Ricardo Jorge Palheira Mestre

Licenciado em Engenharia Informática

Improvements on the KNN Classifier

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática

Orientador : Joaquim Francisco Ferreira da Silva, Prof. Auxiliar,
Universidade Nova de Lisboa

Júri:

Presidente: Prof. Doutora Carla Maria Gonçalves Ferreira (FCT-UNL)

Arguente: Prof. Doutora Maria da Graça de Figueiredo Rodrigues Gaspar (FCUL)

Vogal: Prof. Doutor Joaquim Francisco Ferreira da Silva (FCT-UNL)

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2013

Improvements on the KNN Classifier

To my family.

Acknowledgements

The development of this dissertation has been one of the most significant academic challenges I have ever had to face. However, this report represents the culmination of an academic goal to which I dedicated myself and would not be completed without the help of several people.

First of all, I would like to express my deepest sense of gratitude to my Ph.D. advisor, Dr. Joaquim Ferreira da Silva. I thank him for the continuous guidance and great effort he put into training me in the scientific field. His wisdom, knowledge and commitment to archive great goals inspired and motivated me.

I would like to thank my friends and colleagues António Mota, João Luis, João Silva, Nuno Gomes, Patrícia Espada and Tiago Melo for walking me through the process of writing this dissertation, always offering their strength, help, and companionship.

Finally, I would like to extend these thanks to all who supported and encouraged me throughout this dissertation, in particular my parents, grandparents, girlfriend and friends.

Abstract

The object classification is an important area within the artificial intelligence and its application extends to various areas, whether or not in the branch of science. Among the other classifiers, the *K-nearest neighbor* (KNN) is among the most simple and accurate especially in environments where the data distribution is unknown or apparently not parameterizable. This algorithm assigns the classifying element the major class in the K nearest neighbors. According to the original algorithm, this classification implies the calculation of the distances between the classifying instance and each one of the training objects.

If on the one hand, having an extensive training set is an element of importance in order to obtain a high accuracy, on the other hand, it makes the classification of each object slower due to its $lazy - learning$ algorithm nature. Indeed, this algorithm does not provide any means of storing information about the previous calculated classifications, making the calculation of the classification of two equal instances mandatory. In a way, it may be said that this classifier *does not learn*.

This dissertation focuses on the lazy-learning fragility and intends to propose a solution that transforms the KNN into an eager-learning classifier. In other words, it is intended that the algorithm learns effectively with the training set, thus avoiding redundant calculations.

In the context of the proposed change in the algorithm, it is important to highlight the attributes that most characterize the objects according to their discriminating power. In this framework, there will be a study regarding the implementation of these transformations on data of different types: continuous and/or categorical.

Keywords: artificial intelligence, classification algorithms, KNN, K-nearest neighbor algorithm, lazy-learning, eager-learning, machine learning algorithms

Resumo

A classificação de objectos é uma área importante no âmbito da inteligência artificial e a sua aplicação estende-se às mais variadas áreas, sejam estas ou não no ramo das ciências. Entre os demais classificadores, o algoritmo *K-nearest neighbor* (KNN) está entre os mais simples e eficientes especialmente em ambientes onde a distribuição dos dados é desconhecida ou aparentemente não parametrizável. Este algoritmo atribui ao elemento a classificar, a classe maioritária nos K vizinhos mais próximos. Esta classificação implica, segundo o algoritmo original, o cálculo das distâncias entre a instância a classificar e cada um dos objectos de treino.

Se o fato de um conjunto de treino ser extenso é um elemento importante para se obter uma precisão elevada, por outro lado, torna a classificação de cada objeto mais lenta devido à natureza *lazy-learning* deste algoritmo. Com efeito, este algoritmo não contempla qualquer forma de guardar informação sobre as classificações calculadas anteriormente, o que torna o cálculo da classificação de duas instâncias iguais, obrigatório. De certo modo, poder-se-ia dizer que este classificador *não aprende*.

Esta dissertação debruça-se sobre a fragilidade *lazy-learning* e pretende propor uma solução que transforme o KNN num classificador *eager-learning*. Por outras palavras, pretende-se que o algoritmo aprenda efetivamente com o conjunto de treino, evitando assim, cálculos redundantes.

No contexto da alteração a propor no algoritmo, torna-se importante valorizar os atributos caracterizadores dos objetos segundo o seu poder discriminante. Neste enquadramento, será ainda feito um estudo com vista à aplicação destas transformações em dados de diferentes naturezas: contínuos e/ou categóricos.

Palavras-chave: inteligência artificial, algoritmos de classificação, KNN, *K-nearest neighbor algorithm*, *lazy-learning*, *eager-learning*, *machine learning algorithms*

Contents

List of Figures

List of Tables

1

Introduction

This chapter introduces the topic of this thesis. It will present the context and motivation of this work. The drawbacks associated to this scope will be detailed, as well as the expected contributions as a consequence of the solutions that will be proposed.

Also the outline of the document will be presented.

1.1 Context

In the context of artificial intelligence, pattern recognition is one of the most interesting topics. By definition, pattern recognition is the science that deals with description and classification of objects [MdS00]. Its application extends to several areas beyond computer science such as psychology, psychiatry, ethology, astronomy, biochemistry, microbiology, physics, etc. Regarding computer science, it is common to find references to several applications such as speech recognition, document classification, handwriting recognition, image processing, etc.

Pattern recognition systems/classifiers are typically analysis of information, comprising feature extraction mechanisms to be used in a classification algorithm. The process that selects and analyzes information regarding the characteristics and builds class prototypes is called learning or training [AF03]. The better the attributes are selected, this is, how much they characterize classes, the bigger is the tendency to get them separated in the vector space dictated by the attributes. Training phase is followed by the classification phase which is based on the learning previously built.

It is therefore of great interest to develop effective learning system, this means, that distinguishes relevant features, separates classes sufficiently in space vector in order to

minimize errors when classifying new elements later on. Over time, different methodologies have been developed for learning phase, typically being divided into unsupervised and supervised learning.

In the unsupervised learning there is no knowledge of the set of classes or the classes associated with each training sample. This method has the advantage of learning new classes over time. Therefore it can be considered as a dynamic learning with regarding the set of classes. On the other hand, dispensing human supervision normally shows poorer results in the classification phase of new objects.

Supervised learning is distinguished by a set of characteristics: set of classes to analyze it is known *à priori* and in the training set, each sample is associated with the class to which it belongs. There are several algorithms that fall under this approach, we highlight the classic Naive Bayes classifier, C4.5, SVM and KNN, the latter being the focus of this dissertation.

1.2 Motivation and Problem Description

The KNN classifier has a very simple philosophy. A sample is classified according to the same class that predominates in its K closest instances. In addition to their simple implementation, the main advantages are: has few parameters (only the distance and the value K) and its robustness, supporting noise between classes, this means the classes do not need to be linearly separated, which can be explained for not depending on the distribution model of data, but only from its nearest neighbors. The KNN algorithm is thus a nonparametric classifier. However, KNN as any other classifier has disadvantages. Although withstand noise, the choice of the attributes must be careful. If there are irrelevant or too noisy attributes, they may deviate estimates [Asc03, dat].

The biggest problem of KNN and the main focus of this work has to do with the trend that this classifier has to be slow. This feature is due to the fact that the KNN classifier is a lazy-learning algorithm since for classifying each instance it needs to calculate the distances to all other known instances N (training samples). This algorithm has a time complexity of $\mathcal{O}(N.D)$ for each classification, being D the number of attributes that characterize. This is especially problematic for large data sets and/or with a large number of attributes. In a sense, we can consider that this classifier does not learn, since it can not take any advantage of the classifications already made, as alike as the elements to be classified are.

Besides the disadvantage lazy-learning, another problem of this algorithm is in the data types that it tries to classify. As we know, the KNN classifier is based on K nearest neighbors, but the concept of "nearest" may have different points of view. If the training set contains data characterized by continuous attributes, it is natural that the distance between two samples refers directly as Euclidean distance. However there is room to improve the set of attributes that characterize the training set. One of the typical ways consists in assigning different weights to the attributes as the discriminating power of

each. Although there are already several approaches for assigning weights, these may or may not be appropriate in the context of the solution that will be proposed to minimize the lazy-learning problem.

Often it is up to the algorithm to deal with discrete data. This type of data can also be ordinal or nominal/categorical. This first type of data refers intervals continuous values (eg ranges of heights, weights, etc.) and so, if the attribute is informative enough, the Euclidean distance can be applied. However, categorical data values do not refer nor quantitative or qualitative categories (eg blood type, race, etc.), getting the Euclidean distance apparently inadequate. Typically, the difference between two instances containing categorical information is equal to the number of attributes that contain different values. Although this is a way to find differences, this may not show correctly how different two samples really are.

In order to address this problem, it will be developed a solution that seeks to establish the notion of distance between subcategories of each categorical attribute.

1.3 Outline of the Document

Chapter 2 highlights three types of issues associated with the KNN algorithm. "State of art" approaches are presented as previous research on those subjects is analyzed, revealing their flaws.

In Chapter 3 we discuss the challenges to overcome the previously exposed weaknesses. Also, solutions are outlined and their processes are detailed.

Chapter 4 focuses in testing previously discussed solutions. Datasets are presented in 4.1 and in Subsections 4.2, 4.3 and 4.4 test and performance results of our proposed approach are analyzed and interpreted.

Final conclusions as well as a general balance is covered by Chapter 5. Also, in this chapter we present some thoughts on possible new directions for further research.

2

Related Work

Given the weaknesses of the KNN classifier due to its nature, the "state of art" may be divided into three areas: a) training sample reduction; b) changing the weight of attributes and the weight of vote; c) minimization of the lazy-learning problematic.

In general, the approaches regarding a) attempt to summarize the training set into a minimum number of representatives. The idea is to minimize the inherent computational work in the classification phase of new elements. The proposals regarding b) try to incorporate as informative only attributes with discriminating power, sometimes assigning different importance to different attributes. In addition, each nearest neighbor voting weight may not be the same. For group c) approaches try to reduce the searching space commonly by using tree structures, trying to group elements with similar characteristics, improving classification time without penalizing accuracy levels obtained in the KNN original solution.

While this review may be not exhaustive regarding to all existing documents since the KNN algorithm was designed, it is complete for the three areas in which this algorithm can be improved. I tried to do it by analyzing the most recent work in this area as these would naturally accumulate all the improvements up to date.

2.1 Reducing the Training Sample

In [SAH06], authors proposed an approach for classifying documents by eliminating training sample outliers in oder to obtain more accurate classifications. *Outliers* are elements that deviate significantly from the normal pattern of the class to which they belong.

Aiming the outliers elimination, the authors use *Centroid-Based Document Classifier*

[HK00] to obtain the centroid of each class.

In centroid based classification algorithm (Centroid-Based Document Classifier), documents are represented using vector-space model. In this model, each document d is represented by a vector of terms/keywords. These terms are typically words with discriminating power. In its simplest form, each document is represented by the term-frequency vector (TF) $d_{tf} = (tf_1, tf_2, \ldots, tf_n)$, where tf_i is the i^{th} term frequency in the document. This model has been refined to weigh each term through *Inverse Document Frequency* (IDF) in all training documents set. This is done by multiplying tf_i by $\log(\frac{N}{df_i})$, where N is the number of total of training documents, and df_i the number of documents that contain i^{th} term. Thus, the vector-space model assumes the following form: $d_{tfidf} = (tf_1 \log(\frac{N}{df_1}), tf_2 \log(\frac{N}{df_2}), \ldots, tf_n \log(\frac{N}{df_n}))$. In this model, the similarity between two documents d_i and d_j is measured using the cosine metric:

$$\cos(d_i, d_j) = \frac{d_i.d_j}{||d_i||_2 \times ||d_j||_2} \tag{2.1}$$

where "." denotes, in this case, the inner product of two vectors. Once the document vectors are previously normalized to length 1, Equation 2.1 can be simplified to:

$$\cos(d_i, d_j) = d_i.d_j \tag{2.2}$$

The centroid of a class is defined by:

$$C_c = \frac{1}{||S_c||} \sum_{d \in S_c} d \tag{2.3}$$

where S_c is the set of representative vectors of class' documents. Thus, the centroid is a vector obtained by averaging the weights of the different terms in class' documents.

