Instance selection with threshold clustering for support vector machines

by

Tahany Basir

M.S., King Abdulaziz University, KSA, 2012

AN ABSTRACT OF A DISSERTATION

submitted in partial fulfillment of the
requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Statistics
College of Arts and Sciences

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2023

Abstract

Tremendous advances in computing power have allowed the size of datasets to grow massively. Many machine learning approaches have been developed to deal with massive data by reducing the number of features, observations, or both. Instance selection (IS) is a data mining process that relies on scaling down the number of observations of a dataset. In this research, we focus on IS methods that rely on clustering algorithms, particularly, on threshold clustering (TC). TC is a recent efficient clustering method. Given a fixed size threshold t^*, TC forms clusters of t^* or more units while ensuring that the maximum within-cluster dissimilarity is small. Unlike most traditional clustering methods, TC is designed to form many small clusters of units, making it ideal for IS.

Support vector machines (SVM) is a powerful method for classification. However, training SVM may be computationally infeasible for large datasets—training SVM requires $O(N^3)$ runtime, where N is size of the training data. In this dissertation, we propose a method for IS for training SVM under big data settings called support vector machines with threshold clustering (SVMTC). Our proposed method begins by clustering each class in the training set separately using TC. Then, centroids of all clusters are formed the reduced set. If the data reduction is insufficient, TC may be repeated. SVM is then applied on the reduced dataset. In this way, our proposed method can reduce the training set for SVM by factor $(t^*)^r$ or more, where r is the number of iterations of TC, dramatically reducing the runtime required to train SVM. Furthermore, we prove under the Gaussian radial basis kernel, that the maximum distance between the Gram matrix for the original data—which is used to find support vectors—and the Gram matrix for the reduced data is bounded by a function of the maximum within-cluster distance for TC. Then, we show, via simulation

and application to datasets, that SVMTC efficiently reduces the size of training sets without sacrificing the prediction accuracy of SVM. Moreover, it often outperforms competing methods for IS in terms of the runtime, memory usage, and prediction accuracy.

Next, we explore best practices for applying feature reduction methods for SVMTC when the number of features is large. We investigate the usefulness of various feature selection and feature extraction methods, including principal component analysis (PCA), linear discriminant analysis (LDA), LASSO, and Fisher Scores, as an initial step of SVMTC. For feature reduction methods that select a linear combination of the original features—for example, PCA—we also investigate forming prototypes using the original features or the transformed features. We compare, via application to datasets, the performance of SVMTC under feature reduction methods. We find that LASSO tends to be an effective feature selection method, and overall, show that SVMTC is improved significantly under the proposed methods.

Finally, we perform a comparative study of iterative threshold instance selection (ITIS) and other IS methods. ITIS is a recent extension method of TC that is used as IS. We use simulation to compare between ITIS and competing methods. The results illustrate that ITIS is effective in massive data settings when compared against other instance selection methods like k-means and its variations. In addition, we demonstrate the efficacy of hybrid clustering algorithms that utilize ITIS as an initial step, and show via simulation study that these methods outperform other hybrid clustering methods in terms of runtime and memory without sacrificing performance.

Instance selection with threshold clustering for support vector machines

by

Tahany Basir

M.S., King Abdulaziz University, KSA, 2012

A DISSERTATION

submitted in partial fulfillment of the
requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Statistics
College of Arts and Sciences

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2023

Approved by:

Major Professor
Michael Higgins

Copyright

Abstract

Tremendous advances in computing power have allowed the size of datasets to grow massively. Many machine learning approaches have been developed to deal with massive data by reducing the number of features, observations, or both. Instance selection (IS) is a data mining process that relies on scaling down the number of observations of a dataset. In this research, we focus on IS methods that rely on clustering algorithms, particularly, on threshold clustering (TC). TC is a recent efficient clustering method. Given a fixed size threshold t^*, TC forms clusters of t^* or more units while ensuring that the maximum within-cluster dissimilarity is small. Unlike most traditional clustering methods, TC is designed to form many small clusters of units, making it ideal for IS.

Support vector machines (SVM) is a powerful method for classification. However, training SVM may be computationally infeasible for large datasets—training SVM requires $O(N^3)$ runtime, where N is size of the training data. In this dissertation, we propose a method for IS for training SVM under big data settings called support vector machines with threshold clustering (SVMTC). Our proposed method begins by clustering each class in the training set separately using TC. Then, centroids of all clusters are formed the reduced set. If the data reduction is insufficient, TC may be repeated. SVM is then applied on the reduced dataset. In this way, our proposed method can reduce the training set for SVM by factor $(t^*)^r$ or more, where r is the number of iterations of TC, dramatically reducing the runtime required to train SVM. Furthermore, we prove under the Gaussian radial basis kernel, that the maximum distance between the Gram matrix for the original data—which is used to find support vectors—and the Gram matrix for the reduced data is bounded by a function of the maximum within-cluster distance for TC. Then, we show, via simulation

and application to datasets, that SVMTC efficiently reduces the size of training sets without sacrificing the prediction accuracy of SVM. Moreover, it often outperforms competing methods for IS in terms of the runtime, memory usage, and prediction accuracy.

Next, we explore best practices for applying feature reduction methods for SVMTC when the number of features is large. We investigate the usefulness of various feature selection and feature extraction methods, including principal component analysis (PCA), linear discriminant analysis (LDA), LASSO, and Fisher Scores, as an initial step of SVMTC. For feature reduction methods that select a linear combination of the original features—for example, PCA—we also investigate forming prototypes using the original features or the transformed features. We compare, via application to datasets, the performance of SVMTC under feature reduction methods. We find that LASSO tends to be an effective feature selection method, and overall, show that SVMTC is improved significantly under the proposed methods.

Finally, we perform a comparative study of iterative threshold instance selection (ITIS) and other IS methods. ITIS is a recent extension method of TC that is used as IS. We use simulation to compare between ITIS and competing methods. The results illustrate that ITIS is effective in massive data settings when compared against other instance selection methods like k-means and its variations. In addition, we demonstrate the efficacy of hybrid clustering algorithms that utilize ITIS as an initial step, and show via simulation study that these methods outperform other hybrid clustering methods in terms of runtime and memory without sacrificing performance.

Contents

List of Figures

List of Tables

Acknowledgments

First, I would like to thank Allah because this dissertation would not be accomplished without him. My heartfelt gratitude and appreciation go to my great advisor, Dr.Michael Higgins. He supported and guided me with his wealth of knowledge and experience. I appreciate his patience, motivation, and enthusiasm during working in this research. My sincere appreciation also goes to the supervisory committee— Dr.Juan Du, Dr.Perla Reyes and Dr.Nathan Albin— for providing feedback and insights to enhance this work.

I would like to express my love and special thanks to my mother, who is always there to help because she wants nothing more than for me to succeed in my life. She always encourages me to pursue my studies and reach for more. No words can explain my grief over losing my father during my work on this dissertation. He supported and encouraged me until his last moments. It was his wish for me to earn a doctorate degree and fulfilling this has brought me here.

My heartfelt thanks to my husband, Wafi, and my lovely children, Talah and Ahmed, for their understanding and endless endurance through the duration of my study. I appreciate their patience and willingness to stay abroad for many years. Gratitude also goes to my darling siblings Doa'a, Amany, Mohammed, and Hana'a; my nephews and nieces; my uncle Kamel, and Wafi's family who motivated me to complete my dissertation.

Special thanks to all my friends in the United States and Saudi Arabia, and all the members of the department of statistics at Kansas State University and everyone who supported me. Also, I thank King Abdul-Aziz University that provided me this opportunity to study abroad.

Dedication

This dissertation is dedicated to the greatest person in my life, my darling late father, Ahmad Basir. And also, to my dear mother and to my lovely family.

Chapter 1

Introduction

1.1 Introduction

Advances in computing power have allowed the size of datasets to grow exponentially. Billions of people use the internet for many reasons: making purchases, contacting each other, or sharing files or photos, for example, Beaver et al. (2010) showed that approximately 1 million photos per second are posted on Facebook. Data mining (Fayyad et al., 1996) is essential to deal with such a large amount of data. It is based on extracting valid and understandable patterns of data. Data selection and preprocessing are the core of data mining process. Scaling down the data by selecting relevant and useful patterns of data and eliminating redundant ones is an important data mining process (Liu and Motoda, 2002).

