on the items of this cluster. Otherwise, the k-NN classifier is executed on the set dynamically formed by the union of the items of the nearest cluster and the "peripheral" items of adjacent clusters.

We coded the Hwang and Cho method in C. Algorithms and techniques contributed by the dissertation are compared to this method in experimental studies presented in chapters that follow. Section 6.4 presents the Hwang and Cho method in detail. Moreover, it proposes some improvements that enable it to achieve even better classification performances.

There are many other cluster-based methods of interest. The Cluster-based Tree [140] is a method that is based on searching in a cluster hierarchy and can be used in either metric or non-metric spaces. [128] presented an algorithm for fast k-NN classification that prunes the search space by using the k-means clustering and the triangle inequality. Finally, [69] proposed a cluster-based method for fast time series classification, that can be used for any type of data.

2.3 Multi-attribute indexing methods

Multi-attribute indexes [112, 139, 22] can greatly reduce distance computations during nearest neighbour searches. Therefore, they can be used to speed-up the k-NN classifier. Indexes avoid the exhaustive search of the data by pruning the metric space during search. Chavez et al [22] published a taxonomy involving the most significant indexing algorithms for nearest neighbour searching in metric spaces.

The most significant indexes are based on tree data structures. Some of them are the popular k-dimensional-tree (k-d-tree) [45, 14], the k-dimensional-B-tree (k-d-B-tree) [108], the Vantage Point-tree (VP-tree) [138], the Fukunaga and Narendra algorithm [46], the Geometric Near-neighbour Access Tree (GNAT) [18], the M-tree [26], the Approximation and Elimination Search Algorithm (AESA) [111, 127], the linear AESA [81, 80], and of course, the R-tree [52] and its variations [77]. The branch and bound algorithm proposed in [122] and enhanced in [24] and the incremental algorithm introduced in [60] are approaches that efficiently compute nearest neighbours using indexes of the R-tree family.

Contrary to DRTs, indexes increase the storage requirements. The training data has to be always available and the index must be stored, too. In addition, such methods can be applied only on datasets with moderate dimensionality (e.g., 2-10). In higher dimensions, the use of indexing makes no sense. The phenomenon of "dimensionality curse" renders indexes irrelevant since their performance degrades rapidly and can become worse than that of the exhaustive search of the whole database [129]. Many proposals for speeding up nearest neighbour searches rely on dimensionality reduction in order to effectively apply an index (e.g., Principal Component Analysis (PCA) [67]). However, this can often lead to significant information loss. Moreover, each query (unclassified) item has to be transformed before the search. A model on the effect dimensionality reduction has on the similarity search performance is presented in [2].

2.4 *k*-means clustering

The k-means clustering algorithm [79, 133] is the base of most algorithms and techniques proposed in the dissertation. Therefore, a brief overview of the particular algorithm is presented here. k-means clustering is a simple algorithm that is popular for cluster analysis [41, 63, 16]. It aims to group items into k clusters, where each item belongs to the cluster with the nearest mean[6].

Given a set of items, $X = \{x_1, x_2, \ldots, x_n\}$, k-means clustering aims to assign the n items to k clusters ($k \leq n$) $S = \{S_1, S_2, \ldots, S_k\}$ so as to minimize the following function (within-cluster sum of squares):

$$E = \sum_{j=1}^{k} \sum_{x_i \in S_j} ||x_i - \mu_j||^2$$

where μ_j is the mean of cluster S_j and $||x_i - \mu_j||$ is the chosen distance metric between the data point x_i and the corresponding mean.

Algorithm 8 depicts the pseudo-code of k-means clustering. The algorithm works as follows: Given a dataset $X = \{x_1, x_2, \ldots, x_n\}$ and a set of k initial means $M = \{m_1, m_2, \ldots, m_n\}$, the algorithm builds k clusters (S). Initially, it assigns each item x_i to the nearest cluster mean m_j. This step is called assignment step (lines 6–12). When no item is pending, it re-calculates the new k means, M, by averaging the corresponding items of the clusters (lines 13-21), i.e.,

$$m_j = \frac{1}{|S_j|} \sum_{x_i \in S_j} x_i$$

Then, the algorithm re-executes the assignment step by taking into consideration the new means. As a result, the k means are adjusted at each step. The algorithm terminates when the means do not change in one iteration (cluster consolidation) (line 22). Since each item is closer to the mean of the cluster where it belongs to, the within-cluster sum of squares function is minimized.

Certainly, the algorithm is quite sensitive to the selection of the initial means. Different initial means lead to different clusters. Since the algorithm is very fast, it can be ran many times in order to reduce this effect. Forgy and random partitions are widely-used initialization methods [54]. Forgy randomly selects k items and uses them as the initial means. The random

[6]One should not confuse k of k-NN with k of k-means.

Algorithm 8 K-MEANS

```
Input: $X = \{x_1, x_2, \ldots, x_n\}$, $M = \{m_1, m_2, \ldots, m_k\}$
Output: $S = \{S_1, S_2, \ldots, S_k\}$
 1: for each item $x_i \in X$ do
 2:     $x_i.nearest\_mean \leftarrow$ NULL
 3: end for
 4: repeat
 5:     $move \leftarrow FALSE$
 6:     for each item $x_i \in X$ do
 7:         $nm \leftarrow m_j$, such that $m_j \in M$ is the nearest mean to $x_i$
 8:         if $nm \neq x_i.nearest\_mean$ then
 9:             $x_i.nearest\_mean \leftarrow nm$
10:             $move \leftarrow TRUE$
11:         end if
12:     end for
13:     if $move == TRUE$ then
14:         for $j = 1$ to $k$ do
15:             $S_j \leftarrow \varnothing$
16:             for each $x_i$ with $x_i.nearest\_mean == m_j$ do
17:                 $S_j \leftarrow S_j \cup \{x_i\}$
18:             end for
19:             $m_j \leftarrow$ compute mean item of $S_j$
20:         end for
21:     end if
22: until $move == FALSE$ {no item has moved}
```

partitions method initially assigns a random cluster to each item and then calculates the means. Note that $k$means++ is a more efficient initialization method [11].

Many variations of k-means clustering have been proposed in the literature. Here, we presented only its simplest version. The particular version is the most popular one and has been integrated in many data mining tools. In addition, it has been coded in the context of the dissertation and is utilized by the proposed algorithms. Although our goal is not to review the k-means clustering algorithms, we should mention that [68] presents an efficient version of k-means clustering that is sped-up by the utilization of k-d-tree [45, 14]. The interested readers can find a review of k-means clustering algorithms in [133].

Chapter 3

Data reduction techniques through forming homogeneous clusters

3.1 Introduction

This chapter focuses on data reduction in terms of item reduction and not attribute or dimensionality reduction. It contributes a family of four novel Data Reduction Techniques (DRTs) that are based on the concept of forming homogeneous clusters in the training data, i.e., clusters that contain only items of a particular class. The algorithms are quite simple and can be easily integrated in many existing data mining software tools. Main motives and challenges behind the proposed algorithms constitute the fast and non-parametric preprocessing of the training data. Certainly, high classification performance is also a primary goal.

Section 3.2 proposes an effective prototype abstraction algorithm, which is called Reduction through Homogeneous Clusters (RHC) [92, 91]. It has low preprocessing cost and achieves high reduction rates while maintaining accuracy at high levels. The proposed algorithm is based on a recursive fast clustering procedure that forms homogeneous clusters. The means of these clusters constitute the final condensing set. Moreover, a dynamic RHC version (dRHC) [91] is introduced that retains all the desired properties of RHC and, in addition, it can manage datasets that cannot fit in main memory. Hence, it is appropriate for dynamic / streaming environments where new training data is gradually available. The latter means that new training data can update the condensing set without the need of the "old" removed items. Experimental results, based on several known datasets, illustrate that RHC and dRHC are faster and achieve higher reduction rates than state-of-the-art DRTs, while maintaining high classification accuracy.

The research objective of Section 3.3 is how to improve the quality of the training data. It presents an editing algorithm, which is called Editing through Homogeneous Clusters (EHC) [98]. EHC tries to improve the quality of the training data by removing noise and mislabelled data as well as outliers and overlaps between data regions of different classes. It is based on the clustering procedure of RHC that forms homogeneous clusters. The clusters that contain only one item are considered irrelevant (they contain noise, outlier or close-border items) and are removed. Contrary to all other editing approaches, EHC is independent of input (tuning) parameters. An experimental study with ten datasets shows that EHC is very fast and achieves comparable classification accuracy to the well-known editing algorithms.

Section 3.4 combines the idea of RHC with that of EHC and proposes the Editing and Reduction through Homogeneous Clusters (ERHC) algorithm [89], a variation of RHC algorithm that can handle datasets with noise. ERHC is based on the clustering procedure of RHC. However, the clusters that contain only one item are considered to be noise and are removed. The mean items of the other homogeneous clusters are placed in the condensing set as prototypes. The experimental study illustrates that ERHC is as fast as RHC, but it achieves higher reduction rates and accuracy, especially when the dataset contains noise.

3.2 Data abstraction through homogeneous clusters

3.2.1 Motivation and contribution

Although many condensing and prototype abstraction algorithms that have been proposed in the past decades can be characterized as effective methods, they usually exhibit one or more of the following weaknesses:

1. They usually involve a costly, time-consuming preprocessing step on the training set, which may be prohibitive for large datasets.

2. Many condensing and prototype abstraction algorithms are parametric. The user has to provide the values for a number of parameters in advance. This usually involves tuning via an iterative execution of a trial-and-error procedure.

3. The resulting condensing set of many condensing and prototype abstraction algorithms depends on the order of items in the training set. This means that some algorithms may build a different condensing set when processing the items of a specific training set in a different order.

4. Usually, there is a trade-off between data reduction and classification accuracy. Although some condensing and prototype abstraction algorithms can achieve high reduction rates, the accuracy of the classifier is negatively affected. On the other hand, there are algorithms that produce condensing sets that achieve accuracies close to those achieved by the non-reduced training sets, but their reduction rates are not high.

5. Most condensing and prototype abstraction algorithms are memory-based. This implies that the whole training set must reside in main memory. Thus, they are inappropriate for very large datasets that cannot fit into main memory or for devices with limited main memory (e.g., sensor devices).

6. Most condensing and prototype abstraction algorithms cannot consider new training items after the construction of the condensing set. These algorithms are inappropriate for dynamic/streaming environments [1] where new training items are gradually available.

To address the last two weaknesses one needs incremental (or dynamic) algorithms that are capable of updating their condensing set when additional training data segments become available after the construction of the condensing set and without requiring all previously used training items. The aforementioned observations and the need for fast k-NN classification in large and high dimensional datasets constitute the motivation of the work presented in this section. The contribution of the section is summarized as follows:

- We proposed a prototype abstraction algorithm able to cope with the first four weaknesses. In particular, we propose and evaluate a fast, non-parametric, independent of data order, and easy to implement prototype abstraction algorithm that achieves high reduction rates and accuracy measurements. The algorithm, which we call RHC (Reduction through Homogeneous Clusters) [92], is based on the well-known k-means clustering algorithm, and thus, it can be easily integrated in many existing environments.

- We propose and evaluate a dynamic version of RHC (dRHC) [92, 91] that retains all the properties of RHC and, in addition, it is capable of dynamically updating its condensing set. Consequently, dRHC can deal with very large datasets which cannot fit into the device's memory and it is appropriate for dynamic/streaming environments.

The rest of this section is organized as follows. Subsection 3.2.2 considers in detail the proposed RHC algorithm. Subsection 3.2.3 presents its dynamic variation. In Subsection 3.2.4, both algorithms are experimentally compared to known condensing and prototype abstraction algorithms on several datasets. The experimental results are statistically validated by the Wilcoxon signed ranks test. Finally, Subsection 3.2.5 concludes the section.

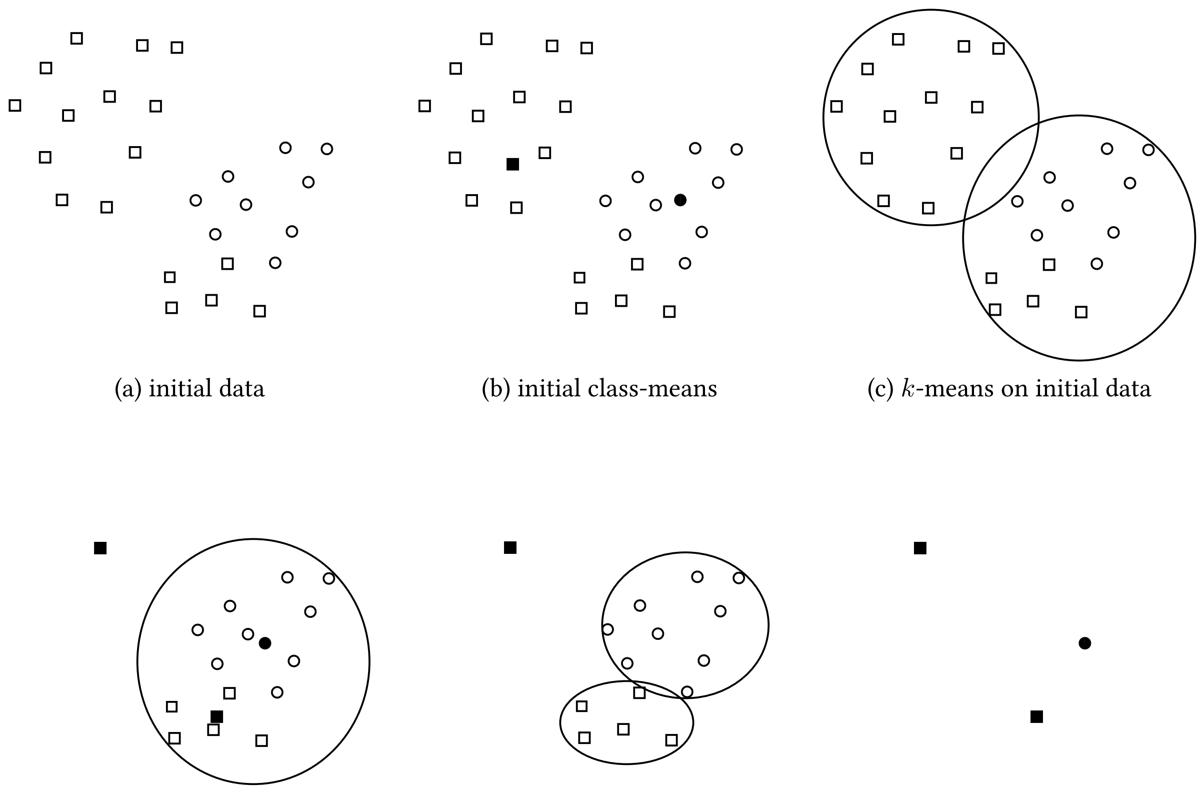
3.2.2 The Reduction through Homogeneous Clusters (RHC) algorithm

The Reduction through Homogeneous Clusters (RHC) algorithm is a non-parametric prototype abstraction algorithm. It is based on a simple idea that recursively applies the well-known k-means clustering. Particularly, RHC keeps on constructing clusters until all of them are homogeneous, i.e., they contain items only of a specific class.

Initially, RHC considers the whole training set as a non-homogeneous cluster. The algorithm begins by computing the mean for each class by averaging the attribute values of the corresponding items in the training set. Therefore, for a dataset with n classes, the algorithm computes n class-means. Then, RHC executes k-means clustering using the n aforementioned class-means as initial means and builds n clusters. For each homogeneous cluster, its mean is placed in the condensing set as prototype. For each non-homogeneous cluster, the above procedure is applied recursively. The algorithm stops when all clusters are homogeneous. In the end, the condensing set contains all the mean items of the homogeneous clusters. Note that using the class-means as initial means for k-Means clustering, the number of clusters is determined automatically.

The mean item m of each cluster or class C, is computing by averaging the n attribute values of items x_i, $i = 1, 2 \ldots |C|$ that belong to C. More formally, the n attributes $m.d_j$ of m is estimated as follows:

$$m.d_j = \frac{1}{|C|} \sum_{x_i \in C} x_i.d_j, j = 1, 2, \ldots, n$$



(a) initial data (b) initial class-means (c) k-means on initial data

(d) Cluster-mean and class-means in a non-homogeneous cluster (e) k-means on a non-homogeneous cluster (f) final set of homogeneous cluster-means (condensing set)

Figure 3.1: Data abstraction through RHC

Figure 3.1 presents a two-dimensional example of RHC execution. Suppose that a dataset contains twenty six items of two classes: squares and circles (Figure 3.1(a)). RHC computes a class-mean for the squares and a class-mean for the circles (Figure 3.1(b)). Then, k-means clustering uses the two class-means as initial means and constructs two clusters. One cluster contains only squares while the other cluster contains items of both classes (Figure 3.1(c)). For the homogeneous cluster, RHC stores the cluster-mean in the condensing set as a prototype of class square (Figures 3.1(d)). For the items of the non homogeneous cluster, RHC recursively builds two homogeneous clusters (Figures 3.1(d,e)). Consequently, two more prototypes are stored in the condensing set. Thus, the final condensing set contains three prototypes instead of the twenty six items of the initial training set (Figure 3.1(f)).

Obviously, RHC generates many prototypes for close-class-border data areas and few prototypes for the "central" class data areas. Therefore, the more classes and noise in the data, the more borders exist, and thus, lower reduction rate is achieved. In effect, when the algorithm is executed over a noise-free dataset, it forms few large clusters. On the other hand, if a dataset with high level of noise is used, many small clusters are constructed. Moreover, by using the class-means as initial means for the k-means clustering, RHC increases the probability of quickly finding large homogeneous clusters and achieving a high reduction rate (the larger the homogeneous clusters constructed, the higher the reduction rate achieved).

Algorithm 9 shows a non-recursive RHC implementation. It uses a queue data structure, $Queue$, to hold unprocessed clusters. Initially, the whole training set (TS) constitutes an unprocessed cluster and is put in $Queue$ (line 3). At each repeat-until iteration, RHC dequeues cluster C from the head of $Queue$ (line 7) and checks whether C is homogeneous or not. If it is (line 8), its mean is placed in the condensing set (CS) as a prototype (line 10) and its items are removed. Otherwise, RHC computes a list of class-means (M), one for each of the distinct classes that exist in C (lines 13–16). Then, RHC calls k-means, with parameters the current non homogeneous cluster C and the list of the initial class-means M to be used as initial means. The result is a new set of unprocessed clusters ($NewClusters$) (line 17) all of which are put into $Queue$ (lines 18–20). The repeat-until loop continues until $Queue$ becomes empty (line 22), i.e., there are no more non-homogeneous clusters.

In effect, RHC combines the idea of RSP3 with that of PSC. It retains their advantages and avoids their weaknesses. Let's recall that PSC is a fast and parametric algorithm, while RSP3 is a non-parametric algorithm that involves high pre-processing cost due to the procedure for finding the most distant items in each subset. Thus, RSP3 is inappropriate for large datasets. Contrary to PSC, RHC is a non-parametric algorithm. Contrary to RSP3, RHC is fast since it is based on a version of k-means clustering that is sped-up by the class-mean initializations. Note that we have adopted the full cluster consolidation (no item re-assignment during a complete pass of data) for the stopping condition of k-means clustering. RHC could have become even faster had we used a more efficient stopping condition. In addition, contrary to CNN-rule, IB2 and many other prototype abstraction and condensing algorithms, the effectiveness of RHC does not depend on the order of items in the training set.

Algorithm 9 RHC

```
Input: $TS$
Output: $CS$
1: {Stage 1: Queue Initialization}
2: $Queue \leftarrow \varnothing$
3: Enqueue($Queue$, $TS$)
4: {Stage 2: Construction of condensing set}
5: $CS \leftarrow \varnothing$
6: repeat
7:     $C \leftarrow$ Dequeue($Queue$)
8:     if $C$ is homogeneous then
9:         $r \leftarrow$ mean of $C$
10:        $CS \leftarrow CS \cup \{r\}$
11:    else
12:        $M \leftarrow \varnothing$ {M is the set of class-means}
13:        for each class $L$ in $C$ do
14:            $m_L \leftarrow$ mean of $L$
15:            $M \leftarrow M \cup \{m_L\}$
16:        end for
17:        $NewClusters \leftarrow K$-MEANS($C$, $M$)
18:        for each cluster $C \in NewClusters$ do
19:            Enqueue($Queue$, $C$)
20:        end for
21:    end if
22: until IsEmpty($Queue$)
23: return $CS$
```

3.2.3 The dynamic RHC algorithm

Like most DRTs, RHC is a memory based technique. This implies that the whole training set must be resident in main memory. RHC cannot manage large datasets that cannot fit in main memory. Therefore, it cannot be executed on a device with limited main memory, without transferring data to a server over a network for processing. This is a costly and time-consuming procedure.

In addition, RHC cannot handle new training items, i.e., it cannot update its condensing set in a dynamic manner. Suppose that RHC is executed over a dataset D and builds a condensing set. Then, suppose that a data segment S with new training items is available and should be taken into consideration. For the construction of an updated condensing set, RHC must be

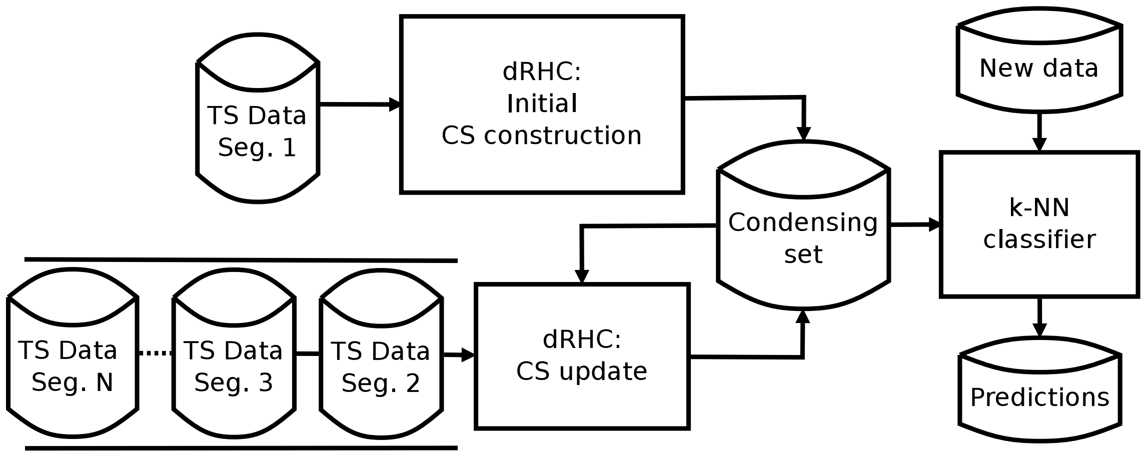


Figure 3.2: Classification procedure with dRHC

executed from scratch over the complete dataset $D \cup S$. This procedure must be repeated whenever a new data segment is available. In such a dynamic environment, all the training items, or at least a recent set of them, must always be available. In other words, storage requirements remain high.

Dynamic RHC (dRHC) is a dynamic variation of RHC. It retains all the advantages of RHC discussed in Subsection 3.2.2, but, it can cope with the weak points of RHC by considering the available data in the form of data segments. On one hand, if the dataset cannot fit in main memory, it is divided into data segments appropriate for the available main memory. On the other hand, in dynamic and/or streaming environments, since training items arrive continuously, they can be considered as data segments. In this case, the concept of data segment is implemented by using a buffer, where the new training items are stored. When the buffer is full, dRHC is run over it. Then, the training items stored in the buffer are removed and the buffer is ready to receive new training items.

The execution of dRHC has two phases: (i) *initial condensing set (CS) construction*, and, (ii) *condensing set (CS) update*. The *Initial CS construction* phase is executed only once, while the *CS update* phase is executed for each arriving data segment. Figure 3.2 depicts the complete procedure of dRHC. The algorithm begins with the *initial CS construction* phase on the available training set (TS Data Seg.1). The procedure is almost similar to RHC execution (see Algorithm 9). The only difference between RHC and the first phase of dRHC is that the latter, for each generated prototype, stores a weight value as an extra attribute. This value is the number of the training items that were clustered together and are represented by the specific prototype in the condensing set.

The *CS update* phase is also based on the concept of cluster homogeneity and the data weights. In particular, while the original RHC and the *initial CS construction* phase of dRHC begin with a single cluster that contains all the items of the training set, the *CS-update phase* of dRHC uses the prototypes of the existing condensing set and a data segment to construct a set of initial clusters and then proceeds similarly to RHC.

Algorithm 10 presents the *CS update* phase of dRHC. It takes an already constructed condensing set ($oldCS$) and a new data segment ($dataSeg$) and returns an updated condensing set ($newCS$). The algorithm begins by building the queue of unprocessed clusters. First, it initializes as many clusters, as the number of prototypes in $oldCS$ (lines 3–6). Then, each item x of $dataSeg$ is assigned to one of these clusters (lines 7–11). Finally, the clusters are enqueued to the queue data structure (lines 12–14).

The algorithm proceeds to generate $newCS$ in a way analogous to RHC, but taking into account the weight values. For a homogeneous cluster, the prototype stored in $newCS$ is the weighted mean of the cluster (lines 19–22). Similarly, for a non-homogeneous cluster C, each class-mean is computed as the weighted mean of the class items (lines 24–29). These class-means play the role of the initial means in the call to k-means for that cluster (line 30). Please, notice that in the case of dRHC, we use a version of k-means that also takes into account the item weights in the determination of the cluster-means. For a cluster (or class) C, each vector attribute d_j, $j = 1, 2, \ldots, n$ of its weight mean m_C (lines 20, 26) is estimated as follows:

$$m_C.d_j = \frac{\sum_{x_i \in C} x_i.d_j \times x_i.weight}{\sum_{x_i \in C} x_i.weight}$$

Old prototypes (from $oldCS$) usually have weights that are greater than one and have higher influence in the computation of a new weighted class or cluster-mean than any item of a new data segment, whose weight is one.

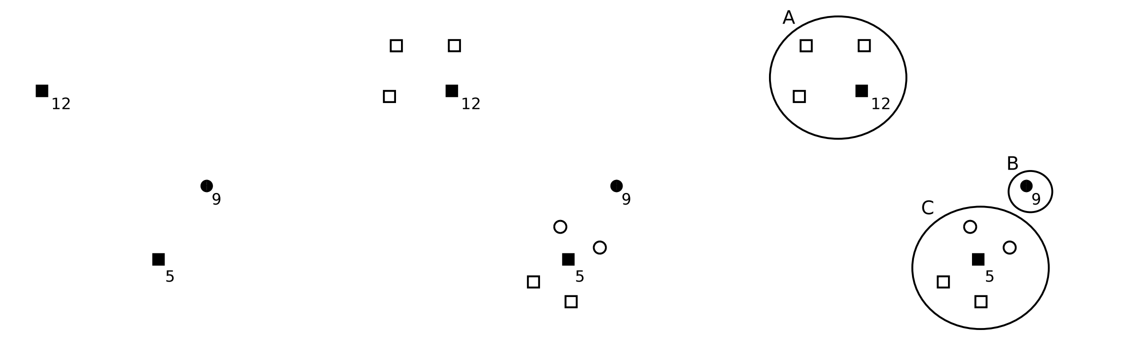
Figure 3.3 presents an example of the *CS update* phase. Suppose that an already constructed condensing set is available (Figure 3.3(a)). It contains the prototypes generated in the example of Figure 3.1 with the corresponding weight values. Moreover, suppose that a new data segment with seven new training item arrives (Figure 3.3(b)). Their initial weight is set to one. Initially, dRHC assigns each new item to the cluster of the nearest prototype (Figure 3.3(c)). Since, no items were assigned to cluster B, the corresponding prototype is not

Algorithm 10 dRHC: The *CS update* phase

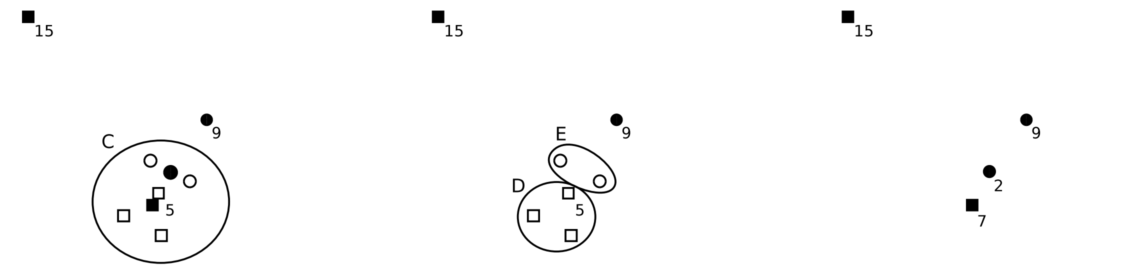
Input: $oldCS$, $dataSeg$

Output: $newCS$

```
1: {Stage 1: Queue Initialization}
2: $Queue \leftarrow \varnothing$
3: $CList \leftarrow \varnothing$ {empty list of clusters}
4: for each prototype $m \in oldCS$ do
5:    add new cluster $C = \{m\}$ in $CList$
6: end for
7: for each item $x \in dataSeg$ do
8:    $x.weight = 1$
9:    find $C_x \in CList$ with the nearest to $x$ mean
10:   $C_x \leftarrow C_x \cup \{x\}$ {do not recompute mean of $C_x$}
11: end for
12: for each cluster $C$ in $CList$ do
13:    Enqueue($Queue$, $C$)
14: end for
15: {Stage 2: Construction of newCS}
16: $newCS \leftarrow \varnothing$
17: repeat
18:    $C \leftarrow$ Dequeue($Queue$)
19:    if $C$ is homogeneous then
20:       $m \leftarrow$ weighted mean of $C$
21:       $m.weight \leftarrow \sum_{x_i \in C} x_i.weight$
22:       $newCS \leftarrow newCS \cup \{m\}$
23:    else
24:       $M \leftarrow \varnothing$ {M is the set of weighted class-means}
25:       for each class $L$ in $C$ do
26:          $m_L \leftarrow$ weighted mean of $L$
27:          $m_L.weight \leftarrow \sum_{x_i \in L} x_i.weight$
28:          $M \leftarrow M \cup \{m_L\}$
29:       end for
30:       $NewClusters \leftarrow K$-MEANS($C$, $M$)
31:       for each cluster $C \in NewClusters$ do
32:          Enqueue($Queue$, $C$)
33:       end for
34:    end if
35: until IsEmpty($Queue$)
36: return $newCS$
```



(a) A condensing set (b) New training data arrival (c) Data assignment to clusters



(d) Repositioning of a prototype and class weighted means in the non-homogeneous cluster (e) k-Means on the non-homogeneous cluster (f) The updated condensing set

Figure 3.3: Data abstraction through dRHC (The *CS update* phase)

modified. Although new items were assigned to cluster A, the latter remains homogeneous. Hence, the weighted mean of cluster A is computed and placed in the condensing set along with its new weight. In effect, the old prototype slightly "moves" towards the new items (Figure 3.3(d)). Cluster C is non-homogeneous. This means that at least one new prototype will be generated. A weighted class-mean is computed for each class in C (Figure 3.3(d)) and k-Means is executed. The result is the construction of two homogeneous clusters (Figure 3.3(e)). The weighted mean of each cluster is computed and placed in the condensing set along with its weight (Figure 3.3(f)).

Considering dRHC, we realize that the *initial CS construction* phase is more expensive than an execution of a *CS update* phase. This is because the *initial CS construction* phase begins from scratch without already constructed clusters. It begins by considering the whole dataset as an unprocessed cluster and a small number of class-means as initial means for the k-means clustering. Consequently, to obtain homogeneous clusters a high number of k-means executions is needed. In contrast, the *CS update* phase begins by assigning new data to already constructed clusters. Now, the probability of having a homogeneous cluster after the new data assignment is high. Of course, the probability of getting homogeneous clusters depends on the level of noise in the data.

We should mention that dRHC creates different condensing sets by examining the data segments in different order. However, the order of data into the data segments is irrelevant. Finally, we note that although dRHC can deal with fast data streams, it does not take into account the phenomenon of concept drift [125] that may exist in data streams.

3.2.4 Performance evaluation

Experimental setup

RHC and dRHC were evaluated using fourteen datasets distributed by the KEEL dataset repository[1] [6]. The same datasets are also available at the UCI machine learning repository[2] [12, 44]. They are summarized in Table 3.1. For comparison purposes, we used three condensing algorithms, namely, CNN-rule, IB2 and PSC, and a prototype abstraction approach, namely, RSP3. We selected these methods because: (i) CNN-rule and RSP3 are popular algorithms that are usually used in many research papers for comparison purposes, (ii) IB2, PSC and RHC have the same goal, that is, fast execution of the reduction procedure (or, low preprocessing cost), (iii) like RSP3 and PSC, RHC is based on the concept of homogeneity, and, (iv) IB2 is a fast algorithm that dynamically builds its condensing set and, thus, it is appropriate to be compared with dRHC. In addition to using condensing sets, we also measured the performance of the conventional 1-NN classifier. We note that the reader can execute RHC and dRHC as well as the aforementioned algorithms over the particular datasets on the web through WebDR[3] (see Appendix A).

All algorithm implementations were written in C and the Euclidean distance was adopted as the distance metric. The thirteen datasets (except the KDD dataset) were used without data

[1] `http://sci2s.ugr.es/keel/datasets.php`
[2] `http://archive.ics.uci.edu/ml/`
[3] `https://ilust.uom.gr/webdr`

Table 3.1: Dataset description

Dataset	**Size**	**Attributes**	**Classes**	**Segment size**
Letter Recognition (LR)	20000	16	26	2000
Magic G. Telescope (MGT)	19020	10	2	1902
Pen-Digits (PD)	10992	16	10	1000
Landsat Satellite (LS)	6435	36	6	572
Shuttle (SH)	58000	9	7	1856
Texture (TXR)	5500	40	11	440
Phoneme (PH)	5404	5	2	500
KddCup (KDD)	494020/141481	36	23	1000
Balance (BL)	625	4	3	100
Banana (BN)	5300	2	2	530
Ecoli (ECL)	336	7	8	200
Yeast (YS)	1484	8	10	396
Twonorm (TN)	7400	20	2	592
MONK 2 (MN2)	432	6	2	115

normalization. The MGT, LS, TXR and ECL datasets are distributed sorted on the class label and this affects the methods that depend on the order of data. Consequently, we randomized the order of the data items for these datasets. With the exception of the KDD dataset, no other data transformation was performed. All experiments were conducted without previous knowledge about the datasets such as data distribution, level of noise, etc.

For each dataset and algorithm, we report three average measurements obtained via five-fold cross-validation: (i) Accuracy, (ii) Reduction Rate, and, (iii) Preprocessing Cost in terms of distance computations. We report the classification accuracy of k-NN for $k = 1$ (1-NN). For all datasets (except the KDD dataset), we used the five already constructed pairs of training/testing sets hosted by the KEEL repository.

The original form of the KDD dataset has 41 attributes. However, for simplifying our experimentation procedure, we removed the three nominal and the two fixed-value attributes that exist in the dataset. In addition, many data items are duplicates. The KDD dataset contains 494,020 items, but only 141,481 of them are unique. Thus, we removed all duplicate items. Actually, duplicates are useless especially in the context of the 1-NN classification. They do not influence classification accuracy and negatively affect classification cost. Note that removal of duplicates is a common process[4] and has been adopted by the prototype selection algorithm presented in [134]. Furthermore, the attribute value ranges of the KDD dataset vary extremely. Therefore, we decided to normalized them to the range $[0, 1]$. Then, we randomized the transformed KDD dataset and divided it into the appropriate pairs for training/testing sets.

Although duplicates are useless during classification, they influence the construction of condensing set. In particular, CNN-rule and IB2 build the same condensing set regardless the number of duplicates in the training set but with higher preprocessing cost. In contrast, condensing sets built by RSP3, PSC, RHC, dRHC are influenced by duplicates because they contribute to the estimation of the mean items. For that reason, our experimental study also includes the original KDD dataset (without discarding the duplicates).

In addition, we wanted to evaluate RHC and dRHC on noise-free data. Hence, we ran our tests twice using an edited and a non-edited version of the training data. For editing purposes, we used ENN-rule and based on [131, 49, 84], we set $k = 3$. Certainly, we did not edit the testing portions of each fold. The KDD, BL, ECL and YS datasets contain some rare or weak classes. ENN-rule eliminates some of these. We note that the execution of ENN-rule over the KDD dataset is an extremely time consuming procedure. It computes $\frac{113{,}185 \times 113{,}184}{2} \times 5$ folds $\simeq$ 32 billion distances. It is worth mentioning that the SH dataset also contains rare classes. However, editing did not eliminate any rare class and, thus, we decided to include the editing procedure for that dataset in our experimentation.

Apart from PSC, all algorithms are non-parametric. For tuning the value of c for PSC (number of clusters built), we run experiments by building $c = r \times j$, $j = 2, 4, \cdots, 10$, clusters, where r is the number of discrete classes in the data, as Lopez et al. did in their experiments [86]. Hence, we built five PSC based classifiers for each dataset.

At this point, we should mention that we coded a RSP3 implementation which may compute a distance more than once. Another RSP3 implementation would not compute the same distances again and again. However, since the distances that have been already computed should be available until the end of the algorithm execution, such an implementation requires more memory and may be inefficient.

Contrary to all other methods, dRHC considers data in segments. To obtain data in a such form, we split the training sets of the datasets into segments. The last column of Table 3.1 shows the size of the data segment. Note that in some datasets, the last data segment may

[4]Many datasets distributed by the KEEL repository have been preprocessed to remove duplicates

not be complete. The size of the data segment corresponds to either the size of the available main memory (scenario of limited main memory) or the size of the data buffer (scenario of streaming/dynamic environments). The purpose of the experiment was not to test the method in a real life situation regarding memory sizes, but to assess the performance of classification on condensing sets that are constructed in a dynamic manner. Therefore, the sizes of the data segments used do not correspond to actual memory sizes. Of course, actual memory sizes can be used in real life situations.

Experimental measurements

The results of our experiments are presented in Tables 3.2–3.4 and Tables 3.5–3.7 for the non-edited and the edited datasets, respectively. Best measurements are in bold. Preprocessing cost measurements are in million distance computations (M). For reference, in Tables 3.2 and 3.5, we report the accuracy values of the conventional k-NN classifier (The 1-NN classifier applied on the non-edited training set). In addition, we present the measurements of ENN-rule. The latter reveal the level of noise in the datasets. We note that, in Table 3.7, preprocessing cost measurements of CNN, IB2, RSP3, PSC, RHC and dRHC algorithms do not include the cost of editing. In these cases, the total preprocessing cost can be computed by adding the preprocessing cost measurements of the ENN-rule column. In addition, reduction rates presented in Table 3.6 correspond to the total reduction rate: editing and data reduction. We note that the measurements of dRHC are estimated after the arrival of all data segments.

An immediate observation is that RHC and dRHC have low preprocessing cost. In almost all cases, the preprocessing cost of dRHC is lower than that of IB2 and PSC, whose major goal is to reduce the preprocessing cost.

Almost in all cases, RHC and dRHC achieve the highest reduction rates. This means that the 1-NN classifier executes faster when using a condensing set generated by these algorithms. Our measurements confirm that RSP3 is a time-consuming approach that achieves low reduction rates. However, in many cases, RSP3 has the highest accuracy, which is very close to the one measured for the conventional 1-NN classifier. RHC and dRHC appear to be more accurate than IB2 and PSC and as accurate as CNN-rule. RHC and dRHC achieved the highest accuracies in five datasets (see the BN and MN2 datasets in Table 3.2 and the BN, TN, MN2

Table 3.2: Experimental results on the non-edited datasets: Accuracy (%)

Dataset	**1-NN**	**ENN**	**CNN**	**IB2**	**RSP3**	**PSC j=2**	**PSC j=4**	**PSC j=6**	**PSC j=8**	**PSC j=10**	**RHC**	**dRHC**
LR	95.83	94.98	92.84	91.98	**95.43**	82.73	85.65	87.14	87.73	88.67	93.59	93.93
MGT	78.14	80.44	74.54	71.97	**74.69**	63.51	63.95	63.95	64.28	64.24	71.97	72.97
PD	99.35	99.30	**98.68**	98.04	99.05	95.73	96.64	96.26	96.90	96.93	98.30	98.49
LS	90.60	90.29	88.21	86.87	**90.57**	82.42	83.29	83.93	83.90	84.32	88.95	88.50
SH	99.82	99.79	**99.76**	99.73	99.75	99.67	98.24	97.93	98.82	95.96	98.09	99.65
TXR	99.02	98.64	97.16	96.35	**98.29**	96.13	94.96	94.84	94.46	94.78	97.04	97.60
PH	90.10	88.14	**87.82**	85.57	86.94	71.41	75.19	75.17	74.70	75.63	85.59	85.38
KDD	99.71	-	**99.66**	99.48	99.60	95.50	96.18	96.68	96.89	96.95	99.39	99.42
BL	78.4	-	70.88	70.72	**73.28**	65.92	66.40	70.88	68.00	68.32	68.64	70.56
BN	86.91	89.36	**85.62**	83.81	84.00	57.60	58.00	56.87	57.49	58.70	83.28	82.79
ECL	79.78	-	72.05	66.97	**73.53**	57.16	63.39	66.97	68.16	66.37	68.76	69.35
YS	52.02	-	49.06	46.02	**50.47**	46.03	45.01	47.84	46.77	47.71	48.85	48.38
TN	94.88	95.69	92.00	89.15	92.68	78.74	79.08	79.78	80.49	80.12	88.69	**93.08**
MN2	90.51	89.58	95.84	93.75	91.22	94.43	95.14	90.06	92.58	93.52	94.68	**97.68**
Avg	88.22	92.62	86.01	84.32	**86.39**	77.64	78.65	79.16	79.37	79.44	84.70	85.56

Table 3.3: Experimental results on the non-edited datasets: Reduction Rate (%)

Dataset	**ENN**	**CNN**	**IB2**	**RSP3**	**PSC j=2**	**PSC j=4**	**PSC j=6**	**PSC j=8**	**PSC j=10**	**RHC**	**dRHC**
LR	4.33	83.54	85.66	61.98	81.40	79.76	79.46	79.88	79.90	88.08	**88.18**
MGT	20.08	60.08	70.60	53.70	70.71	71.05	71.58	71.81	71.60	73.76	**74.62**
PD	0.67	95.36	96.23	89.22	91.44	92.86	93.73	94.42	94.83	96.52	**97.23**
LS	9.07	80.22	84.62	73.19	84.67	84.79	84.84	84.93	84.95	**89.84**	88.35
SH	0.18	99.37	99.44	98.59	96.88	97.68	97.87	98.33	98.54	**99.55**	99.50
TXR	1.24	91.90	93.33	83.31	86.81	89.33	90.62	91.29	91.54	94.70	**94.95**
PH	11.25	76.04	80.85	69.94	81.31	81.56	81.32	81.39	81.54	80.71	**82.34**
KDD	-	99.12	**99.26**	98.54	99.13	99.09	99.09	99.09	99.07	99.19	99.22
BL	-	65.72	69.36	64.64	77.8	77.44	78.04	77.2	75.88	78.00	**78.12**
BN	11.53	77.44	83.27	75.21	85.59	85.70	85.77	**85.89**	85.81	79.68	82.41
ECL	-	59.55	68.77	52.27	74.50	72.19	71.08	67.88	65.65	67.58	**70.26**
YS	-	32.68	44.82	27.36	**55.32**	55.25	53.84	53.81	54.23	49.83	51.23
TN	3.61	82.09	88.25	84.56	95.73	94.85	94.57	94.78	94.98	**96.63**	95.37
MN2	2.08	87.23	91.68	61.33	45.31	49.02	61.16	57.34	60.23	96.47	**96.88**
Avg	6.40	77.88	82.58	70.99	80.47	80.76	81.64	81.29	81.34	85.04	**85.62**

Table 3.4: Experimental results on the non-edited datasets: Preprocessing Cost (millions of distance computations)

Dataset	**ENN**	**CNN**	**IB2**	**RSP3**	**PSC j=2**	**PSC j=4**	**PSC j=6**	**PSC j=8**	**PSC j=10**	**RHC**	**dRHC**
LR	127.99	163.03	23.37	326.52	66.32	110.06	129.16	165.32	169.92	41.85	**19.57**
MGT	115.76	281.49	34.61	511.67	23.95	17.21	22.68	27.09	33.47	**4.08**	26.03
PD	38.65	11.75	1.78	86.66	6.52	15.93	28.48	35.23	36.97	2.88	**1.44**
LS	13.25	17.99	2.22	37.70	2.96	5.85	8.41	10.11	10.50	1.69	**1.53**
SH	1076.46	45.30	8.26	17410.18	127.20	54.07	148.35	222.77	252.61	16.83	**7.68**
TXR	9.68	5.65	0.84	27.63	3.15	7.90	10.71	14.49	16.76	3.63	**0.68**
PH	9.35	13.45	1.96	20.31	1.08	0.94	2.08	2.79	3.12	**0.66**	1.64
KDD	-	384.90	**55.58**	20278.87	212.23	575.80	1161.43	2054.23	1902.41	81.59	57.40
BL	-	0.21	0.04	0.3	0.08	0.12	0.16	0.18	0.24	0.05	**0.03**
BN	8.99	11.49	1.58	18.76	1.91	1.44	2.39	4.63	4.37	**0.56**	1.53
ECL	-	0.06	**0.003**	0.08	0.06	0.11	0.11	0.12	0.15	0.03	0.02
YS	-	1.41	**0.19**	2.12	0.70	1.17	1.64	1.94	1.99	0.84	0.31
TN	17.52	22.13	2.07	37.13	1.76	5.40	6.76	6.93	8.37	1.64	**0.70**
MN2	0.06	0.04	0.006	0.13	0.014	0.07	0.08	0.12	0.13	0.007	**0.004**
Avg	141.77	68.49	9.46	2768.43	32.00	56.86	108.75	181.85	174.36	11.17	**8.47**

Table 3.5: Experimental results on the edited datasets: Accuracy (%)

Dataset	**1-NN**	**ENN**	**CNN**	**IB2**	**RSP3**	**PSC j=2**	**PSC j=4**	**PSC j=6**	**PSC j=8**	**PSC j=10**	**RHC**	**dRHC**
LR	95.83	94.98	92.06	91.38	**94.61**	82.29	85.68	87.00	87.97	88.46	92.72	93.14
MGT	78.14	80.44	**79.26**	78.01	79.09	72.50	72.71	73.33	73.31	73.35	77.78	78.33
PD	99.35	99.30	98.60	98.17	**99.03**	97.30	97.04	97.11	97.29	97.11	98.45	98.57
LS	90.60	90.29	88.66	88.05	**89.90**	83.53	84.60	84.91	84.85	84.99	89.14	88.81
SH	99.82	99.79	**99.73**	99.72	99.67	99.56	98.40	98.53	98.82	98.41	99.58	99.62
TXR	99.02	98.64	96.93	95.75	**97.91**	96.15	95.46	95.26	94.91	95.67	97.11	97.38
PH	90.10	88.14	**86.88**	86.33	86.49	80.74	81.07	81.75	81.42	81.70	85.40	85.55
BN	86.91	89.36	88.87	88.68	88.64	81.98	81.51	82.26	80.68	80.79	88.09	**88.94**
TN	94.88	95.69	92.30	91.22	94.69	82.58	83.14	83.77	85.23	85.49	93.11	**95.45**
MN2	90.51	89.58	95.37	94.46	90.07	95.13	93.98	94.90	93.06	94.21	**96.75**	96.31
Avg	92.52	92.62	91.87	91.18	92.01	87.18	87.36	87.88	87.75	88.02	91.81	**92.21**

Table 3.6: Experimental results on the edited datasets: Reduction Rate (%)

Dataset	**ENN**	**CNN**	**IB2**	**RSP3**	**PSC j=2**	**PSC j=4**	**PSC j=6**	**PSC j=8**	**PSC j=10**	**RHC**	**dRHC**
LR	4.33	87.75	88.88	66.12	81.95	80.25	80.14	80.80	81.22	90.34	**91.00**
MGT	20.08	90.09	92.05	84.20	85.57	85.67	86.61	86.57	86.63	93.06	**93.40**
PD	0.67	96.44	97.00	90.41	91.95	93.50	94.22	95.11	95.70	97.19	**97.79**
LS	9.07	91.44	92.98	85.84	90.25	90.65	90.95	91.26	91.48	**95.09**	94.94
SH	0.18	99.58	99.61	98.88	97.10	97.89	98.04	98.55	98.68	**99.66**	99.65
TXR	1.24	93.45	94.32	85.00	87.82	90.50	91.76	92.60	92.42	95.58	**95.85**
PH	11.25	90.49	91.62	85.13	87.70	88.04	87.80	87.94	87.91	92.10	**92.43**
BN	11.53	95.31	95.87	93.72	95.66	95.78	96.02	**96.28**	96.40	95.66	95.87
TN	3.61	89.49	92.36	89.63	98.55	98.28	98.07	98.02	97.88	98.52	97.85
MN2	2.08	88.84	93.12	62.25	44.34	53.24	60.92	61.16	62.95	**97.05**	96.94
Avg	6.40	92.29	93.78	84.12	86.09	87.38	88.45	88.83	89.13	95.43	**95.57**

Table 3.7: Experimental results on the edited datasets: Preprocessing Cost (millions of distance computations)

Dataset	**ENN**	**CNN**	**IB2**	**RSP3**	**PSC j=2**	**PSC j=4**	**PSC j=6**	**PSC j=8**	**PSC j=10**	**RHC**	**dRHC**
LR	127.99	112.20	18.35	300.51	55.13	94.76	127.84	138.41	178.45	31.05	**15.15**
MGT	115.76	68.61	8.48	318.82	11.44	10.15	11.28	12.42	21.75	**2.83**	6.18
PD	38.65	9.25	1.51	85.16	6.73	17.57	27.65	32.33	33.74	2.83	**1.25**
LS	13.25	6.49	0.99	30.64	2.86	4.83	6.79	9.97	11.82	1.73	**0.72**
SH	1076.46	26.02	6.35	15652.75	107.47	52.46	176.21	189.71	213.61	22.41	**6.05**
TXR	9.68	3.90	0.72	27.04	3.35	10.33	9.60	11.10	15.78	3.00	**0.57**
PH	9.35	5.57	0.86	15.67	0.68	1.04	1.89	2.18	3.15	**0.47**	0.73
BN	8.99	2.50	0.435	14.50	1.39	1.43	2.10	2.28	2.96	0.53	**0.434**
TN	17.52	12.50	1.41	34.20	1.81	3.13	4.02	6.38	9.56	1.36	**0.34**
MN2	0.06	0.03	0.005	0.12	0.01	0.06	0.07	0.12	0.13	0.007	**0.004**
Avg	141.77	24.71	3.91	1647.94	19.09	19.58	36.75	40.49	49.10	6.62	**3.14**

datasets in Table 3.5). In the case of the MN2 dataset, RHC and dRHC are more accurate even than the conventional 1-NN and ENN-rule. Although the rest accuracy measurements of RHC and dRHC are not as good as those of RSP3, they are close enough.

Although IB2 is an one-pass version of CNN-rule, it achieved higher reduction rates with much lower preprocessing cost. However, CNN-rule seems to be more accurate than IB2. It is worth mentioning that the measurements for both algorithms would be different had we examined the same data in a different order.

Concerning the differences between Tables 3.2–3.4 and Tables 3.5–3.7 (non-edited vs edited data), we observe that except the MGT, BN and TN datasets the reported accuracies are almost similar. On the other hand, all algorithms that ran over the edited data, executed faster and generated smaller condensing sets. In the cases of the MGT, LS, PH and BN datasets, ENN-rule removed many irrelevant items ($> 9\%$) and so, the reduction rate differences are obvious. It is worth noting that in the case of the edited data, dRHC, RHC and IB2 generate their condensing set by calculating an extremely low number of distances (see Table 3.7). In addition, we observe that when RHC and dRHC are executed on the edited data they are able to create extremely small condensing sets (see Table 3.6). They contain less than 5% of items of the initial training set on average. Considering the differences between the performance of RHC and dRHC, we conclude that dRHC may be characterized as a slightly better approach than RHC.

We can make a final comment concerning the average measurements (last rows in Tables 3.2–3.7): almost in all cases, RHC and dRHC appear to build the smallest condensing sets with the lowest preprocessing cost, and a high classification accuracy, similar to that of CNN-rule.

Complementary experiments

As already mentioned, in the context of 1-NN classification, duplicates do not influence classification accuracy but affect the classification computational cost. However, they influence the data reduction procedures. Consequently, we wanted to measure the performance of the examined condensing and prototype abstraction algorithms using the original KDD dataset that contains a high number of duplicates. Especially for dRHC that dynamically builds its condensing set based on a weight-based schema, this experiment is essential. Hence, we ran experiments using the original KDD dataset (494020 items). By applying five-fold cross-validation, we obtained the measurements presented in Table 3.8.

Table 3.8: Experimental results on the original KDD dataset: Accuracy (Acc (%)), Reduction Rate (RR (%)) and Preprocessing Cost (PC (millions of distance computations))

Criterion	**IB2**	**RHC**	**dRHC**
Accuracy (%):	**99.87**	99.84	99.82
Reduction Rate (%):	**99.78**	99.77	99.76
Preprocessing Cost (M) :	**209.08**	291.91	223.97

Applying the preprocessing of CNN-rule, RSP3 and PSC on such a large dataset is an extremely time consuming procedure. Especially the execution of RSP3 is prohibitive, due to the costly retrieval of the most distant points in each subset. We decided to exclude those algorithms from this phase of our experimentation because of their high preprocessing cost. Certainly, conventional 1-NN classification is also extremely time-consuming. It needs to compute over 39 billions distances for each fold, and thus, we did not consider it in this experiment.

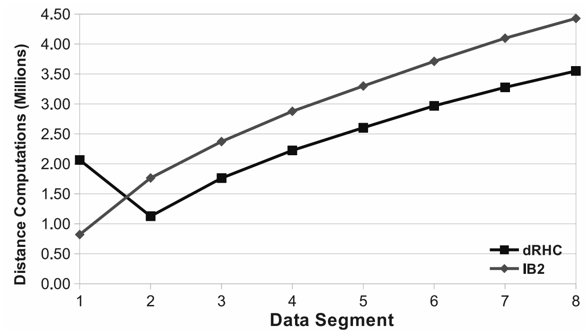
The results shown in Table 3.8 are very similar to the ones presented in Tables 3.2–3.4 concerning the KDD dataset. Of course, since the original KDD dataset contains over 70% more items (duplicates), reduction rate and preprocessing cost measurements in Table 3.8 are higher than those in Tables 3.3 and 3.4. Furthermore, since the testing portions of the original KDD dataset contain more items, higher accuracies are achieved.

Another issue that we wanted to explore is how the size of data segment influences the performance of dRHC. Therefore, we ran experiments by adopting different segment sizes. The corresponding results are presented in Table 3.9. Considering the measurements, we conclude that segment size is rather irrelevant during dRHC execution. None of the comparison criteria is substantially improved by increasing or decreasing the segment size.

