2012

CONTRIBUTIONS TO K-MEANS CLUSTERING AND REGRESSION VIA CLASSIFICATION ALGORITHMS

Raied Salman
Virginia Commonwealth University

CONTRIBUTIONS TO K-MEANS CLUSTERING AND REGRESSION VIA CLASSIFICATION ALGORITHMS

A Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University

by

RAIED SALMAN
Ph.D., Brunel University (United Kingdom), 1989
M.Sc., University of Technology (Iraq), 1978
B.Sc., University of Technology (Iraq), 1976
Dip.Tch., Auckland University (New Zealand), 1996

Director: VOJISLAV KECMAN
ASSOCIATE PROFESSOR, DEPARTMENT OF COMPUTER SCIENCE

Virginia Commonwealth University
Richmond, Virginia
April, 2012

Table of Contents

Page

List of Tables

List of Figures

Abstract

CONTRIBUTIONS TO K-MEANS CLUSTERING AND REGRESSION VIA CLASSIFICATION ALGORITHMS

By Raied Salman, Ph.D.

A Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

Virginia Commonwealth University, 2012

Major Director: Vojislav Kecman
Associate Professor, Department of Computer Science

The dissertation deals with clustering algorithms and transforming regression problems into classification problems. The main contributions of the dissertation are twofold; first, to improve (speed up) the clustering algorithms and second, to develop a strict learning environment for solving regression problems as classification tasks by using support vector machines (SVMs). An extension to the most popular unsupervised *clustering* method, *k*-means algorithm, is proposed, dubbed *k*-means^2 (*k*-means squared) algorithm, applicable to ultra large datasets. The main idea is based on using a small portion of the dataset in the first stage of the clustering. Thus, the centers of such a smaller dataset are computed much faster than if computing the centers based on the whole dataset. These final centers of the first stage are naturally much closer to the locations of the final centers rendering a

great reduction in the total computational cost. For large datasets the speed up in computation exhibited a trend which is shown to be high and rising with the increase in the size of the dataset. The total transient time for the fast stage was found to depend largely on the portion of the dataset selected in the stage. For medium size datasets it has been shown that an 8-10% portion of data used in the fast stage is a reasonable choice. The centers of the 8-10% samples computed during the fast stage may oscillate towards the final centers' positions of the fast stage along the centers' movement path. The slow stage will start with the final centers of the fast phase and the paths of the centers in the second stage will be much shorter than the ones of a classic *k*-means algorithm. Additionally, the oscillations of the slow stage centers' trajectories along the path to the final centers' positions are also greatly minimized.

In the second part of the dissertation, a novel approach of posing a solution of regression problems as the multiclass classification tasks within the common framework of kernel machines is proposed. Based on such an approach both the nonlinear (NL) regression problems and NL multiclass classification tasks will be solved as multiclass classification problems by using SVMs. The accuracy of an approximating classification (hyper)Surface (averaged over several benchmarking data sets used in this study) to the data points over a given high-dimensional input space created by a nonlinear multiclass classifier is slightly superior to the solution obtained by regression (hyper)Surface. In terms of the CPU time needed for training (i.e. for tuning the hyperparameters of the models), the nonlinear SVM classifier also shows significant advantages. Here, the comparisons between the solutions obtained by an SVM solving given regression problem as a classic

SVM regressor and as the SVM classifier have been performed. In order to transform a regression problem into a classification task, four possible discretizations of a continuous output (target) vector **y** are introduced and compared. A very strict double (nested) cross-validation technique has been used for measuring the performances of regression and multiclass classification SVMs. In order to carry out fair comparisons, SVMs are used for solving both tasks - regression and multiclass classification. The readily available and most popular benchmarking SVM tool, LibSVM, was used in all experiments. The results in solving twelve benchmarking regression tasks shown here will present SVM regression and classification algorithms as strongly competing models where each approach shows merits for a specific class of high-dimensional function approximation problems.

CHAPTER 1: INTRODUCTION

Because of the sheer amount and complexity of the information available, for example from, weather data, geophysical data, drug testing data, health care imaging information systems, atomic particle accelerator detector data, www pages, e-commerce data, …, etc, nowadays, engineers and scientists rely heavily on computers to process and analyze data. This is why Machine Learning (ML) has become an emerging topic of research that has been employed by an increasing number of disciplines to automate complex decision-making and problem-solving tasks. This is because the goal of ML is to extract knowledge from experimental data and to use computers for complex decision-making. In other words, decision rules are extracted automatically from data by utilizing the speed and the robustness of the machines. Machine learning techniques can be divided into three major groups based on the types of problems they can solve, namely, *supervised*, *semi-supervised* and *unsupervised* learning.

The supervised learning algorithm attempts to learn the input-output relationship (dependency or function) $f(x)$ by using a training data set $\{\chi = [\mathbf{x}_i, y_i], i = 1, \dots, n\}$ consisting of n pairs $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$, where the inputs $\mathbf{x}$ are m-dimensional vectors $\mathbf{x} \in \Re^m$ and the labels (or system responses) y are discrete (e.g., Boolean) for classification problems and continuous values $y \in \Re$ for regression tasks. Support Vector Machines (SVMs) and Artificial Neural Network (ANN) are two of the most popular techniques in this area.

One of the dissertation contributions will be in introducing and presenting how the regression problems can be transformed into the multi-class classification tasks. This will obviously include some kind of discretization in order to get class labels. Each regression problem can be considered the classification one having as many classes as there are different y_i values in the output vector **y**. At the same time it is easy to see that a classification task with too many classes and very little data in each class (mostly one datapoint only) is a very ill-posed problem. In order to overcome such a learning setting, the next obvious step is to group several datapoints having close values into the same class. Here, we will present a few methods describing how the discretization step can be done and evaluate their performances within a strict experimental environment of a double (nested) *k*-fold crossvalidation. The modeling tool for both regression and multiclass classification class will be SVMs because they have shown excellent results in real world applications.

The second big group of standard learning algorithms is the so-called unsupervised algorithms when there are only raw data $\mathbf{x} \in \Re^m$ without the y_i attached (i.e., there is a 'no-teacher' typically provided by labels in supervised learning). There are many methods used in unsupervised machine learning areas. The most popular (most standard) algorithms are various clustering techniques, (principal or independent) component analysis routines and association rules. The dissertation's largest section is devoted to clustering methods. In fact, one of the contributions of the work done here will be in speeding up *k*-means clustering algorithm by dividing it into two parts; namely into first (fast) stage which uses just a fraction of data for finding initial cluster centers, and in the second (slow and final) stage in which all the data points are used for fine tuning the terminal clusters' centers' positions.

The dissertation will be arranged as follows; chapter 2 introduces backgrounds about the clustering, classification theory and the contribution of the dissertation. Chapter 3 describes, in detail, the proposed method of the 2-stage, *k*-means^2 algorithm with a few examples to demonstrate the validity of the proposed method. Chapter 4 explains the proposed solution of regression problems as multiclass classification tasks within the common framework of kernel machines. The last chapter draws conclusions on the proposed methods and gives suggestions for future work.

CHAPTER 2: BACKGROUND, RELATED WORK AND CONTRIBUTIONS

2.1 Background

Machine learning (ML) techniques are used for solving various problems in contemporary science, medicine, finance, engineering and many other areas. There is almost no field of human activities untouched by ML tools (a.k.a. neural networks, support vector machines, data mining, and/or knowledge discovery, etc). ML tools are aimed at solving classic statistical tasks i.e. clustering (grouping), classification (pattern recognition) and regression (function approximation). Clustering in any data set can be achieved by minimizing the intra-cluster dissimilarity and maximizing the inter-cluster dissimilarity.

A comprehensive clustering techniques review carried out by [Jain, Murty and Flynn, 1999] suggested that all clustering techniques are based on hierarchal, partitional and taxonomy approaches. They further subdivided the clustering techniques into the following aspects:

1-Agglomerative vs. divisive

2- Monothetic vs. polythetic

3- Hard vs. fuzzy

4- Deterministic vs. stochastic

5- Incremental vs. non-incremental

The *k*-means algorithm, which belongs to partitional clustering, is used in clustering method for small datasets; see [McQueen, 1967] for example.

However, the high computational cost of applying the *k*-means algorithm for large and ultra-large data sets (> 1 million and 100 million data respectively) is unavoidable. There-

fore the ability to analyze these datasets in a reasonable amount of time and at a reasonable cost has not kept pace with newer techniques. Until a way is found to unlock these datasets, the information they contain shall continue to be wasted.

The available methods of clustering that handle large datasets mainly use the vector-space, such as *k*-medoids rather than distance-space algorithms, such as *k*-means algorithm. CLARA algorithm has been suggested by [Kaufman and Rousseeuw, 1990] for tackling large datasets. They suggested the use of part of the data to then populate the results for all the data. However, the CLARA method has produced unreliable results since the sample of the datasets does not necessarily reflect the whole dataset and some clusters may be missed entirely. An improvement to the CLARA algorithm, developed by [Ng and Han, 1994], is based on the use of random neighbor samples dubbed as CLARANS. Unfortunately, CLARANS works only for small datasets. Another method, suggested by [Cutting, Karger, Pedersen and Tukey, 1992], uses clustering by means of *Fractionization* and *Refractionization*. Their idea was to split the data into manageable subsets (called fractions) and then apply the hierarchical method to each fraction. The resultant clusters from these fractions are then clustered into $k \epsilon \mathbb{N}$ groups by the same clustering method. In essence it is an iterative approach. The number of groups, k to be estimated must be supplied in advance. The major problem in this method is the formation of the meta-observations (sample data to form the clusters). A different approach, called BIRCH, was proposed by [Zhang, Ramakrishnan and Livny, 1996] and divides the data into sub-clusters, known as "cluster-features". The BIRCH method depends on building trees and sub-trees until the memory is full. This approach fills the memory too fast to be usable, especially for large

datasets. An approach based on distance-space was suggested by [Ganti et.al, 1999], applicable mainly to clustering huge datasets. The algorithms that were developed are referred to as BUBBLE and BUBBLE-FM (improved version) and they basically scan over the database once and use the vector operations to calculate the distance between two points in the space. These algorithms are, in principle, extensions of the BIRCH method proposed early 1996. Both algorithms essentially use the tree and sub-tree to generate or form the clusters. All the previous methods (BUBBLE, BUBBLE-FM, and BIRCH) suffer from one main problem- filling the memory of a computer quite quickly when the data is large. The DBSCAN [Ester, et al., 1996] clustering method, on the other hand, is used for clustering data without the requirements of a priori number of clusters. This is different than the normal *k*-means algorithm, which requires an estimate of the number of clusters before the starting of the algorithm. A more detailed description of DBSCAN is shown in the coming section of 2.2.1.

The most standard and popular approach to clustering is known as the *k*-means algorithm. It starts by arbitrarily selecting starting centers at the outset and computes the first set of dynamic centers. (The adjective dynamic is used to denote the changing i.e., moving character of centers' positions during the clustering). These dynamic centers are used as the starting centers for the next dynamic centers. The process repeats over the whole dataset until the final centers are found. The distances between the consecutive dynamic centers will diminish with each iteration. Obviously, this approach will lead to a computationally prohibitive cost when the dataset is relatively large.

A faster algorithm than the *k*-means algorithm, utilizing two stage clustering, is proposed [Salman et al., a, b, 2011]. This method divides the *k*-means algorithm into two stages: the first one is the *fast stage*, which uses a small part of the data selected at random, satisfying Theorem 1 in section 3.2, and the second stage uses the whole data set and is called the *slow stage*. In the *slow stage* the whole dataset will be used but within less iteration steps than is needed in a classic one stage *k*-means algorithm clustering algorithms. The main drive of the proposed method is to move the dynamic centers closer to the final location using part of the dataset rather than the whole dataset.

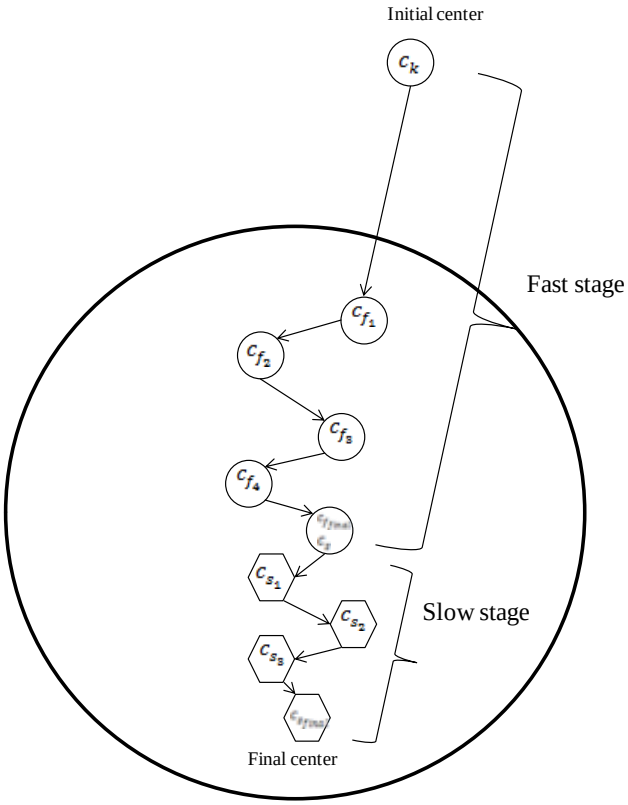


Figure 2.1: The proposed *k*-means^2 algorithm approach schematic diagram shown for one center movement

An illustration of the movement of the centers is depicted in Figure 2.1. It can be observed that the *fast stage* starts the computation from an arbitrarily selected initial center c_k using only a portion of the dataset. This will lead to the next (dynamic) centers, $c_{f_1}, c_{f_2}, \ldots, c_{f_{final}}$. The final center of the fast stage, $c_{f_{final}}, (c_s)$, is used as the starting center for the *slow stage*. In contrast the *slow stage* will use the whole dataset to compute the new dynamic centers, $c_{s_1}, c_{s_2}, \ldots, c_{s_{final}}$iteratively until it terminates at the final center, $c_{s_{final}}$. It is clear that the distances between the dynamic centers in the *fast stage* are generally longer than the positions changes between the dynamic centers during the *slow stage.* Moreover, the time and cost of computation for the transitions in the *fast stage* is much lower than the cost of computation in the shorter transition spaces within the *slow stage* due to the size of the dataset used. Obviously, the number of iterations in the *slow stage* are reduced due to the action of the *slow stage* which effectively moves the arbitrarily selected initial center closer to its final position. The overall cost of computation is expected to be much lower than the cost of computation attributed to the *k*-means algorithm. Consequently, the proposed approach will be able to accommodate much larger data sets with faster results at an even lower overall cost of computation.

The second part of the dissertation proposes solving regression problems as classification ones, a novel approach of solving regression tasks as multiclass classification ones within the common framework of kernel machines. Recently, in [Kecman and Yang 2009], it has been shown how one can solve regression problems by posing them as the multiclass classification tasks. The multiclass classifier used in that paper is the Adaptive Local

Hyperplane (ALH) introduced in [Yang and Kecman 2008]. The choice of the ALH was natural because [Yang and Kecman 2008] showed its very good results on eleven standard benchmarking classification data sets, outperforming seven other machine learning tools including KNN classifier, SVMs and K-local hyperplane (linear manifold) distance nearest neighbor (HKNN) algorithm. The results presented in [Kecman and Yang 2009] were a big encouragement and provided strong motivation for exploring the method of transforming regression problems into the classification ones within a much stricter experimental approach.

A strict comparison of a nonlinear (NL) regressor and an NL multiclass classifier is investigated and an approach to solving the regression problems is proposed.

As for the model comparison environment, the double (nested) cross-validation was implemented. Note that all results presented in [Kecman and Yang 2009] have been obtained by using a standard single (10-fold) cross-validation. This was a fair approach for validating whether the idea of transforming regression problems into classification tasks is feasible at all. However, such an approach is not quite valid for comparing different models because in a single cross-validation all the data points available are used for both hyperparameter determination and accuracy estimation. While such a procedure is good for estimating the hyperparameters of a single model it does not produce a true estimate of the accuracy of the given data set. If the main goal is to compare different ML models on the same data sets and under the same conditions, double cross-validation must be used [Yang and Kecman 2010]. The results in solving the twelve benchmarking regression tasks shown here present SVM regression and classification algorithms as strongly competing models

where each approach shows merits for a specific class of high-dimensional function approximation problems. In this dissertation we have developed algorithms strictly based on SVM's for solving regression problem as classification within the double cross validation experimental environment. Hence, all the results obtained do not depend upon different modeling tools and they show the true properties of the algorithms analyzed.

2.2 Clustering: basic concepts

Clustering is the process of partitioning a collection of objects into groups, called *clusters*, such that "similar" objects fall into the same group. Similarity between objects is captured by a distance function.

Definitions and Notations:

Assume that $\boldsymbol{X} \subset \mathbb{R}^d$ is a finite set of *n* points and *d* is the dimension of the data (features, attributes). The number of clusters is *k* which is an integer >1 since we consider that the data cover more than one cluster. The clustering procedure is to find $\boldsymbol{S} = (S_1, \dots, S_k)$ groups in which the data is divided into the *k* clusters without assigning one point into two or more clusters. Every cluster has one center $c_i,\ i \in k$ such that all centers are defined by $\boldsymbol{c} = (c_1, \dots, c_k)$.

The following conditions are satisfied for the *k*-means algorithm:

$$S_i \neq \emptyset, i = 1, \dots, k \tag{2.1}$$

$$\bigcup_{i=1}^{k} S_i = X \tag{2.2}$$

$$S_i \bigcap S_j = \emptyset, i \neq j, i, j = 1, \dots, k \tag{2.3}$$

Measuring similarity (distance) has to be used in order to decide on the belonging of the point or the vector to any one cluster S_i . There are many similarity measures which can be used. The one that will be used in this dissertation is a modification of the Euclidean dis-

tance. The square root part of the equation has been removed to gain some speed up during the distance calculations. The role of the square root is a scaling factor to the distance which has no effect in our case. Thus the Modified Euclidean distance is given by:

$$d(\mathbf{X}_i, \mathbf{X}_j) = \sum_{l=1}^{d} (x_{i,l} - x_{j,l})^2 \tag{2.4}$$

$$d(\mathbf{X}_i, \mathbf{X}_j) = \|\mathbf{X}_i - \mathbf{X}_j\| \tag{2.5}$$

The basic *k*-means algorithm is shown below in pseudo code format as well as in graphical format as depicted in Figure 2.2:

1. Create k centroids to initialize the algorithm.

2. Assign each of the n data to its closest centroid.

3. Update the centroids of the clusters composed of the recently assigned data.

4. If there is change of at least one centroid go to step 2, otherwise stop.

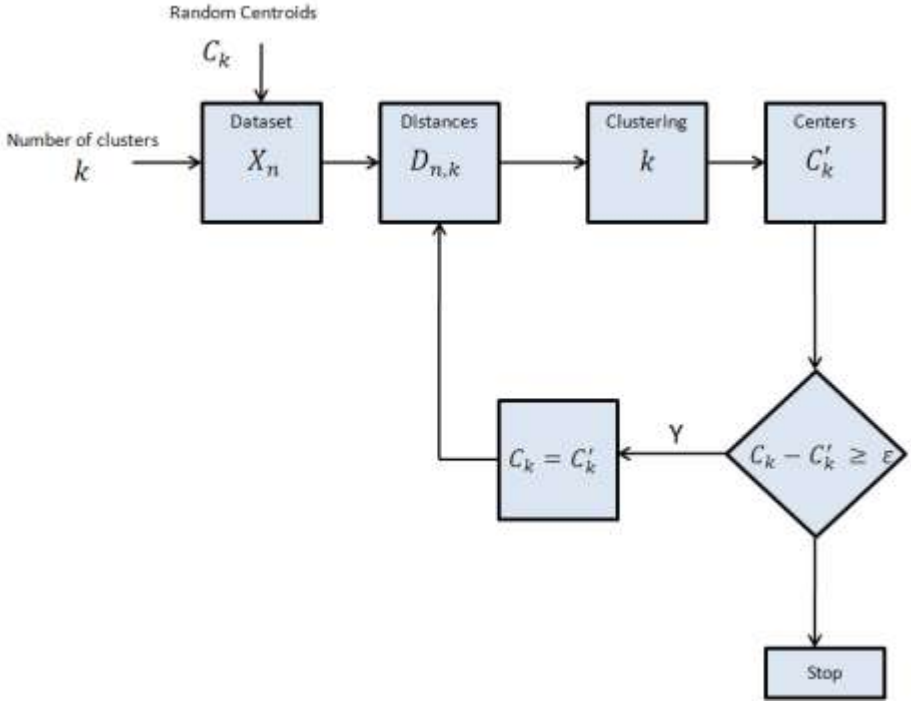


Figure 2.2: Schematic diagram of normal *k*-means algorithm

2.2.1 DBSCAN clustering

DBSCAN [Ester, et al., 1996] is the density-based spatial clustering of applications with noise, which is defined as:

Point q is directly density-reachable from a point p if it is not farther away than a given distance ε, and if p is surrounded by sufficiently many points such that one may consider p and q a part of a cluster. q is density-reachable from p if there is a sequence $P_1, \dots, P_n$ of points with $p_1 = p$ and $p_n = p$ where each p_{i+1} is directly density-reachable from p_i.

One of the main advantages of DBSCAN is that we don't need to know the number of clusters in the data a priori, as opposed to *k*-means. DBSCAN can also find non-linearly separable clusters. The disadvantage of DBSCAN, on the other hand, is that it cannot cluster data sets well with large differences in densities.

The complexity of DBSCAN depends on an indexing structure which, when used, makes the overall run time complexity $\mathcal{O}(n \,.\log(n))$, where n is the number of data points. Without the use of an accelerating index structure, the run time complexity is $\mathcal{O}(n^2)$. The distance matrix of size $(n^2 - n)/2$ is often materialized to avoid distance recomputations. This, however, also needs $\mathcal{O}(n^2)$ memory. Figure 2.3 shows the running time complexity comparison of $\mathcal{O}(n^2)$ and $\mathcal{O}(n \,.\log(n))$ for the DBSCAN algorithm:

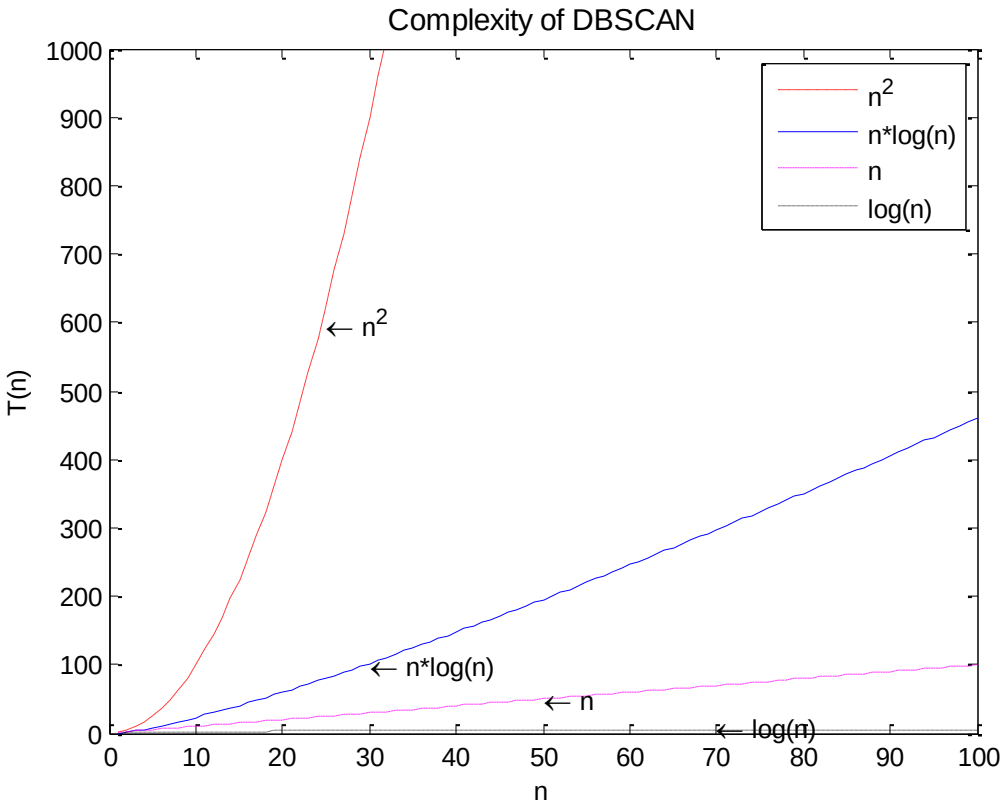


Figure 2.3: The run time complexity trends of a DBSCAN algorithm without indexing is $\mathcal{O}(n^2)$ and with it is $\mathcal{O}(n \,.\log\,(n))$

Thus the DBSCAN is differ from the normal *k*-means algorithm because its time complexity is quadratic while for the *k*-means it is linear (see section 3.1) in terms of a number of data points. Another difference is that the *k*-means algorithm requires an estimate of the number of clusters before the starting of the algorithm.