Classification—a supervised learning technique—aims to obtain a function that is an optimal approximation of the class label for any unseen data point. The classification algorithm is trained on a training set to obtain the optimal classifier. The training set is usually massive and consists of superfluous instances that may be removed to improve the performance of classification methods.

In general, data can be reduced by feature selection methods, which depend on reducing the number of columns by eliminating ineffective variables from a dataset. On the other

hand, instance selection (IS) methods rely on scaling down the data size by reducing the number of observations. The goal of IS is that the performance of the machine learning algorithm when using the selected data should to be as close as possible to the performance of the original data, but with less runtime, and memory usage. Methods for IS include sampling, classification, and clustering algorithms. The main idea of sampling is to draw a subset of data randomly in which each instance has the probability to be selected. Simple random sampling, stratified random sampling, and adaptive sampling are examples of IS by sampling. When data is labeled, IS methods that are associated with classification can be used. In contrast, IS methods related to clustering are used when labels of data are unknown.

Support vector machines (SVM) is a most popular classification method. The goal of SVM is to minimize the empirical classification error and simultaneously maximize the margins in which the support vectors of each class are placed on the margins. SVM obtains a hyperplane that forms the maximum margins between classes. A soft margin is used to overcome class overlapping issues. By introducing slack variables and a regularized parameter, soft margin SVM results in misclassification of some instances.

SVM is limited for small data settings because of its complex computations for optimization, requiring $O(N^3)$ runtime, where N is the number of units. Hence, many methods have been proposed to accelerate the computations of SVM in massive data. Some methods are developed by changing the model, such as twin SVM (TSVM) (Khemchandani et al., 2007) and least squares SVM (LSSVM) (Suykens and Vandewalle, 1999). Other methods work directly on the optimization problem, such as chunking (Cortes and Vapnik, 1995), decomposition (Osuna et al., 1997a), and sequential minimal optimization (SMO) (Platt, 1998) methods. The last type of method to accelerate SVM is IS that is applied to a select subset of data from the training set that are more likely to be support vectors; or to extract pseudo instances as a subset of data to reduce runtime and memory usage.

Many clustering methods for IS have been developed. Threshold clustering (TC) is a

recent clustering method that works efficiently in big data settings (Higgins et al., 2016). It aims to minimize the maximum within-cluster dissimilarity. Unlike most traditional clustering methods, TC is designed to form many small clusters of units, making it ideal for IS. In this research, we use TC as IS to accelerate training SVM in big data settings. Our proposed methods improve training SVM and outperform other IS methods that based on traditional clustering methods.

1.2 Instance Selection

Data have been grown rapidly with development of technology. Working with original data directly cannot keep up with this rapid growth. Data mining (Fayyad et al., 1996) becomes important to work with this huge data. Instance selection (IS) is a preprocessing method that is used to reduce the size of data by extracting a subset that contains only relevant instances from the training set. IS enables state-of-the-art algorithms to work effectively in massive datasets. In addition, IS reduces the execution time and memory of training data by scaling down the size of the data. Evaluation of IS methods are based on the performance of machine learning algorithms under the selected set. The IS method is accurate when the performance of the algorithm of the reduced set under IS method is approximately the same as the performance of the algorithm using the original data.

IS methods are associated with sampling, classification, or clustering approaches(Liu and Motoda, 2002). IS methods are classified in many ways, including wrapper and filter methods. Instances in wrapper methods are selected if they impact the classification accuracy; otherwise, they are removed. Filter methods only consider selection functions (Olvera-López et al., 2010a). Others classify IS methods into noise filters, condensation algorithms, and prototypes selection methods (Grochowski and Jankowski, 2004). In this research, we focus on prototypes selection methods under clustering approach.

1.2.1 Literature Review on Instance Selection

Several instance selection (IS) methods that are associated with classification methods have been developed. The earliest one was based on the nearest neighbor (NN) rule, proposed by Cover and Hart (1967). This type of classification uses large storage space especially with enormous data, and it is unable to manage noisy instances. Hart (1968) proposed condensed nearest neighbor (CNN) that overcomes the storage limitation in the NN rule. The algorithm begins by randomly choosing one instance for each class from training set T. The initial instances are stored in the reduced set $T^{'}$. All instances in T are classified based on instances in $T^{'}$. Misclassified instances are added to the reduced set $T^{'}$. Generalized condensed nearest neighbor (GCNN)—similar to CNN— is proposed in Chou et al. (2006). However, GCNN improves the way of obtaining prototypes by using robust criteria of absorption. For an instance $\mathbf{x}$, the distance between $\mathbf{x}$ and its nearest neighbor, and the distance between $\mathbf{x}$ and its nearest instance from different class are computed. Instances with strong absorption are the selected data. In Ritter et al. (1975), CNN is enhanced based on the selective nearest neighbor (SNN) rule. SNN is carried out by obtaining a selective subset in which each instance in original dataset T is closer to an instance in the same class in the selective set than any instance in T. On the other hand, Wilson (1972) proposed the edited nearest neighbor (ENN) method, which overcomes the limitation of CNN by getting rid of noisy instances to improve classification accuracy. The K nearest neighbors (KNN) are obtained for each instance in the preclassified samples. A class label with largest number of instances through KNN is found. Any instance that does not agree with majority label of its KNN is discarded. However, this method retains interior instances while removing border ones. Also, it is unable to reduce several instances in the training set compared with other IS methods.

In Aha et al. (1991), instance-based (IB) methods were developed. The simple algorithm is IB1, which works as NN rule; however, IB1 normalizes the variables. IB1 is used as a starting point for IB2 and IB3 algorithms. The IB2 algorithm starts by saving instances in

the original set T that are misclassified by instances in the reduced set $T^{'}$. Border datapoints in reduced set $T^{'}$ are saved whereas instances in the middle of dataset of the same class are discarded. IB3 was developed to overcome noise sensitivity of IB2.

Decremental reduction optimization procedure (DROP1-DROP5) are proposed as IS methods in Wilson and Martinez (2000). These procedures rely on associates of an instance $\mathbf{x}$ in which $\mathbf{x}$ is one of their k-nearest neighbors (KNNs). For an instance $\mathbf{x}$, DROP1 begins by obtaining the KNNs of associates of $\mathbf{x}$. Then, the accuracy of KNN of $\mathbf{x}$'s associates is checked after removing $\mathbf{x}$. If the accuracy was not affected, $\mathbf{x}$ is removed. DROP1 can remove noisy instances; however, a noisy instance may not be removed if its neighbors are first removed. DROP2 overcomes the drawback of DROP1: the instance is removed only if its associates in training set T are classified correctly. DROP3 and DROP4 use filtering to remove noisy instances first, then they apply DROP2. DROP5 is relied on DROP2, but it begins by deleting nearest instances of different class.

Brighton and Mellish (2002) introduced the iterative case filtering (ICF) algorithm. This method is based on removing superfluous instances with two properties: reachability and coverage. The reachable set of instance $\mathbf{x}$ contains instances that are from the same class of $\mathbf{x}$ in the largest hypersphere centered in $\mathbf{x}$. In contrast, the coverage set of $\mathbf{x}$ includes all instances that $\mathbf{x}$ is adapted to, practically, the instances in which $\mathbf{x}$ is in their reachable sets. The algorithm begins by performing Wilson Editing (Wilson and Martinez, 1997) to remove noisy instances. The rule of ICF is that if reachable set ($\mathbf{x}$) size is larger than coverage set ($\mathbf{x}$) size, $\mathbf{x}$ is flagged to removal. At the end, all flagged instances are removed.

