AN EMPIRICAL STUDY OF THE SUITABILITY OF CLASS DECOMPOSITION FOR LINEAR CLASSIFIERS

A Dissertation

Presented to

the Faculty of the Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

By

Francisco Ocegueda-Hernandez

December 2012

AN EMPIRICAL STUDY OF THE SUITABILITY OF CLASS DECOMPOSITION FOR LINEAR CLASSIFIERS

Francisco Ocegueda-Hernandez

APPROVED:

Dr. Ricardo Vilalta, Chairman
Dept. of Computer Science

Dr. Stephen Huang
Dept. of Computer Science

Dr. Kam-Hoi Cheng
Dept. of Computer Science

Dr. Zhigang Deng
Dept. of Computer Science

Dr. Klaus Kaiser
Dept. of Mathematics

Dean, College of Natural Sciences and Mathematics

Acknowledgements

This dissertation would not have been possible without the support of many people throughout my years at University of Houston. I owe a great debt of gratitude to each of them.

My greatest thanks are for my parents, my mother who has always been the support for my development and my father who taught me the values of honesty, loyalty, and integrity that lead my life.

I am extremely grateful to Dr. Ricardo Vilalta, who as advisor had the difficult task of guiding my efforts to obtain a Ph.D. degree. I would like to highlight that many times he was more than an advisor. He always provided me with his support, inspiration, encouragement, and guidance, even in situations that were outside his academic responsibility. Most importantly, his spiritual approach to life inspired me immensely to be a better human being. I feel truly privileged to have had the opportunity to work with him.

I would like to express a special thanks to Dr. Stephen Huang, Dr. Kam-Hoi Cheng, Dr. Zhigang Deng, and Dr. Klaus Kaiser for agreeing to be members of my dissertation committee and for their time reviewing this dissertation.

Teamwork is an essential in research work. This dissertation was enriched by discussions and collaborations with other members of the Pattern Analysis Lab (PAL) of the Department of Computer Science. My gratitude goes to each PAL's member for its support. In particular, I give warm thanks to Roberto Valerio for his friendship.

Emotional stability is an important aspect to achieve the goal of obtaining a doctorate. Without the help of my girlfriend, family, and friends would not have enough courage to deal with the various difficulties that arose throughout the past five years. My thanks go to all them.

My very heartfelt thank you goes to my girlfriend, Rebeca Parra, for her unconditional love and for being so helpful in times of disappointment, personal pain, and hopelessness.

My special gratitude goes to my friends who has helped me along the way: Eduardo Saucedo, Roberto Coronado, Nayeli Coronado, Juan Urquiza, Federica Sanchez, Gerardo Mendizabal, Efren Ballesteros, and Barbara Vazquez.

Last, but definitely not the least, I would like to thank the Department of Computer Science of University of Houston for providing students with an extraordinary academic environment. In particular, I appreciate the efforts of those professors who contributed to many of my learning experiences. Also, I am grateful to Yvette Elder for being kind and helpful every time I needed help in processing any paperwork.

AN EMPIRICAL STUDY OF THE SUITABILITY OF CLASS DECOMPOSITION FOR LINEAR CLASSIFIERS

An Abstract of a Dissertation

Presented to

the Faculty of the Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

By

Francisco Ocegueda-Hernandez

December 2012

Abstract

The presence of sub-classes within a data sample suggests a class decomposition approach to classification, where each subclass is treated as a new class. Class decomposition can be effected using multiple linear classifiers in an attempt to outperform a single global linear classifier; the goal is to gain in model complexity while keeping error variance low. In this dissertation, we propose a study aimed at understanding the conditions behind the success or failure of class decomposition when combined with linear classifiers. We identify two relevant data properties as indicators of the suitability of class decomposition: 1) linear separability; and 2) class overlap. We use well-known data complexity measures to evaluate the presence of these properties in a data sample. Our methodology indicates when to avoid performing class decomposition based on such data properties. In addition we conduct a similar analysis at a more granular level for data samples marked as suitable for class decomposition. This extra analysis shows how to improve in efficiency during class decomposition. From an empirical standpoint, we test our technique on several real-world classification problems; results validate our methodology.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Motivation

There is a broad spectrum of successful applications where the learning algorithm employed for a classification task is limited to linear classifiers, e.g. document classification [4, 32], biomedical work [22], face recognition [5], chemistry [15], bioinformatics [23], etc. Linear classifiers have the advantage of keeping the variance component of error low, but may result in high bias (e.g., under non-linear class distributions). One approach that has been extensively studied in recent years is the use of a combination of linear classifiers to replace a single global linear classifier [30, 16, 31, 10, 11, 6, 33, 9, 29]. Such an approach increases model complexity, while keeping variance under control.

One instance of compound linear classifiers is that of class decomposition via clustering; here classes are separated into clusters as a pre-processing step to classification. As an illustration, Figure 1.1 depicts a two-dimensional input space where examples belong to two classes. The dotted line is the decision boundary built by a single global linear classifier. Now, assume a clustering algorithm separates each class into two clusters, whereby we relabel every example to encode class and cluster label; the resulting dataset has now four different classes. The dashed lines are the decision boundaries correctly separating the new four classes. The added flexibility gained by combining linear models does not come with a drastic increase in variance.

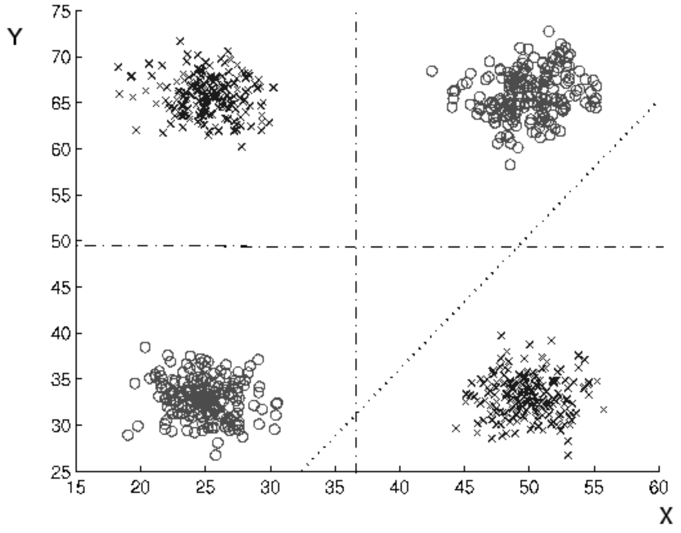
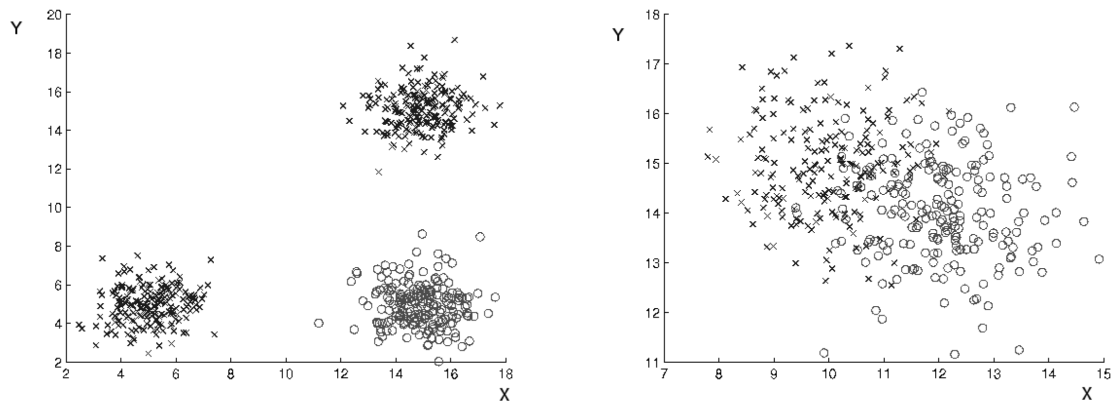


Figure 1.1: XOR dataset.

Several successful case studies employing class decomposition in conjunction with linear classifiers can be found in the literature [30, 31, 9, 29]. Despite promising results, there is a large number of classification problems where a combination of linear classifiers through class decomposition comes unwarranted. For example, Figure 1.2 depicts two distributions where a combination of linear classifiers does not bring an advantage over a single linear classifier. Both distributions are two-dimensional and have two classes. The data distribution shown in Figure 1.2(a) is linearly separable;

although one class (class "x") can be divided into two sub-classes, a single linear classifier suffices to separate both classes. Figure 1.2(b) shows the case where high Bayes error (i.e., high class overlap) obviates any type of class decomposition.



(a) (b)

Figure 1.2: Data distributions not suitable for class decomposition. (a) Data is linearly separable. (b) Classes overlap significantly, i.e., data exhibits high Bayes error.

Given the potential use of class decomposition in scenarios where linear classifiers have proved effective (e.g., text mining, bioinformatics applications, etc.), but where data abounds, it is important to avoid the extra computational cost incurred by the pre-processing (i.e., clustering) step, especially when the problem is not a good fit for class decomposition. Moreover, among problems that do indeed need the pre-processing step, it would be desirable to automatically select the classes that need such decomposition, instead of blindly applying the step to all classes.

Two important questions are as follows: (1) when is a distribution suitable for class decomposition? (i.e., when does the combination of linear classifiers produced by class decomposition outperform a single global linear classifier?); and (2) if a data distribution is suitable for class decomposition, which classes should be decomposed?

This dissertation attempts to answer the questions above by extracting data characteristics from samples by means of data complexity measures; we wish to understand the conditions for success or failure during class decomposition (when used in conjunction with linear classifiers). Specifically, we address the limitations of a combination of linear classifiers obtained by means of class decomposition that reveal two data properties: 1) linear separability between classes; and 2) high overlap between classes.

1.1 Contributions

In general, this dissertation contributes to increasing the understanding of the applicability of linear class decomposition in order to provide systematic user guidance on model selection. Specifically, this dissertation contains the three following contributions:

1. A WEKA-based framework that allows performing empirical model selection analysis for class decomposition in conjunction with any algorithm available in

WEKA. This framework will be publicly available to any end-user who wishes to access it.

2. The use of well-known data complexity measures to identify the presence of data properties in a data sample for the analysis of suitability in the use of class decomposition in conjunction with linear classifiers.

3. A set of practical rules to determine the suitability of class decomposition in a data sample.

1.2 Dissertation Outline

This dissertation is organized as follows. Chapter 2 provides the basic technical background and notation needed to understand the remainder of this dissertation. Chapter 3 presents a literature review on the use of combination of linear classifiers. Chapter 4 introduces our WEKA-based framework for class decomposition. Also, it reports on our experimental analysis and explains our methodology to determine the suitability of class decomposition. Finally, Chapter 5 contains our conclusions and future work.

Chapter 2

Preliminaries

2.1 Basic Notation in Classification

Statistical learning can be effected in various ways. We may be given a set of observations with the aim of establishing the existence of clusters in the data. Or we may know for certain that there are certain number of classes, and the aim is to establish a rule decision whereby we can classify a new observation into one of the existing classes. The reliability of the rule decision usually is measured by the proportion of correct classifications. The former type is known as Unsupervised Learning (or Clustering), the latter as Supervised Learning [19]. In this work when we use the term classification, we refer to Supervised Learning.