2.3 Regression as classification

A novel approach of solving regression problems as the multiclass classification tasks within the common framework of kernel machines is proposed. It has been shown in

[Kecman and Yang 2009], how one can solve regression problems posing them as the multiclass classification tasks. Good results of a multiclass approach averaged over twelve benchmarking regression data sets have been presented. The multiclass classifier used in that paper was the Adaptive Local Hyperplane (ALH) introduced in [Yang and Kecman 2008]. The choice of the ALH was natural because in [Yang and Kecman 2008], it showed extremely good results on eleven standard benchmarking classification data sets, outperforming seven other machine learning tools including KNN classifier, SVMs and K-local hyperplane (linear manifold) distance nearest neighbor (HKNN) algorithm. The results presented in [Kecman and Yang 2009] were a big encouragement and provided a strong motivation for exploring the approach of transforming regression problems into the classification ones within a much stricter experimental approach.

The topic here is to investigate the very characteristics of the proposed approach and to do a strict comparison of a nonlinear (NL) regressor and an NL multiclass classifier in solving regression problems. In order to do fair comparisons, SVMs are used for solving both tasks - regression and multiclass classification. The easily available and most popular benchmarking SVM tool, LibSVM [Hsu and Lin, 2002], was used in all experiments.

As for the model comparison environment, the double (nested) cross-validation was implemented. Note that all results presented in [Kecman and Yang 2009] have been obtained by using a standard single (10-fold) cross-validation. This was a fair approach for validating whether the idea of transforming regression problems into the classification tasks is feasible at all. However, such an approach is not quite valid for comparing different models because in a single cross-validation all the data points available are used for

both hyper-parameter determination and accuracy estimation. While such a procedure is good for estimating the hyperparameters of a single model it does not produce a true estimate of the accuracy on the given data set. If the main goal is to compare different ML models on the same data sets and under same conditions, double cross-validation must be used [Yang and Kecman 2010].

Double cross-validation is a very rigorous scheme for assessing a model's performance [Yang and Kecman 2010]. Here, we evaluate the generalization performance of the SVM regressor and SVM multiclass classifiers by using the double cross-validation procedure. The double cross-validation procedure is structured as the two loop algorithm. In the outer loop, the data set is separated into J_1 roughly equal-sized parts (here, $J_1 = 10$). Each part is held out in turn as the test set, and the remaining 9 parts are used as the training set. In the inner loop, J_2-fold cross-validation is performed over the training set only to determine the best values of hyper-parameters (here, $J_2 = 10$). The best model obtained in the inner-loop is then applied on the test set. The double cross-validation procedure ensures that the class labels of the test data won't be seen when tuning the hyper-parameters, which is consistent with the real-world scenario. Obviously such a rigorous procedure is done in many runs. In our case here, for the Gaussian kernel SVMs and when solving a regression problem which has three hyperparameters (penalty parameter C, shape parameter γ, and the size of ε tube), the number of iteration runs increases very quickly as follows: suppose we want to select the best parameter out of 5 predefined ones for each hyperparameter of a SVM regressor. This amounts to $10 \times 10 \times 5 \times 5 \times 5 = 12{,}500$ runs over the data set. All in the field of ML may well be aware that giving only 5 values for the hyperparameters is

usually a scarce approach. In practice, much bigger sets of values for each hyperparameter are typically used in order to find the best one, and this then leads to the significant training time. In this dissertation, the training times for SVM regressors and SVM multiclass classifiers are also compared.

The results in solving twelve benchmarking regression tasks shown here will present SVM regression and classification algorithms as strongly competing models where each approach shows merits for a specific class of high-dimensional function approximation problems.

2.4 Contributions of the dissertation

The dissertation has two major contributions. First, it introduces an improvement (speed up) to the *k*-means algorithm clustering algorithm and second, it develops an approach of solving regression problems as multiclass classification tasks by using SVMs within the double cross-validation experiment.

k-means algorithm is considered one of the most popular algorithms for clustering unsupervised data. Since the *k*-means algorithm depends mainly on distance calculation between all data points and the centers, the cost will be high when the size of the dataset is big (> 100,000 or 1 million points i.e., samples). The dissertation presents and develops a two stage algorithm, named *k*-means^2 algorithm, with the aim of reducing the time and memory costs of distance computation for huge datasets. The first stage is a fast calculation on a small portion of the dataset whereby the objective is to generate the centers for the following intensive and consequently slow final distance calculation process. The two stages, the initial fast stage and the final slow stage are, in essence, creating the trajectories

of the dynamic intermediate centers in a *d*-dimensional feature space. The cost of the centers' calculation in the fast stage (due to the smaller size of the portion of the dataset used) is much lower then if all the data had been used. Using all the data is in fact what the slow stage is doing while computing the exact terminal locations of the centers. However, because the slow stage starts from the clusters' centers obtained in the first (fast) phase the cost of the centers calculation in the slow stage will also be reduced [Salman and Kecman, 2011] and [Salman et al., a, b 2011]. In the dissertation, we present a speeding up of *k*-means clustering due to novel (algorithmic) approach. In [Li, Salman, Test, Strack and Kecman, 2011] and [Li, Salman, Test, Strack and Kecman, 2012] it was shown how one can handle large datasets by using novel hardware advancements (e.g. GPU based calculations).

The second contribution is in developing an experimental environment for comparisons of various SVMs models in solving regression tasks in two different manners- first as the classic SVMs regressors and second as the SVMs multiclass classifiers [Salman and Kecman, 2012]. In order to transform a regression problem into a classification task several discretization methods have been introduced and experimented with. The two best performing discretizations of a continuous output (target) vector **y** (fixed and varying ε-tube) are presented and compared. The *k*-means discretizations are also presented. A very strict double (nested) cross-validation technique has been used for measuring performances of regression and multiclass classification SVMs. The novel approach and experimental results obtained for twelve benchmarking regression data sets warrant both further theoretical investigations and broad application in practice.

CHAPTER 3: CLUSTERING USING THE 2-STAGE K-MEANS ALGORITHM, K-MEANS^2

The previous chapter introduced the concept of the *k*-means^2 algorithm for the clustering problem and the regression problem solving into multi class classification. This chapter will give a fuller account of the two contributions in this dissertation.

3.1 History of *k*-means algorithm

To our knowledge very little has been reported on the running time required for the k-means algorithm to achieve its goals, [Arthur and Vassilvitskii, 2006]. By means of a seed center selection scheme based on specific probabilities they reduced the worst-case running time scenario as superpolynomial by improving the lower bound from $\Omega(n)$ to $2^{\Omega(\sqrt{n})}$, where n represents the number of points in the dataset. [Hodgson, 1988] has developed another method to reduce the number of iterations but it was not as fine-tuned as [Arthur and Vassilvitskii, 2006]. On the other hand, [Pakhira, 2009] has proven that the number of iterations required by *k*-means algorithm is much less than the number of points. Moreover, [Har-Peled and Sadri, 2005] were unable to bound the running time of *k*-means algorithm, but they proved that for every reclassified point one iteration is required. The time complexity of *k*-means is $\mathcal{O}(NkQ)$, where N is the number of input patterns, k is the number of clusters, and Q is the number of iterations.

[Pakhira, 2009] worked on modifying the *k*-means algorithm to avoid the empty clusters. Pakhira moved the center of every cluster into new locations to ensure that there

will be no empty clusters. The comparison between the modified *k*-means algorithm and the original *k*-means algorithm shows that the number of iterations is higher with the modified *k*-means algorithm method. In the case of the numerical examples which produce empty clusters, the proposed method of Pakhira cannot be compared with any other method since there is no modified *k*-means algorithm available that avoids the empty clusters (cluster with no label). [Bradley and Fayyad, 1998] developed a procedure in which the centers have to pass a refinement stage to generate good starting points. [Wu, 2008] used genetically guided *k*-means algorithm where the possibility of empty clusters will be treated in the mutation stage. Another method of center initializing based on values of attributes of the dataset is proposed by [Khan and Ahmed, 2004]. The latter proposed a method that creates a complex procedure which is computationally expensive.

[Elkan, 2003] developed a method to avoid unnecessary distance calculations by applying the triangle inequality in two different ways, and by keeping track of lower and upper bounds for distances between points and centers. This method is effective when the dimension is more than 1000 and also when the clusters are more than 20. They claimed that their method is many times faster than normal *k*-means algorithm method. In their method the number of distance calculations is n instead of kQn where n is the number of points.

The next section will present the basic concepts and algorithm of the proposed *k*-means^2 algorithm.

3.2 Clustering with *k*-means^2 algorithm: basic concepts

The concept of the *k*-means^2 algorithm in this dissertation is based on minimizing the huge computational effort normally incurred with the application of the *k*-means algo-

rithm. The problem with the *k*-means algorithm is that it carries out the distance computation based on the full dataset. The choice of the initial seeds or starting centers is usually arbitrary. This suggests that if the arbitrary locations are steered towards the final locations by some means with less computational effort then the final tuning of the locations can be carried out with much less computational effort. With our concept we utilize only a small portion of the full dataset in order to steer the arbitrary locations closer to the true final locations of the assumed clusters of the selected portion at a much faster speed of computation. Obviously, the speed advantage referred to is due to the difference in the size of the data used in the computation. We refer to this as the *fast stage*. Achieving better locations from this stage for the centers of the clusters within the small data portion will provide a closer distance to the actual final locations. The stopping criterion of the fast stage is naturally determined when the fast stages reach the centers of the small portion of the dataset. In reality, the final locations are actually nearer to the true locations of the centers of the clusters of the whole dataset. The second (final) stage is referred to as the *slow stage*. A graphical description of the *k*-means^2 algorithm is depicted in Figure 3.1 while the algorithm itself is presented underneath, [Salman et al., a&b, 2011]. It can be seen that the *k*-means algorithm is repeated twice in the *k*-means^2 algorithm, however with a different number of data points used in the two stages.

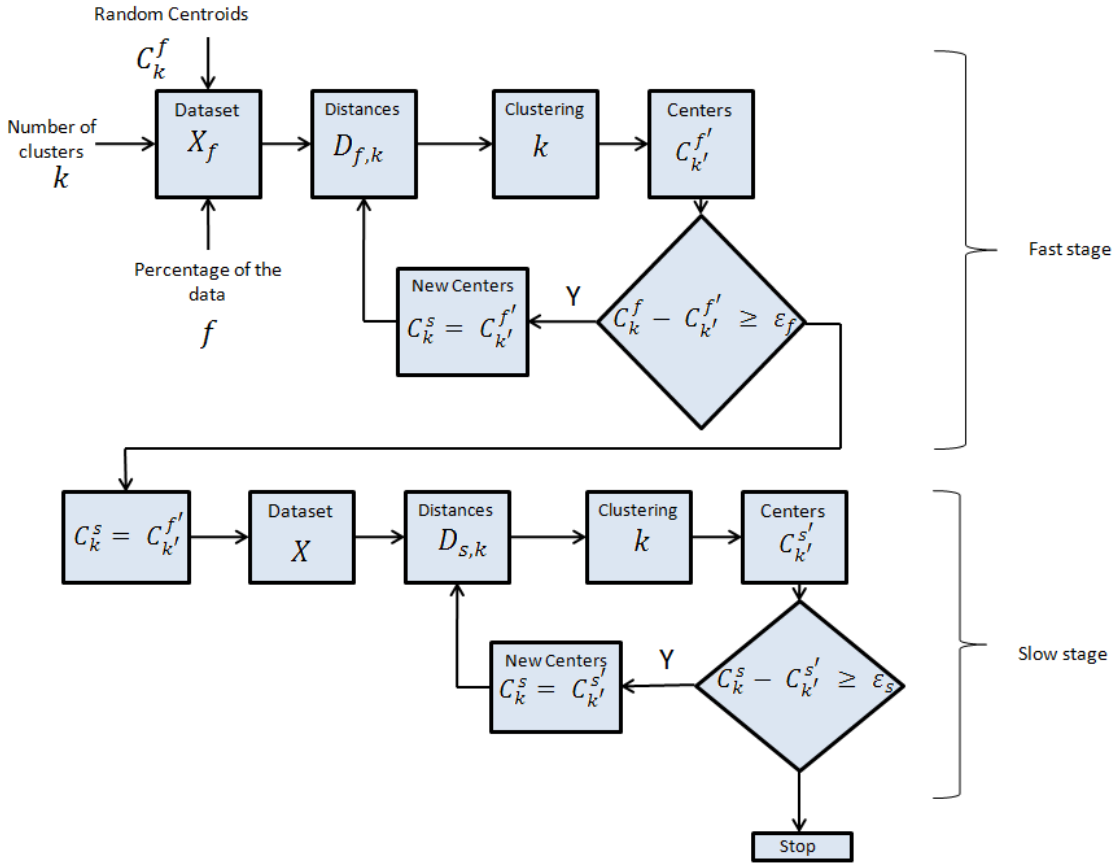


Figure 3.1: Graphical description of the *k*-means^2 algorithm

Algorithm 1: A 2-stage *k*-means^2 algorithm clustering

Input: $X_n, c, per\%, J_f, J_s$; where $J_f, J_s$are stopping criteria for both stages (fast and slow)

Output: $S_n$with clusters

$X_p = per\%$ of X_n

Select X_c from X_n randomly

Fast Stage

Use X_p

While $J_f \leq \mu_f$

For $k \leftarrow 1\ to\ k'$

For $i \leftarrow 1\ to\ n_p$

Calculate the distance

$$d(x_{k'}, x_c) = \left[(x_{k'} - x_{ci})^T (x_{k'} - x_{ci})\right]$$

End for

Find minimum of d

Assign the cluster number to point X_i

Calculate the mean of the clusters μ_f

End for

End while

Slow Stage

Use the whole dataset X_n

While $J_s \leq \mu_s$

For $k \leftarrow 1\ to\ k'$

For $i \leftarrow 1\ to\ n$

Calculate the distance

$$d\left(x_{k'}, x_{\mu_f}\right) = \left[\left(x_{k'} - x_{\mu_{fi}}\right)^T \left(x_{k'} - x_{\mu_{fi}}\right)\right]$$

End for

Find minimum of d

Assign the cluster number to point X_i

Calculate the mean of the clusters μ_s

End for

End while

Algorithm 3.1.

The following section will give an account of the theoretical validation of the proposed two stage *k*-means^2 algorithm.

3.3 Validation of *k*-means^2 algorithm

Lemma 2.1, [Kanungo et al., 2002], describes the calculation of the movement of the centers of the *k*-means algorithm. Since our approach is effectively a two round running of the

k-means algorithm, albeit with a different size of data, it is required to modify the above Lemma in order to accommodate the two stages. The movements of the centers in the fast stage will be presented in Lemma1.

Preliminaries:

Assume, $v \in \mathbb{R}^d$, let $\Delta(u, v)$ denote the squared Euclidean distance between these points, then:

$$\Delta(u, v) = \sum_{i=1}^{d} (u_i - v_i)^2 = (\boldsymbol{u} - \boldsymbol{v})^T \cdot (\boldsymbol{u} - \boldsymbol{v}) \tag{3.1}$$

where $\boldsymbol{u} \cdot \boldsymbol{v}$ denotes the dot product of vectors $\boldsymbol{u}$ and $\boldsymbol{v}$.

Also assume that $X \subset \mathbb{R}^d$, then the summation of the distances from all points of the set X to any point v is:

$$\Delta(X, v) = \sum_{u \in X} \Delta(u, v) \tag{3.2}$$

Let $\Delta(X', c')$ be the squared Euclidean distance between the points of the subset X' and the center c'.

Lemma 1 can be summarized as follows:

The summation of the distances from all points in the *fast-stage* of *k*-means[2] algorithm to any point is equivalent to the summation of the distances of all points in the *fast-stage* of *k*-means[2] algorithm to the centroids of these points plus the product of the number of points in the *fast-stage* and the summation of the distances between the centroids and any other point.

Lemma 1 Let X' be a subset of set X points in $\mathbb{R}^d$, let c' be the centroids of X'. Then, for any $c'' \in \mathbb{R}^d$

$$\Delta(X', c'') = \Delta(X', c') + |X'|\Delta(c', c'') .$$

Proof: Expand $\Delta(X', c'')$:

$$\Delta\left(X', c''\right) = \sum_{u \in X'} \Delta(u, c'') = \sum_{u \in X'} (u - c'') \cdot (u - c'') \tag{3.3}$$

$$= \sum_{u \in X'} \left((u - c') + (c' - c'')\right) \cdot \left((u - c') + (c' - c'')\right) \tag{3.4}$$

$$= \sum_{u \in X'} \left((u - c') \cdot (u - c')\right) + 2\left((u - c') \cdot (c' - c'')\right) + \left((c' - c'') \cdot (c' - c'')\right) \tag{3.5}$$

$$= \Delta\left(X', c'\right) + 2\left((c' - c'') \cdot \sum_{u \in X'} (u - c')\right) + \left|X'\right| \left((c' - c'') \cdot (c' - c'')\right) \tag{3.6}$$

$$= \Delta\left(X', c'\right) + \left|X'\right| \Delta(c', c'') \tag{3.7}$$

This is true if the center of X' is $c', making\ \sum_{u \in X'} (u - c')$ the zero vector.

The movements of the centers in the *slow stage* are presented in Lemma 2.

Lemma 2 can be summarized as follows:

The summation of the distances from all points in the *slow-stage* of *k*-means[2] algorithm to the centroids of the *fast-stage* is equivalent to the summation of the distances of all points in the *slow-stage* of *k*-means[2] algorithm to the centroids of these points plus the product of the number of points in the *slow-stage* and the summation of the distances between the centroids of the *fast-stage* and the centroids of the *slow-stage*.

Lemma 2 Let X be a set of points in $\mathbb{R}^d$, let c be the centers of X. Then, for $c' \in \mathbb{R}^d$

$$\Delta(X, c') = \Delta(X, c) + |X|\Delta(c, c').$$

Proof: Expand $\Delta(X, c')$:

$$\Delta(X, c') = \sum_{u \in X} \Delta(u, c') = \sum_{u \in X} (u - c') \cdot (u - c') \tag{3.8}$$

$$= \sum_{u \in X} \left((u - c) + (c - c')\right) \cdot \left((u - c) + (c - c')\right) \tag{3.9}$$

$$= \sum_{u \in X} \left((u - c) \cdot (u - c)\right) + 2 \left((u - c) \cdot \left(c - c^{'}\right)\right) + \left(\left(c - c^{'}\right) \cdot \left(c - c^{'}\right)\right) \tag{3.10}$$

$$= \Delta(X, c) + 2 \left(\left(c - c^{'}\right) \cdot \sum_{u \in X} (u - c)\right) + |X| \left(\left(c - c^{'}\right) \cdot \left(c - c^{'}\right)\right) \tag{3.11}$$

$$= \Delta(X, c) + |X| \, \Delta\left(c, c^{'}\right) \tag{3.12}$$

This is true if the center of X is c then $\sum_{u \in X}(u - c)$ is the zero vector.

Using the upper derivations of Lemma 2, Theorem 1 below can be summarized as follows:

The centers of the subset of the data will approach the centers of the whole dataset if the subset has the same number of clusters as the number of clusters in all data.

Theorem 1

Let $\boldsymbol{X}$ be a dataset with a set of clusters represented by the vector $\boldsymbol{S} = (S_1, S_2, \dots, S_k)$ having centers represented by the vector $\boldsymbol{c} = (c_1, \dots, c_k)$ and let $\boldsymbol{X}_f \subset \boldsymbol{X}$ be a proper subset of the dataset with the condition that its cluster vector is also $\boldsymbol{S} = (S_1, S_2, \dots, S_k)$ with the vector $\boldsymbol{c}^f = \left(c_1^f, \dots, c_k^f\right)$ representing their centers. Then the subsequent applications of the *k*-means algorithm on $\boldsymbol{X}_f$ would shift the centers $\boldsymbol{c}^f = \left(c_1^f, \dots, c_k^f\right)$ until they approach $\boldsymbol{c} = (c_1, \dots, c_k)$. Therefore if $\boldsymbol{X}_f \Rightarrow \boldsymbol{X}$ then the subsequent applications of the *k*-means algorithm on $\boldsymbol{X}_f$ would shift the centers $\boldsymbol{c}^f = \left(c_1^f, \dots, c_k^f\right)$ until they coincide with

$\boldsymbol{c} = (c_1, \dots, c_k)$, or $c_1^f\left(S_1^f\right), \dots, c_k^f\left(S_k^f\right) \Rightarrow c_1(S_1), \dots, c_k(S_k)$

Proof:

We assume that the whole data set is $\boldsymbol{X}$ of size n and $\boldsymbol{X}_f$ is its subset with size $f \leq n$, where f is an integer representing the number of data in the fast stage. Additionally, the

condition that all clusters $\boldsymbol{S} = (S_1, \dots, S_k)$ exist in $\boldsymbol{X}_f$ must be satisfied, in other words there is no empty cluster. This means that $mean(c_i^f) \neq mean(c_i)$ if $f \neq n$.

However, if $f \Rightarrow n$ then $mean(c_i^f) \Rightarrow mean(c_i)$

This completes the proof.

Continuing from the above Theorem, we can present the following corollary.

Corollary

The centers $c_1^f(S_1^f), \dots, c_k^f(S_k^f)$ produced at the last iteration of the *k*-means by the *fast stage* can be used as the seeds for $\boldsymbol{X}$. Therefore, there is no loss of generality since the centers produced from the fast stage $c_1^f(S_1^f), \dots, c_k^f(S_k^f)$ are located on the path of convergence of the centers for each cluster.

Then the final iteration of the *k*-means would shift $c_1^f(S_1^f), \dots, c_k^f(S_k^f)$ such that they coincide with $c_1, \dots, c_k$. This concludes the proof of the corollary.

Figure 3.2 demonstrates how the distance between the center obtained from all the data c_1 and the center calculated from the fraction of data c_1^f changes depending upon the size of the fraction of original data set used in the fast phase (a toy dataset is used in Figure 3.2). When less than 10% of the dataset is used in the first stage, and since the condition of existence of the same clusters in the subset is not fulfilled, the error index represented by the y-axis ($c_1^f(S_1^f) - c_1(S_1)$) exhibits large values. As the size of the data sets used in the first stage increases, the difference (error) between c_1 and c_1^f asymptotically approaches zero.