On the other hand, clustering is a machine learning method that works with unlabeled data. Liu and Motoda (2002) and Spillmann et al. (2006) suggested IS methods associated with clustering. The general idea is about generating pseudo points from clusters called prototypes; the prototypes are used as selected data. K-means (Lloyd, 1982) is a well-known clustering method that splits the data into k clusters. A set of k data points is selected randomly to be initial centers and nearest units to each center are specified to

form the initial clusters. The nearest points are usually assigned by squared Euclidean distance. The mean of each cluster is computed, then the k centers are replaced with each mean. The process is repeated until the convergence is obtained. From the IS perspective, k means can be used and the rest of data ignored. For N sample size, m variables for each instance, and i iterations of algorithm to be converged, time and space complexities of k-means are $O(Nkmi)$ and $O((k+N)m)$, respectively. On the other hand, scalable k-means is an extension of the k-means algorithm in which clustering can be constructed in one scan of data (Bradley et al., 1998). The idea of scalability is to retain some parts of dataset and summarize the others. In big data settings, the algorithm can be applied effectively because it requires a small space of memory to save small sub-samples. IS by Clustering (CLU) is proposed in Lumini and Nanni (2006), which was based on the same idea. This method applies fuzzy C-means to partition templates into clusters, then centers of clusters are used as prototypes.

Another method that depends on clustering is established in Olvera-López et al. (2010b). This method selects both border and interior data points. It relies on splitting T into clusters based on clustering algorithm so that searching on border data points in small clusters is more effective than searching on the whole T. For a cluster that consists of data points of the same class, the mean of the cluster is calculated, and the cluster is replaced by the nearest prototype to that mean. On the other hand, if a cluster contains data points from different classes, finding the border data points should be done by searching on the majority class in the cluster. In this type of cluster, the nearest data points that belong to each non-majority class are border data points in the majority class. Likewise, border data points of non-majority classes are the nearest data points that are belonged to the majority class.

Squashing is another technique based on clustering that compresses original data without losing any statistical information. The process passes through grouping, momentizing, and generating (GMG) sequentially (DuMouchel et al., 1999). Data is grouped into pins based on the type of variables. For qualitative variables, pins can be assigned directly; however,

for continuous ones, the regions are created by obtaining quantiles of each variable or by inducing categorical values on such set of variables. Within each region, moments of the datapoints in that region are computed. Based on the degree of Taylor series approximations, the number of moments in each region is determined. For each region, squashed data consisting of pseudo datapoints is generated.

Gong et al. (2021) proposed an IS method for KNN rules in which neighbors of each instance are determined at the beginning. Then, for each instance, neighbors determine a piece of evidence of estimation label of this instance. The pieces are merged to compute the conflicts, so that the instance is considered to be near boundary if its conflict is high. At the end, the optimal problem is solved to select boundary instances.

1.3 Threshold Clustering

Threshold clustering (TC) (Higgins et al., 2016) is a recent clustering method which was developed at first for making statistical blocking of big data. This method aims to minimize the maximum distances between any two instances within the same cluster by requiring the minimum number of instances t^* in each cluster. Outside of $(t^* - 1)$-nearest neighbor formulation, and N is data size, the TC algorithm requires $O(t^*N)$ time to be executed in low dimensional settings, which is smaller than other traditional clustering methods such as k-means. This property of the algorithm allows TC to work efficiently in massive data. In general, K-nearest neighbor (KNN) construction is computationally expensive for high-dimensional data. Hence, TC may work better in low-dimensional data because KNN could be executed in $O(KN \log N)$ time (Friedman et al., 1977). Additionally, according to a bottleneck objective (Hochbaum and Shmoys, 1986), TC ensures obtaining approximately optimal clustering. Consider a set that contains all threshold clusterings $\mathbf{B}(t^*)$ where the clusterings are $\mathbf{v}$ such that, for each cluster $V_l \in \mathbf{v}$, then $|V_l| \geq t^*$. Bottleneck threshold partitioning problem (BTPP) is obtaining the optimal clustering that minimizes the maxi-

mum dissimilarity within a cluster. That is, the objective of BTPP is to obtain an optimal clustering $\mathbf{v}^* \in \mathbf{B}(t^*)$ such that

$$\max_{ij \in E(\mathbf{v}^*)} d_{ij} = \min_{\mathbf{v} \in \mathbf{B}(\mathbf{t}^*)} \max_{ij \in E(\mathbf{v})} d_{ij} \equiv \lambda \tag{1.1}$$

where λ is the optimal value of the maximum dissimilarity of units within a cluster, and d_{ij} is dissimilarity between units i and j. The most common dissimilarities are Euclidean distance and Manhattan distance which are defined as follows. For a p-dimensional vector $\mathbf{x}_i = (x_{i1}, \ldots, x_{ip})$, Euclidean distance of each pair of data points i and j is

$$d_{ij} = \sqrt{(x_{i1} - x_{j1})^2 + \ldots + (x_{ip} - x_{jp})^2} = \|\mathbf{x}_i - \mathbf{x}_j\|_2 \tag{1.2}$$

and Manhattan distance of each pair of data points i and j is

$$d_{ij} = |x_{i1} - x_{j1}| + \ldots + |x_{ip} - x_{jp}| = \|\mathbf{x}_i - \mathbf{x}_j\|_1 \tag{1.3}$$

Given a prespecified threshold parameter t^* and graph $G = (V, E)$ - where V is the set of instances, and E is the set of edges that connect pairs of instances, TC process is shown in Algorithm I.

Algorithm I:

1. According to d_{ij}, construct (t^*-1)-nearest neighbor (t^*NN) subgraph $G_{t^*NN} = (V, E_{t^*NN})$.
2. From $G^2_{t^*NN}$, the second power of the $(t^* - 1)$-nearest neighbor subgraph, obtain a maximal independent set of vertices (seeds), $\mathbf{M}$.
3. For each seed $l \in \mathbf{M}$, construct a cluster V_l that includes all instances in G_{t^*NN} that are neighboring to the seed l.
4. If there is any unassigned instance, assign it to a cluster that contains at least one of its neighbor instances in G_{t^*NN}.

At the end, TC is formed by the set of clusters $\mathbf{v} = \{V_l\}$, $l \in \mathbf{M}$. An illustration of performing TC is shown in Figure 1.1.