The distance between each class' centroid and the document vector is computed through Equations 2.2 and 2.3. According to the authors, it was observed that the elements that are too far from the centroid of its class (outliers) may reduce accuracy when classifying new elements. Therefore, those elements should be discarded in order to form a new training set.

To determine which documents should be considered outliers, the authors established a threshold ε, after several experiments. Figure 2.1 shows the sequence of steps for obtaining the new training set, as Figure 2.1(a) illustrates the initial set; Figure 2.1(b) outliers identification; and Figure 2.1(c) the new training set.

Having this new training sample, the original KNN rule is applied, that is, identifying the majority class in the K nearest documents of the document being classified.

In the particular case of the centroid classification-based approach [HK00], the final set of training is reduced to the set of centroids. Thus, this approach can be viewed as a KNN technique in which $K = 1$.

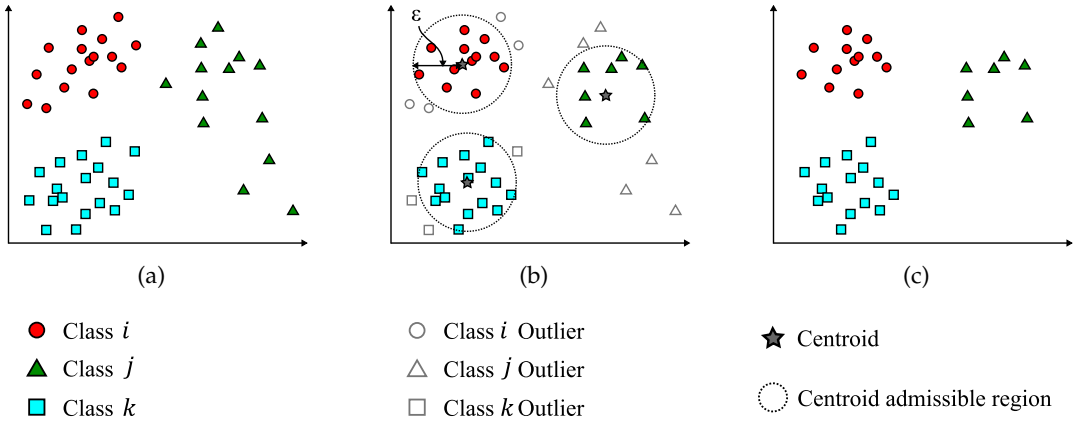

Figure 2.1: Removing outliers and creating the final training set.

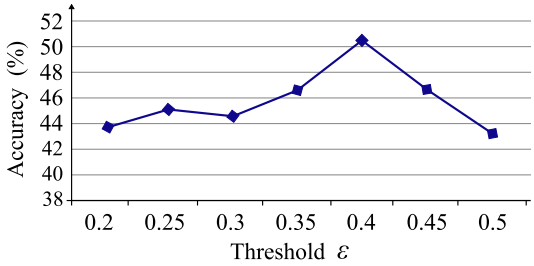

Figure 2.2: Accuracy values using different ε thresholds using 80% of the samples as the training set.

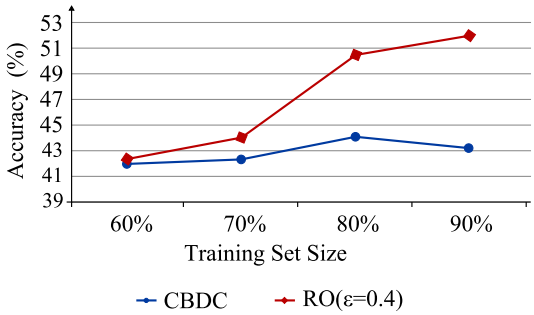

Figure 2.3: Centroid-Based Document Classifier (CBDC) Vs. Removing Outliers (RO) comparison.

The obtained results for the different threshold values are shown in Figure 2.2. Figure 2.3 evaluates the impact of removing the outliers in the Centroid-Based Document Classifier.

This approach has three drawbacks: it is inadequate for multimodal distributions; the threshold value may not be suitable for different densities, and classification remains being lazy-learning.

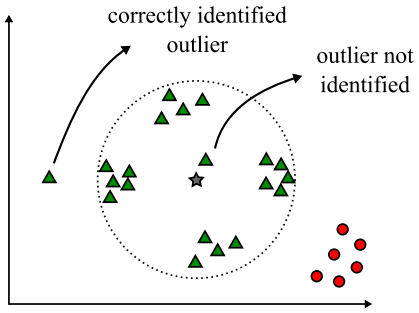

Figure 2.4: Misidentified outlier example.

In the case of each class having multimodal distributions in the vector space as shown in Figure 2.4, it is possible that some outliers are not identified as such. The element next to the centroid of the class identified by triangles would never be identified as an outlier once it is very close to the centroid. However, the leftmost element is correctly identified as an outlier.

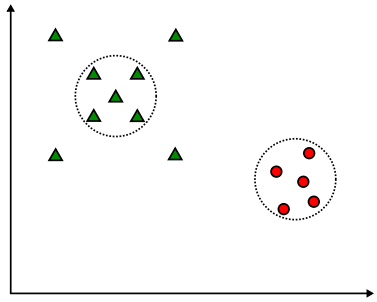

Figure 2.5: Inappropriate threshold example.

Figure 2.5 shows the same threshold (illustrated by the circle) is inadequate to detect outliers in both classes. It is easy to accept that elements outside the circumference should not be considered outliers, as there is a balanced distribution between those four elements relative to the class' centroid. Furthermore, the threshold value is obtained empirically for each training set as there has not been developed any automatic way to compute it.

Finally, for each new instance to be classified, it is still necessary to compute the distance between the element to classify and all training samples, leaving the lazy-learning issue unsolved.

In [ZTD12], the authors propose reducing the number of training samples by introducing the "*Essential Vector*" which is a mechanism that is not clearly explained. However,

according to the authors, the gains in reducing the complexity are significant. Still, the results only address one specific area: documents classification. There is no reference on applying this approach in numerical Vs. categorical attributes.

2.2 Changing Attributes Weight and its Vote Weight

In [AKS], the authors use the *Apriori* – a Data Mining algorithm for automated association rules extraction – in order to improve the initial set of attributes quality, selecting a subset of them that are able to continue to characterize and discriminate training/learning objects' classes. In other words, considering that in the classifier learning process, the training objects are usually characterized by a set of attributes/features that may contain some redundancy (wholly or in part), it would be beneficial that the number of attributes could be as reduced as possible. Indeed, if there is a mechanism which is able to select a subset of those attributes in such way that the objects' classes remain discriminable, the classification training and classification processes will be faster and more simplified.

In the particular case of KNN, given the nature of this classifier, such attribute reduction is reflected by a time efficiency improvement only in the classification phase. In this phase, the object being classified has to be compared with all the training objects since this classifier does not build patterns for each known class.

The approach presented in [AKS] selects the "*Prominent Attributes*" that best characterize the classes and assigns them a weight according to its importance. According to the authors, there were performance increases up to 40%. This improvement was observed especially in classification accuracy compared with the original KNN where all attributes have the same weight, which sometimes causes misclassifications.

This approach may provide significant improvement for most classifiers. However, if the initial attributes in a set of objects contain no redundancy and they have similar discriminating power, this approach will not benefit KNN as this classifier biggest disadvantage relates to its lazy-learning nature. In other words, the improvement shown in this approach does not avoid that each object has to be, still, compared to all training elements, which makes it extremely inefficient in cases where the set is very large.

As in [AKS], other authors, for example [PSFA11], propose improvements in the scope of attribute's weight for computing the distance between the object to be classified and the training elements. One of the first proposals is in [SA76]. In this paper, the author introduces the method of weighting votes (WKNN), where the nearest neighbors are given more weight than the farthest ones by using the weighted distance function. Thus, for a chosen K value, the weight assigned to the i^{th} neighbor follows three steps: it computes the distance from the K^{th} neighbor to the object being classified, lets call it A; the it computes the distance from the i^{th} neighbor to the object being classified, lets call it B; and then, by using the difference between these two distances $(A - B)$ divided by the largest of the distances, this is, by distance A. When the distances are equal to all neighbors, the weight assigned is 1. Different weights are decisive for (weighted) voting to the class

determination. In some cases, WKNN got some improvements compared to the original KNN.

In [ZWZZ07], the authors propose a method they call *Kernel Difference-Weighted Nearest Neighbor* (KDF-WKNN). This method defines the KNN rule as a constrained optimization problem. The authors also propose an efficient solution to compute the different nearest neighbors weights according to their distance to the object to be classified. According to the authors, the KDF-WKNN performs better than the original KNN and WKNN, and it is comparable to some of the "state of the art" methods in terms of classification accuracy.

In [GDZX12], the authors developed a new KNN rule named *Distance-Weighted K-Nearest Neighbor* (DWKNN). The motivation for it lies on the problem of the sensitivity of the K value when using KNN classifier, aiming to improve accuracy results. The results presented by the authors show robustness regarding the chosen K value and good accuracy when compared to other KNN approaches.

Previous approaches ([AKS, SA76, ZWZZ07, GDZX12]) are proposal examples to compute the distances in order to improve the KNN's achieved accuracy, either through filtering the best attributes, or by assigning weights based on distance for voting purposes in the class assignment. However, regarding the accuracy obtained, even in the original version, this classifier is comparable to the best ones in some environments, even managing to overcome some, particularly in circumstances where it is difficult to obtain data distribution models in the vector space defined by characterizing attributes. Also, these approaches do not offer any solution for the biggest disadvantage of KNN when compared to other classifiers, that is, its lazy-learning characteristic.

2.3 Minimizing the Lazy-Learning Problematic

In [WW07] a solution to minimize the lazy-learning problem is proposed. The presented TFKNN method can quickly find the K nearest neighbors. The algorithm uses a SSR tree data structure (a type of $B^+ - Tree$) where each non-leaf nodes are classified according to the distance between its centroid and its parent's centroid. With this structure, it is possible to reduce the search scope, and hence the computation time. We may take an example of this structure in Figure 2.6.

The tree structure is as it follows:

- A non-leaf node is a minimal region that contains all its child nodes;
- All leaf nodes are at same depth and each leaf corresponds to an individual training sample;
- A non-leaf node represents a training set that contains: centroid, radius, distance between parent node and itself; pointer to parent node; number of children and pointers to them;
- Besides the sample, a leaf includes the distance and the pointer to its parent node.

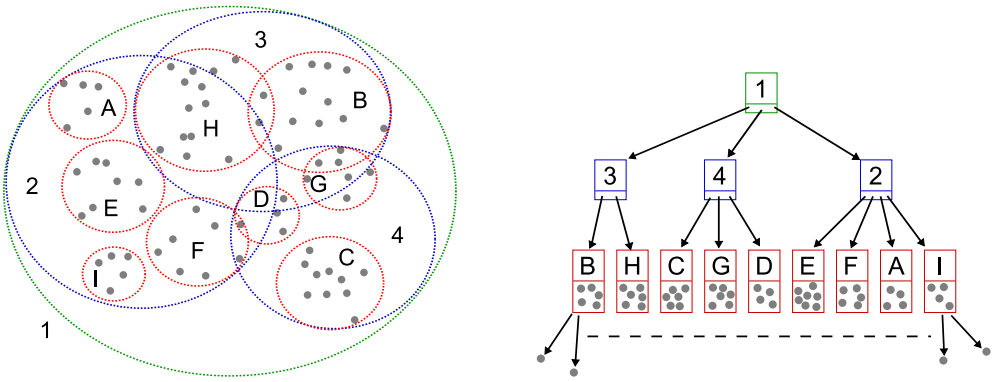

Figure 2.6: SSR Tree structure.

The tree must satisfy the following conditions:

- Root node has at least two child nodes, unless it is a leaf itself;
- All non-leaf nodes have a number of children between a pre-defined minimum and maximum;
- All non-leaf nodes are sorted in ascending order of distances between their centroid and its parent's center point.

Selecting the K closest elements is performed by seeking through the tree taking into account the closest nodes to the object being classified.