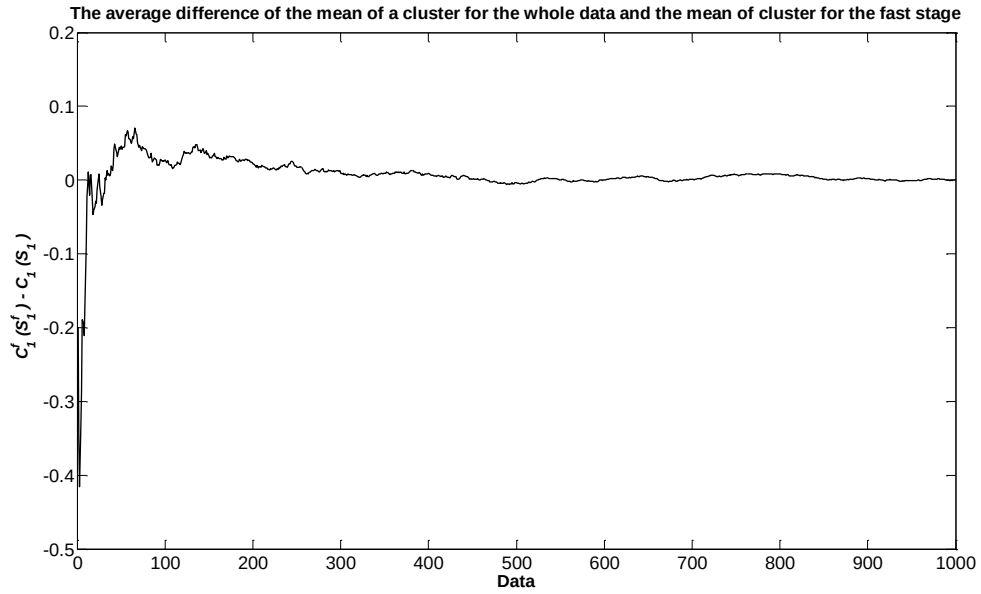


Figure 3.2: Differences ($c_1^f\left(S_1^f\right) - c_1(S_1)$) between the cluster center (for center 1) and the fast stage cluster center are dependent on the size of the data fraction used in the first stage

As mentioned before, the worst-case running time of *k*-means algorithm is a superpolynomial with a lower bound of $2^{\Omega(\sqrt{n})}$ iterations, where n represents the number of points in the dataset. Thus, the lower bound of the number of iterations for a classic *k*-means algorithm working with all *n* data points will be much higher than the lower bound of the fast stage in the worst case scenario, which is $2^{\Omega(\sqrt{n_f})}$ because $n_f << n$. Note that the second, slow stage, works with *n* data points again but not under the worst case scenario because the final centers of the *k* clusters are 'very' close to their terminal locations. On the other hand, as Figure 3.2 indicates, using a very small fraction of the data in the fast stage (as mentioned above below 10%) makes the slow stage fall back into the worst case sce-

nario. In this case $2^{\Omega(\sqrt{n})} < 2^{\Omega(\sqrt{n_f})} + 2^{\Omega(\sqrt{n})}$ and the *k*-means^2 algorithm will need more iterations that the *k*-means method. A general case of a distribution-free *k*-means clustering proving the complexity in a non worst case scenario is extremely difficult. However, there is a result in (Arthur and Vassilvitskii, 2006) which states that "Given data points chosen from independent normal distributions with variance s^2 and with dimension $d = \Omega(n/\log(n))$, *k*-means algorithm will execute in polynomial time with high probability". The experimental results that follow for the distribution-free *k*-means clustering show excellent agreement with the claim made for normally distributed data.

The following section will give selected numerical examples to highlight the advantages of the proposed *k*-means^2 algorithm.

3.4 Numerical examples

To investigate the suitability of the proposed method we run the simulation for many different parameters. The parameters which have to be adjusted to get the best speed up values are: the data size, the dimension of the data, the number of clusters, the stopping criteria of the first clustering stage, the stopping criteria for the second clustering stage and the percentage of the data used for the 2-stage clustering. Another important consideration is the use of the same program for running the normal *k*-means clustering and the 2-stage clustering. The same seeds used for the fast stage in the case of the *k*-means^2 algorithm were used for the *k*-means algorithm. Furthermore, the same computer has been used for running all simulations to avoid discrepancy in computer performance. The computer used was Alienware with an i7 CPU and 6 GB of RAM. Two examples are shown below. The

first one points to the general characteristics of the two stage *k*-means algorithm in which the fast stage is much more dynamic than the second stage. Figures 3.3 and 3.4 show the changes of two coordinates of two-dimensional centers separately, which are just the representatives for all the coordinates of *d*-dimensional data points.

Example 1: A dataset of 800 samples and 2-dimension (3 clusters) is used. The following figures show the movement of one of the centers in the two stage clustering.

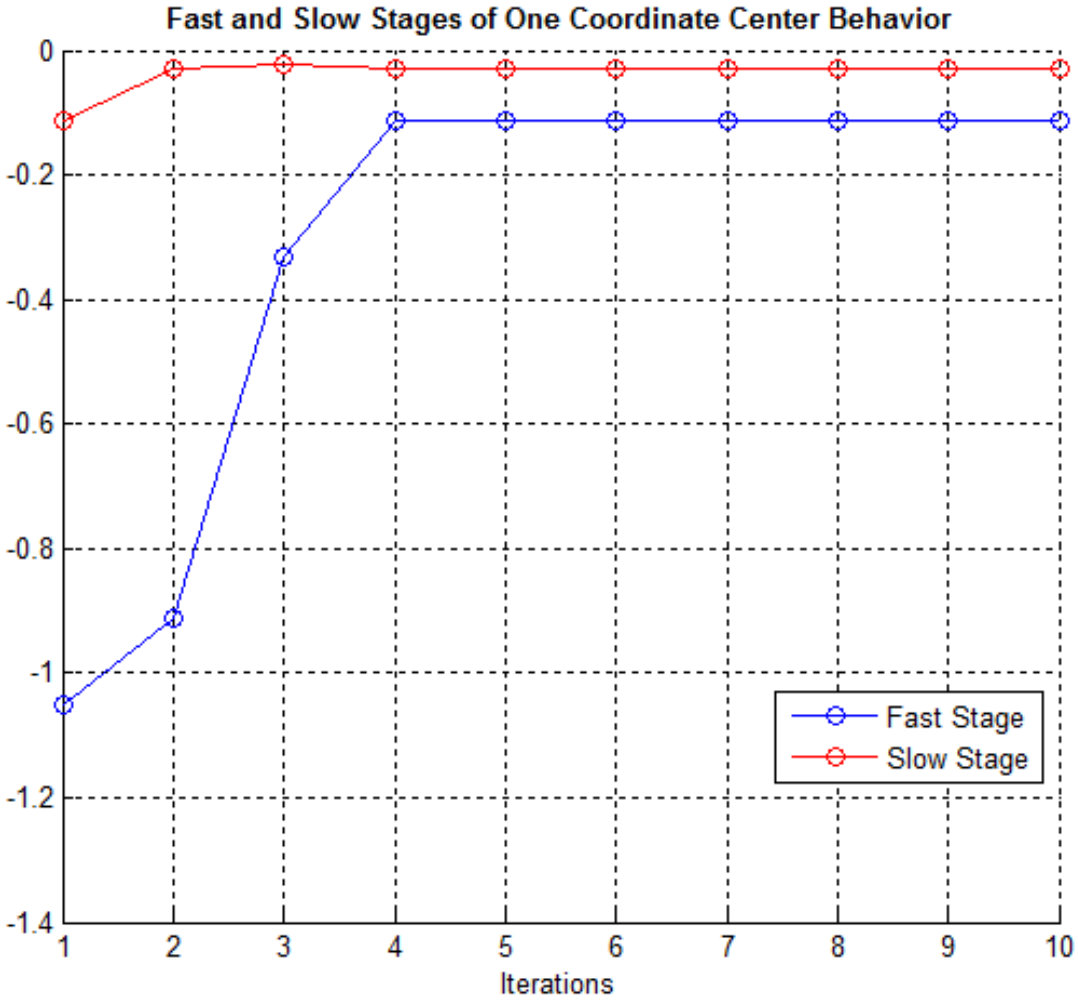


Figure 3.3: The movements of the first coordinate of one center during *fast* and *slow* stages of *k*-means[2] algorithm.

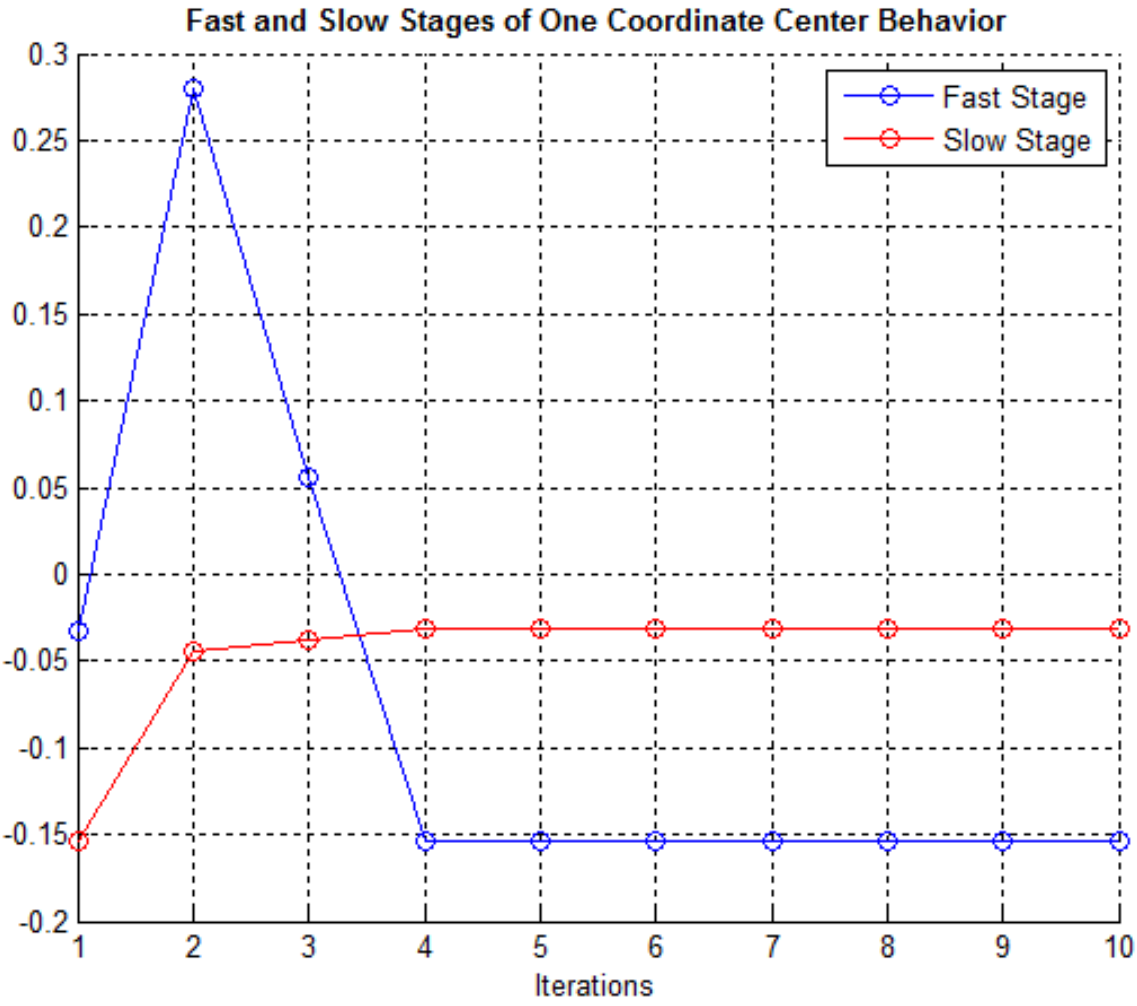


Figure 3.4: The movements of the second coordinate of one center during *fast* and *slow* stages of *k*-means[2] algorithm.

It is very clear from Figures 3.3 and 3.4 that the movement of the red line (the change of the center's first coordinate during the *slow stage*) is very smooth compared with the *fast stage* coordinate movements. The initial value of the red curve (*slow stage movement*) is the same as the terminal value of the blue (*fast*) curve. The number of iterations is higher than necessary purely for clarification purposes.

Moreover, as can be seen from the above graph, the coordinates have not changed much during the *slow stage*. This means that the second (*slow*) stage requires less iterations. Figure 3.5 shows the well known movements of the centers for all three clusters in the same dataset during the fast stage. It can be seen that the movements of the three clusters' centers (shown as colored squares, red, yellow and green) are volatile and they heavily depend on the random choice of the initial center seeds (shown as colored circles).

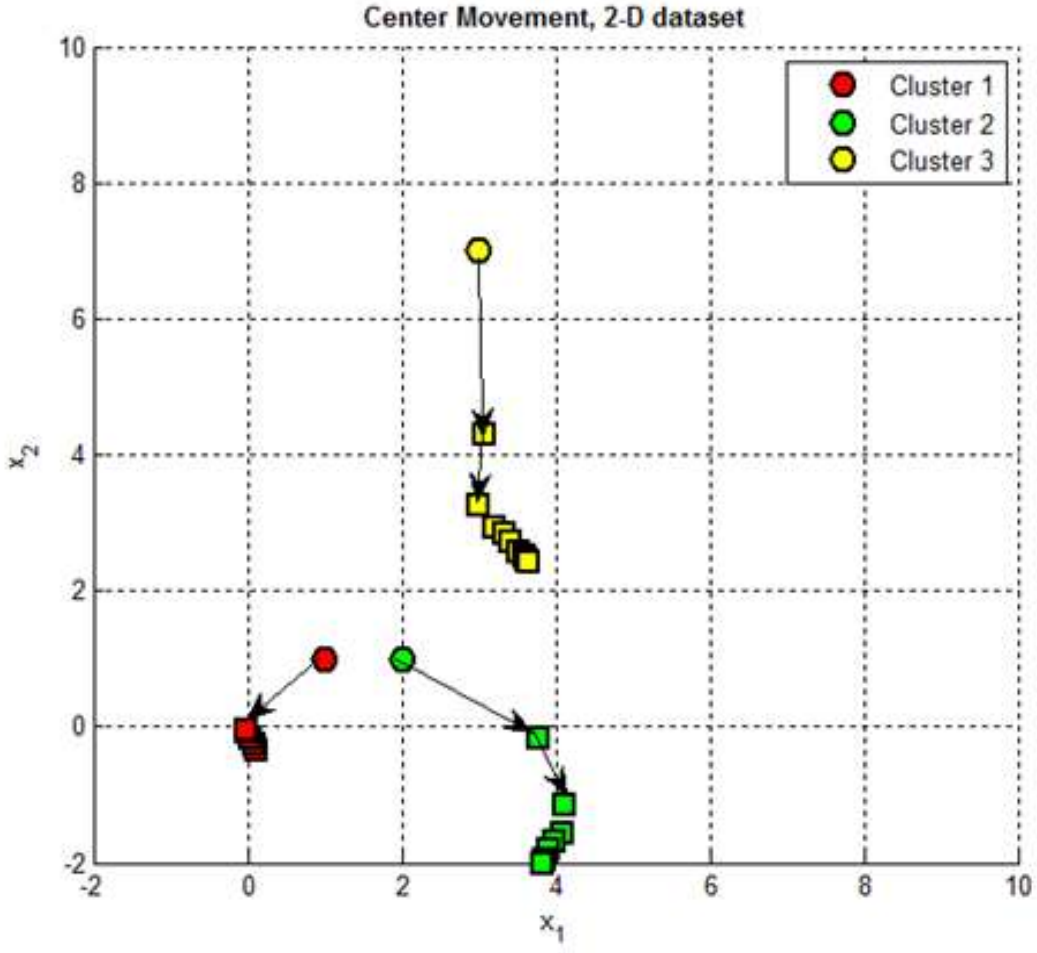


Figure 3.5: The centers movements for the three clusters during the *fast stage*.

The figures 3.6, 3.7 and 3.8 depict the excursions of the three clusters' centers in greater detail using different center seeds than in Figure 3.5 just in order to show the differences in the centers movement during the fast and slow stages. The *fast stage* centers are symbolized with squares while the *slow stage* centers are symbolized with circle shapes. It can be seen that the size of the steps during the fast stage are much bigger than the ones during the slow stage.

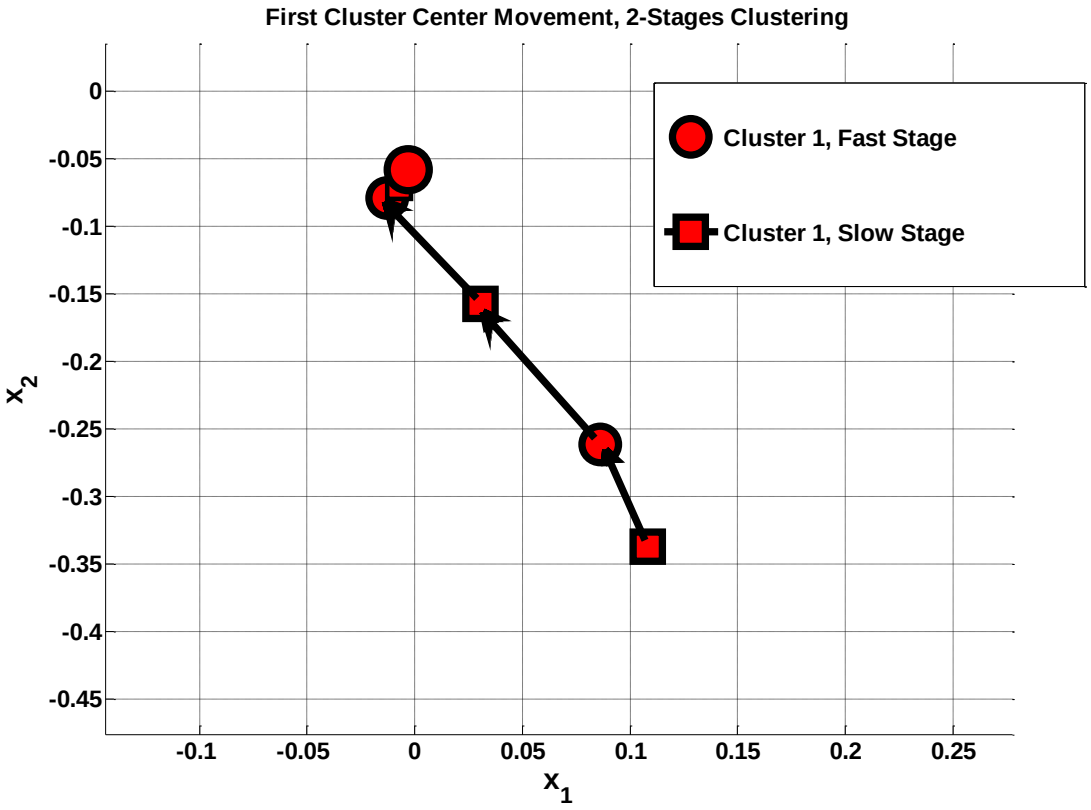


Figure 3.6: Movements of the first cluster centers during the fast (squares) and slow (circles) stages.

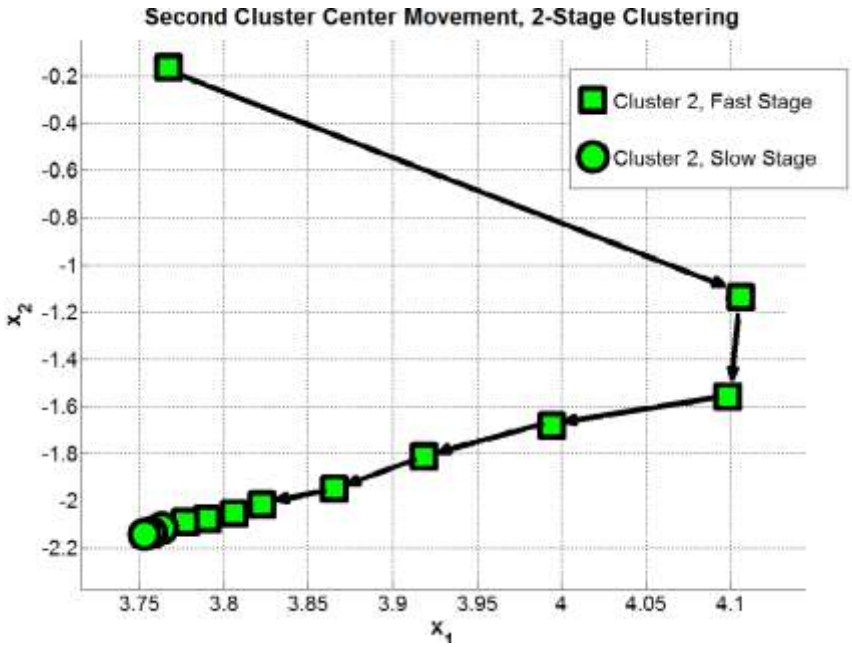


Figure 3.7: Movements of the second cluster centers during the fast (squares) and slow (circles) stages.

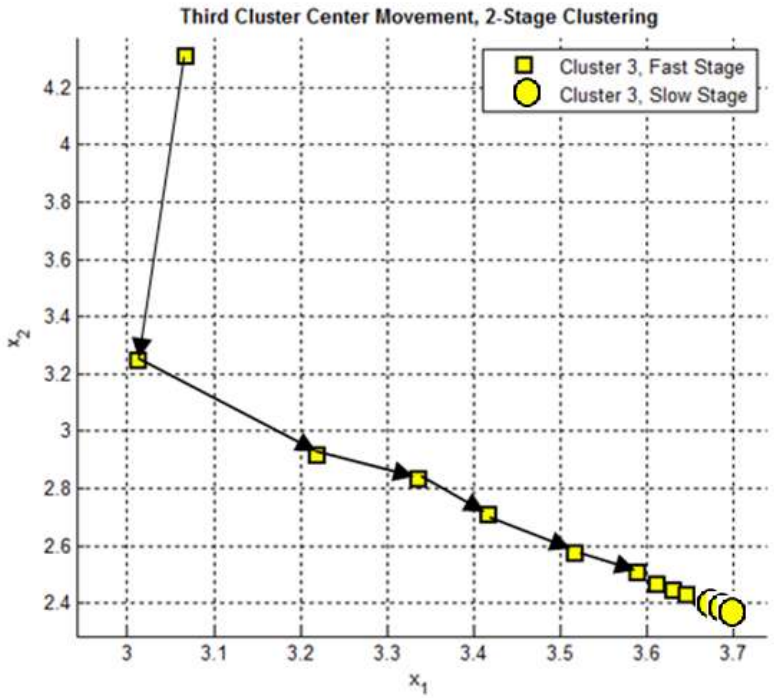


Figure 3.8: Movements of the third cluster centers during the fast (squares) and slow (circles) stages

As can be seen from the above diagrams, the centers usually make much bigger spatial displacements during the fast stage, which is expected to happen in a short time because just a fraction of data points is used in the fast stage. The number of iteration steps during the slow stage is much smaller than the number of iteration steps during the fast stage. The slow phase movements of the centers are also smaller with respect to the fast stage ones.

Example 2

Table 3.1 gives an example of a distribution of points and centers during the fast and slow stages of clustering in tabular form for a real dataset (iris) of two clusters.

Table 3.1: Distribution of points and centers during the fast and the slow stages of clustering (iris)

Stages	Iter No.	Clusters	Old Centers Coordinates			New Centers Coordinates			Points in Clusters	Points
Fast	1	C1	8	4	4	4.867	3.267	1.567	30,38,44	15
		C2	4	4	4	6.16	2.85	4.68	53,58,64 69,72,86 88,93,113 114,138,145	
	2	C1	4.867	3.267	1.567	4.867	3.267	1.567	30,38,44	
		C2	6.16	2.85	4.68	6.16	2.85	4.68	53,58,64 69,72,86 88,93,113 114,138,145	
Slow	1	C1	4.867	3.267	1.567	5.015	3.318	1.636	1-50,58 61,82,94 99	150
		C2	6.16	2.85	4.68	6.323	2.901	4.987	51-57,59-60 62-81,83,93 95-98, 100-150	
	2	C1	5.015	3.318	1.636	5.06	3.226	1.897	1-50,58 61,82,94 99	
		C2	6.323	2.901	4.987	6.396	2.933	5.071	51-57,59-60 62-81,83,93 95-98, 100-150	
	3	C1	5.06	3.226	1.897	5.083	3.205	1.956	1-50,58 61,82,94 99	
		C2	6.396	2.933	5.071	6.409	2.942	5.1	51-57,59-60 62-81,83,93 95-98, 100-150	
	4	C1	5.083	3.205	1.956	5.083	3.205	1.956	1-50,58 61,82,94 99	
		C2	6.409	2.942	5.1	6.409	2.942	5.1	51-57,59-60 62-81,83,93 95-98, 100-150	

Note that in Table 3.1 the fast stage uses only 10% of the whole dataset. The initial centers of the slow stage are the same as the terminal centers reached by the fast stage, and they are shown in different colors. To show that the proposed method of the *k*-means^2 clustering has the same speed up for the real datasets as for the toy ones, twelve datasets taken from the machine learning repositories sites were used as shown in the next section.