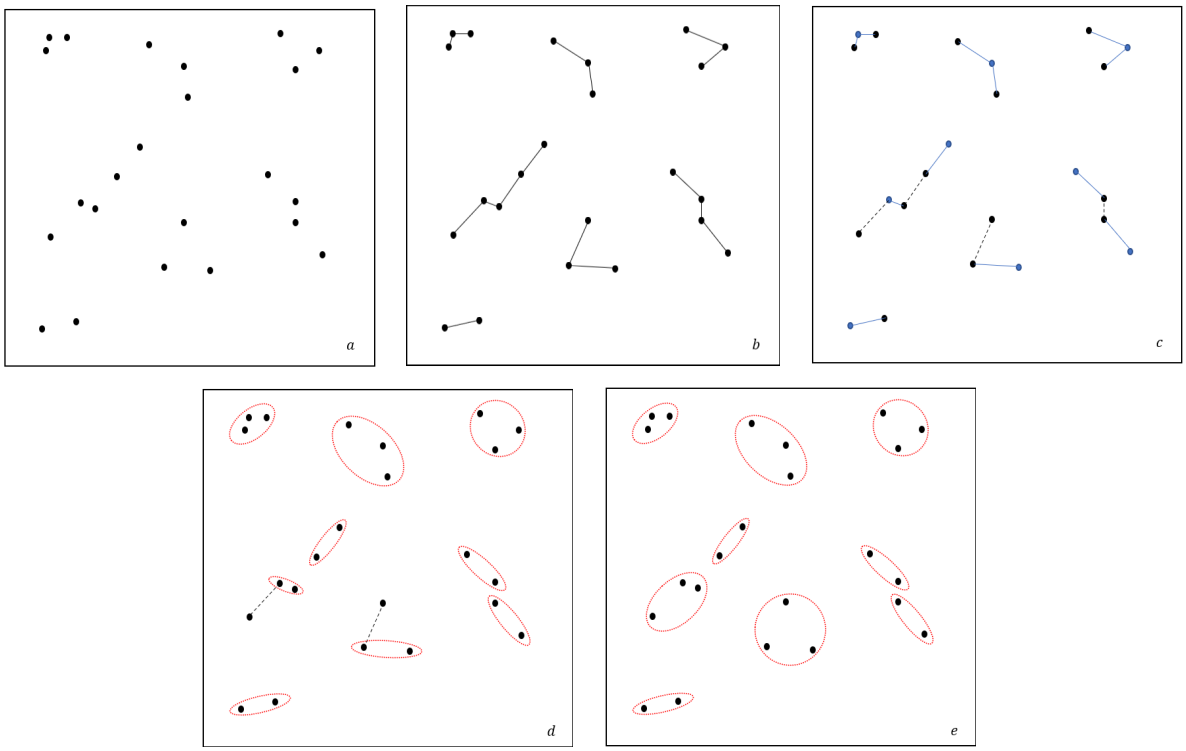

Figure 1.1: *An explanation of threshold clustering as shown in Algorithm I with data of size $N = 23$, and threshold parameter $t^* = 2$. Each datapoint is shown as a black vertex in (a). A 1-nearest neighbors subgraph is found in (b). In (c), seeds which are represented by blue circles are shown. In (d), each seed is grouped with its adjacent vertices making a cluster. In (e), each unassigned vertex is assigned to a cluster that have its nearest seed. Threshold clustering is formed with 9 clusters in which each cluster contains at least $t^* = 2$ datapoints.*

Lemma 1. *There is no distance between two data points in G_{t^*NN} that is greater than the*

maximum distance in the optimal clustering (Higgins et al., 2016),

$$d_{ij} \leq \lambda, \forall ij \in E_{t^*NN}, \tag{1.4}$$

where $G_{t^*NN} = (V, E_{t^*NN})$ *is a subgraph of graph* G *such that* $ij \in E_{t^*NN}$ *only if* i *and* j *are in the same cluster.*

Theorem 1. *Algorithm I is a 4-approximation algorithm, (Higgins et al., 2016).*

$$\max_{ij \in E(\mathbf{c}_{tc})} d_{ij} \leq 4\lambda \tag{1.5}$$

Proof. Assume the clustering obtained from TC algorithm is $\mathbf{c}_{tc}$ in which any within-cluster $ij \in E(\mathbf{c}_{tc})$.

First, from Lemma 1, for any $ij \in E_{t^*NN}$, there exist k in which $ik, kj \in E_{t^*NN}$ so $d_{ij} \leq \lambda$, where E_{t^*NN} obtained from step 1 of TC algorithm.

Now, consider $ij \notin E_{t^*NN}$ in which i is not a seed, but j is the seed, then $d_{ij} \leq d_{ik} + d_{kj} \leq \lambda + \lambda = 2\lambda$.

Last, consider $ij \notin E_{t^*NN}$, but not i nor j is a seed, let the seed in the cluster includes i and j is k. From above, it is shown that $d_{ik}, d_{kj} \leq 2\lambda$, then $d_{ij} \leq d_{ik} + d_{kj} \leq 2\lambda + 2\lambda = 4\lambda$. □

1.3.1 Iterative Threshold Instance Selection

Luo et al. (2019) introduced a novel IS method based on threshold clustering (TC) (Higgins et al., 2016) called iterative threshold instance selection (ITIS). The main point of ITIS is its ability to scale down the size of data by factor of α. The process of ITIS passes through three main steps.

1. With respect to a prespecified threshold t^*, implement TC on data of size N to perform N^* clusters, where each cluster contains at least t^* of instances.

2. Compute the prototype (centroid or medoid) for each of N^* clusters, so there are N^* prototypes.

3. The process is terminated if the data is sufficiently scaled down by a factor of α, otherwise, the data of size N is replaced with N^*, then go to the first step.

ITIS works effectively in a massive data with runtime $O(t^* mN \log N)$ where, m is the number of iterations. However, it has a limitation when sample size is small in which the similarity between each prototype and its data points becomes small.

1.4 Support Vector Machines

Support vector machines (SVM) (Cortes and Vapnik, 1995; Vapnik, 1998) is a well-known supervised learning method that is used for classification. Because of its efficiency, several researchers have been concerned by SVM in many areas such as text recognition (Joachims, 1998), pattern recognition such as face detection (Osuna et al., 1997b), and diseases diagnosis (Bhatia et al., 2008). SVM can classify unseen instances by first obtaining the optimal hyperplane among many potential hyperplanes that split the training data based on class label. The optimal hyperplane maximizes the distance between the closest instances of each class. Based on class label, training data can be separated by a hyperplane linearly or non-linearly.

Consider an N d-dimensional training set $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)$, in which $\mathbf{x}_i \in \mathbb{R}^d$ is an input vector, and $y_i \in \{-1, +1\}$ is an output variable where the joint distribution of $\mathbf{x}_i$ and y_i is unknown. The support vector classifier aims to obtain the following optimal decision hyperplane that separates data into two classes to predict a class of new unseen instance y given the input $\mathbf{x}$.

$$f(\mathbf{x}) : \mathbf{w} \cdot \mathbf{x} + b = 0 \tag{1.6}$$

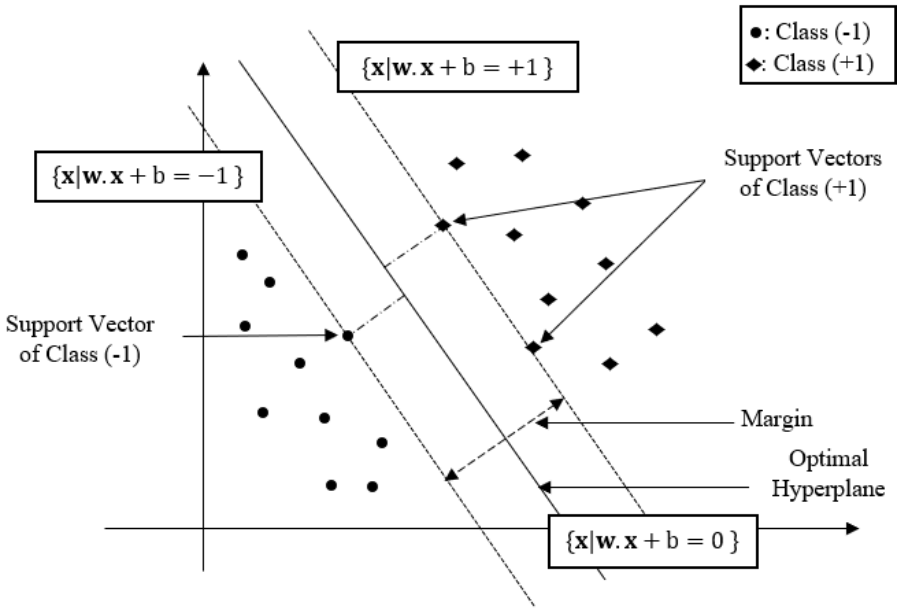

Figure 1.2: *An example of data in a two-dimension space is linearly separated by the optimal hyperplane using SVM.*

where $\mathbf{w} \in \mathbb{R}^d$ is a weight vector orthogonal on the hyperplane and $b \in \mathbb{R}$ is a scalar parameter. The training data is supposed to satisfy the following conditions

$$\begin{cases} \mathbf{w} \cdot \mathbf{x} + b \geq 1, & \text{if } y = +1. \\ \mathbf{w} \cdot \mathbf{x} + b \leq 1, & \text{if } y = -1. \end{cases} \tag{1.7}$$

Equivalently,

$$y(\mathbf{w} \cdot \mathbf{x} + b) \geq 1 \tag{1.8}$$

The margin, which is the distance between the hyperplane and the closest instance of each class, can be computed by $\frac{2}{\|\mathbf{w}\|}$. There are infinite hyperplanes that separate the data; however, we need to obtain the optimal separating hyperplane —the maximal margin classifier— by minimizing $\|\mathbf{w}\|$. By transforming $\|\mathbf{w}\|$ to quadratic optimization problem, the maximal margin classifier is a solution from minimizing the following quadratic programming (QP)

problem

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \tag{1.9}$$

subject to

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, i = 1, 2, \ldots, N \tag{1.10}$$