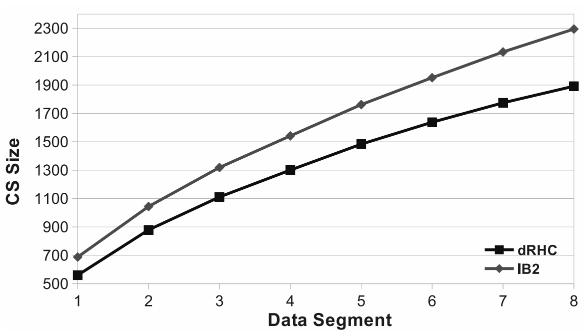
For dRHC and IB2, which are dynamic algorithms, we studied how the sizes of condensing sets are increased over time and estimated the cost needed for the preprocessing of each data segment. Indicatively, Figure 3.4 presents these measurements for the LR and PD datasets in their non-edited form. The measurements are similar for all datasets, so we do not include figures for the other datasets. Let's recall that IB2 examines each individual item and decides whether to put it in the condensing set or not. On the other hand, dRHC considers data in data segments. Regarding the diagram for preprocessing cost measurements, it reports a preprocessing cost value for each *CS update* phase of dRHC and for each pass of t items for IB2, where t is equal to the size of the data segment used by dRHC.

Table 3.9: Using different segment (Memory/Buffer) sizes during dRHC execution on the non-edited datasets (Accuracy (Acc (%)), Reduction Rate (RR (%)) and Preprocessing Cost (PC (millions of distance computations))

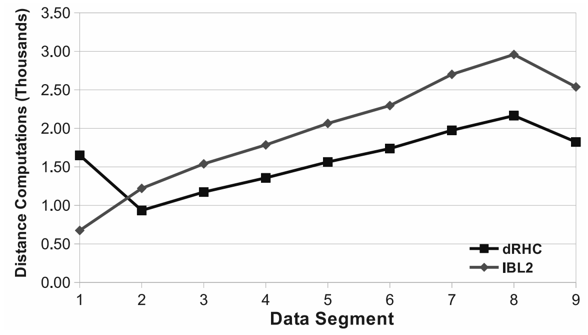
Dataset		Segment size			
		500	**1000**	**2000**	**4000**
LR	Acc:	94.20	93.40	93.93	94.12
	RR:	88.06	88.10	88.18	88.37
	PC:	19.75	19.48	19.57	19.99
MGT	Acc:	72.52	72.68	72.96	72.51
	RR:	72.92	73.63	74.59	75.29
	PC:	30.28	28.45	26.12	22.77
PD	Acc:	98.42	98.49	98.41	98.40
	RR:	97.29	97.23	97.22	96.95
	PC:	1.38	1.44	1.63	2.07
LS	Acc:	88.92	87.69	88.73	89.18
	RR:	88.24	88.94	89.48	89.41
	PC:	1.57	1.43	1.52	1.64
SH	Acc:	99.74	99.68	99.73	99.70
	RR:	99.46	99.48	99.45	99.49
	PC:	8.12	7.70	8.38	7.48
TXR	Acc:	97.31	97.56	97.82	97.29
	RR:	95.25	95.06	94.90	94.56
	PC:	0.67	0.75	1.03	2.59
PH	Acc:	85.38	84.97	85.47	85.64
	RR:	82.34	82.28	81.88	80.99
	PC:	1.64	1.55	1.34	0.82
KDD	Acc:	99.39	99.42	99.42	99.38
	RR:	99.22	99.22	99.24	99.25
	PC:	57.59	57.32	54.58	51.62
BL	Acc:	68.8	-	-	-
	RR:	78.04	-	-	-
	PC:	0.05	-	-	-
BN	Acc:	83.15	83.64	82.42	83.19
	RR:	82.63	81.98	81.63	79.86
	PC:	1.51	1.47	1.22	0.73
YS	Acc:	47.10	48.99	-	-
	RR:	49.85	49.23	-	-
	PC:	0.32	0.49	-	-
TN	Acc:	93.34	92.96	92.69	92.03
	RR:	95.19	95.75	96.37	96.31
	PC:	0.72	0.64	0.65	1.06



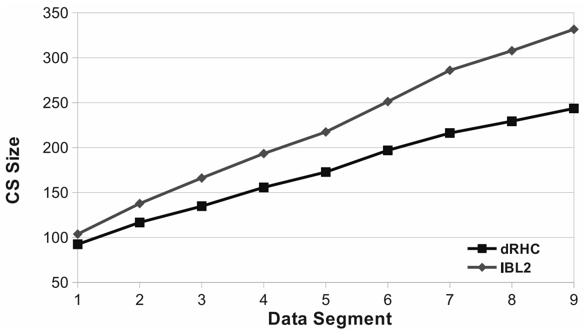
(a) LR:Preprocessing cost



(b) LR:Data reduction



(c) PD:Preprocessing cost



(d) PD:Data reduction

Figure 3.4: Processing per data segment

Both measurements increase over time because the size of the available condensing set increases, and thus, more distances need to be computed over time. The last PD data segment contains 874 items instead of 1000. Therefore, lower preprocessing cost is needed for that segment (see Figure 3.4(c)). Concerning dRHC, as we expected, the *initial CS construction* phase is more time-consuming than the following *CS update* phases (see discussion near the end of Subsection 3.2.3). Considering Figure 3.4 as well as the measurements in Tables 3.2–3.7, one can conclude that dRHC achieves better performance than IB2 in terms of all comparison criteria.

Non-parametric statistical test

We complement the section of the performance evaluation providing the results of a non-parametric statistical test of significance [116]. In particular, we used the Wilcoxon signed ranks test [32] in order to validate the experimental results presented in Tables 3.2–3.7. The test compares DRTs in pairs taking into consideration their performance on each dataset. RHC and dRHC were compared to each other and to each one of the comparison algorithms.

We ran the test four times. Once on the measurements of each comparison criterion (classification accuracy (ACC), reduction rate (RR), preprocessing cost (PC)) and once on the measurements of the overall classification performance criterion. By following the idea presented in [48, 47], we computed the measurements of the overall classification performance by averaging the measurements of the three comparison criteria. Therefore, the overall performance considered the three criteria as having the same significance. Certainly, the computation of the overall classification performance implies that the measurements of the three criteria are in the same range. Therefore, first, we normalized the measurements to the interval $[0, 1]$. Suppose that there are n performance measurements of the criterion a and they must be normalized to the interval $[0, 1]$. The normalized i-th measurement, $i = 1, \ldots, n$ is estimated as follows:

$$norm(a_i) = \frac{a_i - E_{min}}{E_{max} - E_{min}}$$

where E_{min} and E_{max} are the minimum and maximum measurements for a, respectively. Since low preprocessing cost is desirable, we used $1 - normalized(PC)$ instead of $normalized(PC)$.

Also, we ran all the tests twice, one on the measurements obtained on the non-edited datasets and one on the measurements obtained on the edited data. Notice that we do not include the measurements obtained by the experimentation on the original form of the KDD dataset.

Tables 3.10 and 3.11 present the results of the Wilcoxon test. The columns labeled with "w/l/t" show the number of wins, losses and ties respectively (e.g., in Table 3.10, in the overall performance of RHC vs CNN case, "12/2/0" means that RHC was better than CNN 12 times and worse 2 times). The Wilcoxon value (columns labeled with "Wilc.") depicts how significant the difference of the corresponding algorithms is. If it is lower than 0.05, one can claim that the difference between the two methods is statistically significant.

This is true almost in all comparison pair in terms of overall classification performance. Therefore, the results of the test confirm that RHC and dRHC perform better than the other algorithms. It is worth mentioning that the results of the statistical tests presented in Table 3.10 reveal that the difference between dRHC and RHC is statistically significant. Thus, the test confirms that dRHC performs slightly better than RHC. In terms of reduction rate, the tests

Table 3.10: Results of the Wilcoxon signed ranks test on the measurements obtained from the non-edited data

Methods	**ACC**		**RR**		**PC**		**Overall**	
	w/l/t	**Wilc.**	**w/l/t**	**Wilc.**	**w/l/t**	**Wilc.**	**w/l/t**	**Wilc.**
RHC vs CNN	2/12/0	**0.009**	14/0/0	**0.001**	14/0/0	**0.001**	12/2/0	**0.005**
RHC vs IB2	8/5/1	0.311	10/4/0	**0.030**	5/9/0	0.397	10/4/0	**0.022**
RHC vs RSP3	1/13/0	**0.009**	14/0/0	**0.001**	14/0/0	**0.001**	14/0/0	**0.001**
RHC vs PSC (j=2)	13/1/0	**0.002**	10/4/0	0.245	12/2/0	**0.011**	13/1/0	**0.002**
RHC vs PSC (j=4)	12/2/0	**0.002**	10/4/0	0.245	14/0/0	**0.001**	13/1/0	**0.001**
RHC vs PSC (j=6)	13/1/0	**0.004**	9/5/0	0.221	14/0/0	**0.001**	11/3/0	**0.005**
RHC vs PSC (j=8)	13/1/0	**0.002**	10/4/0	0.109	14/0/0	**0.001**	13/1/0	**0.002**
RHC vs PSC (j=10)	14/0/0	**0.001**	11/3/0	0.074	14/0/0	**0.001**	13/1/0	**0.002**
dRHC vs CNN	5/9/0	0.363	14/0/0	**0.001**	14/0/0	**0.001**	14/0/0	**0.001**
dRHC vs IB2	9/5/0	**0.026**	12/2/0	**0.002**	11/3/0	**0.041**	11/3/0	**0.005**
dRHC vs RSP3	2/12/0	**0.026**	14/0/0	**0.001**	14/0/0	**0.001**	14/0/0	**0.001**
dRHC vs PSC (j=2)	13/1/0	**0.001**	10/4/0	0.124	12/2/0	**0.019**	13/1/0	**0.001**
dRHC vs PSC (j=4)	14/0/0	**0.001**	11/3/0	0.064	11/3/0	**0.026**	14/0/0	**0.001**
dRHC vs PSC (j=6)	13/1/0	**0.001**	11/3/0	**0.041**	13/1/0	**0.004**	12/2/0	**0.002**
dRHC vs PSC (j=8)	14/0/0	**0.001**	12/2/0	**0.030**	14/0/0	**0.001**	13/1/0	**0.001**
dRHC vs PSC (j=10)	14/0/0	**0.001**	12/2/0	**0.026**	14/0/0	**0.001**	13/1/0	**0.001**
dRHC vs RHC	10/4/0	**0.048**	11/3/0	0.056	11/3/0	**0.109**	13/1/0	**0.006**

confirm that RHC and dRHC perform better than the other algorithms. Although IB2 has more wins than RHC in terms of preprocessing cost, the difference is not statistically significant ($Wilc. = 0.397$). In all other cases, RHC and dRHC perform better than the other algorithms in terms of preprocessing cost. In terms of accuracy, RHC is statistically better than PSC and dRHC is statistically better than IB2 and PSC. Although RHC has more wins than IB2 (8/5/1 in Table 3.10 and 6/4/0 in Table 3.11), the difference is not statistically significant (note that we have adopted a very strict threshold, i.e., $Wilc. = 0.05$, for the Wilcoxon significance level). In addition, the results in Table 3.10 confirm that RSP3 leads to the most accurate classification. Last but not least, the tests show that RHC and dRHC are as accurate as CNN-rule.

3.2.5 Conclusions

This section presented RHC, a fast non-parametric algorithm for data reduction. It uses k-means clustering to recursively cluster the training dataset into homogeneous clusters. The

Table 3.11: Results of the Wilcoxon signed ranks test on the measurements obtained from the edited data

Methods	**ACC**		**RR**		**PC**		**Overall**	
	w/l/t	**Wilc.**	**w/l/t**	**Wilc.**	**w/l/t**	**Wilc.**	**w/l/t**	**Wilc.**
RHC vs CNN	5/5/0	0.959	10/0/0	**0.005**	10/0/0	**0.005**	7/3/0	0.093
RHC vs IB2	6/4/0	0.114	9/1/0	**0.013**	3/7/0	0.169	7/3/0	0.074
RHC vs RSP3	1/9/0	0.074	10/0/0	**0.005**	10/0/0	**0.005**	10/0/0	**0.005**
RHC vs PSC (j=2)	10/0/0	**0.005**	8/1/1	**0.011**	10/2/0	**0.005**	10/0/0	**0.005**
RHC vs PSC (j=4)	10/0/0	**0.005**	9/1/0	**0.007**	10/0/0	**0.005**	10/0/0	**0.005**
RHC vs PSC (j=6)	10/0/0	**0.005**	9/1/0	**0.007**	10/0/0	**0.005**	10/0/0	**0.005**
RHC vs PSC (j=8)	10/0/0	**0.005**	9/1/0	**0.009**	10/0/0	**0.005**	10/0/0	**0.005**
RHC vs PSC (j=10)	10/0/0	**0.005**	9/1/0	**0.009**	10/0/0	**0.005**	10/0/0	**0.005**
dRHC vs CNN	6/4/0	0.386	10/0/0	**0.005**	10/0/0	**0.005**	8/2/0	**0.017**
dRHC vs IB2	8/2/0	**0.037**	9/0/1	**0.008**	9/0/1	**0.008**	8/2/0	**0.017**
dRHC vs RSP3	3/7/0	0.333	10/0/0	**0.005**	10/0/0	**0.005**	10/0/0	**0.005**
dRHC vs PSC (j=2)	10/0/0	**0.005**	9/1/0	**0.009**	9/1/0	**0.009**	10/0/0	**0.005**
dRHC vs PSC (j=4)	10/0/0	**0.005**	9/1/0	**0.009**	10/0/0	**0.005**	10/0/0	**0.005**
dRHC vs PSC (j=6)	10/1/0	**0.005**	8/2/0	**0.013**	10/0/0	**0.005**	10/0/0	**0.005**
dRHC vs PSC (j=8)	10/0/0	**0.005**	8/2/0	**0.013**	10/0/0	**0.005**	10/0/0	**0.005**
dRHC vs PSC (j=10)	10/0/0	**0.005**	8/2/0	**0.013**	10/0/0	**0.005**	10/0/0	**0.005**
dRHC vs RHC	8/2/0	0.114	6/4/0	0.241	8/2/0	0.093	8/2/0	0.059

condensing set consists of the means of the final clusters. RHC combines the advantages of RSP3 and PSC algorithms while avoiding their drawbacks. Furthermore, we presented dRHC, a dynamic version of RHC, which retains all good properties of RHC and, in addition, it supports frequent updates of the condensing set. Therefore, it can deal with datasets that cannot fit into the main memory and/or streaming training training data which are gradually available.

Experimental results, obtained by using the non-edited and edited versions of several datasets, showed that both algorithms had low preprocessing cost and achieved the highest reduction rates without significant loss of accuracy. We claim that these properties render RHC and dRHC appropriate for environments where fast classification and/or low preprocessing cost are critical. Moreover, we demonstrated that dRHC is independent of the size of data segment (memory/buffer).

3.3 Editing by finding Homogeneous Clusters

3.3.1 Motivation and contribution

The reduction rates of many condensing and prototype abstraction algorithms depend on the level of noise in the training data. In effect, the higher the level of noise, the lower the reduction rates achieved. Therefore, effective application of such algorithms implies removal of noise from the data, i.e., application of an editing algorithm beforehand [29, 76]. Hence, editing has a double goal: accuracy improvement and effective application of condensing and prototype abstraction algorithms.

Although editing algorithms contribute in obtaining high quality training data, they constitute a costly preprocessing step. Moreover, the editing algorithms are parametric, i.e., the user defines the values of certain input (tuning) parameters. This implies time-consuming trial-and-error procedures to tune the parameters. These observations are behind the motivation of the work presented in this section. The contribution is the proposal and evaluation of a fast, non-parametric editing algorithm that is based on a k-means clustering procedure that forms homogeneous clusters. This procedure is similar to that of RHC (see Section 3.2.2). The proposed algorithm is called Editing through Homogeneous Clusters (EHC), leads to accurate k-NN classifiers and has low preprocessing cost.

The rest of the section is organized as follows: Subsection 3.3.2 presents the proposed EHC algorithm. Performance evaluation experiments are presented in Subsection 3.3.3 and, finally, Subsection 3.3.4 concludes the section.

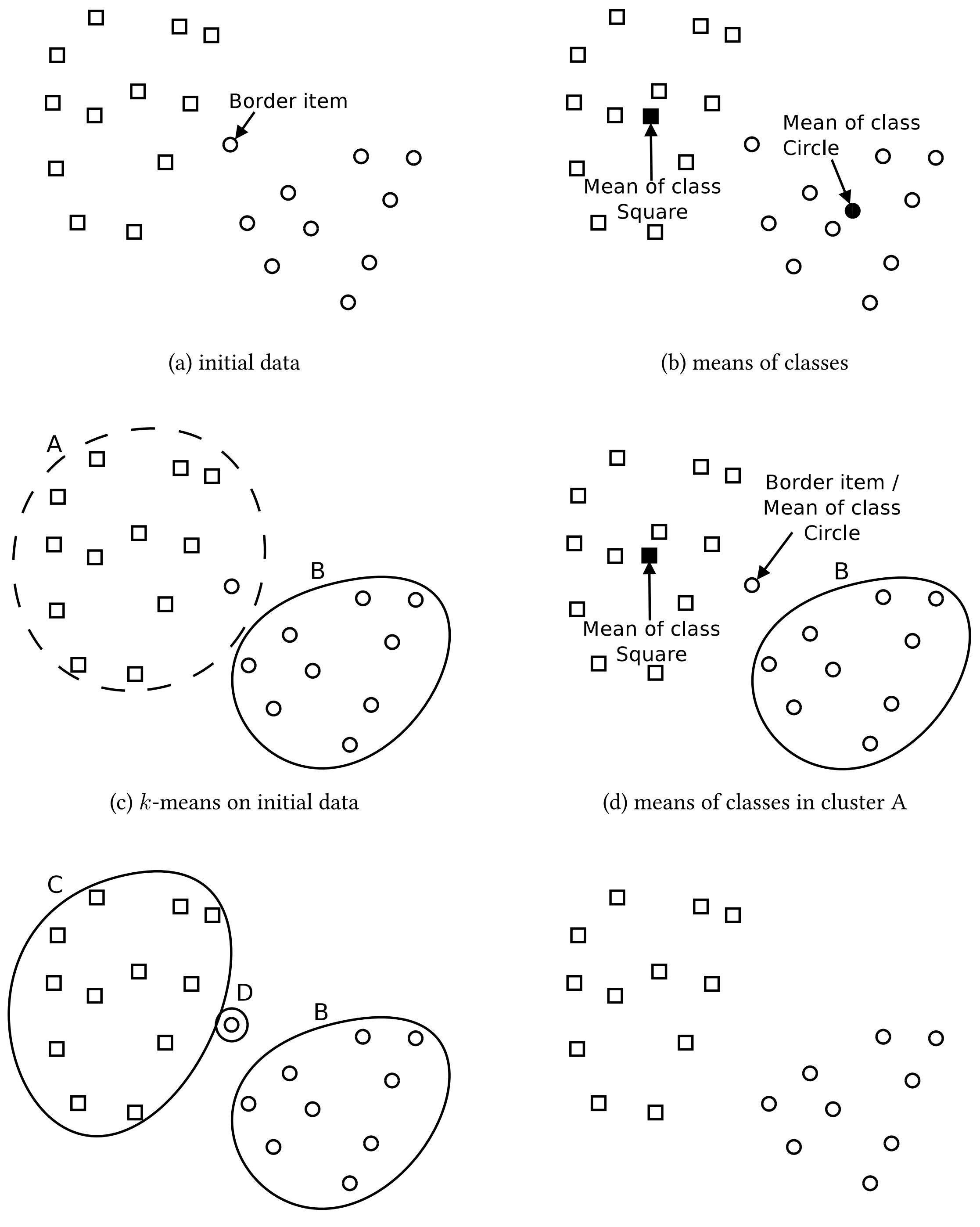
3.3.2 The Editing through Homogeneous Clusters (EHC) algorithm

As mentioned in Section 2.1.2, editing algorithms either extend ENN-rule or are based on the same idea. The proposed EHC algorithm follows a completely different, non-parametric strategy in order to remove noise, mislabelled and close-border data items. Actually, it is based on the idea of RHC. Therefore, EHC iteratively applies k-means clustering on training data until all constructed clusters containing items of a specific class only, i.e., they are homogeneous. In the process, EHC removes all the clusters that contain only one item. We call these clusters one-item clusters. The idea behind EHC is quite simple: one-item-clusters are redundant. These items are probably outliers or lie in a region of a different class (noise) or lie close to a decision-boundaries region, and thus, they must be removed.

Initially, EHC considers the whole training set to be a non-homogeneous cluster. The algorithm computes a mean item for each class (class-mean) by averaging the corresponding items in the non-homogeneous cluster. If the cluster contains items from c classes, EHC computes c means. Then, it applies k-means clustering on the cluster, using the class-means as initial means, and builds c clusters. This clustering procedure is recursively applied on the items of each non-homogeneous cluster built. One-item clusters are removed. The items assigned to non-one-item clusters constitute the edited set. Algorithmically, EHC differs from RHC in the following point: EHC stores in the edited set all training items that have not been assigned to one-item clusters, while RHC computes the mean item for each homogeneous cluster and stores it in the condensing set.

Two examples that demonstrate the operation of EHC are depicted in Figures 3.5 and 3.6. More specifically, Figure 3.5 demonstrates how EHC identifies and removes a close-border item, while Figure 3.6 demonstrates how the algorithm removes an item that is noise. Note that non-homogeneous clusters are depicted with dashed borders. EHC identifies and removes outliers in a similar way. In particular, Figure 3.5(a) presents a dataset with a border item that should be removed. Initially, EHC computes the class-means by averaging the items that belong to each class (Figure 3.5(b)). Then, k-means is executed using the class-means as initial means and identifies two clusters (Figure 3.5(c)): cluster A is non-homogeneous while cluster B is homogeneous. Since cluster B is homogeneous and has more than one item, it is ignored. Then, the class-means in cluster A are computed (Figure 3.5(d)) and k-means is executed on the data of the particular cluster. The result is the construction of two homogeneous clusters (Figure 3.5(e)). Since cluster D is a one-class cluster, it is removed (Figure 3.5(f)). In an analogous way, EHC removes the item that represents noise in Figure 3.6(a). EHC identifies and removes outliers in a similar way.

Algorithm 11 describes a possible implementation of EHC. It utilizes a queue data structure Q in order to hold the unprocessed clusters. Initially, the edited set (ES) is set to be the whole training set (TS) (line 1) and Q holds the whole TS as one unprocessed cluster (lines 2–3). In each algorithm iteration, cluster C is taken from the head of Q and is examined (line 5). If C is homogeneous (line 6), the algorithm counts the items in C and if C is a one-item cluster, its item is removed from ES (lines 7–9). If C is a non-homogeneous cluster, the class-means for all the classes present in it are computed and added to set R (lines 11–14). Set R and cluster C are the input parameters to k-means clustering (line 15). The returned clusters are enqueued



(a) initial data

(b) means of classes

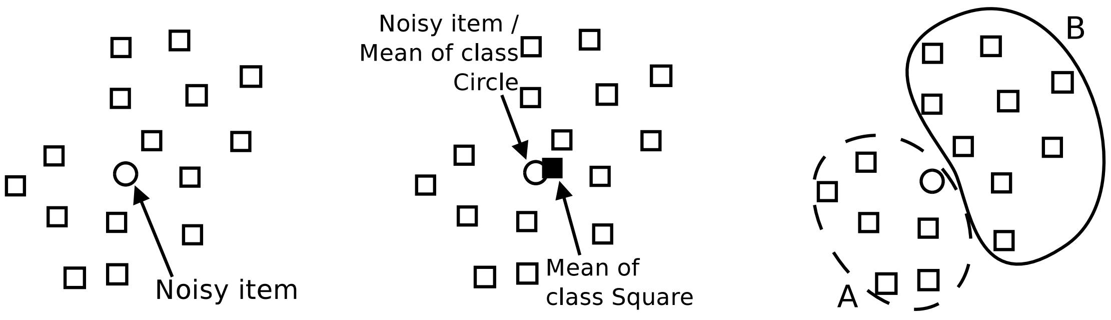
(c) k-means on initial data

(d) means of classes in cluster A

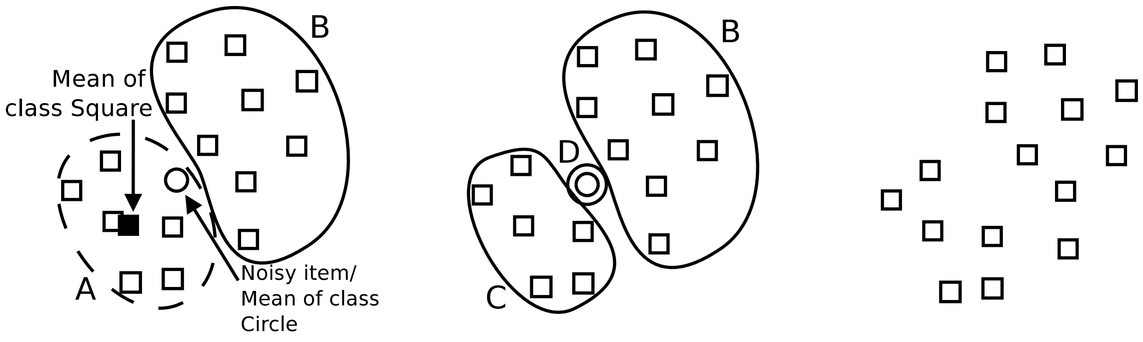
(e) k-means on non-homogeneous cluster A

(f) final edited set

Figure 3.5: EHC: Smoothing decision boundaries



(a) initial data (b) means of classes (c) k-means on initial data



(d) means of classes in cluster A (e) k-means on non-homogeneous cluster A (f) final edited set

Figure 3.6: EHC: Removing noise

in Q (lines 16–18). The loop continues as long as there are non-homogeneous clusters (line 20).

EHC may assign a typical data item (that is not noise or a close-border item) to an one-item cluster and remove it. For instance, suppose that a non-homogeneous cluster with two items is built. EHC will remove both items even when one of them belongs to the major class of the region. Concerning the computational cost, we can easily conclude that EHC is a fast preprocessing algorithm. It uses the fast k-means clustering algorithm that is also sped-up by considering as initial means the means of the classes that are present in each cluster. One expects that the resulting clusters are quickly consolidated and the cost is lower than when opting for random means initialization. It is worth mentioning that contrary to all other editing methods, EHC does not compute distances between "real" items. It computes distances between items and mean items. Moreover, contrary to ENN-rule and some of its variations

Algorithm 11 EHC

```
Input: $TS$
Output: $ES$
 1: $ES \leftarrow TS$
 2: $Q \leftarrow \varnothing$
 3: Enqueue($Q$, $TS$)
 4: repeat
 5:     $C \leftarrow$ Dequeue($Q$)
 6:     if $C$ is homogeneous then
 7:         if $|C| = 1$ then
 8:             $ES \leftarrow ES - C$
 9:         end if
10:     else
11:         $R \leftarrow \varnothing$ {R is the set of class-means}
12:         for each class $M$ in $C$ do
13:             $R \leftarrow R \cup mean\_of(M)$
14:         end for
15:         $Clusters \leftarrow K$-MEANS($C$, $R$)
16:         for each cluster $Cl \in Clusters$ do
17:             Enqueue($Q$, $Cl$)
18:         end for
19:     end if
20: until IsEmpty($Q$) {until all constructed clusters are homogeneous}
21: return $ES$
```

that compute a fixed number of distances regardless the item distribution in the multidimensional space, the number of distances computed by EHC is difficult to predict in advance. It exclusively depends on the item distribution in the data space. The main advantages of EHC are that, contrary to all other editing approaches, it is very fast and to the best of our knowledge is the only non-parametric editing algorithm. Hence, costly and time-consuming trial-end-error procedures for parameter tuning are avoided. Finally, EHC builds the same edited set regardless of data ordering.

3.3.3 Performance evaluation

Experimental setup

The proposed EHC algorithm was coded in C and evaluated on ten datasets. We downloaded eight datasets from the KEEL dataset repository[5] [6] and give theit profile in Table 3.12. Initially, we did not know the level of noise in each dataset. After our experimentation, we realized that the LIR dataset is an almost noise-free dataset and the LS and PH datasets have low levels of noise. Since, we wanted to test how editing behaves on noise-free datasets, we decided to include these datasets in our experimentation. Moreover, we built two additional datasets by adding 10% random noise in LS and PH. We refer to these datasets as LS-n and PH-n respectively. The noise was added by setting the class label of the 10% of the training items to a randomly chosen different class label. No other data transformation was performed. Finally, the Euclidean distance was adopted as the distance metric.

Table 3.12: Datasets description

Dataset	Size (items)	Attributes	Classes
Magic Gamma Telescope (MGT)	19020	10	2
Landsat Satellite (LS)	6435	36	6
Phoneme (PH)	5404	5	2
Letter Image Recognition (LIR)	20000	16	26
Banana (BN)	5300	2	2
Ecoli (ECL)	336	7	8
Pima (PM)	768	8	2
Yeast (YS)	1484	8	10

For comparison purposes, we coded the three state-of-the-art algorithms presented in detail in Section 2.1.2 (ENN-rule, All-kNN and Multiedit). We coded and used an implementation of multiedit that may re-compute the same distance more than once. An implementation that does not re-compute distances, would have to store already computed distances since they should be available until the end of execution. This may lead to an inefficient implementation with high memory requirements. We note that EHC and the aforementioned algorithms are available on WebDR[6] (see Appendix A).

[5]`http://sci2s.ugr.es/keel/datasets.php`

[6]`https://ilust.uom.gr/webdr`

An important issue that we had to address was the tuning of the parameters of the aforementioned algorithms. For all of them, we adopted the settings proposed in [49]. In particular, we used $k = 3$ for ENN-rule, $k = 7$ and $k = 9$ for All-kNN and $n = 3$ and $R = 2$ for multiedit. These settings are very common in many experimental studies in the literature. In addition, we used $k = 5$ for ENN-rule and $n = 5$ for multiedit. Finally, we also measured and present the performance of the conventional 1-NN classifier (classification without editing).

The four editing algorithms were compared to each other in terms of two main criteria: classification accuracy and preprocessing (editing) cost. The latter was estimated by counting the distances computed by each algorithm. Accuracy measurements were estimated by executing the 1-NN classifier on the edited sets. For each algorithm and dataset, we report the average accuracy and cost measurements obtained via a five-fold cross-validation. We used the pairs of training/testing sets distributed by the KEEL repository. Although the reduction rates achieved by each method do not indicate the best performing algorithm, they reveal the percentage of data that is considered as noise by each algorithm. Therefore reduction rates are also reported.

Experimental measurements

The performance measurements of our experimental study are presented in Table 3.13. Each table cell contains three measurements that correspond to the execution of an editing approach on a particular dataset. The three measurements are: accuracy, reduction rate and preprocessing cost in millions of distance computations. The best measurements are in bold.

As we expected, EHC is the fastest approach. It achieves very low average preprocessing cost compared to the other algorithms (see the last row of the table). EHC computes the fewest distances in nine out of ten datasets. Furthermore, we observe that the cost gains are very high for large datasets. Finally, as we predicted in Section 3.3.2, EHC computes a completely different number of distances for the LS, LS-n datasets and the PH, PH-n datasets.

Concerning accuracy measurements, we observe that the proposed algorithm is comparable to ENN-rule and All-kNN. Multiedit has the worst accuracy, especially for the LIR and ECL datasets, where the accuracy measurement is unacceptable. This happens because multiedit removes data that should not be removed. Although the differences in accuracy between EHC, ENN and All-kNN are not statistically significant, we observe that EHC has the highest accu-

Table 3.13: Experimental results: Accuracy (Acc (%)), Reduction Rate (RR (%)) and Preprocessing Cost (PC (millions of distance computations))

Dataset		**1-NN**	**ENN** ($k = 3$)	**ENN** $k = 5$)	**Multiedit** ($n = 3$, $R = 2$)	**Multiedit** ($n = 5$, $R = 2$)	**All$k$NN** ($k = 7$)	**All$k$NN** ($k = 9$)	**EHC**
MGT	Acc	78.14	80.44	80.57	76.75	75.26	80.76	**80.86**	79.52
	RR	-	20.08	19.20	39.98	42.36	29.67	30.38	10.70
	PC	-	115.76	115.76	2,839.55	1,447.93	115.76	115.76	**4.08**
LS	Acc	**90.60**	90.30	90.43	86.79	86.03	90.12	90.16	**90.55**
	RR	-	9.07	9.27	24.13	26.17	13.92	14.51	3.11
	PC	-	13.25	13.25	266.22	139.53	13.25	13.25	**1.69**
PH	Acc	**90.10**	88.14	87.53	80.77	79.72	86.55	86.23	**89.06**
	RR	-	11.25	11.93	34.14	36.91	17.92	19.30	7.36
	PC	-	9.35	9.35	166.22	53.71	9.35	9.35	**0.66**
LIR	Acc	**95.83**	94.98	94.87	70.94	58.35	94.28	94.00	**95.23**
	RR	-	4.33	4.44	43.43	56.59	7.31	7.97	3.95
	PC	-	127.99	127.99	7,214.38	2,900.53	127.99	127.99	**41.85**
BN	Acc	86.91	89.36	89.55	89.83	**90.38**	89.509	89.79	88.60
	RR	-	11.53	10.98	20.12	21.64	17.10	17.51	10.65
	PC	-	8.99	8.99	106.69	60.26	8.99	8.99	**0.56**
ECL	Acc	79.78	81.57	81.86	63.10	46.11	81.26	80.66	**82.16**
	RR	-	20.45	20.45	47.29	60.15	28.63	30.48	17.01
	PC	-	0.036	0.036	0.100	0.055	0.036	0.036	**0.035**
PM	Acc	68.36	71.87	71.75	71.36	68.89	72.65	**73.30**	70.32
	RR	-	30.16	29.43	53.07	58.96	45.56	46.24	16.59
	PC	-	0.19	0.19	0.51	0.26	0.19	0.19	**0.06**
YS	Acc	52.16	56.47	57.07	52.90	50.54	58.29	**58.42**	54.45
	RR	-	45.73	43.89	74.34	80.93	59.90	61.25	29.58
	PC	-	0.70	0.70	1.19	**0.58**	0.70	0.70	0.84
LS-n	Acc	82.58	89.64	89.74	86.47	85.55	89.73	**89.84**	87.55
	RR	-	19.82	18.45	38.33	40.19	29.64	30.17	10.93
	PC	-	13.25	13.25	139.02	78.43	13.25	13.25	**2.00**
PH-n	Acc	82.14	**86.94**	86.70	81.31	79.29	86.31	85.90	86.16
	RR	-	21.20	20.61	44.93	49.85	33.29	34.68	17.66
	PC	-	9.35	9.35	52.65	31.74	9.35	9.35	**0.71**
AVG	Acc	80.66	82.97	**83.01**	76.02	72.01	82.95	82.92	82.36
	RR	-	19.36	18.87	41.98	47.38	28.29	29.25	12.75
	PC	-	29.89	29.89	1,078.65	471.30	29.89	29.89	**5.25**

racy in four out of ten datasets. However, ENN-rule has the highest average accuracy (see last row in Table 3.13).

For the LIR, LS and PH datasets that contain low levels of noise, all editing approaches seem to negatively affect accuracy since the conventional 1-NN classifier is the most accurate approach. However, in all these cases, EHC is the most accurate editing algorithm. In contrast, in the rest seven datasets, most of the editing approaches achieve higher accuracy than the conventional 1-NN classifier. Therefore, it appears that editing constitutes a necessary preprocessing step.

The proposed algorithm has the lowest reduction rate. EHC removes items by using the strict criterion of one-item clusters. For datasets with extremely high levels of noise (e.g. 30% or more), it is not certain that EHC will improve classification accuracy like ENN-rule with an appropriate k value does. On the other hand, EHC is not expected to negatively affect classification accuracy as much as the other methods do.

Non-parametric statistical test

We validated the performance measurements previously presented by applying the Wilcoxon signed ranks test [32]. In effect, we executed the particular test three times: once on the ten measurements of each criterion (classification accuracy (ACC), preprocessing cost (PC)), and once on the ten measurements of the overall classification performance criterion. Similar to [48, 47], the overall performance is the average of the normalized measurements of accuracy and preprocessing cost. Therefore, first, we normalized the measurements of each criterion to the range [0,1]. The overall performance considered accuracy and preprocessing cost as having the same significance. Since low preprocessing cost is desirable, we used $1 - normalized(PC)$. Note that since the reduction rate does not indicate if an algorithm performs better than another, we did not run the test for the particular criterion.

The results of the test are shown in Table 3.14. The columns labelled with "w/l" count the number of wins and loses respectively. The columns with label "Wilc." present the Wilcoxon significance level. If for a pair of algorithms this value is lower than 0.05, one can claim that there is statistically significant difference between the two algorithms.

Considering the table, we can safely conclude that EHC is statistically better than the other algorithms in terms of preprocessing cost. Moreover, it is obvious that EHC is statistically better than multiedit in terms of classification accuracy. On the other hand, the differences

Table 3.14: Results of the Wilcoxon signed ranks test

Methods	ACC		PC		**Overall**	
	w/l	**Wilc.**	**w/l**	**Wilc.**	**w/l**	**Wilc.**
EHC vs ENN (k=3)	4/6	0.126	9/1	**0.013**	4/6	0.333
EHC vs ENN (k=5)	4/6	0.169	9/1	**0.013**	4/6	0.333
EHC vs Multiedit (n=3, R=2)	8/2	**0.017**	10/0	**0.005**	8/2	**0.013**
EHC vs Multiedit (n=5, R=2)	9/1	**0.009**	9/1	**0.013**	9/1	**0.007**
EHC vs All-k-NN (k=7)	4/6	0.386	9/1	**0.013**	4/6	0.646
EHC vs All-k-NN (k=9)	5/5	0.508	9/1	**0.013**	5/5	0.575

between accuracy measurements of all other pairs are not statistically significant. Concerning the overall classification performance, EHC performs better than Multedit (the difference between the particular algorithms is statistically significant).

3.3.4 Conclusions

Classification accuracy achieved by the k-NN classifier strongly depends on the quality of the available training data. Noise and mislabelled data as well as outliers and overlaps between regions of different classes are the reasons of bad classification performance for the particular classifier. Editing algorithms can improve classification accuracy by removing such data. In this section, we proposed a non-parametric algorithm, called Editing through Homogeneous Clusters (EHC), which follows a completely different strategy than the other editing approaches. EHC is based on a clustering procedure that forms homogeneous clusters in the training data. The clusters that contain only one item are considered irrelevant (they represent noise, outlier or close-border items) and are removed. An experimental study with ten datasets showed that the proposed algorithm is very fast and achieves comparable classification accuracy to the state-of-the-art editing algorithms.

3.4 Simultaneous editing and data abstraction by finding homogeneous clusters

3.4.1 Motivation and contribution

As already mentioned, the application of an editing algorithm is an extra and usually costly preprocessing step. In Sections 3.2 and 3.3, we presented RHC and EHC, respectively. RHC is a prototype abstraction algorithm while EHC is an editing algorithm. Both are non-parametric, very fast and based on a similar k-means clustering procedure that finds homogeneous clusters in the training data. The motivation here is to ascertain if the aforementioned algorithms can be effectively combined in a new prototype abstraction algorithm. The contribution is the proposal and the evaluation of an effective fast, non-parametric prototype abstraction algorithm that integrates the idea of editing. It is called Editing and Reduction though Homogeneous Clusters (ERHC) and is a descendant of our RHC and EHC algorithms.

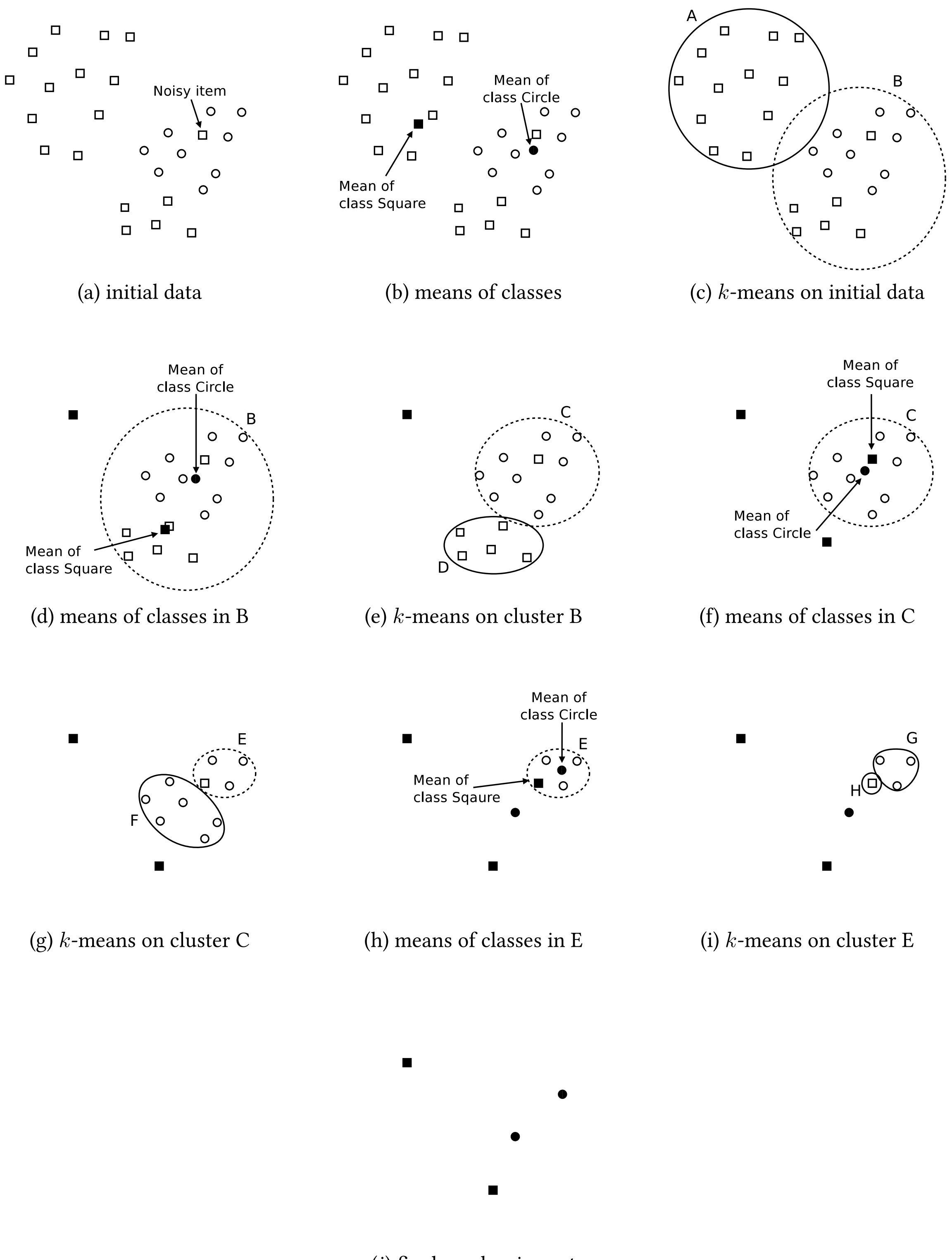
The rest of the section is organized as follows. Subsection 3.4.2 presents the proposed ERHC algorithm. Experimental measurements and the results of the Wilcoxon signed ranks test are presented in Subsection 3.4.3. Finally, Subsection 3.4.4 concludes the presentaion of ERHC.

3.4.2 The Editing and Reduction through Homogeneous Clusters (ERHC) algorithm

ERHC constitutes a combination of RHC and EHC. Practically, it is a RHC variation that can effectively manage datasets with noise. ERHC works in a way similarly to RHC and EHC. More specifically, it works as follows. Initially, the whole training set is considered to be a non-homogeneous cluster. ERHC begins by computing the class-mean for each class by averaging the corresponding items of the cluster. Therefore, for a dataset with n classes, ERHC estimates n class-means. ERHC continues by executing k-means clustering adopting the n class-means as initial means. The result is the construction of n clusters. If a homogeneous cluster is identified and contains a single item, ERHC removes it. Otherwise, its cluster-mean is a prototype and is placed in the condensing set. The above clustering procedure is applied recursively on the items of each non-homogeneous cluster. ERHC terminates when all clusters become homogeneous. The final condensing set built by ERHC contains the cluster-means of the homogeneous clusters that contain more than one item. In effect, RHC, EHC and ERHC, differ to each other on how they treat the homogeneous clusters. Of course, using the class-means as initial means for k-means clustering, the number of clusters built is automatically determined.

The aforementioned procedure is depicted in the example presented in Figure 3.7. Suppose that the initial training set contains two classes, squares and circles (Figure 3.7(a)). ERHC computes two class-means (Figure 3.7(b)). k-means is applied on the training set and builds two clusters, A and B (Figure 3.7(c)). A is homogeneous and contains more than one item. Therefore, its cluster-mean is placed in the condensing set. B is non-homogeneous since it contains items from both classes. Therefore, ERHC computes two class-means (Figure 3.7(d)), and then k-means is applied on B and builds clusters C and D (Figure 3.7(e)). Since D is homogeneous and contains more than one item, its cluster-mean is placed in the condensing set. Since C is non-homogeneous, its class-means are computed (Figure 3.7(f)) and k-means is applied on C building clusters E and F (Figure 3.7(g)). Then, the cluster-mean of the non-one-item-cluster homogeneous cluster F is placed in the condensing set, while the class-means are computed for the non-homogeneous cluster E (Figure 3.7(e)). k-means is applied on E and the result is the clusters G and H. Both are homogeneous (Figure 3.7(i)). However, H is one-item-cluster. Hence, it is removed (it is not represented in the condensing set). The cluster-mean of G is placed in the condensing set (Figure 3.7(j)). The final condensing set contains only four items (reduction rate over 85%). Note that the only difference between RHC and ERHC is that RHC will place in the condensing set a prototype for cluster H. However, this affects the quality of the final condensing set.

ERHC is easy to implement. Algorithm 12 is the pseudo-code of a possible implementation. Like RHC and EHC, It uses a queue structure $Queue$ to store clusters. Initially, $Queue$ holds the whole training set (TS) as an unprocessed cluster (lines 1–2). In each iteration, the head cluster C is dequeued from $Queue$ (line 5). If C is homogeneous and not a one-item-cluster (lines 6–7), its mean is placed in the condensing set (CS) (lines 8–9). If C is non-homogeneous (line 11), a class-mean for each class in C is computed and added to set M (lines 12–16). The latter as well as C is the input to a k-means clustering call (line 17). The resulting clusters $NewClusters$ are enqueued in $Queue$ (lines 17–20). The repeat-until loop (lines 4, 22) terminates when $Queue$ is empty, i.e., all clusters become homogeneous. We should mention that Algorithm 12 differs from RHC presented in Algorithm 9 in one point: the former includes an extra "if" statement regarding the one-item-clusters (line 7).



(a) initial data

(b) means of classes

(c) k-means on initial data

(d) means of classes in B

(e) k-means on cluster B

(f) means of classes in C

(g) k-means on cluster C

(h) means of classes in E

(i) k-means on cluster E

(j) final condensing set

Figure 3.7: ERHC: data abstraction and editing process

Algorithm 12 ERHC

```
Input: $TS$
Output: $CS$
 1: $Queue \leftarrow \varnothing$
 2: Enqueue($Queue$, $TS$)
 3: $CS \leftarrow \varnothing$
 4: repeat
 5:    $C \leftarrow$ Dequeue($Queue$)
 6:    if $C$ is homogeneous then
 7:       if $|C| > 1$ then
 8:          $r \leftarrow$ mean of $C$
 9:          $CS \leftarrow CS \cup \{r\}$
10:       end if
11:    else
12:       $M \leftarrow \varnothing$ {M is the set of class-means}
13:       for each class $L$ in $C$ do
14:          $m_L \leftarrow$ mean of $L$
15:          $M \leftarrow M \cup \{m_L\}$
16:       end for
17:       $NewClusters \leftarrow K$-MEANS($C$, $M$)
18:       for each cluster $C \in NewClusters$ do
19:          Enqueue($Queue$, $C$)
20:       end for
21:    end if
22: until IsEmpty($Queue$)
23: return $CS$
```

Obviously, ERHC is quite similar to RHC. However, we expect that the simple editing mechanism integrated in ERHC can effectively improve classification performance especially when data contains noise. ERHC inherits all the benefits of the algorithm for finding homogeneous clusters. Therefore, it is very fast since it is based on k-means that is accelerated by the class-mean initializations. We used the full cluster consolidation for the stopping condition of k-means clustering. ERHC could have been even be faster had we used another stopping condition. Moreover, ERHC does not depend on the data order in the training set. We note that ERHC is not equivalent to the successive execution of EHC and RHC algorithms (let's call this EHC-RHC). Different clusters are built by the two approaches and consequently different reduction rates are achieved. Also, EHC-RHC has higher preprocessing cost than ERHC since it applies the procedure for finding homogeneous clusters twice: once on the non-edited and once on the edited data. ERHC can simultaneously remove noise from the data and reduce the size of the training set.

3.4.3 Performance evaluation

Experimental setup

ERHC algorithm was coded in C and tested on eleven datasets. Like all other DRTs implemented during the PhD research, ERHC is available on WebDR[7] (see Appendix A). We used nine datasets from the KEEL dataset repository[8] [6]. Six of them were also used in the experimental study presented in Section 3.3. Table 3.15 summarizes the datasets used in this stage of our experimentations. Our study revealed that the LIR, PD, SH and TXR datasets are noise-free. All other datasets contain various levels of noise. In addition, the LS and PH datasets have low levels of noise. Like in the experimental study presented in Subsection 3.3.3, we built two extra versions of these datasets by adding 10% noise. We refer to them as LS-n and PH-n respectively. Practically, 10% noise were added by selecting a different class label for 10% of the training items.

Table 3.15: Datasets description

Dataset	Size (items)	Attributes	Classes
Letter Image Recognition (LIR)	20000	16	26
Pen-Digits (PD)	10992	16	10
Shuttle (SH)	58000	9	7
Texture (TXR)	5500	40	11
Banana (BN)	5300	2	2
Landsat Satellite (LS)	6435	36	6
Magic Gamma Telescope (MGT)	19020	10	2
Phoneme (PH)	5404	5	2
Pima (PM)	768	8	2

In Section 3.2, we compared RHC to CNN-rule, IB2, RSP3, PSC on the non-edited and edited forms of several datasets. Consequently, we do not include experiments for these algorithms. Here, ERHC is compared to RHC, ENN-RHC and EHC-RHC algorithms. ENN-RHC is the

[7]https://ilust.uom.gr/webdr
[8]http://sci2s.ugr.es/keel/datasets.php

successive execution of ENN-rule for editing and RHC for data abstraction, whereas EHC-RHC is the successive execution of EHC for editing and RHC for data abstraction.

We used ENN-rule because it is the reference editing algorithm (see Section 2.1.2) and, as the experimentations in Subsection 3.3.3 showed, it achieves good classification performance. All other editing algorithms are either variations of ENN-rule or are based on the same idea. Of' course, the main disadvantage of ENN-rule is that the user should define the value of k that defines the size of the examined neighbourhood. Here, we ran experiments with $k = 3$. That value is either adopted or suggested by many researchers [131, 49, 84].

The four algorithms were evaluated by estimating three measurements: accuracy, reduction rate, and, preprocessing cost in terms of distance computations. Accuracy was estimated by running the 1-NN classifier over the condensing set built by each algorithm. We report the average values of the measurements obtained via five-fold cross-validation. We used the five pairs of training/testing sets distributed by the KEEL repository. The distances between items were estimated using the Euclidean distance.

Experimental measurements

Table 3.16 presents the performance measurements obtained with the best ones shown in bold font. Preprocessing cost measurements are in million distances. In addition, Table 3.16 reports the accuracies obtained by applying the conventional 1-NN classifier (Conv-1-NN) on the non-edited training data (without data reduction).

For the BN, PM, LS-n and PH-n datasets, one or more algorithms achieved higher accuracy than Conv-1-NN. It is clear that ERHC performs better than RHC (with respect to all comparison criteria). Thus, we conclude that ERHC effectively extends RHC. It is worth mentioning that no editing approach (ERHC included) negatively affects accuracy of noise-free datasets (LIR, PD, SH, TXR).

We expected EHC-RHC to have higher reduction rates than ERHC. However, the results show that this is not always true. In five datasets (LIR, SH, TXR, LS, MGT), ERHC has higher reduction rate than EHC-RHC. Nevertheless, the differences are almost insignificant.

Concerning the preprocessing cost, we observe that, in all cases, EHC-RHC has almost the double cost compared to ERHC and RHC. Also, ENN-RHC is the slowest approach. This is because it involves the execution of the costly ENN-rule procedure. In practice, the actual

Table 3.16: Experimental results: Accuracy (Acc (%)), Reduction Rate (RR (%)) and Preprocessing Cost (PC (millions of distance computations))

Dataset		**Conv-1-NN**	**RHC**	**ENN-RHC**	**EHC-RHC**	**ERHC**
LIR	Acc	95.825	**93.585**	92.720	93.045	92.690
	RR	-	88.081	90.343	90.383	**92.029**
	PC	-	**41.844**	159.039	73.710	**41.844**
PD	Acc	99.354	98.299	98.453	98.472	**98.626**
	RR	-	96.516	97.189	**97.589**	97.448
	PC	-	**2.882**	41.489	5.521	**2.882**
SH	Acc	99.822	98.095	**99.597**	98.481	98.038
	RR	-	99.550	99.658	99.669	**99.690**
	PC		**16.827**	1098.864	32.695	**16.827**
TXR	Acc	99.018	97.036	97.109	96.873	**97.364**
	RR	-	94.705	95.582	95.732	**95.936**
	PC	-	**3.629**	12.675	6.133	**3.629**
BN	Acc	86.906	83.283	**88.094**	87.019	88.000
	RR	-	79.684	**95.660**	93.000	90.330
	PC	-	**0.562**	9.519	1.014	**0.562**
LS	Acc	90.598	88.951	**89.138**	88.392	89.013
	RR	-	89.841	**95.062**	92.273	92.949
	PC	-	**1.693**	14.984	3.192	**1.693**
MGT	Acc	78.144	71.966	**77.781**	74.716	77.014
	RR	-	73.757	**93.057**	83.843	84.456
	PC	-	**4.082**	118.591	7.480	**4.082**
PH	Acc	90.100	85.585	85.400	86.158	**86.565**
	RR	-	80.708	**92.098**	89.008	88.053
	PC	-	**0.658**	9.812	1.161	**0.658**
PM	Acc	68.358	63.281	**72.653**	69.927	69.793
	RR	-	63.577	**91.792**	80.977	80.065
	PC	-	**0.062**	0.219	0.103	**0.062**
LS-n	Acc	82.580	78.819	**88.578**	84.817	85.377
	RR	-	76.632	**95.361**	88.465	87.560
	PC	-	**1.999**	14.744	3.637	**1.999**
PH-n	Acc	82.143	75.407	83.993	81.255	**84.030**
	RR	-	64.246	**92.019**	86.394	81.910
	PC	-	**0.706**	116.164	1.180	**0.706**
Avg	Acc	88.441	84.937	**88.501**	87.196	87.865
	RR	-	82.482	**94.347**	90.667	90.039
	PC	-	**6.813**	145.100	12.348	**6.813**

preprocessing cost of ENN-rule may even be higher: tuning its parameter may require multiple executions of a trial-and-error-procedure.