3.4.1 Real datasets analysis

For the verification of the *k*-means^2 algorithm, a range of real data sets with different portions of the data selected for the fast stage of the algorithm have been used as shown in the following table:

Table 3.2: Speed up of the *k*-means^2 algorithm for twelve real datasets with different sizes

Datasets	**optdigits**	**satimage**	**usps**	**pendigits**	**reuters**	**letter**	**adult**	**w3a**	**shuttle**	**web**	**mnist**	**ijcnn1**	
# Points	5,620	6,435	9,298	10,992	11,069	20,000	48,842	49,749	58,000	64,700	70,000	141,691	
# Classes	10	6	10	10	2	26	2	2	7	2	10	2	
Dimensions	64	36	25	16	8315	16	123	300	7	300	780	22	
% Data portion	**Speed Up for 20 runs (X)**												**Data portion average**
1	1.1581	1.1226	1.4879	1.4565	1.2847	1.1957	2.5204	1.4309	1.4746	1.6446	1.225	1.8592	**1.488**
2	1.1047	1.0675	1.0209	1.5644	1.444	1.2251	2.0893	1.4547	1.5111	1.712	1.1172	2.1865	**1.458**
3	1.2861	1.3971	1.18	1.2237	1.3542	1.2988	2.4677	1.7631	1.8638	1.7229	1.6599	2.0942	**1.609**
4	1.3517	1.3553	1.2829	1.2962	1.1304	1.2225	1.9305	1.6973	1.6104	2.0788	1.6282	2.0977	**1.557**
5	1.5508	1.7986	1.6947	1.4512	1.5092	1.2006	2.3747	1.9858	1.8249	1.9726	1.762	2.4674	**1.799**
6	1.1658	1.936	1.5365	1.6757	2.0174	1.2326	2.6972	2.1093	1.2961	1.9964	1.1256	2.3228	**1.759**
7	1.3833	1.7182	1.4242	1.3561	2.0206	1.3693	2.2295	2.1643	2.2255	2.0637	1.7479	2.3057	**1.834**
8	1.5418	1.6612	1.2283	1.954	2.6366	1.2323	2.3065	2.0174	1.4138	2.321	1.7997	2.2425	**1.863**
9	1.2878	1.8202	1.5817	1.5237	2.6128	1.2285	2.4513	2.2109	1.648	2.1975	1.801	2.3281	**1.891**
10	1.4509	1.6682	1.3311	1.6182	2.0486	1.204	1.8321	2.0464	1.633	2.2008	1.4204	1.9062	**1.697**
20	1.4412	1.4152	1.3062	1.4822	1.6809	1.1922	1.6422	2.1255	1.5742	1.9713	1.5983	1.8021	**1.603**
30	1.576	1.6876	1.3967	1.5699	2.0091	1.2417	1.5861	1.7143	1.5622	1.6853	1.4175	1.606	**1.588**
40	1.7409	1.3368	1.7523	1.517	1.6285	1.1543	1.1942	1.5787	1.2016	1.7111	1.4792	1.5893	**1.490**
50	1.4018	1.2267	1.5538	1.5061	1.4166	1.1934	1.1209	1.4957	1.0407	1.3833	1.2414	1.3588	**1.328**
60	1.2684	1.0495	1.1311	1.229	1.6847	1.129	1.0928	1.3232	1.1452	1.3612	1.8012	1.2398	**1.288**
70	1.3315	1.1919	1.1733	1.1123	1.0947	1.2016	0.9328	1.242	1.0421	1.1856	1.265	1.2941	**1.172**
Speed up average	**1.378**	**1.466**	**1.380**	**1.471**	**1.723**	**1.220**	**1.904**	**1.772**	**1.504**	**1.826**	**1.506**	**1.919**	

Twelve different real datasets have been used and shown in Table 3.2 with different sizes, demonstrating the success of the *k*-means^2 algorithm. The range of the number of points in

the datasets is from 5k to 140k, while the range of the number of classes is from 2 to 26. On the other hand, the dimensions of the datasets range from 7 to 8315. Different portions of the datasets were used in the first stage of the clustering ranges from 1% to 70%. 20 runs were chosen from the *k*-means[2] algorithm and an average of these runs was recorded. At the end of all runs of all portions of the datasets one average was calculated. Two types of averages have been reported in Table 3.2; the first is the speed up average for each dataset (last column) and the second is the data portion average for each dataset (last row). It is clear that the speed up averages increases with the increase of the size of the dataset. However, the data portion averages follow a distinct pattern which starts from low averages, increases for the 8-10% data portion, and decreases with higher percentages. Two main graphs were developed from the above table as shown below:

The following graph shows the speed up of all 12 datasets with percentage of the portion of the data used:

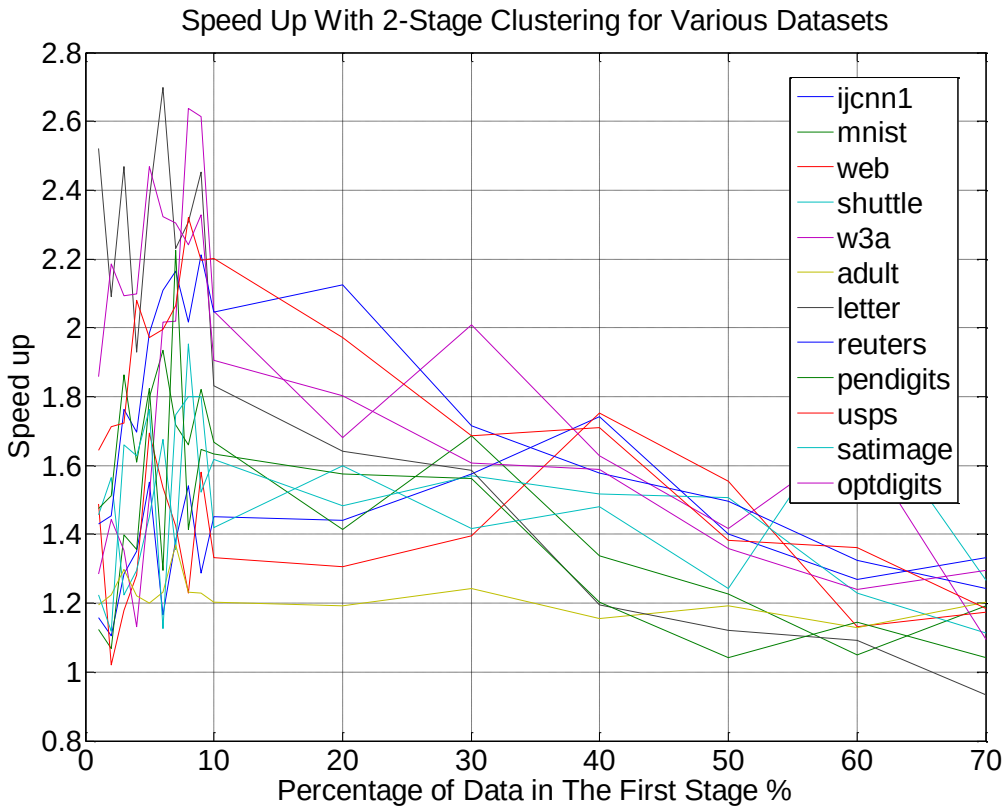


Figure 3.9: Speed up versus percentage of the data used during the fast stage of *k*-means[2] algorithm

It is clear that the use of less than 8-10% of the data in the *k*-means^2 algorithm produces better speed ups. This trend of speed up is applicable for all datasets used regardless of the size, dimension and number of clusters.

The following, Figure 3.10, however, shows the increase of the speed up due to the increase of the data size. The blue line shows the linear estimate of the increase in the speed up.

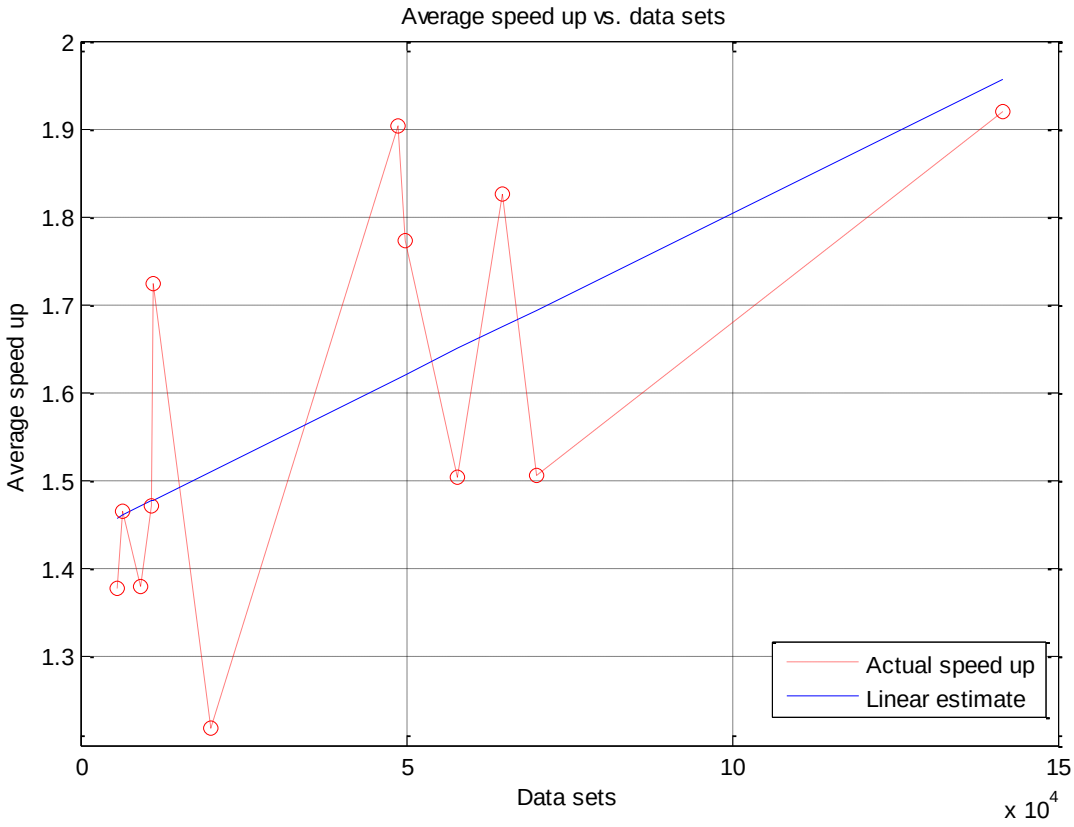


Figure 3.10: Average speed up versus datasets using *k*-mean^2 algorithm

The following graph, Figure 3.11 shows the average speed up with the percentage of the data used. It is clear that 8-9% is the best estimate of the portion of the data used for the first stage of *k*-means^2 algorithm.

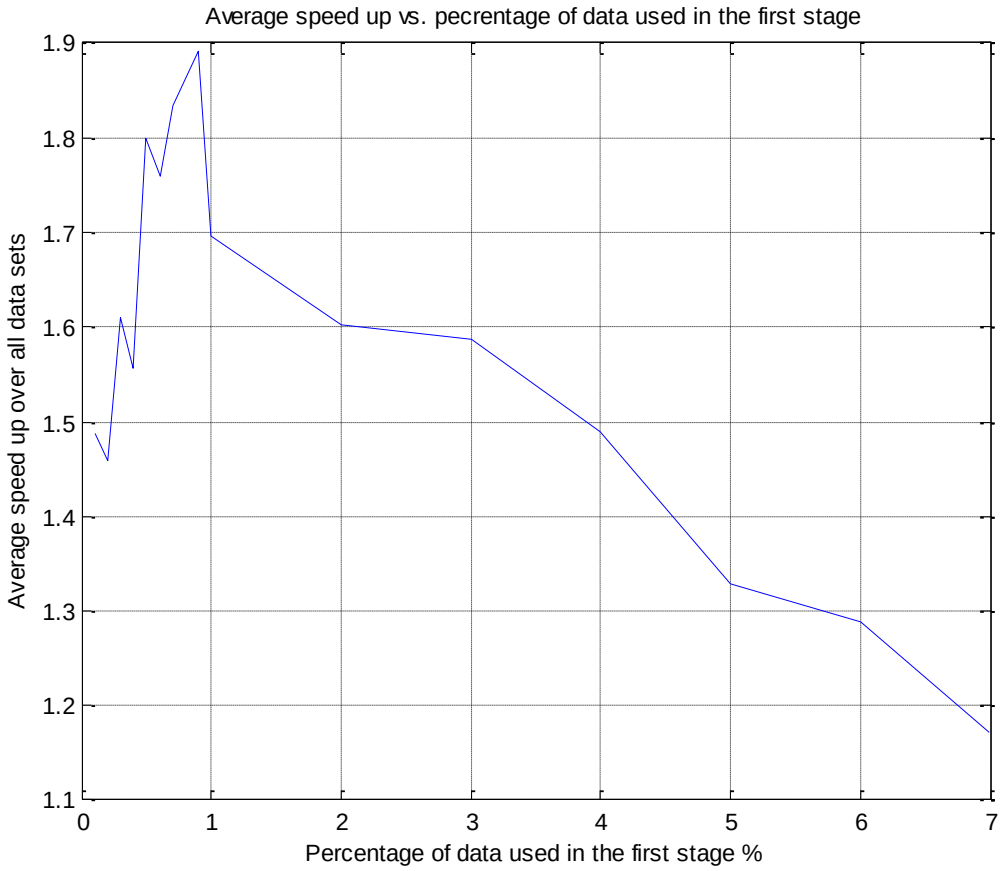


Figure 3.11: Average speed up versus percentage of the data used in the fast stage of *k*-mean^2 algorithm

3.5 Speed up analysis

This section will demonstrate the difference in the computational time exerted by the two methods, the *k*-means and the proposed modified version, the *k*-means^2, in moving from the same set of center seeds to the final set of centers with different stopping criterion and different operating systems. Two different computers with the same CPU speed were used, one running under a 32bit Operating System and the second running under a 64bit Operating System. The operating system will reflect the speed of the computation of the

computer, in which case the 64-bit machine is a faster computer. The results, as depicted in Figure 3.9, show that regardless of the Operating System used the modified *k*-means algorithm has demonstrated a speed up at all ranges of the stopping criterion. The data used for the fast stage clustering is only 10% randomly selected data. The dataset used in this example is "Synthetic", picked up from UCI [Frank and Asuncion, 2010] consisting of 100,000 samples in 10 dimensions and 4 clusters. The speed of the modified *k*-means^2 algorithm is almost twice the speed of normal *k*-means algorithm and it can be attributed to the fact that the *k*-means^2 clustering uses much less data points during the fast clustering stage leading to much less computational effort in the first stage of the process. The speed up is very clear in the high accuracy range when $\mu < 10^{-4}$. The stopping criterion, μ , is defined by $\mu = \delta(c_{final}, c_{final-1})$, i.e. as the difference in the distance between the final center and the preceding one. Depending on the accuracy requirements the range of the metric, μ , is to be determined by the user. In this example the range of μ is selected between 10^{-1} and 10^{-10} as shown in the Figure 3.12.

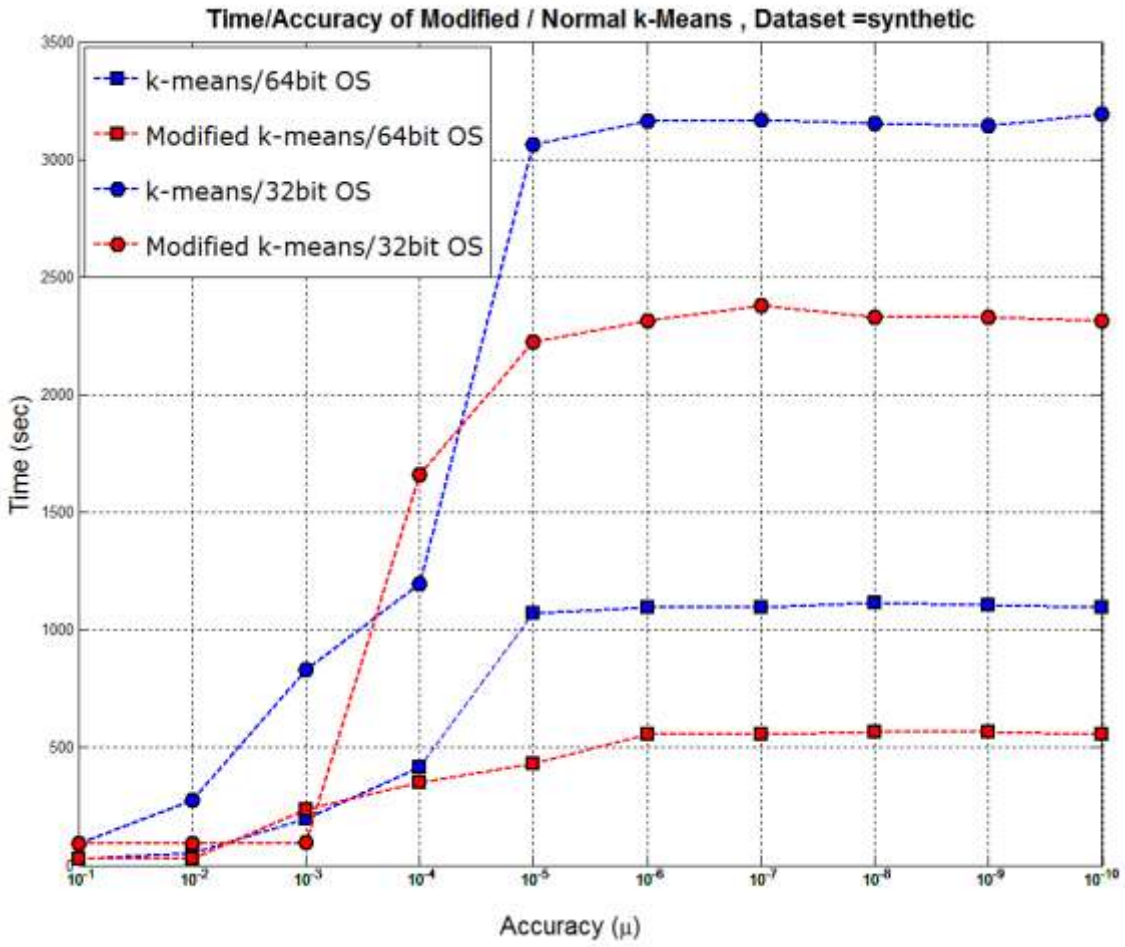


Figure 3.12: Comparison of the computational times between the *k*-means and the *k*-means^2 algorithms for 32 bit and 64 bit computers.

The speed of the normal *k*-means algorithm is shown in blue while the speed of the *k*-means^2 algorithm is shown in red. For example, the CPU time taken for the *k*-means algorithm when μ is 10^{-5} is 1100sec for 64bit OS and 3100sec for 32bit OS while the time for the *k*-means^2 is 400sec and 2250 sec respectively. This particular value of μ gives an advantage greater than 275% in favor of the *k*-means^2 on a 64 bit machine.

Next, we discuss the speed up achieved by using *k*-means^2 algorithm for the same 100,000 10-dimensional data points that should be split into 4 clusters. The speed up of the *k*-means^2 algorithm compared with the normal *k*-means algorithm varies according to the stopping criterion used (i.e., required accuracy) as shown in Table 3.3 and Figure 3.13.

Table 3.3: *k*-means^2 vs. *k*-means algorithms speed up for different stopping criterion and fraction of data used in the fast stage

Speed up of *k*-means^2 for different stopping criterion

	Percentage of the total data for the fast stage				
Stopping	**10%**	**15%**	**20%**	**30%**	**40%**
10^{-1}	1.9	1.8	1.8	1.7	1.5
10^{-2}	3.8	3.5	3.4	3	2.5
10^{-3}	4.7	8.9	3.1	7	4.3
10^{-4}	0.9	1.7	1.1	3	8.5
10^{-5}	2.9	1.6	2.2	2.1	2.4
10^{-6}	2	1.9	2.6	2.3	2.4
10^{-7}	2	1.4	2.4	2.3	1.6
10^{-8}	2	1.4	2.4	2.3	1.6
10^{-9}	2	1.4	2.4	2.3	1.6
10^{-10}	2	1.4	2.4	2.3	1.6

A graphical representation of Table 3.3 is depicted in Figure 3.13.

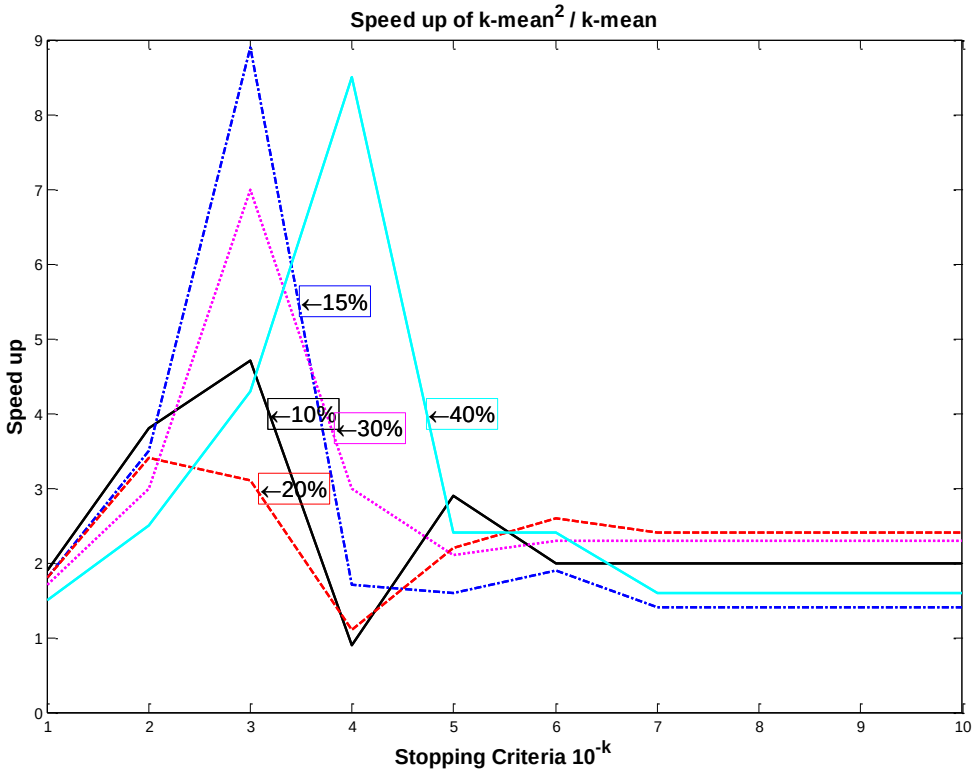


Figure 3.13: *k*-means^2 vs. *k*-means algorithms speed up for different both stopping criterion and fraction of data used in the fast stage

For lower ranges of accuracy 10^{-1} to 10^{-5}, the speed up of computation changes between 1 and 9 times. The speed up is reduced for higher accuracy, e.g., for μ from 10^{-5} to 10^{-10} , it is restricted to the range of $(1 - 2.5)$. However, the range of the random data selected to achieve the fast clustering (computation) actually fluctuates quite rapidly. It can be observed that the optimal range of sample data used in the fast stage of clustering falls between 10% - 20%. In the normal situation a choice of good accuracy range, e.g., (10^{-5} to 10^{-10}) will provide a speed up advantage up to 2.5, as shown in Table 3.3 and Figure 3.13.