The optimization problem in equation (1.9) is a QP problem because the optimization objective is quadratic and its constraints are linear. To simplify the computations, equation (1.9) is transformed from constrained problem into an unconstrained problem by using a Lagrangian problem as follows

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{N} \alpha_i[y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1] \tag{1.11}$$

where $(\alpha_1, \ldots, \alpha_N) \geq 0$ are the Lagrange multipliers. To transform the optimization problem from primal form into dual form, Karush-Kuhn-Tucker (KKT) conditions (Kuhn and Tucker, 2014) are applied into in (1.11), the minimization over $\mathbf{w}$ is

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i \tag{1.12}$$

and minimization over b is

$$\sum_{i=1}^{N} \alpha_i y_i = 0 \tag{1.13}$$

According to KKT conditions of optimization, α_i, the following condition should be satisfied

$$\alpha_i(y_i(\mathbf{w}\cdot\mathbf{x}+b)-1)=0, i=1,2,\ldots,N \tag{1.14}$$

By substituting equation (1.12) in equation (1.11) and based on (1.13), the optimization problem in equation (1.11) can be transformed into a dual form by maximizing the following objective function with respect to α_i

$$Q(\boldsymbol{\alpha})=\sum_{i=1}^{N}\alpha_i-\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j \mathbf{x}_i\cdot\mathbf{x}_j, \tag{1.15}$$

subject to

$$\sum_{i=1}^{N}\alpha_i y_i=0$$

$$\alpha_i\geq 0, i=1,2,\ldots,N$$

Instance $\mathbf{x}_i$ corresponds to non-zero Lagrange multipliers that satisfy $y_i(\mathbf{w}\cdot\mathbf{x}_i+b)=1$ and are called support vectors. Hence, the decision function can be written as

$$f(\mathbf{x}^*)=\text{sign}\left(\sum_{j=1}^{n}\alpha_j y_j\mathbf{x}_j\cdot\mathbf{x}^*+b\right) \tag{1.16}$$

where $\mathbf{x}_j, j=1,2,\ldots,n$ are the support vectors and n is the cardinality of support vectors set, and the sign function takes the values $-1, 0$ or 1 if the variable is $(<0, =0$ or $>0)$ respectively.

Because not all data can be linearly separated by a hyperplane, Cortes and Vapnik (1995) proposed modifications in the optimization problem to obtain the hyperplane that separate the data in this case. A soft margin classifier is used to relax the constraints in (1.10), in which some instances are allowed to be violated under a control (regularized) parameter.

Thus, by introducing slack variables $\xi_i \geq 0, i = 1, 2, \ldots, N$, the optimization problem can be rewritten as

$$\min_{\mathbf{w},b,\xi_i} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N} \xi_i, \tag{1.17}$$

subject to

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i, i = 1, 2, \ldots, N \tag{1.18}$$

$$\xi_i \geq 0, i = 1, 2, \ldots, N \tag{1.19}$$

where $\xi_i > 0$ hold for misclassified instances, hence the penalty term $\sum_{i=1}^{N} \xi_i$ can compute the total misclassifications of the model which is called training error (or empirical error), and C is the penalty parameter of soft margin to control the balance between the empirical error and margin. Thus, the goal of the objective function (1.17) is maximizing the margin and minimizing the training error. As in the separable case, the quadratic optimization problem (1.17) can be solved by Lagrangian problem as follows.

$$L(\mathbf{w}, b, \alpha, \nu, \xi) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{N} \alpha_i[y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 + \xi_i] + C\sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \nu_i\xi_i \tag{1.20}$$

where $\alpha_i \geq 0$ and $\nu_i \geq 0$ are the Lagrange multipliers in which (1.20) maximizes with respect to α_i and ν_i, and minimizes with respect to $\mathbf{w}$, b and ξ_i. Now, minimization over $\mathbf{w}$ and b provides the same conditions in (1.12) and (1.13), and the minimization over ξ_i yields the new condition

$$\alpha_i + \nu_i = C \tag{1.21}$$

Hence, according to $\nu_i \geq 0$, we have

$$0 \leq \alpha_i \leq C \tag{1.22}$$

Thus, the dual form of problem (1.20) is

$$\max_{\alpha_i} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \tag{1.23}$$

subject to

$$\sum_{i=1}^{N} \alpha_i y_i = 0$$

$$0 \leq \alpha_i \leq C, i = 1, 2, \ldots, N$$

where α_i are Lagrange multipliers. Now, there are three types of data points. Instances that correspond to $\alpha_i = 0$ are ignored from the decision function. When $0 < \alpha_i < C$, the corresponding instances are support vectors located on the margins, with zero slack variables. When $\alpha_i = C$, support vectors are located inside the side that could be misclassified.

For a non-linear hyperplane in which data can not be separated linearly, a non-linear mapping of data is carried out from the input space $\mathbb{R}^d$ into a high dimensional feature space $\mathbb{F}$ ($\Phi : \mathbb{R}^d \rightarrow \mathbb{F}$) to obtain the optimal hyperplane linearly in a higher feature space, which could be infinite space. SVM utilizes the kernel trick (Cortes and Vapnik, 1995) to obtain the similarity between two instances by computing the dot product using kernel functions instead of mapping the data explicitly. The kernel trick uses kernel functions to obtain the similarity of the transformed data implicitly. A symmetric kernel function $\mathbf{K}(\mathbf{x}_1, \mathbf{x}_2)$ can be written as dot product expression $\mathbf{K}(\mathbf{x}_1, \mathbf{x}_2) = \phi(\mathbf{x}_1) \cdot \phi(\mathbf{x}_2)$ if the kernel function $\mathbf{K}(\mathbf{x}_1, \mathbf{x}_2)$ is positive semi-definite (Mercer's theorem). Thus, kernel function can be used without obtaining the explicit form of ϕ. The optimization problem in the dual form is modified as

follows

$$\max_{\alpha_i} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \mathbf{K}(\mathbf{x}_i, \mathbf{x}_j) \tag{1.24}$$

subject to

$$\sum_{i=1}^{N} \alpha_i y_i = 0$$

$$0 \leq \alpha_i \leq C, i = 1, 2, \ldots, N$$

The decision function will be as follows

$$f(\mathbf{x}^*) = \text{sign}\left(\sum_{j=1}^{n} \alpha_j y_j \mathbf{K}(\mathbf{x}^*, \mathbf{x}_j) + b \right) \tag{1.25}$$

where $\mathbf{K}(,)$ is *kernel* function. The most popular kernels are Gaussian radial basis function $\mathbf{K}(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2\right)$, where $\gamma = \frac{1}{2\sigma^2}$, and Polynomial function $\mathbf{K}(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + 1)^d$, where d is a polynomial degree. Kernel with a linear function leads to the linear case.

SVM for classification is originally created for binary classification problems. However, the class label in most real datasets is a multiclass. Many approaches to solve multi-class problems from binary problems have been developed. The most common methods for SVM are one-against-rest and one-against-one. For a dataset where class label includes C classes, one-against-rest (Vapnik, 1999) is based on making C binary classifiers in which each classifier is trained to discriminate one class from $C-1$ rest classes. Instances are classified by obtaining the margin of hyperplane, then the predicted class is the one with the largest margin. On the other hand, one-against-one method (Knerr et al., 1990) is based on obtaining all possible binary problems of SVM classifiers. For C class labels, $C(C-1)/2$ binary classifiers are obtained. By voting approach, the method determines the class that

occurs the most.

Solving the QP problem for SVM optimization problem requires $O(N^3)$ time complexity and $O(N^2)$ space complexity, where N is the size of the training set (Platt, 1999). Hence, for massive data, applying SVM requires complex computations that require expensive time and space. To accelerate training SVM, many techniques have been developed to accelerate the high computations. Methods like chunking work directly on the optimization problem. Chunking (Cortes and Vapnik, 1995) is one of the most popular solutions to train SVM classifiers by solving a set of small optimization problems. Decomposition (Osuna et al., 1997a) is another well-known solution to solve QP problems by partitioning the problem into smaller sub-problems. Then, smaller sub-problems are solved analytically. Sequential Minimal Optimization (SMO) is the most common solution (Platt, 1998). The idea is similar to decomposition by solving QP sub-problems analytically but of size two. Experiments illustrate that SMO can train SVM faster than chunking. The other methods to accelerate SVM depend on selecting the most important vectors to train the SVM classifier under IS approaches. The following section covers some methods of training set selection for SVM in detail.