2.1.1 The Classifier Model

Let $(A_1, A_2, \cdots, A_n)$ be an n-component vector-valued random variable, where each A_i represents an attribute or feature; the space of all possible attribute vectors is called the input space $\mathcal{X}$. Let $\{y_1, y_2, \cdots, y_k\}$ be the possible classes, categories, or states of nature; the space of all possible classes is called the output space $\mathcal{Y}$. A classifier C receives as input a set of training examples $T = \{(\mathbf{x}, y)\}$, $|T| = N$, where $\mathbf{x} = (a_1, a_2, \cdots, a_n)$ is a vector or point of the input space and y is a point of the output space. We assume T consists of independently and identically distributed (i.i.d.) examples obtained according to a fixed but unknown joint probability distribution in the input-output space $\mathcal{X} \times \mathcal{Y}$ $(P(x, y))$. The outcome of the classifier C is a function h (or hypothesis) mapping the input space to the output space, $h : \mathcal{X} \to \mathcal{Y}$. Let H_C (hypothesis class) be the set of possible output function of C and let F denote the set of all functions from the input space $\mathcal{X}$ into the output space $\mathcal{Y}$. Due to the inherent bias learning of any classifier, we note that $H_C \subset F$.

2.1.2 The Loss Function and Risk

In the search for an appropriate function h from the hypothesis class H_C, we need to evaluate the reliability of the output of classifier C. We perform this evaluation by using a loss function $L : \mathcal{Y} \times \mathcal{Y} \to \mathcal{R}$; that identifies mistakes made by h, that is, cases where the output of h (predicted value) and the actual value are different. For example in classification, we employ the following 0-1 loss function:

$$L_{0-1}(h(x), y) = \begin{cases} 0 & \text{if } h(x) = y \\ \\ 1 & \text{if } h(x) \neq y \end{cases} \tag{2.1.1}$$

Where $h(x)$ is the predicted value and y is the actual value. In order to evaluate how well hypothesis h would predict in average over any sample drawn from $P(x, y)$, we define a measure known as expected loss or risk associated with h as follow:

$$R(h) = \mathrm{E}[L(h(x), y)] = \int L(h(x), y) dP(x, y) \tag{2.1.2}$$

2.1.3 Empirical Risk Minimization

In classification problems the learning goal is described by a loss function L and its associated risk R. This learning goal is to find a hypothesis t^* (target hypothesis) from F that has an minimal risk R:

$$t^* = \arg\min_{t \in F} R(t) \tag{2.1.3}$$

Given that a classifier C is restricted to use $H_C \in F$, we actually look for:

$$h^* = \arg\min_{h \in H_C} R(h) \tag{2.1.4}$$

Due to the fact that $P(x, y)$ is unknown, the risk $R(h)$ cannot be computed. Therefore, we have to use an approximation to R in order to allow the learning goal to be accomplished. We can approximate R by its empirical counterpart (empirical risk) based on the training set T:

$$R_{emp}(h) = \frac{1}{N}\sum_{i=1}^{N} L(h(x_i), y_i)) \tag{2.1.5}$$

Now, we replace the unknown true risk R by R_{emp} reformuling our learning goal to:

$$h^* = \arg\min_{h \in H_C} R_{emp}(h) \tag{2.1.6}$$

This reformulation for the learning goal is known as the empirical risk minimization (ERM) method.

2.2 Linear Classifiers

Linear classifiers are probably one of the most popular simple algorithms for classification. Numerous works have shown its usefulness in a broad spectrum of applications (e.g. document classification [4, 32], biomedical [22], face recognition [5], chemistry [15], bioinformatics [23], etc). A key aspect of this success is its simplicity due to the utilization of linear functions to distinguish classes by building decision surfaces that are linear functions of the input vector x. Consequently, they produce simple and interpretable models. Moreover, their training have shown to be very efficient in large-scale data in terms of time [4] and memory space [32].

We define a linear classifier C_l as the learning algorithm that has the following family of functions as hypothesis space H_{C_l} [8, 2]:

$$h(x) = w^T x + w_0 \tag{2.2.1}$$

For classification purpose, the following indicator function is used over $h(x)$ output:

$$I(h(x)) = \begin{cases} 1 & \text{if } h(x) \geq 0 \\ -1 & \text{if } h(x) < 0 \end{cases} \tag{2.2.2}$$

In supervised classification, there exists many algorithms for building linear classifiers, the following three algorithms are among the most popular:

- Perceptron. These algorithms are well-known by their ample use in the context of neural networks. They are the basic component (neuron) on these nets which is composed of a set of entries (x_i) with an associated weight (w_i). The entries and their corresponding weights are linearly combined to obtain a value. This value is used as input for a threshold function to generate the predicted class for an example. If the true class is different than the predicted class the weights are adjusted until the thresholded linear combination of the inputs matches the true class.

- Logistic Regression (LR). From a statistical perspective, these methods model the conditional probabilities of the class given the data and use these probabilities to classify examples. The idea is to use linear regression for predicting the outcome of a categorical variable. In order to perform such an analysis they utilize a logistic function to transform the output to a probability score.

In this type of method, the linear model coefficients (w and w_0) are usually estimated using maximum likelihood estimation.

- Support Vector Machines (SVMs). This classification algorithm has its origin in research conducted within the scope of statistical learning theory [27]. The idea behind this learning mechanism is to transform the input space to a new space where the classification problem is linearly separable. This procedure is possible by using non-linear transformations. Although non-linear transformations can be computationally expensive, SVMs are feasible by using the so-called kernel trick (i.e., the utilization of a kernel function in a reproducing kernel Hilbert space to avoid performing explicitly the required data transformation [26]). SVMs search for a special type of linear model in the transformed space, namely, the maximum-margin hyperplane. This hyperplane is defined as the hyperplane that has the largest separation (margin) between the classes. The distance from this hyperplane to the nearest data point on each class, which are known as support vectors, is maximum. The importance of these support vectors is that they are sufficient to define the maximum-margin hyperplane which reduces the amount of computations to perform. The graph in Figure 2.1 displays an example of a maximum-margin hyperplane. In this graph there are two sets of points that belong to either dark or white class. The line that separates these classes is the maximum-margin hyperplane.The support vectors are enclosed in circles.

A more detailed explanation of these algorithms can be found in [8, 2].

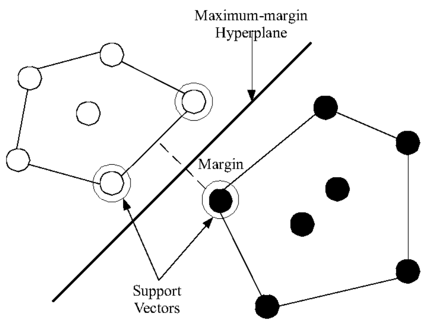


Figure 2.1: Maximum-margin hyperplane.

2.2.1 Linear Support Vector Machines

Linear Support Vector Machines are a special case of SVMs where the employed kernel is a linear kernel. The solution to the classification problem (Equation 2.2.1) is reformulated as:

$$h(x) = \sum_{i=1}^{N} \alpha_i K(\mathbf{x}_i, \mathbf{x}) \tag{2.2.3}$$

where $\{\alpha_i\}$ is a set of real parameters, index i runs along the number of training examples, and $K(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1 \cdot \mathbf{x}_2)$ is a linear kernel function in a reproducing kernel Hilbert space [26].

2.3 Clustering

Clustering is a form of unsupervised classification where the goal is to partition a dataset into data groups based on similarity. This similarity is usually defined as a measure of proximity in a multidimensional space. A clustering algorithm works by finding data groups whose inter-elements similarity values are large compared with the similarity values to elements outside of the data groups. The obtained data groups represent a classification that naturally underlie in the dataset [8, 2]. There exists a plethora of clustering algorithms which can be grouped as follows:

- Partition-based. These algorithms build an user-defined number k of clusters where each cluster has at least one element and each element belongs exclusively to a single group. These methods initially select or compute k-representative elements, then they create k-clusters by assigning each element to its nearest cluster. This assignation is based on a distance measure between the elements and the k-representatives. Once all the elements were assigned to a cluster, a re-estimation of the representative element is performed for each cluster. These last two steps are iterated until a stopping criterion is met. The most popular algorithms in this group are k-means, k-medians and k-medoids.

- Hierarchical-based. These type of clustering methods create a hierarchy of clusters based on a tree data decomposition. Each level of the tree is itself a clustering. Hence, each element belongs to as many clusters as levels have the tree, but each element belongs to a single cluster per level. The tree can be build in a either top-down or bottom-up fashion. Top-down method begins with all points in a single cluster (root node) and repeatedly split the clusters until all the leaf nodes have only a singleton cluster or another stopping criterion is met.

- Density-based: This clustering utilizes a density measure instead of distance measure where clusters are defined as areas of higher density. The algorithm works by grouping those elements inside a pre-determined neighbourhood area having a density value above certain density threshold. DBSCAN and OPTICS are examples of this kind of clustering.

- Model-based. Clustering algorithms in this group assumes that there is a data distribution model behind the dataset. Therefore, this clustering technique focus on finding the most likely model from a distribution family (e.g. Gaussian distributions) to explain the data. Given a fixed number of distributions (one per cluster) to model the dataset, these algorithms look for the distribution parameters that fit better to the dataset. This is done by an iterative optimization process that runs until convergence or a stopping criterion is met. Elements are assigned to the model distribution they most likely belong to. A popular density-based clustering algorithm is Expectation-Maximization(EM).

2.3.1 K-means

k-means is a partition-clustering algorithm which owes its increasing popularity to both its simplicity and interpretability. This method is easy to understand and to implement it. And it is interpretable because this algorithm produces a clustering with k disjoint clusters which are convex (i.e. clusters which are very roughly spherical or elliptical), non-empty and non-overlapped. Moreover, k-means may be computationally faster than other more complex clustering algorithms. In fact, some clustering algorithm implementations uses k-means for initialization purpose (e.g.

EM WEKA implementation) or as a part of them (e.g. Spectral clustering).

k-means works by comprising a group of data points whose inter-point distances are small compared with the distances to points outside of the cluster. This group of data points is obtained as follows: i) A set of k-prototypes $c_i \in \mathcal{R}^n$, where $i = 1, ..., k$, is randomly selected, and in which c_i is a prototype associated with the i-th cluster; ii) data points are assigned to their nearest clusters based on the distances of each data point to its closest vector c_i; and iii) a re-estimation of the k-prototypes c_i is performed based on the minimization of the sum of the squares of the distances of each data point to its assigned cluster in step (ii). This minimization procedure obtains the centres of the clusters (i.e. the mean vector μ_i for the data points that belongs to cluster i) as the new k-prototypes c_i. Algorithm 1 displays a typical pseudocode for k-means algorithm.

Some k-means drawbacks are: i) its need for a pre-determined number of clusters which can make it difficult to know the most appropriate k value to use; ii) its dependency on the initialization (i.e. different initial partitions or values can generate different clusters); and iii) its lack of ability to deal with non-convex clusters.