Although ENN-RHC is the slowest approach, it seems to be slightly more accurate than EHC-RHC and ERHC. Moreover, it achieves higher reduction rate measurements on datasets with high levels of noise. This is because ENN-rule considers as noise more items than EHC and ERHC do.

Non-parametric statistical test

Like the experimental studies presented in Subsections 3.2.4 and 3.3.3, we validated the performance measurements by applying the Wilcoxon signed ranks test [32]. We ran the test four times, i.e., once for each comparison criterion (classification accuracy (ACC), reduction rate (RR), preprocessing cost (PC)) and once on the measurements of the overall classification performance criterion. Here, the overall classification performance takes into account the reduction rates. Therefore, the overall performance is computed by averaging the normalized measurements (to the range $[0, 1]$) of the three criteria (we used $1 - normalized(PC)$). Thus, all criteria are considered as having the same significance.

Table 3.17 shows the results of the test. The columns labelled with "w/l/t" counts the number of wins, losses and ties between the corresponding algorithms while the columns labelled with "Wilc." list the Wilcoxon significance level. As we expected, the difference between ERHC and RHC (Wilcoxon value < 0.05) is statistically significant in terms of accuracy, reduction rate and overall classification performance. RHC has only two wins against ERHC (slightly higher accuracy measurements for the LIR and SH datasets). Of course, ERHC is statistically better than ENN-RHC and EHC-RHC in terms of preprocessing cost. There are not other significant differences between ERHC and the other algorithms. On the other hand, we observe that EHC-RHC is statistically better than RHC in terms of all criteria and than ENN-RHC in terms of preprocessing cost. EHC-RHC is statistically worse than ENN-RHC in terms of accuracy and reduction rate.

3.4.4 Conclusions

In this section, we proposed ERHC, an improved variation of the RHC algorithm that can handle datasets with noise. It can achieve high reduction rates regardless the level of noise in the training data and without requiring as high preprocessing cost as a sequential execution of

Table 3.17: Results of the Wilcoxon signed ranks test

Methods	**ACC**		**RR**		**PC**		**Overall**	
	w/l/t	**Wilc.**	**w/l/t**	**Wilc.**	**w/l/t**	**Wilc.**	**w/l/t**	**Wilc.**
ERHC vs RHC	9/2/0	**0.016**	11/0/0	**0.003**	0/0/11	1	11/0/0	**0.003**
ERHC vs ENN-RHC	4/7/0	0.286	4/7/0	**0.041**	11/0/0	**0.003**	4/7/0	0.248
ERHC vs EHC-RHC	8/3/0	**0.033**	5/6/0	0.328	11/0/0	**0.003**	6/5/0	0.790
EHC-RHC vs RHC	8/3/0	**0.041**	11/0/0	**0.003**	0/11/0	**0.003**	10/1/0	**0,004**
EHC-RHC vs ENN-RHC	3/8/0	**0.033**	4/7/0	**0.041**	11/0/0	**0.003**	4/7/0	0.213

EHC and RHC. The algorithm is based on a k-means clustering procedure that forms homogeneous clusters. The clusters that contain only one item are considered to be noise and are removed. The mean items of the other homogeneous clusters are placed in the condensing set as prototypes. ERHC inherits all the properties of RHC. Therefore, it is a fast non-parametric prototype abstraction algorithm and its performance does not depend on the order of training data. ERHC was empirically evaluated on eleven datasets. The experimental measurements illustrated that ERHC is very fast and achieves high reduction rates and classification accuracy. In all cases, the performance measurements were better than RHC.

Chapter 4

Other proposed data reduction techniques

4.1 Introduction

The research objective of this chapter is also the issue of data reduction for improved k-NN classification. It contributes two additional prototype abstraction algorithms. In Section 4.2, a prototype abstraction version of the well-known IB2 algorithm is proposed. It is called AIB2 [96, 88]. IB2 is an effective prototype selection algorithm (see Subsection 2.1.3). Contrary to many other Data Reduction Techniques (DRTs) and like dRHC (see Subsection 3.2.3), IB2 is a very fast, one-pass condensing algorithm that builds its condensing set in an incremental (dynamic) manner. AIB2 maintains this property of IB2. However, it generates new prototypes instead of selecting them. AIB2 attempts to improve the efficiency of IB2 by positioning the prototypes in the center of the data areas they represent. The empirical experimental study and the Wilcoxon signed ranks test show that AIB2 performs better than IB2.

Section 4.3 proposes a simple, noise-tolerant prototype abstraction algorithm called RkM [95]. RkM is based on the popular k-means clustering algorithm. The conducted experimental study shows that if the condensing set contains only the means of clusters as prototypes, classification performance is not negatively affected as much by the addition of noise in the training data.

4.2 Efficient data abstraction using weighted IB2 prototypes

4.2.1 Motivation and contribution

As already mentioned, IB2 belongs to the popular family of Instance-Based Learning (IBL) algorithms. IB2 is an one-pass incremental[1] condensing algorithm. Actually, it is based on the well known CNN-rule. Contrary to CNN-rule and many other state-of-the-art condensing and prototype abstraction algorithms, IB2 can dynamically build its condensing set. In other words it is an incremental algorithm. It can take into consideration new training items after the construction of the condensing set and without needing the old removed training items. Hence, it can be used in dynamic/streaming environments [1] where new training data is gradually available. In addition, the incremental nature of IB2 allows its execution on training sets that cannot fit into main memory.

In this section, we attempt to improve the performance of IB2 by considering the idea of data abstraction. Our contribution is the development and evaluation of a prototype abstraction version of IB2 that we call Abstraction IB2 (AIB2). The proposed prototype abstraction algorithm retains all the properties of IB2, but it is faster and achieves higher reduction rates and improved classification accuracy.

The rest of the section is organized as follows: Subsection 4.2.2 describes in detail the proposed AIB2 algorithm. Subsection 4.2.3 presents an experimental study where the k-NN classifier is applied on the original training sets and the corresponding condensing sets built by CNN-rule IB2 and AIB2, of nine datasets. The experimental study is complemented by the statistical comparison of the three algorithms through the Wilcoxon signed ranks test. Subsection 4.2.4 concludes the section.

4.2.2 The Abstraction IB2 (AIB2) algorithm

The proposed AIB2 algorithm adopts the idea of IB2 and CNN-rule: items that lie in the "internal" data area of a class are useless and can be removed without loss of accuracy. AIB2 is a prototype abstraction version of IB2. Therefore, AIB2 is a non-parametric, fast, one-pass algorithm. It is also appropriate for dynamic / streaming environments and can handle new class labels. Like IB2, it can be applied on very large datasets that cannot fit into main memory or on devices with limited main memory (e.g., sensor networks), without transferring data to

[1]The term "incremental" is identical in meaning to the term "dynamic" used in Section 3.2

a server over a network for processing. The latter is usually costly and time-consuming. Like IB2, AIB2 does not take into account the phenomenon of concept drift [125] that may exist in data streams.

The main difference between AIB2 and IB2 is the following. The items that are correctly classified by the 1-NN rule are not ignored. In effect, they contribute to the construction of the condensing set by repositioning their nearest prototype in the condensing set. The main idea behind AIB2 is that prototypes should be at the center of the data area they represent. To achieve this, AIB2 adopts the concept of prototype weight. Each prototype has a weight value as an extra attribute that denotes the number of training items it represents. The weight values are used for updating the prototype attributes in the multidimensional space.

Algorithm 13 presents how AIB2 works. Initially, the condensing set (CS) is populated by a random item of the training set (TS) whose weight is initialized to 1 (lines 2–3). For each item x of TS, AIB2 searches in CS and retrieves its nearest prototype NN (line 5). If x is misclassified, it is placed in CS and its weight is initialized to one (lines 6–8). If x is correctly classified, NN's attributes are updated by taking into consideration its current weight and the attributes of x. Effectively, NN "moves" towards x in the multidimensional space (lines 10–12). Finally, the weight of NN is increased by one (line 13) and x is removed (line 15). At the end, CS constitutes the final condensing set. Whenever a training item is available, it either enters CS or repositions the nearest prototype.

Like IB2 and CNN-rule, AIB2 considers that misclassified items are probably close to decision boundaries and so they must be placed in the condensing set. Although the correctly classified items are not placed in the condensing set, they are not ignored. They contribute by updating the nearest prototype without deteriorating reduction rates. The idea is that a condensing set built by AIB2 will contain better prototypes compared to the condensing set built by IB2 and will achieve higher classification accuracy. In addition, we expect that updating the prototypes will reduce the number of items that enter the condensing set, and thus, AIB2 will achieve higher reduction rates and lower preprocessing cost compared to IB2.

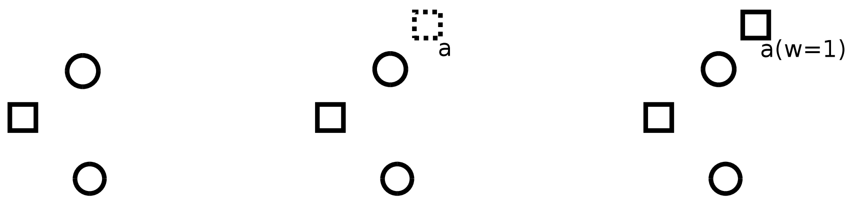
Figures 4.1 and 4.2 illustrate two-dimensional examples of AIB2 execution. Suppose that the current condensing set contains three prototypes, two of class circle and one of class square (Figure 4.1 (a) and Figure 4.2 (a)). Suppose that a new square item, a, arrives (Figure 4.1 (b)). AIB2 should decide whether a will enter the condensing set or it will be used to update an existing prototype. Since a is closer to a prototype of a different class, a enters the condensing set and its weight value is initialized to one (Figure 4.1 (c)). In contrast, suppose that the new

Algorithm 13 AIB2

```
Input: $TS$
Output: $CS$
1: $CS \leftarrow \varnothing$
2: pick an item $y$ of $TS$ and move it to $CS$
3: $y_{weight} \leftarrow 1$
4: for each $x \in TS$ do
5:     $NN \leftarrow$ Nearest Neighbour of $x$ in $CS$
6:     if $NN_{class} \neq x_{class}$ then
7:         $x_{weight} \leftarrow 1$
8:         $CS \leftarrow CS \cup \{x\}$
9:     else
10:        for each attribute $attr(i)$ do
11:            $NN_{attr(i)} \leftarrow \frac{NN_{attr(i)} \times NN_{weight} + x_{attr(i)}}{NN_{weight}+1}$
12:        end for
13:        $NN_{weight} \leftarrow NN_{weight} + 1$
14:    end if
15:    $TS \leftarrow TS - \{x\}$
16: end for
17: return $CS$
```

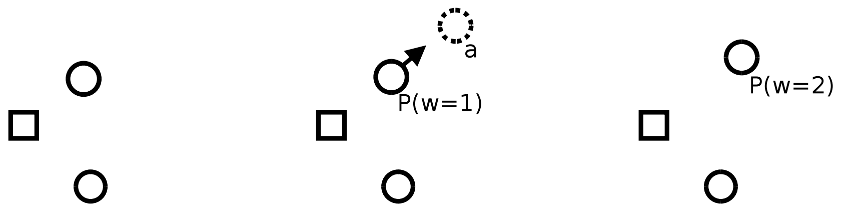
item a is a circle item (Figure 4.2 (b)). Since a is closer to P, which is a prototype of the same class, a updates P. Therefore P moves towards a and its weight is increased by one (Figure 4.2 (c)). In our example, the updated P lies in between the original P and a because the weight of P was one.

By updating the prototypes, AIB2 ensures that each prototype lies near the center of the data area it represents. Notice that the weight of a prototype practically denotes the population of the original items that it represents. Thus, although each time a new training item arrives and is assigned to an existing prototype, it causes the prototype to "move" towards the new item, the larger the weight of the prototype is, the smaller the "move" is towards the new training item. Like IB2, AIB2 prototypes depend on the order of the training items. It is possible that over time and because of a non-uniform arrival of training data items, an AIB2 prototype will gradually move far away from some of the original training items that it represents. Still, AIB2 prototypes with large weights will move very slowly towards the new training data items. This is also a problem with IB2 that contrary to CNN-rule cannot guarantee correct classification of all examined training data.



(a) Current condensing set (3 prototypes) (b) New item arrival (c) The new item enters the condnensing set

Figure 4.1: AIB2 example: new prototype enters the condensing set



(a) Current condensing set (3 prototypes) (b) New item arrival (c) Updating of nearest prototype

Figure 4.2: AIB2 example: repositioning an existing prototype

4.2.3 Performance evaluation

Experimental setup

AIB2, IB2 as well as CNN-rule were evaluated using eight datasets distributed by the KEEL dataset repository[2] [6] and summarized in Table 4.1. Seven datasets (all except the KDD dataset) were used without applying data normalization. The MGT, LS and TXR datasets are distributed sorted on the class label. We randomized the items of these datasets. The Euclidean distance was the distance metric used. Please note that all the aforementioned algorithms and datasets are available on WebDR[3] (see Appendix A).

[2] `http://sci2s.ugr.es/keel/datasets.php`
[3] `https://ilust.uom.gr/webdr`

Table 4.1: Datasets description

Dataset	Size	Attributes	Classes
Letter Image Recognition (LIR)	20000	16	26
Magic Gamma Telescope (MGT)	19020	10	2
Pen Digits (PD)	10992	16	10
Landsat Satellite (LS)	6435	36	6
Shuttle (SH)	58000	9	7
Texture (TXR)	5500	40	11
Phoneme (PH)	5404	5	2
KddCup (KDD)	141481	23	36

The KDD dataset contains 494,020 items and 41 attributes. However, huge amounts of data are duplicates. Like in case of the experimental study presented in Subsection 3.2.4, we removed all duplicates. Moreover, we removed the three nominal and the two fixed-value attributes that exist in the particular dataset, Consequently, the transformed KDD dataset contains 141,481 items and 36 attributes. In addition, the value ranges of KDD attributes vary extremely. Therefore, we normalized all attributes to the interval $[0, 1]$. Finally, we randomized the dataset.

We did not include more algorithms in our experimentation because our purpose was to focus on incremental and non-parametric algorithms. Of course, CNN-rule is non-incremental, but it is considered a reference condensing algorithm and is being used in many papers for comparison purposes. In addition, CNN-rule is the ancestor of IB2 and AIB2 and it makes sense to use it in our experiments.

The three algorithms were evaluated by estimating three measurements: (i) classification accuracy, (ii) reduction rate, and, (iii) preprocessing cost in terms of distance computations. Accuracy measurements were estimated by executing the k-NN classifier with $k = 1$ over the condensing set built by each algorithm. For each dataset and algorithm, we report the average values of these measurements obtained via five-fold cross-validation. With the exception of the KDD dataset, we used the five already constructed pairs of training and testing sets distributed by the KEEL repository. For the transformed KDD, we created the appropriate for cross-validation training/testing pairs. Of course, all measurements were estimated after the arrival of all training items.

The original (non-edited) form of the MGT dataset contains high level of noise. Noisy items are misleading for the three algorithms and negatively affect reduction rates and the corresponding accuracies. Hence, we tried to build a noise-free version of MGT. To achieve this, we ran an editing algorithm before the execution of the DRTs. In particular, we used the ENN-rule algorithm. ENN-rule needs to compute all distances among the training items, i.e., $\frac{N\times(N-1)}{2}$ distances, where N is the number of items. By applying ENN-rule (with $k=3$, see [131, 49, 84]) on each training portion of each fold, we built an extra dataset that we call MGT-ENN. Of course, the testing portions were not edited by the ENN-rule. We tested the three algorithms on both forms of the MGT dataset.

Experimental measurements

The results of our experimental study are presented in Table 4.2. Each column lists the results related to a classifier. Best measurements are in bold. Preprocessing cost measurements are in million distance computations. For reference, Table 4.2 presents the accuracy measurements obtained by the conventional 1-NN classifier on the original training set under the label Conv-1-NN.

We should clarify that, contrary to the case of the experimental study presented in Subsection 3.2.4 the reduction rates of the MGT-ENN dataset do not take into account the items removed by editing. They concern the size of the condensing set in relation to that of the edited set. Note that ENN-rule considered as noise 20.08% of MGT items on average. Therefore, the training portions of the MGT-ENN dataset contains 20.08% fewer items than the MGT dataset on average.

Although the three algorithms did not reach the accuracy levels of Conv-1-NN, they were very close to them. With the exception of the LIR, LS and TXR datasets, CNN-rule was more accurate than IB2 and AIB2. However, it required much higher preprocessing cost and it achieved lower reduction rates than IB2 and AIB2.

As we anticipated, the proposed algorithm performed better than IB2 in most datasets and in terms of all comparison criteria. The KDD dataset is the only dataset where IB2 seems to be better than AIB2. In the LIR, LS and TXR datasets, AIB2 was more accurate than CNN-rule. Although we expected even better performance, we believe that the improvements achieved by AIB2 are noteworthy.

Moreover, as we expected, ENN-rule improved the efficiency of the classification process on the MGT data. Accuracies and reduction rates achieved by the three algorithms on the

Table 4.2: Experimental results: Accuracy (Acc (%)), Reduction Rate (RR (%)) and Preprocessing Cost (PC (millions of distance computations))

Dataset		Conv-1-NN	CNN-rule	IB2	AIB2
LIR	Acc:	95.83	92.84	91.98	**94.12**
	RR:	-	83,54	85.66	**88.12**
	PC:	-	163.03	23.37	**20.10**
MGT	Acc:	78.14	**74.54**	71.97	73.36
	RR:	-	60.08	70.60	**71.90**
	PC:	-	281.49	34.61	**33.05**
MGT-ENN	Acc:	80.44	**79.26**	78.01	78.81
	RR:	-	87.62	90.07	**91.06**
	PC:	-	68.61	8.48	**7.65**
PD	Acc:	99.35	**98.68**	98.04	98.33
	RR:	-	95.36	96.23	**97.19**
	PC:	-	11.75	1.78	**1.38**
LS	Acc:	90.60	88.21	86.87	**89.42**
	RR:	-	80.22	84.62	**86.72**
	PC:	-	17.99	2.22	**1.92**
SH	Acc:	99.82	**99.76**	99.73	99.72
	RR:	-	99.37	99.44	**99.46**
	PC:	-	45.30	8.26	**7.89**
TXR	Acc:	99.02	97.16	96.35	**97.69**
	RR:	-	91.90	93.33	**94.95**
	PC:	-	5.65	0.84	**0.66**
PH	Acc:	90.10	**87.82**	85.57	84.92
	RR:	-	76.04	80.85	**81.75**
	PC:	-	13.45	1.96	**1.84**
KDD	Acc:	99.71	**99.66**	99.48	99.41
	RR:	-	99.12	**99.26**	99.21
	PC:	-	384.90	**55.58**	58.78
Average	Acc:	92.56	**90.88**	89.78	90.64
	RR:	-	85,92	88.90	**90.04**
	PC:	-	110.24	15.23	**14.81**

MGT-ENN dataset (edited data) are higher than the corresponding measurements measured for the MGT dataset (non-edited data).

Non-parametric statistical test

We complement this section by presenting the results of a Wilcoxon signed ranks test [32]. The test was ran four times: once on the nine measurements of each comparison criterion (classification accuracy (ACC), reduction rate (RR), preprocessing cost (PC)), and once on the nine measurements of an extra criterion which estimates the overall classification performance. This criterion adopts the idea presented in the statistical comparissons in [48, 47]. More specifically, the overall classification performance measurements were computed by averaging the normalized measurements of the three criteria. Therefore, the measurements were normalized to the interval $[0, 1]$. Since the higher preprocessing cost, the lower the classification performance, we used $1 - normalized(PC)$ as the preprocessing cost. The overall classification performance criterion combines the three criteria by considering all of them as having the same significance.

Table 4.3 illustrates the results of the Wilcoxon tests. The columns with labels "W/L" counts the number of wins and loses, respectively. The columns with label "Wilcoxon" lists the Wilcoxon significance level. When that value is lower than 0.05, we claim that the difference between the compared algorithms is statistically significant. Of course, this threshold is very strict. We observe that this is true in the most cases. In terms of accuracy, CNN-rule has more wins than AIB2 and AIB2 has more wins than IB2. However, there is not statistical difference between the corresponding algorithms. In addition, AIB2 is statistically better than CNN-rule and IB2 in terms of reduction rate, and AIB2 is statistically better than CNN-rule in terms of preprocessing cost. On the other hand, although it is clear that AIB2 is faster than IB2, this is not statistically supported (note that we have adopted the strict threshold $Wilcoxon = 0.05$). Finally, we observe that AIB2 is statistically better than the other algorithms in terms of the overall classification performance. Therefore, AIB2 is preferable when the three criteria have the same significance.

Concerning CNN-rule and IB2 (last table row), we observe that, in all cases, there is statistical significant differences between their performance measurements. IB2 is statistically better than CNN-rule in terms of reduction rate and preprocessing cost. In contrast, CNN-rule is statistically better than IB2 in terms of accuracy.

Table 4.3: Results of Wilcoxon signed ranks test

Methods	ACC		RR		PC		**Overall performance**	
	W/L	**Wilcoxon**	**W/L**	**Wilcoxon**	**W/L**	**Wilcoxon**	**W/L**	**Wilcoxon**
AIB2 vs CNN	3/6	0.767	9/0	**0.008**	9/0	**0.008**	9/0	**0.008**
AIB2 vs IB2	6/3	0.066	8/1	**0.015**	8/1	0.086	7/2	**0.028**
IB2 vs CNN	0/9	**0.008**	9/0	**0.008**	9/0	**0.008**	9/0	**0.008**

4.2.4 Conclusions

This section proposed the AIB2 algorithm, an prototype abstraction algorithm that is based on the well-known condensing IB2 algorithm. The main concept behind AIB2 is that each generated prototype should be near the center of the data area it represents. Like IB2 and contrary to CNN-rule and many other condensing and prototype abstraction algorithms, AIB2 is incremental. This property renders AIB2 appropriate for dynamic domains where new training data is gradually available and for datasets that cannot fit into the main memory.

The experimental results illustrated that AIB2 can achieve higher reduction rates and classification accuracy and lower preprocessing cost than IB2. In addition, AIB2 is preferable to CNN-rule when preprocessing cost and / or reduction rates are more crucial criteria than accuracy and, of course, when an incremental algorithm is required. The improvement offered by AIB2 was statistically confirmed by the Wilcoxon signed ranks test.

4.3 A simple noise-tolerant prototype abstraction algorithm

4.3.1 Motivation and contribution

We have already mentioned that the size of the condensing set, and, therefore, the effectiveness of condensing and prototype abstraction algorithms, depends on the level of noise in the training data as well as the number of distinct classes (the more classes, the more close-class-border items selected or generated). Consequently, a complete preprocessing step may include an editing procedure to remove the noise from the training data. However, editing may be inappropriate because either it may be not able to remove all items that are noise or it may remove useful data. Moreover, editing constitutes an extra, costly preprocessing step that may be unacceptable in some application domains. Thus, there is a need for a noise-resistant or noise-tolerant algorithm. These observations are behind the motivation of the work presented in this section.

The contribution of this section is summarized as follows:

- to examine how the addition of noise affects the performance of two state-of-the-art DRTs, the condensing algorithm CNN-rule and the prototype abstraction algorithm RSP3, and,
- to propose the use of the cluster means produced by k-means clustering on the training sub-datasets belonging to each class as a simple, noise-tolerant prototype abstraction algorithm.

The rest of the section is organised as follows. In Subsection 4.3.2, the noise-tolerant Reduction through k-Means (RkM) algorithm is presented. Subsection 4.3.3 presents a weighing-based RkM variation. Subsection 4.3.4 shows the experimental results and Subsection 4.3.5 concludes the section.

4.3.2 The Reduction through *k*-Means (R*k*M) algorithm

A simple but effective prototype abstraction approach could use k-means clustering. More specifically, we propose the use of the k-means for the construction of the condensing set. Certainly, there are many approaches that use clustering either to summarize similar items [31, 30, 61] or to condense the initial training set [86, 85, 87, 93, 40, 8, 82] for speed-up purposes. Here, we focus on the noise tolerance of DRTs and our purpose is to ascertain if the k-means approach can improve the classification performance. We term this simple prototype abstraction approach Reduction through k-Means clustering (RkM) [95].

RkM involves a clustering preprocessing step on the training dataset. Particularly, k-means clustering is executed on the items of each class. Therefore, for each class, a number of clusters is identified and the class is represented by the mean vectors of these clusters. Considering this simple prototype abstraction approach, it is clear that noise belonging to a class (i.e., items that lie in the data area of another class) and class outliers are represented by a cluster mean lying in the main area of this class. Therefore, we expect RkM to be a noise-tolerant DRT and its effectiveness to not depend as much on an editing procedure like CNN-rule, RSP3 and many other condensing and prototype abstraction algorithms do.

Algorithm 14 RkM

```
Input: $DRF, TS$
Output: $CS$
 1: $CS \leftarrow \varnothing$
 2: for the set of items $X$ that belong to each class $C$ do
 3:     $nc \leftarrow \lceil \frac{|X|}{DRF} \rceil$
 4:     $M \leftarrow$ Use random $nc$ items of $X$ as the initial means for $k$-means clustering
 5:     $NewClusters \leftarrow K$-MEANS($X$, $M$)
 6:     for each cluster $CLU \in NewClusters$ do
 7:         $m \leftarrow$ mean of $CLU$
 8:         $CS \leftarrow CS \cup \{m\}$ labelled by $C$
 9:     end for
10: end for
11: return $CS$
```

An issue that needs to be addressed is the determination of the number of mean items that should represent each class. We deal with this issue by introducing a parameter called Data Reduction Factor (DRF). For each class C, RkM builds $\lceil \frac{|C|}{DRF} \rceil$ clusters, where $|C|$ is the number of items that belong to C. The use of ceiling ensures that each class will be represented by at least one mean. Consequently, the condensing set built by RkM stores only the means produced by k-means clustering. Although the DRF parameter needs to be adjusted through a trial-and-error-procedure, it allows the user to determine the desirable trade-off between accuracy and reduction rate (or, in other words, computational cost). In effect, low DRF values lead to accurate classifiers with gains in execution cost, whereas, high DRF values build very fast classifiers that achieve lower accuracy.

The RkM procedure is outlined in Algorithm 14. The algorithms accepts a training set (TS) and a DRF value and returns a condensing set (CS). Effectively, RkM applies k-means clustering (line 5, see Algorithm 8) on the items of each classes (line 2). After each k-means execution, the cluster means built for the particular class are placed in the CS as prototypes (lines 6–9). The mean item m of cluster CLU (line 7), is computing by averaging the n attribute values of the items, x_i, $i = 1, 2 \ldots |CLU|$, that belong to C. Each mean attribute $m.d_j, j = 1, 2, \ldots n$ is estimated as follows:

$$m.d_j = \frac{1}{|CLU|} \sum_{x_i \in CLU} x_i.d_j, j = 1, 2, \ldots n$$

4.3.3 A weight-based RkM variation

In the RkM algorithm, all prototypes (i.e., mean items) contribute the same to the classification predictions. We attempt to improve the performance of RkM by weighting the means. More specifically, we implemented and tested a weight-based RkM variation.

The weight-based RkM algorithm adds a weight to each mean. The weights are computed by taking into account three factors. Each of them has an equal contribution to the classification. The first factor is the number of items that have been summarized in the specific mean. The second factor considers the dispersion of each cluster. Namely, after the cluster construction for each class, the algorithm computes their dispersion. The lower the dispersion value is, the more the corresponding mean contributes to the classification. In a D-dimensional space, the dispersion value of a cluster C with a mean R is estimated by the following formula:

$$Dispersion_C = \sum_{j=1}^{D} \frac{1}{|C|} \sum_{i=1}^{N} (R_j - item_{i,j} \in C)^2$$

The last factor is based on the distance between the new unclassified item and each mean. The closer a mean is, the higher weight is computed. The final weight of R is computed by adding the normalized to the interval $[0, 1]$, individual weights of the three factors:

$$R_{weight} = norm(w_{f_1}) + norm(w_{f_2}) + norm(w_{f_3}),$$

where

$$norm(w_{f_i}) = \frac{w_{f_i} - min(w_{f_i})}{max(w_{f_i}) - min(w_{f_i})}, i = 1, 2, 3$$

The first two factors are computed for each prototype during preprocessing. The third one depends on the position of the new item and is computed during the classification step. When a new item arrives and must be classified, the k nearest prototypes are retrieved, and then, it is classified to the class that obtains the higher sum of weights.

Unfortunately, the performance of the weight-based RkM algorithm did not meet our expectations. For the datasets we used, it did not show significantly different performance from that of RkM. Here, we describe this variation but omit its performance from the experimental study presented in the following subsection. We should mention that during our experimentation, we tried different combinations of the aforementioned factors. However, none achieved noteworthy performance (higher than that of RkM).

4.3.4 Performance evaluation

Experimental Setup

RkM, CNN-rule and RSP3 were tested against each other using the datasets summarized in Table 4.4. In this experimental study, the first four datasets were retrieved from the UCI Machine Learning Repository[4] [12, 44], while the TXR and PH datasets from the KEEL dataset Repository[5] [6]. We split the LIR, MGT, TXR and PH datasets into training and testing sets using a random sampling procedure. Concerning the LS and PD datasets, we used the training/testing splits provided by the UCI Repository. Of course, RkM can be executed on-line through WebDR[6] (see Appendix A).

To evaluate the performance of the algorithms on datasets with noise, we constructed and tested eight versions of the LIR, LS, PD, and TXR datasets with varying degree of noise (from 0% to 70% with step=10). The first version was the original dataset (without extra noise), whereas the last version contained 70% noise. The MGT and PH datasets have only two classes and so they can not afford high levels of additional noise. Hence, we used four versions for these datasets with noise ranging from 0% to 30%. Note that the original MGT dataset already contains a considerably high level of noise in its original form.

The datasets with noise for all datasets were constructed by adding random noise of a specific probability to the datasets. For instance, when a dataset with 20% noise was needed, the class attribute of each item of the corresponding training set was modified by selecting another random class-attribute with a probability of 0.2. Therefore, for each tested algorithm, eight or four pairs of performance measurements (reduction rate and classification accuracy) were obtained. For each dataset, we present a pair of figures (Figure 4.3–4.8): figures denoted by (a) show the reduction rates of each algorithm, whereas, figures denoted by (b) show the corresponding classification accuracy values. The latter figures include an extra curve for the conventional k-NN (classification by scanning the original training data). Finally, we note that

[4]http://archive.ics.uci.edu/ml/
[5]http://sci2s.ugr.es/keel/datasets.php
[6]https://ilust.uom.gr/webdr

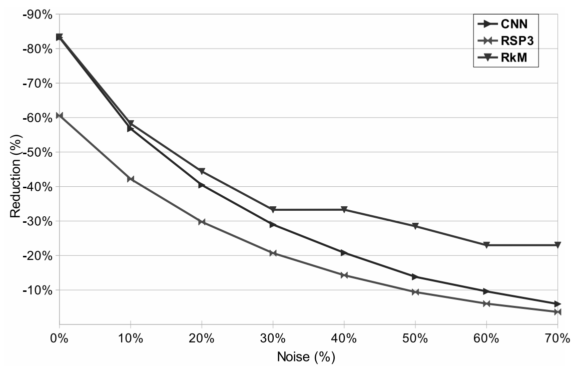
Table 4.4: Datasets description

Dataset	Training items	Testing items	Attributes	Classes
Letter Recognition (LIR)	15000	5000	16	26
Landsat Satellite (LS)	4435	2000	36	6
Pen Digits (PD)	7494	3498	16	10
Magic Gamma Telescope (MGT)	14000	5020	10	2
Texture (TXR)	4400	1100	40	11
Phoneme (PH)	4000	1404	5	2

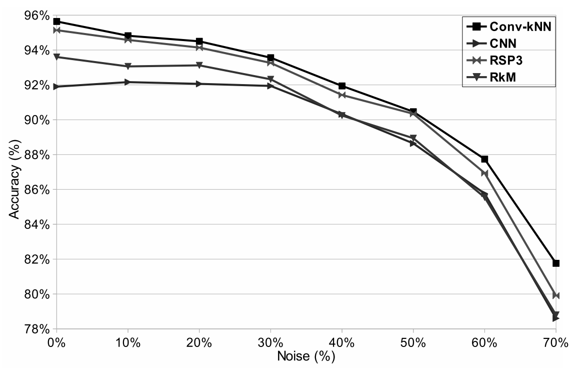
the datasets were used without data normalization or any other data transformation, and that the distance metric used was the Euclidean distance.

We compared the performance of RkM to that of CNN-rule and RSP3 that are good representatives of the two different algorithm categories, i.e., condensing and prototype abstraction algorithms. Initially, each one of these algorithms was executed in order to produce the corresponding condensing set by scanning the initial training data. Then, for each item of the testing set, the k-NN classifier made a classification prediction by searching for nearest neighbours over the condensing set produced. The chosen k parameter values for all the aforementioned classifiers were the ones that achieved the highest classification accuracy. In effect, we ran each experiment many times for different k values and kept and report the best one. Ties during voting of the major class were resolved by choosing the class of the nearest neighbour. Note that we did not follow a tuning procedure that implies that the best parameters should be obtained by using only the training set. Since all algorithms include only the k parameter during the classification step, for all of them, we simply report the highest accuracy achieved by the k parameter when it classifies the testing portion using the corresponding training portion.

For RkM, recall that DRF determines the number of means (prototypes) that will be generated for each one class. The value of this parameter can be used to adjust the computational cost vs. accuracy trade-off required by the application domain. In our experiments, we wanted to build RkM classifiers that would achieve accuracy comparable to that of CNN and RSP3. In particular, in most cases, DRF values were appropriately adjusted to achieve an accuracy between the corresponding accuracy values of CNN and RSP3.

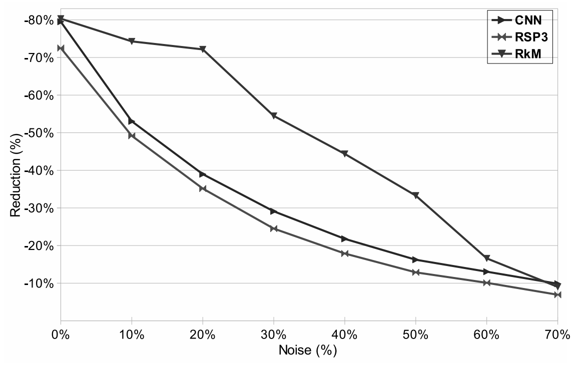


(a) Reduction Rate

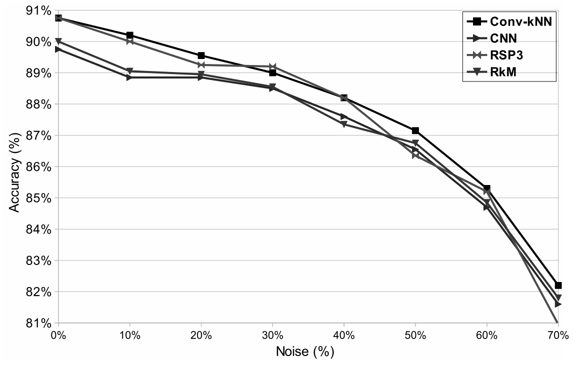


(b) Accuracy

Figure 4.3: LIR (Reduction Rate and Accuracy)



(a) Reduction Rate

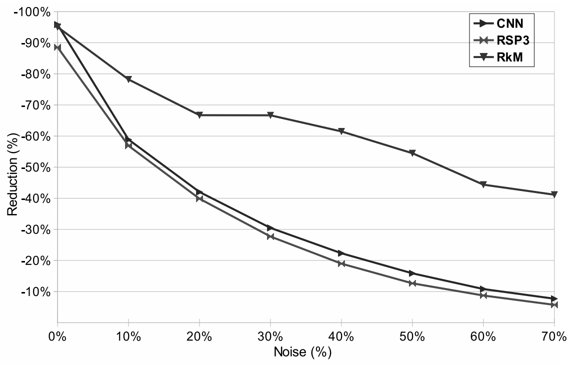


(b) Accuracy

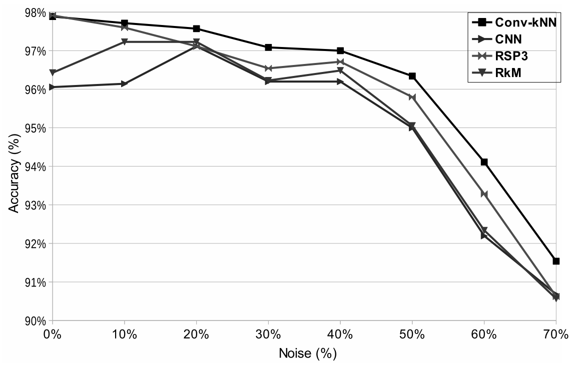
Figure 4.4: LS (Reduction Rate and Accuracy)

Experimental measurements

As we expected, the experimental measurements illustrated in Figures 4.3–4.8 demonstrate that RkM can be characterized as a noise-tolerant approach. The preprocessing procedures of CNN-rule and RSP3 are not able to reduce as much the size of the condensing set when the initial training set contains noise. In the case of CNN-rule, this is because more items are misclassified and so, they are moved to the condensing set. In the case of RSP3, the algorithm keeps on splitting the many non-homogeneous groups that are produced. On the other hand, RkM does not seem to be affected as much by the addition of noise. Almost in all cases, CNN-rule achieved higher reduction rates than RSP3 and RSP3 achieved higher accuracy values than CNN-rule.

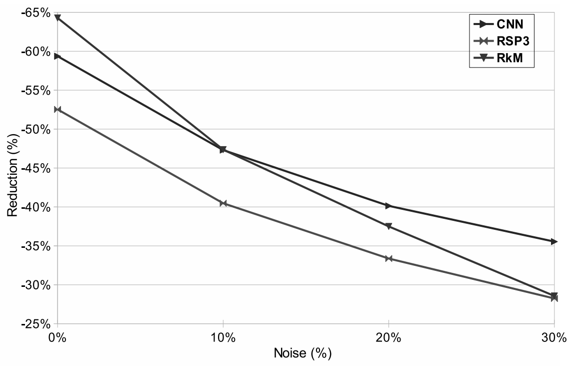


(a) Reduction Rate

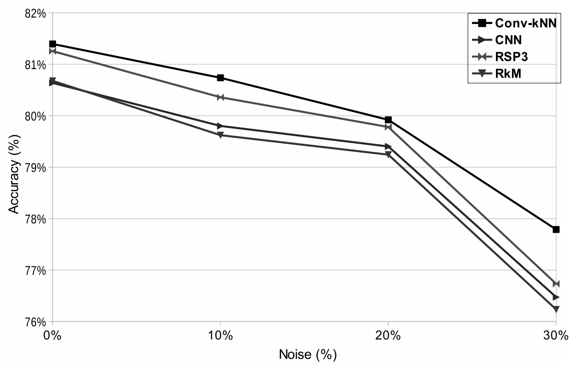


(b) Accuracy

Figure 4.5: PD (Reduction Rate and Accuracy)



(a) Reduction Rate

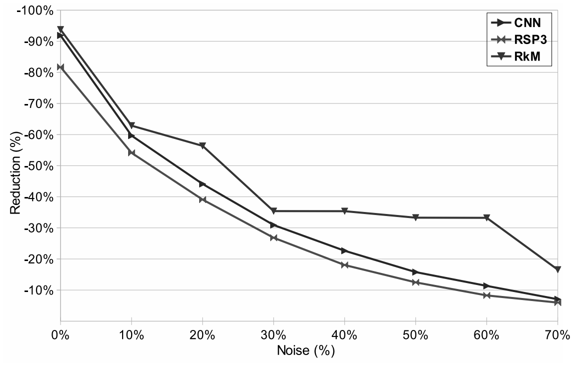


(b) Accuracy

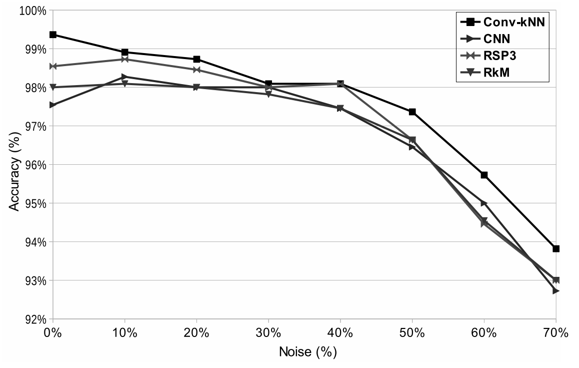
Figure 4.6: MGT (Reduction Rate and Accuracy)

The reduction rates achieved by RkM were higher in the cases of the LS (Figure 4.4), PD (Figure 4.5) and PH (Figure 4.8) datasets than that of the LIR (Figure 4.3) and TXR (Figure 4.7) datasets. This is because, in the latter datasets, the classifiers that use the condensing sets built by CNN-rule and RSP3 achieved high accuracy values that the corresponding RkM classifier can not easily reach. Consequently, RkM classifiers that use condensing sets with more prototypes (lower DRF values) are required in order to achieve comparable performance. However, even in the cases of the LIR and TXR datastes, it is obvious that the reduction rates of RkM are higher than that of the other two algorithms.

In the case of the MGT dataset, the RkM classifier did not perform well. However, in the original version of the MGT dataset, which already contains many items that are noise,

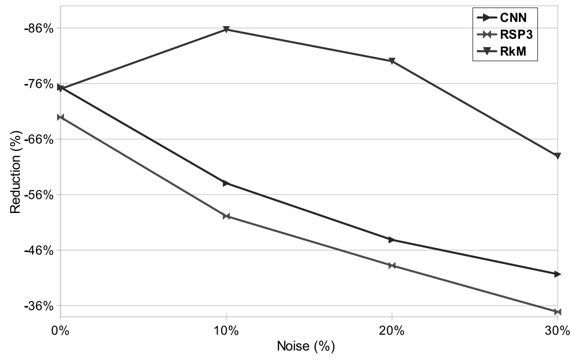


(a) Reduction Rate

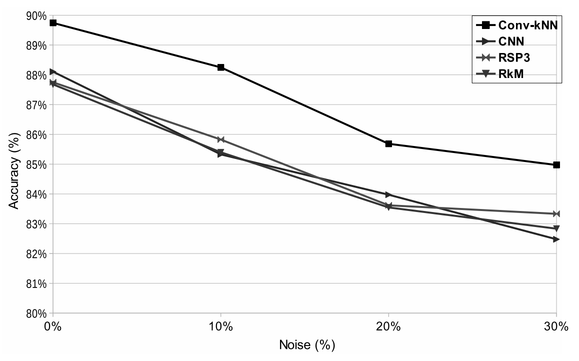


(b) Accuracy

Figure 4.7: TXR (Reduction Rate and Accuracy)



(a) Reduction Rate

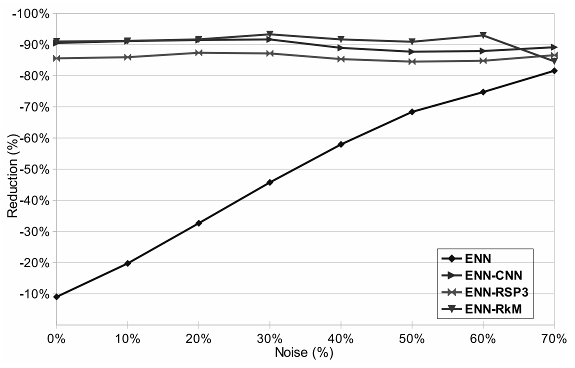


(b) Accuracy

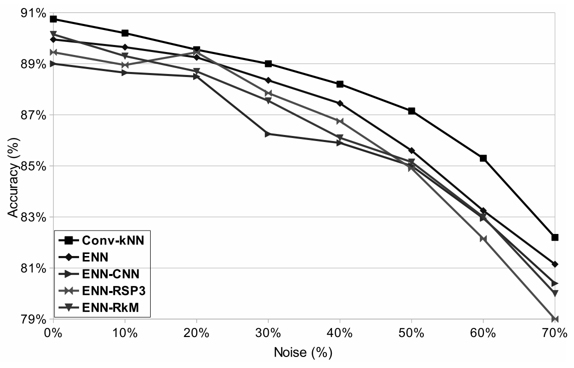
Figure 4.8: PH (Reduction Rate and Accuracy)

RkM had higher reduction rates than CNN-rule and RSP3 and achieved the same accuracy as CNN-rule. Finally, in many cases, the RkM classifier achieved higher accuracy values than those we present here. However, they involved lower reduction rates. Our purpose was to ascertain whether RkM is a noise-tolerant approach and so, we focused on the reduction rates that it achieved. The user of RkM can define the desirable trade-off between accuracy and cost through the DRF parameter.

RkM performed very well on edited datasets. We repeated the experiments on the the LIR, PD, and LS datasets after first applying editing on each dataset in order to remove the noise. For editing purposes, we used the ENN-rule with $k = 3$ [131, 49, 84]. RkM managed to reach or exceed the reduction rates of CNN-rule and RSP3, while achieving comparable ac-



(a) Reduction Rate



(b) Accuracy

Figure 4.9: Edited LS (Reduction Rate and Accuracy)

curacy. Figure 4.9 shows the results obtained on the LS dataset. Finally, we should mention that the accuracy measurements presented may have been slightly different had we used different training/testing splits. However, we are mainly interested in the reduction rates that the presented methods achieved and so, the slightly different accuracy measurements are not critical.

Concluding this section, we focus on the preprocessing cost needed for the construction of the condensing sets of the three algorithms. Certainly, these cost measurements are relevant only once, in the beginning of the data mining process. However, there are many application domains that periodically accept new training items and so the reconstruction of the condensing set may be necessary. Thus, the preprocessing cost may be a critical issue.

Table 4.5 presents these measurements for each original dataset in terms of distance computations. Considering the results, we conclude that RSP3 is the most expensive approach. This is because RSP3 contains a function that finds the most distant items in each group. The preprocessing cost of RkM depends on the DRF value. The last column of Table 4.5 lists the DRF values that we used in order to build the RkM classifiers presented in Figures 4.3–4.8 (noise= 0%). For the LIR, LS, PD and TXR datasets, RkM was cheaper than the other algorithms. On the other hand, for the MGT and PH datasets, its cost was considerably higher. If the application domain cannot afford the RkM preprocessing step, it can be sped-up by adopting a more efficient stopping condition for k-means than the complete cluster consolidation that we used.

Table 4.5: Experimental results: Preprocessing cost (distance computations)

Dataset	**CNN-rule**	**RSP3**	**RkM**	**DRF**
LIR	145,386,010	291,151,380	11,576,185	6
LS	13,545,272	28,929,950	5,771,993	5.1
PD	7,940,953	70,561,629	4,409,382	20
MGT	217,900,759	412,752,916	323,718,012	2.8
TXR	5,189,518	25,361,045	1,278,585	18
PH	13,532,827	17,847,352	22,809,139	4

4.3.5 Conclusions

In this section, we examined how the state-of-the-art algorithms CNN-rule and RSP3 are negatively affected by the addition of noise in the training data. In addition, we demonstrated that the well-known k-means clustering can be used as a noise-tolerant prototype abstraction algorithm. The experimental results showed that the Reduction through k-Means (RkM) algorithm can be used to achieve comparable accuracy to CNN-rule and RSP3 but with much higher reduction rates, especially on datasets with high levels of noise. Even on datasets without noise, RkM can compete with CNN-rule and RSP3 in terms of data reduction rate and classification accuracy.

Chapter 5

Hybrid methods for fast *k*-NN classification

5.1 Introduction

In this chapter, new hybrid methods for fast k-NN classification are proposed. Hybridization is accomplished by combining two different speed-up strategies in order to achieve the desirable classification performance. The chapter contributes three hybrid methods as well as variations of these methods. Contrary to prototype abstraction and condensing algorithms, the methods introduced by this chapter do not reduce the storage requirements of the training data. However, they can effectively speed-up the classification processes.

Section 5.2 proposes the combination of the minimum distance and the conventional k-NN classifiers in a hybrid schema. The goal is fast classification without the need of preprocessing [101]. The minimum distance classifier [38] is an extremely fast classification approach but it usually achieves much lower accuracy than the k-NN classifier. The proposed algorithm aims at the reduction of computational cost, by keeping classification accuracy at a high level. The experimental results obtained illustrate that the proposed approach can be applicable in dynamic, time-constrained environments where model-free classifiers are required.

Section 5.3 presents an adaptive hybrid and cluster-based method for speeding up the k-NN classifier [102, 90]. It reduces the computational cost as much as possible while maintaining the classification accuracy at high levels. The method is based on k-means clustering and consists of two main parts: (i) a preprocessing algorithm that builds a two-level, cluster-based data structure, and, (ii) a hybrid classifier that classifies new items by accessing either the first or the second level of the data structure according to a set of user-defined criteria (input parameters). The proposed approach was tested on seven datasets and the experimental measurements were statistically validated by the Wilcoxon signed ranks test. The results show that the proposed method can be used either to achieve high accuracy at a slightly higher computational cost or to reduce the cost at a minimum level by slightly sacrificing accuracy.

The main disadvantage of the aforementioned method is that the preprocessing and the classification algorithms are parametric and require a trial-and-error procedure to properly adjust their parameters. In effect, the performance of classification is controlled by these parameters. In Section 5.4, a non-parametric hybrid method for fast k-NN classification is introduced [94, 97]. It is also based on a two-level speed-up data structure and on classifiers that access this structure. The method follows the idea of the homogeneous clusters presented in Chapter 3. Furthermore, the section demonstrates how the particular method can improve the classification performance on condensing sets built via Data Reduction Techniques (DRTs). The proposed method was evaluated using eight datasets and was compared to known speed-up methods. The experimental measurements were also validated with the Wilcoxon signed ranks test. The results show that the new method leads to fast and accurate classification, and, in addition, it involves low pre-processing computational cost.

5.2 Fast hybrid classifiers based on the minimum distance and the k-NN classifiers

5.2.1 Motivation and contribution

Although condensing and prototype abstraction algorithms are usually effective, they introduce a costly and, in some cases, complicated preprocessing step, since they build a condensing set to speed-up the nearest neighbours searching procedure. In effect, the condensing set is the classification model. When the classification is applied on databases where frequent content changes occur (dynamic environments), the repeated execution of this step, that ensures the effectiveness of the condensing set, is prohibitive. In such dynamic environments, there is a need of a lazy, model-free classifier.

The Minimum Distance Classifier (MDC) [38] can be used as a very fast classification approach. MDC can be characterized as a very simple prototype abstraction algorithm since it computes a representative item (centroid or mean) for each one class. This is the vector obtained by averaging the attribute values of the items of each class. When a new item t is to be classified, the distances between t and all centroids are calculated and t is classified to the class of the nearest centroid. MDC avoids the high computational cost of scanning the whole training data. On the other hand, the accuracy achieved by MDC depends on how the items of the training set are distributed in the multidimensional space. It is worth mentioning that the centroid-based model introduced by MDC has been successfully applied for document categorization [55].

The motivation of the work presented in this section is to explore how the speed of MDC can be exploited in order to achieve fast and accurate classification that does not depend on models (e.g., indexes, condensing sets). Moreover, the motivation includes addressing the problem of classifying large, high-dimensional datasets, where a sequential, exhaustive search of the whole training set is required to locate the k nearest neighbours (conventional k-NN classifier). This is the case when indexing is not applicable and dimensionality reduction negatively affects the performance. Furthermore, we want to avoid complicated preprocessing procedures such as data reduction. The simple centroid based model of MDC involves only one pass over the training data to compute the mean vector of each class.

This section introduces a new fast hybrid classification algorithm that does not need any speeding-up data structure (i.e., indexing) or any transformation of the training data (i.e., data reduction). The reduction of the classification cost is achieved by the combination of the conventional k-NN and the minimum distance classifiers. The proposed algorithm begins by computing a centroid for each class. Then, it tries to classify new items by examining the centroids and by applying certain criteria. If the set criteria are not satisfied, it proceeds by applying k-NN search over the entire training set. The contribution of the work is summarized as follows:

- A novel, model-free classification algorithm is proposed that is independent of data dimensionality and can be applied for repeated classification tasks on dynamic databases, where frequent content changes occur, since it avoids expensive preprocessing procedures on the training data, and,

- The main classification algorithm and two variations that extend its basic idea are considered in relation to the set goal of achieving high accuracy while reducing the classification cost as much as possible.

The rest of this section is organized as follows. Subsection 5.2.2 considers in detail the proposed classification algorithm and its variations. In Subsection 5.2.3, experimental results based on various datasets are presented. The work concludes in Subsection 5.2.4.

5.2.2 The proposed algorithms

This section presents a fast, hybrid and model-free classification algorithm [101]. In addition to the main algorithm, two variations that achieve extra classification cost savings are proposed. In the order presented, each one variation comprises an extension to its predecessor. As expected, the improvement comes at the cost of a decrease in accuracy.

The algorithm and its variations are based on the same idea: They initially search for nearest neighbours in a new, smaller dataset constructed by one pass over the training data. This dataset contains only a representative item for each one class. Upon failure to meet the set acceptance criteria, classification proceeds by the conventional k-NN classifier. Each representative item is computed by calculating the average value of each attribute in each one class. Thus, the computed vector can be considered to comprise the centroid of the cluster corresponding to the class. Therefore, if the initial dataset contains one thousand items in fifteen dimensions and ten classes, the new dataset will have only ten items in fifteen dimensions. Evidently, the more the dataset of centroids is used, the less the execution time involved. The following subsections below, outline the main classification algorithm and its two variations.

Fast Hybrid Classification Algorithm

The Fast Hybrid Classification Algorithm (for simplicity FHCA) is based on the difference of the distances between the new unclassified item and the centroids (see Algorithm 15). More specifically, for each new item x, the algorithm takes into consideration the two nearest centroids A, B (A is the nearest and B is the second nearest) and their distances from x: d(x, A), d(x, B). If the difference between d(x, A) and d(x, B) exceeds a predefined threshold (line 5), x is classified to belong to the class represented by centroid A (line 6), otherwise the k nearest neighbours are retrieved from the initial training set (TS) in order to determine the class of x (lines 8-10).

Algorithm 15 FHCA

```
Input: $TS$, $Threshold$, $k$
 1: Scan $TS$ to compute the centroids
 2: for each unclassified item $x$ do
 3:     Compute the distances between $x$ and the centroids
 4:     Retrieve the nearest centroid $A$, and the second nearest centroid $B$, using the Euclidean
        distance metric
 5:     if (distance($x$, $B$) - distance($x$, $A$)) $\geq$ Threshold then
 6:         Classify $x$ to the class of centroid $A$
 7:     else
 8:         Retrieve the $k$ NNs from $TS$
 9:         Find the major class (the most common one among the $k$ NNs. In case of a tie, it is
            the class of the single Nearest Neighbour)
10:         Classify $x$ to the major class
11:     end if
12: end for
```

FHCA is more accurate than the two variations presented in the following paragraphs. In some cases, it reaches the classification accuracy of the conventional k-NN classifier, at a significantly less classification cost. However, the performance depends on the value of the predefined threshold.