1.4.1 Training Set Selection for SVM

To train SVM for classification, only support vectors that are setting on margins are used; others are kept in memory needlessly. To reduce overload storage and computational time, approaches that construct refined sets from massive training sets for SVM have been developed. Some methods rely on selecting support vectors candidates that are then used to construct reduced training sets. These approaches have been developed based on clustering or analyzing the geometry of data. This method can easily discard useless vectors from training sets and can be generalized to multi-class data (Lyhyaoui et al., 1999). Other methods are associated with clustering to construct prototypes, then the prototypes are used as a reduced training set.

De Almeida et al. (2000) proposed IS method that applied k-means clustering to construct the reduced set. The entire dataset is clustered into k clusters. Then, for clusters with the same class, only centers are added to the reduced set. In contrast, all instances in clusters include different classes are added to reduced set. Koggalage and Halgamuge (2004) suggested a simple method based on clustering that improves the proposed method in De Almeida et al. (2000).

Songfeng et al. (2003) improved the reduced SVM (RSVM) (Lee and Mangasarian, 2001) by using the idea of clustering training data. An unsupervised clustering algorithm (Li et al., 2001) is implemented to each class separately. After obtaining centers of each class, these centers are utilized as support vectors to solve QP problem.

Tran et al. (2003) used k-means to reduce the training set. The algorithm begins by partitioning each class in training data into clusters using k-means, then centroids of each cluster are combined and used as a reduced training set to train SVM. Wang and Xu (2004) suggested a heuristic SVM (HSVM). This method begins by initializing a similarity threshold and computing a similarity measure between every two instances. Thus, the instance with a similarity that is larger than a threshold is removed. Then, remaining instances are grouped into prespecified k groups. For each group, the closest one to the mean of the group and other instances in that group are computed. Again, any instance with a similarity to the center point that is larger than the threshold is removed. The remaining instances are used as a reduced training set to train SVM.

In Cao and Boley (2006), an approximate SVM approach based on prototypes was introduced. The idea relies on clustering data by kernel k-means, then extracting representatives from clusters to be utilized as a reduced set. Training SVM on this reduced set is faster than using the original dataset while still preserving accuracy.

Zeng et al. (2008) proposed a method called small enclosing ball SVM (SebSVM) to reduce time computations of solving QP problem. Unlike the previous methods, it extracts candidates in the feature space to avoid overload computations in the input space. Therefore,

training data should be mapped into the feature space at the beginning. Then, for each class, a ball with the smallest radius enclosing the training data is formed in the feature space. For each ball, convex hull (CH) instances that are in boundary are determined. At the end, the reduced training set involves these CH instances.

In Chau et al. (2013), the convex-concave hull idea is explored to reduce the training set for SVM. The process begins by obtaining a CH for each class separately. For each edge of CH, concave hulls are obtained based on computing the angle and number of KNN of each vertices of each edge to finally form a convex-concave hull (CCH). Instances positioned in the boundary of CCH are selected in the reduced training set.

K-means clustering is used in Shen et al. (2016) to reduce a training set. The entire training data is clustered using k-means with a prespecified k. Clusters are categorized as homogeneous clusters and heterogeneous clusters. Some of homogeneous ones are removed by using a max-min cluster distance algorithm. Heterogeneous ones are subclustered to be used along with non-removed homogeneous clusters. The distance density of each remaining cluster is computed by using Fisher's discriminant analysis (FDA) to construct a boundary between two types of data: dense (instances positioned near the centroid of cluster) and sparse (instances positioned far from the centroid). Finally, dense vectors are removed, and sparse ones form the refined set.

On the other hand, some IS methods do not rely on clustering. Zhang et al. (2008) applied the KNN method to extract support vectors. It explored instances positioned on a boundary that are more likely to be support vectors. The process begins by computing KNN (linear case) or Euclidean distance in kernel space (non-linear case) for all instances in the training set. For each instance, if at least one of its KNN is from different class, this instance is supposed to be support vector.

Guo and Boukir (2015) proposed IS method which is based on the ensemble margin. They improved their previous work (Guo et al., 2010; Guo and Boukir, 2013) by replacing the classic bagging with random forests and small votes instance selection (SVIS) to work

more effectively in big data settings. The margins of the entire training data are computed in which the lower the value of instance's margin, the more likely the instance is in decision boundary. Then, the first K instances with smallest margins are selected, where K is a prespecified parameter.

Recently, a novel IS method for SVM was proposed in Aslani and Seipel (2021). In this method, border instances are determined based on one of the following cases. First, an instance is retained if its class is identical to its neighbors, whereas its neighbors are removed if the similarity index is at least two. An instance also is saved if its class and its neighbors are non identical, but its nearest neighbors from other classes are near to it. In this case, the nearest neighbors from other classes are retained. The last case, when an instance and its neighbor have different classes, but nearest neighbors from other classes are not near to it, nearest neighbors from other classes are retained, but the neighbors that are near to the instance are removed.

1.5 Feature Reduction

Datasets with large number of features have grown rapidly with development of technology. Dealing with raw data might lead to overfitting and poor performance of learning algorithms. Dimensionality reduction becomes important to improve the performance of algorithms and reduces the running time of algorithms. Feature selection and feature extraction approaches are the most popular techniques of dimensionality reduction. Feature selection aims to select the most relevant features. Feature extraction transforms that data onto low-dimensional subspace that saves most relevant information. Feature selection preserves the nature of the data, but some information can be lost when some features are discarded. Feature extraction reduces the feature space without losing a lot of information; however, the new features are not usually interpretable. In this dissertation, we focus on the feature reduction that is specified for classification problems.

Feature selection is usually classified into filters, wrappers, embedded, and hybrid methods. Filters methods extract the subset of features based on the performance measures. wrappers extracts based on the performance of modeling algorithm. Hybrid techniques combine filters and wrappers methods to improve the performance of the learning algorithm. Embedded methods extract the subset features within the learning algorithm process. Some feature selection methods determine the importance of features based on the ability of features to maintain data similarity such as Relief (Robnik-Šikonja and Kononenko, 2003), Laplacian score (He et al., 2005), trace ratio criterion (Nie et al., 2008), and Fisher score (Duda et al., 2012). Some other feature selection methods use sparse regularization to minimize the errors by making some feature coefficients to be too small (or zero), then corresponding features are removed. Least absolute shrinkage and selection operator (LASSO) (Tibshirani, 1996) is one of the most popular ℓ_p-norm regularize method. Other methods are based on statistical measures, such as T-scores (Davis and Sampson, 1986) and chi-square scores (Liu and Setiono, 1995), in which the features with high scores are most important.

On the other hand, many studies have demonstrated feature extraction methods. Principal components analysis (PCA) and linear discriminant analysis (LDA) are the most popular. Pearson (1901) introduced PCA; however, several variants have risen. They include dual PCA (Ghodsi, 2006), which is based on singular-value decomposition (SVD) instead of eigenvalue decomposition, and kernel PCA (Schölkopf et al., 1997), which is used for nonlinear data. LDA or Fisher discriminate analysis (FDA) was proposed by Fisher (1936). Kernel LDA (Mika et al., 1999) is one of variant of LDA that uses kernel function to work in nonlinear data. More details of feature reduction methods that are used in this work are presented.

1.5.1 Principal Components Analysis

Principal components analysis (PCA) is an unsupervised feature extraction method that was proposed by Karl Pearson in 1901. PCA aims to transform the correlated features into

uncorrelated features called principal components that represent data with the least loss of information. PCA aims to preserve the variation of the data.

PCA extracts the first principal component that represents the highest variance of data. The following component is orthogonal on the first one, and so forth. Consequently, the new components are uncorrelated.