Algorithm 1 k-means algorithm Pseudocode

```
Input: Dataset $T = \{x_1, x_2, ..., x_m\}$ where $x_j \in \mathcal{R}^n$ and $j = 1, ..., m$, Number of cluster $k \geq 1$
Output: Clusters $C_1, C_2, ..., C_k$
 1: Initialization
 2: Select $k$ data points $(c_1, c_2, ..., c_k)$ from $T$ randomly
 3: Let $C_i \Leftarrow \varnothing$ where $i = 1, ..., k$
 4: for $j = 1$ to $m$ do
 5:     $I = \arg\min_{i \in \{1..k\}} distance(x_j, c_i)$
 6:     $C_I \Leftarrow C_I \cup \{x_j\}$
 7: end for
 8: $change = true$
 9: Main Loop
10: while $change$ do
11:     $change = false$
12:     New $k$-prototypes estimation
13:     for $i = 1$ to $k$ do
14:         $c_i = ComputeMeanVector(C_i)$
15:     end for
16:     Assignation to clusters based on new $k$-prototypes
17:     for $j = 1$ to $m$ do
18:         $I = \arg\min_{i \in \{1..k\}} distance(x_j, c_i)$
19:         if $x_j \in C_i$ and $i \neq I$ then
20:             $change = true$
21:             Move $x_j$ from $C_i$ to $C_I$
22:         end if
23:     end for
24: end while
25: return $C_1, C_2, ..., C_k$
```

2.4 Class Decomposition Algorithm

Class decomposition via clustering is a pre-processing step that has been successfully used to enhance the performance of linear models [29, 9]. It works by partitioning each class into clusters, and by relabelling examples comprised by each cluster with a new class. This decomposition attempts to discover the intrinsic local distribution of subclasses for each class, which can be seen as an indicator of problem complexity. Knowledge about problem complexity can then lead to selecting an appropriate classifier, and has proved to be particularly useful for linear models. In particular, class decomposition allows increasing the capacity of linear classifiers through a piecewise model-building approach. Let's revisit Figure 1.1, where we show a two-dimensional input space with two classes, and where each class has two subclasses; the subclass distribution follows the XOR concept. Clearly, a simple linear classifier is inappropriate here. In contrast, the combination of linear models, where each model separates a subclass from the rest, allows the construction of a more flexible (compound) decision boundary. Class decomposition enable us to cope with distributions where classes spread over disparate regions of the input space.

The mechanism for class decomposition used in this study comprises the following steps: 1) separate the training data T into sets of examples of the same class. That is, T is separated into subsets $T = \{T_j\}$, where each T_j comprises all examples in T labelled with class y_j , $T_j = \{(x, y) \in T \mid y = y_j\}$. 2) for each subset T_j, a clustering algorithm is applied to find clusters of examples grouped together according to some distance metric over the input space. Let c_i^j be the set of such clusters. We map

Algorithm 2 Algorithm for Class Decomposition using Linear Classifiers

```
Input: Training dataset $T = \{(\mathbf{x}, y)\}$, $M$ number of classes, Clustering algorithm $CL$, Linear Classifier $LC$, Integer $k \geq 2$
Output: $\mathcal{CDLC}$: Classifier built by applying class decomposition using linear classifiers to $T$
1: Class Decomposition
2: Let $T' \Leftarrow \varnothing$
3: for $j = 1$ to $M$ do
4:     Let $T_j = \{(x, y) \in T \mid y = y_j\}$
5:     Let $c_1^j, c_2^j, ..., c_k^j$ be the partition of $T_j$ into $k$ clusters by using $CL$.
6:     Let $T_j' \Leftarrow \varnothing$
7:     for $i = 1$ to $k$ do
8:         for each $(x, y_j) \in c_i^j$ do
9:             Rename $y_j$ to $(j, i)$
10:            Let $T_j' \Leftarrow T_j' \cup c_i^j$
11:        end for
12:    end for
13:    Let $T' \Leftarrow T' \cup T_j'$
14: end for
15: Training
16: Train $LC$ using $T'$ and assign the built classifier to $\mathcal{CDLC}$
```

each T_j into a new set T_j' by renaming every class label to indicate not only the class, but also the cluster for each example. One simple way to do this is by making each class label a pair (a, b), where the first element represents the original class, and the second element represents the cluster. In that case, $T_j' = (x, y_j')$, where $y_j' = (y_j, i)$ whenever example x is assigned to cluster c_j^i. Finally each new subset T_j' is simply the union of all sets of examples of the same class relabelled according to the cluster to which each example belongs, $T' = \bigcup_{j=1}^{k} T_j'$. See Algorithm 2 for a pseudocode description of class decomposition algorithm.

2.5 Data Complexity Measures

Complexity analysis applied to classification has been extensively studied from a theoretical perspective. Recently, an empirical approach has generated considerable research interest [14, 1, 24, 13, 17, 20]. Much of this research has employed a set of measures to characterize data complexity during classification by means of geometrical and topological properties. The idea is to establish a connection between data characterization and classifier performance. In [14], three data complexity measures are introduced: (1) class overlap measured according to feature discriminatory power; (2) class separability measured according to the length and linearity of the class boundary; and (3) geometry, topology and density of classes, which assume the problem is composed of several manifolds spanned by each class; the shape, position, and interconnectedness of these manifolds provide hints on how well the classes are separated, and on the density or population of each manifold. The complexity of a task is characterized by these measures. Such characterization provides a quantitative perspective to study the learnability of class boundaries, and is of great relevance for operational guidance during model selection. When datasets are characterized by these measures, similar properties are expected to correlate with problems of similar complexity.

In this paper, we employ data complexity measures to analyze the conditions under which class decomposition via clustering improves predictive performance (using linear classifiers). We briefly describe the set of complexity measures employed for our experiments.

Training error of a linear classifier (L2). Linear class separability can be estimated by computing the training error of a linear classifier. Let $g(x)$ be a linear classifier, we define $L2$ as:

$$\mathrm{L2(T)} = \frac{1}{\mathrm{N}} \sum_{\mathrm{i=1}}^{\mathrm{N}} \mathrm{I}_{\{\mathrm{g(x_i)} \neq \mathrm{y_i}\}} \tag{2.5.1}$$

where $I_{\{\cdot\}}$ is an indicator function. A value of zero in L2 indicates the problem is perfectly linearly separable.

Ratio of average intra/inter class nearest neighbor distance (N2). We compare the within-class example separation with the example separation across classes. For each input instance (x_i, y_i), we compute the following:

$$\mathrm{D_I(x_i, y_i)} = \min_{\mathrm{(x,y) \in T | y = y_i}} dist(x_i, x) \tag{2.5.2}$$

$$\mathrm{D_O(x_i, y_i)} = \min_{\mathrm{(x,y) \in T | y \neq y_i}} dist(x_i, x) \tag{2.5.3}$$

$$N2(T) = \frac{\sum_{i=1}^{N} \mathrm{D_I(x_i, y_i)}}{\sum_{i=1}^{N} \mathrm{D_O(x_i, y_i)}} \tag{2.5.4}$$

where $dist()$ is a distance function. Low values for $N2$ suggest examples of the same class lie closely in the feature space. High values indicate high dispersion among examples of the same class.

Leave-one-out error rate of the one-nearest neighbor classifier (N3). We capture the relative closeness of examples from different classes, by computing the leave-one-out error rate of the one-nearest neighbor classifier (the kNN classifier with $k = 1$, or 1NN). Low values point to a large margin.

Chapter 3

Literature Review

Recently, there has been growing interest in the utilization of different forms of class decomposition to employ combination of local linear classifiers for classification. In this chapter, we review state-of-the-art methods inside this approach.

In class decomposition approach, the goal of data decomposition is to find subclasses for the use of a composite classifier. The problem to be addressed is that of determining the optimal number of subclasses and how these subclasses are obtained. Cheng et al. [7] proposed a Profile Support Vector Machine (PSVM) which is a localized approach to Support Vector Machine. The idea is to decompose the training set into clusters by employing a supervised clustering algorithm which produces balanced class clusters. For each cluster, a local linear SVM model is built. During testing phase, an example is classified by finding the nearest cluster and invoking its corresponding local linear SVM model. This work demonstrated its effectiveness on both temporal and spatial data. A similar method to [7] was proposed in Segata et al. [25]. This method is based on a different decomposition of the training set, namely, k-nearest neighborhoods. A k-nearest neighborhood, which enclosed the k-nearest neighbors for a given point, is constructed for each element of the training set. In order to decrease the number of neighborhoods, a set C of k-nearest neighborhoods covering the whole training set is obtained by a post-processing step. A local SVM model is built for each element in C. In this work, the local models present a level of redundancy (i.e., some models share training points) which is tuned by the parameter k. For a test point, the model centered on the training point which is the nearest in terms of the neighborhood is used. This work differs from [7] in two aspects: i) the neighborhood construction, it does not take into account the class label, consequently, some of the elements of C could contain only points belonging to one class, and then its corresponding local model is just the majority rule avoiding the training of a SVM; and ii) a local model selection mechanism is introduced which allows building of local models with higher complexity of linear models. Locally linear classification by pairwise coupling (LLC-PC) is another method in this area. This compound method works by decomposing complex classes into linearly separable subclasses, learning a linear classifier for each pair, and combining these pairwise classifiers into a single classifier. A study of three different combination schemas for LLC-PC is introduced in Chen et al. [6]. It also proposed several global criterion functions for measuring the goodness of subclasses, and presents a supervised greedy clustering algorithm to optimize the proposed criterion functions. This work emphasizes the importance of having an appropriate criterion function for clustering when it will be used for LLC-PC while works in [7] and [25] do not optimize an explicit criterion function for partitioning data. [29], [31], [30], and [9] are similar in using the concept of class decomposition via clustering as a pre-processing. This process works by decomposing the classes into sub-classes by clustering as a pre-processing step. The original dataset is relabeled to consider the new classes, and then a linear classifier is trained using the relabeled dataset. This increment in the number of classes allows the use of a combination of linear classifiers at more granular level. Vilalta et al. [29] proposed the use of class decomposition via clustering to improve the performance of low-variance classifiers (e.g., linear classifiers). Fradkin [9] presented a empirical study on this technique to evaluate the impact of the number of cluster per class and analyze effects of the training set size on the results of the class decomposition via clustering. Wu and Chen [31] utilized this method for dealing with class imbalance problem. They performed the class decomposition via clustering within each large class to produce sub-classes with relatively balanced sizes. They also provided a systematic analysis of time and space complexity of this approach. A different decomposition perspective is presented by Chang et al. [3]. This method uses a decision tree to decompose the dataset. They showed that this decision tree decomposition based is valuable because it can classify some data points by its own means and it is efficient in determining the parameter values that maximize the validation accuracy. This work derived generalization error bound for the compound classifier which is a missing component in all the aforementioned works. The works presented in [28] and [33] utilize an ensemble perspective to the class decomposition approach. Verma and Rahman [28] presented a novel cluster-oriented ensemble classifier. The proposed ensemble classifier is a two layers algorithm. The first layer contains a set of base classifiers to learn the cluster boundaries, while the second layer is a fusion classifier used to classify based on the cluster confidences (first layer output). A key aspect of this work is its use of multi-clustering. Zhou et al. [33] presented a different ensemble technique for decomposing the classes, namely, Data-driven Error Correcting Output Coding (DECOC). The idea is based on using a code matrix to decompose a multi-class problem into multiple binary problems.