When the threshold parameter is set to zero, the centroid-based approach (MDC) classifies all the new samples (since the “if” condition calculates to “true”). On the other hand, if the threshold is set to a relatively high value, it is possible that all new items are classified by the conventional k-NN classifier (the “else” clause in line 7 in Algorithm 15). These properties indicate that the threshold value adjustment should be made carefully, as it comprises a classification cost vs. accuracy trade-off decision for the application considered.

FHCA - Variation I

The first FHCA variation (FHCA-V1), which is illustrated in Algorithm 16, is an extension of the main FHCA algorithm, since it uses the distance difference criterion the same way FHCA does. In addition, FHCA-V1 attempts to classify even more new incoming items without falling back to the conventional k-NN classifier. In particular, if the distance difference criterion fails to classify the incoming item x (lines 6 and 7 in Algorithm 16), FHCA-V1 considers the region of influence of each one class centroid involved. We define the class region of influence to be the average distance of the training set class items from the corresponding centroid. In case x lies within the region of influence of only one class (class A in Figure 5.1), x is classified to belong to the class in question (lines 8 and 9 in Algorithm 16). Otherwise, if x lies within the region of influence of more than one class, the algorithm proceeds as in the conventional k-NN classifier (lines 11-13 in Algorithm 16). In practice, the only one difference between FHCA and FHCA-V1 is the "else if" part of pseudo-code in lines 8 and 9 of Algorithm 16.

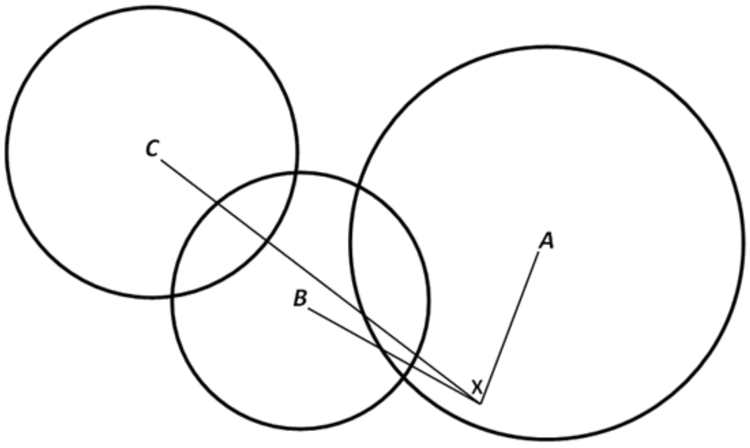


Figure 5.1: FHCA-V1 classification case

Contrary to the FHCA, FHCA-V1 requires two preprocessing passes, one for calculating the class representative items (centroids, as in FHCA: line 1 in Algorithms 15 and 16), and one for calculating the class regions of influence in the training set (line 2 in Algorithm 16). Even this extra pass over the training set (TS) is insignificant compared to the preprocessing procedures involved by DRTs.

FHCA - Variation II

The second FHCA variation (FHCA-V2) extends FHCA-V1 to include one more classification criterion. The latter handles the case where the unclassified item x lies within more than one class regions of influence (Figure 5.2, lines 10 and 11 in Algorithm 17). In this case, x is classified to the class of the nearest centroid whose region of influence embraces it. The example in Figure 5.2 illustrates such a case: suppose that the distance difference criterion is not able to classify x, i.e. x lies closer to A than to B, but the difference between the two distances does not reach the predefined threshold. Also, suppose that x lies within the regions

Algorithm 16 FHCA-V1

```
Input: $TS$, $Threshold$, $k$
 1: Scan $TS$ to compute the class centroids
 2: Re-scan $TS$ to compute the region of influence of each one class centroid
 3: for each unclassified item $x$ do
 4:     Compute the distances between $x$ and the class centroids
 5:     Find the nearest centroid $A$, and the second nearest centroid $B$, using the Euclidean
        distance metric
 6:     if (distance($x$, $B$) - distance($x$, $A$)) $\geq$ Threshold then
 7:         Classify $x$ to the class of centroid $A$
 8:     else if $x$ belongs to the region of influence of only one class then
 9:         Classify $x$ to this class
10:     else
11:         Retrieve the $k$ NNs from $TS$
12:         Find the major class (the most common one among the $k$ NNs. In case of a tie, it is
            the class of the Nearest Neighbour)
13:         Classify $x$ to the major class
14:     end if
15: end for
```

of influence of both A and B. In this case, FHCA-V2 classifies x to belong to the class of the nearest centroid (the class of centroid A in Figure 5.2), skipping the classification cost of the conventional k-NN classifier. It is only in cases where unclassified items fail to meet both of the FHCA-V1 and FHCA-V2 set criteria that the algorithm proceeds to apply the conventional k-NN classifier. In this respect, FHCA-V2 involves less computational overhead when compared to FHCA, FHCA-V1, and, of course, the conventional k-NN classifier.

Discussion

A key factor for the proposed classifier and its variations is the adjustment of the threshold (input) parameter. In the case of FHCA, the value of this parameter influences the number of new incoming items classified by the centroid-based model: the more the centroid-based approach is used, the less is the classification cost involved. In the cases of FHCA-V1 and FHCA-V2, the focus is on the unclassified items that cannot be classified by the distance difference criterion. Two additional centroid-based classification criteria are introduced, in an attempt to avoid the classification cost of the conventional k-NN classifier. The latter becomes the only one option available when both of the FHCA-V1 and FHCA-V2 set criteria fail to classify the unclassified

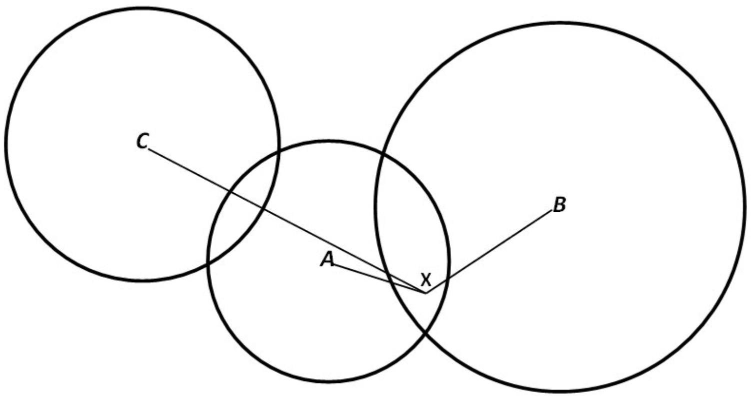


Figure 5.2: FHCA-V2 classification case

Algorithm 17 FHCA-V2

```
Input: $TS$,$Threshold$, $k$
 1: Scan $TS$ to compute the class centroids
 2: Re-scan $TS$ to compute the region of influence of each one class centroid
 3: for each unclassified item $x$ do
 4:     Compute the distances between $x$ and the class centroids
 5:     Find the nearest centroid $A$, and the second nearest centroid $B$, using the Euclidean
        distance metric
 6:     if (distance($x$, $B$) - distance($x$, $A$)) $\geq$ Threshold then
 7:         Classify $x$ to the class of centroid $A$
 8:     else if $x$ belongs to the region of influence of only one class then
 9:         Classify $x$ to this class
10:     else if $x$ belongs to the regions of influence of more than one class then
11:         Classify $x$ to the class of nearest centroid whose region of influence embraces $x$
12:     else
13:         Retrieve the $k$ NNs from $TS$
14:         Find the major class (the most common one among the $k$ NNs. In case of a tie, it is
            the class of the Nearest Neighbour)
15:         Classify $x$ to the major class
16:     end if
17: end for
```

item. Obviously, FHCA-V2 utilizes the centroids dataset as much as possible (i.e., in all three set criteria) and represents the fastest variation. In contrast, FHCA is the slowest of the three approaches, since it applies centroid-based classification solely on the basis of the distance difference criterion.

A threshold auto-adjustment method is relatively easy to implement in the form of a routine that accepts a value for the desirable accuracy level, and it iteratively considers a number of different threshold values. Having reached the desirable accuracy level, the routine returns the corresponding threshold value.

It is noted that the worst-case scenario for the proposed classification approaches is when the centroid-based part does not classify any new item. In this case, the execution time involves the k-NN figure, the small overhead of the centroids creation (one pass of the training data) and the small overhead introduced by the cost of distance computations between new data and the class centroids (e.g., if there are ten classes, ten distances must be computed for each new item). In effect, preprocessing cost is almost insignificant.

The proposed algorithm can be modified so that upon failure to use the centroid-model, the k-NN part takes into account only the items of the two classes with the nearest centroids to the query point. In this case, both the classification cost and the classification accuracy are highly depended on the number of classes. Experiments conducted on the datasets presented in Section 5.2.3, have shown that although this approach is very fast, the accuracy is significantly reduced. Consequently, it is not included in the experimental study of the following subsection.

5.2.3 Performance evaluation

Experimental setup

The proposed algorithms were coded in C and tested using five datasets distributed by the UCI machine learning repository[1] [12, 44]. Table 5.1 summarizes the datasets used. The last table column lists the k value found to achieve the highest accuracy. Since, the k-NN classifier requires the computation of all distances between each item of the testing set and the training data, the classification cost of the k-NN classifier can be easily estimated by multiplying the contents of second table column by the contents of third column. For example, $15,000 \times 5,000 = 75,000,000$ distances are computed for the letter recognition dataset.

[1]`http://archive.ics.uci.edu/ml/`

Table 5.1: Datasets description

Dataset	**Training data**	**Testing data**	**Attributes**	**Classes**	**Best *k***
Letter recognition	15000	5000	16	26	4
Magic gamma telescope	14000	5020	10	2	12
Pendigits	7494	3498	16	10	4
Landsat satellite	4435	2000	36	6	4
Shuttle	43500	14000	9	7	2

In addition, classification procedures are usually applied to training data with high level of noise. The removal of noise introduces an extra preprocessing procedure (i.e., execution of an editing routine). However, we are interested in developing classifiers that do not need any preprocessing task (either to remove the noise or to build a speed-up model such as condensing set or multi-attribute indexing structures).

To evaluate the performance of the proposed algorithm on data with noise, three more datasets were developed by adding random noise to three of the datasets of Table 5.1. Particularly, 40% of noise was added to the Letter recognition, Pendigits and Landsat satellite datasets. Namely, for each item of the training set of these datasets, the class attribute was modified by choosing other class attribute with a probability of 0.4. By executing the conventional k-NN classifier on these noisy datasets, it was found that the highest accuracy is achieved for $k = 13$, $k = 18$ and $k = 21$ respectively (the more the level of added noise, the higher the value of k needed to achieve the highest accuracy). The other two datasets were not transformed into a noisy mode, since the Magic telescope dataset has already a high level of noise and the Shuttle dataset is a skewed dataset with two very rare classes and approximately 80% of the data belonging to one of the seven classes.

The proposed algorithm and its variations use the distance difference Threshold (T) as a parameter. This parameter should be adjusted, so that the desirable trade-off between accuracy and classification cost can be achieved. For this reason, several experiments with different T values were conducted. For FHCA and FHCA-V1, only two T values are reported (T_1 and T_2). The first value builds an accurate classifier (comparable to the conventional k-NN classifier, when possible), and the second value builds a faster classifier which achieves up to 5% lower accuracy than the one build with the first value. For FHCA-V2, which is the fastest approach, only the T value that achieves a high accuracy is reported. Table 5.2 presents the T values for

Table 5.2: T parameter values

Dataset	**FHCA** (T_1)	**FHCA** (T_2)	**FHCA-V1** (T_1)	**FHCA-V1** (T_2)	**FHCA-V2**
Letter recognition	1.6	0.8	1.7	0.7	1.4
Magic gamma telescope	17	10	16	6	14
Pendigits	44	18	35	17	35
Landsat satellite	31	13	31	8	34
Shuttle	30	25	35	28	25
Letter recognition (noisy)	0.9	0.5	0.9	0.5	1.3
Pendigits (noisy)	26	13	33	12	14
Landsate satellite (noisy)	19	14	21	12	10

Table 5.3: Experimental results for CNN-rule: Reduction Rate and Preprocessing Cost in distance computations

Dataset	**Condensing set size**	**Reduction rate (%)**	**Computations**	**Best** k
Letter recognition	2517	83.22	145,386,010	1
Magic gamma telescope	5689	59.36	217,900,759	22
Pendigits	312	95.84	7,940,953	1
Landsat satellite	909	79.50	13,545,272	6
Shuttle	300	99.31	57,958,973	1
Letter recognition (noisy)	11806	21.29	205,175,615	16
Pendigits (noisy)	5822	22.31	51,891,038	20
Landsat satellite (noisy)	3469	21.78	17,918,173	30

each dataset. Please note that since the performance of the classifiers are estimated in terms of two comparison criteria (accuracy and classification cost), there is not a unique dominant parameter value. A good parameter adjustment in terms of accuracy may deteriorate the classification cost and vice versa. Therefore, we cannot tune parameters. In effect, in the next subsection, we just present the performance measurements of good representative cases of the algorithms.

FHCA and its variations involve the same centroid-based part of pseudocode (lines 5, 6 in Algorithm 15 and lines 6, 7 in Algorithm 16) which, effectively, comprises a MDC component. In this respect, the MDC component classifies the same number of items in each one testing set, despite the classification approach used (FHCA, FHCA-V1 or FHCA-V2). For comparison purposes, the experimental measurements of a 'MDC, only' approach are included in Table 5.4 of the following paragraph.

Furthermore, for comparison purposes, the CNN-rule was executed. Table 5.3 presents the experimental results obtained from the execution of the CNN-rule reduction procedure on the eight datasets (the five original and the three datasets with extra noise). Column four lists the number of distance computations needed for the construction of the condensing set. The last column lists the k value found to achieve the highest accuracy when the resulting condensing set is used to classify the items of the corresponding testing sets. Table 5.4 lists the performance of CNN-rule: accuracy and classification cost obtained from the execution of the k-NN classifier on the condensing set built by CNN-rule (CNN k-NN). Of course, these values do not contain the preprocessing computational cost (fourth column in Table 5.3) introduced by the condensing set construction (preprocessing step on the available training data).

Experimental measurements

Here, the proposed classifier and its variations are compared to each other and to the conventional k-NN, Minimum Distance, and, CNN k-NN classification algorithms, by setting the T parameters values presented in Table 5.2. For each one dataset, two experimental measurements were taken for each one classification approach: (i) classification accuracy, and (ii) distance computations as a percentage of the distance computations needed for the conventional k-NN. To give a feeling of the actual classification cost, for the conventional k-NN this second measurement represents the actual number of distance computations. Experimental results obtained by varying the value of T are not presented here[2].

[2]Detailed experimental results and diagrams available at:
`http://users.uom.gr/~stoug/SISAP2011.zip`

Table 5.4: Experimental results: Accuracy (Acc (%)), Computational Cost (%)

Dataset		**FHCA (T_1)**	**FHCA (T_2)**	**FHCA-V1(T_1)**	**FHCA-V1(T_2)**	**FHCA-V2**	**CNN k-NN**	**MDC**	**k-NN**
Letter recognition	Acc:	95.24	90.78	92.06	87.00	71.46	91.9	58.08	95.68
	Cost:	84.39	64.93	76.63	55.15	27.33	16.78	0.17	75,000,000
Magic gamma telescope	Acc:	80.02	75.26	74.72	72.00	72.39	80.64	68.92	81.39
	Cost:	44.11	23.48	28.98	9.64	10.34	40.66	0.01	70,230,000
Pendigits	Acc:	97.08	92.02	88.54	87.22	86.54	96.05	77.76	97.88
	Cost:	62.74	30.89	32.2	20.40	19.92	4.16	0.13	26,214,012
Landsat satelite	Acc:	90.05	85.1	83.00	80.70	82.40	89.75	77.50	90.75
	Cost:	57.03	25.38	30.83	10.13	20.28	20.50	0.14	8,870,000
Shuttle	Acc:	99.82	98.19	95.15	95.12	81.57	99.85	79.57	99.88
	Cost:	53.23	39.77	43.44	35.06	11.29	0.7	0.02	630,750,000
Letter recogn. (noisy)	Acc:	91.06	86.06	89.14	84.36	62.72	90.32	53.98	91.82
	Cost:	83.05	64.69	78.47	61.71	21.47	78.71	0.17	75,000,000
Pendigits (noisy)	Acc:	96.17	91.71	93.31	88.65	78.7	96.20	75.90	97.00
	Cost:	67.88	38.73	66.74	29.23	4.85	77.69	0.13	26,214,012
Landsat satellite (noisy)	Acc:	87.80	85.05	86.55	82.30	75.05	87.6	71.40	88.30
	Cost:	63.33	47.58	63.13	36.08	8.28	78.22	0.14	8,870,000

Letter recognition dataset: FHCA almost reached the accuracy level of k-NN, when the threshold value was set to $T = 1.6$. In particular, FHCA was found to achieve an accuracy of 95.24% and a 15.6% reduction in the classification cost. To obtain a faster classifier, one should decrease the threshold value (T). For instance, when T was set to 0.8, FHCA achieved an accuracy of 90.78%, with 35% lower cost than that of k-NN. FHCA-V1 was affected by the extended overlapping of centroid regions of influence in the given dataset. As a result, the centroid-based part of FHCA-V1 managed to classify only a few more items (8%–10%) than that of FHCA. For $T = 1.6$, FHCA-V1 was measured to classify the testing data with an accuracy of 91.96% and 25% less classification cost. Finally, for $T = 1.6$, FHCA-V2 needed only 27% of the distance computations. However, the accuracy was only 71.6%.

Magic gamma telescope dataset: In this dataset, the proposed algorithm and its variations were measured to perform better than in the previous dataset. In addition, their measurements are comparable to these of CNN k-NN. Concerning FHCA, $T = 17$ comprises a good choice for the threshold parameter, since FHCA achieves an accuracy value of over 80% while it reduces the cost by almost 56%. Although not shown in Table 5.4, for $T = 38$, FHCA can achieve an accuracy of 81.36% that is very close to the accuracy of k-NN but with only a 10% improvement in classification cost. A fast classifier can be developed by setting $T = 10$. In this case, FHCA makes predictions with an accuracy of 75.26% with only 23.48% of the k-NN cost. FHCA-V1 achieved its best accuracy performance (74.72%) for $T = 16$, with 28.98% of the classification cost. For $T = 6$, FHCA-V1 has only 9.64% of the k-NN cost (90.36% reduction) and classifies the testing data with an accuracy of 72%. FHCA-V2 was found to never exceed the accuracy value of 73% (for $T = 14$, accuracy: 72.39%). However, FHCA-V2 executed very fast (for $T = 14$, the cost was reduced by almost 90%).

Pendigits dataset: Concerning FHCA, two reference-worthy experimental measurements are obtained by setting $T = 44$ and $T = 18$. These adjustments achieved an accuracy of 97.08% and 92.02% respectively and had 62.74% and 30.89% of the k-NN cost. It is noted that for $T = 55$, FHCA reached the k-NN accuracy, executing with almost 23% lower cost. Furthermore, for $T = 32$, the cost is 51.4% and the accuracy 95%. FHCA-V1 achieved its best accuracy (88.54%) with 32.2% of the cost for $T = 35$. For $T = 17$, FHCA-V1 achieved an accuracy of 87.22% and required almost 25% of the k-NN cost. FHCA-V2 was measured to execute faster. For $T = 35$, it needs 19.92% of the distance computations of k-NN and it achieves an accuracy of 86.54%.

Landsat satellite dataset: FHCA performed a little better than k-NN. More specifically, for $T = 38$, it achieved the best possible accuracy value (90.85%), with a 32.8% decrease in computations. For all other threshold values, it fell a little behind in accuracy, but executed even faster. For instance, by setting $T = 31$, FHCA achieved an accuracy of 90.05% (very close to the k-NN accuracy) and spent 57.03% of the k-NN classification cost. Also, for $T = 13$, only 25% of the cost was required (accuracy: 85.1%). Finally, for $T = 26$, FHCA achieved an accuracy of 89% with almost half the cost. The two variations performed almost the same in accuracy, with FHCA-V1 executing faster than FHCA-V2. For $T = 34$, the two variations achieved their best accuracy levels (83.05% and 82.4%) and had 67% and 80% lower cost than k-NN, respectively.

Shuttle dataset: This is an imbalanced (skewed) dataset. Approximately 80% of the data belongs to one class. Therefore the default accuracy is 80%. When classification tasks execute against such datasets, the main goal is to obtain a very high accuracy (e.g. over 99%). As shown in Table 5.4, this goal is successfully fulfilled by k-NN and CNN k-NN. Additionally, to the very high accuracy that it achieved, CNN k-NN executed extremely fast, since it scanned a very small condensing set (only 300 items). This happened because the CNN-rule managed to reduce the training data at the minimum level.

For the dataset in question, FHCA achieved an accuracy of 99.82% with almost the half the cost (53.23%) of k-NN, for $T = 30$. By setting $T = 25$, the was reduced by over 60%, but the accuracy fell to 99.19%. It is worth mentioning that FHCA achieved its best accuracy (99.84%) by setting $T = 37$ and had a cost reduction by 15%. The two variations did not achieve a reference-worthy accuracy. However, the proposed algorithm and its variations managed to classify with high accuracy the testing set items belonging to rare classes using their centroid-based part. There are many application domains where the correct prediction of rare classes is very critical (e.g. earthquake prediction, rare diseases, etc). In the shuttle dataset, there are two very rare classes both having only 17 items in the training set and 6 items in the testing set. For any T value, FHCA, FHCA-V1, and FHCA-V2 made 5 correct and 1 incorrect predictions.

Letter recognition dataset (noisy): For all noisy datasets, CNN k-NN and FHCA-V2 were affected by the addition of noise. The CNN-rule did not manage to drastically reduce the training (noisy) sets, and so, the classification cost gains were not significant. In contrast, the experimental results showed that FHCA and FHCA-V1 were not affected. In particular, for $T = 0.5$, the FHCA accuracy was 86.06% and its cost was 35.31% lower than k-NN. For $T = 0.9$, the accuracy was 91.06% and the cost was 16.95% lower than k-NN. FHCA achieved even better accuracy, but the cost savings were not significant. On the other hand, FHCA-V1 had an accuracy of 89.14% and 84.36%, for $T = 0.9$ and $T = 0.5$ respectively. The corresponding cost savings were 21.54% and 38.29%.

Pendigits dataset (noisy): On this dataset, FHCA reached the accuracy level of CNN k-NN at a 10% lower cost. Considering the additional high cost introduced by the construction of the condensing set of the CNN approach, the cost gains are actually much higher for FHCA. Moreover, FHCA reached the accuracy of k-NN with the same cost as CNN k-NN. Finally, for $T = 13$, FHCA achieved an accuracy of 91.71% and had only 38.75% of the k-NN cost. Similarly, FHCA-V1 achieved an accuracy of 93.31% and 88.65% by setting $T = 33$ and $T = 12$ respectively. The corresponding savings in classification cost were 33.26% and 70.77% respectively.

Landsat satellite dataset (noisy): The results are similar to the ones obtained on the pendigits (noisy). FHCA had higher accuracy than CNN k-NN with 15% less cost. For $T = 21$, FHCA-V1 had an accuracy of 86.55% with 63.17% of the cost. Finally, for $T = 12$, FHCA-V1 achieved an accuracy of 82.3% and only 36.08% of the k-NN cost.

Discussion

For the datasets (i) Letter recognition, (ii) Pendigits, (iii) Landsat satellite, and, (iv) Shuttle, the proposed algorithms seem to be slower than CNN k-NN, however they are model-free, since they do not need any speed-up model produced by costly preprocessing procedures. The calculation of the class centroids is quite simple and inexpensive and can be executed before each classification task to take into account the latest database changes. Furthermore, in the case of the magic gamma telescope dataset, FHCA reached the accuracy of CNN k-NN with the same classification cost. This is attributed to the noise that exists in this dataset.

As expected, Hart's CNN-rule was affected by the addition of noise. The preprocessing procedure of the CNN-rule could not significantly reduce the items of the datasets that contain noise. Thus, for these datasets, in addition to the cost introduced by the condensing set construction, the sequential search of the condensing set involved a relatively high classification cost. Contrary to CNN-rule, FHCA and FHCA-V1 were not significantly affected by the noise. The experimental results on the three noisy datasets showed that the latter approaches manage to reach and exceed the CNN k-NN accuracy at a lower classification cost. FHCA-V2 is also affected by the addition of noise. This is because the third classification criterion, which handles the cases where the new item lies within more than one class regions of influence, cannot make predictions with high accuracy.

Last but not least, it should be noted that contrary to CNN-rule, the adaptive schema offered by the proposed approach allows for the development of classifiers that reach the accuracy of the conventional k-NN classifier with significant savings in the classification cost.

5.2.4 Conclusions

In this section, fast, hybrid and model-free classifiers were proposed. Speed-up was achieved by combining the minimum distance and the k-NN classifiers. Initially, the fast centroid-based model of MDC attempts to classify the new incoming item. Upon failure, the new item is classified via the k-NN approach. Although the proposed approach is quite simple, it managed to speed-up the classification process. We consider that it can be useful in cases where data updates are frequent, thus, preprocessing of the training data for data reduction is prohibitive, or multidimensional index construction involving dimensionality reduction does not achieve acceptable classification accuracy.

Performance evaluation results demonstrated that significant classification cost reduction can be achieved, whereas, accuracy remains at high levels. In particular, the main classification algorithm (FHCA) met our expectations since it reached the accuracy level of the k-NN classifier and was not affected by addition of noise. The two proposed variations of FHCA can be used in applications where there is a need for a less accurate but very fast classification approach. The decision on which of the three variations should be used and which threshold value is the most appropriate one depends on the application domain. Namely, these decisions should be made by taking into consideration the most critical parameter, i.e., the trade-off between accuracy and classification cost.

The effectiveness of the centroid-based model that the proposed classifier uses in order to speed-up the classification procedure is depended on the data distribution in the multidimensional space. In particular, it can be affected by the shape and the size of the clusters that the items of each class form. In the next section, we address this issue by using more than one representative items for each class in the training data.

5.3 An adaptive fast hybrid method for *k*-NN classification

5.3.1 Motivation and contribution

The idea of combining the strategies of data abstraction and cluster based methods (see Section 2.2) with the goal of fast k-NN classification is behind the motivation of the work presented in this section. The contribution is the development of an adaptive, hybrid and cluster-based method for speeding-up the k-NN classifier.

More specifically, we develop a fast cluster-based preprocessing algorithm that builds a two level data structure and efficient classifiers that access either the first or the second level of the data structure and perform the classification task. The first level stores a number of cluster means (centroids) for each class. The second level stores the set of items belonging to each cluster. Therefore, a prototype abstraction algorithm and a cluster-based method are combined in a hybrid classification schema to achieve the desirable performance.

The rest of this section is organized as follows. Subsection 5.3.2 considers in detail the proposed classification method. Subsection 5.3.3 presents the experimental study based on seven datasets. The results are validated by the Wilcoxon signed ranks test. Subsection 5.3.4 concludes the work.

5.3.2 The proposed adaptive hybrid method

The proposed method [102, 90] consists of two main parts. The first part is a Two Level Data Structure (TLDS) built by a clustering preprocessing procedure. We call this procedure TLDS Construction Algorithm (TLDSCA). The second part is a Fast Hybrid Classifier (FHC) that accesses TLDS to perform classification. The following paragraphs present TLDSCA and two versions of the proposed classifier. In the end, some general comments about the proposed method are presented.

Two-Level Data Structure Construction Algorithm

TLDS is built in a way similar to the construction of the condensing set of RkM algorithm [95] (see Section 4.3). Thus, TLDS is built by a repeated execution of k-means clustering on the training data of each class. In particular, for each class, k-means builds a number of clusters. TLDS consists of a list of cluster centroids (i.e., the mean vectors of all clusters for all classes) together with their corresponding class label that we call the first level of TLDS. Each element of this list points to a list containing the items assigned to the specific cluster centroid. We refer to the collection of these lists of items as the second level of TLDS. We will be using the term prototypes for the cluster centroids of the first level of TLDS.

The number of clusters built is determined by the Data Reduction Factor (DRF), that is a user-predefined parameter. For each class c, the number of clusters nc is estimated by:

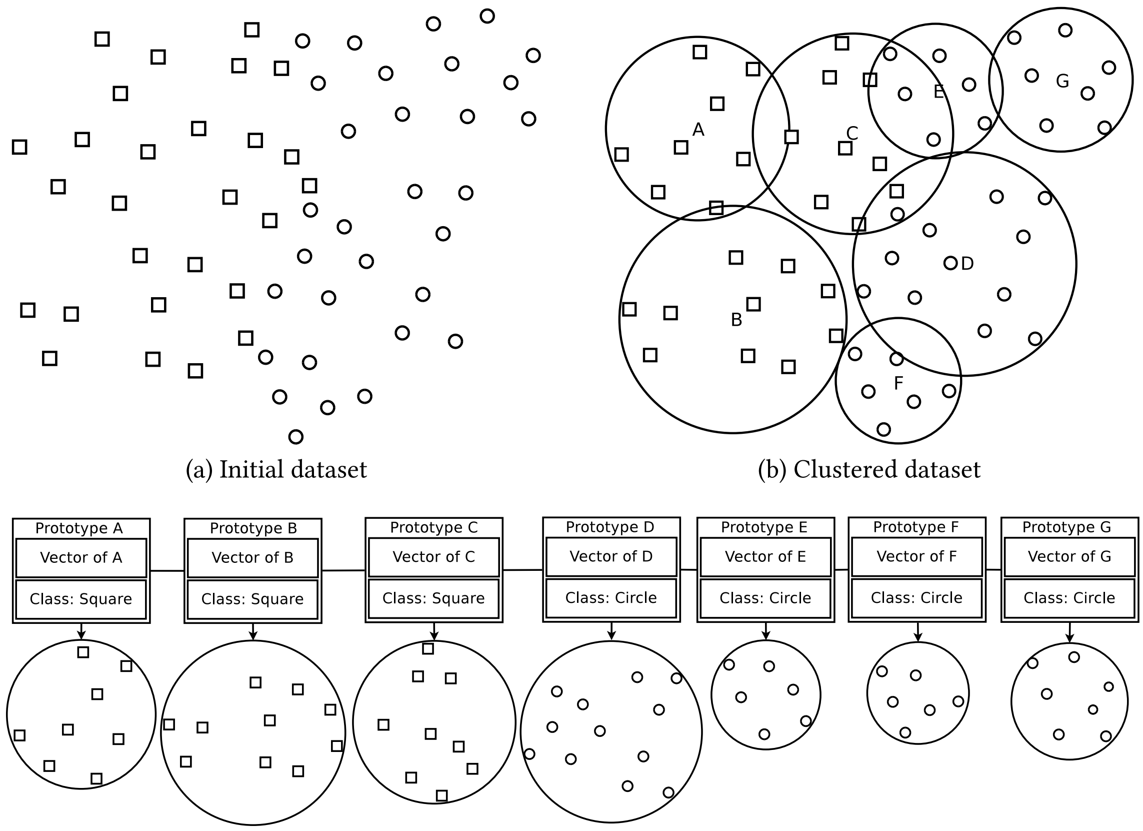
$$nc = \lceil \frac{|c|}{DRF} \rceil$$

where $|c|$ is the population of c. Therefore, DRF determines the length of TLDS. Figure 5.3 illustrates a two-dimensional example with two classes, square and circle. The initial training set contains 27 squares and 31 circles (Figure 5.3(a)). Thus, if $DRF = 10$, the classes square and circle will be represented by 3 and 4 prototypes, respectively (Figure 5.3(b)). The result of TLDSCA will be the TLDS depicted in Figure 5.3(c). Class square is represented by prototypes A–C, whereas class circle is represented by prototypes D–G.

TLDSCA is easy to implement (see Algorithm 18). It takes as input a training set and a DRF value and returns a TLDS. Initially, for each class c, it estimates the number of clusters that will be built (lines 3–4). The algorithm continues by calling the k-means function with parameters the set of items that belong to c (SC) and a list (M) of nc random initial means for k-means clustering (line 6) (see Section 2.4). Then, the resulting clusters are added in TLDS (lines 7–9).

TLDSCA is fast because it is based on k-means clustering. Of course, the computational cost depends on how fast the clusters are consolidated. In addition, DRF also influences the cost of the algorithm. Typically, the higher the DRF value, the fewer and larger the clusters created, and consequently, the lower cost involved. Of course, the quality of the clusters built by k-means clustering depends on the initial means used. We adopt a random initialization. However, the quality of the clusters can be improved by adopting a more efficient initialization method (see Section 2.4).

We should mention that the idea of creating multiple class representatives via clustering has also been proposed by Datta and Kibler in [31, 30]. Their goal was the representation of distant and disjoint groups formed by items of the same class and the construction of a classifier capable of managing symbolic (nominal) attributes.



(a) Initial dataset

(b) Clustered dataset

(c) Two-level data structure

Figure 5.3: k-means clustering on the items of each class ($DRF = 10$) and the resulted two-level data structure

Algorithm 18 TLDSCA

Input: TS, DRF, **Output**: $TLDS$

```
1: $TLDS \leftarrow$ empty list of records of the form $[class, prototype, list\_of\_items]$
2: for each class $c$ in $TS$ do
3:     $SC \leftarrow$ set of items $\in c$
4:     $nc \leftarrow \lceil \frac{|SC|}{DRF} \rceil$
5:     $M \leftarrow nc$ random items $\in SC$ {initial means for $k$-Means}
6:     $NewClusters \leftarrow K$-MEANS($SC$, $M$)
7:     for each $CL \in NewClusters$ do
8:         add in $TLDS$ element $[c, CL_{centroid}, CL_{items}]$
9:     end for
10: end for
11: return $TLDS$
```

Fast Hybrid Classifier I

The first version of the proposed classifier (FHC-I) works by accessing either the first or the second level of TLDS. It uses three input parameters that let the user define the desirable trade-off between accuracy and classification cost. The idea behind the algorithm is quite simple (see Algorithm 19). When a new item x arrives and has to be classified (line 1), FHC-I initially scans the first level of TLDS and retrieves the pk nearest prototypes to x (line 2). We call this procedure first level search. If the acceptance criterion introduced by the $npratio$ parameter is met, these prototypes, through a majority vote, determine the class where x belongs to (line 3–5). Upon failure, x is classified by searching for the k "real" nearest neighbours within the clusters dynamically formed by the union of clusters indexed by the pk nearest prototypes (lines 7–9). This procedure is called second level search. Obviously, the more the items classified without the need of the second level search, the lower is the classification cost involved. Possible ties during the majority class voting of either the first or the second level search are resolved using the single nearest neighbour rule.

FHC-I uses parameter $npratio$ to decide when to switch to a second level search. $npratio$ is a ratio that defines how many nearest prototypes should determine the majority class (the most common class among the pk nearest prototypes) in order to classify a new item (see lines 3–5). For example, suppose that the input parameters are set to be $k = 3$, $pk = 10$, and $npratio = 0.7$. Furthermore, suppose that a new item x has to be classified and a TLDS with 100 clusters is available. FHC-I, initially, examines the 10 nearest prototypes from the first level of TLDS. If seven or more of them belong to the majority class, then x is classified to this class. Otherwise, FHC-I proceeds to a second level search. The three "real" nearest neighbours are retrieved from the data subset formed by the union of clusters indexed by the ten (pk) nearest prototypes, and they determine the class of x. Even in the case of the second level search, FHC-I avoids searching 90 clusters. We note that if we choose $npratio = 0$, all incoming items are classified by a first level search. In effect, they are classified by the condensing set built by RkM algorithm [95].

Fast Hybrid Classifier II

The performance of FHC-I depends on the distribution of the training items with respect to the classes. If they are uniformly distributed, each class is represented by a similar number of prototypes in TLDS. Therefore, all unclassified items have the same probability to be classified

Algorithm 19 FHC-I

```
Input: $TLDS$, $pk$, $npratio$, $k$
1: for each unclassified item $x$ do
2:     $pkprototypes$ ← Find the $pk$ nearest to $x$ prototypes $\in$ {prototypes of $TLDS$}
3:     $SMC_1$ ← set of items $\in$ majority class $MC_1$ among $pkprototypes$
4:     if $\frac{|SMC_1|}{pk} \geq npratio$ then
5:         Classify $x$ to $MC_1$
6:     else
7:         $NNs$ ← Find the $k$ nearest neighbours to $x$ in the set formed by the union of the
           clusters indexed by the $pkprototypes$
8:         $MC_2$ ← Find the majority class among $NNs$
9:         Classify $x$ to $MC_2$
10:    end if
11: end for
```

by a second level search. In contrast, in cases of non-uniform distributions and since the value of $npratio$ is the same for all classes, the probability of performing a second level search depends on which is the majority class of the first level search. Items belonging to rare classes always lose during the voting of the first level search and are classified by a second level search.

The fast hybrid classifier II (FHC-II) attempts to better manage imbalanced datasets. It considers the size of the classes and tries to reduce "costly" second level searches. FHC-II estimates $npratio$ instead of using a pre-specified value for it. This is accomplished by using a range of $npratio$ values defined by parameters $npratio_{low}$ and $npratio_{high}$. The value of $npratio$ is dynamically adjusted to be between the particular range and depends on the majority class determined by the first level search.

Algorithmically, FHC-II is similar to FHC-I. However, for each class c, FHC-II counts how many training items $|c|$ belong to c (or how many prototypes are created for c) and notes the corresponding min and max values. For each class c, FHC-II computes $npratio \in [npratio_{low}, npratio_{high}]$ as follows:

$$npratio = norm \times (npratio_{high} - npratio_{low}) + npratio_{low}$$

where

$$norm = \frac{|c| - min}{max - min}$$

For each new item, the ratio of majority class MC_1 votes during first level search should be greater than the estimated $npratio$ in order to avoid a second level search (see lines 2–5 in Algorithm 19).

For instance, suppose that a training set contains three classes, A, B and C with 3000, 2000 and 1000 items respectively. Also, suppose that $npratio_{low} = 0.5$ and $npratio_{high} = 1$. If class A is voted to be the majority class during the first level search, then $npratio = 1$ (because $norm = 1$). Namely, all np nearest prototypes must vote the majority class in order to classify the new item without the need of a second level search. Similarly, if class B is the majority class of the first level search, $npratio = 0.75$ (because $norm = 0.5$). Finally, if class C is the majority class then $npratio = 0.5$ (because $norm = 0$). That is, the value of $npratio$ is adjusted for each class in the range $[npratio_{low}, npratio_{high}]$ depending on the size of the class, i.e., the smaller the class the lower the $npratio$ and vice versa.

When correct prediction of "weak" (or rare) classes is critical, FHC-I should be used instead of FHC-II. FHC-II should be adopted when correct predictions for all classes have the same significance.

Discussion

Considering the proposed classifier, it is obvious that a new item that lies in a close-border area, is classified by a second level search. On the other hand, an item that lies in the "internal" area of a class, is classified by first level search. Thus, FHC is neither a cluster-based method nor a prototype abstraction algorithm, since it dynamically decides on how to classify a new item. When it classifies via a first level search it behaves like a prototype abstraction method. On the other hand, when it resorts to a second level search it behaves like a cluster-based method that uses a dynamically-formed subset of the initial training set. Therefore, the method is hybrid. Of course, contrary to prototype selection and abstraction algorithms and like cluster-based methods and indexing methods, the proposed method does not reduce storage requirements.

Concerning the parameters, pk and $npratio$ (or $npratio_{low}$ and $npratio_{high}$) should be determined by taking into account the DRF value that was used for TLDS construction. If accuracy is more critical than cost and a TLDS with few and large clusters is available, pk and $npratio$ should have high values. If cost is more critical and a TLDS with many and small clusters is available, low pk and $npratio$ values are recommended. Considering DRF: low DRF values are recommended for building accurate classifiers with high cost savings and high DRF values for building fast classifiers without significant loss of accuracy. If our needs are not specified at the time that TLDSCA is executed, an intermediate DRF value is the most appropriate. In this case, the trade-off can be determined afterwards by adjusting pk and $pkratio$.

When FHC performs a second level search, it accesses a subset of the initial training set formed by the union of the pk clusters. Since each cluster contains items of a specific class, this subset does not contain items of irrelevant classes (it does not contain outliers of classes which are not represented by the pk prototypes) and, thus, we claim that accuracy is not affected as much by these outliers. Taking into account this property, FHC may be more accurate than the conventional-k-NN classifier (the one that uses the non-edited training set) without the need of editing for noise removal. Of course, noise removal can increase the cluster quality and the overall performance.

5.3.3 Performance evaluation

Experimental setup

The proposed method was coded in C and was evaluated using seven datasets distributed by the KEEL repository[3] [6] and summarized in Table 5.5. All datasets were used without normalization. The Euclidean distance was adopted as the distance metric. We compared the performance of our method to six methods: three condensing and two prototype abstraction algorithms as well as one cluster based method. We used CNN-rule, IB2, PSC, RSP3 and Hwang and Cho method (HCM). We selected the particular methods because: (i) CNN-rule, IB2 and RSP3 are popular and are usually used in many papers for comparison purposes, (ii) we consider TLDSCA to be a fast algorithm, and, hence, we wanted to compare it to IB2 and RHC that have been proven to be fast algorithms, and, (iii) the proposed method, PSC, HCM and RHC are based on k-means clustering, hence, an experiential comparison between them was desirable. The MGT, LS and TXR datasets are distributed sorted on the class label. This affects the data order-dependent methods (i.e. CNN-rule and IB2). We randomized these datasets.

We compare the methods by reporting three average measurements obtained via five-fold cross-validation for each one: (i) accuracy, (ii) classification cost, and, (iii) preprocessing cost. Costs were estimated by counting distance computations. We used the five already constructed pairs of training/testing sets distributed by the KEEL repository. Moreover, we wanted to

[3]http://sci2s.ugr.es/keel/datasets.php

Table 5.5: Datasets description

Dataset	Size	Attributes	Classes
Letter Image Recognition (LIR)	20000	16	26
Magic Gamma Telescope (MGT)	19020	10	2
Pen-Digits (PD)	10992	16	10
Landsat Satellite (LS)	6435	36	6
Shuttle (SH)	58000	9	7
Texture (TXR)	5500	40	11
Phoneme (PH)	5404	5	2

evaluate all methods on noise-free data. Therefore, we ran all experiments twice, one on the non-edited and one on the edited training sets. For editing purposes, we used ENN-rule by setting $k = 3$ (according to [131, 49, 84], $k = 3$ is a good value).

All methods involve a k parameter during the classification step: The condensing and prototype abstraction algorithms execute the k-NN classifier on condensing set. Similarly, when FHC performs a second level search, it retrieves and examines the k nearest neighbours. HCM applies the k-NN classifier on the reference set (see Section 2.2). For all methods and datasets, we used $k = 1$.

The condensing and prototype abstraction algorithms involve parameter k since they apply the k-NN classifier on the condensing set during the classification step. Similarly, when FHC performs a second level search, it retrieves and examines k nearest neighbours. HCM applies the k-NN classifier on the reference set. For all methods and datasets, we used $k = 1$.

In addition, our method provides three extra parameters: DRF, pk, and $npratio$. We used the following values: (i) $DRF = 2^i, i = 1, 2, \dots 8$ (for the SH dataset, $i = 3, 4, \dots 8$), (ii) $pk = 2, 5, 7, 10, 12, \dots, 27, 30$, and, (iii) for FHC-I, $npratio = 0.5, 0.7, 1$. Therefore, we built $8 \times 12 \times 3 = 288$ FHC-I classifiers. To facilitate the presentation, we plot the most accurate FHCs for each reported classification cost (see the performance diagrams in Figures 5.4–5.10). In effect, the performance of a classifier was omitted if it achieved lower accuracy and involved higher classification cost than another classifier that achieved equal or higher accuracy and involved lower classification cost. Our purpose was not to fine tune the parameters for each dataset, but to understand how each parameter influences classifier construction and performance. In real life applications, the parameters should be chosen by taking into consideration the significance of accuracy and classification cost as well as the dataset used.

HCM also uses three parameters: C is the number of clusters, L is the number of adjacent clusters, D is the distance threshold used to define the core and peripheral items (see Section 2.2 for details). We set $C = \lfloor\sqrt{\frac{n}{2^i}}\rfloor$, $i = 1, \ldots, 7$, where n is the number of items. Thus, for each dataset, we built 8 classifiers. The first classifier (for $i = 1$) is based on the rule of thumb that defines $C = \lfloor\sqrt{\frac{n}{2}}\rfloor$ [78]. We decided to build classifiers that use small C values based on the observation that Hwang and Cho defined $C = 10$ for a training set with 60919 items. Also, we set $L = \lfloor\sqrt{C}\rfloor$ as Hwang and Cho did in their experiments. Moreover, following the approach of Hwang and Cho, we considered as peripheral items, those whose distance from the cluster centroid was greater than the double average distance among the items of each cluster (i.e., $D = 2$).

Another issue that needs attention is the number of clusters that PSC uses. Like Lopez et al. did in their experiments [86], we ran experiments by building $c = r \times j$, $j = 2, 4, \cdots, 10$, clusters, where r is the number of classes. Therefore, we constructed five PSC based classifiers for each dataset.

Since only a first level search can be used to classify new items, we included its performance in the comparisons. We call this method first level search classifier (FLSC). It carries out the whole task when $npratio = 0$. In effect, FLSC is identical to RkM (see Section 4.3).

Pre-processing performance

Tables 5.6–5.8 present the preprocessing cost measurements in millions of distance computations. Each cell has two values. The first value (ned) corresponds to preprocessing costs estimated on the non-edited data whereas the second value (ed) on the edited data. The first table row shows the preprocessing cost needed for the execution of the editing procedure of ENN-rule. Of course, the measurements on the edited data do not contain the cost of editing.

RSP3 is the most time-consuming method, since it must retrieve the pair of the farthest points in each subset. CNN-rule is faster than RSP3. IB2 and RHC seem to be the fastest approaches. The preprocessing cost measurements of all other methods depend on the parameter used and the size and distribution of the data in the multidimensional space. Considering the measurements of TLDSCA, we can conclude that its preprocessing performance is satisfactory. We note that we have adopted the full cluster-consolidation as well as the random initialization of the means. Thus, TLDSCA could have become even faster had we used more efficient consolidation and initialization criteria.

Table 5.6: Experimental results: Preprocessing Cost of DRTs (millions of distance computations)

Method		LIR	MGT	PD	LS	SH	TXR	PH
ENN	ned	127.99	115.76	38.65	13.25	1076.46	9.68	9.35
CNN	or.	163.03	281.49	11.75	17.99	45.30	5.65	13.45
	ed	112.20	68.61	9.25	6.49	26.02	3.90	5.57
IB2	or.	23.37	34.61	1.78	2.22	8.26	0.84	1.96
	ed	18.35	8.48	1.51	0.99	6.35	0.72	0.86
RSP3	or.	326.52	511.67	86.66	37.70	17410.12	27.63	20.31
	ed	300.51	318.82	85.16	30.64	15652.75	27.04	15.67
RHC	or.	41.84	4.08	2.88	1.69	16.83	3.63	0.66
	ed	31.05	2.83	2.83	1.73	22.41	3.00	0.47
PSC j=2	or.	66.32	23.95	6.52	2.96	127.20	3.15	1.08
	ed	55.13	11.44	6.73	2.86	107.47	3.35	0.68
PSC j=4	or.	110.06	17.21	15.93	5.85	54.07	7.90	0.94
	ed	94.76	10.15	17.57	4.83	52.46	10.33	1.04
PSC j=6	or.	129.16	22.68	28.48	8.41	148.35	10.71	2.08
	ed	127.84	11.28	27.65	6.79	176.21	9.60	1.89
PSC j=8	or.	165.32	27.09	35.23	10.11	222.77	14.49	2.76
	ed	138.41	12.42	32.33	9.97	189.71	11.10	2.18
PSC j=10	or.	169.92	33.47	36.97	10.50	252.61	16.76	3.12
	ed	178.45	21.75	33.74	11.82	213.61	15.78	3.15

In typical data mining tasks, preprocessing is executed only once. Hence, these measurements may not be so significant in real life applications. However, preprocessing cost is a comparison criterion and its measurements should be evaluated taking into account the performance that the corresponding classifiers achieve in terms of accuracy and classification cost.

Classification performance

Each classifier was executed twice, once on the non-edited and once on the edited data. Figures 5.4–5.10 illustrate the performance measurements. Each figure presents two diagrams, one for the non-edited and one for the edited data. For each classifier, the diagrams report the measurements of classification costs on the x-axis (in terms of millions or thousands distance computations) and the corresponding accuracy values on the y-axis. The closer to the "upper-left" corner of the diagram a classifier's point lies, the higher is the performance achieved.

Table 5.7: Experimental results: Preprocessing Cost of HCM (millions of distance computations)

Method		LIR	MGT	PD	LS	SH	TXR	PH
HCM i=1	ned	88.88	111.22	28.80	12.52	744.82	8.10	9.87
	ed	74.60	58.85	26.88	9.13	867.40	7.92	5.58
HCM i=2	ned	88.12	131.45	17.57	9.84	490.57	5.58	5.15
	ed	80.87	49.53	17.53	7.81	557.07	5.40	4.15
HCM i=3	ned	63.66	73.69	11.27	6.34	399.23	4.80	3.70
	ed	55.31	33.39	12.06	6.32	366.82	3.46	2.65
HCM i=4	ned	47.53	52.88	7.61	3.75	334.99	4.27	1.55
	ed	30.23	27.84	7.12	3.32	254.83	4.58	1.78
HCM i=5	ned	26.35	27.69	5.97	3.39	105.13	3.41	1.33
	ed	31.45	18.16	5.21	3.06	146.12	2.68	1.03
HCM i=6	ned	18.98	19.51	3.32	2.12	100.66	1.67	0.70
	ed	25.96	12.65	2.99	1.37	106.02	1.82	0.77
HCM i=7	ned	10.89	7.50	1.70	0.94	34.78	0.52	0.74
	ed	9.93	5.16	1.81	0.67	37.18	0.54	0.48

Table 5.8: Experimental results: Preprocessing Cost of TLDSCA (millions of distance computations)

Method		LIR	MGT	PD	LS	SH	TXR	PH
TLDSCA i=1	ned	19.62	404.23	18.38	10.56	-	3.50	34.69
	ed	18.48	258.18	17.78	8.26		3.34	26.27
TLDSCA i=2	ned	15.79	291.78	14.29	9.18	-	2.74	24.65
	ed	14.63	221.87	13.78	7.67		2.60	23.51
TLDSCA i=3	ned	11.58	252.06	10.60	6.87	3898.83	1.99	17.50
	ed	10.13	177.57	10.78	5.92	4267.59	1.98	18.00
TLDSCA i=4	ned	7.74	192.26	7.28	4.45	2879.39	1.32	12.36
	ed	7.01	145.16	7.21	4.51	3027.05	1.38	9.75
TLDSCA i=5	ned	4.80	159.82	4.54	3.75	1983.80	0.94	7.92
	ed	4.25	92.64	5.06	3.07	2115.25	0.99	6.47
TLDSCA i=6	ned	2.69	105.35	2.95	1.82	1537.64	0.58	7.00
	ed	2.37	64.63	3.29	1.79	1585.51	0.58	4.29
TLDSCA i=7	ned	1.23	54.04	1.68	1.15	855.87	0.28	3.27
	ed	1.15	46.55	1.60	0.95	847.11	0.29	2.95
TLDSCA i=8	ned	0.60	34.55	0.69	0.47	551.25	0.08	1.21
	ed	0.58	21.79	0.61	0.42	535.73	0.08	1.55

Since we want to clearly indicate classifiers of high performance, the diagrams present only a subset of performance points (points of some classifiers are omitted because they are out of the diagram range). To facilitate the presentation, for the parametric methods (FHC-I, FLSC, HCM and PSC), the diagrams present the most accurate classifiers for each reported classification cost. In addition, we do not show the parameter values used to build the corresponding classifiers[4].

Table 5.9 shows accuracies and classification costs for the conventional 1-NN classifiers, i.e., classifiers that use the non-edited data (1-NN (ned)) or the edited data (1-NN (ed)). Although editing is used to improve accuracy, ENN-rule achieves that only in the case of the MGT dataset. This happens because MGT contains high levels of noise. Although, the LS and PH datasets also contain some noisy items, ENN does not improve accuracy. The other datasets are almost noise-free.

Table 5.9: Experimental results of Conventional 1-NN: Accuracy (Acc (%)) and Computational Cost (CC (millions of distance computations))

Method		**LIR**	**MGT**	**PD**	**LS**	**SH**	**TXR**	**PH**
1-NN (ned)	Acc	95.83	78.14	99.35	90.60	99.82	99.02	90.10
	CC	64.00	57.88	19.34	6.63	538.24	4.84	4.67
1-NN (ed)	Acc	94.98	80.44	99.30	90.29	99.79	98.64	88.14
	CC	61.23	46.26	19.21	6.02	537.24	4.78	4.14

Almost in all cases, FHC-I achieves high performance (see Figures 5.4–5.10). In the cases of the LIR, MGT, PD, LS and PH datasets, it is more accurate than 1-NN. This happens because when FHC-I performs a second level search, it accesses a training subset that does not contain items of irrelevant classes. With the exception of the SH dataset, FHC-I achieves better performance than all DRTs. For the SH dataset, FHC-I can achieve higher accuracy than DRTs, but at a higher classification cost. Although FHC-I is more accurate than HCM on all datasets, in the cases of the LIR and SH datasets the latter may be preferable when very fast classifiers are required. Of course, FHC-I performs better than FLSC. However, the latter achieves noteworthy performance that is comparable to the other speed-up methods.

All diagrams in Figures 5.4–5.10 show that when the speed-up methods are executed over edited data, they are faster than when they are executed over non-edited data. Nevertheless,

[4]Tables with complete parameter values and performance measurements are available on url: `http://users.uom.gr/~stoug/IGPL_experiments.zip`

in some cases, either the classification cost gains are not very high or accuracy is significantly reduced. In the case of the noisy MGT dataset, editing is necessary for all methods.