The following section will give an account of the performance of the *k*-means^2 algorithm for large datasets.

3.6 Speed of computation of *k*-means and *k*-means^2 algorithms for large datasets

In this section we will deal with large toy datasets in order to find out the behavior of the two algorithms. Again the same machine will be used for a range of datasets 100k, 200k, 300k, 400k, 500k, 600k, 700k, 800k, 900k, 1M, 2M, 3M, 4M, 5M, and 6M with 12-dimensions and 4 clusters, where M stands for a million. We maintained the same value of the stopping criterion, i.e. 10^{-6}. In this case the portion of the data used in the first stage is reduced much further down to 0.1%. The results are shown in Figure 3.14.

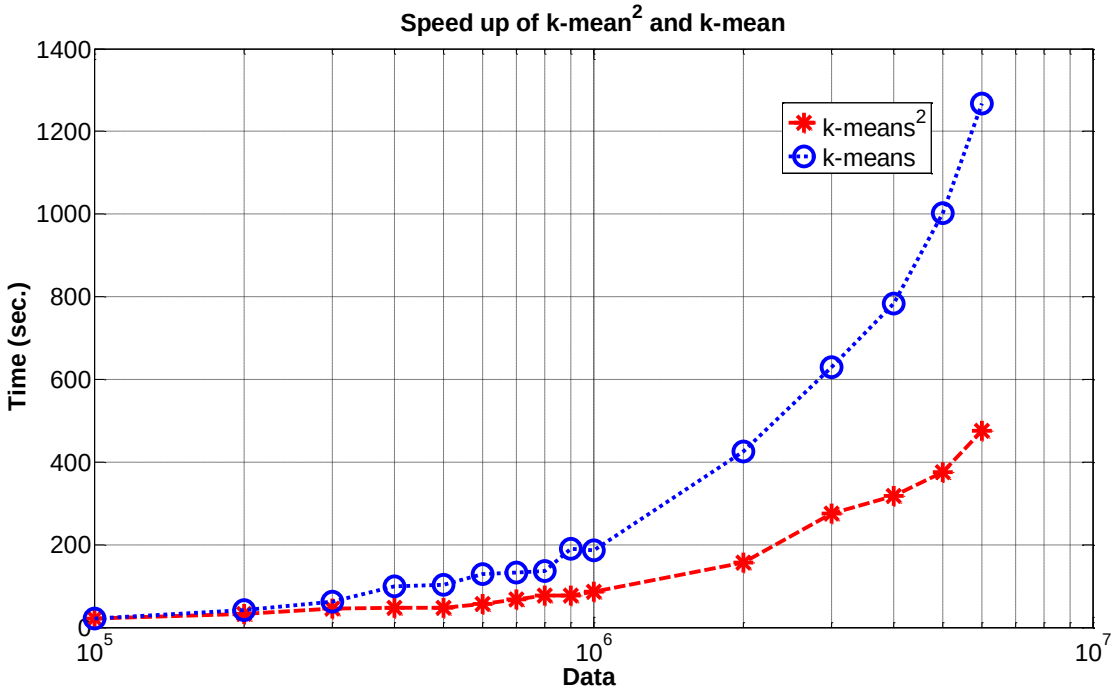


Figure 3.14: The CPU times for normal *k*-means and the *k*-means^2 algorithms for huge datasets.

The blue dotted line in Figure 3.14 refers to the normal *k*-means clustering method while the red dashed line refers to the *k*-means^2 algorithm. Obviously, the CPU time difference between the two algorithms largely favors the *k*-means^2 and the time advantage in fact increases at an almost exponential rate. For example, the CPU time required for the

normal *k*-means clustering using the 6M dataset is almost 1250 seconds, while the time for the 2-stage method is only 450 seconds, giving about a 3 times faster performance of *k*-means^2 . At the same time, for a 1M dataset size, the speed up of a *k*-means^2 algorithm is still significant but slightly smaller - *k*-means^2 is 2 times faster than *k*-means. Such reduction in CPU time consumed for calculation is very useful and cost effective in larger datasets.

The trend in the speed up advantage is clearly shown in Figure 3.15 and demonstrates a steady increase upward.

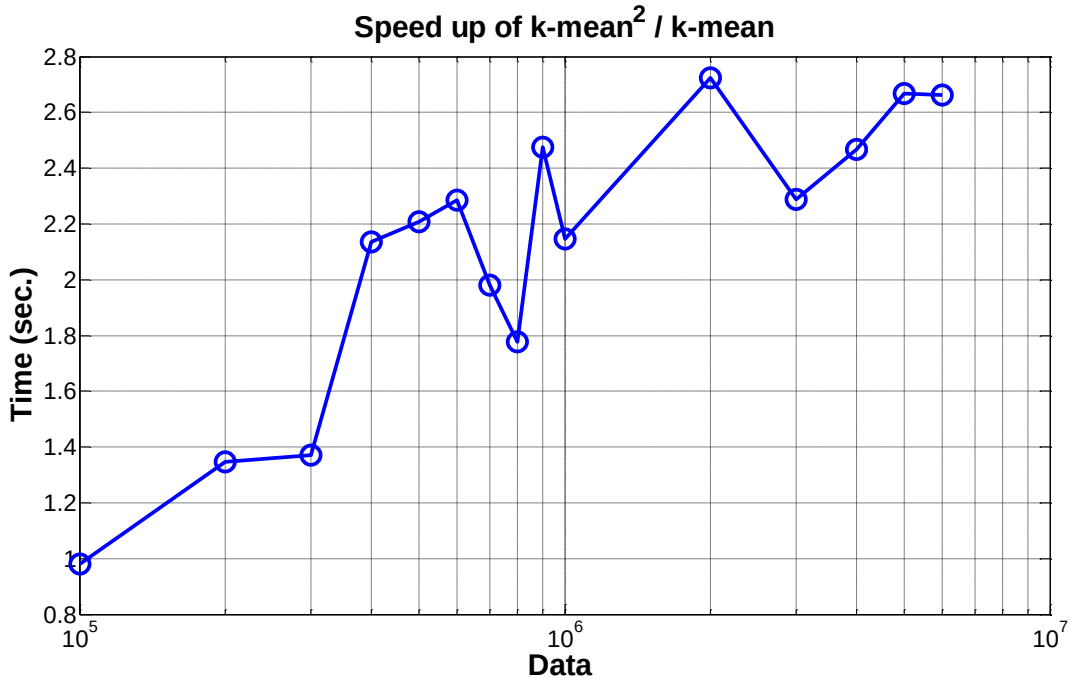


Figure 3.15: Speed up of *k*-means^2 / *k*-means algorithms

Hardware limitations particularly in the availability of a more powerful computer prevented us from testing the method for data larger than 6M points. This could be a subject for future work.

Analysis of the center convergence is presented and shown in the next section.

3.7 Center convergence

In this section we will analyze performances of a classic *k*-means and a novel *k*-means^2 algorithm by showing the dynamic changes of differences between the cluster center during the iterations and the final cluster center (a mean of the cluster obtained).

To avoid having to guess the proper number of clusters, the classification datasets were chosen instead of clustering datasets. In this case we assume that we know the number of clusters as a priori knowledge to avoid spending time on finding the proper number of clusters while we are concentrating on proofing the advantages of using *k*-means^2 in speeding up the algorithm process.

Here we are using a dataset chosen at UCI, [Alcock and Manolopoulos, 1999]. The dataset is "Synthetic Control Chart Time Series" with 600 points, 60-dimesion and 6 clusters. We only show the changes of a distance in a single coordinate for both *k*-means and *k*-means^2 algorithms. The results for the *k*-means algorithm are shown in Figure 3.16 while the results of running the *k*-means^2 are shown in Figure 3.17.

The movement measured as the difference between the mean of the cluster and the centers is shown for the *k*-means algorithm for over 300 iterations. As expected, the difference is seen to fluctuate rather rapidly initially and gradually dies away.

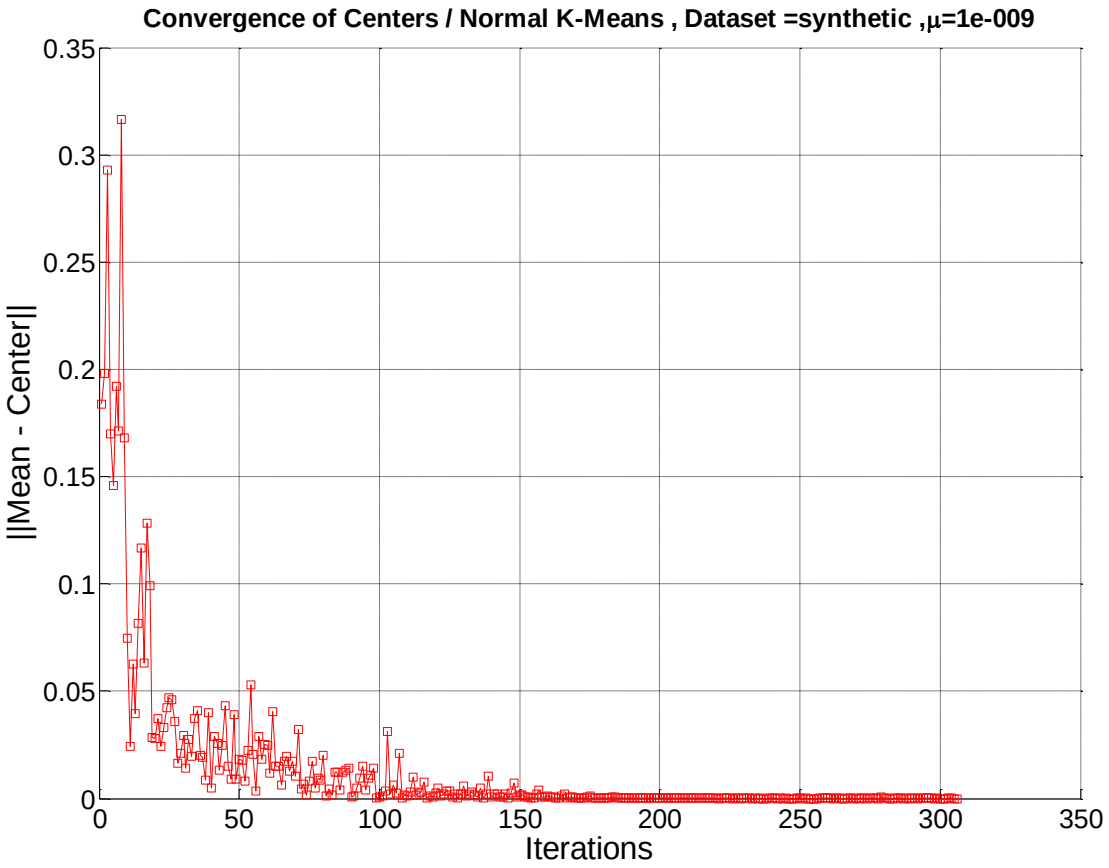


Figure 3.16: Dynamic changes of differences between the cluster center during the iterations and the final cluster center for the *k*-means algorithm

As for the *k*-means^2 algorithm, the stopping criterion for the *fast stage* was selected to be 10^{-3} and for the *slow stage*, double precision was chosen as 10^{-6}, which is the same as for the normal *k*-means method. The two stages of the *k*-means^2 algorithm are shown in two different colors for clarity. The blue represents the fast stage convergence while the red represents the slow stage one. It is important to remember that the blue color represents the changes for just a portion of the dataset (i.e. 10%), while the red one displays the changes of the differences between the mean and centers obtained by using all the data.

Because the fast stage uses a small portion of data, the CPU time needed for the blue curve to converge is much smaller than the time needed for the same portion of the trajectory in Figure 3.16. Furthermore, the terminal center reached by the fast stage (blue) is actually the seed for the slow stage (red) which is in this way much closer to the final cluster mean and thus needs less iterations to reach it.

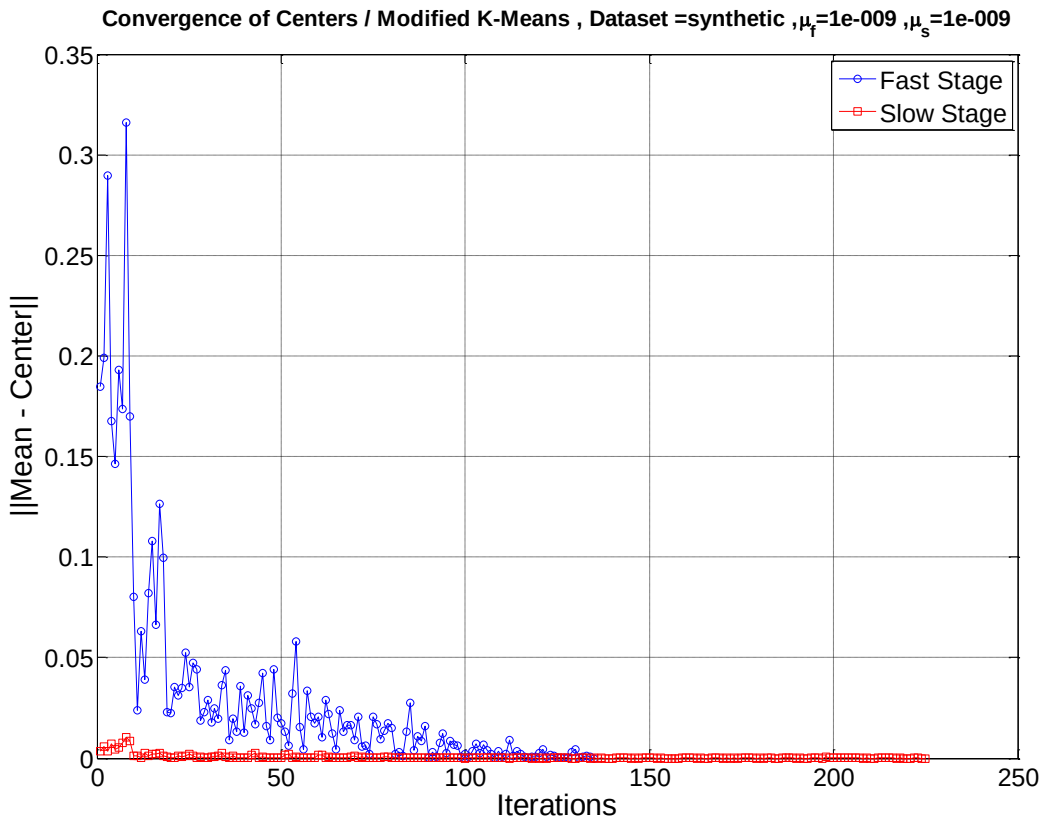


Figure 3.17: Dynamic changes of differences between the cluster center during the iterations and the final cluster center for the *k*-means[2] algorithm - *fast stage* (blue), *slow stage* (red)

3.8 Conclusions

Because the *k*-means algorithm makes use of the full dataset in order to carry out its clustering action, the computational power is expected to rise exponentially with the in-

crease in dataset size (large and ultra large datasets). The present proposal targets the problem of moving the seeds towards the final cluster's centers destination with as much data as possible. Therefore, the use of a portion of the data may solve the problem. However, one condition has to be satisfied before the reduction in the dataset can be justified. The condition is that the portion of the data selected must ensure the existence of the same clusters, as shown in Theorem 1. Hence, the new proposal can be seen as a two stage *k*-means algorithm.

Admittedly, the longer the distance between the seeds and the cluster's centers the more iteration is required. For the *k*-means algorithm it will require the use of the full dataset in order to move the seeds towards the actual centers of the clusters whereas it only requires a small portion of the dataset in the *k*-means^2 algorithm in order to carry out this movement. Hence, a great computational expense reduction can be achieved with this modification.

The effect of the cluster's locations and the dataset's sizes are analyzed and discussed. The normal *k*-means clustering method requires a longer time to achieve the clustering compared with the 2-stage method for large datasets (more than 1Million points).

CHAPTER 4: REGRESSION AS CLASSIFICATION

4.1 Regression by classification using support vector machines

Here, we further develop the novel approaches to the solution of regression problems by transforming them into multiclass-classification tasks. Such an approach can be solved in two steps. The first is to find a proper discretization criterion. This involves finding out which discretization method is the most appropriate for transforming a regression into a multi-class classification.. The second step is mainly the selection of a proper classification method. Hence, the whole process would entail the discretization of the output variables y_i into the discrete labels and selection of one of the available classifications techniques such as rule-based, decision-trees, neural network or SVM to perform the classification. Classification by decision-tree for high dimensional data does not work properly due to the memory limitation as well as due to the high computational cost. However, decision-tree may work satisfactorily for data having less than 20 features.

A few early attempts to solve the regression problem as the multiclass classification tasks by using the rule-based decision and decision trees have been presented by [Weiss and Indurkhya, 1995] and [Torgo and Gama, 1996].

The latter, [Torgo and Gama, 1996], presented a methodology that enables the use of existent classification inductive learning systems on problems of regression. This goal was achieved by transforming regression problems into classification problems. They have performed an extensive empirical evaluation using two decision tree classifiers, C4.5 and CN2, on four real world domains.

The research of [Torgo and Gama, 1996] has two phases. The first part is to provide three different discretizations methods as follows:

1- Equally probable intervals (EP); to create *N* intervals with the same number of data
2- Equal width intervals (EW); to create *N* intervals with the same range
3- *K*-means clustering (KM); to create *N* intervals that minimize the sum of distances of each element to the center of the cluster. It starts with an EP approximation but then it tries to move the elements to their nearest centers.

The second part is to enable the use of these methodologies with other classification systems. As to the first goal, they were able to prove (through empirical evaluation on four real datasets) that two of their proposed discretizations methodologies outperformed the method used in the work of [Weiss and Indurkhya, 1993]. These experiments also revealed that the best methodology is dependent on both the regression domain as well as on the classification system used, thus providing strong evidence for the search-based discretizations method. With respect to the second goal they have used their methodologies with two decision tree classifiers, CN2 and C4.5.

The problem in the three methods mentioned above is the a priori knowledge of the number of intervals. By calculating the means of an iterative search approach, [Torgo and Gama, 1996] claimed that they had overcome the problem. Furthermore, they developed two ways to modify the three splitting methods mentioned above based on estimated predictive accuracy results. The first is called ‘Varying the number of intervals’ (VIN). This method is based on trying several values of the number of intervals with the current splitting strategy followed by incrementing the number of intervals by constant value. The se-

cond method is called ‘Selective specialization of individual classes’ (SIC). The basic idea of this method is to improve the previously tried set of intervals. They start with any given number of intervals and during the CV-evaluation they calculate the error estimates of each individual discrete class. Then, they look for the individual error estimate. The median of these errors is calculated and any error above the median is specialized.

However, no comparison to any other methods was provided in order to evaluate the proposed methods. They also did not show the accuracy which can actually be achieved by their approach but they do claim that their methods are superior to the others. They did not provide any comparison of the results whereby relative merits cannot be highlighted either.

Here, a novel framework of solving regression problem into multiclass classification using Support Vector Machines (SVMs) is proposed. The obvious and direct way of transforming the regression problem into the classification task is to perform the discretizations of the target vector **y** by a *fixed ε-tube* size into a set of *N* classes. The problem in the fixed ε-tube discretizations is that it may lead to the empty classes. A slightly better form of discretizations is achieved by specifying the *minimal size of the* ε-tube. If in such a minimal tube there are no entries, the size of the tube is doubled. Obviously, this leads to the *varying* sizes of the tubes for different classes but it improves the accuracy by avoiding grouping too many data in the flat part of the regression (hyper) surface. The following graphical representation, Figure 4.1, shows the difference between the two proposed discretization methods for 16 data points and 5 classes in the case of a varying epsilon tube populated by at least one sample in the class (class boundaries are shown by

dashed red line). At the same time, discretization by a fixed epsilon tube leads to 5 classes but with class 4 being empty (class boundaries are shown by solid blue line).

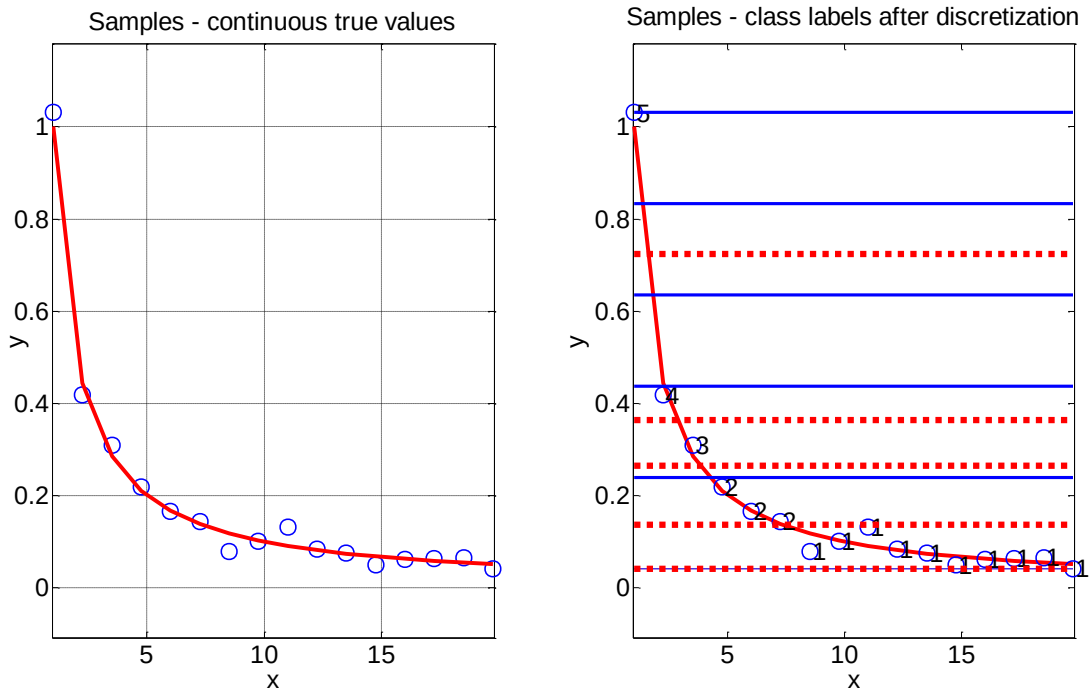


Figure 4.1: Graphical representations of the two splitting methods, *fixed ε-tube* and *varying sizes of ε-tube*

4.1.1 Preliminaries and definitions of SVM

Support Vector Machines are based on the concept of decision planes that define decision boundaries. A decision plane is one that separates between a set of objects having different class memberships. Support Vector Machine (SVM) performs classification tasks by constructing hyperplanes in a multidimensional space that separates cases of different class labels. SVM supports both regression and classification tasks and can handle multiple continuous and categorical variables. To construct an optimal hyperplane, SVM employs an iterative training algorithm, which is used to minimize the error function.

The learning problem setting for SVMs is as follows [Huang, Kecman and Kopriva, 2005]: there is some unknown and nonlinear dependency (mapping, function) $y = f(\mathbf{x})$ between some high-dimensional input vector x and the scalar output y (or the vector output y as in the case of multiclass SVMs). There is no information about the underlying joint probability functions here. Thus, one must perform distribution-free learning. The only information available is a training data set $\{\mathcal{X} = [\mathbf{x}(i), y(i)] \in \Re^m \times \Re, i = 1, \ldots, n\}$, where n stands for the number of training data pairs and is therefore equal to the size of the training data set $\mathcal{X}$. Often, y_i is denoted as d_i (i.e., t_i), where $d(t)$ stands for a desired (target) value. Hence, SVMs belong to the supervised learning techniques. The basic model for the error function of SVM is:

$$R = \sum_{i=1}^{n} L_\varepsilon \quad + \quad \Omega(n, h) \tag{4.1}$$

Where L_ε is a SVMs' loss function (Closeness to data), h is a VC dimension, and Ω (Capacity of a machine) is a function bounding the capacity of the learning machine. In classification problems, L_ε is typically the 0-1 loss function, and in regression problems L_ε is the so-called Vapnik's ε –insensitivity loss (error) function:

$$L_\varepsilon = |y - f(\mathbf{x}, \mathbf{w})|_\varepsilon = \begin{cases} 0, & \text{if } |y - f(\mathbf{x}, \mathbf{w})| \le \varepsilon \\ |y - f(\mathbf{x}, \mathbf{w})| - \varepsilon, & \text{otherwise} \end{cases} \tag{4.2}$$

where ε is the radius of a tube within which the regression function must lie, after the successful learning. (Note that for ε = 0, the interpolation of training data will be performed) and **w** is the weight vector subject to training.