Chapter 4

An Empirical Study of the Suitability of Class Decomposition for Linear Models

4.1 Related Work

Several studies have demonstrated that using linear classifiers and class decomposition via clustering together, can improve predictive accuracy when compared to the use of a single global linear classifier. In [29], an empirical study of class decomposition shows a couple of clear successful cases when using linear classifiers. A similar study reported by [9] elaborates deeper on the effect of the number of clusters employed during class decomposition, but results are not conclusive to determine if increasing the number of clusters is beneficial. More recently, [30, 31] successfully applies class decomposition to deal with the class imbalance problem; however, the analysis lacks insight about data properties that favor the decomposition of a single majority class to eliminate class imbalance.

A common conclusion among these studies is that class decomposition hardly improves performance when used in conjunction with complex classifiers (e.g. decision trees, support vector machines with radial basis functions or high order polynomials as kernels, etc.). Class decomposition is useful to enhance low variance classifiers exclusively (e.g., naive Bayes, linear classifiers).

Previous work fails to elucidate data properties that favor the utilization of class decomposition with linear classifiers. Moreover, there are no guidelines pointing to the classes that seem more favorable for decomposition; in almost all cases, class decomposition is applied indiscriminately over all classes. In this study we address these challenges by exploiting information that lies in the data itself, using a set of well-known data complexity measures.

4.2 Empirical Study

We begin our study by first describing an empirical analysis of the use of class decomposition in conjunction with linear classifiers using several real-world datasets. We postpone analysis of the suitability of class decomposition to Section 4.2.5. We describe a number of experiments to compare the classification performance of several classifiers, including a single global linear classifier, a classifier with high capacity, and several composite classifiers obtained through different settings during class decomposition. We also compute data complexity measures (i.e., $L2$, $N2$, and $N3$) to correlate data properties with classification performance.

4.2.1 Experimental Datasets

In the experiments, we use several datasets from the UCI Machine Learning Database Repository[1]. Among them two data sets, Heart-c and Credit-g, are binary classification datasets. The Heart-c (the c stands for the Cleveland database) dataset contains results from real-world heart disease diagnosis where the goal is to distinguish presence from absence of heart disease in a patient. The Credit-g (the g stands for the German credit database) dataset is about the information of whether the customer shows to be a good or bad according to information related to its credit history. The rest five datasets are multi-classification datasets from different real-world problems which were tackled by the pattern recognition community. PenDigit, SatImage, Image and Vehicle datasets are from computer vision application domain. The aim for these datasets is to detect objects or segments in an image using image processing features (e.g., the SatImage dataset contains the multi-spectral values of pixels in 3×3 neighborhoods in a satellite image). The Vowel dataset is from voice recognition area. This dataset was designed for the task of speaker independent recognition of the eleven steady state vowels of British English.

[1]UCI Repository is available at http://www.ics.uci.edu/mlearn

Table 4.1 displays the datasets and their characteristics. As can be seen in this table, Vehicle, SatImage, PenDigit, and Image datasets contain only numeric attributes; while Vowel, Heart-c and Credit-g datasets use a mixed of nominal and numeric attributes. The datasets show in Table 4.1 stand as representative examples of success and failure in the use of class decomposition for linear classifiers based on results reported in [29, 9, 31, 30]. Vowel, Vehicle, Pendigit, and SatImage datasets are reported as successful cases. On the contrary, Image, Heart-c and Credit-g datasets are reported as failed cases.

Table 4.1: Datasets and characteristics

Id.	Dataset	Examples	Num. Attributes	Type of Attributes	Classes	Success	Reported in
1	Vowel	990	13	Mixed	11	Yes	[29, 9, 31]
2	Vehicle	846	18	Numeric	4	Yes	[29, 9]
3	PenDigit	10992	16	Numeric	10	Yes	[9, 31]
4	SatImage	6435	36	Numeric	6	Yes	[9, 31]
5	Image	2310	19	Numeric	7	No	[9]
6	Heart-c	303	13	Mixed	2	No	[29]
7	Credit-g	1000	20	Mixed	2	No	[29]

4.2.2 Experimental Setup

4.2.2.1 Class Decomposition Implementation in Weka

Despite multiple previous work in class decomposition, there is no a publicly available implementation of this general algorithm. Providing a public and common implementation may help researchers to create their own use of class decomposition method and compare their results. In order to fulfill this need, we have developed our own implementation of class decomposition algorithm inside the well-known WEKA machine-learning class library [12]. WEKA is a complete experimental framework amply used by the machine-learning community, therefore the inclusion of the class decomposition algorithm inside WEKA provides both researchers and practioners with a working implementation, which they can use in their research. Moreover, this WEKA-based implementation allows using all the features and funcionalities of WEKA as data pre-processing and visualization tools, a broad collection of algorithms for feature selection, classification and clustering and several enviroments for experimentation. We have made our modified WEKA tool (WEKACD) available at http://www2.cs.uh.edu/˜ocegueda/tools/clsdcomp.htm/.

Figure 4.1(a) shows the location of our class decomposition implementation (ClassificationViaClassDecompClustering) inside WEKA Explorer tool (i.e., Classify tab - meta folder). In contrast, Figure 4.1(b) displays the graphical user interface (GUI) of this implementation. In order to use ClassificationViaClassDecompClustering, we describe the required options to set as follows:

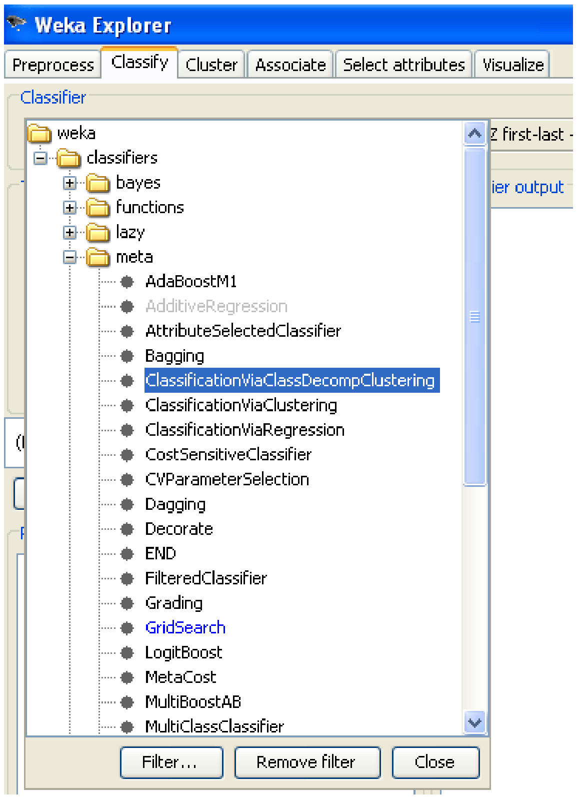
- classIndices. Range of classes to be used for class decomposition. This is an useful feature for running class decomposition only for a subset of classes (e.g., 2,5,7). The default range value is: first-last (This range value implies that class decomposition will be applied to each class).

- classifier: Base classifier to be used for classification after decomposing the classes. Although our study is limited to linear classifier, our implementation allows using any available classifier in WEKA. The default classifier is a support vector machine implementation (SMO).

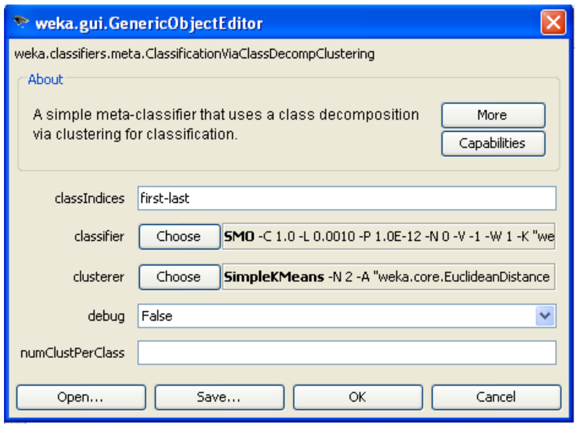
- clusterer: Clustering algorithm to be used for decomposing the classes. The default clustering algorithm is *k*-means.

- numClustPerClass: Comma-separated list of number of cluster per class (e.g., 3,2,6). The number of elements in this list must match the number of selected classes (classIndices option). The elements list will be used according to the order provided in the classIndices option. For example, if the selected classes were (2,5,7) and the list was (3,2,6) then class with index 2 is decomposed using 3 clusters, class with index 5 is decomposed using 2 clusters and class with index 7 is decomposed using 6 clusters.

The rest of the funcionalities for Figure 4.1(b) work as any classifier inside WEKA Explorer-Classify framework (see for details [12]). Moreover, ClassificationViaClass-DecompClustering is available for all the frameworks part of WEKA.



(a)



weka.classifiers.meta.ClassificationViaClassDecompClustering

A simple meta-classifier that uses a class decomposition via clustering for classification.

classIndices	first-last
classifier	**SMO** -C 1.0 -L 0.0010 -P 1.0E-12 -N 0 -V -1 -W 1 -K "we
clusterer	**SimpleKMeans** -N 2 -A "weka.core.EuclideanDistance
debug	False
numClustPerClass	

(b)

Figure 4.1: Class Decomposition Algorithm in WEKA: (a) Location in WEKA Explorer Classify Tab; (b) GUI - Class Decomposition Algorithm;

4.2.2.2 The Clustering Algorithm Selection

Three different clustering algorithms have been employed for class decomposition among the works presented in Section 4.1, namely, Expectation-Maximization (EM) [18], k-means [8] and x-means[21]. Our aim in selecting a clustering algorithm for class decomposition is to choose an algorithm that compromises between simplicity and performance to facilitate the analysis of our results. k-means algorithm is the simplest one among the aforementioned clustering algorithms. In order to verify that this simplicity does not incur in a high penalty in terms of classification performance for class decompostion, we run our class decomposition algorithm over the datasets using the three clustering algorithms implemented in WEKA: i)EM; ii)SimpleKMeans; and iii) XMeans. We fix both the number of cluster to 2 and the classifier to be a linear classifier. On each dataset, we report the average of a 10-fold cross validation.

Table 4.2 displays the results of a classification performance comparison in using the three different clustering algorithms for class decomposition. The first column describes the dataset used for our experiments. The second column reports on the accuracy of the class decomposition using k-means (CD-k). The third column shows accuracy for the use of EM (CD-EM). The fourth column shows the accuracy of the utilization of x-means (CD-x). Numbers enclosed in parentheses represent standard deviations. A significant difference with respect to the linear classifier (SMO) is shown in bold. Our tests of significance assume a t-student distribution with a 95% confidence level. NA stands for not applicable.

Table 4.2: Classification performance comparison for class decomposition using three different clustering algorithms.

Dataset	**CD-k**	**CD-EM**	**CD-x**
Vowel	73.03(6.53)	76.57(4.41)	NA
Vehicle	72.47(4.14)	72.58(3.76)	71.99(3.71)
PenDigit	99.1(0.23)	99(0.34)	99.15(0.34)
SatImage	86.31(0.92)	85.66(1.13)	**87.8(0.54)**
Image	91.73(1.78)	91.73(1.42)	91.43(1.67)
Heart-c	80.18(5.84)	83.48(6.58)	NA
Credit-g	73(3.68)	74.6(3.53)	NA

As can be seen from Table 4.2, there is only one dataset with significant difference between k-means and x-means, namely, SatImage. Because k-means shows a good compromise between simplicity and performance, we employ k-means as the clustering algorithm for class decomposition in all our experiments. Moreover, k-means algorithm deals with both types of attributes (i.e., numeric and mixed), whereas x-means algorithm only works with numeric datasets.