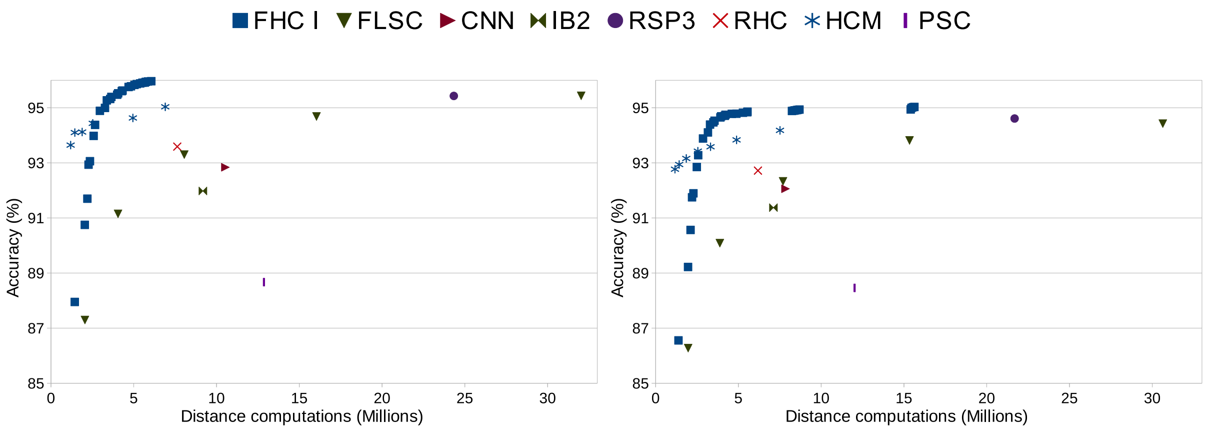
A final comment about the preprocessing and classification results is that the proposed method can perform comparable to or better than the other methods. The user can adapt the method to the application requirements by appropriately adjusting its parameters. We conclude that the proposed method can be adjusted to achieve high accuracy with gains in classification cost or to reduce the classification cost at a minimum level with slightly lower accuracy.

Non-parametric statistical test

Our experimental study is complemented with the results of a non-parametric statistical test. We ran the Wilcoxon signed ranks test [32] in order to validate the results presented in the previous subsections. The Wilcoxon test compares the speed-up methods in pairs taking into account their performance on each dataset. The test was run four times. Once for each comparison criterion, accuracy (ACC), classification cost (CC), preprocessing cost (PC) and once on the measurements of the overall classification performance. By following the idea presented in [48, 47], the measurements of the overall performance were estimated by averaging the normalized to the interval $[0, 1]$ measurements of the three aforementioned comparison criteria, thus, assuming that they all have the same significance. Here, we should mention that since low values for costs are desirable, we used the values $1 - normalized(CC)$ and $1 - normalized(PC)$ in the place of the normalized values for CC and PC, respectively.

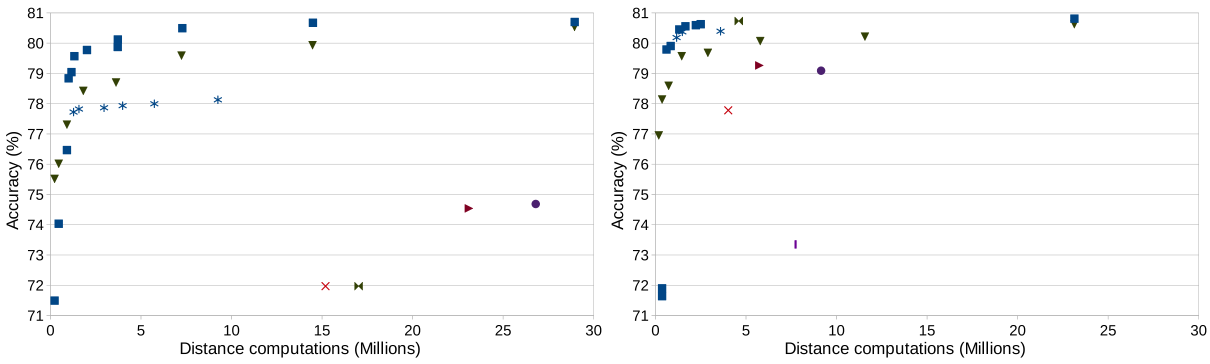
We ran the tests twice, once on the results obtained from the non-edited data and once from the edited data. Of course, we could not test all variations of the parametric classifiers (FHC-I, FLSC, PSC, HCM). Therefore, for each one of these methods, we used the same fixed parameter values for all datasets. The fixed values were: (i) for FHC-I: $DRF = 32$, $npratio : 1$ and $pk = 5$; (ii) for FLSC: $DRF = 16$ and $pk = 1$; (iii) for PSC: $j = 10$; (iv) for HCM: $i = 4$. We should mention that these values were not obtained via tuning. Therefore, they do not correspond to the classifiers with the highest accuracy or the lowest cost performance. In effect, these parameter values constitute a typical setting for each method (default values) and can be thought of as a starting point for tuning the parameters.

Tables 5.10 and 5.11 present the results of the tests for the non-edited and edited datasets respectively. Columns "w/l/t" list the number of wins/losses/ties for each pair. Columns "Wilc."



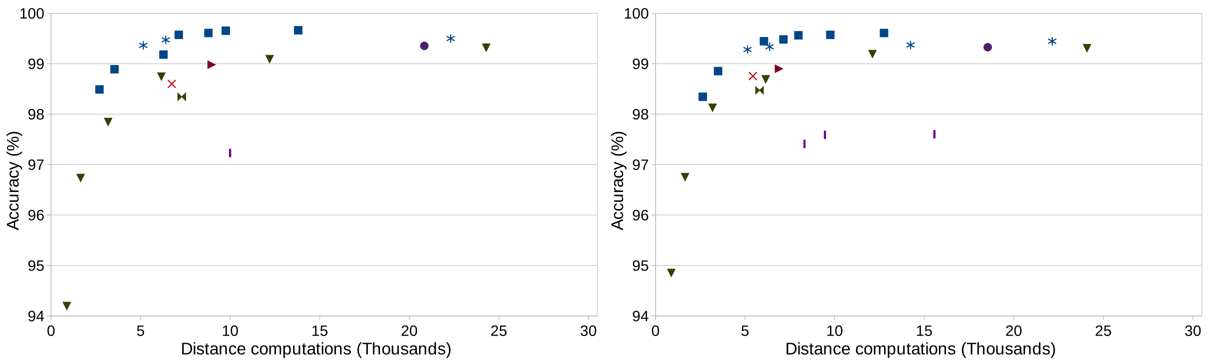
(a) Non-edited data (b) Edited data

Figure 5.4: LIR (Accuracy and Classification Cost)



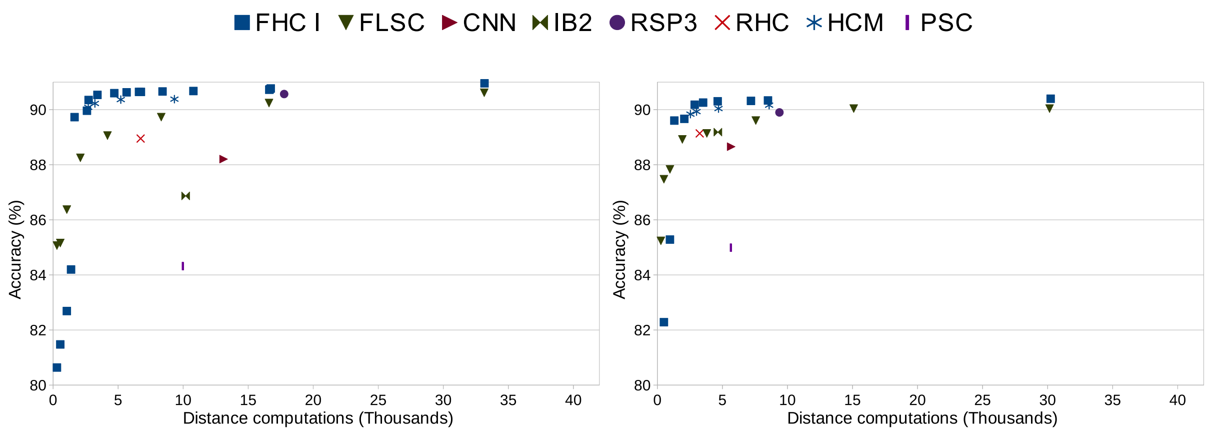
(a) Non-edited data (b) Edited data

Figure 5.5: MGT (Accuracy and Classification Cost)



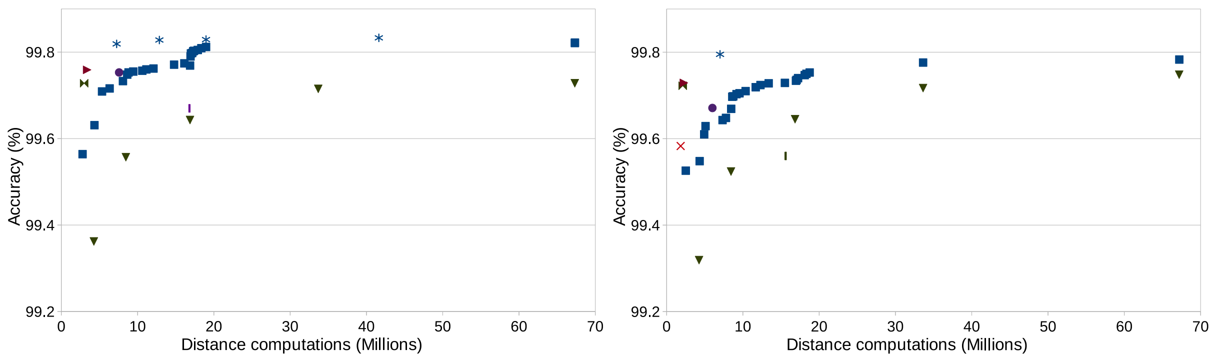
(a) Non-edited data (b) Edited data

Figure 5.6: PD (Accuracy and Classification Cost)



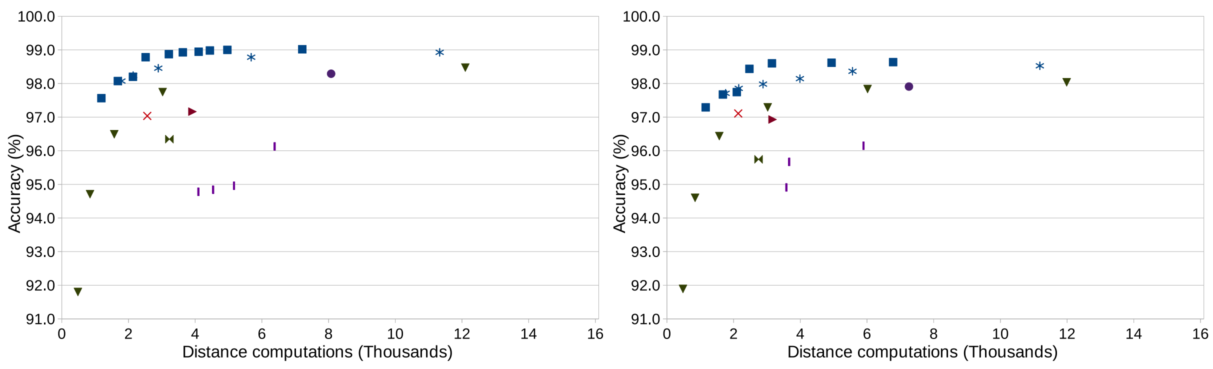
(a) Non-edited data (b) Edited data

Figure 5.7: LS (Accuracy and Classification Cost)



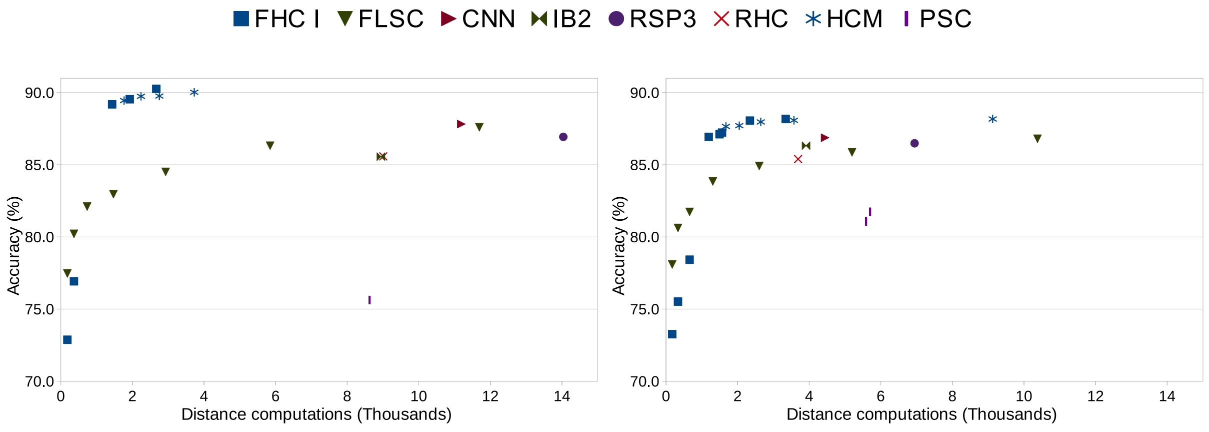
(a) Non-edited data (b) Edited data

Figure 5.8: SH (Accuracy and Classification Cost)



(a) Non-edited data (b) Edited data

Figure 5.9: TXR (Accuracy and Classification Cost)



(a) Non-edited data

(b) Edited data

Figure 5.10: PH (Accuracy and Classification Cost)

list the significance level. When that measure is lower than 0.05 (values in bold in Tables 5.10 and 5.11), one can claim that the difference between the two methods is statistically significant. Note that $Wilc. = 0.05$ is a very strict threshold. The tests confirm that FHC-I with the typical parameter values is an accurate method. In many cases, although FHC-I has more wins, the test does not confirm statistically significant difference. For example, concerning the overall performance on the non-edited data, FHC-I has 6 wins against CNN, IB2, RHC and PSC, and 5 wins against HCM. However, the dominance is not statistically validated. This happens because we have adopted a very strict threshold (i.e., $Wilc = 0.05$) for the Wilcoxon level and, probably, we used a relatively small sample of datasets.

FHC-II performance

Four of the eight datasets are imbalanced. The LS dataset contains six classes with 626, 703, 707, 1358, 1508, 1533 items respectively. The MGT dataset has two classes with 12332 and 6688 items. Similarly, the SH dataset has seven classes with 45589, 8903, 3267, 171, 49, 13, 10 items and the PH dataset has two classes with 3818 and 1586 items. FHC-I does not manage fairly the items of all classes. FHC-II reduces the probability of second level searches for the "weak" classes. This leads to even faster classifiers.

We ran FHC-II twelve times for each dataset using the following settings: (i) $pk = 15, 30$, (ii) $DRF = 16, 32$ and (iii) $(npratio_{low}, npratio_{high})$ values: $(0.7, 1)$, $(0.5, 1)$, $(0.3, 0.7)$. These

Table 5.10: Results of the Wilcoxon signed ranks test on the measurements obtained from the non-edited data

Methods	**ACC**		**CC**		**PC**		**Overall**	
	w/l/t	**Wilc.**	**w/l/t**	**Wilc.**	**w/l/t**	**Wilc.**	**w/l/t**	**Wilc.**
FHC-I vs CNN	7/0/0	**0.018**	6/1/0	0.176	6/1/0	0.237	6/1/0	0.176
FHC-I vs IB2	7/0/0	**0.018**	6/1/0	0.176	1/6/0	0.128	6/1/0	0.176
FHC-I vs RSP3	5/2/0	0.176	6/1/0	**0.043**	7/0/0	**0.018**	7/0/0	**0.018**
FHC-I vs RHC	7/0/0	**0.018**	6/1/0	0.237	2/5/0	0.310	6/1/0	0.176
FHC-I vs PSC	7/0/0	**0.018**	6/1/0	0.128	4/3/0	0.866	6/1/0	0.091
FHC-I vs HCM	5/2/0	0.063	6/1/0	**0.043**	4/3/0	0.612	5/2/0	0.237
FLSC vs CNN	4/3/0	0.866	6/1/0	0.237	6/1/0	0.237	6/1/0	0.237
FLSC vs IB2	4/3/0	0.310	6/1/0	0.237	1/6/0	0.128	6/1/0	0.237
FLSC vs RSP3	1/6/0	0.128	6/1/0	0.237	7/0/0	**0.018**	7/0/0	**0.018**
FLSC vs RHC	5/2/0	0.499	6/1/0	0.237	2/5/0	0.176	6/1/0	0.237
FLSC vs PSC	7/0/0	**0.018**	5/2/0	0.398	4/3/0	1.000	6/1/0	0.237
FLSC vs HCM	0/7/0	**0.018**	3/4/0	0.176	3/4/0	0.398	0/7/0	**0.018**
FHC-I vs FLSC	7/0/0	**0.018**	7/0/0	**0.018**	7/0/0	**0.018**	7/0/0	**0.018**

Table 5.11: Results of the Wilcoxon signed ranks test on the measurements obtained from the edited data

Methods	**ACC**		**CC**		**PC**		**Overall**	
	w/l/t	**Wilc.**	**w/l/t**	**Wilc.**	**w/l/t**	**Wilc.**	**w/l/t**	**Wilc.**
FHC-I vs CNN	6/0/1	**0.028**	6/1/0	0.237	4/3/0	0.866	6/1/0	0.237
FHC-I vs IB2	6/1/0	**0.043**	6/1/0	0.237	1/6/0	0.128	6/1/0	0.237
FHC-I vs RSP3	6/1/0	0.128	6/1/0	0.128	7/0/0	**0.018**	7/0/0	**0.018**
FHC-I vs RHC	7/0/0	**0.018**	6/1/0	0.237	2/5/0	0.237	6/1/0	0.237
FHC-I vs PSC	7/0/0	**0.018**	6/1/0	0.237	4/3/0	0.866	6/1/0	0.128
FHC-I vs HCM	4/3/0	0.237	6/1/0	**0.028**	4/3/0	0.612	5/2/0	0.398
FLSC vs CNN	4/3/0	1.000	6/1/0	0.237	4/3/0	0.735	6/1/0	0.237
FLSC vs IB2	2/5/0	0.499	6/1/0	0.237	1/6/0	0.128	6/1/0	0.237
FLSC vs RSP3	2/5/0	0.091	6/1/0	0.237	7/0/0	**0.018**	6/1/0	**0.028**
FLSC vs RHC	5/2/0	0.735	6/1/0	0.237	2/5/0	0.176	5/2/0	0.310
FLSC vs PSC	7/0/0	**0.018**	5/2/0	0.612	4/3/0	0.866	6/1/0	0.237
FLSC vs HCM	0/7/0	**0.018**	3/4/0	0.176	2/5/0	0.310	0/7/0	**0.018**
FHC-I vs FLSC	7/0/0	**0.018**	7/0/0	**0.018**	7/0/0	**0.018**	7/0/0	**0.018**

methods were compared to four FHC-I methods built using the same pk and DRF values and $npratio = 1$. Figures 5.11–5.14 present the results. Each FHC-II method is denoted with the following sequence: $DRF, pk, npratio_{low}, npratio_{high}$ in the figure's legend. Similarly, FHC-I is noted with $DRF, pk, npratio$.

In the case of the noisy MGT dataset, FHC-II improves both accuracy and classification cost measurements. In the case of the LS dataset, all FHC-II classifiers built using the $(0.3, 0.7)$ range of $npratio$ values as well as the one that uses settings $DRF = 16, pk = 15, npratio_{low} = 0.5$, $npratio_{high} = 1$ are ineffective. They reduce costs, but they also reduce accuracy. All other FHC-II classifiers execute faster than the FHC-I classifiers without loss of accuracy. In the cases of the SH and PH datasets, FHC-I may be preferable to FHC-II. The latter executes slightly faster than FHC-I. However this speed-up affects the accuracy.

5.3.4 Conclusions

This section proposed an adaptive hybrid method for fast k-NN classification. The method involves the construction of a two level data structure and classifiers that make predictions using either the first or the second level of this structure. Actually, the method combines the idea of data reduction with that of cluster-based methods in a hybrid schema. The method lets the user determine the trade-off between accuracy and classification cost. Therefore, it can be used either to improve accuracy at a lower cost, or to reduce cost at a minimum level without sacrificing accuracy. Experiments showed that cost improvements may be achieved, with the accuracy remaining high and comparable to that of the conventional k-NN.

5.4 Hybrid *k*-NN classification based on homogeneous clusters

5.4.1 Motivation and contribution

In Section 5.3, we demonstrated that the ideas of data reduction and cluster-based methods can be combined in a hybrid classification method to achieve the desirable performance. In particular, we proposed a pre-processing algorithm to construct a data structure and fast algorithms to classify new items by accessing this structure. The main disadvantage of our method was

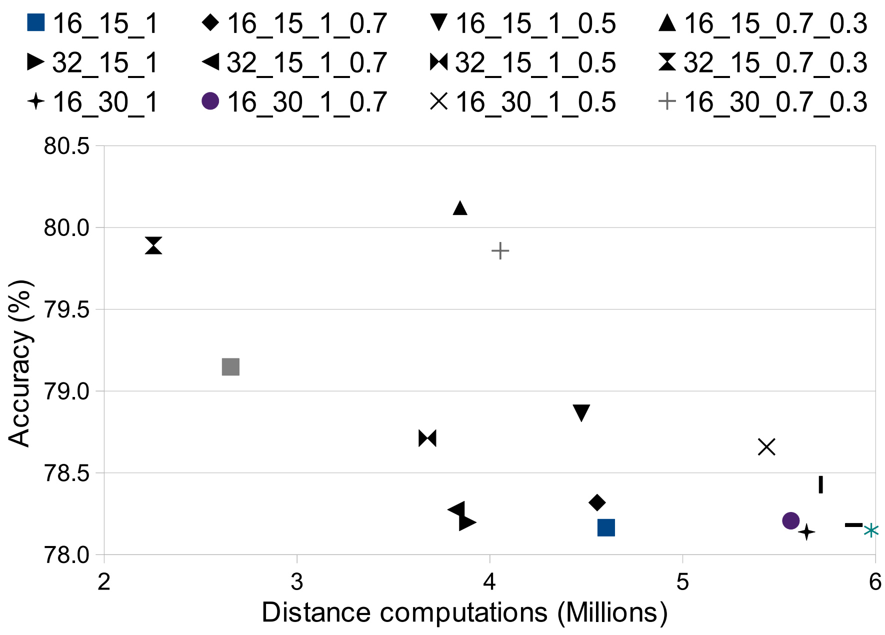


Figure 5.11: MGT:FHC-I vs FHC-II)

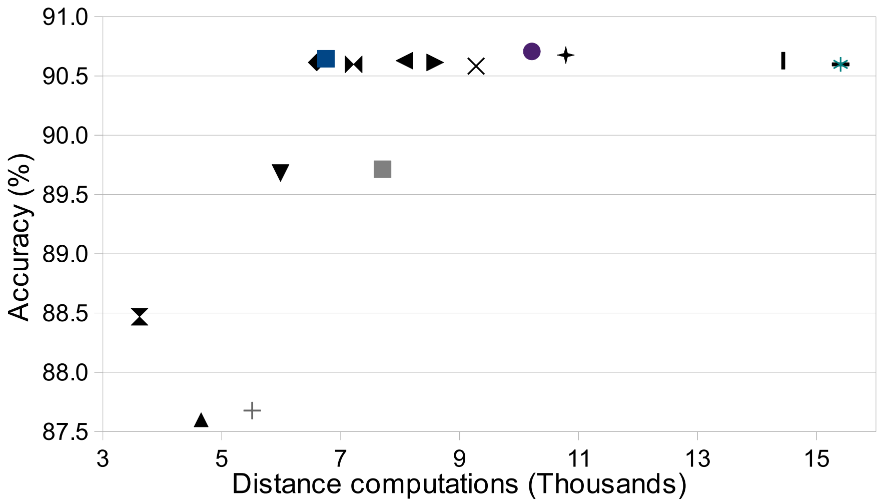


Figure 5.12: LS:FHC-I vs FHC-II)

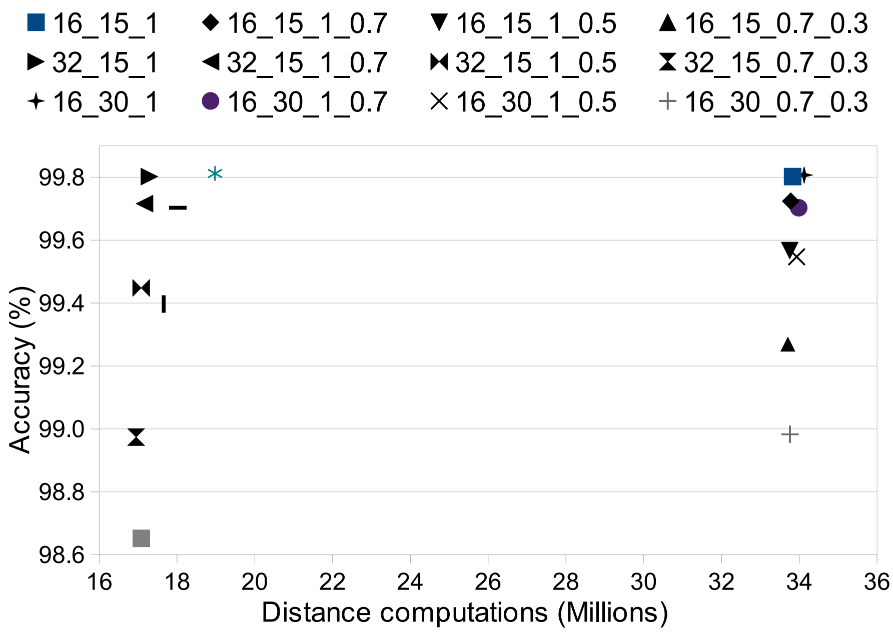


Figure 5.13: SH:FHC-I vs FHC-II)

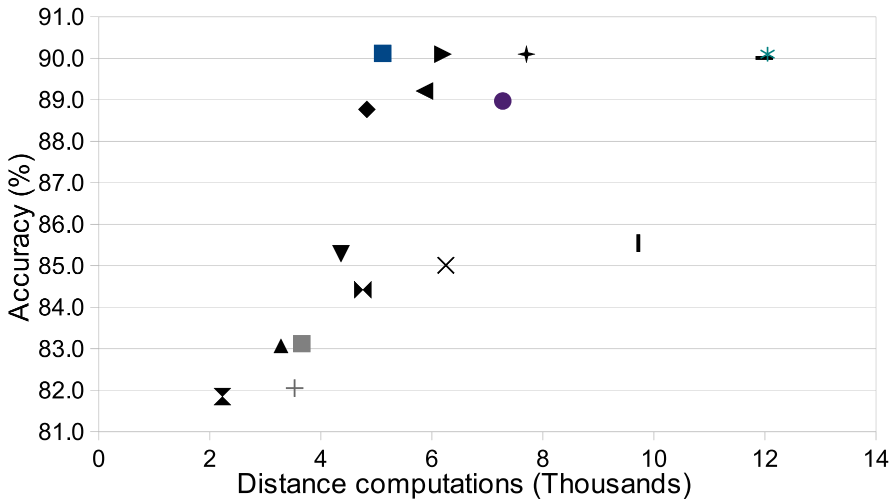


Figure 5.14: PH:FHC-I vs FHC-II)

that the algorithms were parametric and required a trial-and-error procedure to adjust their input parameters.

Here, we extend this idea. More specifically, the motivation is the development of a fast and non-parametric classification method for large and high dimensional data. The proposed method follows the concept of forming homogeneous clusters (clusters that contain items of a particular class only) presented in Chapter 3. The contribution of the work presented in this section is: (i) the development of an efficient and non-parametric classification method that combines two different speed-up strategies, namely, data reduction and cluster-based methods, (ii) the proposal of a variation of our method that is applied on data stored in condensing sets and is able to further improve the performance of DRTs. The main goal of the variation is extra fast classification.

The rest of this section is organized as follows. Subsections 5.4.2 and 5.4.3 consider in detail the proposed classification method and its variation, respectively. Subsection 5.4.4 presents the experimental evaluation and the results of the Wilcoxon signed ranks test, and, finally, Subsection 5.4.5 concludes the section.

5.4.2 The proposed SUDS classification method

Like FHC method, The proposed classification method includes two major stages: (i) pre-processing, which is applied on the training items in order to construct a two-level Speed-up Data Structure (SUDS), and, (ii) classification, which uses SUDS and applies the proposed hybrid classifier. In this subsection, we present the pre-processing algorithm as well as the hybrid classifier.

Speed-Up Data Structure Construction Algorithm

The pre-processing algorithm builds SUDS by finding homogeneous clusters in the training data. A cluster is homogeneous if it contains items of a specific class only. The SUDS Construction Algorithm (SUDSCA) repetitively executes the k-means clustering algorithm until all of the identified clusters become homogeneous. SUDS is a two-level data structure. Its first level is a list of means (centroids or representatives) of the identified homogeneous clusters. Each one represents a data area of a specific class and indexes the "real" cluster items that are in the second level of SUDS. Figure 5.15 shows how SUDS is constructed and Algorithm 20 summarizes the steps of the corresponding algorithm.

SUDSCA follows the strategy of RHC (see Section 3.2.2). Consequently, the first step of SUDSCA is to find the mean items of each class in the training set by averaging its items (Figure 5.15(b)). Then, it executes the k-means clustering using these class means as initial means. Thus, for a dataset with M classes, SUDSCA initially identifies M clusters (Figure 5.15(c)). SUDSCA continues by analyzing the M clusters. If a cluster is homogeneous, it is added to SUDS. The cluster mean is added to the first level of SUDS as representative of the specific class and indexes the cluster items that are added to the second level. On the other hand, for each non-homogeneous cluster X, k-means is executed on its items and identifies as many clusters as the number of distinct classes in X following the aforementioned procedure(Figure 5.15(d)). The repetitive execution of k-means terminates when all constructed clusters are homogeneous. Following this algorithm, SUDSCA constructs few large clusters for internal class data areas, and many small clusters for close-class-border data areas.

Like all DRTs presented in Chapter 3, SUDSCA can be easily implemented using a simple queue data structure that stores the unprocessed clusters. Initially, the whole training set (TS) constitutes an unprocessed cluster and it becomes the head of the queue (line 2 in Algorithm 20). In each iteration, SUDSCA checks if cluster C in the head of the queue is homogeneous or not (line 6). If it is, the cluster is added to SUDS (lines 7–9). Otherwise, the algorithm computes a mean item for each class ($ClassCentroids$) present in C (lines 11-15). SUDSCA continues by calling the k-means clustering for the items of C (line 16) (see Section 2.4). This procedure returns a list of clusters ($NewClusters$) that are added to the queue structure (line 17-19) as unprocessed clusters. This procedure is repeated until the queue becomes empty (line 21), which means that all constructed clusters are homogeneous.

Contrary to TLDSCA (see Algorithm 18) proposed in Section 5.3, SUDSCA is non-parametric. It automatically determines the length of SUDS (i.e., the number of clusters) based on the dataset used. Certainly, SUDSCA extends the idea of the algorithms presented in Chapter 3. Here, our purpose is not the development of a prototype abstraction algorithm, but the development of a hybrid, non-parametric method that combines the idea of data abstraction with that of cluster-based methods. We note that SUDSCA does not depend on the order of items in the training set. Contrary to SUDSCA, other speed-up methods based on k-means clustering highly depend on the selection of the initial means [62, 86, 85, 87].

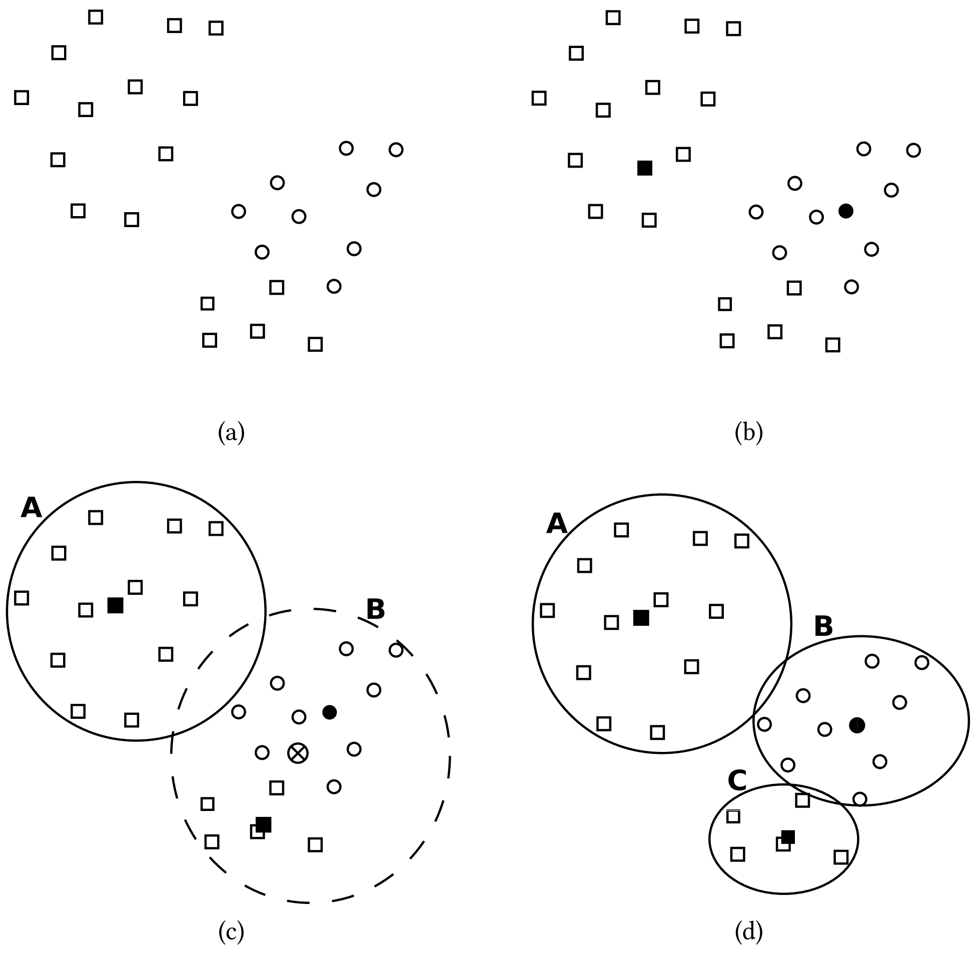


Figure 5.15: SUDS construction by finding homogeneous clusters in the training dataset

Algorithm 20 SUDSCA

```
Input: $TS$
Output: $SUDS$
1: $Queue \leftarrow \varnothing$
2: Enqueue($Queue$, $TS$)
3: $SUDS \leftarrow \varnothing$
4: repeat
5:    $C \leftarrow$ Dequeue($Queue$)
6:    if $C$ is homogeneous then
7:       $M \leftarrow$ mean of $C$
8:       Put $M$ into the first level of $SUDS$
9:       Put the items of $C$ into the second level of $SUDS$ and associate them to $M$
10:   else
11:      $ClassCentroids \leftarrow \varnothing$ {M is the set of class means}
12:      for each class $L$ in $C$ do
13:         $Centroid_L \leftarrow$ mean of $L$
14:         $ClassCentroids \leftarrow ClassCentroids \cup Centroid_L$
15:      end for
16:      $NewClusters \leftarrow K$-MEANS($C$, $ClassCentroids$)
17:      for each cluster $X \in NewClusters$ do
18:         Enqueue($Queue$, $X$)
19:      end for
20:   end if
21: until IsEmpty($Queue$)
22: return $SUDS$
```

Hybrid classifiers based on homogeneous clusters

The second part of the proposed method is a classifier that uses SUDS. It is called Hybrid Classification Algorithm based on Homogeneous Clusters (HCAHC) and is described in Algorithm 21. Although it has similar points to FHC (see Section 5.3), it has two major differences: (i) HCAHC accesses a completely different data structure than FHC; (ii) Contrary to FHC, HCAHC does not use the $npratio$ parameter. More specifically, when a new item x arrives and must be classified (line 1 in Algorithm 21), HCAHC initially scans the first level of SUDS and retrieves the Rk nearest representatives to x (lines 2-4). We call this scan a first level search. If all Rk retrieved representatives vote a specific class, x is classified to this class (lines 5-6). Otherwise, HCAHC goes to the second level of SUDS and x is classified by searching its k “real”

Algorithm 21 HCAHC

```
Input: $SUDS$, $Rk$, $k$
1: for each new item $x$ do
2:     Scan 1st level of $SUDS$ and retrieve the $Rk$ nearest representatives to $x$
3:     Find the majority class $MC_1$ of the $Rk$ nearest representatives (ties are resolved by nearest representatives)
4:     $MCC \leftarrow$ COUNT(representatives of the majority class)
5:     if $MCC == Rk$ then
6:         Classify $x$ to $MC_1$
7:     else
8:         Scan within the set formed by the union of clusters of the $Rk$ representatives and retrieve the $k$ Nearest Neighbours (NNs) to $x$ {Second level search}
9:         Find the majority class $MC_2$ of the $k$ NNs (ties are resolved by single nearest neighbour)
10:        Classify $x$ to $MC_2$
11:    end if
12: end for
```

nearest neighbours within the data subset dynamically formed by the union of the clusters of the Rk representatives (lines 8-10). We call this search a second level search.

A second level search usually involves higher classification cost than a first level search. However, even in this case, HCAHC searches only a small subset of the initial training data. For instance, suppose that SUDSCA has built a SUDS with 200 nodes and we have set $Rk = 8$. HCAHC performs the first level search and retrieves the eight nearest representatives. Suppose that not all eight of them belong to the same class. As a result, HCAHC searches for the k nearest neighbours in the union of the eight clusters that correspond to the eight representatives and performs the classification. Even in this case, HCAHC significantly prunes the search space by ignoring the items of the rest 192 clusters.

A new item can be classified via either a first or a second level search. Practically, the first level search is a prototype abstraction algorithm (similar to RHC), while the second level search is a cluster-based method. That is why HCAHC is a hybrid method. Furthermore, when HCAHC performs a second level search, it accesses an almost noise-free subset of the initial training set. Since each cluster contains items of a specific class only, the subset (union of the Rk clusters) will not contain noisy items of other irrelevant classes, i.e., classes which are not represented by the Rk representatives. Thus, classification performance may not be affected as much by noisy data of other classes. Of course, the length of SUDS and the size of the constructed clusters depends on the level of noise. The more noise in the training set, the smaller size of the constructed clusters and the higher is the final number of homogeneous clusters (or length of SUDS).

Since we aim to a non-parametric method, we must find a way to automatically determine Rk. In the experiments of the following subsection, we have tested the effect of the value of Rk on the performance of our method. In addition, we adopt the empirical rule:

$$Rk = \lfloor \sqrt{|SUDS|} \rfloor$$

where $|SUDS|$ is the number of nodes (clusters) in SUDS. The adoption of this empirical rule was initially motivated by the rule of thumb [78]:

$$C = \lfloor \sqrt{\frac{n}{2}} \rfloor$$

which is used for determining the number of clusters in the context of k-means clustering. Certainly, we adopted the empirical rule by evaluating the performance of HCAHC on multiple datasets.

5.4.3 SUDS classification method over condensing sets

The motive behind the SUDS method presented in Subsection 5.4.2 is fast classification. In addition, we claim that our method could improve the performance of prototype abstraction and condensing algorithms. More specifically, we suggest our SUDS classification method to be applied on the data stored in a condensing set that has been constructed by a condensing or prototype abstraction algorithm. Then, a classifier that uses SUDS will be executed faster than a classifier that searches for nearest neighbours in the full condensing set and without a negative impact on accuracy. Since SUDSCA will be applied on a condensing set (i.e., a small dataset), the preprocessing overhead introduced will be almost insignificant.

Suppose that a condensing set stores the close-class-border items of a dataset with four classes (Figure 5.16(a)). Figures 5.16 (b)–(d) demonstrate the execution of SUDSCA. The result is the construction of a SUDS which contains five clusters. The SUDS construction procedure is similar to that presented in Figure 5.15.

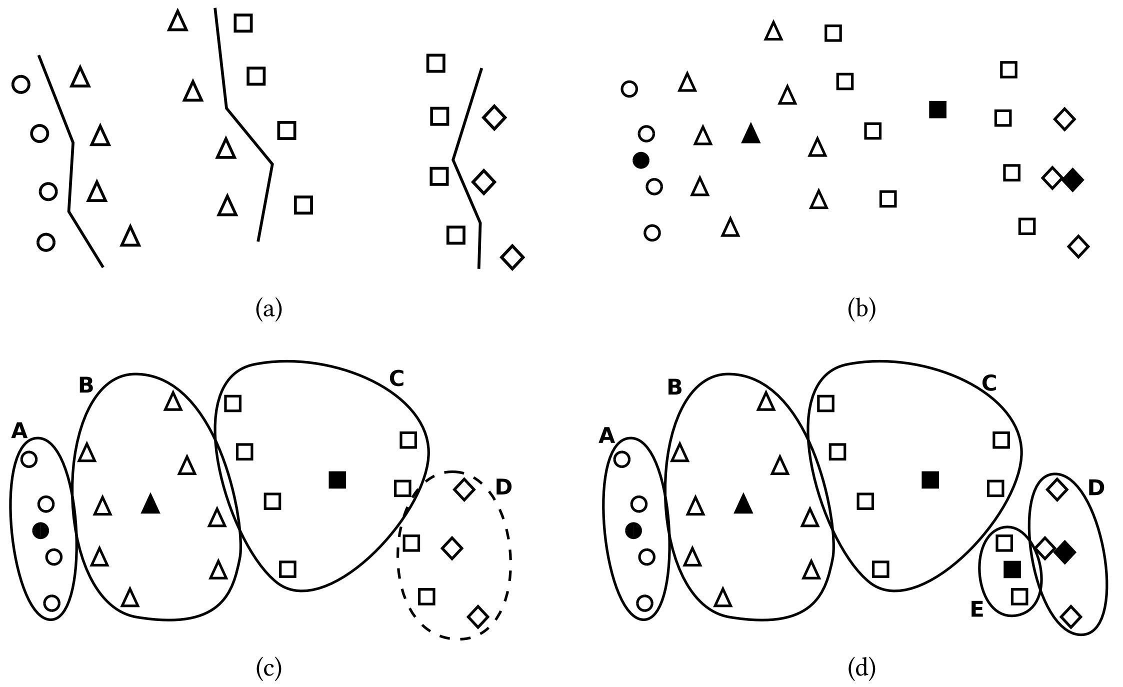


Figure 5.16: SUDS construction by finding homogeneous clusters in the condensing set

Here, the classifier that uses SUDS to perform the classification is almost similar to HC-AHC. However, there is a major difference: it is rare that all Rk nearest representatives will belong to the same class. Almost in all cases, HCAHC proceeds to a second level search. Therefore, when the SUDS method is applied on a condensing set, the "if statement" (line 5 in Algorithm 21) regarding the first level classification is unnecessary (actually, it is meaningful only when Rk is very small). Therefore, a classifier that avoids classification though a first level search is a sufficient approach. We call this approach Homogeneous Clusters Algorithm (HCA) and we present it in Algorithm 22.

Contrary to HCAHC, HCA is not a hybrid classifier. Basically, it is a typical cluster-based method. For each new item, it dynamically forms a training subset (reference set) of the initial condensing set on which the search for the nearest neighbours is executed. The training subset is the union of the Rk clusters that have been constructed by SUDSCA. As in the case of HCAHC, HCA adopts the empirical rule $Rk = \lfloor\sqrt{|SUDS|}\rfloor$ for the determination of Rk.

Algorithm 22 HCA

```
Input: $SUDS$, $Rk$, $k$
1: for each new item $x$ do
2:     Scan 1st level of $SUDS$ and retrieve the $Rk$ Nearest Representatives to $x$
3:     Scan within the set formed by the union of clusters of the $Rk$ representatives and retrieve the $k$ Nearest Neighbours (NNs) to $x$
4:     Find the majority class $MC$ of the $k$ NNs (ties are resolved by the nearest neighbour)
5:     Classify $x$ to $MC$
6: end for
```

5.4.4 Performance evaluation

The experimentation includes two main stages[5]. First, we evaluate the performance of SUDS classification method over non-edited data, and then, over condensing sets built by four DRTs. In all cases, our experiments were conducted using eight datasets distributed by the KEEL dataset Repository[6][6] (see Table 5.12). Like in the other experimental studies of the dissertation, the Euclidean distance is used as distance metric.

The SUDS classification method was compared to five known speed-up methods by measuring three comparison criteria: classification accuracy, classification cost, and, preprocessing cost. We evaluated: (i) CNN-rule, (ii) the fast IB2 algorithm, (iii) RSP3, (iv) the cluster-based method proposed by Hwang and Cho [62] (HCM), and, (v) our RHC algorithm.

All methods were evaluated using five-fold cross-validation. For each one of the seven datasets (except KDD), we used the five already constructed pairs of training/testing sets hosted by the KEEL repository. These sets are appropriate for five-fold cross-validation. With the exception of KDD, we run all experiments without any previous knowledge about the datasets. Therefore, we did not perform normalization or any other data transformation.

The KDD dataset contains 41 attributes and 494,020 items. However, three attributes are nominal, two attributes are fixed-value and huge amounts of data are duplicates. For simplifying our experiments and like in the experimental studies presented in Sections 3.2.2 and 4.2, we removed these attributes and all duplicate items. Therefore, the transformed dataset contains 36 attributes and 141,481 unique items. Please note that the impact of duplicates on the classification process has been documented in Subsection 3.2.4. Moreover, we observed

[5]Detailed experimental results are available at
`http://users.uom.gr/~stoug/AIRJ_experiments.zip`

[6]`http://sci2s.ugr.es/keel/datasets.php`

Table 5.12: Datasets description

Dataset	Size	Attributes	Classes
Letter Recognition (LIR)	20000	16	26
Magic Gamma Telescope (MGT)	19020	10	2
Pen-Digits (PD)	10992	16	10
Landsat Satellite (LS)	6435	36	6
Shuttle (SH)	58000	9	7
Texture (TXR)	5500	40	11
Phoneme (PH)	5404	5	2
KddCup (KDD)	141481	23	36

extreme variation on the value ranges of the attributes of KDD. Thus, we normalized all attributes to the interval $[0, 1]$. Finally, we randomized the dataset and prepared it for five-fold cross-validation.

Performance of SUDS classification method over the original data

Experimental setup CNN-rule, IB2, RSP3 and RHC are non-parametric methods, that is, they do not use user-defined parameters in order to reduce the training data. On the other hand, HCM is parametric. In addition to parameter k (number of nearest neighbours to search), which is used by all methods during the classification step, it uses three extra parameters: (i) C: the number of clusters constructed by the k-means clustering, (ii) D: the distance threshold used to divide each cluster into core and peripheral sets, and, (iii) L: the number of adjacent clusters that will be used. C and D are used during the preprocessing step, while L during the classification step (see Section 2.2 or [62] for details). For each dataset, we built eight HCM classifiers using eight different C values. More specifically, each classifier $i = 1, \ldots, 8$, used $C = \lfloor\sqrt{\frac{n}{2^i}}\rfloor$, where n is the number of training items. The first classifier, i.e. $i = 1$, is based on the rule of thumb $C = \lfloor\sqrt{\frac{n}{2}}\rfloor$ [78]. We decided to build additional classifiers that use smaller C values than $C = \lfloor\sqrt{\frac{n}{2}}\rfloor$ based on the observation that Hwang and Cho defined $C = 10$ for a dataset with 60919 items. For the other two parameters, we adopted the values suggested by Hwang and Cho in their experiments.

Although SUDSCA is non-parametric, HCAHC is a parametric classifier. In addition to k, it uses the Rk parameter. We built 29 HCAHC classifiers, for $Rk = 2, 3, \ldots, 30$, and we also considered the automatic determination of Rk (see Subsection 5.4.2). We refer to that classifier as HCAHC-sqrt. To facilitate the presentation of the measurements, for both HCAHC and HCM, we report only the most accurate classifiers for each reported cost, i.e., the performance of a classifier was omitted if it was less accurate and involved higher classification cost than another classifier that was more accurate with lower classification cost.

During the classification step, all methods involve the k parameter. DRTs perform k-NN classification using their condensing set, while HCM does this over a small reference set that is dynamically formed for each new item. Finally, HCAHC searches for k nearest neighbours when it performs a second level search. We used the best k values for each method and dataset, i.e., the value that achieved the highest classification accuracy. In effect, we ran the cross-validation many times for different k values and report the best one. Of course, we did not use different k values for each fold. We note that we did not follow the typical tuning procedure that implies that the best parameters should be obtained by using only the training data. Since all methods involve the k parameter, for all methods, we simply report the highest accuracy achieved by the k parameter when it classifies the testing portion using the corresponding training portion.

For the first seven datasets, we run all experiments twice: on the non-edited datasets and on the edited (noise-free) datasets. For editing purposes, we used ENN-rule by setting $k = 3$ [131, 49, 84]. The goal was to study how noise affects the performance of each method. The complete procedure that we followed during our experimentation is shown in Figure 5.17. For the KDD dataset, ENN-rule eliminates all items of some rare classes. More specifically, the edited form of the KDD dataset contains fewer than 23 classes (from 4 to 7 fewer classes - depending on the fold). Therefore, we decided to skip the edited version of the KDD dataset.

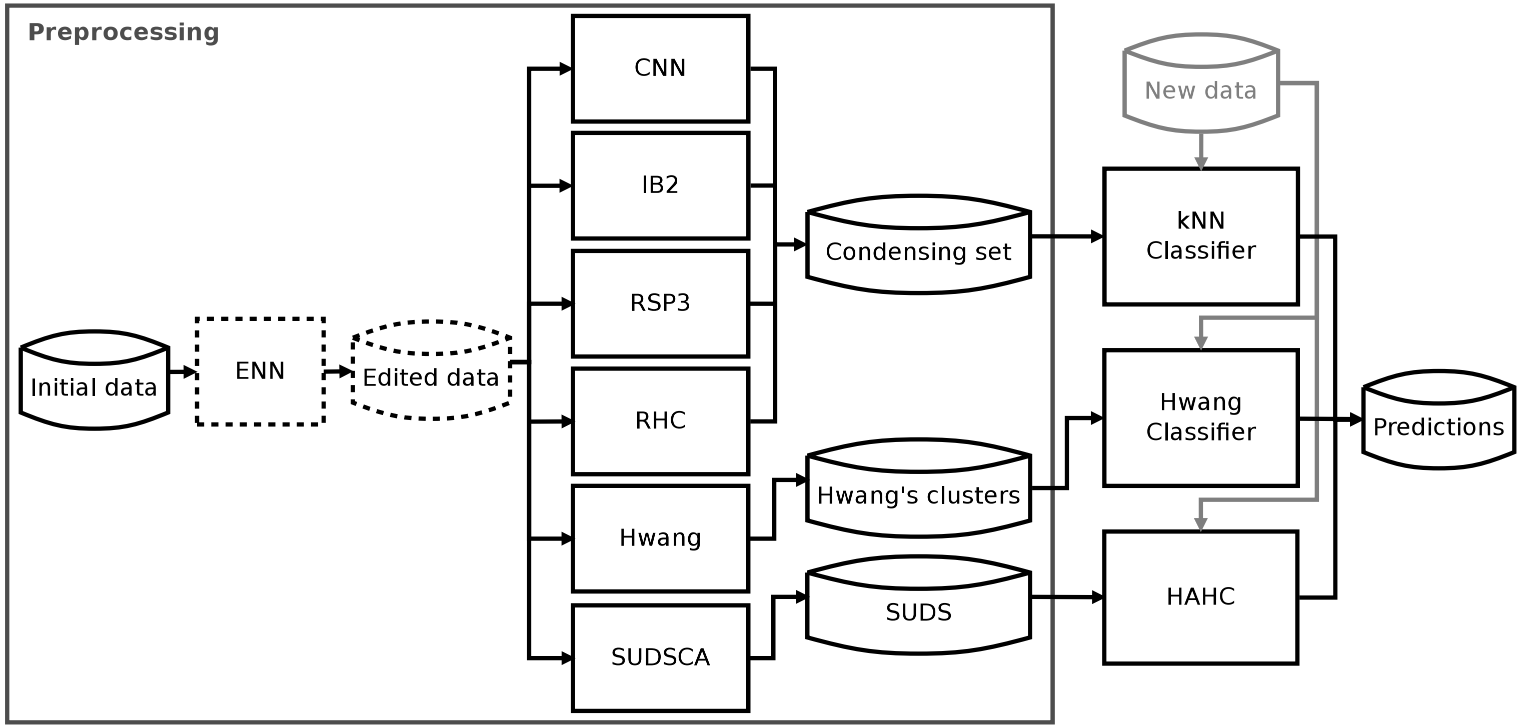


Figure 5.17: Classification procedure of the experimental study

Pre-processing performance Tables 5.13 and 5.14 present the pre-processing costs in terms of millions of distance computations (how many distances were computed during pre-processing) performed by each method on the non-edited and the edited data. As we expected, SUDSCA and RHC were executed very fast in comparison to the other approaches, and were comparable in performance to IB2, which is a one-pass algorithm. This happened because: (i) the construction of SUDS and of the RHC condensing set are based on the repetitive execution of the fast k-means clustering algorithm, and, (ii) in both cases, k-means uses the mean items of the classes as initial means, and thus, clusters are consolidated quickly.

Table 5.13: Experimental results: Preprocessing Cost of SUDSCA on the non-edited data (millions of distance computations)

Dataset	**CNN**	**IB2**	**RSP3**	**HCM**				**RHC/ SUDSCA**
				i=1	**i=3**	**i=5**	**i=7**	
LIR	163.03	23.37	326.52	88.88	63.66	26.35	10.89	41.85
MGT	277.18	34.61	511.67	120.22	72.64	21.74	10.64	4.09
PD	11.75	1.78	86.66	28.80	11.27	5.97	1.70	2.88
LS	18.59	2.22	37.70	16.74	12.44	4.54	0.81	1.69
SH	45.30	8.26	17410.12	744.82	399.23	105.13	34.78	16.83
TXR	5.57	0.84	27.63	14.86	7.43	3.89	0.83	3.63
PH	13.45	1.96	20.31	9.87	3.70	1.33	0.74	0.65
KDD	384.90	55.58	20278.87	5440.21	2155.75	955.06	309.97	81.59
Avg.	114.97	16.08	4837.43	808.05	340.77	140.50	46.29	19.15

Table 5.14: Experimental results: Preprocessing Cost of SUDSCA on the edited data (millions of distance computations)

Dataset	**CNN**	**IB2**	**RSP3**	**HCM**				**RHC/ SUDSCA**
				i=1	**i=3**	**i=5**	**i=7**	
LIR	112.20	18.35	300.51	74.60	55.31	31.45	9.93	31.05
MGT	70.27	8.51	318.82	85.28	33.80	16.33	7.06	2.83
PD	9.25	1.51	85.15	26.76	12.06	5.18	1.83	2.83
LS	7.09	1.02	30.63	13.76	9.60	3.43	0.87	1.74
SH	26.02	6.35	15652.75	867.40	367.52	146.12	37.18	22.41
TXR	4.39	0.71	27.04	14.76	7.16	4.27	0.82	3.00
PH	5.57	0.86	15.67	5.58	2.65	1.03	0.48	0.47
Avg.	33.54	5.33	2347.22	155.45	69.73	29.69	8.31	9.19

Concerning the other methods, RSP3 was the most time consuming approach. HCM for $i \geq 5$ is executed very fast. However, in real applications, the user must perform a trial-end-error procedure for determining the parameters. This may render pre-processing a hard and extremely time consuming procedure. Although CNN-rule is quite faster than RSP3, its pre-processing cost remains at high levels.