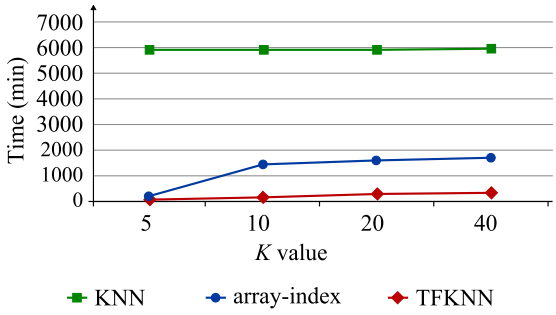

Figure 2.7: Classification time comparison on 1000 samples.

Figure 2.7 shows important improvements in classification time when compared to the original KNN and *array-index* KNN [YW05] (which is a method that uses the KNN rule applied to a *fuzzy decision tree*). However, the authors state that this approach needs to be improved in terms of classification accuracy, which suggests a decrease in performance at this domain.

The authors do not mention the problem regarding the nature of the attributes (categorical Vs. numerical) in the context of this approach.

In my point of view, the construction of the tree could be less exhaustive, changing slightly its structure. In other words, if in a given node, all samples below would belong to

the same class, then it would not be necessary to divide further down and the subsequent searches would be faster.

In [Jua11], the authors propose a tree approach in order to minimize the lazy-learning problem, the TKNN. The algorithm characterizes the training elements based on the highest similarity that separates the training set in *Core Sets*, thereby reducing the search range. Tree nodes correspond to each training set elements' average, in each core, being its elements close enough, according to the Euclidean distance. When a new training element comes up, the algorithm decides which Core Set it belongs to; the decision may result in the creation of a new Core Set.

Different sequences of scanning training set elements may end up building different trees. One of the objectives of the approach is to be able to build compact trees. TKNN's accuracy depends on the distance's radius value which was set initially and it is a parameter that influences the number of Core Sets. The tree nodes are therefore the Core Sets that belong to different regions, each one characterized by elements near to each other. Thus, different regions distribution determines tree structure and consequently its search efficiency in the classification phase.

In order to ensure a good performance, training samples are used to estimate the best radius distance to be used.

According to the results, in some tests, TKNN shows better accuracy values; but slightly worse for some data sets. The algorithm takes linear time proportional to the training set size.

In my point of view, this approach has a drawback: if class distribution is not separated enough in the vector space, each Core Set will represent a region containing multiple classes elements, which would lead to misclassifications as is shown in Figure 2.8. The Core Set depicted by the lower circumference contains elements of more that just one class could result in subsequent misclassifications.

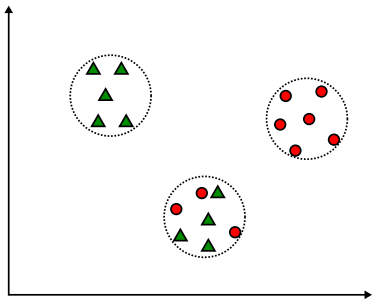

Figure 2.8: Identified Core Set examples.

An algorithm is presented in [PQS04] that aims to improve the original KNN's accuracy and which scope is the pattern recognition in image domain. This technique uses an important tool *Mean Vector*, which is used to reduce the search range. The core idea of this approach focuses on rejecting vectors that are impossible to be included in the K

closest vectors set. Thus, classification time is reduced and the accuracy of the classification keeps close to the original KNN, according to the authors. Basically, to find the K closest elements, the original vectors are transformed by *Haar Wavelet* and sorted by their *Approximate Coefficients* (the Mean Vectors) calculated in the same Wavelet scope - see [PQS04] for more details. To extract image features, the spline wavelet is used. According to the authors, when the size of features vector is not a power of 2, the length of the vector transformed by Haar Wavelet increases, which implies the need for more computational memory. The authors do not discuss the potential application of this technique in different contexts rather than images, taking into account the attributes of different data types (numerical Vs. categorical).

2.4 Resume

Given the weaknesses mentioned in the introduction of this chapter and the solutions proposed by different authors, the following should be emphasized:

(a) The training sample reduction proposals may lead to misclassification, especially in the cases where class boundaries are not well defined. However, in many cases, it ends up by improving classification. This group of solutions does not aim to solve the lazy-learning problem;

(b) In the scope of weighted attributes, the proposed solutions are efficient if the attributes are expected to have different discriminant capabilities or there is some redundancy between them; but only in these cases. Regarding the efficiency of assigning weight to votes depending on the distance of nearest neighbors, we can say that in cases where the training set is distributed into clearly separated class' clouds, these techniques minimize the sensitivity regards the user's selected K value. In fact, if the classes are clearly separated, for any K chosen value, there is a tendency for the nearest neighbor to determine the class assignment, since it got the most weighted vote, hence reducing the K value sensitivity. However, this weight assignment becomes detrimental in cases where the quality of the attributes does not allow a clear classes distribution into separate clouds. In short, the impact of assigning weights to attributes is highly dependent on the quality of the attributes. This class of solutions does not address the lazy-learning issue, as well;

(c) In order to minimize the lazy-learning problem, the trees structures have shown great potential as they reduced classification times. However, the solutions proposed in this scope still have some weaknesses on the building process; although they can be simplified, not being as deep. This point has consequences in terms of temporal and spatial complexity. Besides, choosing representing elements from local clouds may result in a classification accuracy decrease, when clouds contain different classes. In this scope, it is particularly important that trees can have the ability to accommodate

attributes of different natures (categorical, numeric, ...). This issue is not mentioned by the authors.

3

Improving KNN – Solutions

Considering that the original KNN does not use previous classifications as a learning source, this algorithm tends to be very inefficient when working with large data sets, as it was mentioned before.

In order to cope this problem, providing KNN algorithm some intelligence, we considered a set of alternative structures organized in order to implement a learning (training) phase. Although this training phase may be heavy, it happens only once for each data set. Then, once this phase is completed for a given data set, the performance in the classification phase is highly rewarding.

3.1 Structures to Support the Learning Phase

The learning phase consists of two steps: division and pre-classification. In the division, each dataset attribute represents a dimension. The range of each attribute (represented in Figure 3.1(a)) is segmented, so the original vectorial space is split successively in smaller blocks (as show in Figure 3.1(b) and Figure 3.1(c)) while the data structure is created. The final blocks granularity is defined by a criterion which be explained later. In order to conclude the learning phase, each one of those final blocks is associated with a class.

The idea is to divide the vectorial space such that there is a very high probability that each block is *populated* by elements of just one class.

As it was mentioned before, alternative structures were considered to implement both phases (learning and classification), so they can be compared. Thus, each alternative is presented below in four steps: introduction, learning phase (division and pre-classification), classification phase and resume. Examples will be given regarding a simple dataset (Figure3.2) that contains fifteen samples over two dimensions and three classes.

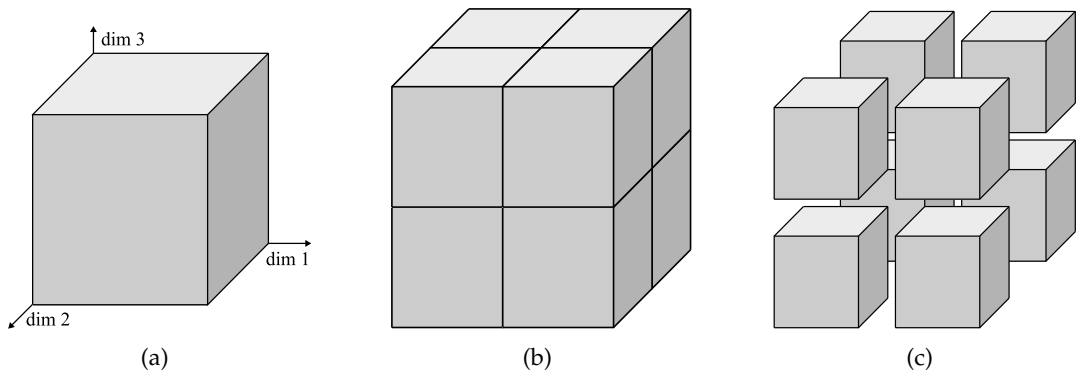

Figure 3.1: Data structure division step.

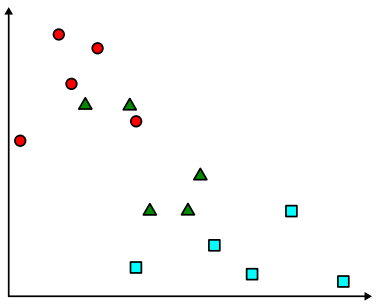

Figure 3.2: Simple dataset with fifteen samples over two dimensions and three classes.

3.1.1 Fixed Grid

The first alternative approach regarding the implementation of an intelligent KNN algorithm consists of a rough division of each dimension range as it is divided into I equally intervals.

Basically, Fixed Grid transforms the dataset into a multidimensional matrix.

3.1.1.1 Learning Phase

After computing interval size for each dimension, the resulting intersections will be represented in a I^D matrix, which I call pre-classification matrix, where D is the number of attributes.

As the matrix structure is set, each cell now will represent a region limited through the interval size of each dimension. Each one of those regions will associate a class. This associated class will be the result of classifying the region's central point using the original KNN. This means, comparing the distances from the region's central point to the whole dataset and selecting the majority class in the K nearest samples. Thus, the learning-phase space complexity is $\mathcal{O}(I^D)$ and this phase has an $\mathcal{O}(I^D \times ND)$ time complexity, where N is the number of samples in the dataset.

In the example below (Figure 3.3) each dataset attribute range is divided in three equal divisions.

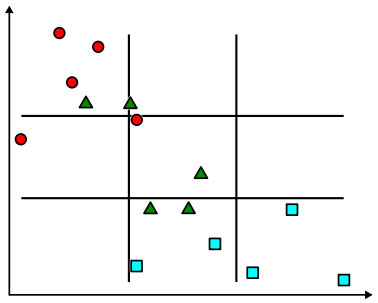

Figure 3.3: Fixed Grid structure using three intervals.

3.1.1.2 Classification

The upside of this structure is the classification-phase time. The classification of a new element (or a sample) involves computing each dimension's index and accessing the pre-classification matrix. Recall that each pre-classification matrix cell contains an associated class representing the whole cell region. Therefore, classifying a new element is an $\mathcal{O}(D)$ time operation.

Looking at Figure 3.3, if we want to classify a new element whose attributes, for example, point to the top left rectangle $[1, 3]$, and $(K = 5)$ is used, its pre-classification will associate the "red circle" class two this new element. Actually, if we were about to classify any sample within this cell region, it would always result in "red circle" for $(K = 5)$.

3.1.1.3 Resume

Although being a very simple way for providing KNN with intelligence, it is also a very rudimentary and rigid method due to its inability to adapt to the dataset. Since every dimension is divided into the same I equally spaced intervals, this does not guarantee that each cell contains a clear majority of elements of the same class. Besides, it may result in contiguous empty cells (having no samples) or it may result in an avoidable over-division of the space when contiguous cells are populated with samples from just one and the same class.

The learning-phase time complexity and the disadvantages mentioned above motivated me to research other alternative structures. However, this will be used as base of comparison to other structures.

3.1.2 QuadTree

The QuadTree structure is a first evolution from the Fixed Grid.

As said before, Fixed Grid structures division is susceptible to produce *empty divisions* and over-divided regions. In order to solve these problems, a matrix based structure still would not be optimal. To maintain the advantages of the fast indexing regions, this QuadTree solution is based on a tree data structure.

The QuadTree is a tree based structure in which each node may have two child nodes

for each attribute in the data set. Each node represents a region and all dataset samples contained in it. In this structure, a node does not contain child nodes (leaf) either because it does not contain any samples in its region or because its data has what we consider a majority class. When a subset of data consists of high percentage of a given class, it is considered to have a majority class. For this high percentage I choose 99%.

3.1.2.1 Learning Phase

Division is a recursive and breadth-first process that for a node (containing data) will split each dimension in two equal halves, resulting in 2^D child nodes, where D is the number of dimensions. Each one of these child nodes will repeat the process until either it has no data or a majority class arises.

This is considered to be a more efficient method due to its division rules since empty regions will not be divided, therefore there will be no "over-computation". The same happens to divisions where a class is considered to be the major class. In other words, this resolves the problem of the overly divided regions. These two situations are presented in the example at Figure 3.4.