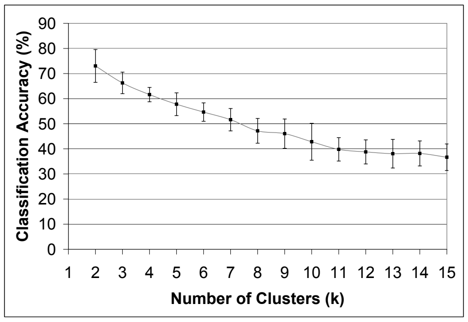
The success in the use of class decomposition depends on an appropriate selection of the number of cluster k for decomposing classes. Unfortunately, there is no method for setting an optimal value a priori, therefore, an empirical evaluation is required in order to choose an appropriate value for k. This evaluation is limited to a set of values where the value with the highest classification performance is selected. For our empirical study, we are interested in analyzing classification results in function of k within a range of values. This range of values is where the classification performance changes from an improvement to a loss or vice versa. The idea is to capture the classification performance behavior along as we move from a lower number of clusters to a higher number of clusters. Accordingly, an appropriate set of values for k for our study should contain values for both success and failure conditions.

In order to determine such a set of values for k in our experiments, we run our class decomposition algorithm for each dataset setting the k parameter to values from 2 to 15. Figure 4.2 and Figure 4.3 depict the classification accuracies obtained by the 14 class decomposition evaluations on each of the 7 datasets. As can be seen from these figures, an appropriate subset of values is the set $\{2, 3, 4, 6, 10\}$ which contains in all cases the highest classification accuracy and keeps the graph trends showing cases for deterioration in performance. For example, in Vowel dataset (see Figure 4.2(a)) the highest accuracy is achieved by two clusters, and the graph trend shows that when the number of cluster increases, the classification accuracies decreases along with this increment in the number of clusters. On the other hand, in SatImage dataset the classification accuracies improve along with increments in the number of clusters up to reach the highest classification accuracy in $k = 10$ and then the graph trend remains steady along the rest of value.

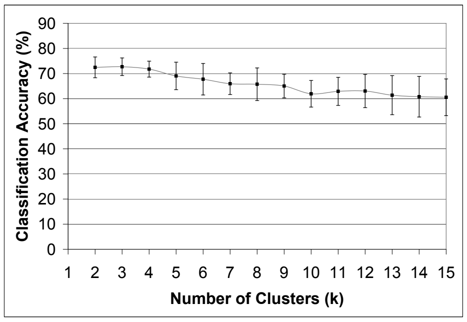
In our experiments, we limit the value of k to the set $\{2, 3, 4, 6, 10\}$ which provides useful information to characterize the behavior of class decomposition in the datasets. Morevover, this set avoids the extra computational cost incurred by the use of an unnecessary larger set.

4.2.2.3 The Linear Classifier Algorithm Selection

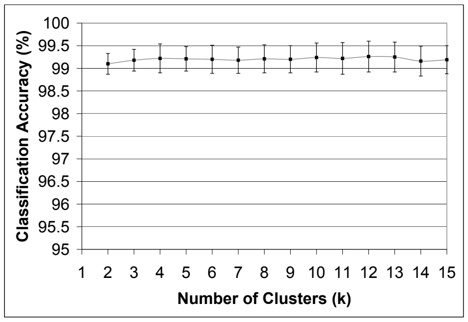
An important consideration in selecting an appropriate linear classifier for our study is its stability over the selected datasets. That is, it should be well-behaved in terms of classification performance avoiding to be the worst classification performance for any dataset. Among the different linear classifier options, the research work in class decomposition has focused on three types: i) SVM-based implementations that use an SVM classifier with a linear kernel as its linear classifier; ii) Logistic-Regression-based methods which utilize a ridge logistic regression model for the linear classifier; and iii) Bayesian-based approach that uses a Bayesian logistic regression model as its linear classifier. In order to select a linear classifier for our experiments, we run our class decomposition algorithm on each dataset using a 10-fold cross validation. We utilize three implementations present in weka of linear classifiers (one for each type): i) SMO; ii) SimpleLogistic (LR); and iii) Bayesian Logistic Regression (BLR). We employ the set of values $\{2, 3, 4, 6, 10\}$ for k and use the default settings for the classifiers. We also include the classification performance of the base classifier without class decomposition for analysis purposes by means of $k = 1$. In order to make feasible the utilization of LR algorithm for PenDigit and SatImage, we employ a stratified sample of 10%.



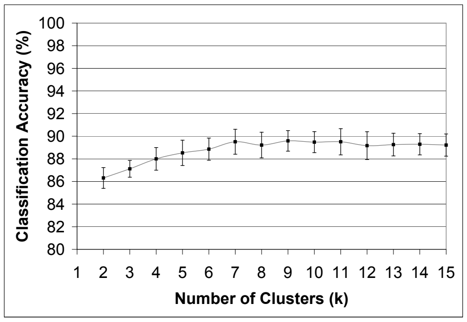
(a)



(b)

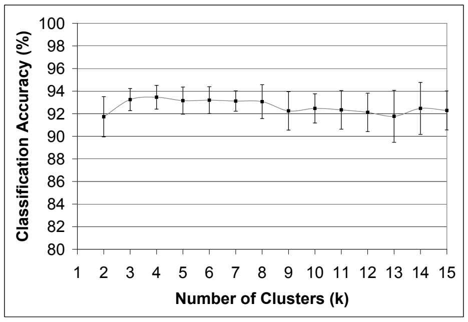


(c)

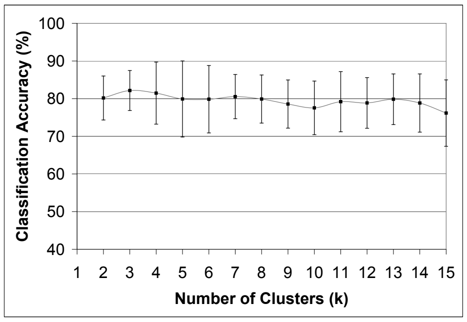


(d)

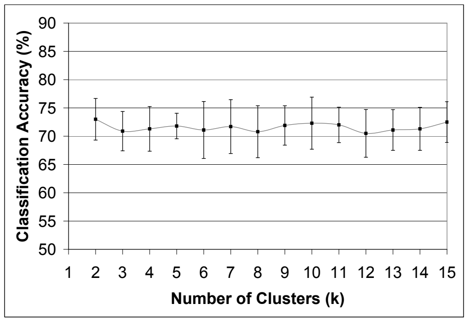
Figure 4.2: Classification performance for class decomposition using k-means (k takes values from 2 to 15) : (a) Vowel dataset; (b) Vehicle dataset; (c) PenDigit dataset; and, (d) SatImage dataset.



(a)



(b)



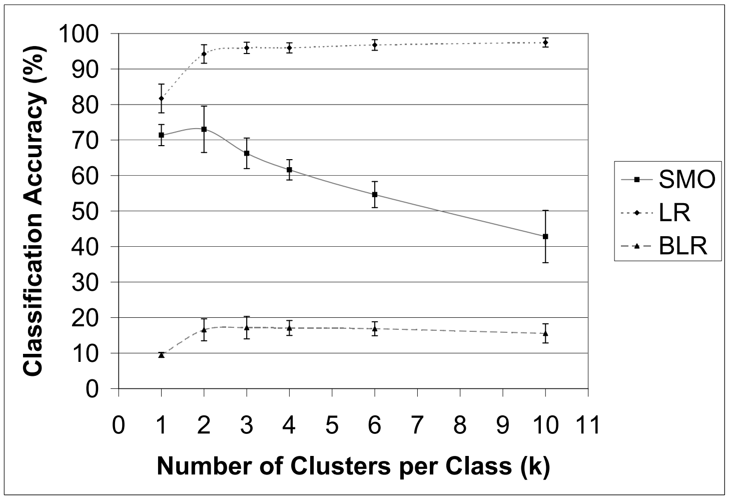
(c)

Figure 4.3: Classification performance for class decomposition using k-means (k takes values from 2 to 15) : (a) Image dataset; (b) Heart-c dataset; and, (c) Credit-g dataset.

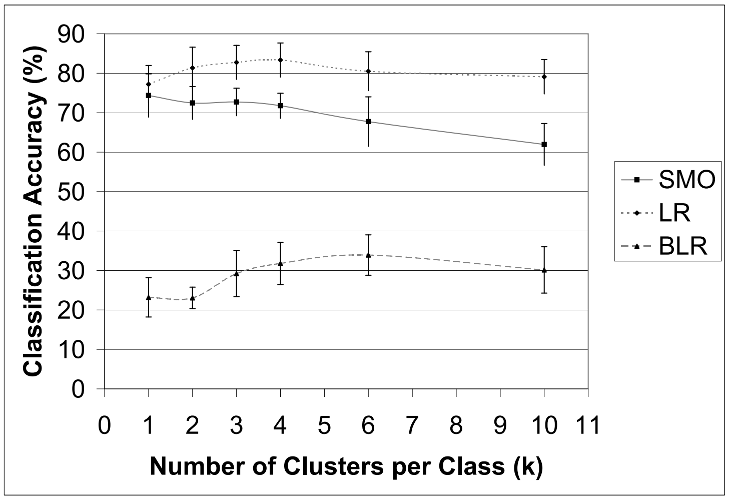
Figure 4.4, Figure 4.5, Figure 4.6, and Figure 4.7 show classification accuracy comparison using the three different linear classifiers as base classifiers for each dataset. LR* in graphs (Figure 4.5(a) and Figure 4.5(b)) means that this algorithm was evaluated using a sample of 10% for those datasets. We observe from these graphs that the classifier (BLR) usually leads to the worst performance in all cases. Moreover, the use of class decomposition does not help improving enough to be competitive against the other two classifiers. Whereas SMO and LR classifiers compete against each other in getting the best performance in all datasets. SMO classifier does better than LR in PenDigit and SatImage datasets, while performing worse in Vowel and Vehicle datasets. For Image, Heart-c, and Credit-g datasets, both algorithms LR and SMO show similar performance behavior. Although SMO and LR give comparable results in almost all cases, it should, however, be noted that LR is a more computationally demanding algorithm. Given these observations, we opt for using SMO as base classifier in our study. This selection is supported by its stability over the selected datasets.