4.2 Transforming regression into classification

Regression problems are defined as follows: l instances (samples, measurements) with d input features are given as the input points or vectors $\mathbf{x}_i = (x_{i1}, \ldots, x_{id})^T$, each $\mathbf{x}_i$ is associated with known output value y_i, and $y_i \in \Re^1$. The difference in respect to the classification problems is that the output values are real numbers now. However, this setting can also be looked at as the classification problem where each y_i is treated as the 'class label'. Hence, each regression problem is a multiclass classification problem with maximally l classes. Relabeling continuous y_i values into the classes is readily done after sorting y_i and then just orderly assigning the labels. However, this obviously leads to several undesired consequences, the most important one being that we don't want to model the noise always present in data and we would like to control the variance of the model. In order to filter the noise out, as well as to reduce the variance of the model, the output vector **y** is approximated by the SVM regressor after defining the so-called 'ε-insensitivity zone' (a.k.a. ε-tube).

The 'weights' of the SVM model are usually obtained by finding the dual variables (α in classification, and both α and α^* in regression) first. More precisely, the SVM models are defined as:

$$f(\mathbf{x}) = \sum_{i=1}^{N_{SV}} w_i k(\mathbf{x}_i, \mathbf{x}) + b \tag{4.3}$$

where, in classification:

$$w_i = \alpha_i y_i \tag{4.4}$$

and in regression:

$$w_i = \alpha_i - \alpha_i^* \tag{4.5}$$

While the two final models look alike, the corresponding dual Lagrangian (QP) problems to be solved for α-*s* are fairly different;

in classification:

$$\begin{aligned} L_{\mathrm{d}}(\boldsymbol{\alpha}) &= \sum_{i=1}^{l}\alpha_i - \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}\alpha_i\alpha_j y_i y_j K(\mathbf{x}_i,\mathbf{x}_j), \\ &\text{s.t.} \qquad 0 \le \alpha_i \le C,\ i = 1,\ldots,l \text{ and} \\ &\sum_{i=1}^{l}\alpha_i y_i = 0, \end{aligned} \tag{4.6}$$

and in regression:

$$\begin{aligned} L_{\mathrm{d}} &= \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)y_i - \varepsilon\sum_{i=1}^{l}(\alpha_i + \alpha_i^*) - \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)K(\mathbf{x}_i,\mathbf{x}_j), \\ &\text{s.t.} \qquad 0 \le \alpha_i \le C, 0 \le \alpha_i^* \le C,\ i = 1,\ldots,l \quad \text{and} \quad \sum_{i=1}^{l}\left(\alpha_i - \alpha_i^*\right) = 0 \end{aligned} \tag{4.7}$$

The fundamental difference in solving the QP problems (4.6) and (4.7) is in the size of the corresponding Hessian matrix **H**. In classification, **H** is an (*l*, *l*) matrix, while in regression it is a (2*l*, 2*l*) one. Hence, it's of a double size and inherently nonsingular.

Here, we will attempt something else. The regression problems will be transformed into the multiclass classification tasks by a discretization.

There are several ways in which the discretization can be performed. The obvious and direct way of transforming the regression problem into the classification task is to perform the discretization of the target vector **y** by a fixed ε-tube size into a set of *N* classes (as already mentioned above). The size of the ε-insensitivity zone controls the accuracy of the approximation and this step is similar to defining the ε-tube in the SVMs for a regres-

sion. However, unlike in the SVMs regression problem defined by (4.7), the SVM classifier (obtained after the discretization) solves the multiclass classification task posed, as in (4.6). Figure 4.2 shows the simple example; a one-dimensional noisy hyperbolic function (left) and the class label of each training sample after using $\varepsilon = 0.0931$ i.e., after the discretization of $\mathbf{y}$ into $N = 10$ classes (right). The solid curve shown in both graphs is a noiseless function. Dashed lines in the right hand graph show class boundaries and the mean value of each class is shown as a solid line (the mean value of class 2 is explicitly pointed at with a text).

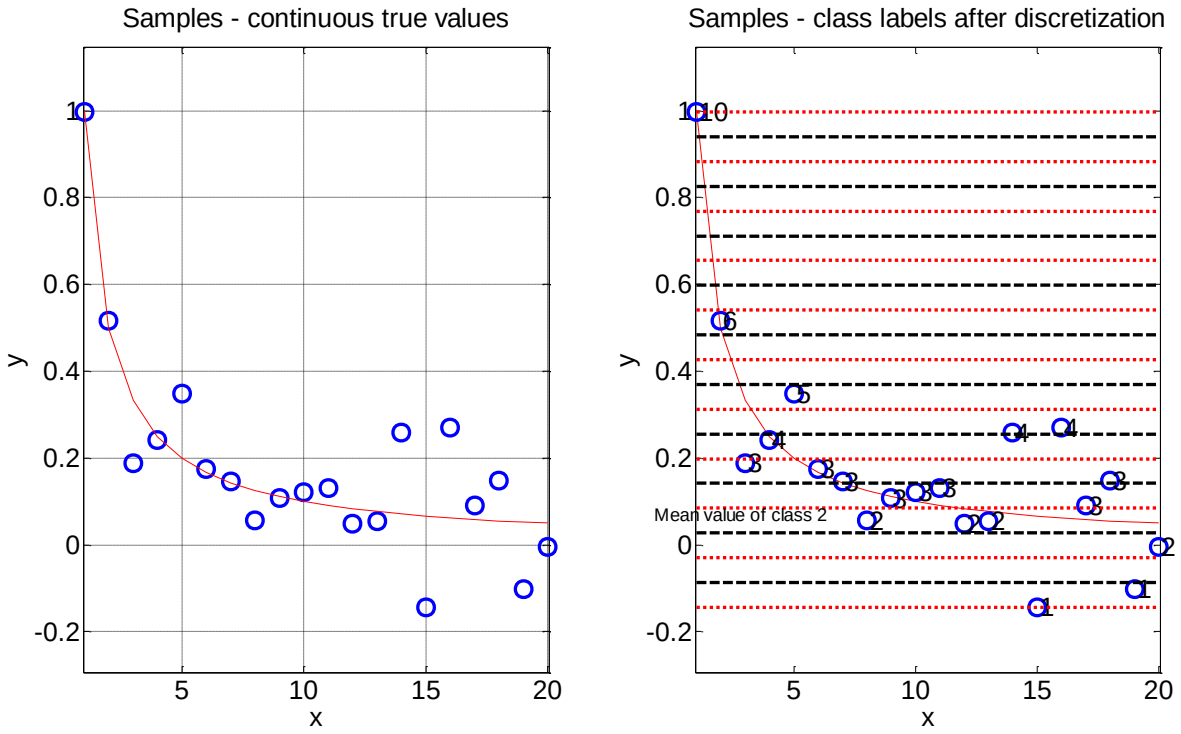


Figure 4.2: Class assignments in a regression task with equal ε-tube (hyperbolic function)

Figure 4.3 shows the simple toy example; a one-dimensional noisy sine function (left) and the class label of each training sample after using $\varepsilon = 0.226$ i.e., after the discretization of $\mathbf{y}$ into $N = 10$ classes (right). Solid line shown is a noiseless sine function.

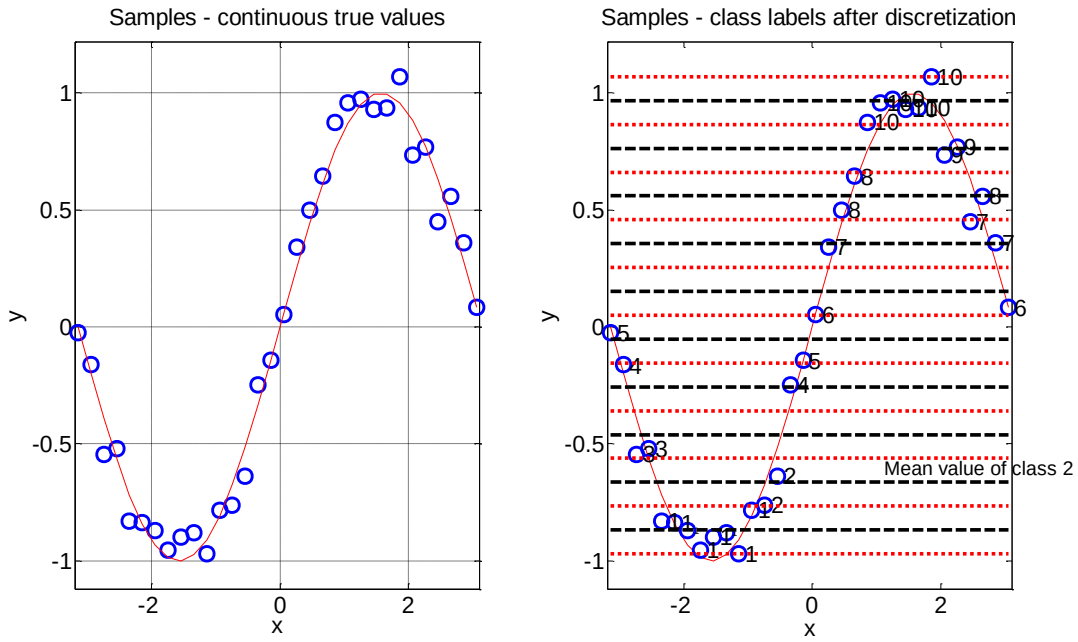


Figure 4.3: Class assignments in a regression task with equal ε-tube (sine function)

The procedure for transforming the continuous values of the response (output) vector $\mathbf{y}$ into the set of labels is the same for any high-dimensional $\Re^d \to \Re^1$ mapping. The $\Re^1 \to \Re^1$ regression example shown in Figure 4.3 is used for simplicity of visualization only. After the discretization, a vector of N classes' mean values is saved. In the example above this vector, $\boldsymbol{\mu}$ = [0.0592 0.1608 0.2624 0.3640 0.4656 0.5672 0.6689 0.7705 0.8721 0.9737]^T.

The SVM multiclass classification produces the vector $\mathbf{y}_p$ containing the predicted class labels which are then replaced with the mean values of each class. This works as fol-

lows in Figure 4.3; suppose that the ALH classifier would predict the belongings of the 8 data points to be $\mathbf{y}_p = [4\ \ 3\ \ 3\ \ 3\ \ 2\ \ 1\ \ 1\ \ 1]^T$. These labels would then be translated into the following 8 final approximated values of the noisy data points $\mathbf{y}_a = [0.3640\ \ 0.2624\ \ 0.2624\ \ 0.2624\ \ 0.1608\ \ 0.0592\ \ 0.0592\ \ 0.0592]^T$.

Note that a discretization into N classes with an equal ε-tube over the whole range leads to empty classes. Here, classes 5, 7, 8 and 9 don't have any data points. In other words, later, a reassigning of predicted class labels as the continuous values will lead to higher accuracies in the flat portions of the function. Finer discretization increases the accuracy but it may also result in an over-fitting behavior of the model. The right level of a discretization (i.e., the best number of classes N, or the best size of the ε-tube) should be determined by *cross-validation*. The same is valid for the penalty parameter C, and the shape parameter γ (if Gaussian kernel is used which is the case here). Hence, there are three hyper-parameters in the SVM multiclass classification which should be determined by cross-validation. Recall that in SVM regression the same three hyper-parameters must be tuned, but the size of the Hessian **H** is doubled.

A slightly better method of discretization is achieved by specifying the minimal size of the ε-tube. For a given minimal ε-tube, during the discretization, care is taken such that in each class there is at least one training data point. The right minimal size of the ε-tube should also be determined by the cross-validation. Obviously, this leads to the varying sizes of the tubes for different classes but it improves the accuracy by avoiding grouping too many data in the flat part of the regression (hyper)Surface. The results of the second type of discretization are shown in Figure 4.4.

The third possible way to do the discretization of continuous values is to divide them by the so-called equal-frequency discretization. This is done by dividing the output vector **y** value range into a number of intervals (classes) so that (approximately) the same number of training data points are in each interval (class). For example, if one chooses to perform the discretization into $N = 10$ classes, each interval will contain about 10% of the training data. However, such a discretization didn't lead to good results and it is no longer considered here.

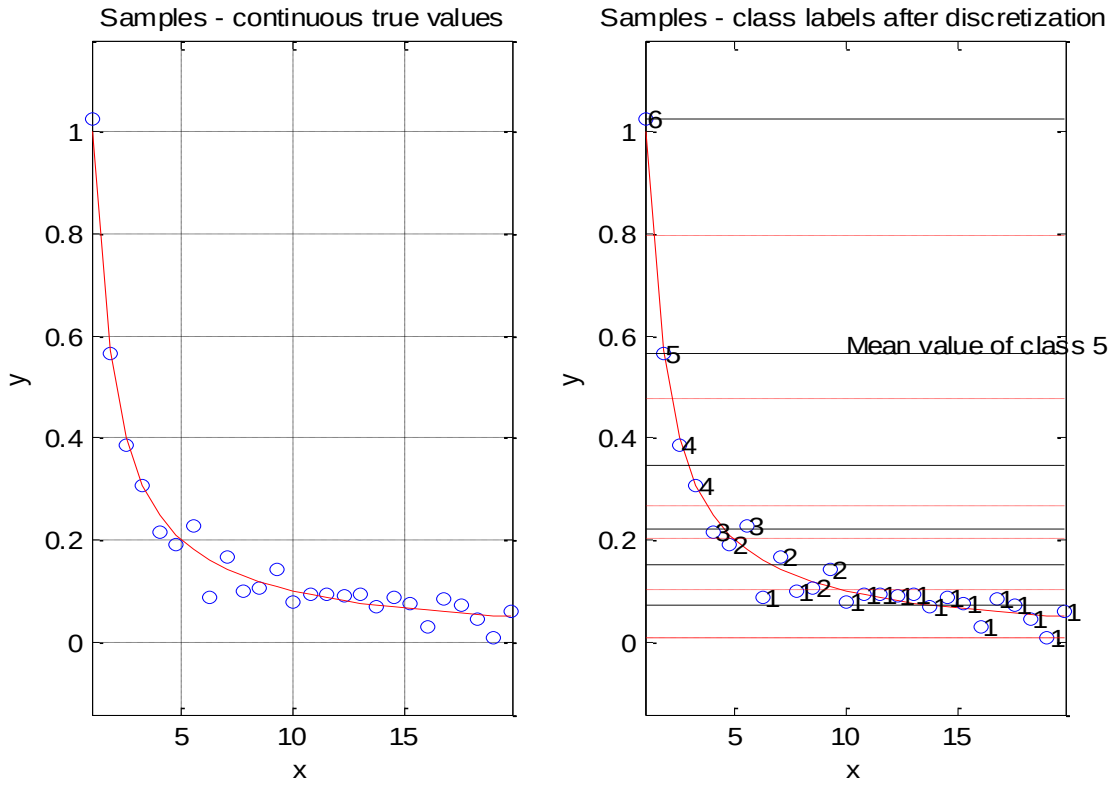


Figure 4.4: Class assignments in a regression task with varying ε-tube

Once the approximation $\mathbf{y}_a$ to a test vector **y** is calculated, the relative percentage error of the approximation is calculated by using the following expression,

$$e = 100\frac{\|\mathbf{y} - \mathbf{y}_a\|_2}{\|\mathbf{y}\|_2} \tag{4.8}$$

The fourth possible method of discretization of y_i values is to use the *k*-means algorithm to divide them according to the center of clusters of the data. In this method we try to build *N* intervals that minimize the sum of the distances of each point of a cluster to the centroid of the cluster. The suggested number of clusters is the number of discretization intervals. The two popular examples used in this case were the sine and the hyperbolic functions, as shown in Figure 4.5 and Figure 4.6, respectively.

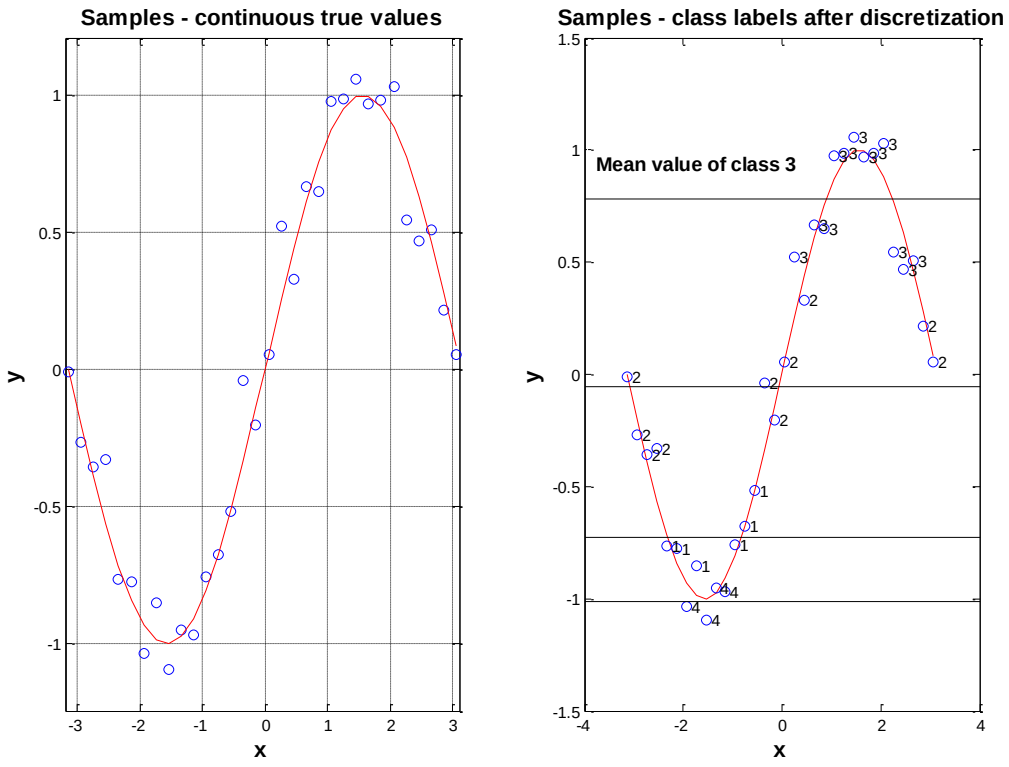


Figure 4.5: Class assignments in a regression task with *k*-means algorithm (sine wave)

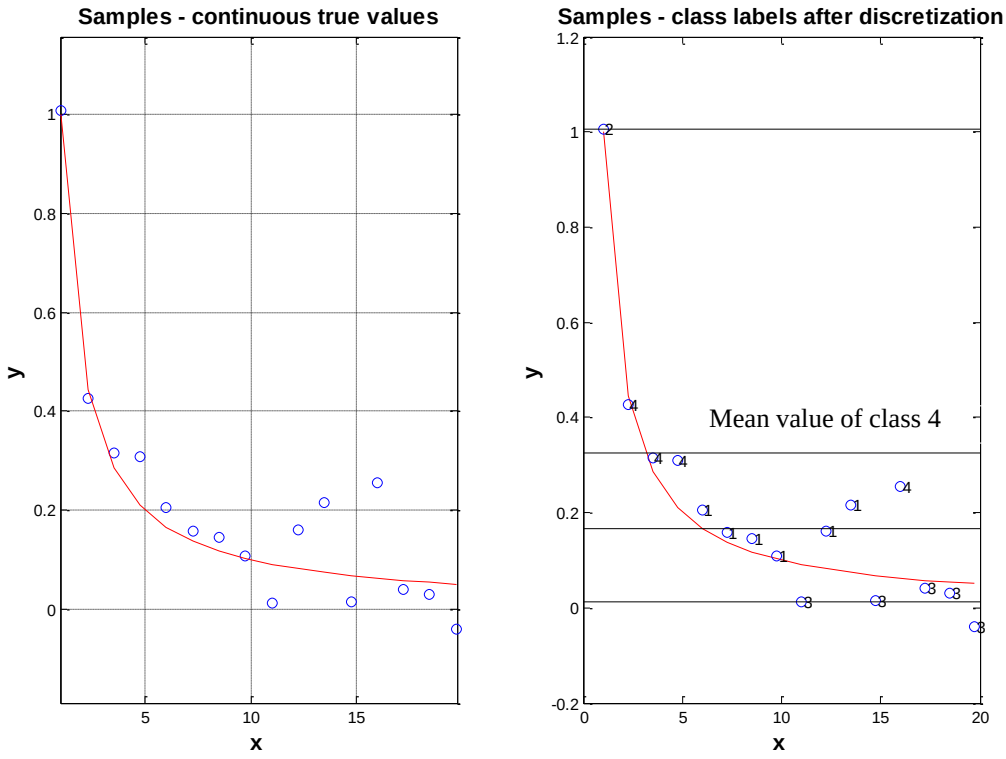


Figure 4.6: Class assignments in a regression task with *k*-means algorithm (hyper)

Note that in the use of *k*-means algorithm for the class assignment, each class is surrounding the centroid of its cluster.

The CAIM (class-attribute interdependence maximization) by [Kurgan and Cios, 2004] is aimed at a discretizations of a continuous data based on a top-down method, and it is popular for its reduced complexity. CAIM has also been investigated in this work as another method of discretization. The CAIM algorithm is designed to work with supervised data only and its functioning can be summarized as follows: assume that we have a mixed-mode data set consisting of *M* examples, and that each example belongs to only one of the *S* classes. *F* denotes continuous attributes. Then, there exists a discretization scheme *D* on *F*, which discretizes the continuous domain of attribute *F* into *n* discrete intervals bounded

by pairs of numbers (boundary points). Unfortunately, CAIM has not been good in discretizing a data coming from the regression problem (as described in section 4.2), where we have *l* instances (samples, measurements) with *d* input features given as the input points or vectors $\mathbf{x}_i = (x_{i1}, \ldots, x_{id})^T$, and a corresponding output value $y_i \in \Re^1$.

4.3 Regression data sets and the results

In order to show the performance of the proposed approach in solving regression tasks, twelve benchmarking regression data sets are selected for the study (Some preliminary results on the 12 datasets have been given in [Salman and Kecman, 2012]). They are *Boston housing* data [Asuncion and Newman 2007] and [Harrison and Rubinfeld 1978]; *prototask* (*comp-active*) data base [Delve 2008]; *concrete* compressive strength [Asuncion and Newman 2007] and [Yeh 1998]; *servo* data and *wine* data [Asuncion and Newman 2007]. The remaining data sets; *Machine CPU*, *Pyrimidines*, *MPG*, *Diabetes*, *PhD*, *Abalone*, *Triazines* are taken from the following sources: UCI (University of California) [Asuncion and Newman 2007], Liaad, and Princeton websites [Uci, 2012], [Liaad, 2012] and [Princeton, 2012].

In the tables and graphs below the comparisons between the SVM classifier solving regression problems (by using three ways of discretization) and an SVM regressor are done for twelve data sets. The details of all twelve data sets can be found in the references given and the simulation results are shown in Table 4.1 (a) and (b).