After this vector space division phase is finished, a pre-classification phase (similar to the one used in the Fixed Grid) is applied. In this phase, each leaf will be associated to a class. This class may be set by one of the two following ways:

- Simple Distance Heuristic (SDH) – as in the the Fixed Grid structure, each leaf's central point will be used to compute the distance from itself to each of all data sample; then, as in the case of Fixed Grid, the nearest K samples decide which class represents the region;
- Hierarchical Distance Heuristic (HDH) – it is similar to SDH, but instead of considering all data samples, it just takes into account the K closest samples considering the hierarchical structure where the leaf's central point belongs to. In other words, if the number of samples within the leaf's region is less than K, the leaf's parent level is considered in order to get enough samples (K). If still not enough, this step may be repeated. Again, as in the resulting data samples are used as in SDH to locate the K nearest ones and the leaf's class representative.

The process that defines QuadTree structure regions, this is, the dividing phase, takes in average, time $\mathcal{O}(N)$ using SDH heuristic, while its spatial complexity will be $\mathcal{O}(N)$. For the pre-classification phase, it takes time $\mathcal{O}(LND)$, where L if the number of resulting leafs after division phase. When using HDH, this pre-classification phase is more dependent on the data distribution through classes then the SDH.

3.1.2.2 Classification

To classify a sample is as direct as search through the pre-classification tree. This means the classification of a sample has complexity $\mathcal{O}(\log_{2^D}(N))$.

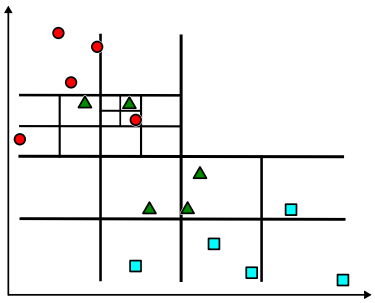

Figure 3.4: QuadTree structure example.

3.1.2.3 Resume

The main advantage of this QuadTree structure is that it handles regions better, avoiding to over compute either blank or meaningful regions. This results in an improvement on both training and sample classification times.

3.1.3 KD-Tree

As the QuadTree tree structure makes successive *blind divisions*, always splitting each dimension in two halves. This isn't optimal in datasets containing really dense clusters (not containing a majority class). The problem with this is that successive iterations may not be splitting those clusters. KD-Tree structure tries to overcome that brack.

Like QuadTree, KD-Tree is also a tree based structure. The big difference between these structures is the way they divide the dataset attribute ranges. Instead of having each node dividing all dimensions in two halves, each KD-Tree node divides just one dimension at a time based on its dimension subsample mean value. Its child nodes will repeat the process using other dimension.

Each of KD-Tree node has a flag indicating if the node itself is a left or right child, the dimension and value where this node spits the subset and it may have two child nodes.

3.1.3.1 Learning Phase

Like QuadTree, KD-Tree division is a recursive and breadth-first process. Each node computes the contained subset mean value over a given dimension. This value is set to the node and it is used to split the contained subset in two parts. Each one of these child nodes will repeat the process using another dimension until either it has no data or it has a majority class.

This way, even the clusters that might took QuadTrees structure several iterations to start dividing, would be detected and split earlier.

After the KD-Tree structure is settled, once again, each leaf will be associated with a class same way as QuadTrees. This means using one of the two heuristics referred in the previous subsection (in Sub-subsection 3.1.2.1).

Creating the tree structure for KD-Tree when using SDH heuristic takes in average time $\mathcal{O}(N)$ as spatial complexity will be $\mathcal{O}(N)$. Pre-classifying this tree structure, as in QuadTree, takes time $\mathcal{O}(LND)$.

In Figure 3.5 it is possible to notice how divisions are made at each iteration on each axis (dimension). Every division considers the mean value of its dimension/attribute subset.

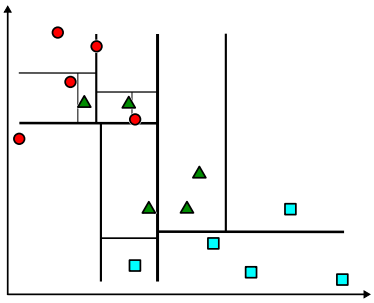

Figure 3.5: KD-Tree structure example.

3.1.3.2 Classification

Analogously to QuadTree, a sample's classification is as complex as searching through the pre-classification tree. This means the classification of a sample has complexity $\mathcal{O}(\log_2(N))$.

3.1.3.3 Resume

Contrary to QuadTree that may overloop in very condensed clusters, KD-Tree forces its separation in early iterations. Also, because KD-Tree uses dimension's mean values to split subsets, it does not allow the formation of blank divisions, which results in fewer divisions. These are the reasons why KD-Tree's learning phase turns to be faster than QuadTree.

We can compare both examples shown in Figure 3.4 and Figure 3.5. Figure 3.4 resulted in a QuadTree with 22 leafs which 10 have no data in its regions while Figure 3.5 shows that KD-Tree divided the same set on 11 leafs.

3.1.4 Interval Tree

Although this structure still follows the same ideas as the previous ones, it has a different approach regarding the division phase.

The problem when making divisions in the QuadTree and KD-Tree structures is defining the dividing point on the dataset. Instead of roughly dividing the whole dataset, Interval Tree (InTree) structures tries to divide the set in order to minimize the number of future divisions.

Despite the differences regarding division phase, InTree structure is the same as the KD-Tree as pre-classification and classification phases work the same way. However,

InTree nodes have three additional informations: a flag indicating if the node itself is a left or right child; the dimension identification and value where it spits the subset. Each node may have two child nodes.

3.1.4.1 Learning Phase

Instead of trying to divide the dataset until it either has no data or it has a majority class, for each InTree iteration, it computes all class ranges at a given dimension. The class whose limit is closer to the dimension's mean value of the dataset will be used for dividing the set (as shown in Figure 3.6). This way, every iteration guarantees at least one class is full contained in one of these new divisions.

As in the previous structures, the final tree structure will be scanned and each leaf will be associated to a class. This association is made as before, using either of the two presented heuristics. This is: either by running the original KNN algorithm, using its leaf's central point to compute distances to the whole set, or considering the first K samples in its hierarchy.

Once the InTree is slightly different than KD-Tree, so its division phase is also different and takes time $\mathcal{O}(CN)$, where SDH heuristic is considered and C represents the number of classes in the dataset. This difference can be explained by the need of finding the class' limit which is the nearest one to the dimension's center on each iteration. The spatial complexity for this phase is $\mathcal{O}(N)$. Similarly to QuadTree and KD-Tree structures, pre-classification time $\mathcal{O}(LND)$.

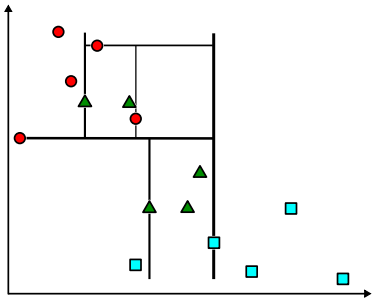

Figure 3.6: InTree structure example.

3.1.4.2 Classification

Classifying a sample using the InTree structure is just like QuadTrees and KD-Tree structures where its complexity is as searching through pre-classification tree, this means $\mathcal{O}\log_2(N)$ on average.

3.1.4.3 Resume

This structure has a very simple philosophy. It is expected that this approach results in fewer divisions than QuadTree and KD-Tree. In fact, despite the example dataset (Figure

3.2) contains no clearly separated clusters, InTree final structure contains just 7 leafs. Comparisons concerning performances of these different approaches will be shown in Sections 4.2 and 5.2.

3.2 Weighted Attributes

In general, a dataset may be seen as the content of a single database table where each column represents a characteristic. Those characteristics (attributes) may be indispensable to define a group's class or may not be necessary at all. The attributes choice should be careful and meaningful, but in practice, there are attributes which have more discriminant power than others. In these cases, KNN classification may be mislead once all attribute are treated as having the same importance. One way to face this problem is setting a weight to each attribute. These weights are usually used as a multiplying factor in order to set different importances to the attributes according to each attributer's discriminating capacity.

There are already some approaches regarding this methodology, however, once data tree structures were adapted in the context of this thesis, an adapted weight setting approach was also developed in which we believe it may help to increase classification precision and potentiates the reduction of attributes (feature reduction).

In order to assign weights to attributes, it is necessary to create an evaluation method for setting their discriminant power. We could say that the ideal attribute would present different average values for each class, but a constant value within each class. Figure 3.7 shows an example where attribute 1 can be considered discriminant but attribute 2 cannot. In fact, attribute 1 presents different centroid (average) values for each class, but similar centroid values for attribute 2. Besides, attribute 2 present higher dispersion values considering their centroid for class i and class j, than attribute 1. These two factors indicate the attribute 1 should be considered better than attribute 2.

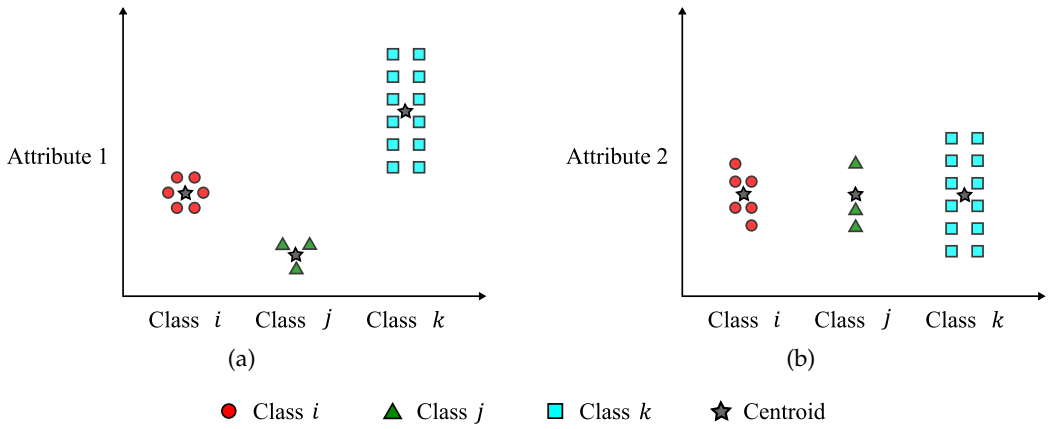

Figure 3.7: Different attribute set discriminant power example.

Thus, we value two factors in attributes: higher variances among class centroids and

lower variance within each class regarding its elements.

We define *Importance* of an attribute (Equation 3.1), in short, as being the quotient of how much the attribute value deviates through classes and how much it deviates inside each class. In other words, we can set the Importance as being the ratio between the variance of the average value of that attribute for each class (Equation 3.2) and the average of the local variances (Equation 3.5). By local variances we mean the variance of the attribute value within each class, regarding its elements (Equation 3.6).

Keeping this idea in mind, we can define the Importance of attribute a, $Imp(a)$, by the expressions below presented.

$$Imp(a) = \frac{VM(a)}{MV(a)} \tag{3.1}$$

$$VM(a) = \frac{1}{\#Classes}. \sum_{c_i \in Centroids} (c_i - \overline{c})^2 \tag{3.2}$$

$$\overline{c} = \frac{1}{\#Classes}. \sum_{c_i \in Centroids} c_i \tag{3.3}$$

$$c_i = \frac{1}{\#Elements\ in\ Class_i}. \sum_{e \in Class_i} v_{e,a} \tag{3.4}$$

$$MV(a) = \frac{1}{\#Classes}. \sum_{c_i \in Classes} LV(a, c_i) \tag{3.5}$$

$$LV(a, c_i) = \frac{1}{\#Elements\ in\ Class_i}. \sum_{e \in Class_i} (v_{e,a} - v_{.,a})^2 \tag{3.6}$$

$$v_{.,a} = \frac{1}{\#Elements\ in\ Class_i}. \sum_{e \in Class_i} (v_{e,a}) \tag{3.7}$$

To calculate how much an attribute a deviates through classes we start by computing each class centroid c, this is, computing mean value for attribute a using class elements $v_{e,a}$ as expressed by expression (Equation 3.4). Having this result, the average centroid value is set as $\overline{c}$, shown in expression (Equation 3.3). Then, these values are used to determine how much those centroids deviate from their average value, and it is set as VM by expression (Equation 3.2). As it was mentioned, the higher the $VM(a)$ value the more important the attribute is, it will be placed as the dividend of $Imp(a)$.