4.2.3 Evaluation of Classification Performance

We report on a series of experiments to identify datasets suitable for class decomposition. We evaluate predictive accuracy for each dataset using the following classifiers: i) a single global linear classifier using Support Vector Machines with a linear kernel (SMO); ii) five composite linear classifiers generated by the use of class decomposition with $k = 2, 3, 4, 6, 10$ (CD2, CD3, CD4, CD6, and CD10 respectively); and iii) a high-capacity classifier using kNN with $k = 1$ (1NN). On each dataset, we report

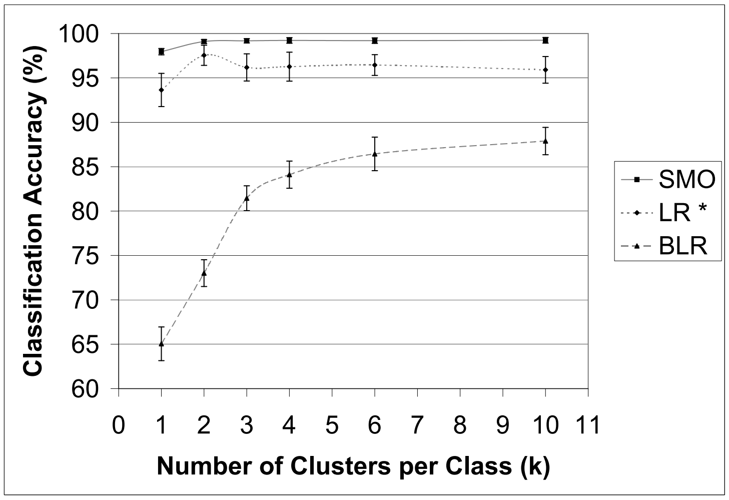


(a)

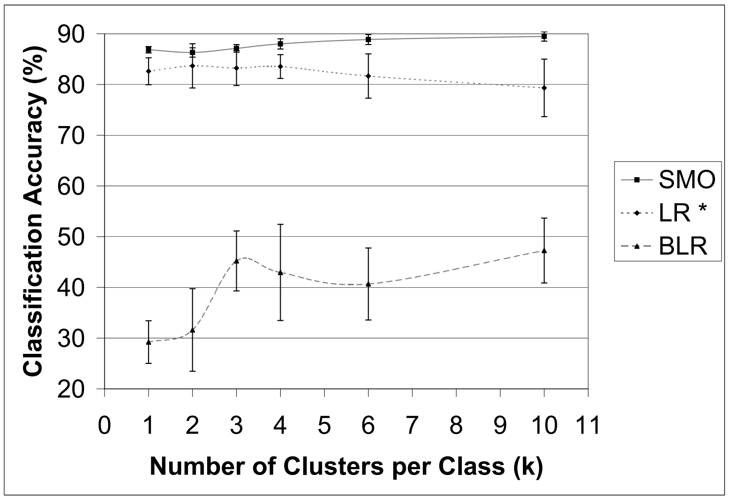


(b)

Figure 4.4: Classification performance for class decomposition using three different linear classifiers (k takes values 2,3,4,6 and 10) : (a) Vowel dataset and (b) Vehicle dataset.

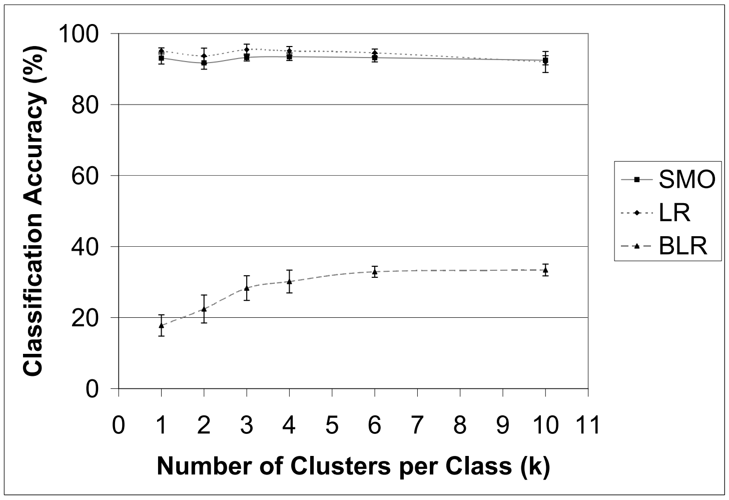


(a)

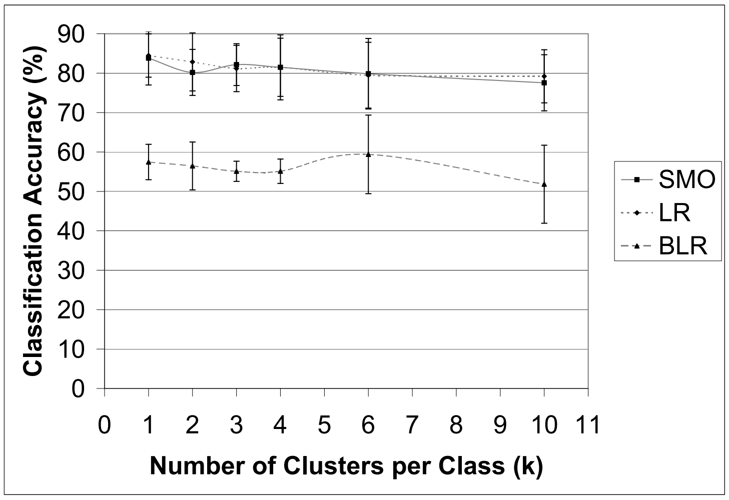


(b)

Figure 4.5: Classification performance for class decomposition using three different linear classifiers (k takes values 2,3,4,6 and 10) : (a) PenDigit dataset and (b) SatImage dataset.



(a)



(b)

Figure 4.6: Classification performance for class decomposition using three different linear classifiers (k takes values 2,3,4,6 and 10) : (a) Image dataset and (b) Heart-c dataset.

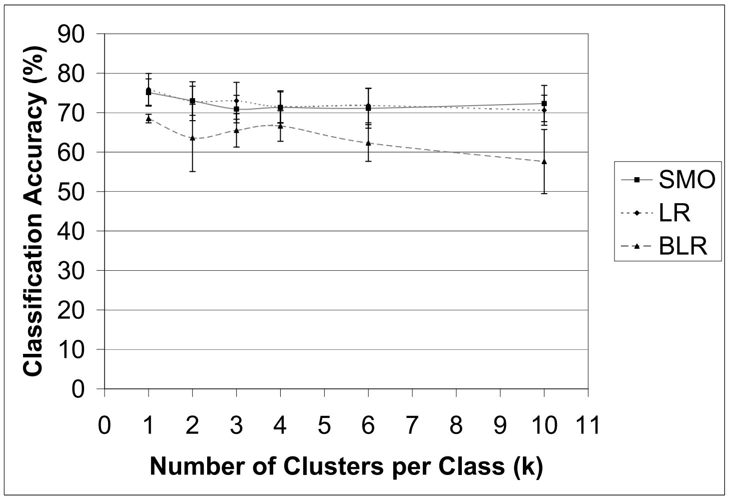


Figure 4.7: Classification performance for class decomposition using three different linear classifiers(k takes values 2,3,4,6 and 10) : Credit-g dataset.

the average of a 10-fold cross validation.

Table 4.3 displays our results. The first column describes the dataset used for our experiments. The second column reports on the accuracy of the linear classifier (SMO). The third to seventh columns show accuracy for the composite classifiers (CD2, CD3, CD4, CD6, and CD10). The eight column shows the accuracy of the high-capacity classifier (1NN). Numbers enclosed in parentheses represent standard deviations. A significant difference with respect to the linear classifier (SMO) is shown in bold. Our tests of significance assume a *t*-student distribution with a 95% confidence level.

Table 4.3: Classification accuracy

Id.	**SMO**	**CD2**	**CD3**	**CD4**	**CD6**	**CD10**	**NN**
1	71.41 (2.98)	73.03 (6.53)	66.26 (4.29)	61.62 (2.86)	54.65 (3.67)	42.83 (7.36)	**99.29 (0.83)**
2	74.36 (5.48)	72.47 (4.14)	72.71 (3.50)	71.76 (3.18)	67.74 (6.27)	61.95 (5.30)	69.86 (4.47)
3	97.96 (0.36)	**99.10 (0.23)**	**99.18 (0.24)**	**99.22 (0.32)**	**99.20 (0.31)**	**99.24 (0.32)**	**99.36 (0.17)**
4	86.85 (0.62)	86.31 (0.92)	87.12 (0.74)	88.00 (1.00)	**88.86 (0.98)**	**89.48 (0.93)**	**90.21 (1.16)**
5	93.07 (1.66)	91.73 (1.78)	93.25 (0.98)	93.46 (1.05)	93.20 (1.19)	92.47 (1.29)	**97.14 (0.62)**
6	83.80 (6.80)	80.18 (5.84)	82.16 (5.30)	81.47 (8.24)	79.85 (8.95)	77.55 (7.12)	75.88 (5.06)
7	75.10 (3.45)	73.00 (3.68)	70.90 (3.48)	71.30 (3.95)	71.10 (5.04)	72.30 (4.60)	72.00 (3.09)

Our results show only two datasets suitable for class decomposition, namely, PenDigit and SatImage. In these cases, class decomposition shows a significant gain in performance with respect to the linear classifier. For the PenDigit dataset, class decomposition using $k = 2$ is sufficient to achieve a significant gain. For the SatImage dataset, a significant improvement is obtained using $k = 6$. No performance improvement is observed in all other datasets. The high-capacity classifier (1NN) significantly outperforms the linear classifier (SMO) on 4 datasets, namely, Vowel, PenDigit, SatImage and Image. We observe that in the two datasets where 1NN outperforms SMO but CD does not (i.e., Vowel and Image), there is an inflection point (k^*) in the k value (i.e., maximum performance). k values greater than k^* showed a decrease in performance. For the Vowel dataset $k^* = 2$ and for the Image dataset $k^* = 4$. This result may seem counter-intuitive, because the greater the k, the more similar the behavior of the composite classifier to 1NN. That is, if 1NN is able to outperform the linear classifier, then we would expect that increasing k during class decomposition would produce a corresponding increase in performance. Finally, we observe that in two datasets (Heart-c and Credit-g), neither class decomposition nor 1NN outperforms the linear classifier.The loss in classification performance between SMO and 1NN serves as an indicator of data properties that lead to failure in the use of class decomposition.

4.2.4 Evaluation of Data Complexity Measures

Our experiments aim at identifying the presence of two data properties: linear separability and class overlap. Complexity measures are obtained through two steps: i) transformation of a multi-class classification problem into a set of 2-class classification problems (pairwise coupling). If there are k-classes, we build $\frac{k\times(k-1)}{2}$ two-class datasets. ii) computation of data complexity measures for each two-class problem generated in step (i); we compute $L2$, $N3$, and $N2$ described in Section 2.5, and an estimation of the difference in classification performance between a linear classifier (SMO) and a higher complexity classifier (NN),computed as $L2 - N3$.

Table 4.4 displays our estimations. The first column describes the dataset used for our experiments. The next columns (2-5) report on average values for $L2$, $N3$, $L2 - N3$, and $N2$. The sixth column captures the presence of linear separability; if $L2 = 0$ then Yes, otherwise No. The seventh column indicates low or high presence of class overlap; if $N2 <= 0.5$ then Low, otherwise High. The eighth column (CD) is marked as $(+)$ if there is presence of both non-linear separability and low overlap, otherwise it is marked as $(-)$. The $(+)$ mark suggests the use of class decomposition.

4.2.5 Data Complexity Analysis on the Suitability of Class Decomposition

We now analyze the suitability of class decomposition under linear classifiers, using the results from our empirical study (see Section 4.2).

Table 4.4: Averages on data complexity measures (per dataset)

Dataset	**L2**	**N3**	**L2-N3**	**N2**	**LS**	**Overlap**	**CD**
Vowel	0.1021	0.0000	0.1021	0.2070	No	Low	+
Vehicle	0.2423	0.1208	0.1215	0.5760	No	High	−
PenDigit	0.0086	0.0011	0.0075	0.2141	No	Low	+
SatImage	0.0410	0.0217	0.0193	0.3839	No	Low	+
Image	0.0229	0.0067	0.0162	0.1773	No	Low	+
Heart-c	0.1820	0.2570	-0.0750	0.7570	No	High	−
Credit-g	0.2970	0.3390	-0.0420	0.8690	No	High	−

4.2.5.1 Dataset Suitability

We focus first on dataset properties, and introduce a set of relevant definitions for our analysis. Let T be a data sample ($T \subset \mathcal{X} \times \mathcal{Y}$), let $M(T)$ be the optimal (along parameter k) compound model that arises from using class decomposition in conjunction with linear classifiers in T, and let $L(T)$ be a single linear classifier obtained from T.