Table 4.1 (a): Percentage errors and CPU times for 12 data sets obtained by SVM classifiers and an SVM regressor (bold indicates better result)

Data/Algorithm	SVM classifier varying ε- tube			SVM classifier fixed ε- tube			SVM classifier k-means			SVM Regressor		
	Error %	Std %	CPU time sec/run	Error %	Std %	CPU time sec/run	Error %	Std %	CPU time sec/run	Error %	Std %	CPU time sec/run
Servo	**19.597**	2.581	0.054	20.040	1.782	0.082	20.685	0.719	0.186	24.890	1.469	0.426
Boston Housing	**12.642**	0.236	0.929	12.746	0.333	0.626	18.960	0.465	1.580	12.754	0.201	0.475
CPU	**26.560**	0.025	0.468	31.110	0.023	0.170	34.670	0.029	6.072	27.800	0.034	0.925
Pyrimidines	**10.480**	0.005	0.040	11.320	0.006	0.043	11.400	0.005	0.033	12.200	0.005	0.429
Prototask	**3.200**	0.012	428.000	**3.200**	0.012	98.000	21.364	0.026	85.458	3.300	0.007	345600.000
Concrete	14.704	0.225	7.490	14.654	0.225	6.790	17.576	0.246	4.612	**13.397**	0.235	16.300
MPG	11.878	0.173	1.567	12.258	0.358	1.794	16.631	0.278	1.308	**10.655**	0.222	1.206
Wine	11.824	0.071	3.730	11.889	0.074	4.880	12.065	0.093	3.450	**11.761**	0.101	2.640
Diabetes	12.350	0.006	0.158	12.880	0.004	0.119	12.940	0.003	0.543	**11.350**	0.003	0.143
PhD	24.400	0.021	0.053	26.240	0.015	0.032	25.550	0.019	0.060	**20.130**	0.016	0.039
Abalone	23.200	0.001	17.787	35.640	0.000	9.190	23.380	0.001	7.916	**21.680**	0.000	23.652
Triazines	21.830	0.006	0.176	23.970	0.002	0.057	**20.810**	0.005	0.191	20.820	0.006	2.122
Average Error	16.055	0.280	38.371	17.996	0.236	10.149	19.669	**0.157**	**9.284**	**15.895**	0.192	28804.030
exclude prototask			2.950			2.162			2.359			4.396
Mean on spiky data	**14.496**			15.683			21.416			16.189		
Mean on smooth data	17.169			19.647			18.422			**15.685**		

The basic characteristics of the data sets used, as described in Table 4.1 (b), are as follows: *Boston housing* is a collection of 506 samples with 12-dimensional input vectors,

the *prototask* data set contains 8192 instances with 21-dimensional input vectors, *concrete* compressive strength data consists of 1030 samples with 8-dimensional inputs, the *servo* data set contains only 167 samples with 4-dimensional feature vectors, and the *wine* data is a collection of 1599 12-dimensional feature vectors. Note that the wine data's outputs are integer values corresponding to 6 types of wine but in the references, the wine data set is treated as the regression problem. The *CPU* (machine CPU) data set contains 209 samples with 6-dimensions only. The *Pyrimidines* data set contains 74 samples with 27-dimensions only. The *MPG* data set contains 398 samples with 7-dimensions only. The *Diabetes* data set contains 43 samples with 2-dimensions only. Note that the *Diabetes* data was taken from the Liaad website. The *PhD* data set contains 73 samples with 4-dimensions only, taken from the Princeton website. The *Abalone* data set contains 4177 samples with 8-dimensions only. The *Triazines* data set contains 186 samples with 60-dimensions only.

Here we have compared the performances of four SVM models, all of which use Gaussian kernels - three models are solving regression problems as multiclass classification tasks and one model is a standard SVM regressor. The three SVM classifiers differ only in the way the discretization (described above) has been carried out.

There are several things to point out in Table 4.1. First and foremost, one should note that there are two groups of regression problems (based on the behavior of output vector **y**) which are dubbed here as 'spiky' and 'smooth' datasets. One can see that in approximating spiky regression problems, the SVM classifiers have slightly outperformed the SVM regressor. It is opposite for modeling smooth regression problems. Below, the discussion and presentation of a smoothness of vector **y** will be presented and commented.

Furthermore, out of three classifiers, the SVM classifier with the varying ε-tube has shown the best overall performance averaged over all twelve data sets. As for the CPU time needed to finish the training, SVM classifiers have shown better performances as well. This is particularly obvious for the largest data set *prototask,* when the SVM regressor needed four days to finish the training. Generally, once the data set goes over several thousands of training samples the training speed-up of the SVM classifiers becomes obvious. Note that all the results have been obtained within the double (nested) cross-validation experimental procedures, guaranteeing the most objective results and the true performance range on the future data samples. Also note that the LibSVM software has been used and the same number of hyperparameters has been tested in all the simulations; ($C = \{5 \dots 10^4\}$, $\sigma = \{10^{-4} \dots 1\}$, $N = \{3 \dots 100\}$, $\varepsilon = \{0.001 \text{ to } 0.6\}$ of $\max(\mathbf{y})$).

Below, there are two graphical presentations of the results given in Table 4.1 in order to see them better. The results are shown as the error bar graphs in Figure 4.7 and as the 3-dimensional bar graphs in Figure 4.8. SVM Classifier with varying epsilon results are shown as the red curve and SVM Regressor results as the blue one in Figure 4.7.

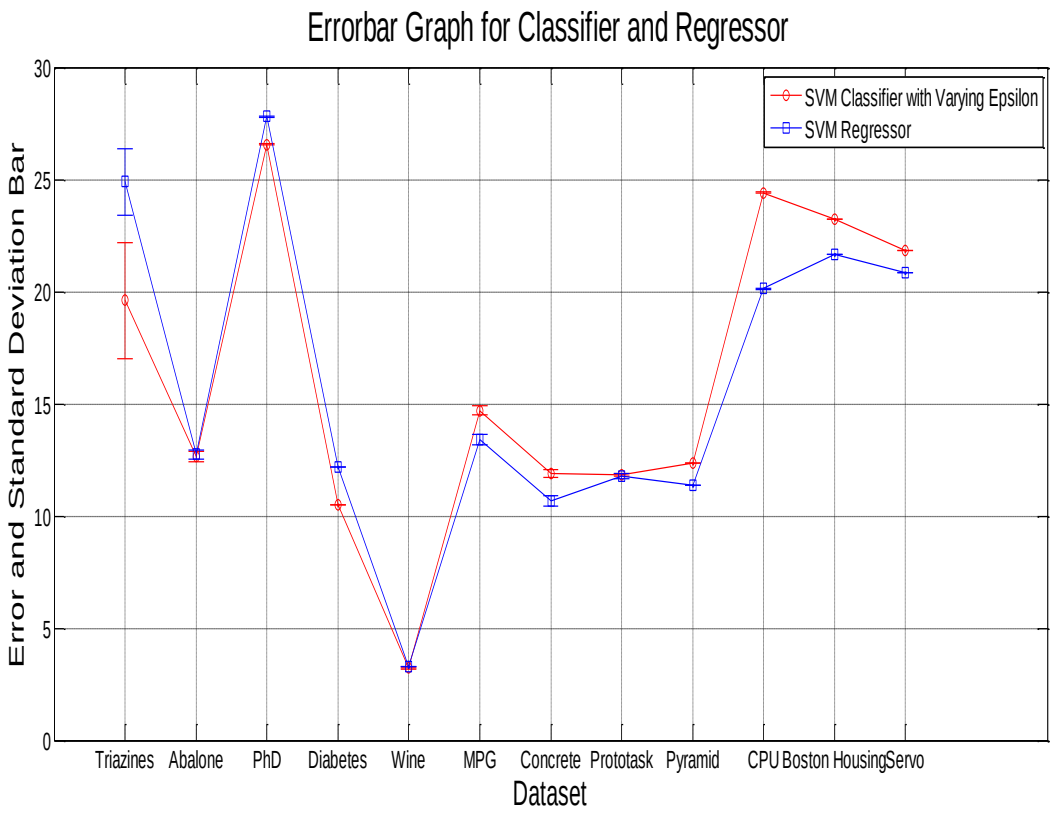


Figure 4.7: The mean error and mean squared errors of the 12 datasets

The figure above shows both the mean errors and the standard errors for the twelve datasets used in our validation. It is just a graphical representation of Table 4.1 in which one can see again that for the first 5 (spiky) datasets solving a regression problem as the classification one is better than solving it as the regression problem. Another way of presenting Table 4.1 (b) is shown in Figure 4.8

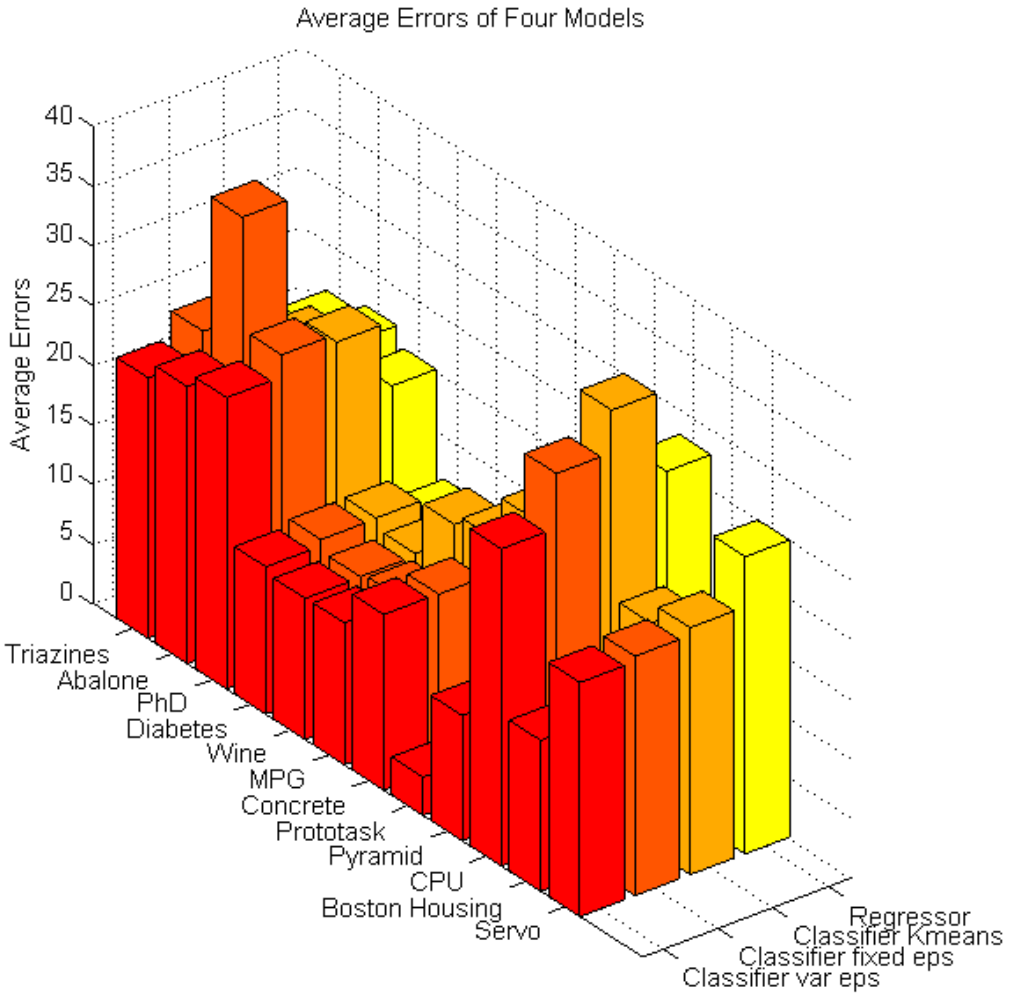


Figure 4.8: Average errors of 12 datasets based on 4 models

The best parameters (C, σ and the level of a Discretization) used to obtain the SVM classifiers and SVM regressor are shown in Table 4.1 (b).

Table 4.1 (b): Best parameters (C, σ and the level of a Discretization) as well as the size and sources of 12 data sets obtained by SVM classifiers and an SVM regressor

Data/Algorithm	Size and Source			Best Parameters		
	Samples	Dimensions	Source	Best C Parameter	Best Gama Parameter	Best Sampling
Servo	167	4	[Uci, 2012]	[7500 10000]	[0.1 0.25]	[100]
Boston Housing	506	12	[Uci, 2012]	[750 1000 2500 5000 7500]	[0.0025 0.004 0.005]	[12 13 14 15]
CPU	209	6	[Liaad, 2012]	[2500 5000 7500 10000]	[0.025 0.05]	[14 15 16 20 22 24 27]
Pyrimidines	74	27	[Liaad, 2012]	[8 10 15 20 25]	[.001 .005 .01 .02]	[0.1]
Prototask	8195	21	[Uci, 2012]	[10 15 20 25 40]	[0.075 0.1]	[36 41 45 46]
Concrete	1030	8	[Uci, 2012]	[5 10]	[0.05 0.075]	[0.25]
MPG	398	7	[Liaad, 2012]	[10 100]	[0.05 0.1]	[0.001 .0025 0.005]
Wine	1599	12	[Uci, 2012]	[5 10]	[0.05 0.075]	[0.25]
Diabetes	43	2	[Liaad, 2012]	[1000 20000]	[0.01 0.02 0.005]	[0.6]
PhD	73	4	[Princeton, 2012]	[10000]	[0.1]	[0.001]
Abalone	4177	8	[Liaad, 2012]	[1000]	[0.001]	[0.001 0.005]
Triazines	186	60	[Liaad, 2012]	[100 1000 10000]	[0.01 0.1 1]	[3 6 9 10]

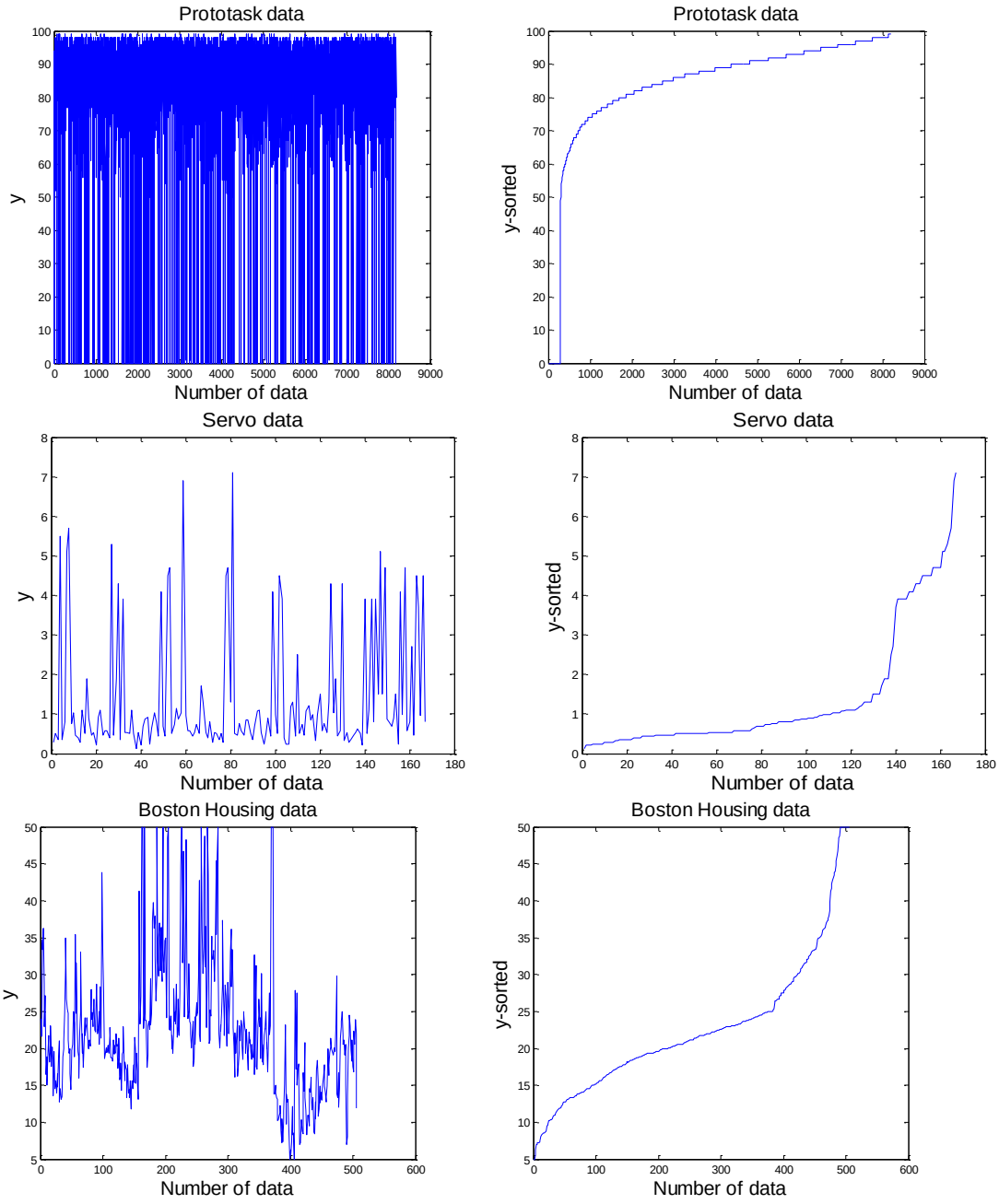
There is one more intriguing and noticeable matter raised by the results in Table 4.1. Namely, both the SVM classifiers and the SVM regressor are competing tightly for the

primacy in solving the regression problems used here by showing distinctly different performances in terms of the accuracy for the two types of the function approximation problems. SVM classifiers show better results for the *prototask, CPU, Pyrimidines* and *servo* data and slightly better accuracy than the SVM regressor for the *Boston housing data* (actually accuracies are almost equal). The SVM regressor displays better performance for the *concrete, MPG, Wine, Diabetes, PhD, Abalone* and *Triazines* data set. Some explanations and hints about these results are related to the very nature of the regression problems and they may be obtained by looking at the target values y_i and their distributions as shown in Figures 4.9 and Figures 4.10 respectively.

It seems that the SVM regressor suits better regression problems where the target values are 'uniformly' or 'smoothly' distributed over the range as shown in Figure 4.10, while in the problems *prototask, CPU, Pyrimidines, Boston* and *servo,* where the SVM classifiers show clear advantages, the output vector **y** shows a very 'spiky' behavior, see Figure 4.9, primarily covering the upper part of the range (*prototask*) or the lower one (*servo*) and making irregular spikes to the lower values (*prototask*) i.e., higher ones (*servo*). Sorted output values y_i for the same regression data are shown in the right side of the graphs of Figure 4.9 and Figure 4.10.

The 'spiky' functions (*prototask* and *servo*) resemble the Poissonian, or exponential, distribution, in which a great deal of target values y_i will after the discretization fall into a small number of not-well-balanced classes. The experimental results show that when faced with the 'spiky' nature of the regression problems, a discretization and multi-class classification through the use of SVMs will most likely lead to a better function approximation.

The output and sorted output for all datasets are shown in Figures 4.9 and 4.10: (blue 'spiky', red, 'smooth'):



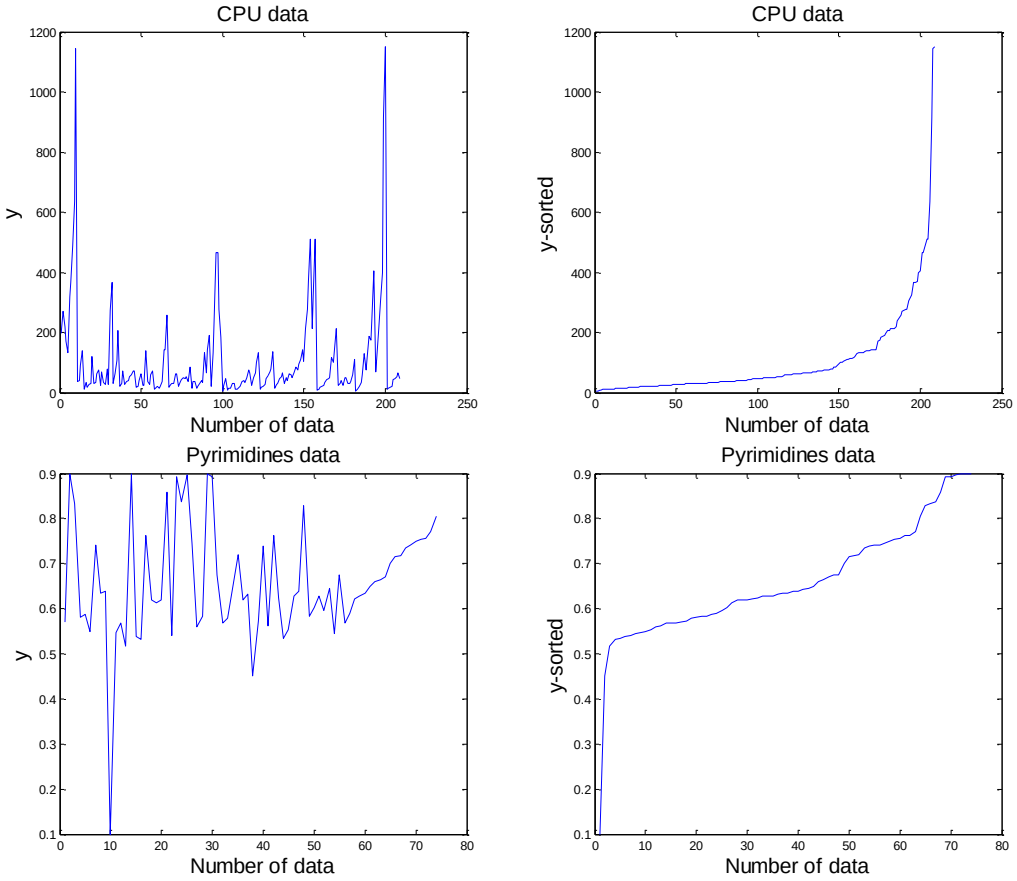
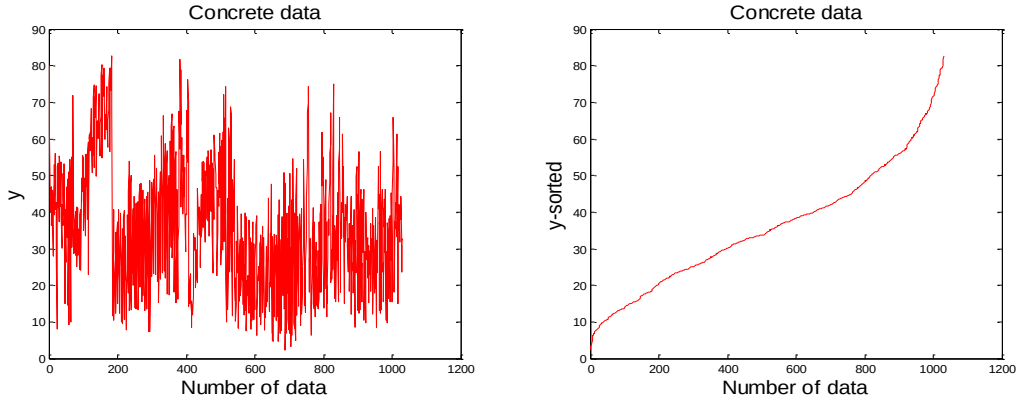
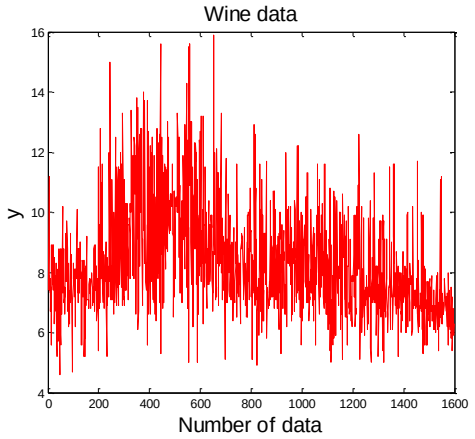
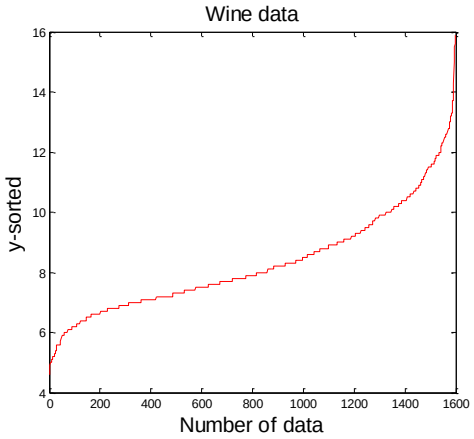
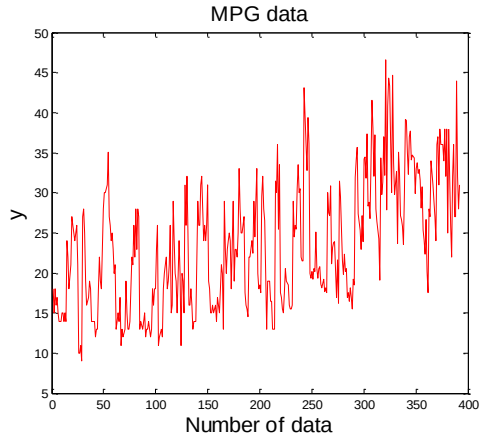