Finally, as we expected, DRTs and SUDSCA are faster when executed on the edited datasets. This happens because: (i) the edited sets contain fewer items than the non-edited training sets, and, (ii) noisy items have a negative effect on the methods. One one hand, in RHC, RSP3 and SUDCA that use the concept of homogeneity, noisy items lead to many and small groups of items and, thus, these methods involve a high cost to build them. On the other hand, in CNN-rule, noisy items lead to the execution of many algorithm data passes and of course to a large condensing set. The preprocessing cost of HCM is not affected by noisy items. However, for noisy datasets, even this method involved lower preprocessing cost because of the smaller size of the edited sets.

Nevertheless, to obtain an edited (i.e., noise-free) dataset, the execution of an extra preprocessing procedure is necessary (see Figure 5.17). This procedure involves additional preprocessing cost. Hence, the total preprocessing cost contains the cost of ENN-rule that is $\frac{N*(N-1)}{2}$, where N is the number of training items. Table 5.15 shows the cost required by ENN-rule as well as the size of the edited sets and the corresponding reduction rate. We observe that the MGT, PH, and LS datasets contains noise, while in the rest datasets, the level of noise is almost insignificant.

Table 5.15: Experimental results of ENN-rule

	LIR	**MGT**	**PD**	**LS**	**SH**	**TXR**	**PH**
Edited set size:	15306.8	12160.2	8734	4681	46314.2	4345.6	3837.4
Accuracy (%):	94.98	80.96	99.30	90.41	99.79	98.64	880.14
Reduction Rate (%):	4.33	20.08	0.67	9.07	0.18	1.24	11.25
ENN Cost:	127.99	115.76	38.65	13.25	1076.46	9.68	9.35

Classification performance We performed the classification step by using eight classification methods on the eight datasets. The methods used were: (i) Conventional k-NN (conv-k-NN), (ii) IB2, (iii) HCAHC, (iv) HCAHC-sqrt, (v) RHC, (vi) CNN-rule, (vii) RSP3, and, (viii) HCM. The performance measurements of conv-k-NN are shown in Table 5.16 while the

Table 5.16: Experimental results: Conventional-k-NN vs HCAHC over the non-edited data (Accuracy (Acc (%)), Classification Cost (millions of distance computations))

Dataset	**Conv-k-NN**		**HCAHC**		
	Acc (%)	**Cost**	**Acc (%)**	**Cost**	***Rk***
LIR	96.01	64.00	95.90	9.41	43
MGT	81.32	57.88	81.27	16.26	63
PD	99.37	19.34	99.33	2.45	20
LS	91.22	6.63	91.25	1.10	15
SH	99.82	538.24	99.82	227.26	29
TXR	99.02	4.84	99.02	0.71	18
PH	90.10	4.67	90.23	1.08	11
KDD	99.71	3202.68	99.71	429.31	10
Avg.	94.5715	487.29	94.5663	85.97	

measurements of the speed-up methods are depicted in Figures 5.18-5.23. In particular, each figure presents two diagrams for each dataset, one corresponding to the non-edited training set and one to the edited set.

The figures show the cost measurements (in terms of millions or thousands distance computations) on the x-axis and the corresponding accuracy on the y-axis. The cost measurements indicate how many distances were computed in order to classify all testing items. Since, we used a cross-validation schema, cost measurements are average values.

Almost in all cases, HCAHC and HCAHC-sqrt had very good performance. HCAHC can even reach the accuracy level of conv-k-NN (see Table 5.16). All diagrams show that rule $Rk = \lfloor\sqrt{|SUDS|}\rfloor$ is a good choice for the determination of Rk. With the exception of the SH and KDD datasets, HCAHC achieved better classification performance than all DRTs. On the other hand, although HCAHC and HCAHC-sqrt achieved higher accuracy than HCM in all datasets, for the MGT, SH, PH and KDD datasets, HCM may be preferable because it achieved accuracies close to those of HCAHC and HCAHC-sqrt at a lower classification cost.

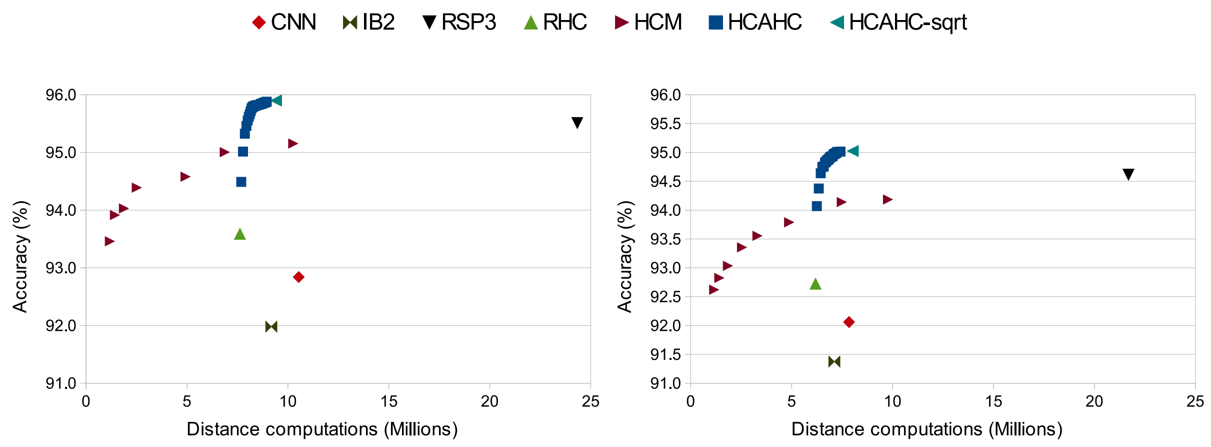
Table 5.16 compares the classification performance on the non-edited datasets of conv-k-NN with that of HCAHC with an Rk value that achieves the highest possible accuracy (note that we have not conducted experiments for $Rk > 30$ and for $Rk \neq \lfloor\sqrt{|SUDS|}\rfloor$). Please observe that the accuracy measurements are almost similar with significant gains in classification cost. In the LS and PH datasets, HCAHC achieved slightly better accuracy than conv-k-NN. This is because the reference sets formed by HCAHC during the second level searches do not contain items of irrelevant classes.

Concerning the SH and KDD datasets, all DRTs built very small condensing sets. Therefore, the k-NN classifiers that were applied on these condensing sets, were not only accurate but very fast as well. HCAHC and HCAHC-sqrt were able to achieve even higher accuracy levels than all DRTs but, it involves higher classification cost.

All figures show that when the speed-up methods are performed over the edited data, they are faster than when they performed over the non-edited data. However, in some cases, either the cost gains are not very high or the classification accuracy is significantly reduced. On the other hand, in the case of the MGT dataset, which is a dataset that contains high level of noise, editing is a necessary preprocessing procedure for all speed-up methods. In Figure 4.6, we observe that the cost gains are high, while the classification accuracy is not reduced.

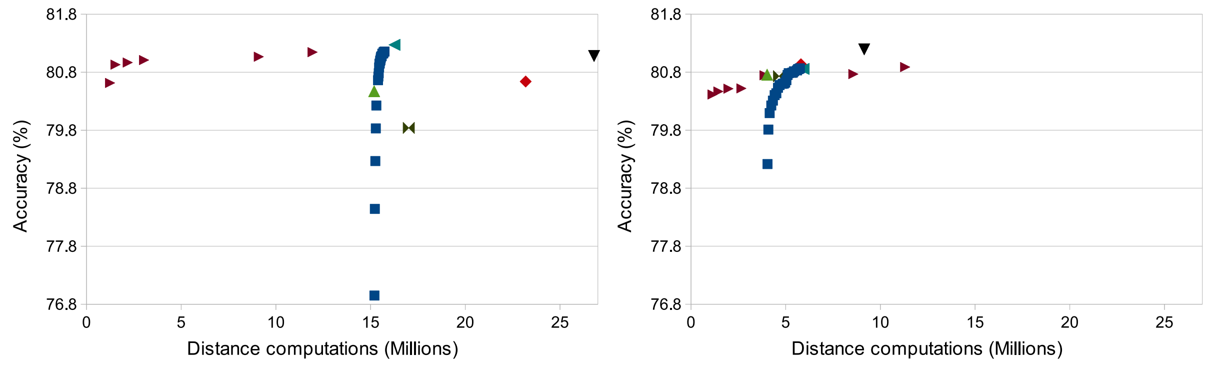
Non-parametric statistical test The experimentation is complemented by the results of the Wilcoxon signed ranks test [32]. Like the other statistical studies presented in the dissertation, the test was run four times. Once for each comparison criterion (accuracy (ACC), classification cost (CC), preprocessing cost (PC)) and once on the measurements of the overall classification performance. The measurements of the overall classification criterion were estimated by averaging the normalized to the range $[0, 1]$ measurements of the three aforementioned criteria, thus, assuming that they all have the same significance. Since low values for costs are desirable, we used the values $(1 - normalized(CC))$ and $(1 - normalized(PC))$ in the place of the normalized values for CC and PC, respectively.

All the tests were run twice, one on the measurements obtained from the edited datasets and one on the measurements obtained from the edited datasets. Of course, we could not include tests for all variations of HCAHC and HCM (for the different tested values of Rk and i parameters respectively). Since the performance of the algorithms are estimated in terms of three comparison criteria, there is not a unique dominant parameter value. A good parameter adjustment in terms of one criterion may deteriorate the measurements of the other criterion and vice versa. Therefore, for each dataset we chose a good representative variation for each one of these parametric algorithms. Our criterion was the high performance, i.e., relatively high accuracy and low classification and preprocessing cost. The parameter values for the selected classifiers are shown in Table 5.17.



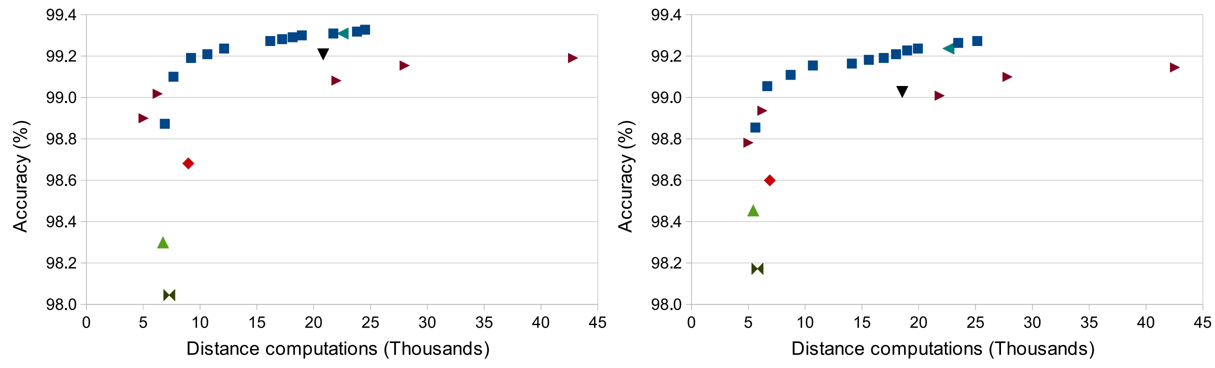
(a) Non-edited data (b) Edited data

Figure 5.18: LIR (Accuracy and Classification Cost)



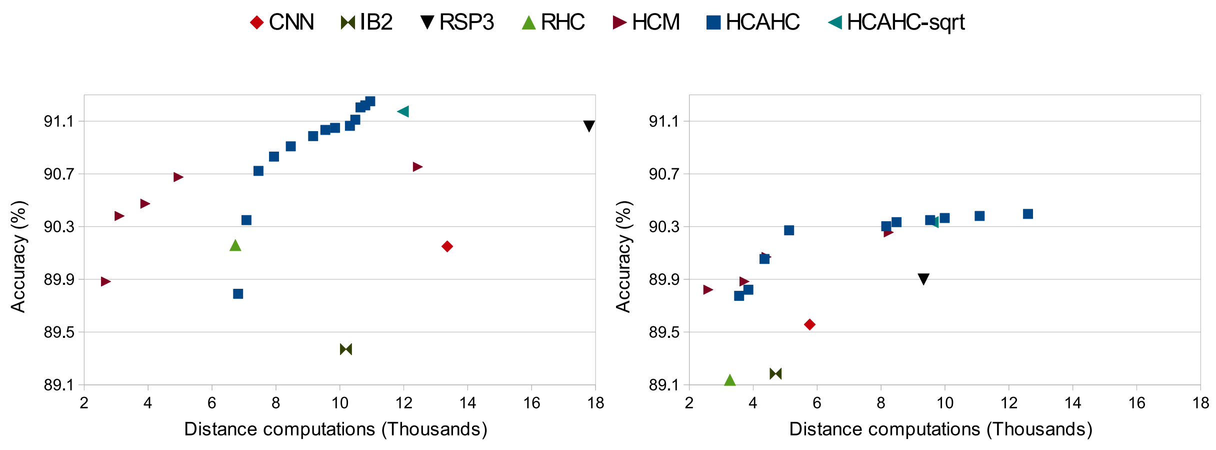
(a) Non-edited data (b) Edited data

Figure 5.19: MGT (Accuracy and Classification Cost)



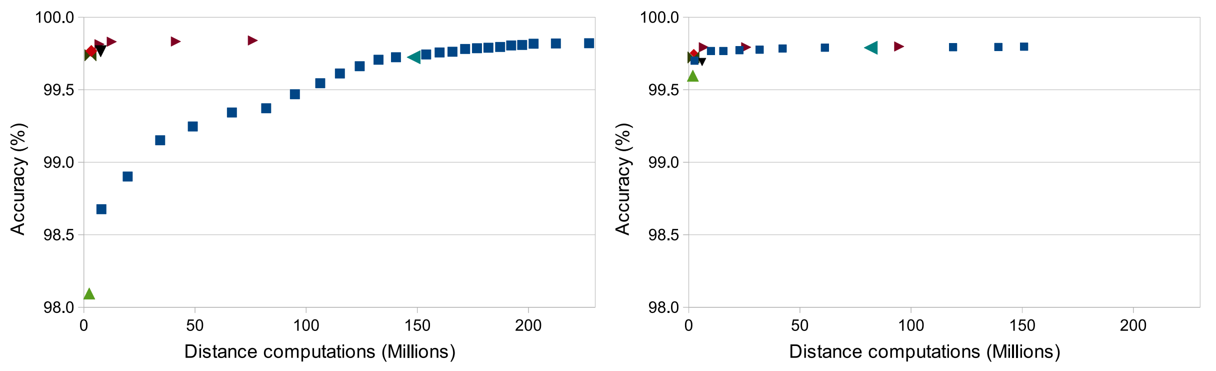
(a) Non-edited data (b) Edited data

Figure 5.20: PD (Accuracy and Classification Cost)



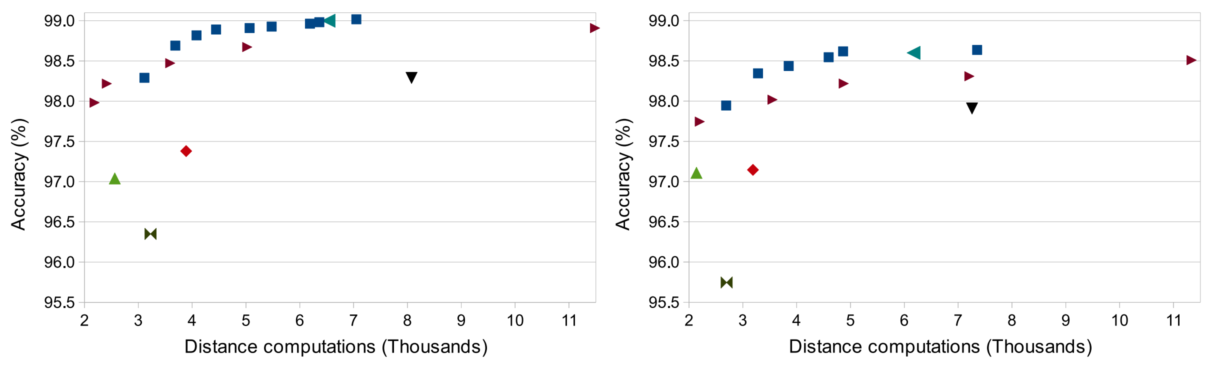
(a) Non-edited data (b) Edited data

Figure 5.21: LS (Accuracy and Classification Cost)



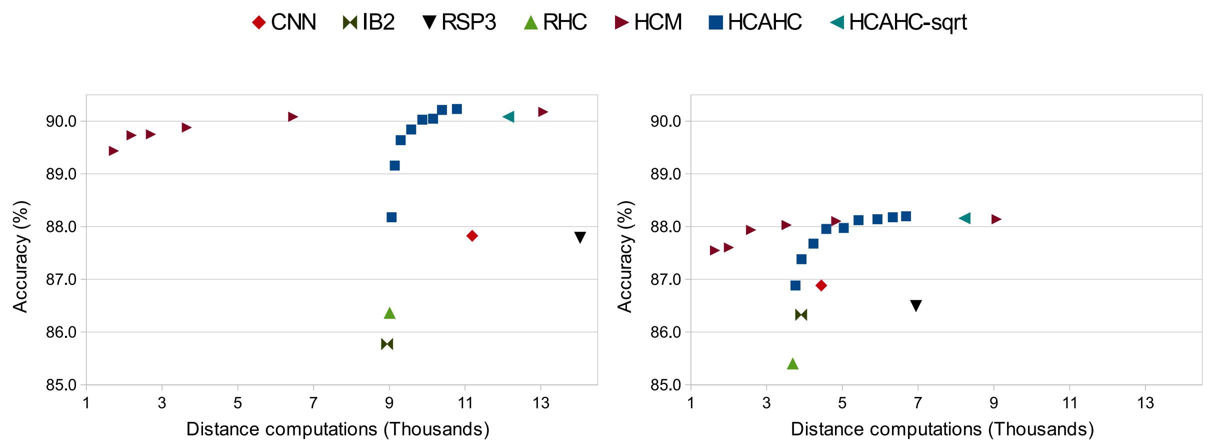
(a) Non-edited data (b) Edited data

Figure 5.22: SH (Accuracy and Classification Cost)



(a) Non-edited data (b) Edited data

Figure 5.23: TXR (Accuracy and Classification Cost)



(a) Non-edited data (b) Edited data

Figure 5.24: PH (Accuracy and Classification Cost)

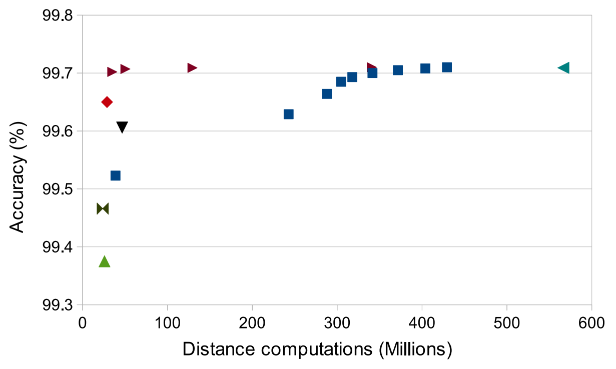


Figure 5.25: Non-edited KDD (Accuracy and Classification Cost)

Additionally to the parametric HCAHC (HCAHC-Rk), we ran the test for HCAHC-sqrt. Consequently, HCAHC-sqrt and HCAHC-Rk are compared to each one of the four other speed-up methods. Note that the execution of HCAHC implies the execution of SUDSCA during the preprocessing phase. Tables 5.18 and 5.19 illustrate the results of Wilcoxon signed ranks tests. The columns labelled by "w/l/t" count the number of wins, loses and ties respectively for each pair of methods. The columns labelled "Wilc." presents the Wilcoxon significance level. We consider that If that value is not higher than 0.05, the difference between the pair of methods is significant. Of course, $Wilc = 0.05$ is a very strict threshold.

The test confirms that HCAHC is an accurate method. Almost in all cases, the Wilcoxon significance level is lower than the threshold of 0.05. In addition, in two cases, it is 0.063.

Table 5.17: Parameter values selected for the Wilcoxon signed ranks test

	LIR	MGT	PD	LS	SH	TXR	PH	KDD
Non-edited data								
Rk	10	23	6	15	9	5	11	2
i	7	3	7	4	3	5	4	1
Edited data								
Rk	8	17	5	5	2	3	7	-
i	7	5	7	4	1	5	3	-

Although HCAHC approaches have always more wins than the other methods in terms of accuracy, in some cases, there is not significant difference. On the other hand, HCAHC seems to be statistically worse than some other algorithms in terms of classification cost. In terms of preprocessing cost, HCAHC is statistically better than CNN and RSP3. Although HCAHC has more wins against HCM in terms of preprocessing cost, its dominance is not statistically supported. Finally, we observe that HCAHC have more wins than all other methods in terms of overall classification performance. However, in many cases, It is not statistically supported. This is because we have adopted the very strict threshold of 0.05 as well as our sample is small. Please notice that there are many cases where the Wilcoxon value is 0.063 (slightly higher than the threshold adopted). The difference between the corresponding algorithms can be also characterized as statistically significant. Furthermore, the test shows that there is not significant difference between HCAHC-sqrt and HCM-i in terms of preprocessing cost. However, considering that both the preprocessing and classification algorithms of HCM are parametric and SUDSCA and HCAHC-sqrt are not parametric, one concludes that the SUDS method is preferable. Since RHC and HCAHC are based on a similar procedure of forming homogeneous clusters, they have the same preprocessing cost. On the other hand, HCAHC is better in terms of accuracy and worse in terms of classification cost than RHC.

Performance of the SUDS classification method over condensing sets

Experimental setup The second part of our experimentation concerns the performance of the SUDS method when applied on condensing sets. To build the condensing sets, we executed the four DRTs that we had used in the study of the previous subsection, i.e. (i) CNN, (ii) IB2, (iii) RSP3, and (iv) RHC, on the training sets presented in Table 5.12. Then, we applied the SUDS method on the four resulting condensing sets. Of course, SUDS can be combined with

Table 5.18: Results of the Wilcoxon signed ranks test on the measurements obtained from the non-edited data

Methods	ACC		CC		PC		**Overall**	
	w/l/t	**Wilc.**	**w/l/t**	**Wilc.**	**w/l/t**	**Wilc.**	**w/l/t**	**Wilc.**
HCAHC-sqrt vs CNN	7/1/0	**0.017**	3/5/0	0.401	8/0/0	**0.012**	6/2/0	0.674
HCAHC-sqrt vs RSP3	7/1/0	**0.017**	5/3/0	0.889	8/0/0	**0.012**	7/1/0	0.161
HCAHC-sqrt vs IB2	7/1/0	**0.017**	1/7/0	0.069	3/5/0	0.401	6/2/0	0.674
HCAHC-sqrt vs RHC	8/0/0	**0.012**	0/8/0	**0.012**	0/0/8	1.000	6/2/0	0.575
HCAHC-sqrt vs HCM-i	7/1/0	**0.025**	1/7/0	**0.025**	6/2/0	0.161	5/3/0	0.889
HCAHC-Rk vs CNN	6/2/0	**0.036**	4/4/0	0.674	8/0/0	**0.012**	6/2/0	0.208
HCAHC-Rk vs RSP3	6/2/0	0.161	7/1/0	0.161	8/0/0	**0.012**	7/1/0	0.123
HCAHC-Rk vs IB2	7/1/0	**0.025**	2/6/0	0.327	3/5/0	0.401	6/2/0	0.161
HCAHC-Rk vs RHC	8/0/0	**0.012**	0/8/0	**0.012**	0/0/8	1.000	6/2/0	0.161
HCAHC-Rk vs HCM-i	6/2/0	0.161	2/6/0	0.093	6/2/0	0.161	6/2/0	0.263

Table 5.19: Results of the Wilcoxon signed ranks test on the measurements obtained from the edited data

Methods	ACC		CC		PC		**Overall**	
	w/l/t	**Wilc.**	**w/l/t**	**Wilc.**	**w/l/t**	**Wilc.**	**w/l/t**	**Wilc.**
HCAHC-sqrt vs CNN	6/1/0	**0.043**	0/7/0	**0.018**	7/0/0	**0.018**	5/2/0	0.310
HCAHC-sqrt vs RSP3	6/1/0	0.063	3/4/0	0.866	7/0/0	**0.018**	7/0/0	**0.018**
HCAHC-sqrt vs IB2	7/0/0	**0.018**	0/7/0	**0.018**	2/5/0	0.176	5/2/0	0.310
HCAHC-sqrt vs RHC	7/0/0	**0.018**	0/7/0	**0.018**	0/0/7	1.000	5/2/0	0.310
HCAHC-sqrt vs HCM-i	6/1/0	**0.028**	1/6/0	0.063	5/2/0	0.237	5/2/0	0.735
HCAHC-Rk vs CNN	5/2/0	0.063	3/4/0	0.866	7/0/0	**0.018**	6/1/0	**0.043**
HCAHC-Rk vs RSP3	6/1/0	0.128	7/0/0	**0.018**	7/0/0	**0.018**	7/0/0	**0.018**
HCAHC-Rk vs IB2	6/1/0	**0.028**	1/6/0	0.091	2/5/0	0.176	5/2/0	0.063
HCAHC-Rk vs RHC	7/0/0	**0.018**	0/7/0	**0.018**	0/0/7	1.000	5/2/0	0.063
HCAHC-Rk vs HCM-i	6/1/0	0.063	5/2/0	0.237	5/2/0	0.237	6/1/0	0.063

any condensing or prototype abstraction algorithm. Once again, we executed all experiments twice, once on the non-edited data and once on the edited data of ENN-rule (with $k = 3$). Figure 5.26 depicts the procedure that we followed during this stage of our experimentation.

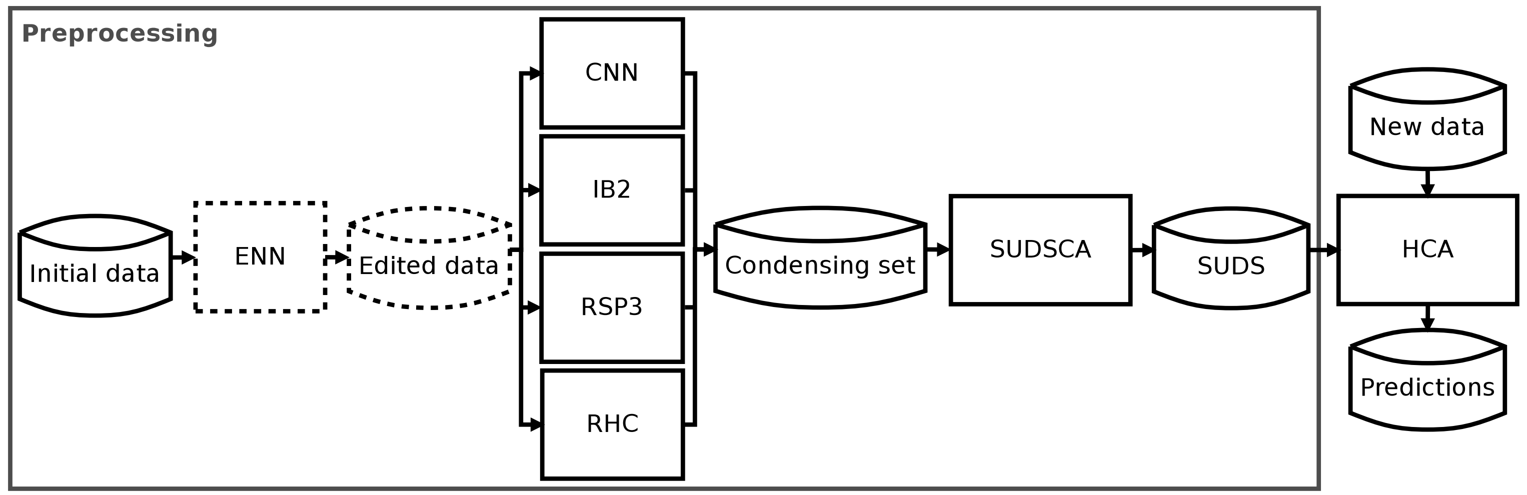


Figure 5.26: Classification using SUDS method on the condensing set

During the classification step, we executed HCA (Algorithm 22). Like HCAHC, HCA uses the Rk parameter. We executed several experiments with different Rk parameter values. In addition, we built an HCA classifier that uses the empirical rule: $Rk = \lfloor \sqrt{|SUDS|} \rfloor$. In all cases, this rule was proven to be a good choice for the determination of Rk. For that reason, in Figures 5.27-5.34, we have included only the performance of the HCA classifier that uses the empirical rule.

Additionally to the performance of HCA, Figures 5.27-5.34 present the performance measurements obtained by the four DRTs (i.e., application of the k-NN classifier on their condensing sets). In this way, we can easily conclude whether the SUDS method can improve the performance of data reduction. Finally, for each method and dataset, we used the k parameter that achieved the highest classification accuracy.

Preprocessing performance SUDS construction constitutes an extra preprocessing step that is applied after the construction of condensing sets (see Figure 5.26). Thus, additionally to the cost shown in Tables 5.14 and 5.15, the total preprocessing cost involves the cost overhead of SUDSCA execution. Table 5.20 shows the preprocessing overheads for each DRT on the non-edited and edited training sets.

The preprocessing cost overheads depend on the size of each condensing set. RSP3 achieved the lowest reduction rates, and so, it involves the highest overhead. In contrast, RHC achieves the highest reduction rates and thus it involves a small overhead. In all cases, overheads added by SUDSCA are almost insignificant. Considering that the preprocessing is executed only once, the small preprocessing overhead does not constitute a problem in real life data mining applications. Actually, efficient data preprocessing implies fast predictions during the classification step.

Classification performance Figures 5.27-5.34 show the performance measurements. Each figure presents two diagrams, one for the non-edited datasets and one for the edited datasets. The diagrams show the performance of the four DRTs when using or not using the SUDS method. Each diagram shows the performance of eight classifiers, two for each DRT: (i) k-NN classifier over its condensing set, (ii) HCA that uses the empirical rule over SUDS.

As we anticipated, the HCA classifier performed better than the k-NN classifier over condensing sets. An HCA classifier avoids a large number of distance computations and, at the same time, keeps the classification accuracy as high as the k-NN classifier. This happens be-

Table 5.20: Experimental results: preprocessing overhead of SUDSCA on condensing sets (millions of distance computations)

Dataset	Non-edited data				Edited data			
	CNN	IB2	RSP3	RHC	CNN	IB2	RSP3	RHC
LIR	2.777	2.158	8.489	1.512	1.688	1.478	7.425	1.072
MGT	1.375	0.914	1.577	0.802	0.220	0.159	0.358	0.131
PD	0.072	0.045	0.218	0.035	0.043	0.034	0.161	0.025
LS	0.208	0.122	0.333	0.076	0.054	0.042	0.104	0.025
SH	0.061	0.052	0.188	0.035	0.034	0.028	0.134	0.024
TXR	0.071	0.054	0.243	0.043	0.038	0.035	0.264	0.032
PH	0.102	0.080	0.144	0.083	0.032	0.025	0.055	0.024
KDD	0.323	0.247	0.624	0.285	-	-	-	-
Avg.	0.624	0.459	1.477	0.359	0.301	0.257	1.215	0.190

cause HCA avoids the distance computations between a new item and items that have been assigned to distant clusters. The performance improvements are significant in both edited and non-edited data. However, they are higher in the case of the non-edited data.

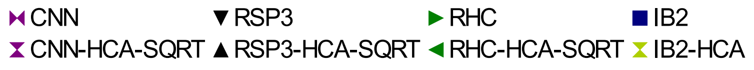
For RSP3, the cost gains are higher than those of the other two methods. For instance, in the cases of the LIR, PD, SH, TXR datasets, the class labels of the testing items were predicted four or three times faster. The performance improvements for CNN-rule and RHC are not as high but they deserve to be mentioned.

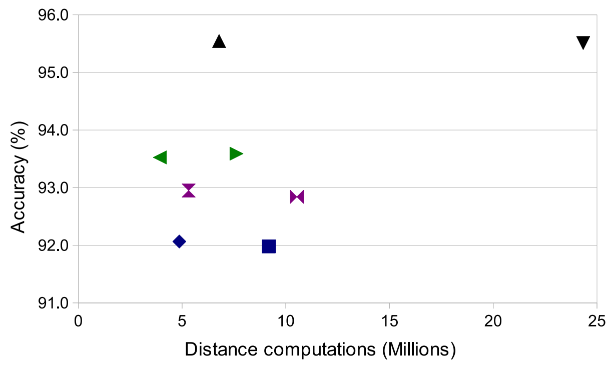
A final comment: SUDS classification method is able to significantly speed-up the predictions of the class labels without loss of accuracy when it is applied on data stored in the condensing set built by DRTs by adding a small cost overhead during the preprocessing phase. This approach is appropriate when extremely fast classification is required.

5.4.5 Conclusions

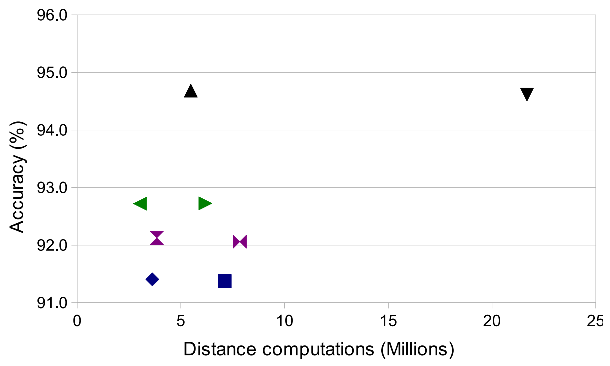
In this section, we presented and evaluated a new classification method. The motivation of our work was the development of a non-parametric method that has low pre-processing cost and is able to classify new items fast and with high accuracy.

We presented an efficient classification method that includes a non-parametric fast preprocessing algorithm that builds a two-level data structure, and a classifier that makes predictions by accessing this structure. In addition, based on the same motivation, we applied the proposed classification method on data stored in the condensing sets constructed by DRTs.



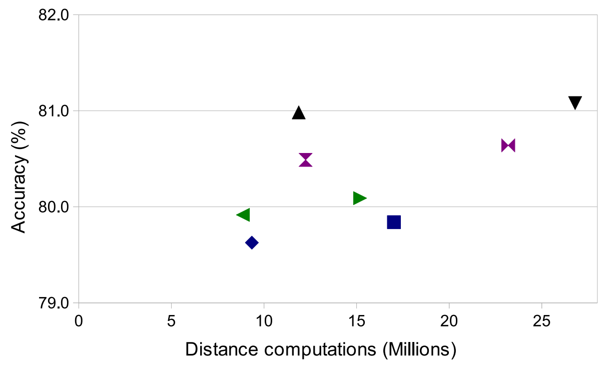


(a) Non-edited data

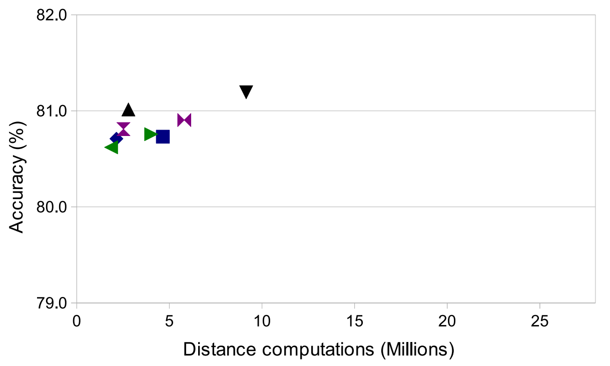


(b) Edited data

Figure 5.27: LIR (Accuracy and Classification Cost)

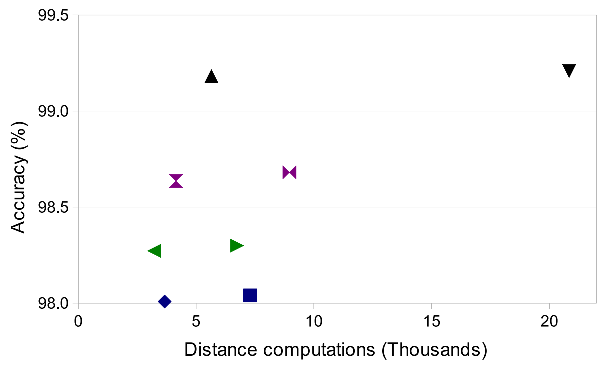


(a) Non-edited data

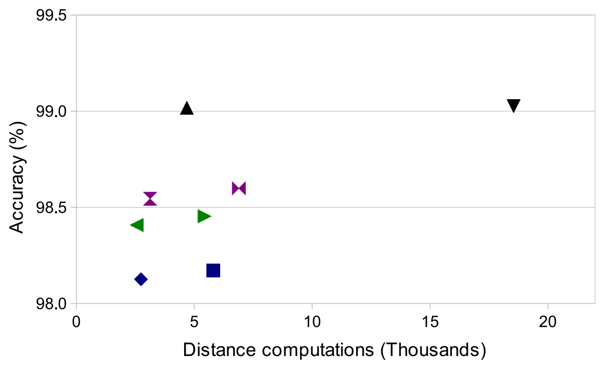


(b) Edited data

Figure 5.28: MGT (Accuracy and Classification Cost)

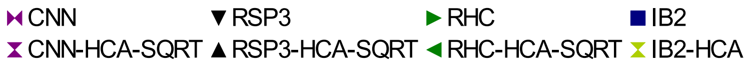


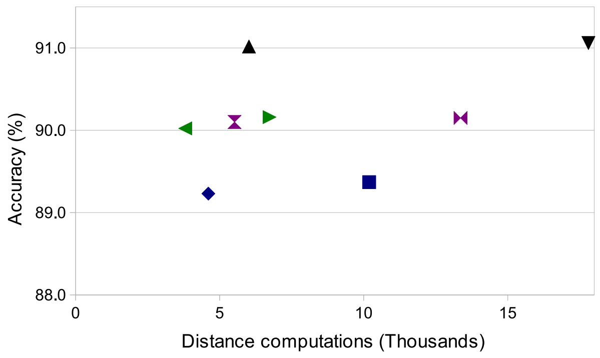
(a) Non-edited data



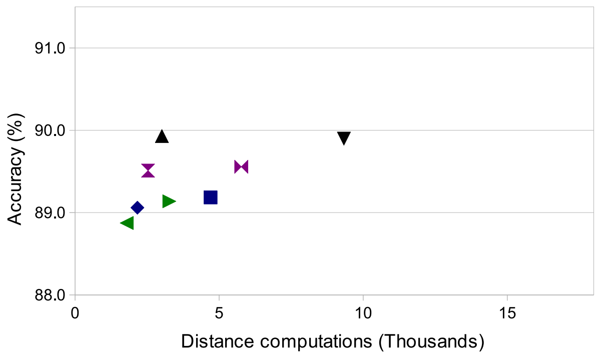
(b) Edited data

Figure 5.29: PD (Accuracy and Classification Cost)



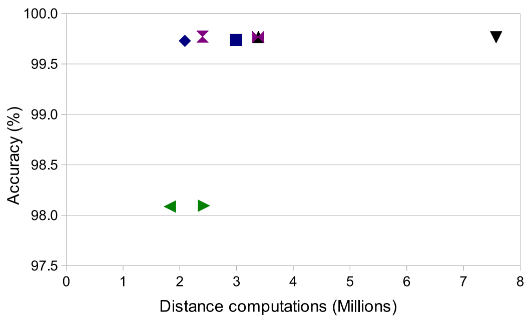


(a) Non-edited data

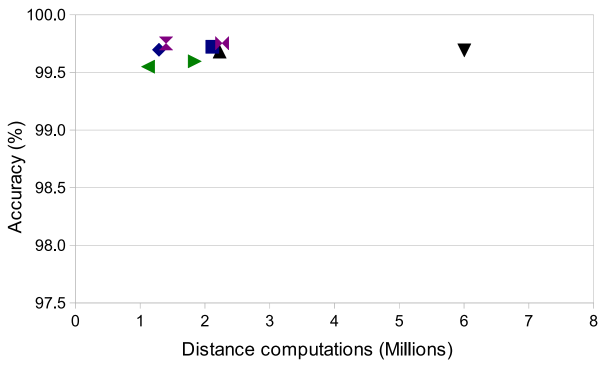


(b) Edited data

Figure 5.30: LS (Accuracy and Classification Cost)

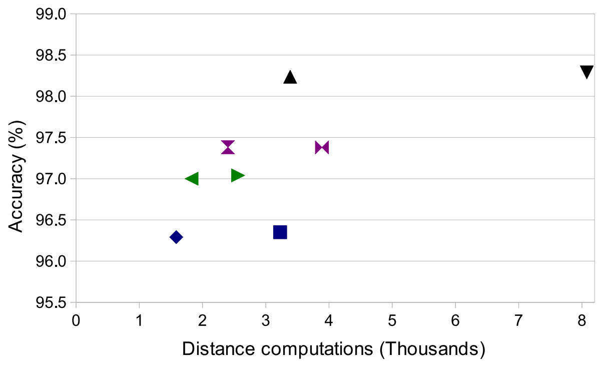


(a) Non-edited data

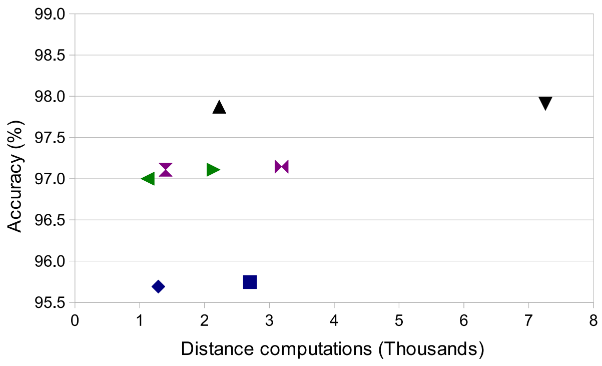


(b) Edited data

Figure 5.31: SH (Accuracy and Classification Cost)

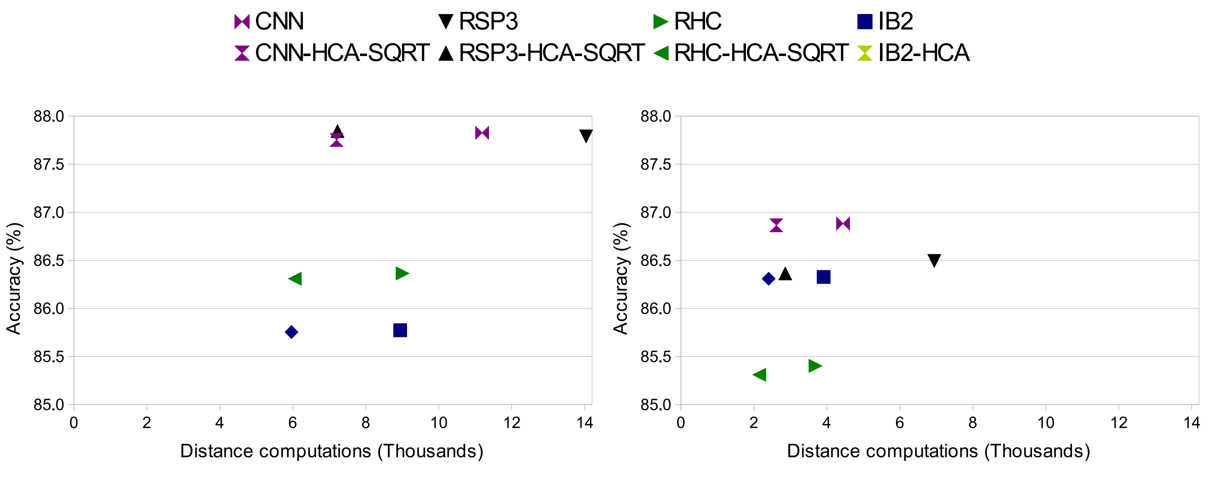


(a) Non-edited data



(b) Edited data

Figure 5.32: TXR (Accuracy and Classification Cost)



(a) Non-edited data

(b) Edited data

Figure 5.33: PH (Accuracy and Classification Cost)

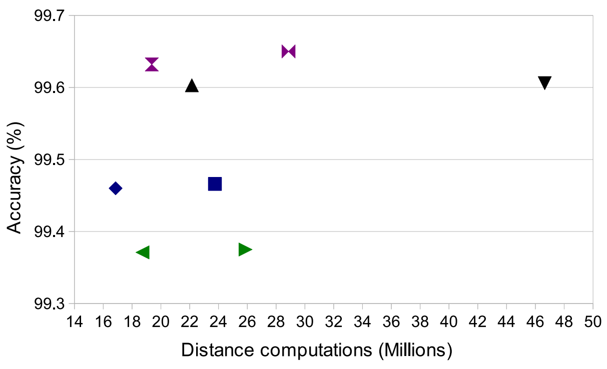


Figure 5.34: Non-edited KDD (Accuracy and Classification Cost)

The goal of this adoption is to improve the performance of such techniques. The proposed HCAHC and HCA classifiers are parametric since they use parameter Rk (number of cluster representatives to use in a first level search). However, we demonstrated that Rk can be automatically determined and render the proposed classification method non-parametric. Experimental results based on eight datasets showed that the proposed method achieved the aforementioned goals.

Chapter 6

Additional research tasks and experimentation

6.1 Introduction

This chapter presents enhancements of existing speed-up methods as well as some additional research tasks and experimentations.

Section 6.2 focuses on fast time-series classification. It presents an experimental study where known non-parametric prototype abstraction and condensing algorithms are evaluated on time-series data [104, 103]. In effect, it proposes the adoption of "general-purpose" Data Reduction Techniques (DRTs) for fast time-series classification.

Section 6.3 deals with the Prototype Selection by Clustering (PSC) algorithm [86, 85, 87]. It is a recently proposed condensing algorithm whose goal is the fast generation of the condensing set (i.e., low preprocessing cost) rather than high reduction rates. However, the section demonstrates that the reduction rate and the classification accuracy achieved by PSC can be improved by generating a large number of clusters [93].

In Section 6.4, an extensive experimental study of the Hwang and Cho cluster-based method [62] is presented [100]. The effectiveness of the particular method is based on the adjustment of three parameters. The results of the study illustrate that if the parameters are carefully defined, one can obtain better classification performance than that shown in [62].

6.2 Applying general-purpose data reduction techniques for fast time-series classification

6.2.1 Time-series classification

The classification methods that are based on similarity search have been proven to be effective for time-series data analysis. More specifically, the one-nearest neighbour (1-NN) classifier is a widely-used time-series classification approach. It has been adopted in many time-series classification systems because of its simplicity and effectiveness. However, its efficiency depends on the size of the training set as well as on data dimensionality. For large and high dimensional time-series training sets, the high computational cost involved renders the application of such classifiers prohibitive. Of course, time-series classification performance can be improved through indexing, representation and/or data reduction.

Indexing accelerates classification, but, as already mentioned, works well only in low dimensionality spaces. Thus, one must first use a dimensionality reduction technique to acquire a representation of the original data in lower dimensions. A representation may be considered as a transformation technique that maps a time-series from the original space to a feature space, retaining the most important features. There have been several time-series representations proposed in the literature, mainly for the purpose of reducing the intrinsically high dimensionality of time-series [36].

Data reduction has recently been exploited for fast time-series classification. More specifically, [19] and [134] propose prototype selection algorithms for speeding-up 1-NN time-series classification. The main disadvantage of these methods is that they are parametric. The user must define the size of the condensing set by trial-and-error.

6.2.2 Motivation and contribution

The work presented in this section has been motivated by the following two observations: (i) to the best of our knowledge, state-of-the-art non-parametric condensing and prototype abstraction algorithms have not been evaluated neither on original time-series nor on their reduced dimensionality representations, and, (ii) prototype abstraction algorithms that we have proposed (RHC, AIB2) have not been evaluated on time-series data.

The contribution of this section is the experimental evaluation of two condensing algorithms, namely, CNN-rule and IB2, and three prototype abstraction algorithms, namely, RSP3, RHC and AIB2. The algorithms are evaluated both against original time-series datasets and their reduced dimensionality representations. Section 6.2.3 presents the experimental setup and the results obtained.

Our study adopts the Piecewise Aggregate Approximation (PAA) [70, 137] time-series representation method. The goal is to investigate the degree to which classification accuracy gets affected when applying data reduction on dimensionally reduced time-series. PAA is an effective and very simple dimensionality reduction technique that segments a time-series into h consecutive sections of equal-width and calculates the corresponding mean for each section. The series of these means is the new representation of the original data.

6.2.3 Experimental study

Experimental setup

The five DRTs were evaluated on seven known time-series datasets distributed by the UCR time-series classification/clustering website[1]. Table 6.1 summarizes on the datasets used. All datasets are available in a training/testing form. We merged the training and testing parts and then we randomized the resulting datasets. No other data transformation was performed. The similarity measure we used was the Euclidean distance. Please note that the aforementioned techniques and the datasets presented in Table 6.1 are available on WebDR[2] (see Appendix A).

We report on the experiment we conducted with a certain value for the parameter of the PAA representation. We applied the PAA representation on time-series by setting the number of dimensions equal to twelve ($h = 12$). Most of the research work provides experimental results with values of h ranging from 2 to 20. We found that lower values of h have a negative effect on the classification accuracy, whereas higher values produce time-series that cannot be efficiently indexed by multi-dimensional indexing methods. Hence, we decided to use $h = 12$.

All experiments were run twice, once on the original time-series and once on their 12-dimensional representations. By this way, we wanted to test how the combination of data reduction and dimensionality reduction affects the performance of 1-NN time-series classification.

[1]`http://www.cs.ucr.edu/~eamonn/time_series_data/`
[2]`https://ilust.uom.gr/webdr`

Table 6.1: Time-series datasets description

Time-series dataset	Size (time-series)	Length (Attributes)	Classes
Synthetic Control (SC)	600	60	6
Face All (FA)	2250	131	14
Two-Patterns (TP)	5000	128	4
Yoga (YG)	3300	426	2
Wafer (WF)	7164	152	2
Sweadish Leaf (SL)	1125	128	15
CBF	930	128	3

We evaluated the five DRTs by estimating four measurements, namely, accuracy, classification cost, reduction rate, and, preprocessing cost. The cost measurements were estimated by counting the distance computations multiplied by the number of time-series attributes (time-series length). Of course, the reduction rate and classification cost measurements relate to each other: the lower the reduction rate, the higher is the classification cost. However, classification cost measurements can express the cost introduced by the dimensionality of data. We report on the average values of these measurements obtained via five-fold cross-validation.

Experimental measurements

Tables 6.2 and 6.3 presents the experimental measurements. Table 6.2 presents the results obtained on the original datasets while Table 6.3 presents the results obtained on the 12-dimensional representations of the datasets we got after applying PAA on them. Both tables present the measurements obtained by applying the 1-NN classifier on the non-reduced data (conventional 1-NN). Each cell of the table contains the four measurements obtained by first applying a DRT on the original or 12-dimensional time-series datasets (preprocessing step) and then by using 1-NN on the resulting condensing set (classification step). The cost measurements are in million distance computations. The preprocessing cost measurements do not contain the small cost overhead introduced by PAA execution. Actually, this cost is almost insignificant.

It is noted that 1-NN classification on the 12-dimensional datasets is very fast. In most cases, the preprocessing and classification cost are extremely low, while classification accuracy remains at high, acceptable levels. Therefore, a first conclusion is that one can obtain efficient

Table 6.2: Experimental results on the original datasets: Accuracy (Acc (%)), Classification Cost (CC (millions of distance computations)), Reduction Rate (RR(%)) and Preprocessing Cost (PC (millions of distance computations))

Dataset		Original dimensionality					
		Conv. 1-NN	**CNN**	**IB2**	**RSP3**	**RHC**	**AIB2**
SC	Acc :	91.67	90.17	89.00	98.33	98.67	**99.83**
	CC :	3.46	0.67	0.53	1.38	**0.09**	0.34
	RR :	-	80.50	84.67	60.08	**97.29**	90.13
	PC :	-	7.77	1.31	16.22	2.39	**1.14**
FA	Acc :	95.07	91.60	91.02	**95.46**	93.02	92.94
	CC :	106.11	19.87	18.38	51.65	**12.93**	16.08
	RR :	-	81.28	82.68	51.32	**87.81**	84.84
	PC :	-	216.36	48.96	533.70	140.41	**43.27**
TP	Acc :	**98.50**	94.68	93.60	98.10	93.72	97.06
	CC :	512.00	85.66	76.83	243.51	**55.50**	61.88
	RR :	-	83.27	85.00	52.44	**89.16**	87.92
	PC :	-	1169.75	205.95	2085.42	**150.49**	177.88
YG	Acc :	**93.76**	91.58	89.55	92.85	90.94	90.49
	CC :	742.26	138.56	108.92	229.82	**93.85**	100.26
	RR :	-	81.33	85.33	69.04	**87.36**	86.49
	PC :	-	1854.74	254.41	4072.30	**162.61**	240.73
WF	Acc :	**99.87**	99.69	99.62	99.82	99.55	99.65
	CC :	1248.30	13.59	11.72	26.88	**9.37**	9.71
	RR :	-	98.91	99.06	97.85	**99.25**	99.22
	PC :	-	165.88	31.42	7196.75	63.69	**25.78**
SL	Acc :	52.36	49.87	48.18	52.00	**52.80**	51.56
	CC :	25.92	15.94	14.80	19.00	**12.80**	14.65
	RR :	-	38.51	42.89	26.69	**50.60**	43.49
	PC :	-	112.17	31.39	1537.07	57.01	**31.02**
CBF	Acc :	98.39	98.17	97.63	**99.78**	98.60	99.68
	CC :	17.71	1.29	1.15	1.97	**0.40**	0.59
	RR :	-	92.74	93.49	88.87	**97.74**	96.67
	PC :	-	15.06	3.50	78.48	7.26	**2.01**
Avg	Acc :	89.94	87.97	86.94	**90.91**	89.62	90.17
	CC :	379.40	39.37	33.19	82.03	**26.42**	29.07
	RR :	-	79.51	81.87	63.76	**87.03**	84.11
	PC :	-	505.96	82.42	2217.13	83.37	**74.55**

Table 6.3: Experimental results on the datasets with 12 dimensions: Accuracy (Acc (%)), Classification Cost (CC (millions of distance computations)), Reduction Rate (RR (%)) and Preprocessing Cost (PC (millions of distance computations))

Dataset		12 dimensions					
		Conv. 1-NN	**CNN**	**IB2**	**RSP3**	**RHC**	**AIB2**
SC	Acc :	98.50	97.00	95.83	**98.83**	98.17	98.50
	CC :	0.69	0.06	0.05	0.12	**0.03**	**0.03**
	RR :	-	90.75	93.13	82.96	**95.75**	95.13
	PC :	-	0.89	0.13	3.45	0.52	**0.10**
FA	Acc :	**87.91**	83.78	82.31	87.07	84.49	84.36
	CC :	9.72	2.89	2.53	4.80	**2.08**	2.22
	RR :	-	70.23	74.01	50.58	**78.59**	77.21
	PC :	-	30.36	5.95	50.91	13.16	**5.30**
TP	Acc :	**97.56**	93.52	91.38	96.66	94.34	94.48
	CC :	48.00	8.22	6.86	20.42	6.69	**5.39**
	RR :	-	82.89	85.72	57.45	86.06	**88.77**
	PC :	-	103.86	17.34	196.00	17.63	**14.56**
YG	Acc :	**92.36**	90.39	88.03	91.03	90.03	89.67
	CC :	20.91	4.41	3.50	6.71	3.13	**3.12**
	RR :	-	78.91	83.26	67.90	85.02	**85.06**
	PC :	-	52.23	8.04	110.56	**4.26**	7.30
WF	Acc :	**99.79**	99.62	99.51	99.40	99.25	99.50
	CC :	98.55	1.21	1.01	1.86	1.01	**0.99**
	RR :	-	98.77	98.97	98.11	98.97	**99.00**
	PC :	-	15.63	2.57	495.63	4.64	**2.44**
SL	Acc :	**52.62**	49.07	48.62	51.20	51.20	49.78
	CC :	2.43	1.54	1.37	1.78	**1.32**	1.35
	RR :	-	36.76	43.67	26.69	**45.69**	44.40
	PC :	-	11.33	2.86	56.00	4.99	**2.84**
CBF	Acc :	**100.00**	99.57	99.35	99.68	99.57	99.46
	CC :	1.66	0.06	0.06	0.12	0.04	**0.04**
	RR :	-	96.34	96.56	92.63	97.47	**97.55**
	PC :	-	0.66	0.19	7.32	0.70	**0.14**
Avg	Acc :	**89.82**	87.57	86.43	89.12	88.15	87.96
	CC :	25.99	2.63	2.20	5.12	2.04	**1.88**
	RR :	-	79.24	82.19	68.05	**83.94**	83.87
	PC :	-	30.71	5.30	131.44	6.56	**4.67**

and effective time-series classifiers by combining prototype selection or abstraction algorithms with time-series dimensionality reduction representations.