Definition 4.2.1 *T is class decomposable, if $R(M(T)) < R(L(T))$ where $R(\cdot)$ is the risk or generalization error of a classifier.*

Definition 4.2.2 *T is linearly separable, if $L2(T) = 0$.*

We start our analysis through a concrete example. Let us denote the sample shown in Figure 1.2(a) as T_l; it is clearly linearly separable[2] (i.e., $L2(L(T_l)) = 0$), but it is not class decomposable (i.e., $R(M(T_l)) \geq R(L(T_l))$). In general, we should avoid the use of class decomposition for linearly separable classification problems because there is no guarantee of classification improvement.

We now turn to another concrete example where the data sample is not linearly separable. Let us represent as $T_{\bar{l}}$ the sample shown in Figure 1.1. Here $L2(T_{\bar{l}}) > 0$), but the sample is class decomposable (i.e., $R(M(T_{\bar{l}})) < R(L(T_{\bar{l}}))$). We observe in Figure 1.1, that the data is separable, meaning Bayes error is negligible. This fact allows us to gain insight about the conditions for success in the use of class decomposition. Specifically, when a sample is not linearly separable, but it is separable through increased model capacity, then the use of class decomposition is beneficial.

Following the analysis above, we now look into class separability without the restriction for linearity. We make use of the Nearest Neighbor(NN) algorithm, particularly, data complexity measure $N3$. We introduce the following definition:

Definition 4.2.3 *T is said to be separable, if* $N3(T) = 0$.

We can now state that $T_{\bar{l}}$ is not linearly separable (i.e., $L2(T_{\bar{l}}) > 0$) but it is separable (i.e., $N3(T_{\bar{l}}) = 0$); and as expected $T_{\bar{l}}$ is class decomposable (i.e.,

[2]We note that our definition for linear separability is limited to T; the training error is frequently an optimistic approximation to the true error.

$R(M(T_{\bar{l}})) < R(L(T_{\bar{l}})))$.

The example above enables us to move forward to the general case where $L2(T) > 0$ and $N3(T) > 0$. We stated that when a sample is not linearly separable, but Bayes error is negligible, an opportunity exists for improvement using class decomposition. We argue such scenario can be captured by looking at a positive difference in the classification performance between a linear classifier and a high-capacity classifier. We make the following definition:

Definition 4.2.4 *The complexity error gap when using class decomposition in T is E(T) := L2(T)- N3(T).*

In our definition, we take the performance of a nearest neighbor algorithm (1NN) as an approximation to the classifier with best case performance using class decomposition. The rationale behind this is to establish an optimistic upper bound in performance improvement.

We now introduce the definition of *reducible error*, to understand when to expect a performance gain using class decomposition.

Definition 4.2.5 *T is said to have a reducible error, if $E(T) > 0$.*

The definition enables us to understand the advantage that comes when generating a combination of linear classifiers through class decomposition. Specifically, linear classifiers exhibit high bias and low variance, and as such are less affected by noisy data, i.e., they are stable. If the error difference between a linear classifier and 1NN is negative, then the likelihood of noisy data increases, producing an irrecoverable error for 1NN. Class decomposition is not appropriate here. As an illustration, Figure 1.2(b) shows a two-dimensional sample characterized by a high degree of overlap between the two classes. Class decomposition does not result in any improvement because of high Bayes error. To avoid the use of class decomposition in this type of scenario, we will say that T has low class overlap, if $N2(T) < \gamma$, where $0 < \gamma < 1$ will be a user-defined constant.

We are now ready to define data properties favorable for class decomposition. Previous work [29, 9, 31] suggests the presence of a heterogeneous class distribution (i.e., a distribution where sub-classes are widely spread over the feature space) as a condition for the success in the use of class decomposition. Previous work fails to detect the presence of this condition in a sample. We propose a simple and measurable condition to identify the success of class decomposition by verifying the presence of two data properties: i) there is a positive difference between the error of a linear classifier and the error of a high-capacity classifier (i.e., there is room for improvement when we increase model complexity by means of class decomposition); and ii) class overlap or Bayes error is minimal. We formalize these conditions with the following proposition:

Proposition 4.2.1 *A data sample T is class decomposable if T meets the following two conditions:*

(i) T has reducible error.

(ii) T has low class overlap.

We now return to our empirical study to validate our proposition. The eighth column (CD) in Table 4.4 marks as (+) those samples that are class decomposable, and as (-) otherwise. Vowel, PenDigit, SatImage, and Image datasets are marked as (+); these datasets are characterized by having both a reducible error and low class overlap (γ was set to 0.5). Under these two data conditions, class decomposition showed to be beneficial when compared to a single linear classifier, as shown in Table 4.3. We note that only in two datasets (PenDigit and SatImage) is this difference significant. Samples marked as (-) are rejected for various reasons. Vehicle dataset is rejected because $N2 > 0.5$, regardless of the difference in $L2 - N3$. For Heart-c and Credit-g datasets, the difference $L2 - N3$ is indeed negative.

Results show the value of Proposition 4.2.1 for the successful use of class decomposition. Specifically, when omitting significant differences, sensitivity and specificity are 100% for (+) and (-) classes; under significant differences, sensitivity is 100% for class (+) but there is a decrease in the specificity for class (-) to 60%. From a practical standpoint, Proposition 4.2.1 can be employed to identify the cases where no decomposition is recommended. For cases when it is recommended, an additional analysis can be effected to improve in computational efficiency as described next.

4.2.5.2 Class Suitability

In this section we deepen our analysis by looking at learning performance on individual classes. Our methodology follows the next steps: i) for each class C_i, we define a new set of subproblems, one for every pairwise coupling of C_i and all other classes; subproblems are stored under the set S_{C_i}; ii) compute averages for $L2$, $N3$, $L2-N3$, and $N2$ in S_{C_i}; iii) for each sub-problem in S_{C_i}, create the corresponding k-decomposition via clustering using the following values for $k = 2, 3, 4, 6, 10$, and evaluate predictive accuracy for a single global linear classifier and a composite linear classifier generated through class decomposition (with k clusters per class). For each sub-problem, report on accuracy estimated through 10-fold cross validation.

Table 4.5 displays results for the Vowel dataset (a dataset reported as a successful case for decomposition in two previous reports [29, 9]). The first column describes the class under analysis. The 2nd-5th columns report on the average value of $L2$, $N3$, $L2-N3$, and $N2$ respectively. The sixth column reports on the accuracy of the linear classifier (SMO). The seventh column reports on the accuracy for the best composite classifier obtained among the different values of k using class decomposition (CD*). The eighth column reports the difference in accuracy between (SMO) and (CD*). The ninth column shows the number of clusters employed by the best linear composite classifier.

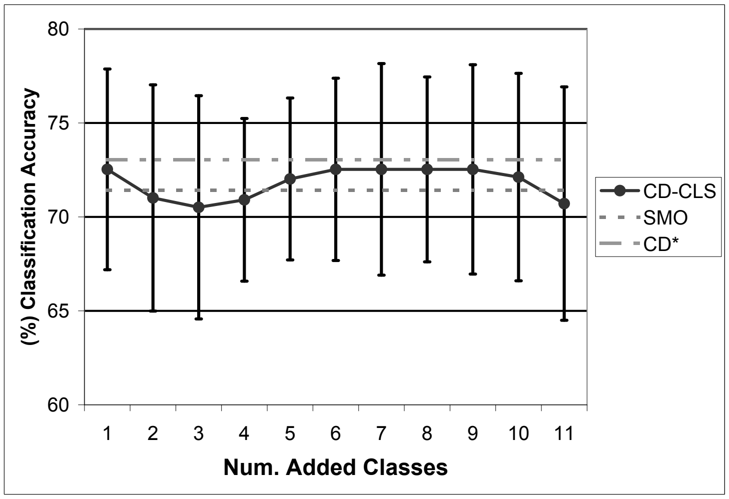
Table 4.5: Data complexity measures and performance per class (Vowel dataset)

Id (Class Name)	**L2**	**N3**	**L2-N3**	**N2**	**SMO**	**CD***	**Diff.**	**#Cls.**
1 (hid)	0.0979	0.0000	0.0979	0.1548	98.50	99.56	1.06	2
2 (hId)	0.1274	0.0000	0.1274	0.1968	95.22	98.61	3.39	3
3 (hEd)	0.1002	0.0000	0.1002	0.2000	97.00	99.33	2.33	3
4 (hAd)	0.0837	0.0006	0.0831	0.1914	98.28	99.39	1.11	2
5 (hYd)	0.0939	0.0006	0.0933	0.2139	96.50	96.78	0.28	2
6 (had)	0.1488	0.0018	0.1470	0.2427	93.22	96.39	3.17	2
7 (hOd)	0.1331	0.0006	0.1325	0.2177	96.39	98.72	2.33	2
8 (hod)	0.0612	0.0006	0.0606	0.1946	97.95	98.56	0.61	3
9 (hUd)	0.0945	0.0000	0.0945	0.2573	95.83	97.39	1.56	2
10 (hud)	0.0688	0.0000	0.0688	0.1908	98.44	99.00	0.56	3
11 (hed)	0.1145	0.0006	0.1139	0.2168	96.78	99.00	2.22	2

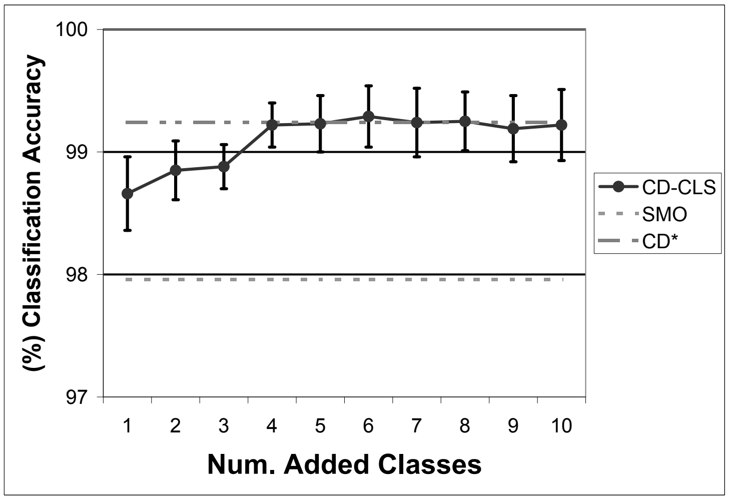
If for each class in Table 4.5 we apply Proposition 4.2.1 to predict the suitability of class decomposition, we would observe all classes labeled as (+) (i.e., all sub-problems are suitable for class decomposition). The eighth column shows a positive value on every class. This type of analysis can be employed to select just a sub-set of classes for decomposition. We can, for example, sort all sub-problems in descending order based on the value of the eighth column (i.e., the difference in performance between SMO and CD*), and iteratively add classes until we reach an inflection point. Specifically, we begin by decomposing the class with highest performance difference (8th column, Table 4.5), and keep adding more classes (in decreasing order, 8th column) until all classes are included.