In expression (Equation 3.6), each local variance, $LV(a, c_i)$ value uses all attribute a values of class c elements, represented as $v_{e,a}$ above and computes its mean value as $v_{.,a}$ as in expression (Equation 3.7). As lower local variances correspond to stronger attributes, $MV(a)$ was placed as the divisor of $Imp(a)$. The lowest possible value for $Imp(a)$ is 0, corresponding to a useless attribute. There is no upper bound for this metric.

3.3 Converting Categorical Data to Continuous Data

As mentioned earlier, continuous attributes such as ages, heights, weights, etc, allow precise measurements. Any data set of this data type can be placed in ascending or descending order. So, the distance/difference between two values can be highly accurate computed.

On the other hand, categorical data usually represents characteristics such as gender, blood type, marital status, etc. Despite categorical attributes may take numerical values, they may not represent mathematical values, for example categories codes or color representations, among others. Therefore there is no intuitive way to compute differences between two values from different categories of this data type. Typically, this problem is solved by setting this difference as 1 if values are different or 0 if they are the same. Not only does this method is not sensitive enough to accurately evaluate categories regarding their real *distances*, but it also does not take into account the impact of the category values in terms of class distribution. Moreover, using categorical data on the data structures presented before would result in highly inaccurate classifications, once the approach we are using is highly dependent on the accuracy of distance measurements.

To address this problem, instead of adapting the structure in any way that would fit this data type, our approach involves transforming categorical data to numerical data. In other words, for each categorical attribute, we transform all possible values into numeric values. The major obstacle applying this method is that even if it was possible to assign distinct numeric values to each categorical value of a categorical attribute, it is even harder to set the displaying order.

Table 3.1: Eye color distribution over hemispheres.

Gender	Eye Color	Hemisphere
Male	Blue	North
Male	Green	North
Male	Green	North
Male	Green	South
Male	Green	South
Male	Dark	North
Male	Dark	South
Female	Blue	North
Female	Blue	North
Female	Green	North
Female	Green	North
Female	Green	South
Female	Dark	North
Female	Dark	South
Female	Dark	South

As an example, by Table 3.1, apparently there is no obvious way to order the different

colors concerning the ability to discriminate both classes (North or South).

However, this table hides important information that can lead us to find such order. The transformation proposed in this thesis takes conditional probabilities as its base. Thus, if we consider the given example for the eye color attribute, we get the following conditional probabilities:

Table 3.2: Eye color conditional probabilities.

	$\boldsymbol{P}(\textbf{EyeColor}\|\textbf{Hemispher})$	
Eye Color	**South**	**North**
Blue	0	0.33
Green	0.5	0.44
Dark	0.5	0.22

Considering the South and North classes as being the edges of the interval $[0, 1]$ respectively, it is possible to compare different eye color values. If we consider the blue eyes conditional probabilities (from Table 3.2), even if it is known that only 33.3% of all individuals in the north hemisphere have blue eyes, we can conclude that any individual who has blue eyes is for surely from the north hemisphere. It is possible to make such affirmation due its representation in the overall world, due to its percentage: $\frac{0.33}{(0+0.33)} = 1$. This introduces a function $\mathit{ScaleCat}$ defined as:

$$\mathit{ScaleCat}(Cat, Class_i, Class_j) = \frac{P(Cat|Class_i)}{P(Cat|Class_i) + P(Cat|Class_j)}$$

where for example, Cat is "Blue" and $Class_i$ and $Class_j$ is "South" and "North". When applying this function to all color eye attribute values we get:

$$\begin{aligned}\mathit{ScaleCat}(\text{"}Green\text{"}, \text{"}North\text{"}, \text{"}South) &= \frac{P(\text{"}Green\text{"}|\text{"}North\text{"})}{P(\text{"}Green\text{"}|\text{"}North\text{"}) + P(\text{"}Green\text{"}|\text{"}South\text{"})} \\ &= \frac{0.44}{(0.44 + 0.5)} \\ &= 0.47\end{aligned}$$

$$\mathit{ScaleCat}(\text{"}Dark\text{"}, \text{"}North\text{"}, \text{"}South\text{"}) = \frac{0.22}{(0.22 + 0.5)} = 0.31$$

$$\mathit{ScaleCat}(\text{"}Blue\text{"}, \text{"}North\text{"}, \text{"}South\text{"}) = \frac{0.33}{(0.33 + 0)} = 1$$

Since the goal is to determine the distances between all categorical values, no matter how close they are from each interval extreme (0 or 1), it is only important to maintain the class order when calling the $\mathit{ScaleCat}(.,.,.)$ function over each categorical value. In other words, if $\mathit{ScaleCat}(\text{"}Green\text{"}, \text{"}South\text{"}, \text{"}North\text{"})$ is used to compute the position of "Green" in the "South" – "North" scale, then, in order to compute the position

of "Blue" in that scale, only $ScaleCat("Green","South","North")$ must be used; not $ScaleCat("Green","North","South")$.

This approach enables more thorough measurements since distances do not only take the two typical values (0 and 1 for different and equal classes respectively). Although it would be easy to say that individuals having blue eyes would be somehow more related with the north hemisphere, we were not able to quantify how much they would differ from dark and green eyes individuals. By this approach we transform data in such way we can even say that the difference between green and dark eyed individuals is smaller than either of them towards blue eyed individuals. In a general perspective, this means that we can generate an axis as show below in Figure 3.8.

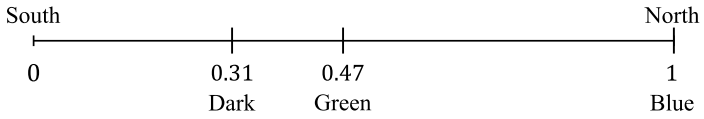

Figure 3.8: Eye color attribute transformation axis.

The same process must be applied to the other attribute and its categories, that is, $ScaleCat("Male","North","South")$ and $ScaleCat("Feale","North","South")$ must be computed.

This method ables the comparison of categorical attributes values between any two classes. So in this particular example, the dataset which contains just two categorical attributes over two classes would be transformed into another dataset but containing two numerical attributes.

In a general way, any dataset containing A categorical attributes over K classes will be transformed into a new set containing $\binom{K}{2} \times A$ numerical attributes. This may end up in a large number of new attributes, as being the result of all combinations representing all possible pairs of classes. Apparently, this turns out to be the downside of this solution once increasing the number of attributes implies a greater computational weight in the learning phase. Despite that, it may happens that not all new attributes are meaningful once their implicit $ScaleCat(.,.,.)$ values present no discriminant ability in the context of some pairs of classes. Thus, it is possible to use the previously presented attribute's weights or PCA (Principal Component Analysis) to select and use only the more meaningful attributes.

In [CAdC11], the *Value Difference Metric* measures distances between pairs of categorical values. Although, by using it, it is not possible to sort those values in a numerical axis. The authors in [IPM09] also focus the problem of defining a distance between pairs of values of the same categorical attribute. They consider it to be a difficult task once those categorical values are not ordered. Instead of the learning-classification context, the authors try to obtain good clustering performance by using numerical instead of categorical data. The key intuition of their approach, *DILCA*, is that the distance between two values of a categorical attribute X can be determined by the way in which the values of the other attributes Y are distributed in the dataset objects: if they are similarly distributed in the groups of objects in correspondence of the distinct values of X a low value of distance is

obtained. More specifically, it is a two-step method. In the first step, for each categorical attribute X, first a suitable context constituted by a set of attributes $Y \neq X$ is identified, such that each attribute belonging to the context is correlated to the attribute X. In the second step, a distance matrix between any pair of values (x_i, x_j) of X is computed: the distribution of x_i and x_j in objects having the same values for the context attributes is taken into account. The distance calculation is based on the *Symmetric Uncertainty* and Entropy; See [IPM09] for more details. The authors present good accuracy values for clustering formation, reaching 95% for some cases. However, due to different context where this thesis lays (leaning-classification context), accuracies obtained are not comparable. Nevertheless, when compared to DILCA method, we may say that the solution proposed in this thesis to transform categorical data needs no information about the nature of data distribution.

4

Results

4.1 Dataset Collection

In order to test the developed tools, a collection of datasets were considered. Gathering datasets was somehow a subjective task since there were no evaluation methods to select them. Still, some characteristics were considered. The content of the considered datasets should be, if possible, meaningful or helpful to some purpose. Another important feature in this gathering was the number of classes in the dataset and having a reasonable balance between class distributions. Furthermore, in order to evaluate the tools proposed in this thesis, dataset should have a considerable number os attributes (at least three).

Restrictions on attribute's quality were not applied, since feature correlation and low quality of the attributes is a reality and may be present in any dataset.

To test the developed tools, it was essential to have datasets based on numerical and categorical data. Considered datasets containing numerical data were:

- Iris (IRIS) dataset is one of the most popular dataset in pattern recognition. It describes 3 types of iris plant over 4 numerical features reflecting sepal and petal length/width. It has a perfect balanced class distribution, having 50 records for each of 3 classes;

- Vehicles is a computer generated dataset that tries to mimic what could be the result of a mundane vehicle inspection using a radar at any point on the motorway. It has 3100 records and 3 numerical features which describe 3 classes of vehicles: bikes, cars and trucks. Bikes represent about 19.4% of data while cars 64.5% and trucks 16.1%;

- Auto-Mpg Data (AMPG) is an adapted dataset which aims to classify the origin of an automobile based on 4 numerical attributes: fuel consumption in mpg, engine displacement, engine horsepower and automobile's total weight. In this dataset, the whole 392 automobiles can come from 3 continents: America, Europe or Asia, whose distributions are respectively 62.5%, 17.3% and 20.2%;

- Haberman's Survival Data (Haberman) contains 306 cases from a study conducted between 1958 and 1970 at the University of Chicago's Billings Hospital on the survival of patients who had undergone surgery for breast cancer. Each case is characterized by 4 numerical attributes: age of patient at time of operation, patient's year of operation and number of nodes detected. Each case is classified into two classes which indicate if patient has survived or not within 5 year. In 73.5% of the cases, patients survived;

- Thyroid Disease (TD) is a study on hyperthyroidism regarding 215 subjects. Each subject is characterized through 5 numerical features regarding hyperthyroidism tests: T3 Resin, Thyroxine and Triiodothyronine hormones and 2 Thyroid-Stimulating Hormone tests. Hyperthyroidism is present in 16.3% of the subjects;

- Blood Transfusion Service Center Data Set (BT) contains information from a Blood Transfusion Service Center in Hsin-Chu City in Taiwan. This dataset was elaborated in order to build a FRMTC model and describes 748 donors at random from the donor database. Donators data consist on 4 numerical attributes: months since their last donation, number of donations, total blood donated in c.c. and the number of months since first donation. Individuals are separated in two classes, representing whether if he/she donated blood in March 2007. Having 2 classes, 23.8% of the individuals have donated at the specified month. The datasets based on categorical data were:

- Census-Income (Census) is also one of the most popular dataset in pattern recognition. It consists on an extraction done by Barry Becker from the 1994 Census database. From the 30725 samples considered in this dataset, about 24.9% are classified as making over 50K a year. Each sample has categorical information relative to a person's: age, work class, education, marital-status, relationship, race and gender;

- Car Evaluation Database (CED) contains information regarding automobile quality. It was derived from a simple hierarchical decision model originally developed for the demonstration of DEX (a system shell for decision support). This dataset contains 1728 examples with 6 categorical attributes: buying price, maintenance price, doors, passenger capacity, luggage boot size and safety. Automobiles can be classified in 4 quality classes: "unacceptable", "acceptable", "good" and "very good", whose distributions are respectively 70%, 22.2%, 4% and 3.8%;

- Nursery Dataset (Nursery) reflects a hierarchical decision model originally developed to rank applications for nursery schools. It was used during several years in 1980's when there was excessive enrollment to these schools in Ljubljana, Slovenia, and the rejected applications frequently needed an objective explanation. The 12960 subjects in dataset were evaluated based on several characteristics: parents' occupation, child's nursery, form of the family, number of children in family structure, housing conditions, financial standing of the family and family's social and health conditions. Final classification may result in 5 possible evaluations which are distributed in the following way: "not recommended" (33.33%), "recommended" (0.02 %), "very recommended" (2.53%), "priority" (32.91%) and "special priority" (31.21%);
- Wisconsin Breast Cancer Database (WBC) was obtained from the University of Wisconsin Hospitals, Madison from Dr. William H. Wolberg. Its data contains information about 699 individuals whose nodles had been tested for breast cancer. Each subject's test is characterized over 9 features: clump thickness, uniformity of cell size (and shape), marginal adhesion, single epithelial cell size, bare nuclei, normal nucleoli and bland chromatin mitoses. Having 2 possible classes, 65.5% of subjects are classified as "benign";
- Contraceptive Method Choice (CMC) is a subset of the 1987 National Indonesia Contraceptive Prevalence Survey. The samples are married women who were either not pregnant or do not know if they were at the time of interview. Those interviews result in a dataset with 1473 samples over 9 categorical features: age, education, husband's education, successful births, religion, employed (yes/no), husband's occupation, "standard-of-living index", media exposure. Every interview results in a contraceptive usage class. These classes distributions are: "no use" (42.7% of the dataset), "long-term" (22.6%) or "short-term" (34.7%);
- Hayes-Roth & Hayes-Roth (Roth) dataset involves human subjects study containing 132 instances. It has information over 4 categorical attributes: hobby, age, educational level and marital status. This dataset includes 3 classes, which are distributed in the following proportions: 38.6%, 38.6% and 22.7%.