It is worth mentioning that the three prototype abstraction algorithms, RSP3, RHC and AIB2, achieved higher classification accuracy than the conventional 1-NN. In the case of the SC dataset, accuracy improvement was very high. Almost in all cases, RSP3 achieved the highest accuracy. However, it is the slowest method in terms of both preprocessing and classification cost (RSP3 had the lowest reduction rates). The high preprocessing cost measurements are attributed to the costly procedure for finding the most distant items in each created subset (see Subsection 2.1.4 for details).

RHC, AIB2 and IB2 had much lower preprocessing cost than the other two methods. This happened because IB2 and AIB2 are one-pass algorithms and RHC is based on a version of k-means that is sped-up by the class mean initializations (see Subsection 3.2.2 for details). In addition, RHC builds the smallest condensing sets. In all cases, RHC achieved higher reduction rates than the other DRTs. Thus, the corresponding classifiers had the lowest classification costs.

The classification accuracy achieved by RHC was usually higher than IB2 and CNN-rule and as high as AIB2. In some cases, RHC and AIB2 were more accurate than RSP3. Considering the above, one may conclude that, since RHC and AIB2 deal with all comparison criteria, they are efficient and effective speed-up methods for time-series data. Finally, the experimental results illustrate that AIB2 is an efficient variation of IB2. In all cases, AIB2 achieves higher performance than IB2.

None of the algorithms can be said to comprise the best speed-up choice. If classification accuracy is the most critical criterion, RSP3 may be preferable. On the other hand, if fast classification and/or fast construction of the condensing set are more critical than accuracy, RHC or AIB2 may be a better choice.

6.2.4 Conclusions

Efficient and effective time-series classification is an open research issue that has attracted the interest of the data mining community. This section proposed the use of non-parametric state-of-the-art prototype selection and abstraction algorithms for efficient and effective time-series classification.

The experimental study conducted demonstrates that by combining prototype selection or abstraction algorithms with dimensionality reduction, one can obtain accurate and very fast time-series classifiers. In addition, the study reveals that prototype abstraction algorithms are preferable to prototype selection algorithms when applied on time-series data. The prototype abstraction algorithms examined in the study can achieve accuracy higher than the conventional 1-NN classifier.

6.3 Fast and accurate *k*-NN classification using prototype selection by clustering

6.3.1 Motivation and contribution

Prototype Selection by Clustering (PSC) is a condensing algorithm that has been recently proposed by Lopez et al. [86, 85, 87]. Its main goal is the fast construction of the condensing set (low pre-processing cost). High reduction rate and classification accuracy continue to be desirable but constitute secondary goals. PSC is based on cluster generation. The main goal of PSC is achieved by the creation of a small number of clusters for each class.

The motivation of the work presented in this section is to examine (a) whether the creation of a larger number of clusters can improve the classification accuracy and the reduction rate of PSC algorithm, and, (b) how the goal of low pre-processing cost gets affected. The contribution is an extensive experimental study that compares our improved version of PSC [93] with two state-of-the-art DRTs, the condensing algorithm CNN-rule and the prototype abstraction algorithm RSP3.

The rest of this section is organised as follows. Subsection 6.3.2 presents the PSC algorithm and explores how the construction of multiple clusters can improve its performance. Subsection 6.3.3 presents the experimental results, and Subsection 6.3.4 concludes the section.

6.3.2 Prototype Selection by Clustering

PSC achieves the goal of the low preprocessing cost by creating a small number of clusters by using the fast and well-known k-means clustering (see Subsection 2.4). In effect, it tries to keep the close-class-border items as well as some items that lie in non-close-border data area.

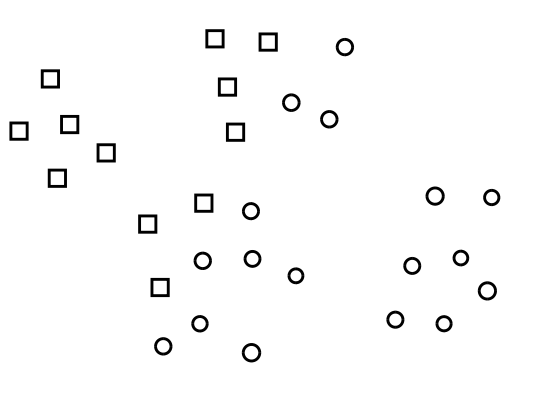
PSC is based on the following simple idea: homogeneous clusters (i.e., clusters that contain items of a specific class) contain items that lie in non-close-border data areas. On the other hand, non-homogeneous clusters contain close-border items. Initially, PSC uses k-means clustering in order to partition the training data into clusters. For each homogeneous cluster, the nearest to the cluster mean item is put in the condensing set as prototype. For each non-homogeneous cluster, the items that define the decision boundaries are placed into the condensing set.

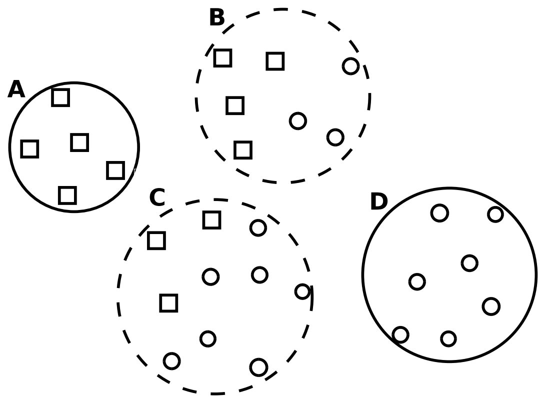
More formally, PSC, initially, creates $|C|$ clusters, C_i where $i = 1, 2, \ldots, |C|$. For each homogeneous cluster C_i, PSC places the nearest item $p \in C_i$ to the cluster mean in the condensing set. This item is the prototype that represents the whole data area of that cluster and is called non-border prototype. For each non-homogeneous cluster C_i, PSC chooses a set of prototypes as follows: Initially, it finds the majority class T_M in C_i. Then, for each item $p_j \in T_i$, $i \neq M$, it puts in the condensing set the item $p_M \in T_M$ that is the nearest to $p_j \in T_i$. Also, it puts in the condensing set, item $p_{C_i} \in T_i$ that is the nearest to p_M (p_{C_i} may be different than p_j). The prototypes collected from a non-homogeneous cluster are called border prototypes.

The PSC routine is summarized in Figure 6.1. Initially k-means clustering identifies four clusters in the training data. Clusters A and D are homogeneous. For these clusters, PSC keeps the nearest items to the cluster means as non-border prototypes. They represent the corresponding clusters data area. On the other hand, Clusters B and C are non-homogeneous. Thus, PSC analyses the items of the clusters and keeps only the border prototypes by applying the methodology described in the previous paragraph.

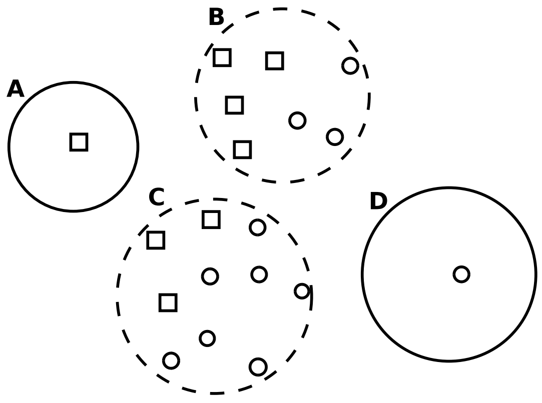
Of course, the selected number of border and non-border prototypes depends on the number of clusters that are initially created ($|C|$). The higher the $|C|$ value, the more homogeneous clusters are generated and the more non-border prototypes are collected. In contrast, the larger the clusters, the more border prototypes selected and the lower reduction rate achieved. Lopez et al. considered a small number of clusters in order to achieve fast execution of the algorithm. In particular, in their experiments [86], they built only $r \times j$, $j = 2, 4, \cdots, 10$, clusters, where r is the number of discrete classes. We claim that a larger number of clusters could improve the classification performance in terms of classification accuracy and reduction rate.

It is worth mentioning that the input parameter of PSC does not allow the user to control the size of the condensing set (number of prototypes). The parameter concerns the number of the clusters that will be created. The number of the prototypes collected depends on how the data is distributed in the data space (level of noise, overlaps between classes, shape of


(a) training data



(b) clustered training data



(c) finding non-border prototypes

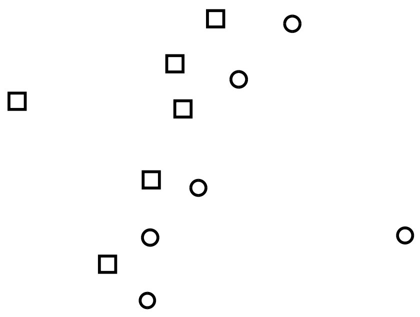

(d) finding border prototypes

Figure 6.1: Prototype Selection by Clustering

the class data region, etc). Although the condensing set built by PSC does not depend on the order of items in the training set, the choice of the initial means for k-means clustering affect the contents of the condensing set. Therefore, different condensing sets built by selecting different initial means. Please note that an implementation of PSC is available in WebDR[3] (see Appendix A) and can be executed on-line.

6.3.3 Performance evaluation

Experimental setup

We compared the performance of CNN-rule, RSP3 and PSC by applying the k-NN classifier on the condensing sets they generate. In all cases, we used the k value that achieved the highest classification accuracy. Possible ties during nearest neighbour voting (two or more classes collecting the same highest number of votes) were resolved by choosing the class of the nearest neighbour. We should mentioned that the value of the k parameter was not tuned by the typical tuning procedure that implies that the best parameter value should be obtained by using only the training set. Since the three algorithms use only the k parameter during the classification step, for all of them, we simply report the highest accuracy achieved by the k parameter when it classifies the testing data using the corresponding training data.

We used a five-fold cross-validation schema on six datasets distributed by the KEEL Dataset Repository[4][6]. Thus, we run five training/testing set experiments for each dataset and each algorithm and we report the averages. Of course, only the training sets was preprocessed by the algorithms. We used the five already constructed pairs of training/testing splits distributed by the KEEL repository. The six datasets are summarized in Table 6.4. All algorithm runs were executed on the original datasets, i.e., without normalization. Moreover, we used the Euclidean distance as the distance metric.

The three algorithms are compared by estimating three metrics: (i) accuracy, (ii) reduction rate and (iii) preprocessing cost in terms of million distance computations (we counted the distances computed during the procedure of the condensing set construction). For each dataset, we present one diagram for each metric. For PSC, we built 24 condensing sets. Each one was built by using different number of clusters, $k = r \times CL$ clusters, where r is the number of discrete classes. CL takes 25 different values: $CL = 2, 4, 6, 8, 10, 20, \cdots, 190, 200$. The x-axis

[3]`https://ilust.uom.gr/webdr`

[4]`http://sci2s.ugr.es/keel/datasets.php`

Table 6.4: Datasets description

dataset	Size	Attributes	Classes
Letter Recognition (LIR)	20000	16	26
Pen-Digits (PD)	10992	16	10
Landsat Satellite (LS)	6435	36	6
Shuttle (SH)	58000	9	7
Texture (TXR)	5500	40	11
Phoneme (PH)	5404	5	2

of each comparison diagram represents the CL values. CNN-rule and RSP3 are not parametric approaches and so their performance is not affected when varying the CL value.

Experimental measurements

Figures 6.2-6.7 present the comparison measurements of the three methods on the six datasets. Each figure includes three diagrams, one for each metric, i.e., accuracy., reduction rate and preprocessing cost. The accuracy diagrams include one extra curve for the measurements achieved by the conventional k-NN classifier (Conv-k-NN), i.e., k-NN over the original training data (without data reduction).

In all cases 6.2-6.7, CNN-rule executed faster and achieved higher reduction rate than RSP3. On the other hand, with the exception of the PH dataset, RSP3 achieved higher accuracy measurements than CNN-rule. In some cases, the accuracy of RSP3 is close to the one of Conv-k-NN.

With the exception of the PH dataset (Figure 6.7), PSC achieves the highest reduction rates in all datasets when $10 \leq CL \leq 50$. This means that the corresponding k-NN classifiers executed faster than the classifiers built using the rest of the CL values. In the case of the PH dataset, reduction rate continues to improve with higher CL values.

Moreover, in the cases of the LIR (Figure 6.2), PD (Figure 6.3), LS (Figure 6.4), and TXR (Figure 6.6) datasets, for $CL \leq 50$, the preprocessing cost measurements of PSC were lower than or close to those of RSP3. In the case of the SH dataset (Figure 6.5), which is the largest one, RSP3 generates its condensing set at an extremely high computational preprocessing cost. This is the result of the farthest point computations in the subsets created during RSP3 execution (see Subsection 2.1.4).

In the cases of the LS (Figure 6.4) and SH (Figure 6.5) datasets, PSC could not reach the accuracy levels of the other two methods, but it was quite close. In all other datasets (Figures 6.2, 6.3, 6.6, 6.7), PSC achieved higher accuracy measurements than CNN-rule and RSP3. Furthermore, the PSC classifiers with condensing sets built using CL values greater than ten, were more accurate and achieved higher reduction rate than those built by lower CL values (Lopez et al. case [86]). However, the generation of these condensing sets was an "expensive" procedure since it computed more distances. For non-dynamic environments, there is no need for periodical condensing set reconstruction. Thus, we claim that these measurements may not be so significant since the condensing set is built only once.

We conclude that PSC is an adaptive algorithm that can be used either for fast condensing set generation but with lower reduction rate and accuracy (this is the scenario presented by lopez et al.), or for accurate and fast k-NN classification but with "expensive" condensing set generation. The desirable performance can be achieved by tuning the CL parameter.

6.3.4 Conclusions

In this section, we compared three known DRTs, namely, CNN-rule, RSP3, and PSC. In addition, we demonstrated how the creation of a large number of clusters can improve the performance of PSC. The experimental measurements derived by a cross-validation schema on six datasets indicate that PSC can reach and exceed the classification performance of the other two state-of-the-art algorithms. Therefore, it can be used either when fast execution of the data reduction procedure is required, or when reduction rate and/or classification accuracy are more critical than the preprocessing cost. The desirable trade-off between preprocessing cost and accuracy/reduction rate can be defined by appropriately adjusting the parameter that defines the number of clusters that will be created.

6.4 An extensive experimental study on Hwang and Cho method

6.4.1 Motivation and contribution

Hwang and Cho have proposed a cluster-based method for fast k-NN classification [62]. It is an adaptive approach which provides three parameters. Its effectiveness depends on the

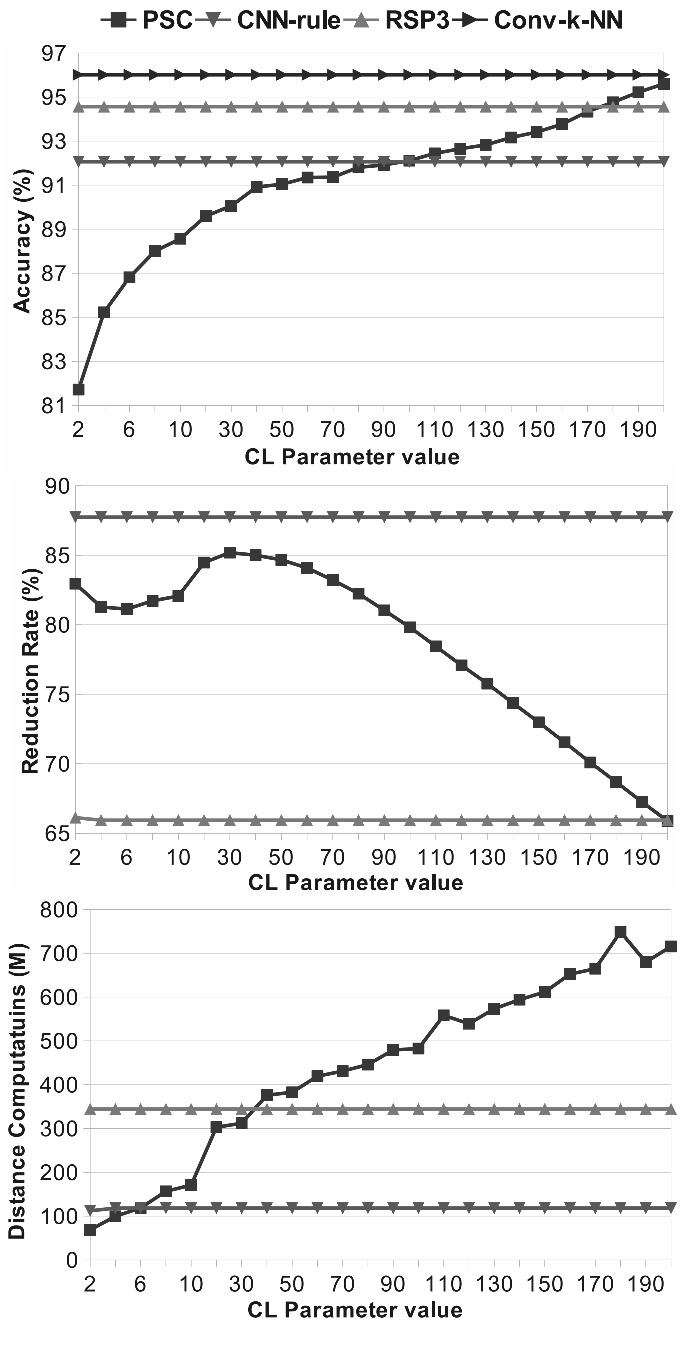


Figure 6.2: LIR (Accuracy, Reduction Rate, Preprocessing Cost)

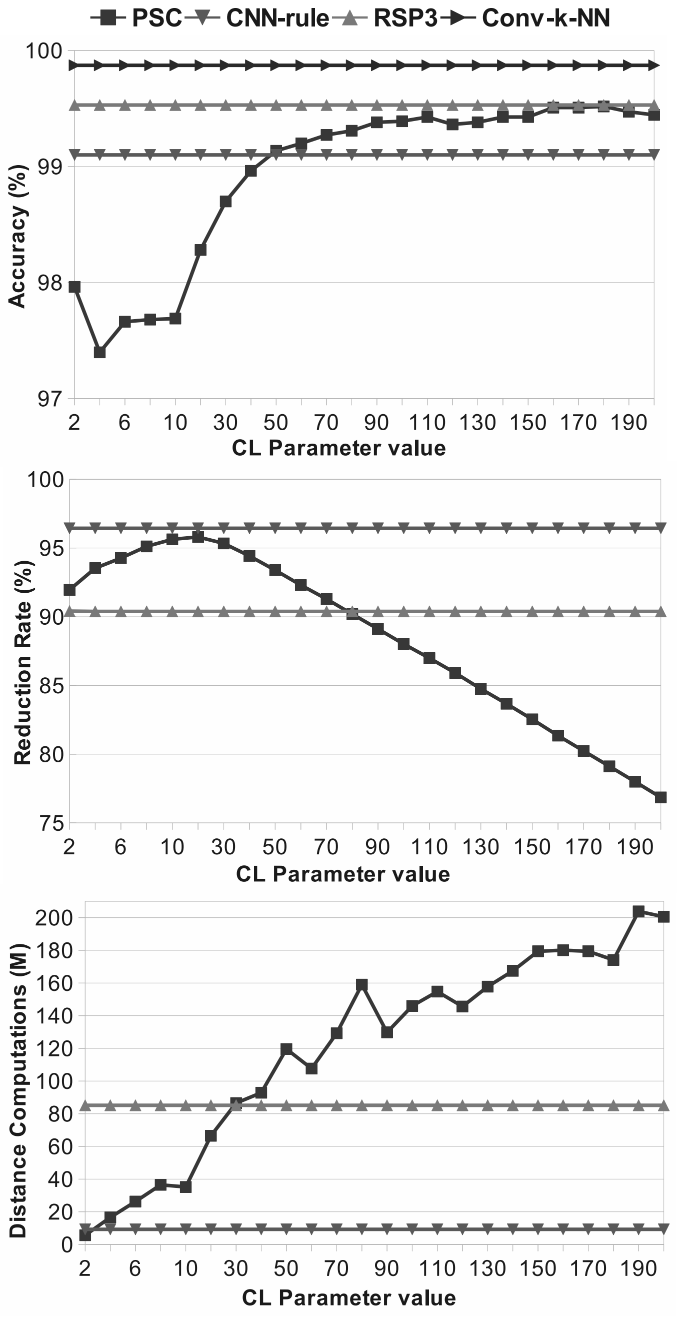


Figure 6.3: PD (Accuracy, Reduction Rate, Preprocessing Cost)

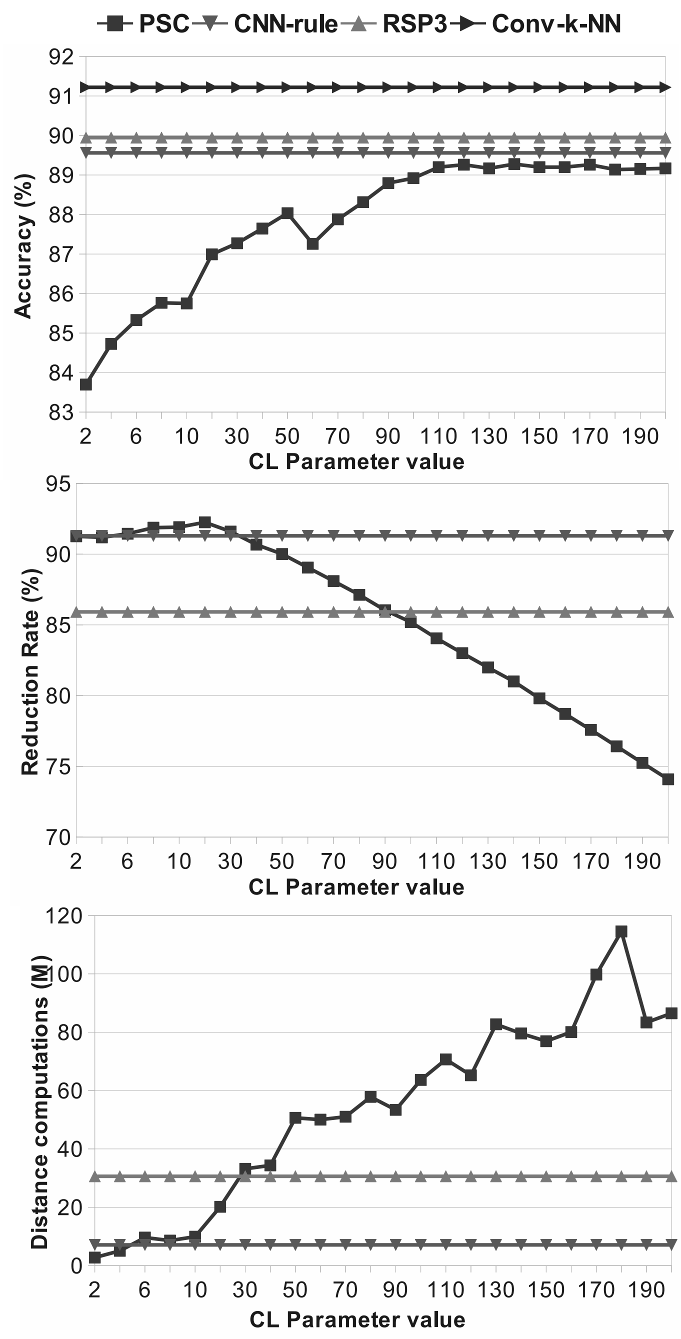


Figure 6.4: LS (Accuracy, Reduction Rate, Preprocessing Cost)

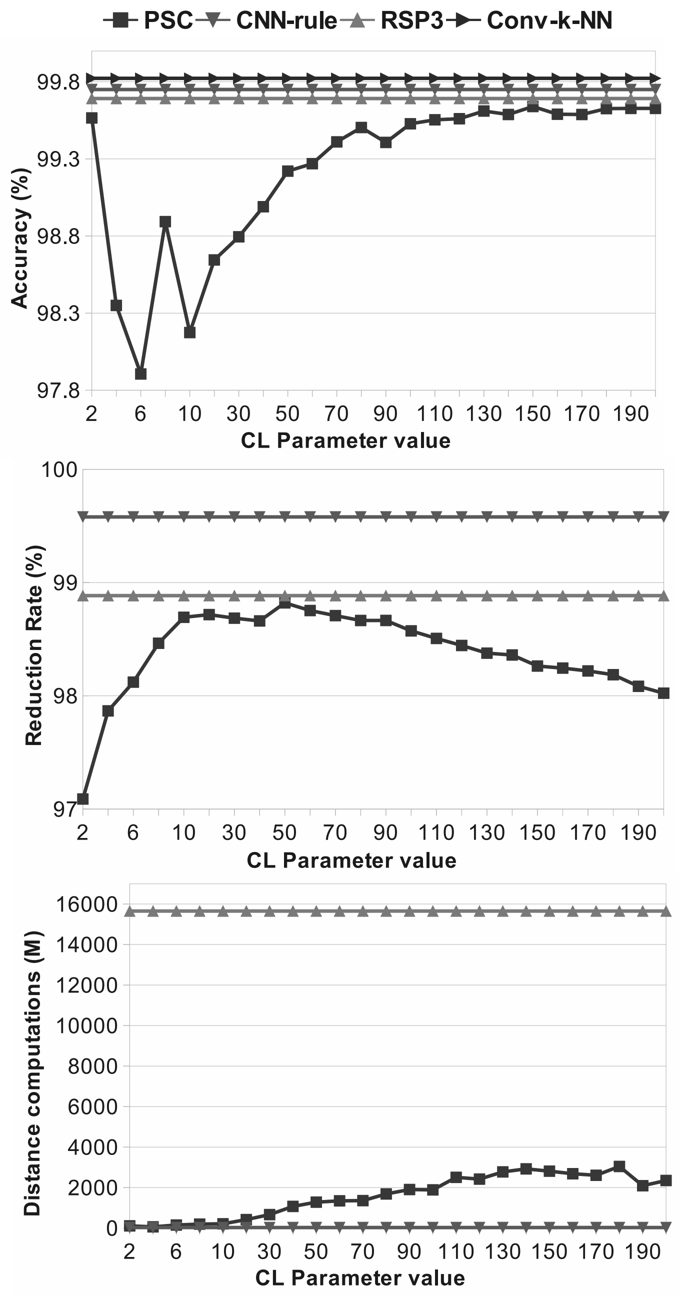


Figure 6.5: SH (Accuracy, Reduction Rate, Preprocessing Cost)

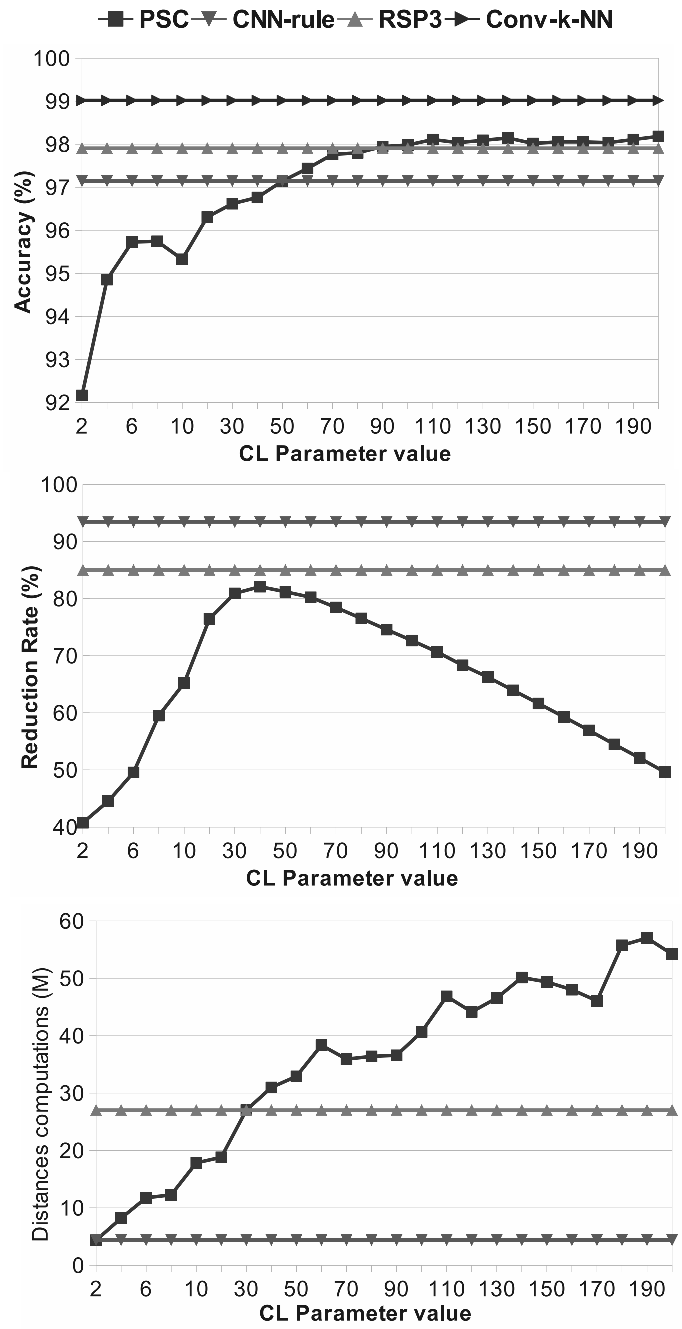


Figure 6.6: TXR (Accuracy, Reduction Rate, Preprocessing Cost)

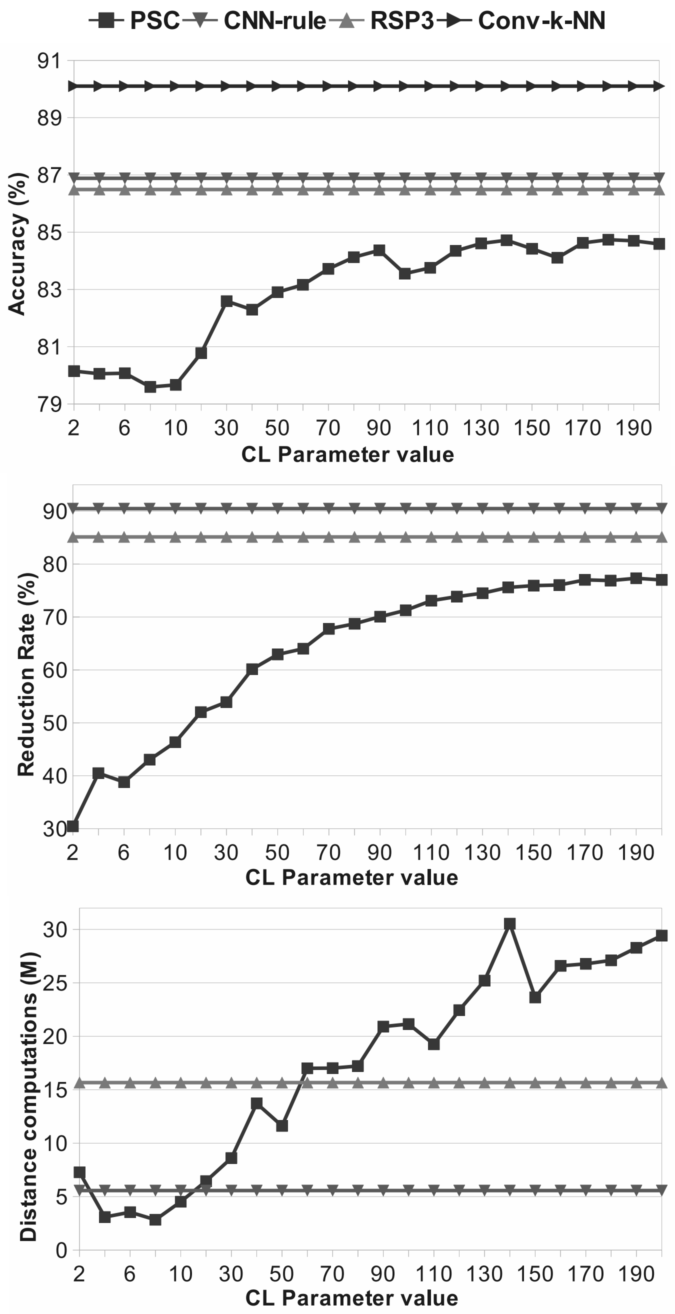


Figure 6.7: PH (Accuracy, Reduction Rate, Preprocessing Cost)

adjustment of these parameters. Hwang and Cho presented experimental results obtained by specific parameter values and based on only one dataset. Moreover, they did not use the Euclidean distance as the distance metric. These observations constitute the motivation of the work presented in this section. The contribution of this work is an extensive experimental study on this method. It includes experiments on five datasets using different parameter values. Also, we use the Euclidean distance as the distance metric.

The rest of this section is organized as follows. Section 6.4.2 considers in detail the Hwang and Cho method and discusses the adaptive schema that it provides. In Section 6.4.3, we present an extensive experimental study based on five datasets. The work concludes in Section 6.4.4.

6.4.2 The Hwang and Cho method

The Hwang and Cho method (HCM) is an effective speed-up approach for k-NN classification. The method is outlined in Algorithms 23 and 24 . During preprocessing, it uses the well-known k-means clustering (see Section 2.4) to find clusters in the training set (TS) (lines 1–2 in Algorithm 23). Afterwords, each one cluster is divided into two sets that are called "peripheral set" (ps) and "core set" (cs). In particular, the cluster items lying within a certain distance from the cluster mean (centroid) are placed into cs, while the rest, more distant from the centroid, items are placed into the ps (lines 3–14 in Algorithm 23).

When a new item x must be classified (classification step, see Algorithm 24), the method finds the nearest cluster C_1. If x lies within the core area of C_1 (line 3), it is classified by retrieving its k nearest neighbours from C_1 (lines 4,8–10). Otherwise, the k nearest neighbours are retrieved from the reference set R formed by the items of the nearest cluster and the "peripheral" items of the L most adjacent clusters (lines 6,8–10).

If the clusters were not divided and only the items of the nearest cluster were used to classify the new item (regardless of how distant from the centroid it was), many training items in the nearby clusters would be ignored. Therefore, Hwang and Cho proposed the use of some nearby clusters as a safer approach. The main innovation in their method is that it uses only the peripheral items of these additional adjacent clusters. If all items (not only the peripheral) of these clusters were used, the classification computational cost would have been much higher.

A key factor of HCM is the determination of the threshold that defines which items will be core and which peripheral. This is very critical since it determines how many items are accessed during classification. Hwang and Cho consider as peripheral items, those whose distance from the cluster centroid is greater than the double average distance among the items of each cluster. Consequently, the average distance among the items in each cluster and the corresponding cluster centroid must be computed (line 8 in Algorithm 23, line 3 in Algorithm 24).

In this study, we do not use a particular threshold as Hwang and Cho did in their experiments (they used the double average distance). We introduce parameter D to be responsible for the splitting of the clusters into core and peripheral sets. An item x is placed into the peripheral set of cluster C, if:

$$Distance(x, centroid of C) > D \times C_{AvgDist}$$

For example, if $D = 1.5$, the "peripheral sets" contain items that are more than 1.5 times the average distance away from the cluster centroid. The determination of D is a critical issue and it should be made by considering the available number of clusters and the desirable trade-off between classification accuracy and classification cost.

Another issue that must be addressed is related to the number of clusters that are constructed (determination of the k parameter in k-means clustering) and the number of adjacent clusters that are examined when the new item lies within the peripheral area of the nearest cluster (L parameter). Hwang and Cho empirically define $k = 10$ and $L = \lfloor\sqrt{k}\rfloor$ for the dataset they used in their experimental study.

6.4.3 Performance evaluation

The experimental study was conducted using five datasets distributed by the UCI Machine Learning Repository[5] [12, 44]. The datasets are presented in Table 6.5. The fifth column lists the k value found to achieve the highest accuracy when using the k-NN classifier to classify the testing data by scanning the whole training set (conv-k-NN). The computational cost was estimated by counting the distance computations needed to carry out the whole classification task. Contrary to Hwang and Cho, we estimated all distances using the Euclidean distance. All datasets were used without data normalization or any other transformation. Also, in all HCM experiments, we chose the k values of the k-NN classifier that achieved highest classification

[5] `http://archive.ics.uci.edu/ml/`

Algorithm 23 HCM-preprocessing

Input:TS, D, k

Output:$CLUSTERS$

```
 1: $initial\_means \leftarrow$ use the first $k$ items of $TS$ as initial means
 2: $CLUSTERS \leftarrow K-$MEANS($TS$, $initial\_means$)
 3: for each cluster $C \in CLUSTERS$ do
 4:     $C_{AvgDist} \leftarrow$ Compute the average distance of the items in $C$ from the corresponding
        cluster centroid
 5:     $C_{cs} \leftarrow \varnothing$
 6:     $C_{ps} \leftarrow \varnothing$
 7:     for each item $t_i$ in $C$ do
 8:         if Distance($t_i$, Centroid of $C$) $\leq D \times C_{AvgDist}$ then
 9:             $C_{cs} \leftarrow C_{cs} \cup \{t_i\}$
10:         else
11:             $C_{ps} \leftarrow C_{ps} \cup \{t_i\}$
12:         end if
13:     end for
14: end for
15: return ($CLUSTERS$)
```

Algorithm 24 HCM-classification

Input:$CLUSTERS, k, D, L$

```
 1: for each unclassified item $x$ do
 2:     Find $L$ nearest to $x$ clusters (based on clusters centroids), $C_1, C_2, \ldots, C_L$ where $C_1$ is
        the nearest, $C_2$ is the second nearest and so on
 3:     if Distance($x$, Centroid of $C_1$) $\leq D \times C_{AvgDist}$ then
 4:         $R \leftarrow C_1$
 5:     else
 6:         $R \leftarrow C_1 \cup C_{2_{ps}} \cup C_{3_{ps}} \cup \ldots \cup C_{L_{ps}}$
 7:     end if
 8:     Retrieve the $k$ Nearest Neighbours from $R$
 9:     Find the major class (the most common one among the $k$ Nearest Neighbours)
10:     Classify $x$ to the major class
11: end for
```

Table 6.5: Datasets description and performance of the conventional k-NN classifier (Accuracy (%) and Computational Cost (millions of distance computations))

Dataset	**Train/Test dataset size**	**Attributes**	**Classes**	**Best k**	**Accuracy (%)**	**Cost**
Letter Image Recognition (LIR)	15000/5000	16	26	4	95.68	75
Magic Gamma Telescope (MGT)	14000/5020	10	2	12	81.39	70.28
Pendigits (PD)	7494/3498	16	10	4	97.89	26.21
Landsat Sattelite (LS)	4435/2000	36	6	4	90.75	8.87
Shuttle (SH)	43500/14500	9	7	2	99.88	630.75

accuracy. We resolved possible ties during class voting by selecting the class of the nearest neighbour.

We define $L = \lfloor\sqrt{k}\rfloor$ as Hwang and Cho did in their experiment. Concerning the k parameter that determines the number of clusters that are formed, we built 8 classifiers for each dataset. Classifier$_i$ uses $k = \lfloor\sqrt{\frac{n}{2^i}}\rfloor$ clusters, $i = 1, \dots, 8$, where n is the number of items in the training set. Classifier$_1$ is based on the rule of thumb that defines $k = \lfloor\sqrt{\frac{n}{2}}\rfloor$ [78]. We decided to build classifiers that use low k values based on the observation that Hwang and Cho set $k = 10$ for a training set with 60919 items. For each classifier, we chose a varying value for D (1, 1.5, and 2). Thus, we built and evaluated $8 \times 3 = 24$ classifiers for each dataset.

In Figures 6.8–6.12, for each dataset, the performance of the most accurate classifiers for a given cost are presented[2]. The figures do not present the performance of conv-k-NN that is mentioned in Table 6.5. In particular, in Figures 6.8–6.12, the classifiers built by the three D values (1, 1.5 and 2) are compared to each other.

For the first three datasets (Figures 6.8–6.10), the classifiers built for $D = 1$ seem to perform better than the ones built for $D = 1.5$ and $D = 2$. In the cases of the LIR and MGT datasets, the superiority of the classifiers$_{D=1}$ is obvious. In the case of the LIR dataset, the two classifiers$_{D=1}$ presented in Figure 6.8 are build by setting $k = \lfloor\sqrt{\frac{15000}{2^1}}\rfloor = 86$, $L = \lfloor\sqrt{86}\rfloor = 9$ and $k = \lfloor\sqrt{\frac{15000}{2^5}}\rfloor = 21$, $L = \lfloor\sqrt{21}\rfloor = 4$, respectively. In the MGT dataset, the parameter values of the most accurate classifier are $D = 1$, $k = 59$ and $L = 7$. Finally, in the PD dataset, the fastest and slowest classifier$_{D=1}$ is built by setting $k = 61$ and $k = 15$ respectively.

For the LS and SH datasets (Figure 6.11 – 6.12) there is not a dominant D parameter value in terms of performance and accuracy. In the LS dataset, the most accurate classifier is built

[2]Detailed experimental results available at:`http://users.uom.gr/~stoug/RSRM.zip`

by setting $D = 1$ and $k = 16$, while the fastest classifier that achieves an accuracy value over 89.2% is built using $D = 1.5$ and $k = 23$. In the SH dataset, the results are more confusing. This is because the SH dataset is an imbalanced (skewed) dataset (approximately 80% of the items belong to one class). However, in the SH dataset, all classifiers presented in Figure 6.12 manage to achieve higher accuracy than that of the conv-k-NN.

6.4.4 Conclusions

In this section, we presented an extensive experimental study on the Hwang and Cho method. In all experiments we used the Euclidean distance. The classification performance of the method depends on the determination of k (number of clusters built) and D parameters. In all cases, they should be adjusted by taking into consideration the application domain and the desirable trade-off between classification accuracy and classification cost. The experimental measurements indicate that if accuracy is more critical than cost, low D and high k and L values (e.g. $D = 1$) lead to an efficient classification method. On the other hand, if cost is more critical than accuracy, higher D and lower k and L values may be more appropriate.

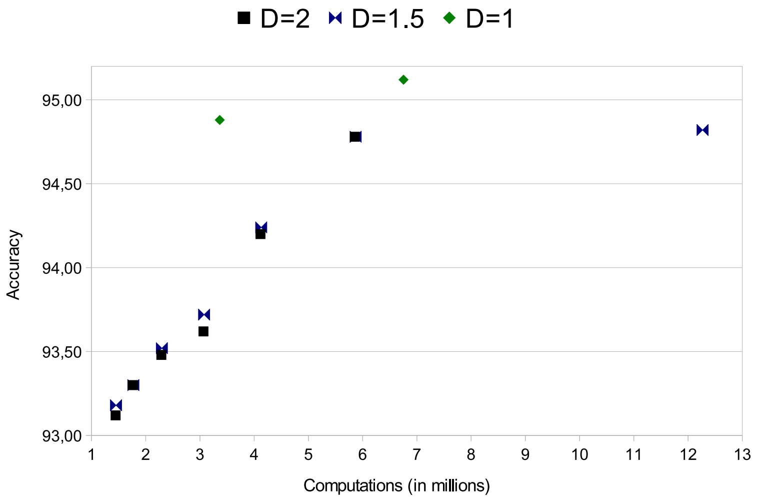


Figure 6.8: LIR (Accuracy and Computational Cost)

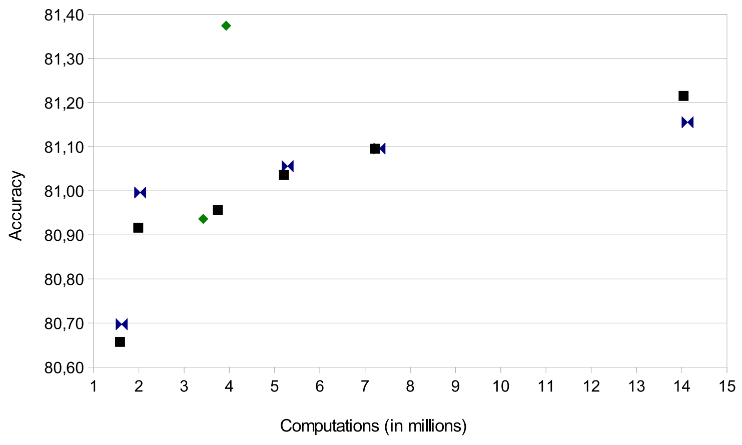


Figure 6.9: MGT (Accuracy and Computational Cost)

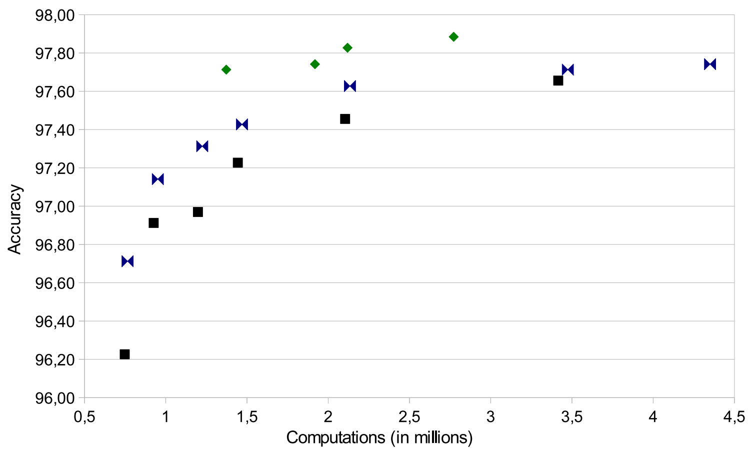


Figure 6.10: PD (Accuracy and Computational Cost)

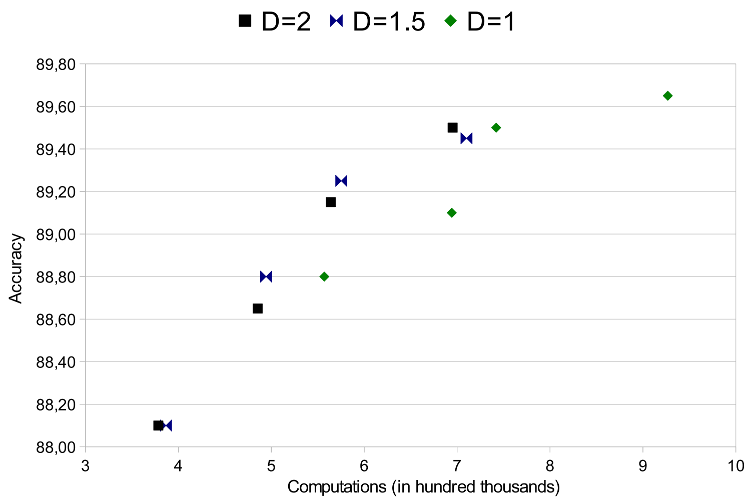


Figure 6.11: LS (Accuracy and Computational Cost)

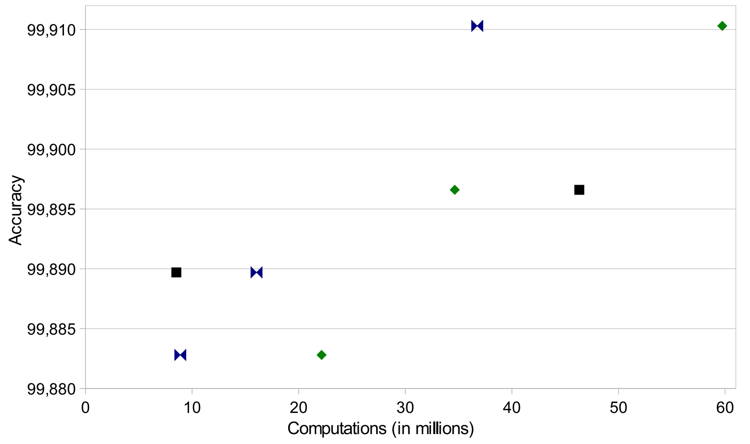


Figure 6.12: SH (Accuracy and Computational Cost)

Chapter 7

Conclusions and future work

Improving the performance of the k-NN classifier is an active research problem. In the context of this problem, the dissertation introduced various algorithms and techniques as well as enhancements for existing methods. The proposed approaches can effectively deal with the drawbacks and the weaknesses of the classifier. The main findings and contribution of the conducted research are summarized below and directions for future work are suggested.

A major part of the dissertation deals with the concept of data reduction for efficient and effective k-NN classification. In that context, the dissertation presented novel Data Reduction Techniques (DRTs) as well as improvements and experimentations for existing techniques. In Chapter 3, the family of DRTs based on homogeneous clusters and the corresponding experiments were presented in detail. The family includes four algorithms, namely, RHC, dRHC, EHC, and, ERHC. Although each algorithm aims to a specific goal, they all follow the common strategy of forming homogeneous clusters (clusters that contain items of a specific class). Main motives behind the four algorithms constitute the fast and non-parametric (independent of tuning parameters via trial-and-error procedures) preprocessing of the training set. Certainly, high accuracy and reduction rates are also important goals.

RHC is a simple "general purpose" prototype abstraction algorithm that achieves high performance. dRHC is an incremental variation of RHC. It incrementally constructs its condensing set. Consequently, dRHC is appropriate for dynamic / streaming domains where new training data is gradually available. Moreover, dRHC effectively manages large datasets that can not fit in main memory and/or it can be executed on devices with limited main memory (e.g., sensor devices). EHC is an editing algorithm. It aims to improve the quality of the training data - and as a consequence the classification accuracy - rather than speeding-up classification. This is accomplished through fast and non-parametric preprocessing that removes outliers and noisy, mislabelled and close-border training items. Last but not least, ERHC is a simple variation of RHC that integrates the idea of EHC editing. Therefore, it can achieve high reduction rates regardless the level of noise in the training data and requiring as high preprocessing cost as RHC.

In Chapter 4 two prototype abstraction algorithms were presented. First, a prototype abstraction variation of the IB2 condensing algorithm [5, 4], called AIB2, was investigated. AIB2 inherits all the properties of IB2 but achieves improved classification performance. It is based on the idea of re-positioning prototypes in order to lie in the center of the data area they represent. Then, a noise-tolerant prototype abstraction algorithm was presented. It was shown that a condensing set that stores only the means of the clusters built by k-means clustering [79, 133] iteratively executed on the set of training items belonging to each class contributes toward noise tolerance during the classification process.

Research on data reduction also occupies two sections of Chapter 6. Section 6.3 introduced improvements for the recently proposed PSC condensing algorithm [86, 85, 87]. In particular, it demonstrated that the performance of PSC can be improved if a high number of clusters is built during preprocessing. Section 6.2 demonstrated that typical non-parametric DRTs can be effectively applied on time series data. The experimental study on several times-series datasets showed that the prototype abstraction algorithms can achieve even higher accuracy than the conventional k-NN classifier. Furthermore, in Section 5.4.3, we presented a simple cluster-based method that improves the performance of DRTs by skipping redundant distance computations, between unclassified items and prototypes that lie far from them during the classification step. The proposed method performs classification tasks faster than DRTs, without loss of accuracy and by requiring an extremely small extra preprocessing overhead.

WebDR[1] (see Appendix A) is also a contribution of the dissertation. It allows users to execute data reduction experiments on-line via an interactive web interface. All the contributed DRTs as well as the DRTs that have been implemented for comparison purposes during the PhD research have been integrated in the particular web application. All the conducted experiments related to data reduction can be verified using WebDR.

In Chapter 5, hybrid methods for fast and accurate k-NN classification were presented. The methods do not reduce the size of the training data since they require that training items

[1]`https://ilust.uom.gr/webdr`

are always available. In effect, the proposed methods combine different speed-up approaches in order to accelerate the classification. Initially, we introduced a hybrid classifier and two variations that combine the conventional k-NN and the minimum distance [38] classifiers. The main advantage is that the proposed classifiers are model free, i.e., they do not require expensive preprocessing on the training data. Then, a classification method that combines the idea of data reduction with that of cluster-based methods was proposed. The method consists of a preprocessing algorithm that builds a two-level data structure and efficient algorithms that access this structure in order to perform fast and accurate classification. Finally, a hybrid, non-parametric method that extends the aforementioned idea was presented. It is based on forming homogeneous clusters and constructing a speed-up data structure.

In Section 6.4, we proposed enhancements for the Hwang and Cho method [62]. We demonstrated that if the input parameters are carefully determined, one can obtain better classification performance than the one achieved in the study of Hwang and Cho.

In all cases, the proposed algorithms were experimentally evaluated on known datasets and compared to popular speed-up methods. In addition, in many cases, the experimental results were statistically validated using the Wilcoxon signed ranks test. Through the experimental studies, we demonstrated that the proposed algorithms satisfied the goals for which they had been developed and led to improved classification tasks.

The challenge of big data has emerged new research directions in the context of k-NN classification. Traditional algorithms and techniques for efficient and effective k-NN classification fail to manage fast and/or large data streams [1] with or without concept drifts [125] and complex data (imbalanced datasets, items that belong to more than one classes, etc). Hence, directions for future work could be:

- Development of non-parametric one-pass DRTs that take into account the phenomenon of concept drift that may exist in data streams.
- Enhancements and modifications on existing algorithms and techniques so that they can cope with large and fast data streams (with or without concept drift).
- Parallel implementations of DRTs for fast construction of condensing sets.
- Development of DRTs that can be applied in complex problems such as multi-label classification [124, 20].