Figure 4.8(a) shows accuracy when incorporating this iterative approach. As a reference, we also show performance for both the single linear classifier (SMO) and the best class decomposition classifier (CD*). As can be seen in Table 4.5, class 2 corresponds to the highest performance difference. In principle, we could decompose this class only and still achieve good results (Figure 4.8(a)). In summary, a simple analysis can show a reduced number of classes in true need for decomposition.

In order to verify that these results are consistent with our previous analysis, we report on the same type experiment for the PenDigit Dataset. Results are given in Table 4.6 and Figure 4.8(b). We observe in Figure 4.8(b), that the maximum performance value is reached using only 6 classes, namely, $1, 2, 6, 8, 9, 10$. This analysis saves on the computational cost of decomposing four additional classes.



(a)



(b)

Figure 4.8: Cumulative classwise classification performance for: (a) Vowel Dataset; (b) PenDigit Dataset.

Table 4.6: Data complexity measures and performance per class (PenDigit dataset)

Id (Class Name)	**L2**	**N3**	**L2-N3**	**N2**	**SMO**	**CD***	**Diff.**	**#Cls.**
1 (0)	0.0086	0.0008	0.0078	0.1681	99.48	99.93	0.45	6
2 (1)	0.0210	0.0015	0.0195	0.2232	99.49	99.78	0.28	6
3 (2)	0.0066	0.0017	0.0049	0.2249	99.82	99.89	0.07	6
4 (3)	0.0045	0.0020	0.0025	0.2368	99.72	99.78	0.06	10
5 (4)	0.0031	0.0005	0.0026	0.2144	99.89	99.95	0.06	3
6 (5)	0.0146	0.0008	0.0138	0.2090	99.33	99.88	0.55	10
7 (6)	0.0010	0.0006	0.0005	0.2047	99.95	99.96	0.02	6
8 (7)	0.0071	0.0013	0.0058	0.2226	99.74	99.89	0.15	3
9 (8)	0.0105	0.0005	0.0100	0.2161	99.36	99.95	0.60	6
10 (9)	0.0091	0.0011	0.0079	0.2209	99.68	99.82	0.14	10

Chapter 5

Summary, Conclusions, and Future work

5.1 Summary

A central aim in machine learning and data mining is to facilitate the use of learning algorithms to a vast comunity of practioners and researchers. This goal is achieved when both tools and guidelines are available for end-users. The major original contribution of this thesis is to facilitate the use of the class decomposition algorithm providing practioners with both an implementation and guidelines of the conditions for the suitability of this algorithm. The implementation is strongly supported by the fact of being part of a widely used and consolidated experimental framework (WEKA), while the guidelines were conceived as a set of practical rules.

5.2 Conclusions

In this dissertation, we describe a study aimed at understanding under what conditions a data sample is suitable for class decomposition (using linear classifiers). We focus our methodology in the identification of two data properties: 1) linear separability using $L2$ and $N3$ measures; and 2) class overlap using $N2$ measure. Our analysis is based on experiments performed on seven real-world domains; results support the following conclusions:

1. if the classification problem is linearly separable ($L2 = 0$), then class decomposition should be avoided;

2. if the classification problem is not linearly separable ($L2 > 0$), and a high-capacity model is able to separate the classes ($N3 = 0$), then this sample is suitable for class decomposition;

3. if the classification problem is not linearly separable ($L2 > 0$) and a high-capacity model is not able to separate the classes ($N3 > 0$), then we propose using the difference $L2 - N3$ as an indicator for the use of class decomposition. If $L2 - N3 <= 0$ then class decomposition should be avoided because there is no room for performance improvement. If $L2 - N3 > 0$ the use of class decomposition can be effected when such a difference is above an user-defined threshold value (defined by γ).

4. If there is high overlap between classes (i.e., $N2$ is above a threshold), then class

decomposition should be avoided because there is no guarantee for improvement. In addition, the proposed method can be applied at a more granular level to identify those classes specifically suitable for decomposition. This last method adds in computational efficiency by avoiding class decomposition when unnecessary.

5.3 Future Work

Our research exhibits the following limitations:

1. The use of a nearest neighbor algorithm carries high computational cost.
2. The threshold values are chosen empirically.
3. There is a lack of a time-consuming analysis for the used data complexity measures.

These limitations suggests a number of possible directions for future work that builds on our work:

1. To use an approximation to $N3$ with a lower computational cost, by computing the fraction of examples lying along the class boundary by means of a minimum spanning tree.
2. To perform a similar analysis of $N2$ as in [24] to set the $N2$ threshold according to the dimensionality and the type of attributes (e.g., nominal, numeric or mixed).

3. To elaborate a study about the computational cost for the data complexity measures employed in our method.

4. To perform a similar empirical study including feature selection.

5. To develop a parallel implementation for improving the computational cost of computing the data complexity measures.

Bibliography

[1] Mitra Basu and Tin Kam Ho, editors. *Data Complexity in Pattern Recognition.* Springer-Verlag, London, 2006.

[2] Christopher M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 1st edition, 2006.

[3] Fu Chang, Chien-Yang Guo, Xiao-Rong Lin, and Chi-Jen Lu. Tree Decomposition for Large-Scale SVM Problems. *Journal of Machine Learning Research*, 99:2935–2972, Oct 2010.

[4] Kai-Wei Chang and Dan Roth. Selective block minimization for faster convergence of limited memory large-scale linear models. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011*, pages 749–754, 2011.

[5] Nitesh V. Chawla and Kevin W. Bowyer. Designing multiple classifier systems for face recognition. In *Proceedings of the 6th International Conference on Multiple Classifier Systems, Seaside, CA, USA, June 13-15, 2005,*, pages 407–416, 2005.

[6] Feng Chen, Chang-Tien Lu, and Arnold P. Boedihardjo. On locally linear classification by pairwise coupling. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM), December 15-19, 2008, Pisa, Italy*, pages 749–754, 2008.

[7] Haibin Cheng, Pang-Ning Tan, and Rong Jin. Efficient algorithm for localized support vector machine. *IEEE Trans. Knowl. Data Eng.*, 22(4):537–549, 2010.

[8] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification.* Wiley New York, 2nd edition, 2001.

[9] Dmitriy Fradkin. Clustering inside classes improves performance of linear classifiers. In *Proceedings of the 20th IEEE International Conference on Tools with Artificial Intelligence (ICTAI), November 3-5, 2008, Dayton, Ohio, USA*, volume 2, pages 439–442. IEEE Computer Society, 2008.

[10] Zhouyu Fu, Antonio Robles-Kelly, and Jun Zhou. Mixing linear svms for nonlinear classification. *IEEE Transactions On Neural Networks*, 21:1963–1975, December 2010.

[11] Kun Gai and Changshui Zhang. Learning discriminative piecewise linear models with boundary points. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI), Atlanta, Georgia, USA, July 11-15, 2010*, pages 444–450, 2010.

[12] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, 2009.

[13] Tin Ho. Data complexity analysis: Linkage between context and solution in classification. In Niels da Vitoria Lobo, Takis Kasparis, Fabio Roli, James Kwok, Michael Georgiopoulos, Georgios Anagnostopoulos, and Marco Loog, editors, *Structural, Syntactic, and Statistical Pattern Recognition*, volume 5342 of *Lecture Notes in Computer Science*, pages 986–995. Springer Berlin / Heidelberg, 2008.

[14] Tin Kam Ho and Mitra Basu. Complexity measures of supervised classification problems. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(3):289–300, 2002.

[15] Hong-Dong Li, Yizeng Liang, and Qing-Song Xu. Support vector machines and its applications in chemistry. *Chemometrics and Intelligent Laboratory Systems*, 95(2):188–198, february 2009.

[16] Yujian Li, Bo Liu, Xinwu Yang, Yaozong Fu, and Houjun Li. Multiconlitron: A general piecewise linear classifier. *IEEE Transactions on Neural Networks*, 22(2):276–289, 2011.

[17] Julian Luengo and Francisco Herrera. Domains of competence of fuzzy rule based classification systems with data complexity measures: A case of study using a fuzzy hybrid genetic based machine learning method. *Fuzzy Sets and Systems*, 161(1):3–19, 2010.

[18] Geoffrey J. McLachlan and Thriyambakam Krishnan. *The EM Algorithm and Extensions (Wiley Series in Probability and Statistics)*. Wiley-Interscience, 2nd edition, March 2008.

[19] Donald Michie, D. J. Spiegelhalter, C. C. Taylor, and John Campbell, editors. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, Upper Saddle River, NJ, USA, 1994.

[20] Yusuke Nojima, Shinya Nishikawa, and Hisao Ishibuchi. A meta-fuzzy classifier for specifying appropriate fuzzy partitions by genetic fuzzy rule selection with data complexity measures. In *Proceedings of the IEEE International Conference on Fuzzy Systems (FUZZ)*, pages 264 –271, june 2011.

[21] Dan Pelleg and Andrew W. Moore. X-means: Extending k-means with efficient estimation of the number of clusters. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 727–734, 2000.

[22] Erinija Pranckeviciene, Richard Baumgartner, and Ray L. Somorjai. Using domain knowledge in the random subspace method: Application to the classification of biomedical spectra. In *Proceedings of the 6th International Conference on Multiple Classifier Systems, Seaside, CA, USA, June 13-15, 2005,*, pages 962–971, 2005.

[23] Franck Rapaport, Emmanuel Barillot, and Jean P. Vert. Classification of array-cgh data using fused svm. *Bioinformatics*, 24(13):i375–i382, july 2008.

[24] Jose S. Sanchez, Ramon A. Mollineda, and Jose M. Sotoca. An analysis of how training data complexity affects the nearest neighbor classifiers. *Pattern Anal. Appl.*, 10:189–201, July 2007.

[25] Nicola Segata and Enrico Blanzieri. Fast and scalable local kernel machines. *J. Mach. Learn. Res.*, 99:1883–1926, August 2010.

[26] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[27] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory (Information Science and Statistics)*. Springer, 1999.

[28] Brijesh Verma and Ashfaqur Rahman. Cluster-oriented ensemble classifier: Impact of multicluster characterization on ensemble classifier learning. *IEEE Trans. on Knowl. and Data Eng.*, 24(4):605–618, April 2012.

[29] Ricardo Vilalta, Murali-Krishna Achari, and Christoph F. Eick. Class decomposition via clustering: A new framework for low-variance classifiers. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM), 19-22 December 2003, Melbourne, Florida, USA*, pages 673–676, 2003.

[30] Junjie Wu. K-means based local decomposition for rare class analysis. In *Advances in K-means Clustering*, Springer Theses, pages 125–153. Springer Berlin Heidelberg, 2012.

[31] Junjie Wu, Hui Xiong, and Jian Chen. Cog: local decomposition for rare class analysis. *Data Mining and Knowledge Discovery*, 20(2):191–220, 2010.

[32] Hsiang-Fu Yu, Cho-Jui Hsieh, Kai-Wei Chang, and Chih-Jen Lin. Large linear classification when data cannot fit in memory. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI), Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 2777–2782, 2011.

[33] Jie Zhou, Hanchuan Peng, and Ching Y. Suen. Data-driven decomposition for multi-class classification. *Pattern Recognition*, 41(1):67–76, 2008.