Table 4.1 summarizes dataset's main characteristics.

4.2 Data Structures

4.2.1 Characterization of the Tests

Data structures were developed so that by introducing a learning phase, the classification of new objects would become faster. As important as reducing classification time, maintaining accuracy is imperative.

As data structures were developed to process numerical data, all numerical datasets were taken into account and used. In order to compare the data structures, all (numerical)

Table 4.1: Dataset collection features.

Name	Data Type	Samples	Attributes	Classes
IRIS	Numerical	150	3	3
Vehicles	Numerical	3100	4	3
AMPG	Numerical	392	5	3
Haberman	Numerical	306	4	2
TD	Numerical	215	5	2
BT	Numerical	748	4	2
Census	Categorical	30725	7	2
CED	Categorical	1728	6	4
Nursery	Categorical	12960	8	5
WBC	Categorical	699	9	2
CMC	Categorical	1473	9	3
Roth	Categorical	132	4	3

datasets were split 150 times into two disjoint subsets, the training set, containing 97% of the whole data, and test sets. Each time, one of these sets used the training set as the knowledge base and tried to classify the remaining samples (as new objects) in the test set resorting to all data structures. This way, in order to compare the performance of the different datasets, all datasets are processed using the same dataset split, considering several K nearest neighbors (up to 13). This was repeated for the 150 splits.

Those same data set splits will be maintained for testing weighted attributes, which are described in Section 3.2. Furthermore, the same data splitting method is applied to categorical data for testing categorical-to-continuous data conversion in Section 3.3. Original KNN and Fixed Grid structures performance values were used as reference.

Testing data structures also involves testing the following variants:

- Original KNN runs as expected, using euclidean distances. Its results will be used as basis for comparison for all developed data structures;
- Fixed Grid is a very straightforward structure as it does not include variants. As referred, it has disadvantages associated. However, as it was the starting point in developing other structures, its test results are relevant to follow up the structures enhancement process;
- QuadTree has 4 variants. The first variant interferes with the learning phase process, more specifically with each one of the two heuristics to be used, as it was detailed in the learning phase in Subsection 3.1.2. The second one relies on a study about the impact of data normalization in the final outcome, which can be used or not;
- KD-Tree uses the exact same 4 variants as the QuadTree. This was considered due to these data structures similarity;
- InTree is analogous to QuadTree and KD-Tree in so many aspects that it makes sense to have the same variants to be tested.

4.2.2 The Most Suitable K

Somehow, the burden of choosing the K value in the classification process of the Original KNN algorithm may be seen as a consequence of its lazy-learning nature. Using the non-lazy-learning approach proposed on this thesis, it is possible to recommend a K value (or a short set of K values) which were previously evaluated as being the most suitable one to get the best accuracy for each specific dataset.

Every time a data structure variant is tested on a dataset split (a run), the test set is classified 13 times, corresponding to the 13 considered K values. The highest and lowest accuracy obtained from these tests are kept, and the highest one can be recommend the user. As we will see in next subsection (Subsection 4.2.3), for each test combining a data structure variant and a dataset, terms $\%H$ and $\%L$ represent this extremes.

Lower extremes were not discarded from some tables and figures in order to show the influence K value has in the accuracy for each case.

4.2.3 Results

The following graphics are the results from the 150 runs on each possible combination between datasets, data structures and its own variants. To interpret these results it is mandatory to understand that each graphic represents the tests over a data structure variant. Each data structure graphic contains pairs os columns. Each pair represents the highest and lowest accuracy results on average for the 150 runs for a dataset – by highest and lowest we mean the best and the worst suitable K nearest neighbor accuracy value –. Our reference accuracy values are also present an intervals. The interval between the highest and the lowest accuracy values (in average) for Original KNN data structure is represented by two black small horizontal lines. For the Fixed Grid, interval is represented by the two red lines also meaning highest and lowest accuracy.

As an example, let us take the case of Figure 4.7, which depics the accuracy value results for the QuadTree structure tests using normalized data and SDH heuristic over all datasets. Thus, taking the BT dataset pair of columns, the dark and light blue bars represent respectively the best and the worst accuracy considering the different K values, for this QuadTree structure variant. The top and bottom black lines represent the best and the worst accuracy obtained for the considered set of K values using the Original KNN. The top and bottom red lines have the same meaning but for the Fixed Grid.

In this particular example, we can conclude that for BT dataset, this QuadTree structure variant classifies new objects obtaining higher accuracy values than both Original KNN and Fixed Grid. As for the lowest accuracy values, while it has been beaten by the Original KNN, it is still better than the Fixed Grid.

Analyzing Figure 4.1 lead us to the conclusion that by using data structures which introduce a new learning phase, we now are able to classify new objects several times faster then the original algorithm – notice that for a better reading, the milliseconds axis is not linear.

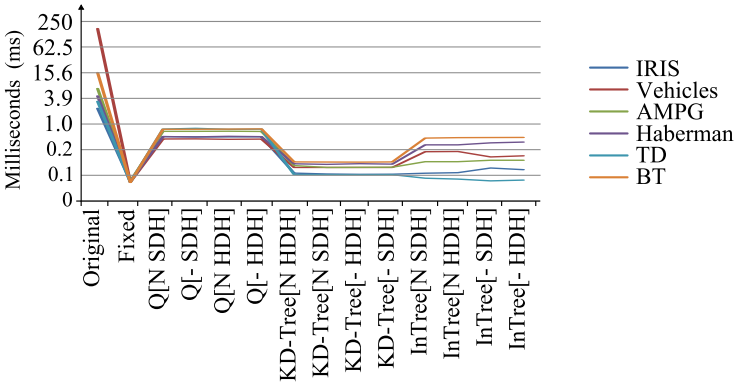

Figure 4.1: Data structures classification times.

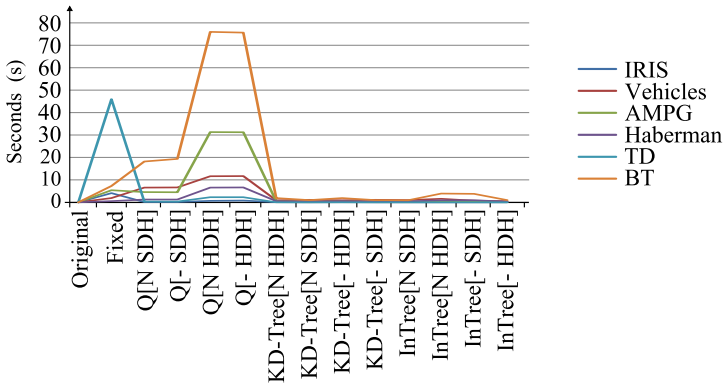

Figure 4.2: Data structures learning times.

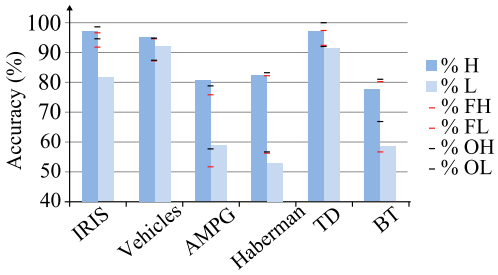

Figure 4.3: QuadTree classification test1s using HDH.

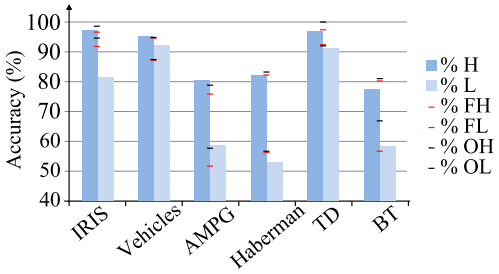

Figure 4.4: QuadTree classification tests using HDH.

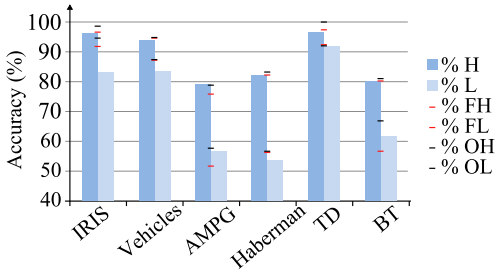

Figure 4.5: QuadTree classification tests using SDH.

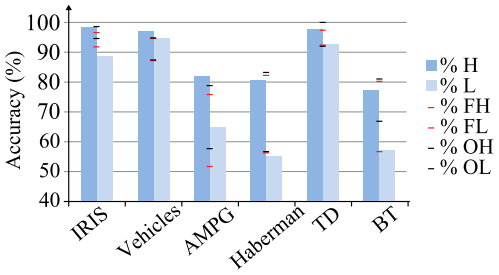

Figure 4.6: QuadTree classification tests using normalized data and HDH.

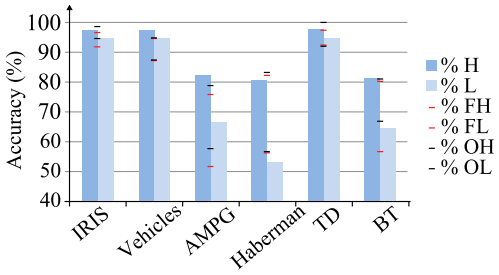

Figure 4.7: QuadTree classification tests using normalized data and SDH.

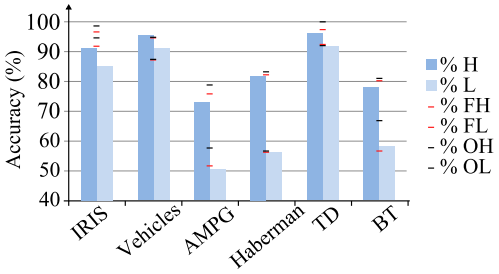

Figure 4.8: KD-Tree classification tests using HDH.

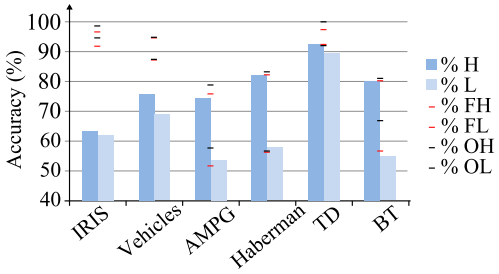

Figure 4.9: KD-Tree classification tests using SDH.

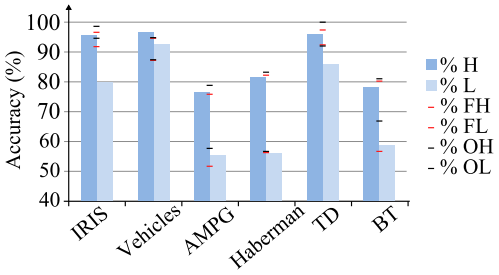

Figure 4.10: KD-Tree classification tests using normalized data and HDH.