Consider a dataset with d features, $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_d$, principal components are defined by the linear combination of all features as follows

$$\begin{aligned}
\mathbf{x}_1^* &= a_{11}\mathbf{x}_1 + a_{12}\mathbf{x}_2 + \ldots + a_{1d}\mathbf{x}_d \\
\mathbf{x}_2^* &= a_{21}\mathbf{x}_1 + a_{22}\mathbf{x}_2 + \ldots + a_{2d}\mathbf{x}_d \\
&\vdots \\
\mathbf{x}_d^* &= a_{d1}\mathbf{x}_1 + a_{d2}\mathbf{x}_2 + \ldots + a_{dd}\mathbf{x}_d
\end{aligned}$$

where $a's$ are principal components coefficients. The first linear combination $(\mathbf{x}_1^*)$ represents the first principal component that maximizes the variance of $\mathbf{x}_1^*$, subject to

$$a_{11}^2 + a_{12}^2 + \ldots + a_{1d}^2 = 1$$

The second principal component is defined by the second linear combination $(\mathbf{x}_2^*)$ that maximizes the variance of $(\mathbf{x}_2^*)$, subject to

$$a_{21}^2 + a_{22}^2 + \ldots + a_{2d}^2 = 1$$

and $(\mathbf{x}_1^*)$ is orthogonal to $(\mathbf{x}_2^*)$. The remaining principal components are derived by the same manner in which $a_{j1}^2 + a_{j2}^2 + \ldots + a_{jd}^2 = 1$, $j = 3, 4, \ldots, d$, and for $i \neq j$, $\mathbf{x}_i^*$ and $\mathbf{x}_j^*$ are orthogonal.

Principal components $(\mathbf{x}_j^*)$, $j = 1, 2, \ldots, d$ are extracted in which $\mathbf{x}_j^*$ is the eigenvector of the sample covariance matrix $\mathbf{\Sigma}$ with the largest eigenvalue λ_j. To determine which of

the most important principal components should be used, the components that explain the large ratio of variance are selected. Also, scree plot can be used to identify the principal components that can be used.

1.5.2 Linear Discriminant Analysis

Linear discriminant analysis (LDA) is a supervised feature extraction approach that aims to extract lower dimensions of the original data. The general process of LDA goes through three steps. First, for each class, the distance between the center of the class and instances is computed. The second step is computing the distance between centers of all classes, called between-class variance. Then, the lower dimension is obtained by minimizing the ratio of within-class variance to between-class variance.

Consider a training dataset $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ consisting of two classes C_1 and C_2, and let a unit vector $\mathbf{w} \in \mathbb{R}^d$, the optimization problem to obtain the best $\mathbf{w}$ that discriminates the two classes is

$$\max_{\mathbf{w}} \frac{\mathbf{w}^\top \mathbf{S}_b \mathbf{w}}{\mathbf{w}^\top \mathbf{S}_w \mathbf{w}}$$

where, $\mathbf{S}_b = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^\top$ is the between-class scatter matrix, whereas within-class scatter matrix is

$$\mathbf{S}_w = \sum_{\mathbf{x}_i \in C_1} (\mathbf{x}_i - \boldsymbol{\mu}_1)(\mathbf{x}_i - \boldsymbol{\mu}_1)^\top + \sum_{\mathbf{x}_i \in C_2} (\mathbf{x}_i - \boldsymbol{\mu}_2)(\mathbf{x}_i - \boldsymbol{\mu}_2)^\top$$

and $\boldsymbol{\mu}_j = \frac{1}{N_j} \sum_{\mathbf{x}_i \in C_j} \mathbf{x}_i$, $j = 1, 2$.

If $\mathbf{S}_w$ is non-singular, the solution is obtained by finding the largest eigenvector of $\mathbf{S}_w^{-1}\mathbf{S}_b$, then, the new data reduction is extracted that contains at most $c - 1$ features as follows

1. Compute $\mathbf{S}_w$ and $\mathbf{S}_b$.

2. Solve the eigenvalue problem $\mathbf{S}_b\mathbf{v} = \lambda \mathbf{S}_w\mathbf{w}$ to obtain all non-zero eigenvectors $\mathbf{W} =$

$[\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_k]$, where $k \leq c-1$.

3. Obtain the projection coordinates.

1.5.3 Least Absolute Shrinkage and Selection Operator

Least absolute shrinkage and selection operator (LASSO) is a regression analysis approach designed by Tibshirani (1996). It is a robust method that is used for regularization and feature selection tasks. The main goal of LASSO is to reduce the prediction error by keeping only the features correspond to non-zero coefficients. The process is penalizing coefficients of features and shrinking some to zero. One tuning parameter of LASSO controls regularization process.

Consider a response variable $\mathbf{y} \in \mathbb{R}^N$ and a matrix of predictor variables $\mathbf{X} \in \mathbb{R}^{N \times d+1}$, to estimate LASSO parameters ($\boldsymbol{\beta}$), the optimization problem is

$$\min_{\beta_0, \boldsymbol{\beta} \in \mathbb{R}^{d+1}} \|\mathbf{y} - \beta_0 - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda \|\boldsymbol{\beta}\|_1$$

where $\lambda > 0$ is a tuning parameter. The L_1 penalty, $\lambda\|\boldsymbol{\beta}\|_1$, generates sparse solutions of the previous optimization problem. Thus, the selected features are the ones that correspond to non-zero coefficients.

When a response variable $\mathbf{y} \in \{0, 1\}$ is binary, logistic regression is commonly used. Consider $y_i = \mathbf{I}(h_i = 0), i = 1, 2, \ldots, N$, the logistic regression can be written

$$\log \frac{P(H = 1 | \mathbf{X} = \mathbf{x})}{P(H = 0 | \mathbf{X} = \mathbf{x})} = \beta_0 + \mathbf{X}\boldsymbol{\beta}$$

The corresponding optimization problem is

$$\min_{\beta_0, \boldsymbol{\beta} \in \mathbb{R}^{d+1}} - \left\{ \frac{1}{N} \sum_{i=1}^{N} y_i (\beta_0 + \mathbf{x}_i^\top \boldsymbol{\beta}) - \log \left(1 + \exp(\beta_0 + \mathbf{x}_i^\top \boldsymbol{\beta})\right) \right\} + \frac{\lambda}{2} \left\{ 2\alpha \|\boldsymbol{\beta}\|_1 + (1 - \alpha) \|\boldsymbol{\beta}\|_2^2 \right\}$$

where $0 \leq \alpha \leq 1$, $\alpha = 1$ for LASSO.

When a response variable $\mathbf{y}$ has C classes in which $\mathbf{y} \in \{1, 2, \ldots, c\}$, logistic regression is commonly used. Consider $y_i = \mathbf{I}(h_i = 1), i = 1, 2, \ldots, N$, the model is extended to a multinomial model as

$$P(H = c|\mathbf{X} = \mathbf{x}) = \frac{\exp\left(\beta_0^{(c)} + \boldsymbol{\beta}^{(c)T}\mathbf{x}\right)}{\sum_{j=1}^{c} \exp\left(\beta_0^{(j)} + \boldsymbol{\beta}^{(j)T}\mathbf{x}\right)}$$

The corresponding optimization problem is

$$-\left\{\frac{1}{N}\sum_{i=1}^{N}\left[\sum_{k=1}^{c} y_i\left(\beta_0^{(k)} + \mathbf{x}_i^\top \boldsymbol{\beta}^{(k)}\right) - \log\sum_{j=1}^{c}\exp\left(\beta_0^{(j)} + \mathbf{x}_i^\top \boldsymbol{\beta}^{(j)}\right)\right]\right\}$$
$$+\frac{\lambda}{2}\left\{2\alpha\sum_{l=1}^{d}\|\boldsymbol{\beta}_l\|_q + (1-\alpha)\|\boldsymbol{\beta}\|_F^2\right\}$$

Where $\boldsymbol{\beta}$ is $d \times C$ matrix of coefficients.