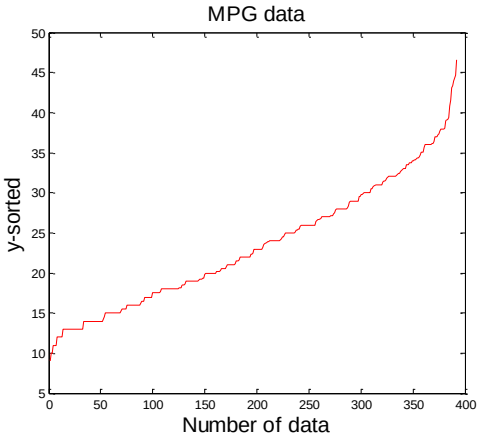
Figure 4.9: Output values and sorted output values for regression problems (spikey)

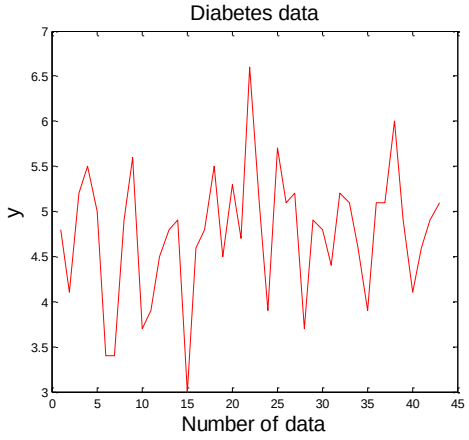


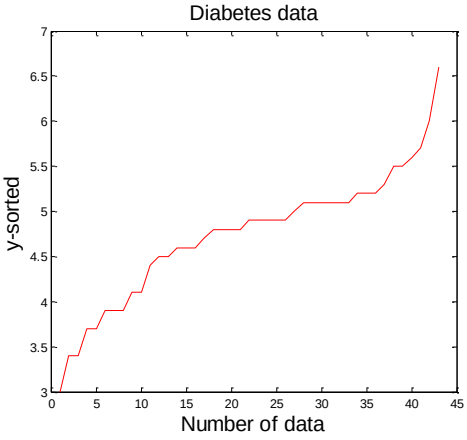

Wine data
y
Number of data


Wine data
y-sorted
Number of data


MPG data
y
Number of data


MPG data
y-sorted
Number of data


Diabetes data
y
Number of data


Diabetes data
y-sorted
Number of data

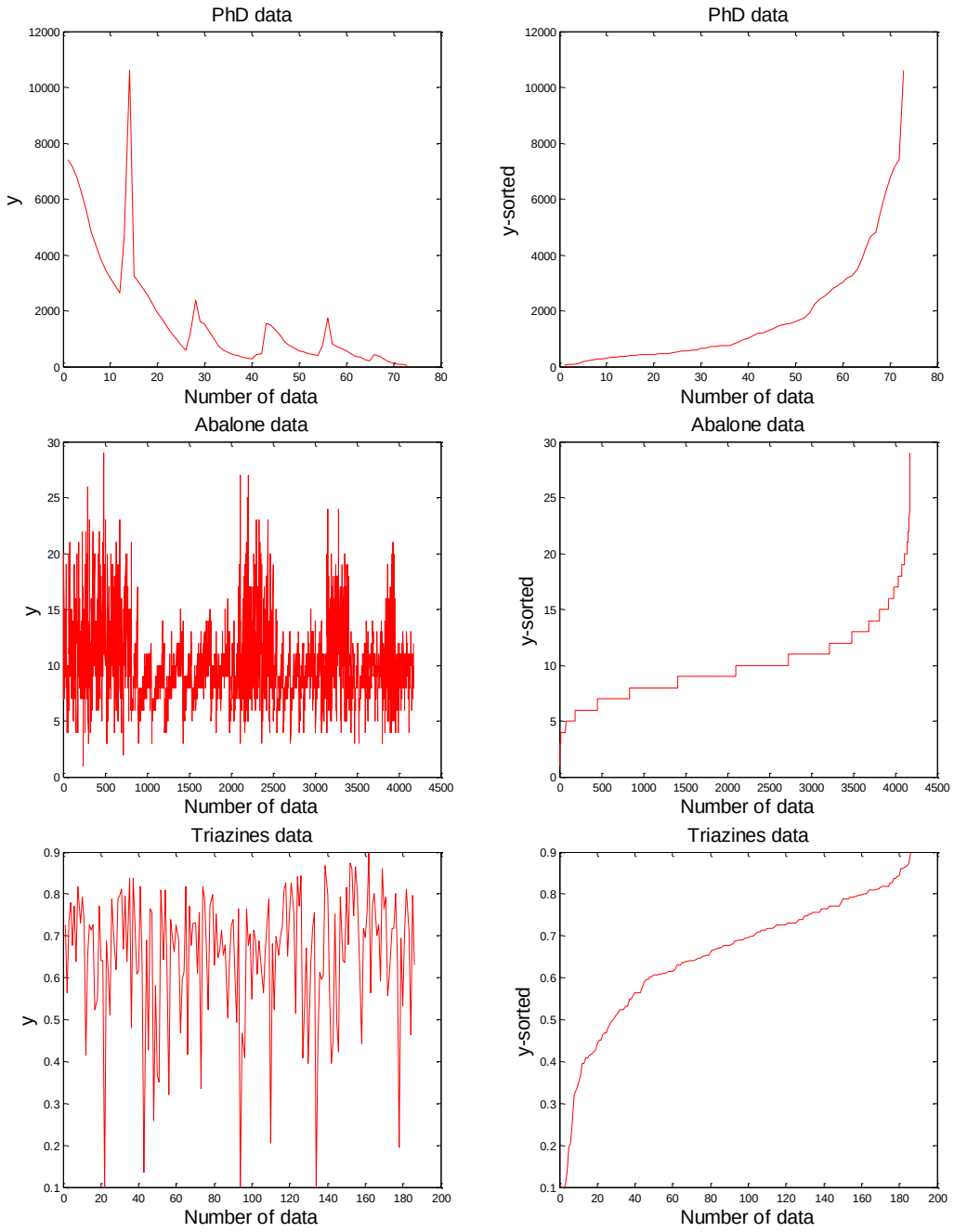


Figure 4.10: Output values and sorted output values for regression problems (smooth)

The results of the *Servo*, *Boston*, *CPU*, *Pyrimidines* and *Prototask* datasets show the spiky behavior of the output. The other datasets show the smooth behavior of the outputs. The

use of the SVM classifier with varying ε-tube produced good results for the spiky datasets over other classifiers. On the other hand, the use of the SVM regressor produced good results using the smooth datasets comparing with other SVMs. The SVM classifier with *k*-means discretization produced better results with the Triazines data set comparing with the varying ε-tube and the SVM regressor as shown in Fig 4.10.

There is one more challenging problem (actually a set of questions and some hints regarding the theoretical properties of the novel approach in solving regression problems) that arose by the results obtained, which is beyond the scope of this dissertation. Basically, it concerns the theory of bounds for both the multi-class classification and regression problems. (The next remarks follow strictly [Steinwart 2011]). First, there is a basic question which arises naturally from the results obtained and it is whether, in general, the bounds on classification are tighter than the ones on regression. (If so, in addition to showing better experimental results, one may get strong theoretical support for using a classification approach when solving regression tasks). One may also be interested in some other theory-based questions, such as whether it is easier to prove classification bounds or regression ones.

Some possible hints to the answers are as follows [Steinwart 2011]: The very bounds we are talking about here usually refer to comparing the empirical error of our decision function with its true error. (The other types of bounds used are usually called oracle inequalities). Since the function classes in regression are typically somewhat larger by the nature of the loss function, bounds in regression are often worse. However, there are some exceptions, such as when the best predictor is in the hypothesis class and the loss is suffi-

ciently convex (e.g. square loss). In this case, regression bounds may be better. Note however that for general regression data sets it may be highly unlikely to get the predictor in the hypothesis class.

As for the easiness of proving the bounds, one thing is certain - good bounds are always hard to prove, independent of whether they are for classification or regression. Thus, one may state the arguments quite differently and compare combinatorial bounds (in classification) vs. continuous ones (in regression). If the sharpness is not of primary interest, then generic off-the-shelf bounds are typically straight-forward to apply, and the only tricky part is the used complexity measure (VC-dimension, covering numbers, Rademacher averages, ...). A somewhat unifying approach that can be implemented is the McDiarmid inequality, which is based (after symmetrization) on Rademacher averages. The latter can be bounded by several concepts such as VC dimension or covering numbers. In the future, while answering some of the questions raised here, some techniques along the lines of those described above can be found in the book on density estimation [Devroye and Lugosi 2001]. As for the bounds in the multi-class classification problems (which is the problem to solve after discretization of the continuous output values y_i), multiclass classification also requires a loss function, and in most cases the above techniques should work. This is most likely the reason why nobody looked deeper into that question until now. The issue of the bounds will be a matter of further theoretical investigation of the approach for solving regression problems as the classification tasks presented in this dissertation; and their solutions will shed lights on the questions of whether to solve regression problems as regression or as multiclass classification tasks

4.4 Conclusions

This work shows how regression problems, after discretization, can be solved as multiclass classification ones. SVM classifier and regressor are the modeling tools here. Four slightly different methods of discretizations are introduced and compared. On average, the SVM classifier with varying ε-tube shows the best results. In addition to being better regressors (although the differences in accuracy are not too big), the SVM classifiers are also superior in terms of CPU times needed for training SVM. This is particularly pronounced when the number of training data surpasses several thousand samples. In such cases, one should rely on solving regression problems as multiclass classification tasks. Based on the tendency of bounds in regression to be often worse than the ones in classification, the proposed approach of solving regression problems as classification ones is quite attractive from a theoretical point of view as well. This may well be one of the most important explorations in the future. The model comparisons have been done by implementing double (nested) cross-validation i.e., resampling, structured as the two loop algorithm, which is a very rigorous scheme for assessing models' performances. In such an experimental environment, not all the data points available are used for both hyper-parameter determination and accuracy estimation simultaneously. On the contrary, data for the hyperparameter selection are strictly separated from the data for the accuracy estimation, making this double cross-validation the only objective environment for the fair comparisons of different models. Thus, if the main goal is to compare different ML models on the same data sets and under the same conditions, double cross-validation must be used, as was done here.

CHAPTER 5: CONCLUSIONS AND FUTURE WORK

The concept of the *k*-means^2 algorithm in this dissertation is based on minimizing the huge computational effort normally incurred with the application of the classical *k*-means algorithm. The problem with the *k*-means algorithm is that it carries out the distance computation between consecutive centers based on the full dataset. The choice of the initial seeds or starting centers is usually arbitrary. This means that if the arbitrary locations are somehow steered towards the final locations with less computational effort, then the final tuning of the locations can be carried out with much less computational effort. This dissertation focuses on minimizing the computational effort in moving the seeds as close as possible towards the centers of the associated clusters using only a small portion of the full dataset. The goal was accomplished by repeating the *k*-means steps twice using two different sizes of data, hence the name "*k*-means^2 algorithm". The first part of the computation is carried out by selecting a small portion of the original dataset, transitioning the centers at a much faster rate towards the final center locations. The *k*-means algorithm, in contrast, uses the whole dataset throughout the computation process. By virtue of the fast speed of computation of the first stage, it is referred to as the *fast phase*. The condition imposed on this stage is that the portion of data selected must ensure the existence of the same number of clusters already present within the whole dataset. After the fast stage converges to the final centers achievable in this phase, the second stage takes over starting from the fast centers' final positions and uses the whole data set in the second, final, stage . It is clear that in the second stage the movement of the centers will be at a much slower pace simply

because the whole data set is used, requiring a great amount of computational effort. This is why the stage is referred to as the *slow stage*. The number of steps in the two stages differ in that the *fast stage* will undergo more iterations but at a faster rate than the *slow stage*, which moves at a slower rate but with fewer steps and at a higher resolution. The choice of the portion of the dataset in the *fast stage* depends largely on the size of the dataset itself. It is reasonable to assume that large and ultra large data sets, for example, will provide sufficient data with very small portions for the *fast stage* to be able to move the arbitrarily chosen initial seeds well inside the clusters, while smaller datasets may not be able to provide sufficient data for the *fast stage* to move the seeds appropriately inside the clusters. However, for small data sets there may not be a need to do the clustering in two stages anyway. The role of the *slow stage* is actually to fine-tune the final locations of the centers.

The second goal of this dissertation is to solve a regression problem using multiclass classification and to design the experimental environment for rigorous comparisons of kernel models acting as both regressors and multiclass classifiers on the same data sets. The transformation of regression into classification requires discretizing the regression output vector **y** into several classes. Then a multiclass classification will be performed in order to produce a model. The results obtained after transforming regression problems into classification tasks are encouraging in terms of both the final accuracy that the models can achieve and the CPU time needed for the training. For the twelve benchmarking data sets selected for the study, two groups of regression problems arose: 'spikey' and 'smooth' according to the output vector behavior. The SVM classifier with the varying ε-tube has shown the best performance over the 'spikey' data sets. The SVM regressor, on the other

hand, has performed better over the 'smooth' datasets. As for the CPU time needed to finish the training, SVM classifiers have shown better performances as well.

The implementation of the *k*-means^2 algorithm on a single machine is limited to the size of the RAM and to the CPU speed. For example, if the size of the dataset is 20G then the *k*-means method will take few days to complete the task. Therefore it is imperative to apply the *k*-means^2 method on a grid of parallel computers. This will take two lines of future work. One suggestion for future work is the use of Hadoop (the free framework on multi-computers). The second suggestion is the use of one or multiple GPUs.

Possibly the most valuable extension of this work might and would be in the field of developing theoretical insights about the bounds for both the multi-class classification and regression problems. This research should give an answer to the question whether, in general, the bounds on classification are tighter than the ones on regression. Such a work may then shed more light on the experimental results obtained in this dissertation.

REFERENCES

[Alcock and Manolopoulos, 1999]
R. J. Alcock and Y. Manolopoulos, "Time-Series Similarity Queries Employing a Feature-Based Approach," 7th Hellenic Conference on Informatics. August 27-29. Ioannina, Greece 1999.

[Arthur and Vassilvitskii, 2006]
Arhter, D. and Vassilvitskii, S.: "How Slow is the *k*Means Method?", SCG'06, Sedona, Arizona, USA. (2006).

[Asuncion and Newman 2007]
A. Asuncion, D. J. Newman, "UCI ML Repository", [http://www.ics.uci.edu/~mlearn/MLRepository.html], Irvine, CA, University of California, School of Information and Computer Science, 2007.

[Bradley and Fayyad, 1998]
Bradley, P. S. and Fayyad, U. M.: "Refining Initial Points for Kmeans Clustering", Technical Report of Microsoft Research Center, Redmond, California, USA, (1998).

[Cutting, Karger, Pedersen and Tukey, 1992]
Cutting, D., Karger, D., Pedersen, J. and Tukey, J., 'Scatter/gather: A cluster-based approach to browsing large document collections', In Proceedings of the Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. Copenhagen, Denmark (Eds N. Belkin, P. Ingwersen and A. Pejtersen), pp. 318-329, 1992, ACM Press.

[Delve 2008]
Delve, Data for Evaluating Learning in Valid Experiments, [http://www.cs.toronto.edu/~delve], The University of Toronto, Toronto, Canada, 2008

[Devroye and Lugosi 2001]
L. Devroye, G. Lugosi, Combinatorial Methods in Density Estimation, Springer-Verlag, 208 p., 2001

[Elkan, 2003]
Elkan, C.: "Using the Triangle Inequality to Accelerate K –Means". Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003), Washington DC, (2003).

[Ester, et al., 1996]
Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu (1996-). "A density-based algorithm for discovering clusters in large spatial databases with noise". In Evangelos Simoudis, Jiawei Han, Usama M. Fayyad. Proceedings of the Second International, Conference on Knowledge Discovery and Data Mining (KDD-96). AAAI Press. pp. 226–231. ISBN 1-57735-004-9.

[Frank and Asuncion, 2010]
Frank, A. & Asuncion, A. (2010). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[Ganti et al, 1999]
Ganti, V.; Ramakrishnan, R.; Gehrke, J.; Powell, A.; French, J.; , "Clustering large datasets in arbitrary metric spaces," Data Engineering, 1999. Proceedings., 15th International Conference on , vol., no., pp.502-511, 23-26 Mar 1999

[Har-Peled and Sadri, 2005]
Har-Peled, S. and Sadri, B.: "How fast is the *k*-means method?", Algorithmica, 41(3):185–202, (2005).

[Harrison and Rubinfeld 1978]
D. Harrison, D. L. Rubinfeld, "Hedonic prices and the demand for clean air", *J. Environ. Economics & Management*, vol.5, pp. 81-102, 1978.

[Hodgson, 1988]
Hodgson, M. E.: "Reducing computational requirements of the minimum-distance classifier", Remote Sensing of Environments. *25*, 117–128, (1988).

[Hsu and Lin, 2002]
Hsu, C.-W. and Lin, C.-J. (2002), "A simple decomposition method for support vector machines", *Machine Learning* 46(1-3), 291–314.

[Huang, Kecman and Kopriva, 2005]
Te-Ming Huang, Vojislav Kecman, Ivica Kopriva, 'Kernel Based Algorithms for Mining Huge Data Sets', November 22, 2005, Springer

[Jain, Murty and Flynn, 1999]
Jain, A. K., Murty, M. N. and Flynn, P. J., 'Data Clustering: A Review', ACM Computing Surveys, Vol. 31, No. 3, September 1999

[Kanungo et al., 2002]
T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. A local search approximation algorithm for *k*-means clustering. In Proc. 18th Annu. ACM Sympos. Comput. Geom., pages 10-18, 2002.

[Kaufman and Rousseeuw, 1990]
L. Kaufman and P. J. Rousseeuw, 'Finding Groups in Data: An Introduction to Cluster Analysis', John Wiley & Sons, Inc., New York, NY, 1990

[Khan and Ahmed, 2004]
Khan, S. S. and Ahmed, A.: "Cluster center initialization for Kmeans algorithm. Pattern Recognition" Letters, vol. 25, no. 11, pp. 1293-1302, (2004).

[Kecman and Yang 2009]
V. Kecman , T. Yang, Adaptive Local Hyperplane for Regression Tasks, Proc. of IJCNN 2009, IEEE International Joint Conference on Neural Networks, pp. 1566 – 1570, June 14 – 19, Atlanta, GA, USA, 2009

[Kurgan and Cios, 2004]
L. Kurgan, K.J. Cios, CAIM discretization algorithm, IEEE Transactions on Knowledge and Data Engineering 16 (2) (2004) 145–153.

[Liaad, 2012]
Retrieved Feb 11, 2012, from http://www.liaad.up.pt/~ltorgo/Regression/DataSets.html.

[Li, Salman, Test, Strack and Kecman, 2011]
Li Q., Salman R., Test E., Strack R., Kecman V., Parallel Multi-task Cross Validation for Support Vector Machine Using GPU, Journal of Parallel and Distributed Computing, Elsevier, DOI: 10.1016/j.jpdc.2012.02.011, to appear, 2012

[Li, Salman, Test, Strack and Kecman, 2012]
Li Q., Salman R., Test E., Strack R., Kecman V., GPUSVM: A Comprehensive CUDA Based Support Vector Machine Package, Cent. Eur. J. Comp. Sci., Vol.1, No.1, pp. 1-22, 2011

[McQueen, 1967]
J. B. McQueen, "Some methods of classification and analysis in multivariate observations," in *Proc. Of fifth Barkley symposium on mathematical statistics and probability*, pp. 281 - 297, 1967.

[Ng and Han, 1994]
R. Ng and J. Han, 'Efficient and Effective Clustering Methods for Spatial Data Mining', Proceedings of International Conference on Very Large Data Bases, Santiago, Chile, Sept. 1994, pp.144−155.

[Pakhira, 2009]
Pakhira, Malay K.: "A Modified *k*-means Algorithm to Avoid Empty Clusters". International Journal of Recent Trends in Engineering, Vol. 1, No. 1, May (2009).

[Princeton, 2012]
Retrieved Feb 11, 2012, from http://data.princeton.edu/wws509/datasets/#phd.

[Quinlan, 1993]
Quinlan, J., 'Combining instance-based and model-based learning', International Conference on Machine Learning, 1993, pp. 236-243

[Salman et al., a-2011]
Salman R., Kecman V., Li Qi, Strack R. and Test E., "Two-Stage Clustering with *k*-means Algorithm," WIMO 2011 Conference, Ankara, Turky, June 2011.

[Salman and Kecman, 2011]
Salman R., Kecman V., "The Effect of Cluster Location and Dataset Size on 2-Stage *k*-means Algorithm", 10th IEEE workshop on Electronics, Control, Measurement and Signals 2011, June 1-3, 2011

[Salman et al., b-2011]
Salman R., Kecman V., Li Qi, Strack R. and Test E., 'FAST *K*-MEANS ALGORITHMCLUSTERING', International Journal of Computer Networks & Communications (IJCNC) Vol.3, No.4, July 2011

[Salman and Kecman, 2012]
Raied Salman and Vojislav Kecman,'Regression as Classification', IEEE Southeast Con 2012 conference, Orlando Florida, March 2012

[Steinwart 2011]
I. Steinwart, Personal communication, 2011

[Uci, 2012]
Retrieved Jan 10, 2012, from http://www.liaad.up.pt/~ltorgo/Regression/DataSets.html.

[Vapnik, 1995]
Vapnik, V. N. (1995). The Nature of Statistic Learning Theory, Springer-Verlag, New York.

[Wu, 2008]
Wu, F. X.: "Genetic weighted *k*-means algorithm for clustering large-scale gene expression data". BMC Bioinformatics, vol. 9, (2008).

[Yang and Kecman 2008]
T. Yang, V. Kecman, "Adaptive Local Hyperplane Classification", *Neurocomputing*, 71, pp.3001-3004, 2008.

[Yang and Kecman 2010]
T. Yang and V. Kecman, Machine Learning by Adaptive Local Hyperplane Algorithm: Theory and Applications, VDM-Verlag, Saarbrücken, Germany, 2010

[Yeh 1998]
I-C. Yeh, "Modeling of strength of high performance concrete using artificial neural networks", *Cement and Concrete Research*, Vol. 28, No. 12, pp. 1797-1808, 1998.

[Zhang, Ramakrishnan and Livny, 1996]
Zhang, T., Ramakrishnan, R. and Livny, M., 'BIRCH: An efficient data clustering method for very large databases', In Proceedings of the Fifteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pp. 103-114. 1996, ACMPress.

Vitae

Raied Salman received his first Ph.D. from Brunel University (England / UK) in Electrical Engineering/Control Engineering in 1989. He received a Bachelor's degree in Electrical Engineering and a Master's degree in Systems Engineering from The University of Technology (Baghdad / Iraq). He also received a Diploma in Teaching from Auckland University (Auckland/ New Zealand), majoring in Information Technology. He received his second Ph.D. in Computer Science, from Virginia Commonwealth University (Richmond, Virginia, U.S.A.), in the area of data mining.

Dr Salman has many years of teaching experience in the Computer Science and Information Technology fields. His research interests include Machine Learning, Data Mining, Support Vector machines, Clustering, Classification, and Artificial Intelligent.