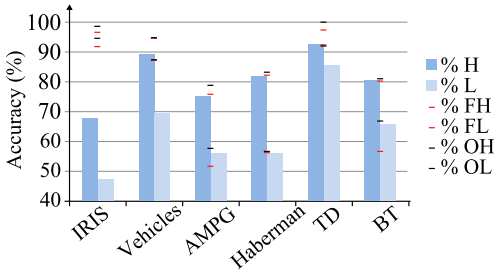

Figure 4.11: KD-Tree classification tests using normalized data and SDH.

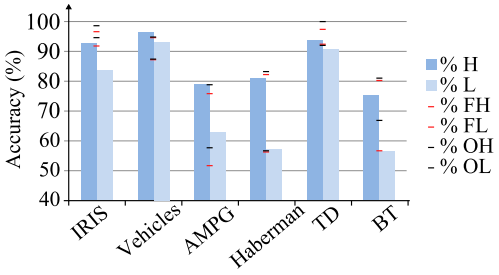

Figure 4.12: InTree classification tests using HDH.

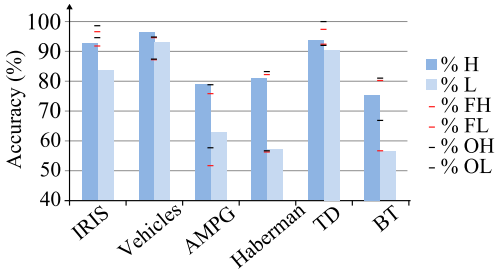

Figure 4.13: InTree classification tests using SDH.

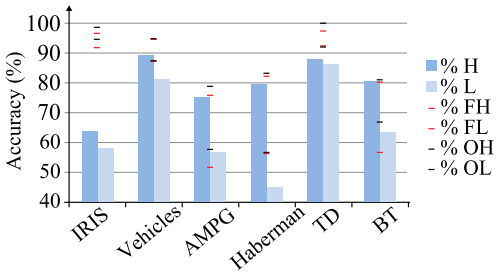

Figure 4.14: InTree classification tests using normalized data and HDH.

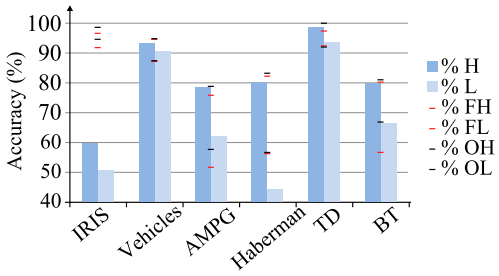

Figure 4.15: InTree classification tests using normalized data and SDH.

Despite not all data structures variants have similar learning times as is illustrated in Figure 4.2, variants using HDH heuristic instead of SDH tend to be slower. This fact has to do with HDH nature, which as referred in Sub-subsection 3.1.2.1, unlike SDH, this variant implies a local hierarchical searching which has a slightly higher computational weight.

Several conclusions may be extracted from Table 4.2 and Figures 4.7, 4.6, 4.5, 4.4, 4.11, 4.10, 4.9, 4.8, 4.15, 4.14, 4.13 and 4.12. Thus, regarding the most suitable K values, which corresponds to $\%H$ lines in tables for each data structure variant, not only some have a low accuracy loss as some actually increase it, comparing to Original KNN. The best case values found are when using the QuadTree with normalizing data and using SDH. As can be seen, this variant surpasses the Original KNN algorithm in 50% of all data sets used, and 83% for the Fixed Grid.

QuadTree with normalized data is also very good using the HDH, that is, QuadTree [N HDH]. In this case, it still beats the Original KNN in 50% of all data sets, but now only in 63% when comparing to Fixed Grid. Despite other data structures (KD-Tree and InTree variants) do not easily overcome the Original KNN accuracy values, they tend to lose about 6%.

Despite the good performance of QuadTree, where its accuracy is at the same level of the Original KNN, in a general way, all data structures using SDH heuristic have its accuracy 5.19%, in average, below Original KNN, but when using HDH heuristic that loss is only 3.23%.

Considering the learning phase, there is a big difference between QuadTree approach (Sub-subsection 3.1.2.1) and both KD-Tree (Sub-subsection 3.1.3.1) and InTree (Sub-subsection 3.1.4.1) concerning the splitting/division criteria. In fact, in each iteration, QuadTree splits every dimension in 2 equal halves, considering each dimension range, contrary to KD-Tree and InTree. Despite having small different division criterion, these two approaches, split in each iteration just one dimension at the time, which minimizes the number of final divisions – recall that a division is a final vectorial space delimited region where 99% of its elements are from the same class –. This is reflected in leaning phase time, which is faster for these two approaches. Although, this also means that the division granularity is coarser than in the QuadTree case. Taking into account the sample distribution in each dataset used, the results lead us to conclude that the more blend the dataset classes

Table 4.2: Classification accuracy for different data structures.

		IRIS	Vehicles	AMPG	Haberman	TD	BT
Original KNN	%H	**98.40**	**94.59**	**78.72**	**83.13**	**99.90**	**80.96**
	%L	94.53	87.24	57.50	56.67	91.81	66.81
Fixed Grid	%H	**96.53**	**94.41**	**75.72**	**82.07**	**97.33**	**80.09**
	%L	91.73	87.03	51.61	56.20	92.29	56.52
QuadTree [- HDH]	%H	**96.93**	**94.99**	**80.56**	**82.13**	**96.86**	**77.57**
	%L	81.60	91.99	58.94	52.87	91.14	58.49
QuadTree [- SDH]	%H	**96.13**	**94.01**	**79.17**	**82.20**	**96.67**	**80.20**
	%L	83.07	83.79	56.67	53.73	92.19	61.74
QuadTree [N HDH]	%H	**98.53**	**97.09**	**81.78**	**80.60**	**97.52**	**77.42**
	%L	88.80	94.44	64.67	55.07	92.57	57.22
QuadTree [N SDH]	%H	**97.33**	**97.30**	**82.28**	**80.27**	**97.62**	**81.33**
	%L	94.67	94.65	66.28	53.20	94.76	64.49
KD-Tree [- HDH]	%H	**91.33**	**95.46**	**72.94**	**81.80**	**96.10**	**77.86**
	%L	85.20	91.05	50.33	56.20	91.81	58.12
KD-Tree [- SDH]	%H	**63.47**	**75.68**	**74.11**	**82.07**	**92.48**	**79.83**
	%L	62.13	68.72	53.83	57.87	89.52	54.87
KD-Tree [N HDH]	%H	**95.47**	**96.70**	**76.44**	**81.80**	**96.00**	**78.09**
	%L	80.00	92.59	55.28	56.20	85.90	58.70
KD-Tree [N SDH]	%H	**67.73**	**89.07**	**75.11**	**82.00**	**92.48**	**80.61**
	%L	47.20	69.39	55.94	56.00	85.81	65.91
InTree [- HDH]	%H	**63.87**	**89.34**	**75.28**	**79.67**	**87.81**	**80.35**
	%L	58.00	81.36	56.72	45.07	86.19	63.51
InTree [- SDH]	%H	**92.80**	**96.33**	**79.00**	**80.93**	**93.71**	**75.39**
	%L	83.60	93.04	62.72	57.00	90.48	56.32
InTree [N HDH]	%H	**88.40**	**97.11**	**78.00**	**82.00**	**98.57**	**75.48**
	%L	78.13	94.76	61.67	57.87	96.57	56.81
InTree [N SDH]	%H	**59.60**	**93.10**	**78.56**	**80.27**	**98.57**	**79.91**
	%L	50.67	90.73	62.06	44.33	93.62	66.61

are in the vectorial spaces, the more the fine-grained divisions obtained by QuadTree favors the classification accuracy. The IRIS dataset, which is the most challenging one of this dataset collection, where classes are partially blended, is a good example of this behavior: QuadTree variants best K values lose in average $1.17\% = \frac{1.47\%+2.27\%+(-0.13\%)+1.07\%}{4}$ accuracy regarding the Original KNN, while KD-Tree and InTree lose 18.9% and 22.23%, respectively.

As described earlier in Sub-subsections 3.1.3.1 and 3.1.4.1, while KD-Tree splits regions considering sample's average value for just one dimension, InTree splits on the class border closest to the referred sample's average value. This small difference tries to approximate the *natural* borders among classes. Although, the results reveal slight differences, we can conclude that this evolution is positive as InTree increases accuracy values by 0.39% by comparison with KD-Tree.

These results invite us to research, in future, to keep the necessary granularity to

assure the good classification accuracy obtained with QuadTree, and to improve the InTree philosophy of finding the natural borders among classes as learning phase time is substantially lower than in QuadTree.

Although IRIS dataset caused some losses among data structure variants, just taking into account the rest of datasets, global accuracy loss for all data structure variant is about 2.24% – notice that, in this case, QuadTree variants present a 0.58% accuracy loss. Even including IRIS dataset, QuadTree variants lose, in average, only 0.68%. The best performance was obtained by the QuadTree [S SDH] variant where its average accuracy even surpasses the Original KNN algorithm by 0.07%, considering all datasets.

Analyzing the impact of normalizing data, after some simple computation, we can conclude there normalizing results in a slight decrease for KD-Tree and InTree variants. Although, a small increase was verified for QuadTrees.

Table 4.3 shows the recommended K(or short set of K) values to be used in classification phase, by the user, for each combination of variants and datasets. As an example, for classifying objects using the IRIS and QuadTree [N HDH] variant combination, the recommended K value is 3. This feature results from the non-lazy-learning nature of the global approach proposed in this thesis.

Table 4.3: Recommended K values for data structures variants.

		IRIS	**Vehicles**	**AMPG**	**Haberman**	**TD**	**BT**
Original KNN	%H	5 7 8 12	3	1	13	1	6
Fixed Grid	%H	13	2	1	13	2	4
QuadTree(- HDH)	%H	3	1	1	5	1 3	13
QuadTree(- SDH)	%H	5	2	1	13	4	13
QuadTree(N HDH)	%H	3	7	7	11	3	13
QuadTree(N SDH)	%H	3	8 12	10	13	3	10
KD-Tree(- HDH)	%H	4	3	1	13	2	13
KD-Tree(- SDH)	%H	3	3	1	10	1	6 8
KD-Tree(N HDH)	%H	2	7	1	13	9 10 12 13	12
KD-Tree(N SDH)	%H	2	8	1	13	10	2
InTree(- HDH)	%H	3 8	13	13	13	1 3 5 6 7 8	12
InTree(- SDH)	%H	3	1	3	12 13	12 13	6
InTree(N HDH)	%H	3	13	5	9	2 3 4 5 6 7 8	12
InTree(N SDH)	%H	12 13	13	4	11	9	13

4.3 Weighted Attributes

4.3.1 Characterization of the Tests

As said in the Subsection 4.2.1, in order to test the impact of adding weights in attributes, the previously generated 150 groups of training and testing data were reused, thus making it possible to test every data structure in the same conditions.

In each run, the training set is computed and the importance of attributes is applied. Once all classification accuracy tests are done, they can be directly compared with the ones in the previous subsection, where weights were not applied.

4.3.2 Results

Table 4.4 shows differences between accuracy results obtained in Table 4.2 and the tests when applying weights. As an example, classifying Vehicles samples using QuadTree [- SDH] improved its accuracy that before stood in 94.01%, when applying weighted attributes: it reached 96.66%, thus improving its percentage accuracy by 2.65.

Globally, by using attribute weights, we consider that accuracy slightly improves by 0.18% when using attribute weights. Some simple computations from Table 4.4 lead us to conclude that 75% of all tests (combinations of data structures variants and datasets) maintained accuracy, and 39% presented better results.

Although the general values in Table 4.4 are small, there is a relevant difference in KD-Tree [- SDH] case where it got the best accuracy improvement processing Vehicles samples (11.41%), but also the highest decrease for TD dataset (9.05%). This suggests that the performance of KD-Tree tends to be highly dependent on the dataset nature. We believe that by splitting the whole data by arbitrary dimensions (one at the time), it may happen that split dimension corresponds to a strongly weighted attribute, resulting in non optimized divisions.

During the computation of the weights, we noticed that attribute weights are, in some datasets, very different. This may point for some poor attribute quality. Thus, the cases where the number of attributes reaches some tens (which is not case of any of the datasets used), we should apply attribute weights in order to select the most discriminant/informative attributes and so, try to reduce the number of final features by discarding the meaningless ones.