- DRTs for imbalanced training data (e.g. reduction rates according to the size of each class, avoidance of rare class item reduction, oversampling of rare and weak classes, etc).

Appendix A

WebDR: A Web workbench for Data Reduction

A.1 System description

All contributed Data Reduction Techniques (DRTs) (i.e., RHC, dRHC, EHC, ERHC, AIB2, RkM) as well as the DRTs that were coded and used for comparison purposes during the experimental studies (i.e., CNN-rule, IB2, RSP3, PSC, ENN-rule, All-k-NN, Multiedit) have been integrated in a web application, which we call WebDR (Web workbench for Data Reduction)[1] [99]. In effect, WebDR offers all DRTs implemented in the context of the dissertation available on-line. The motivation behind its development was the absence of software that allows execution of k-NN classification through data reduction over the web.

More specifically, WebDR allows the users to plan and run experiments and measure the classification performance through an interactive web interface over several known datasets distributed by the KEEL[2] [6] or/and the UCI[3] [12, 44] dataset repositories and time-series datasets distributed by the UCR time-series classification/clustering website[4]. All the available datasets can be explored in detail using the "dataset explorer" tool that is available in WebDR (see Figure A.2). The performance of DRTs is evaluated by measuring the three criteria presented in Subsection 2.1.1.

[1] `https://ilust.uom.gr/webdr`

[2] `http://sci2s.ugr.es/keel/datasets.php`

[3] `http://archive.ics.uci.edu/ml/`

[4] `http://www.cs.ucr.edu/~eamonn/time_series_data/`

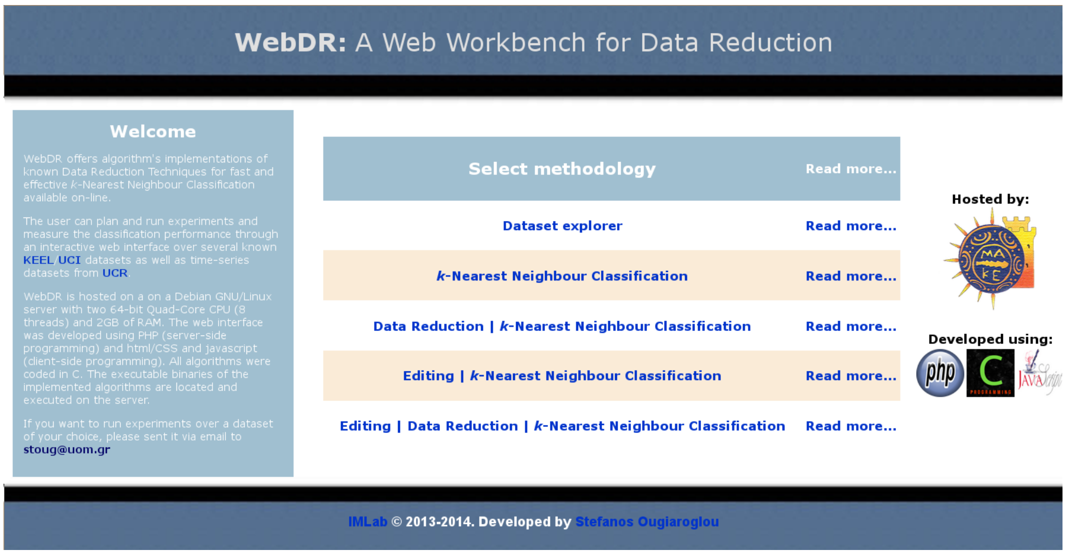
All the possible preprocessing types presented in Subsection 2.1.1 can be executed by WebDR (see Figure 2.1). The main page of the application offers four links (see Figure A.1). Each one leads to the corresponding type of preprocessing. The reported performance measurements are averages obtained via five-fold cross-validation. All datasets built during preprocessing are available to the users in a five-fold form (five pairs of training and testing sets). They can be downloaded and used by the user locally. Of course, the number of the nearest neighbours and the DRT specific parameters (if any) can be adjusted through the interface. Note that WebDR adopts the Euclidean distance as the distance metric.

Currently, WebDR is hosted on a Debian GNU/Linux server with two 64-bit Quad-Core CPUs and 2GB of main memory. All algorithms were coded in C. The web interface was developed using PHP (server-side programming) and html/CSS and javascript (client-side programming). The executable binaries of the implemented algorithms are located and executed on the server.

WebDR aspires to support teaching and research on data reduction. We consider that it can be used by the academia for educational and experimental purposes. In the future, we plan to integrate more DRTs and datasets in WebDR. Moreover, we will develop a mechanism that will allow users to run experiments on their own datasets.

A.2 Case study

The screen-shots presented in Figures A.1– A.6 demonstrate a case study of an experiment through WebDR. More specifically, the figures present a complete data reduction preprocessing procedure, i.e., successive application of editing and data abstraction, over the Landsat satellite dataset (see Figure A.2), the application of the k-NN classifier (with k=1) on the condensing set built and the corresponding performance results. ENN-rule with $k = 3$ is utilized for editing and RHC is used for data abstraction. Please note that the successive execution of both types of preprocessing is not mandatory. WebDR allows the execution of only editing or data condensing / abstraction during preprocessing. Of course, only classification (without preprocessing) is also offered (see Figure A.1).



WebDR: A Web Workbench for Data Reduction

Welcome

WebDR offers algorithm's implementations of known Data Reduction Techniques for fast and effective *k*-Nearest Neighbour Classification available on-line.

The user can plan and run experiments and measure the classification performance through an interactive web interface over several known **KEEL**/**UCI** datasets as well as time-series datasets from **UCR**.

WebDR is hosted on a on a Debian GNU/Linux server with two 64-bit Quad-Core CPU (8 threads) and 2GB of RAM. The web interface was developed using PHP (server-side programming) and html/CSS and javascript (client-side programming). All algorithms were coded in C. The executable binaries of the implemented algorithms are located and executed on the server.

If you want to run experiments over a dataset of your choice, please sent it via email to **stoug@uom.gr**

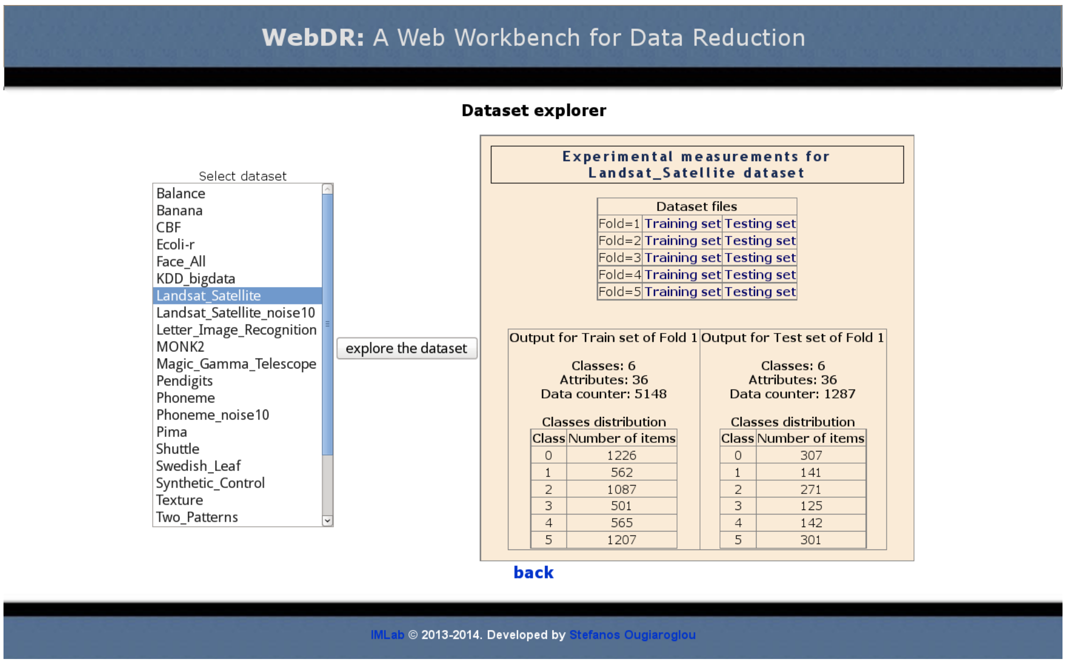
Select methodology	**Read more...**
Dataset explorer	**Read more...**
***k*-Nearest Neighbour Classification**	**Read more...**
Data Reduction | *k*-Nearest Neighbour Classification	**Read more...**
Editing | *k*-Nearest Neighbour Classification	**Read more...**
Editing | Data Reduction | *k*-Nearest Neighbour Classification	**Read more...**

Hosted by:

Developed using:

IMLab © 2013-2014. Developed by Stefanos Ougiaroglou

Figure A.1: Initial page of WebDR



WebDR: A Web Workbench for Data Reduction

Dataset explorer

Select dataset

Balance
Banana
CBF
Ecoli-r
Face_All
KDD_bigdata
Landsat_Satellite
Landsat_Satellite_noise10
Letter_Image_Recognition
MONK2
Magic_Gamma_Telescope
Pendigits
Phoneme
Phoneme_noise10
Pima
Shuttle
Swedish_Leaf
Synthetic_Control
Texture
Two_Patterns

Experimental measurements for Landsat_Satellite dataset

Dataset files		
Fold=1	Training set	Testing set
Fold=2	Training set	Testing set
Fold=3	Training set	Testing set
Fold=4	Training set	Testing set
Fold=5	Training set	Testing set

Output for Train set of Fold 1

Classes: 6
Attributes: 36
Data counter: 5148

Classes distribution

Class	Number of items
0	1226
1	562
2	1087
3	501
4	565
5	1207

Output for Test set of Fold 1

Classes: 6
Attributes: 36
Data counter: 1287

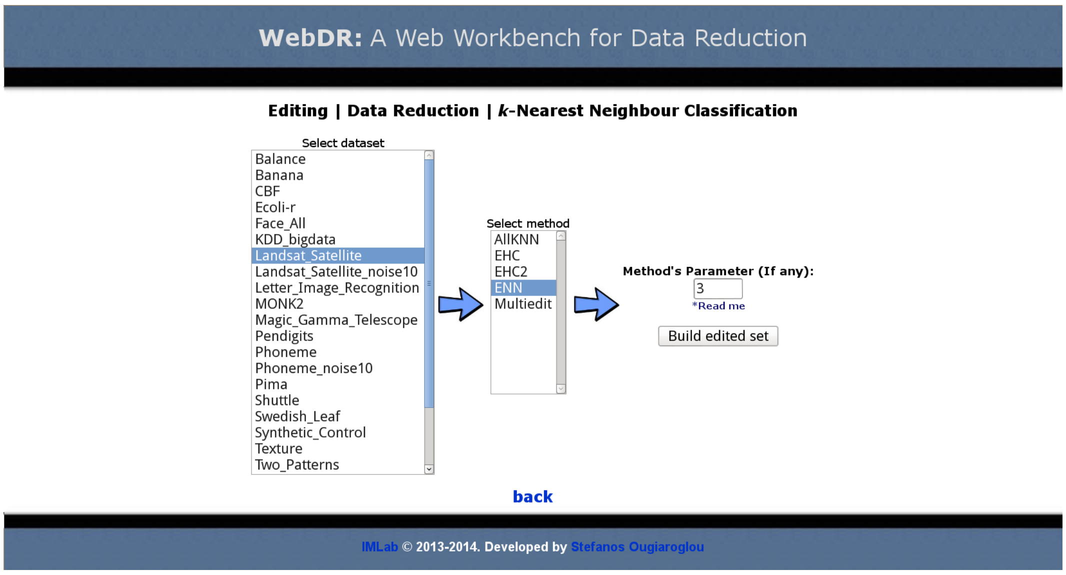
Classes distribution

Class	Number of items
0	307
1	141
2	271
3	125
4	142
5	301

back

IMLab © 2013-2014. Developed by Stefanos Ougiaroglou

Figure A.2: Dataset explorer tool: Exploration of the Landsat Satellite dataset

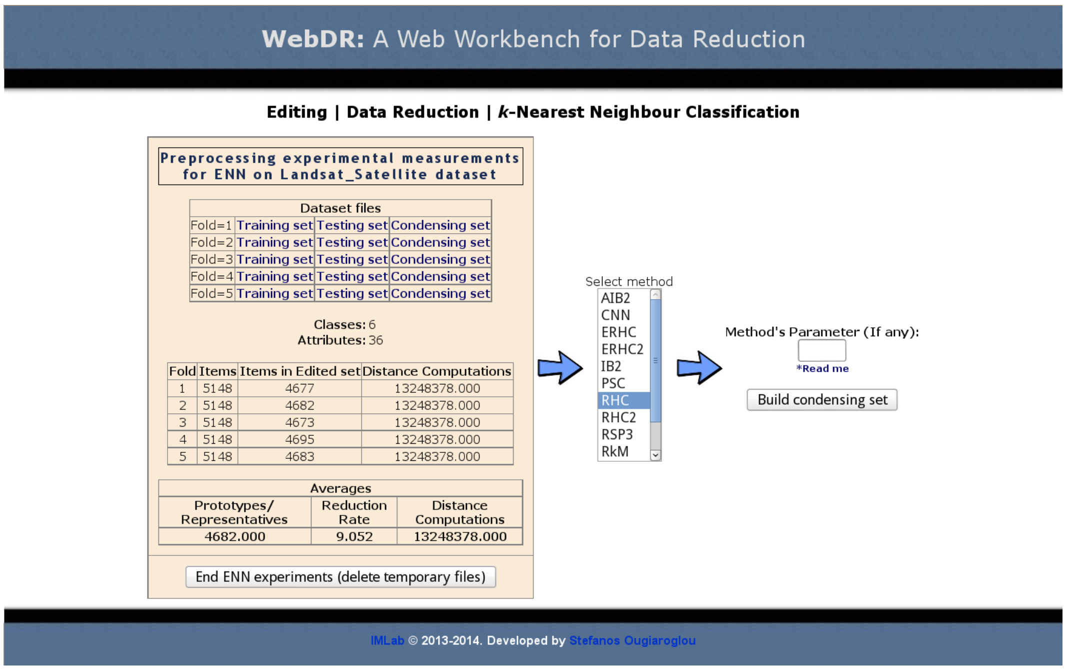


Select dataset: Landsat_Satellite

Select method: ENN

Method's Parameter (If any): 3

Figure A.3: Editing of the Landsat Satellite dataset through ENN-rule (with k=3)


WebDR: A Web Workbench for Data Reduction
Editing | Data Reduction | k-Nearest Neighbour Classification
Select method
AIB2
CNN
ERHC
ERHC2
IB2
PSC
RHC
RHC2
RSP3
RkM
Method's Parameter (If any):
*Read me
Build condensing set
End ENN experiments (delete temporary files)
IMLab © 2013-2014. Developed by Stefanos Ougiaroglou

Preprocessing experimental measurements for ENN on Landsat_Satellite dataset

Dataset files			
Fold=1	Training set	Testing set	Condensing set
Fold=2	Training set	Testing set	Condensing set
Fold=3	Training set	Testing set	Condensing set
Fold=4	Training set	Testing set	Condensing set
Fold=5	Training set	Testing set	Condensing set

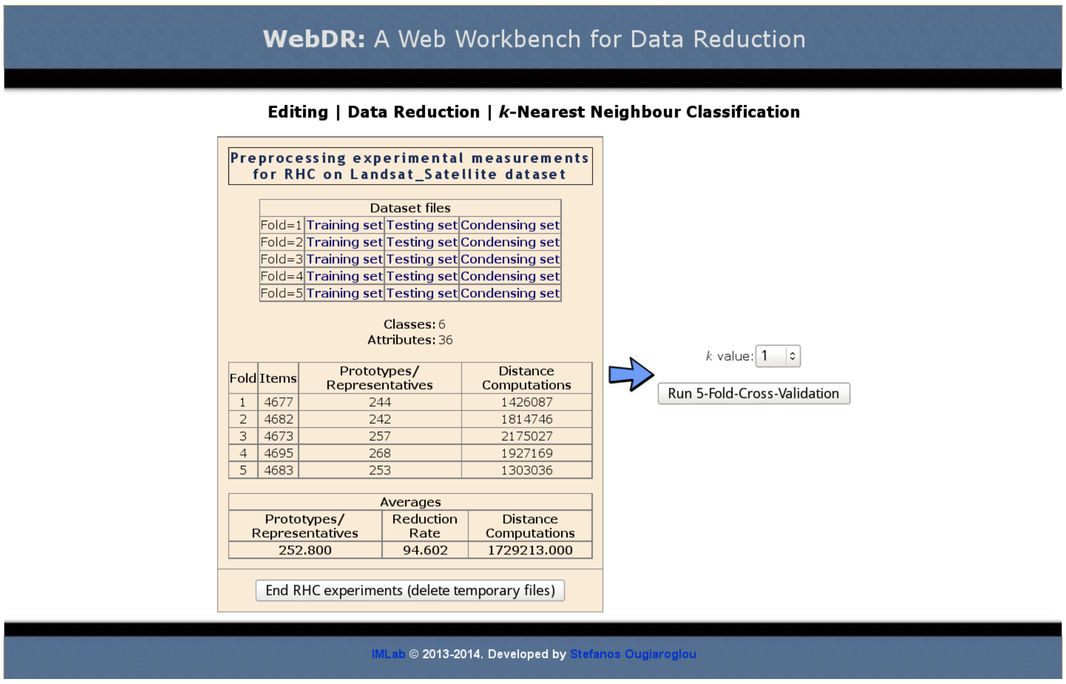
Classes: 6
Attributes: 36

Fold	Items	Items in Edited set	Distance Computations
1	5148	4677	13248378.000
2	5148	4682	13248378.000
3	5148	4673	13248378.000
4	5148	4695	13248378.000
5	5148	4683	13248378.000

Averages		
Prototypes/ Representatives	Reduction Rate	Distance Computations
4682.000	9.052	13248378.000

Select method: RHC

Figure A.4: Results of ENN-rule and data abstraction through RHC


WebDR: A Web Workbench for Data Reduction

Editing | Data Reduction | *k*-Nearest Neighbour Classification

Preprocessing experimental measurements for RHC on Landsat_Satellite dataset

Dataset files			
Fold=1	Training set	Testing set	Condensing set
Fold=2	Training set	Testing set	Condensing set
Fold=3	Training set	Testing set	Condensing set
Fold=4	Training set	Testing set	Condensing set
Fold=5	Training set	Testing set	Condensing set

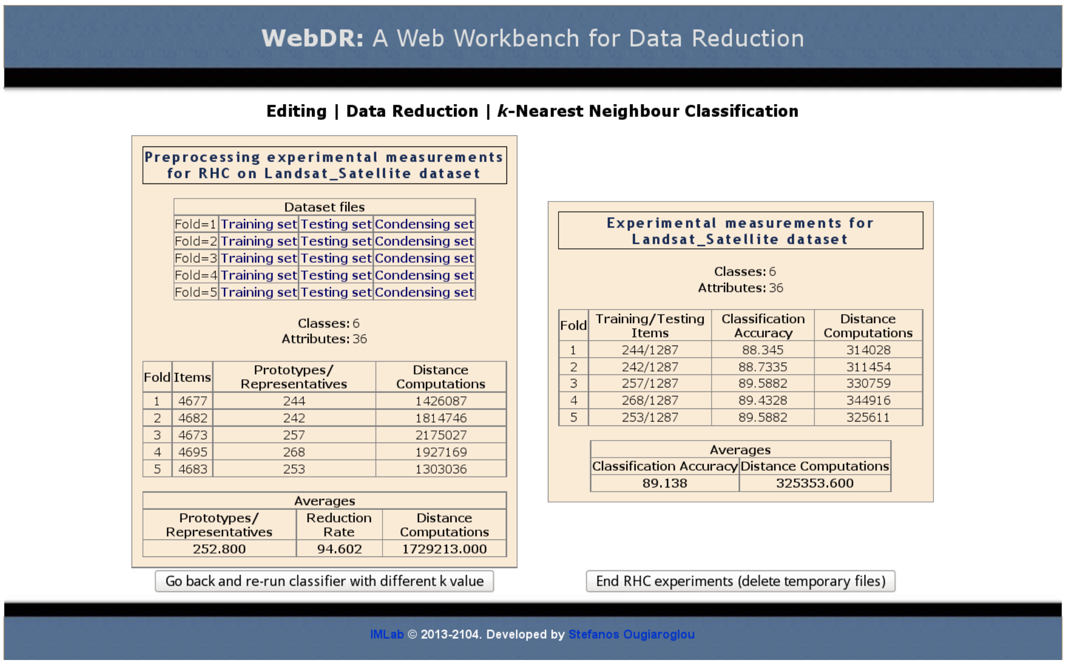
Classes: 6
Attributes: 36

Fold	Items	Prototypes/ Representatives	Distance Computations
1	4677	244	1426087
2	4682	242	1814746
3	4673	257	2175027
4	4695	268	1927169
5	4683	253	1303036

Averages		
Prototypes/ Representatives	Reduction Rate	Distance Computations
252.800	94.602	1729213.000

End RHC experiments (delete temporary files)
k value: 1
Run 5-Fold-Cross-Validation
IMLab © 2013-2014. Developed by Stefanos Ougiaroglou

Figure A.5: Results of RHC and selection of $k = 1$ for the classification step


WebDR: A Web Workbench for Data Reduction

Editing | Data Reduction | *k*-Nearest Neighbour Classification

Preprocessing experimental measurements for RHC on Landsat_Satellite dataset

Dataset files			
Fold=1	Training set	Testing set	Condensing set
Fold=2	Training set	Testing set	Condensing set
Fold=3	Training set	Testing set	Condensing set
Fold=4	Training set	Testing set	Condensing set
Fold=5	Training set	Testing set	Condensing set

Classes: 6
Attributes: 36

Fold	Items	Prototypes/ Representatives	Distance Computations
1	4677	244	1426087
2	4682	242	1814746
3	4673	257	2175027
4	4695	268	1927169
5	4683	253	1303036

Averages		
Prototypes/ Representatives	Reduction Rate	Distance Computations
252.800	94.602	1729213.000

Experimental measurements for Landsat_Satellite dataset

Classes: 6
Attributes: 36

Fold	Training/Testing Items	Classification Accuracy	Distance Computations
1	244/1287	88.345	314028
2	242/1287	88.7335	311454
3	257/1287	89.5882	330759
4	268/1287	89.4328	344916
5	253/1287	89.5882	325611

Averages	
Classification Accuracy	Distance Computations
89.138	325353.600

Go back and re-run classifier with different k value
End RHC experiments (delete temporary files)
IMLab © 2013-2104. Developed by Stefanos Ougiaroglou

Figure A.6: Results of the k-NN classification

Appendix B

Publications

B.1 Journals

1. Stefanos Ougiaroglou, Georgios Evangelidis, "**RHC: Non-parametric cluster-based data reduction for efficient k-NN classification**", Pattern Analysis and Applications, Springer, accepted with minor revision, revision is under review (Impact factor: 0.814)

2. Stefanos Ougiaroglou, Georgios Evangelidis, "**Efficient k-NN classification based on homogeneous clusters**", Artificial Intelligence Review, Springer (Impact factor: 1.565)

3. Stefanos Ougiaroglou, Georgios Evangelidis, "**Efficient data abstraction using weighted IB2 prototypes**", Computer Science and Information Systems (ComSIS), accepted, to appear (Impact factor: 0.549)

4. Stefanos Ougiaroglou, Georgios Evangelidis, Dimitris A. Dervos "**FHC: An adaptive fast hybrid method for k-NN classification**", Logic Journal of the IGPL, Oxford journals, accepted with major revision, revision is under review (Impact factor: 1.136)

5. Stefanos Ougiaroglou, Georgios Evangelidis, "**Efficient editing and data abstraction by finding homogeneous clusters**", under review

B.2 Book chapters

1. Stefanos Ougiaroglou, Leonidas Karamitopoulos, Christos Tatoglou, Georgios Evangelidis, Dimitris A. Dervos, “**Applying prototype selection and abstraction algorithms for efficient time series classification**”, In “Artificial Neural Networks - Methods and Applications in Bio-/Neuroinformatics (Series in Bio-/Neuroinformatics)”, Springer, accepted, to appear

B.3 Conferences

1. Stefanos Ougiaroglou, Georgios Evangelidis, “**EHC: Non-parametric editing by finding homogeneous clusters**”, In proceedings of 8th International Symposium on Foundations of Information and Knowledge Systems (FoIKS 2014), Springer/LNCS, accepted, Bordeaux, France, 2014

2. Stefanos Ougiaroglou, Georgios Evangelidis, “**WebDR: A Web Workbench for Data Reduction**”, In proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD 2014), Springer/LNAI, demo paper, accepted, Nancy, France, 2014

3. Stefanos Ougiaroglou, Georgios Evangelidis, “**AIB2: An abstraction data reduction technique based on IB2**”, In proceedings of 6th Balkan Conference in Informatics (BCI 2013), ACM ICPS, pp. 13-16, Thessaloniki, Greece, 2013

4. Stefanos Ougiaroglou, Leonidas Karamitopoulos, Christos Tatoglou, Georgios Evangelidis, Dimitris A. Dervos, “**Applying general-purpose data reduction techniques for fast time series classification**”, In proceedings of 23rd International Conference on Artificial Neural Networks (ICANN 2013), Springer/LNCS 8131, pp. 34-41, Sofia, Bulgaria, 2013

5. Stefanos Ougiaroglou, Georgios Evangelidis, “**A fast hybrid k-NN classifier based on homogeneous clusters**”, In proceedings of 8th IFIP WG12.5 International Conference on Artificial Intelligence Applications and Innovations (AIAI 2012), IFIP AICT 381, Springer, pp. 327-336, Halkidiki, Greece, 2012

6. Stefanos Ougiaroglou, Georgios Evangelidis, “**Efficient dataset size reduction by finding homogeneous clusters**”, In proceedings of 5th Balkan Conference in Informatics (BCI 2012), ACM ICPS, pp. 168-173, Novi Sad, Serbia, 2012

7. Stefanos Ougiaroglou, Georgios Evangelidis, “**Fast and accurate k-nearest neighbour classification using prototype selection by clustering**”, In proceedings of 16th Panhellenic Conference on Informatics (PCI 2012), IEEE CPS, pp. 168-173, Piraeus, Greece, 2012

8. Stefanos Ougiaroglou, Georgios Evangelidis, Dimitris A. Dervos, “**An adaptive hybrid and cluster-based model for speeding up the k-NN classifier**”, In proceedings of 7th International Conference on Hybrid Artificial Intelligence Systems (HAIS 2012), Springer/LNCS 7209, pp. 163-175, Salamanca, Spain, 2012

9. Stefanos Ougiaroglou, Georgios Evangelidis, “**A simple noise-tolerant abstraction algorithm for fast k-NN classification**”, In proceedings of 7th International Conference on Hybrid Artificial Intelligence Systems (HAIS 2012), Springer/LNCS 7209, pp.210-221, Salamanca, Spain, 2012

10. Stefanos Ougiaroglou, Georgios Evangelidis, Dimitris A. Dervos, “**A fast hybrid classification algorithm based on the minimum distance and the k-NN classifiers**”, In proceedings of 4th International Conference on SImilarity Search and APplications (SISAP 2011, ACM), pp. 97-104, Lipari island, Italy, 2011

11. Stefanos Ougiaroglou, Georgios Evangelidis, Dimitris A. Dervos, “**An extensive experimental study on the cluster-based reference set reduction for speeding-up the k-NN classier**”, In proceeding of International Conference on Integrated Information, IC-ININFO 2011, pp. 12-15, Kos island, Greece, 2011

Bibliography

[1] C.C. Aggarwal. *Data Streams: Models and Algorithms*. Advances in Database Systems Series. Springer Science+Business Media, LLC, 2007.

[2] Charu C. Aggarwal. On the effects of dimensionality reduction on high dimensional similarity search. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '01, pages 256–266, New York, NY, USA, 2001. ACM.

[3] Jesús S. Aguilar, José C. Riquelme, and Miguel Toro. Data set editing by ordered projection. *Intell. Data Anal.*, 5(5):405–417, October 2001.

[4] David W. Aha. Tolerating noisy, irrelevant and novel attributes in instance-based learning algorithms. *Int. J. Man-Mach. Stud.*, 36(2):267–287, February 1992.

[5] David W. Aha, Dennis Kibler, and Marc K. Albert. Instance-based learning algorithms. *Mach. Learn.*, 6(1):37–66, January 1991.

[6] Jesús Alcalá-Fdez, Alberto Fernández, Julián Luengo, Joaquín Derrac, and Salvador García. Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework. *Multiple-Valued Logic and Soft Computing*, 17(2-3):255–287, 2011.

[7] Jesus Alcala-Fdez, Luciano Sanchez, Salvador Garcia, Maria Jose del Jesus, Sebastian Ventura, Josep Maria Garrell i Guiu, Jose Otero, Cristobal Romero, Jaume Bacardit, Victor M. Rivas, Juan Carlos Fernandez, and Francisco Herrera. Keel: a software tool to assess evolutionary algorithms for data mining problems. *Soft Comput.*, 13(3):307–318, October 2008.

[8] Hosein Alizadeh, Behrouz Minaei-Bidgoli, and Saeed K. Amirgholipour. A new method for improving the performance of k nearest neighbor using clustering technique. *JCIT*, 4(2):84–92, 2009.

[9] Fabrizio Angiulli. Fast condensed nearest neighbor rule. In *Proceedings of the 22nd international conference on Machine learning*, ICML '05, pages 25–32, New York, NY, USA, 2005. ACM.

[10] Fabrizio Angiulli. Fast nearest neighbor condensation for large data sets classification. *IEEE Trans. on Knowl. and Data Eng.*, 19(11):1450–1464, November 2007.

[11] David Arthur and Sergei Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1027–1035, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[12] K. Bache and M. Lichman. UCI machine learning repository, 2013.

[13] Ricardo Barandela and Eduardo Gasca. Decontamination of training samples for supervised pattern recognition methods. In *Proceedings of the Joint IAPR International Workshops on Advances in Pattern Recognition*, pages 621–630, London, UK, UK, 2000. Springer-Verlag.

[14] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975.

[15] Jürgen Beringer and Eyke Hüllermeier. Efficient instance-based learning on data streams. *Intell. Data Anal.*, 11(6):627–650, December 2007.

[16] P. Berkhin. A survey of clustering data mining techniques. *Grouping Multidimensional Data*, pages 25–71, 2006.

[17] Henry Brighton and Chris Mellish. Advances in instance selection for instance-based learning algorithms. *Data Min. Knowl. Discov.*, 6(2):153–172, April 2002.

[18] Sergey Brin. Near neighbor search in large metric spaces. In *Proceedings of the 21th International Conference on Very Large Data Bases*, VLDB '95, pages 574–584, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[19] Krisztian Buza, Alexandros Nanopoulos, and Lars Schmidt-Thieme. Insight: efficient and effective instance selection for time-series classification. In *Proceedings of the 15th Pacific-Asia conference on Advances in knowledge discovery and data mining - Volume Part II*, PAKDD'11, pages 149–160, Berlin, Heidelberg, 2011. Springer-Verlag.

[20] Andre C.P.L.F. Carvalho and Alex A. Freitas. A tutorial on multi-label classification techniques. In Ajith Abraham, Aboul-Ella Hassanien, and Václav Snášel, editors, *Foundations of Computational Intelligence Volume 5*, volume 205 of *Studies in Computational Intelligence*, pages 177–195. Springer Berlin Heidelberg, 2009.

[21] Chin-Liang Chang. Finding prototypes for nearest neighbor classifiers. *IEEE Trans. Comput.*, 23(11):1179–1184, November 1974.

[22] Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín. Searching in metric spaces. *ACM Comput. Surv.*, 33(3):273–321, September 2001.

[23] C. H. Chen and Adam Jóźwik. A sample set condensation algorithm for the class sensitive artificial neural network. *Pattern Recogn. Lett.*, 17(8):819–823, July 1996.

[24] King Lum Cheung and Ada Wai-Chee Fu. Enhanced nearest neighbour search on the r-tree. *SIGMOD Rec.*, 27(3):16–21, September 1998.

[25] Chien-Hsing Chou, Bo-Han Kuo, and Fu Chang. The generalized condensed nearest neighbor rule as a data reduction method. In *Proceedings of the 18th International Conference on Pattern Recognition - Volume 02*, ICPR '06, pages 556–559, Washington, DC, USA, 2006. IEEE Computer Society.

[26] Paolo Ciaccia, Marco Patella, and Pavel Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, VLDB '97, pages 426–435, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[27] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theor.*, 13(1):21–27, September 2006.

[28] B. V. Dasarathy. *Nearest neighbor (NN) norms : NN pattern classification techniques.* IEEE Computer Society Press, 1991.

[29] B. V. Dasarathy, J. S. Sánchez, and S. Townsend. Nearest neighbour editing and condensing tools–synergy exploitation. *Pattern Analysis & Applications*, 3(1):19–30, 2000.

[30] Piew Datta and Dennis Kibler. Symbolic nearest mean classifiers. In *Proceedings of the fourteenth national conference on artificial intelligence and ninth conference on Innovative applications of artificial intelligence*, AAAI'97/IAAI'97, pages 82–87. AAAI Press, 1997.

[31] Piew Datta and Dennis F. Kibler. Learning symbolic prototypes. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, pages 75–82, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[32] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30, December 2006.

[33] V. Susheela Devi and M. Narasimha Murty. An incremental prototype set building technique. *Pattern Recognition*, 35(2):505–513, 2002.

[34] P. A. Devijver and J. Kittler. On the edited nearest neighbor rule. In *Proceedings of the Fifth International Conference on Pattern Recognition.* The Institute of Electrical and Electronics Engineers, 1980.

[35] Elena Deza and Michel M. Deza. *Encyclopedia of Distances.* Springer, Berlin, Heidelberg, 2009.

[36] Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn Keogh. Querying and mining of time series data: experimental comparison of representations and distance measures. *Proc. VLDB Endow.*, 1(2):1542–1552, August 2008.

[37] Pedro Domingos and Michael Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Mach. Learn.*, 29(2-3):103–130, November 1997.

[38] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis*, pages 98–105. John Wiley and Sons, 1973.

[39] Sahibsingh A. Dudani. The distance-weighted k-nearest-neighbor rule. *Systems, Man and Cybernetics, IEEE Transactions on*, SMC-6(4):325–327, 1976.

[40] Christoph F. Eick, Nidal Zeidat, and Ricardo Vilalta. Using representative-based clustering for nearest neighbor dataset editing. In *Proceedings of the Fourth IEEE International Conference on Data Mining*, ICDM '04, pages 375–378, Washington, DC, USA, 2004. IEEE Computer Society.

[41] Brian S. Everitt, Sabine Landau, and Morven Leese. *Cluster Analysis*. Wiley Publishing, 4th edition, 2009.

[42] Hatem A. Fayed and Amir F. Atiya. A novel template reduction approach for the k-nearest neighbor method. *Trans. Neur. Netw.*, 20(5):890–896, May 2009.

[43] Hatem A. Fayed, Sherif R. Hashem, and Amir F. Atiya. Self-generating prototypes for pattern classification. *Pattern Recogn.*, 40(5):1498–1509, May 2007.

[44] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[45] Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, September 1977.

[46] K. Fukunage and P. M. Narendra. A branch and bound algorithm for computing k-nearest neighbors. *IEEE Trans. Comput.*, 24(7):750–753, July 1975.

[47] Salvador García, José Ramón Cano, and Francisco Herrera. A memetic algorithm for evolutionary prototype selection: A scaling up approach. *Pattern Recogn.*, 41(8):2693–2709, August 2008.

[48] Salvador Garcia, Joaquin Derrac, Jose Cano, and Francisco Herrera. Prototype selection for nearest neighbor classification: Taxonomy and empirical study. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(3):417–435, March 2012.

[49] Milton García-Borroto, Yenny Villuendas-Rey, Jesús Ariel Carrasco-Ochoa, and José Fco. Martínez-Trinidad. Using maximum similarity graphs to edit nearest neighbor classifiers. In *Proceedings of the 14th Iberoamerican Conference on Pattern Recognition: Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, CIARP '09, pages 489–496, Berlin, Heidelberg, 2009. Springer-Verlag.

[50] G. W. Gates. The reduced nearest neighbor rule. ieee transactions on information theory. *IEEE Transactions on Information Theory*, 18(3):431–433, 1972.

[51] Marek Grochowski and Norbert Jankowski. Comparison of instance selection algorithms ii. results and comments. In *Artificial Intelligence and Soft Computing - ICAISC 2004*, volume 3070 of *LNCS*, pages 580–585. Springer Berlin / Heidelberg, 2004.

[52] Antonin Guttman. R-trees: a dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, SIGMOD '84, pages 47–57, New York, NY, USA, 1984. ACM.

[53] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009.

[54] Greg Hamerly and Charles Elkan. Alternatives to the k-means algorithm that find better clusterings. In *Proceedings of the Eleventh International Conference on Information and Knowledge Management*, CIKM '02, pages 600–607, New York, NY, USA, 2002. ACM.

[55] Eui-Hong Han and George Karypis. Centroid-based document classification: Analysis and experimental results. In *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, PKDD '00, pages 424–431, London, UK, UK, 2000. Springer-Verlag.

[56] J. Han, M. Kamber, and J. Pei. *Data Mining: Concepts and Techniques*. The Morgan Kaufmann Series in Data Management Systems. Elsevier Science, 2011.

[57] P E Hart. The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, 14(3):515–516, 1968.

[58] Kazuo Hattori and Masahito Takahashi. A new edited k-nearest neighbor rule in the pattern classification problem. *Pattern Recognition*, 33(3):521 – 528, 2000.

[59] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 1998.

[60] Gísli R. Hjaltason and Hanan Samet. Distance browsing in spatial databases. *ACM Trans. Database Syst.*, 24(2):265–318, June 1999.

[61] Eduardo R. Hruschka, Estevam R. Jr. Hruschka, and Nelson F. Ebecken. Towards efficient imputation by nearest-neighbors: A clustering-based approach. In *Australian Conference on Artificial Intelligence*, pages 513–525, 2004.

[62] Seongseob Hwang and Sungzoon Cho. Clustering-based reference set reduction for k-nearest neighbor. In *Proceedings of the 4th international symposium on Neural Networks: Part II–Advances in Neural Networks*, ISNN '07, pages 880–888, Berlin, Heidelberg, 2007. Springer-Verlag.

[63] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Comput. Surv.*, 31(3):264–323, September 1999.

[64] Mike James. *Classification algorithms.* Wiley-Interscience, New York, NY, USA, 1985.

[65] Norbert Jankowski and Marek Grochowski. Comparison of instances seletion algorithms i. algorithms survey. In *Artificial Intelligence and Soft Computing - ICAISC 2004*, volume 3070 of *LNCS*, pages 598–603. Springer Berlin / Heidelberg, 2004.

[66] Yuan Jiang and Zhi hua Zhou. Editing training data for knn classifiers with neural network ensemble. In *Lecture Notes in Computer Science, Vol.3173*, pages 356–361. Springer, 2004.

[67] I.T. Jolliffe. *Principal component analysis.* Springer series in statistics. Springer-Verlag, 2002.

[68] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(7):881–892, July 2002.

[69] Leonidas Karamitopoulos and Georgios Evangelidis. Cluster-based similarity search in time series. In *Proceedings of the 2009 Fourth Balkan Conference in Informatics*, BCI '09, pages 113–118, Washington, DC, USA, 2009. IEEE Computer Society.

[70] Eamonn J. Keogh and Michael J. Pazzani. A simple dimensionality reduction technique for fast similarity search in large time series databases. In *Proceedings of the 4th Pacific-Asia Conference on Knowledge Discovery and Data Mining, Current Issues and New Applications*, PADKK '00, pages 122–133, London, UK, UK, 2000. Springer-Verlag.

[71] Sang-Woon Kim and B.J. Oommen. Enhancing prototype reduction schemes with lvq3-type algorithms. *Pattern Recognition*, 36(5):1083 – 1093, 2003.

[72] T. Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.

[73] T. Kohonen, M. R. Schroeder, and T. S. Huang, editors. *Self-Organizing Maps*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 3rd edition, 2001.

[74] Janet Kolodner. *Case-based Reasoning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[75] Jiang Li, Michael T. Manry, Changhua Yu, and D. Randall Wilson. Prototype classifier design with pruning. *International Journal on Artificial Intelligence Tools*, 14(1-2):261–280, 2005.

[76] M. Lozano. *Data Reduction Techniques in Classification processes (Phd Thesis)*. Universitat Jaume I, 2007.

[77] Yannis Manolopoulos, Alexandros Nanopoulos, Apostolos N. Papadopoulos, and Y. Theodoridis. *R-Trees: Theory and Applications (Advanced Information and Knowledge Processing)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[78] K. Mardia, J. Kent, and J. Bibby. *Multivariate Analysis*. Academic Press, 1979.

[79] J. McQueen. Some methods for classification and analysis of multivariate observations. In *Proc. of 5th Berkeley Symp. on Math. Statistics and Probability*, pages 281– 298, Berkeley, CA : University of California Press, 1967.

[80] Luisa Micó, Jose Oncina, and Rafael C. Carrasco. A fast branch &amp; bound nearest neighbour classifier in metric spaces. *Pattern Recogn. Lett.*, 17(7):731–739, June 1996.

[81] María Luisa Micó, José Oncina, and Enrique Vidal. A new version of the nearest-neighbour approximating and eliminating search algorithm (aesa) with linear preprocessing time and memory requirements. *Pattern Recogn. Lett.*, 15(1):9–17, January 1994.

[82] R.A. Mollineda, F.J. Ferri, and E. Vidal. An efficient prototype merging strategy for the condensed 1-nn rule through class-conditional hierarchical clustering. *Pattern Recognition*, 35(12):2771 – 2782, 2002.

[83] Roberto Odorico. Learning vector quantization with training count (lvqtc). *Neural Networks*, 10(6):1083 – 1088, 1997.

[84] J. Arturo Olvera-López, J. Ariel Carrasco-Ochoa, J. Francisco Martínez-Trinidad, and Josef Kittler. A review of instance selection methods. *Artif. Intell. Rev.*, 34(2):133–143, August 2010.

[85] J. Arturo Olvera-López, J. Francisco Martínez-Trinidad, and J. Ariel Carrasco-Ochoa. Mixed data object selection based on clustering and border objects. In *Proceedings of the Congress on pattern recognition 12th Iberoamerican conference on Progress in pattern recognition, image analysis and applications*, CIARP'07, pages 674–683, Berlin, Heidelberg, 2007. Springer-Verlag.

[86] Jose Arturo Olvera-Lopez, Jesus Ariel Carrasco-Ochoa, and Jose Francisco Martinez Trinidad. A new fast prototype selection method based on clustering. *Pattern Anal. Appl.*, 13(2):131–141, 2010.

[87] José Arturo Olvera-López, Jesús Ariel Carrasco-Ochoa, and José Francisco Martínez Trinidad. Object selection based on clustering and border objects. In Marek Kurzynski, Edward Puchala, Michal Wozniak, and Andrzej Zolnierek, editors, *Computer Recognition Systems 2*, volume 45 of *Advances in Soft Computing*, pages 27–34. Springer, 2008.

[88] Stefanos Ougiaroglou and Georgios Evangelidis. Efficient data abstraction using weighted ib2 prototypes. *Computer Science and Information Systems (ComSIS)*, pages (accepted, to appear).

[89] Stefanos Ougiaroglou and Georgios Evangelidis. Efficient editing and data abstraction by finding homogeneous clusters.

[90] Stefanos Ougiaroglou and Georgios Evangelidis. FHC: an adaptive fast hybrid method for k-nn classification. *Logic Journal of the IGPL*, pages (accepted, to appear).

[91] Stefanos Ougiaroglou and Georgios Evangelidis. RHC: Non-parametric cluster-based data reduction for efficient k-nn classification. *Pattern Analysis and Applications*, pages (accepted, to appear).

[92] Stefanos Ougiaroglou and Georgios Evangelidis. Efficient dataset size reduction by finding homogeneous clusters. In *Proceedings of the Fifth Balkan Conference in Informatics*, BCI '12, pages 168–173, New York, NY, USA, 2012. ACM.

[93] Stefanos Ougiaroglou and Georgios Evangelidis. Fast and accurate k-nearest neighbor classification using prototype selection by clustering. In *16th Panhellenic Conference on Informatics (PCI), 2012*, pages 168–173, 2012.

[94] Stefanos Ougiaroglou and Georgios Evangelidis. A fast hybrid k-nn classifier based on homogeneous clusters. In *Artificial Intelligence Applications and Innovations*, volume 381 of *IFIP Advances in Information and Communication Technology*, pages 327–336. Springer Berlin Heidelberg, 2012.

[95] Stefanos Ougiaroglou and Georgios Evangelidis. A simple noise-tolerant abstraction algorithm for fast k-nn classification. In *Proceedings of the 7th international conference on Hybrid Artificial Intelligent Systems - Volume Part II*, HAIS'12, pages 210–221, Berlin, Heidelberg, 2012. Springer-Verlag.

[96] Stefanos Ougiaroglou and Georgios Evangelidis. AIB2: An abstraction data reduction technique based on ib2. In *Proceedings of the 6th Balkan Conference in Informatics*, BCI '13, pages 13–16, New York, NY, USA, 2013. ACM.

[97] Stefanos Ougiaroglou and Georgios Evangelidis. Efficient k-nn classification based on homogeneous clusters. *Artificial Intelligence Review*, pages 1–23, 2013.

[98] Stefanos Ougiaroglou and Georgios Evangelidis. EHC: Non-parametric editing by finding homogeneous clusters. In Christoph Beierle and Carlo Meghini, editors, *Foundations of Information and Knowledge Systems*, volume 8367 of *Lecture Notes in Computer Science*, pages 290–304. Springer International Publishing, 2014.

[99] Stefanos Ougiaroglou and Georgios Evangelidis. WebDR: A web workbench for data reduction. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, ECML/PKDD'14, page accepted, Berlin, Heidelberg, 2014. Springer-Verlag.

[100] Stefanos Ougiaroglou, Georgios Evangelidis, and Dimitris A. Dervos. An extensive experimental study on the cluster-based reference set reduction for speeding-up the k-nn

classifier. *CoRR / Proceeding of International Conference on Integrated Information (IC-InInfo 2011), Kos island, Greece, 2011*, abs/1309.7750:12–15, 2011.

[101] Stefanos Ougiaroglou, Georgios Evangelidis, and Dimitris A. Dervos. A fast hybrid classification algorithm based on the minimum distance and the k-nn classifiers. In *Proceedings of the Fourth International Conference on SImilarity Search and APplications*, SISAP '11, pages 97–104, New York, NY, USA, 2011. ACM.

[102] Stefanos Ougiaroglou, Georgios Evangelidis, and Dimitris A. Dervos. An adaptive hybrid and cluster-based model for speeding up the k-nn classifier. In *Proceedings of the 7th international conference on Hybrid Artificial Intelligent Systems - Volume Part II*, HAIS'12, pages 163–175, Berlin, Heidelberg, 2012. Springer-Verlag.

[103] Stefanos Ougiaroglou, Leonidas Karamitopoulos, Christos Tatoglou, Georgios Evangelidis, and Dimitris A. Dervos. Applying prototype selection and abstraction algorithms for efficient time series classification. In *Artificial Neural Networks - Methods and Applications in Bio-/Neuroinformatics*, Series in Bio-/Neuroinformatics, pages (accepted, to appear). Springer Berlin Heidelberg.

[104] Stefanos Ougiaroglou, Leonidas Karamitopoulos, Christos Tatoglou, Georgios Evangelidis, and Dimitris A. Dervos. Applying general-purpose data reduction techniques for fast time series classification. In *Artificial Neural Networks and Machine Learning – ICANN 2013*, volume 8131 of *Lecture Notes in Computer Science*, pages 34–41. Springer Berlin Heidelberg, 2013.

[105] Stefanos Ougiaroglou, Alexandros Nanopoulos, Apostolos N. Papadopoulos, Yannis Manolopoulos, and Tatjana Welzer-Druzovec. Adaptive k-nearest-neighbor classification using a dynamic number of nearest neighbors. In *Proceedings of the 11th East European conference on Advances in databases and information systems*, ADBIS'07, pages 66–82, Berlin, Heidelberg, 2007. Springer-Verlag.

[106] José C. Riquelme, Jesús S. Aguilar-Ruiz, and Miguel Toro. Finding representative patterns with ordered projections. *Pattern Recognition*, 36(4):1009 – 1018, 2003.

[107] G. Ritter, H. Woodruff, S. Lowry, and T. Isenhour. An algorithm for a selective nearest neighbor decision rule. *IEEE Trans. on Inf. Theory*, 21(6):665–669, 1975.

[108] John T. Robinson. The k-d-b-tree: a search structure for large multidimensional dynamic indexes. In *Proceedings of the 1981 ACM SIGMOD international conference on Management of data*, SIGMOD '81, pages 10–18, New York, NY, USA, 1981. ACM.

[109] R. Roiger and M. W. Geatz. *Data Mining: A Tutorial Based Primer.* Addison Wesley, 2003.

[110] L. Rokach. *Data Mining with Decision Trees: Theory and Applications.* Series in machine perception and artificial intelligence. World Scientific Publishing Company, Incorporated, 2007.

[111] Enrique Vidal Ruiz. An algorithm for finding nearest neighbours in (approximately) constant average time. *Pattern Recognition Letters*, 4(3):145 – 157, 1986.

[112] H. Samet. *Foundations of multidimensional and metric data structures.* The Morgan Kaufmann series in computer graphics. Elsevier/Morgan Kaufmann, 2006.

[113] J. S. Sánchez, R. Barandela, A. I. Marqués, R. Alejo, and J. Badenas. Analysis of new techniques to obtain quality training sets. *Pattern Recogn. Lett.*, 24(7):1015–1022, April 2003.

[114] José Salvador Sánchez. High training set size reduction by space partitioning and prototype abstraction. *Pattern Recognition*, 37(7):1561–1564, 2004.

[115] Nicola Segata, Enrico Blanzieri, Sarah Jane Delany, and Pádraig Cunningham. Noise reduction for instance-based learning with a local maximal margin approach. *J. Intell. Inf. Syst.*, 35(2):301–331, October 2010.

[116] D. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures.* A Chapman & Hall book. Chapman & Hall/CRC, 2011.

[117] J.S. Sánchez, F. Pla, and F.J. Ferri. On the use of neighbourhood-based non-parametric classifiers. *Pattern Recognition Letters*, 18(11–13):1179 – 1186, 1997.

[118] J.S. Sánchez, F. Pla, and F.J. Ferri. Prototype selection for the nearest neighbour rule through proximity graphs. *Pattern Recognition Letters*, 18(6):507 – 513, 1997.

[119] Fadi Thabtah. A review of associative classification mining. *Knowl. Eng. Rev.*, 22(1):37–65, March 2007.

[120] I. Tomek. An experiment with the edited nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics*, 6:448–452, 1976.

[121] I. Tomek. Two modifications of cnn. *Systems, Man and Cybernetics, IEEE Transactions on*, SMC-6(11):769–772, Nov 1976.

[122] Godfried Toussaint. Proximity graphs for nearest neighbor decision rules: Recent progress. In *34th Symposium on the INTERFACE*, pages 17–20, 2002.

[123] Isaac Triguero, Joaquín Derrac, Salvador Garcia, and Francisco Herrera. A taxonomy and experimental study on prototype generation for nearest neighbor classification. *Trans. Sys. Man Cyber Part C*, 42(1):86–100, January 2012.

[124] Grigorios Tsoumakas and Ioannis Katakis. Multi-label classification: An overview. *Int J Data Warehousing and Mining*, 2007:1–13, 2007.

[125] Alexey Tsymbal. The problem of concept drift: definitions and related work. Technical Report TCD-CS-2004-15, The University of Dublin, Trinity College, Department of Computer Science, Dublin, Ireland, 2004.

[126] Fernando Vázquez, J. Salvador Sánchez, and Filiberto Pla. A stochastic approach to wilson's editing algorithm. In *Proceedings of the Second Iberian conference on Pattern Recognition and Image Analysis - Volume Part II*, IbPRIA'05, pages 35–42, Berlin, Heidelberg, 2005. Springer-Verlag.

[127] Enrique Vidal. New formulation and improvements of the nearest-neighbour approximating and eliminating search algorithm (aesa). *Pattern Recogn. Lett.*, 15(1):1–7, January 1994.

[128] Xueyi Wang. A fast exact k-nearest neighbors algorithm for high dimensional search using k-means clustering and triangle inequality. In *The 2011 International Joint Conference on Neural Networks (IJCNN)*, pages 1293 –1299, August 2011.

[129] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB '98, pages 194–205, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[130] D. Randall Wilson and Tony R. Martinez. Instance pruning techniques. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, pages 403–411, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[131] D. Randall Wilson and Tony R. Martinez. Reduction techniques for instance-basedlearning algorithms. *Mach. Learn.*, 38(3):257–286, March 2000.

[132] Dennis L. Wilson. Asymptotic properties of nearest neighbor rules using edited data. *IEEE trans. on systems, man, and cybernetics*, 2(3):408–421, July 1972.

[133] Junjie Wu. *Advances in K-means Clustering: A Data Mining Thinking*. Springer Publishing Company, Incorporated, 2012.

[134] Xiaopeng Xi, Eamonn Keogh, Christian Shelton, Li Wei, and Chotirat Ann Ratanamahatana. Fast time series classification using numerosity reduction. In *Proceedings of the 23rd international conference on Machine learning*, ICML '06, pages 1033–1040, New York, NY, USA, 2006. ACM.

[135] Qiaobing Xie, Charles A. Laszlo, and Rabab K. Ward. Vector quantization technique for nonparametric classifier design. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(12):1326–1330, 1993.

[136] Chen-Wen Yen, Chieh-Neng Young, and Mark L. Nagurka. A vector quantization method for nearest neighbor classifier design. *Pattern Recogn. Lett.*, 25(6):725–731, April 2004.

[137] Byoung-Kee Yi and Christos Faloutsos. Fast time sequence indexing for arbitrary lp norms. In *Proceedings of the 26th International Conference on Very Large Data Bases*, VLDB '00, pages 385–394, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[138] Peter N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, SODA '93, pages 311–321, Philadelphia, PA, USA, 1993. Society for Industrial and Applied Mathematics.

[139] Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. *Similarity Search - The Metric Space Approach*, volume 32. Springer, 2006.

[140] Bin Zhang and Sargur N. Srihari. Fast k-nearest neighbor classification using cluster-based trees. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(4):525–528, April 2004.

[141] G. P. Zhang. Neural networks for classification: A survey. *Trans. Sys. Man Cyber Part C*, 30(4):451–462, November 2000.

[142] Harry Zhang. The optimality of naive bayes. In Valerie Barr and Zdravko Markov, editors, *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference, Miami Beach, Florida, USA*, pages 562–567. AAAI Press, 2004.

[143] Ke-Ping Zhao, Shui-Geng Zhou, Ji-Hong Guan, and Ao-Ying Zhou. C-pruner: an improved instance pruning algorithm. In *2003 International Conference on Machine Learning and Cybernetics*, volume 1, pages 94 – 99, nov. 2003.