Table 4.4: Data structures differences applying attribute weights.

		IRIS	**Vehicles**	**AMPG**	**Haberman**	**TD**	**BT**
QuadTree [- HDH]	%H	3.47	−1.86	−1.67	−0.13	−0.10	−0.12
QuadTree [- SDH]	%H	1.87	−2.65	0.11	−0.13	−0.19	−1.01
QuadTree [N HDH]	%H	0.00	0.00	0.06	0.00	0.00	0.09
QuadTree [N SDH]	%H	0.00	0.00	0.00	0.00	0.00	0.00
KD-Tree [- HDH]	%H	0.27	−1.03	−2.11	−0.87	0.29	−0.23
KD-Tree [- SDH]	%H	0.93	−11.41	−1.78	−1.53	9.05	−0.29
KD-Tree [N HDH]	%H	0.13	0.00	0.06	0.07	0.00	0.09
KD-Tree [N SDH]	%H	0.13	0.00	−0.06	0.00	0.00	−0.06
InTree [- HDH]	%H	−0.13	−4.39	−1.72	3.60	−0.29	0.78
InTree [- SDH]	%H	−1.07	−0.59	−0.72	2.80	−1.33	0.81
InTree [N HDH]	%H	0.00	0.00	0.00	0.00	0.00	0.00
InTree [N SDH]	%H	0.00	0.00	0.00	−0.07	0.00	0.00

4.4 Converting Categorical Data to Continuous Numerical Data

4.4.1 Characterization of the Tests

The goal of applying such conversion is to be able to use categorical information in the developed structures studied on this approach. Since the quality of the data structures was already proven in Subsection 4.2.3 in keeping low losses comparing to the Original KNN, now it is only essential to guarantee that the quality of this categorical-to-continuous data conversion is high, in order to maintain the accuracy level of the global classification process.

To measure the quality of data conversion, first we tested categorical datasets over the Original KNN as referred in Subsection 4.2.1. Then, we re-run the test but with previously categorical-to-continuous converted data. To avoid computational overweight, since this conversion may result in a significant larger number of new attributes, we applied PCA to filter the most important ones, limiting the number of resulting attributes to 7 – in future work this final selection can be automatic–. Finally, accuracy results for the same dataset information (both categorical and categorical-to-continuous converted versions) are compared.

4.4.2 Results

Table 4.5 (which is supported by Figure 4.16) shows that for all datasets, with no exception, accuracy values were increased either for the lower values as for the higher values – recall that %H and %L stand for the values obtained by the most and less suitable K values –. Since, in the context on the Original KNN, several K values must be considered, this table shows %H and %L values. For both cases and for all data sets, on average, accuracy is increased by 8.8%. Highest values increased by 5.52% on average while the lowest got increased by 12.07%, on average.

This improvement may seem strange for the reader, as the original data shows worts results than the converted one. However, this can be explained by the fact that the Original KNN distance criterion for measuring the distance tends to be very rough. As an example, let us suppose that a categorical dataset includes an attribute composed by three categorical values: "Under-18", "Adults" and "Senior", where all samples are classified in one of the following classes: "Attended" (A) and "Not attended" (NA), meaning that they a person may or may not have attended to a certain political debate. According to the Original KNN distance criterion for the categorical values, the distance between "Senior" and "Adult" is the same as the distance between "Senior" and "Under-18" since categfories are different in both cases. This crition is clearly rough as we can assume that "Under-18" people are not usually interested in politics, contrary to both other categories which we also assume they have similar attendance. This lack of sensitivity will be reflected in classification performance. By applying our $ScaleCat(.,.,.)$ function, where conditional probabilities capture these details, this conversion gains accuracy when measuring real

world differences between categorical values. In practice, in this case, the *gap* between "Senior" and "Adult" would be much shorter than "Senior" and "Under-18".

Table 4.5: Impact of converting categorical datasets into continuous numerical data.

		Original KNN [C]	**Original KNN [C2N]**
Census	%H	81.90	82.28
	%L	76.27	76.50
Cars Q	%H	92.41	96.08
	%L	75.54	89.49
Nursery	%H	81.37	92.34
	%L	66.95	89.40
BWC	%H	97.27	98.03
	%L	92.10	94.35
CMC	%H	46.34	50.70
	%L	27.05	34.76
Roth	%H	78.83	91.83
	%L	33.50	59.33

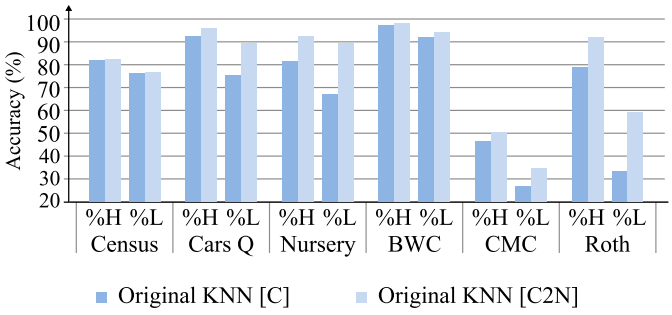

Figure 4.16: Data conversion accuracy variation.

5

Conclusion

5.1 Main Contributions

Next, the main contributions archived in this thesis are summarized.

5.1.1 A non-Lazy-Learning Approach for KNN Classifier

The main motivation for this thesis lied on addressing the lazy-learning problem of KNN algorithm. The approach developed in this work archived this goal. In fact, classification times were indeed reduced maintaining acceptable accuracy levels, and in some cases even increasing them. This is an important step since the Original KNN, we may say, *is not able to learn* because every time an object has to be classified, the original algorithm ignores the previous classifications results and as no knowledge from any learning process. Despite learning phase in this approach may take a while, once it is completed, classification times really pay it off. Due to this learning phase nature of this approach, classification time can be considered instantaneous.

Some variants based on three different data structures were adapted and developed in this context. Tests showed that QuadTree data structure using normalized data is able to guarantee the same accuracy level, as it loses in average 0.68% compared to the original KNN algorithm. The other data structures showed shorter learning phase times, but losing some accuracy, when compared to QuadTree.

5.1.2 Recommending a K Value for each Dataset

As a consequence of the developed leaning phase, it was possible to keep the most suitable K value for each dataset. Although this was not an initial goal of this thesis, the feature

of recommending a reliable K value to users came up in the path of this developments. As there may be more than one most suitable K value for each dataset, the user may be interested in using one those K values. So, the K is recommended instead of imposed.

5.1.3 Development of an Attribute Weight Criterion

Once there may be datasets with a large number of attributes, which would result in a very weighted computation process when using them on the developed approach detailed in Section 3.1, a metric to weight the informative power of attributes was developed. Test showed that this metric may help to increase classification precision and even potentiates the reduction of attributes, since some datasets' attribute set revealed a large weight range, when the developed metric was applied.

5.1.4 A New Approach for Categorical to Numerical Data

As the data structures developed in the context of this non-Lazy-Learning approaches were build considering numerical data, they did not fully support the information in the categorical datasets. So, this feature became a goal in our work path. Applying this new statistical method did not literally transform each attribute's categoric values into numerical values, but instead, it creates new scope values between pairs of classes. Based on conditional probabilities, this method translates categorical values into new attributes, according to their discriminant power within those pairs of classes. Hence transforming the whole previously categorical data into a new more informative dataset.

Test results revealed transformed datasets consistently increase significantly KNN classifier accuracy levels when compared to the same datasets categoric values. In fact, those accuracy levels increased 12.07%, in average. Although this solution was used in KNN algorithm, its usage is limitless as it can be used in any other contexts regarding categorical information.

5.2 Reflections and Future Work

This thesis may be seen as having different tools that compensate some of the KNN algorithm weaknesses. Thus, any of each one may be seen as autonomous if used in different contexts. Although the development of these tools made us able to achieve some important goals, final results are not the only thing to keep in mind. The mind set when developing such solutions creates space for potential researches.

Despite developed non-lazy-learning solutions involved a new learning phase that may take a while, but once it is completed, classification times really make it worth. Still, it feels there is room for improving those solutions by tuning the structures dividing criteria.

Taking into account the good results obtained in the categorical data conversion and the simplicity involved in this method, it seems promising to do some further research on this topic.

Bibliography

[AF03] João Ascenso and Ana Fred. Reconhecimento de padrões, 2003. [Online]. Available: http://ltodi.est.ips.pt/jascenso/padroes/teoricas/Aula%202%20-%20Introdu%C3%A7%C3%A3o%20ao%20RP.pdf. [Accessed: 14-Jan-2013].

[AKS] R. Agarwal, B. Kochar, and D. Srivastava. A novel and efficient KNN using modified apriori algorithm.

[Asc03] João Ascenso. Metodos não parametricos, 2003. [Online]. Available: http://ltodi.est.ips.pt/jascenso/padroes/teoricas/Aula%207%20-%20Aprendizagem%20N%C3%A3o%20Supervisionada.pdf. [Accessed: 14-Jan-2013].

[CAdC11] George Cavalcanti and Tiago Assunção de Carvalho. Distâncias para dados categóricos, 2011. [Online]. Available: http://www.cin.ufpe.br/~dlf2/2011.1/AM/6.distancias_dados_categoricos.pdf. [Accessed: 14-Jan-2013].

[dat] Data mining algorithms in R/Classification/kNN - wikibooks, open books for an open world. [Online]. Available: http://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Classification/kNN. [Accessed: 14-Jan-2013].

[GDZX12] J. Gou, L. Du, Y. Zhang, and T. Xiong. A new distance-weighted k-nearest neighbor classifier. 2012.

[HK00] E. H. Han and G. Karypis. Centroid-based document classification: Analysis and experimental results. *Principles of Data Mining and Knowledge Discovery*, page 116–123, 2000.

[IPM09] Dino Ienco, Ruggero G. Pensa, and Rosa Meo. Context-based distance learning for categorical data clustering. In *Proceedings of the 8th International Symposium on Intelligent Data Analysis: Advances in Intelligent Data Analysis VIII*, IDA '09, pages 83–94, Berlin, Heidelberg, 2009. Springer-Verlag.

[Jua11] L. Juan. TKNN: an improved KNN algorithm based on tree structure. In *Computational Intelligence and Security (CIS), 2011 Seventh International Conference on*, page 1390–1394, 2011.

[MdS00] Joaquim Marques de Sá. Reconhecimento de padrões, 2000. [Online]. Available: http://paginas.fe.up.pt/~jmsa/recpad/ [Accessed: 14-Jan-2013].

[PQS04] Jeng-Shyang Pan, Yu-Long Qiao, and Sheng-He Sun. A fast k nearest neighbors classification algorithm. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, (4):961–963, 2004.

[PSFA11] Adrian Perez-Suay, Francesc J. Ferri, and Jesús V. Albert. An online metric learning approach through margin maximization. In *Proceedings of the 5th Iberian conference on Pattern recognition and image analysis*, IbPRIA'11, page 500–507, Berlin, Heidelberg, 2011. Springer-Verlag.

[SA76] Dudani Sahibsingh A. The distance-weighted k-nearest neighbor rule. *Systems, Man and Cybernetics, IEEE Transactions on*, (325-327), 1976.

[SAH06] K. Shin, A. Abraham, and S. Han. Improving kNN text categorization by removing outliers from training set. *Computational Linguistics and Intelligent Text Processing*, page 563–566, 2006.

[WW07] Yu Wang and Zheng-ou Wang. A fast KNN algorithm for text categorization. *Proceedings of 6th International Conference on Machine Learning and Cybernetics - IEEE Press*, pages 19–22, 2007.

[YW05] Wang Yu and Zheng-ou Wang. Text categorization rule extraction based on fuzzy decision tree. volume 4, pages 2122–2127, GuangZhou, China, 2005.

[ZTD12] W. Zhao, S. Tang, and W. Dai. An improved kNN algorithm based on essential vector. *Electronics and Electrical Engineering*, 123(7):119–122, 2012.

[ZWZZ07] W. Zuo, K. Wang, H. Zhang, and D. Zhang. Kernel difference-weighted k-nearest neighbors classification. *Advanced Intelligent Computing Theories and Applications. With Aspects of Artificial Intelligence*, page 861–870, 2007.