1.5.4 Fisher Scores

Fisher scores (F-scores) is a statistical feature selection technique that computes the importance of each feature in the dataset. The discriminative power of each feature is measured. F-scores measure how features can assign values that minimize the distance of instances in the same class and maximize the distance of instances from different classes. Based on F-scores, the features are ranked. The higher the F-scores, the more important the features are.

To define the F-score of a feature, suppose a training dataset $\mathbf{X} \in \mathbb{R}^{N \times d}$ with a class variable of c categories, F-score of j^{th} feature $(\mathbf{x}_j)$ is defined as

$$FS(\mathbf{x}_j) = \frac{\sum_{k=1}^{c} N_k(\mu_{j(k)} - \mu_j)^2}{\sum_{k=1}^{c}\sum_{i=1}^{N_k}(x_{ij(k)} - \mu_{ij(k)})^2}$$

where N_k denotes the sample size of class $k = 1, 2, \ldots, c$, μ_j is the mean of feature j, $\mu_{j(k)}$ is the mean of feature j in class k, $x_{ij(k)}$ is the sample i in feature j of the class k.

1.6 Organization of the Dissertation

This chapter has introduced instance selection (IS) and reviewed most common methods; explored support vector machines (SVM); and training set selection methods to accelerate SVM; discussed threshold clustering (TC); reviewed of feature selection (FS) methods. The remaining of the dissertation is organized as follows. In Chapter 1.5, we discuss how TC is used as IS for SVM. We propose a method to accelerate SVM. Experiments are performed to compare the proposed method with other IS methods. We also derive a theoretical characteristic of our method. The development of our proposed method is considered in Chapter 3. We offer two methods using feature extraction and feature selection methods to improve our method. Experiments are carried out to compare the performance of our method under dimensionality reduction techniques. In Chapter 4, we review iterative threshold instance selection (ITIS) and illustrate IS methods based on clustering. Then, simulation study shows a comparison between ITIS and other IS methods. Additionally, iterative hybridized threshold clustering (IHTC) based on ITIS is compared with other hybrid clustering methods. Chapter 5 reviews the entire work and explores future work.

Chapter 2

Instance Selection with Threshold Clustering for Support Vector Machines

2.1 Introduction

Support vector machines (SVM) (Cortes and Vapnik, 1995; Vapnik, 1998) is a most popular classification method. The main property of SVM classifier is its ability to predict a new unseen instance based on small numbers of data points called support vectors; the remaining instances are discarded. SVM has been shown to be an effective classification technique—even in high dimensional settings—and has been implemented in many areas such as object classification, pattern recognition, and disease diagnosis. For instance, the SVM classifier provides effective results in classifying emails as spam or not from a given Gmail inbox (Singh et al., 2018). However, training SVM requires significant computational time—a standard implementation of SVM requires $O(N^3)$ runtime where N is the number of observations in the training set—making it computationally infeasible to perform under large-to-massive data settings (with respect to N).

To reduce overload storage and computational time, instance selection (IS) methods for SVM that construct reduced training sets have been developed. Some methods rely on selecting support vector candidates that are then used to construct reduced training sets. These approaches have been developed based on clustering such as SVM-KM method (De Almeida et al., 2000) that is based on applying k-means on the training set, then for homogenous cluster only the centroid is added while for heterogenous one the entire datapoints in that cluster are added. Yu et al. (2003) used hierarchical micro-clustering to extract the candidate support vectors. Other approaches are based on analyzing the geometry of data, (Abe and Inoue, 2001) used Mahalanobis distance to extract the datapoints that are in the decision boundary. These methods can easily discard useless vectors from the training sets and be generalized to multi-class data. However, these methods need large runtime and memory usage. Zhang and King (2002) applied β-skeleton algorithm to obtain the valuable datapoints. Some IS methods are based on constructing prototypes after clustering the data, then prototypes are used as selected set. This type of method reduces data effectively based on clustering method used for partitioning the data. However, traditional clustering methods-such as fuzzy C-means and k-means are not efficient in big data settings. Determining the number of clusters k may lead to poor results if k is improperly selected. Also, when the value of k in k-means is large, an algorithm to obtain the centroids may require several iterations to converge, which requires significant runtime.

Threshold clustering (TC) (Higgins et al., 2016) is a recent clustering method that was proposed originally for statistical blocking for massive data. The objective of TC is to minimize the maximum distance between each of two instances within the same cluster by only requiring the minimum number of instances t^* in each cluster. For data size N and threshold parameter t^*, the TC algorithm terminates in $O(Nt^* \log N)$ times, which is much smaller than other traditional clustering methods such as k-means when the number of clusters is large; the time complexity for k-means is $O(Nkmi)$, where k is the number of clusters, m is number of attributes, and i is number of iterations. This property allows TC

to work efficiently with massive data. Additionally, TC is designed to construct many small clusters.

In this chapter, we exploit the efficiency of TC to propose an IS method for SVM under massive data settings. Our proposed method, named support vector machines with threshold clustering (SVMTC), is based on constructing prototypes—data points or pseudo-data points that represent a group of units from original dataset. The process of SVMTC is carried out as follows. TC is applied separately on a training set of each class, in which each cluster consists at least t^* instances. Then, prototypes for all clusters of each class are constructed. Finally, SVM is trained on the prototypes. If the reduction is insufficient, TC is iterated prior to fitting SVM.

We show that, via SVMTC, the maximum distance between kernel matrix in optimization problem of SVM obtained by using the original data and approximate kernel matrix after replacing each instance by its prototype can be bounded if prototypes are constructed from TC. Then, using simulations and real data applications, we show that our proposed method reduces the time of SVM training data while maintaining the performance. In big data experiments, SVM may be impossible to be trained; however, SVMTC allows SVM to be trained in a short time while maintaining its efficiency.

The rest of the chapter is organized as follows. In Section 2.2, we describe SVM, IS methods, and TC as IS. A training set selection method by using TC is proposed in Section 2.3. Also, we provide theoretical framework and time complexity of the proposed method. Simulation study and experiments are presented in Section 2.4. We present discussion about the proposed method and the results in section 2.5.

2.2 Preliminaries

2.2.1 Support Vector Machines

Support vector machines (SVM) (Cortes and Vapnik, 1995; Vapnik, 1998) is a common supervised learning method that is used for classification. The SVM classifier can classify unseen instances by first obtaining the optimal hyperplane among many potential hyperplanes that splits the training data based on class label. The optimal hyperplane maximizes the distance between the closest instances of each class. Based on class label, training data can be separated by a hyperplane linearly or nonlinearly.

Consider an N d-dimensional training set $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)$, in which $\mathbf{x}_i \in \mathbb{R}^d$ is an input vector, and $y_i \in \{-1, +1\}$ is an output variable. The goal of the support vector classifier is to obtain the following decision hyperplane to predict a class of a new unseen instance y given the input $\mathbf{x}$.

$$f(\mathbf{x}) : \mathbf{w} \cdot \mathbf{x} + b = 0 \tag{2.1}$$

where $\mathbf{w} \in \mathbb{R}^d$ is a weight vector orthogonal on the hyperplane and $b \in \mathbb{R}$ is a scalar parameter. The training data is supposed to satisfy the following condition

$$y(\mathbf{w} \cdot \mathbf{x} + b) \geq 1 \tag{2.2}$$

The optimization problem in dual form of the objective function with respect to α is to maximize

$$Q(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \tag{2.3}$$

subject to

$$\sum_{i=1}^{N} \alpha_i y_i = 0$$

$$\alpha_i \geq 0, i = 1, \ldots, N$$

Instances $\mathbf{x}_i$ that correspond to nonzero Lagrange multipliers that satisfy $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) = 1$ are called support vectors. Hence, the decision function can be written as follows

$$f(\mathbf{x}^*) = \text{sign}\left(\sum_{j=1}^{n} \alpha_j y_j \mathbf{x}_j \cdot \mathbf{x}^* + b\right) \tag{2.4}$$

where $\mathbf{x}_j$ are the support vectors and n is the cardinality of support vectors set.

For the nonseparable case, a soft margin classifier is used to relax the constraints in which some instances are allowed to be violated. Thus, by introducing slack variables $\xi_i \geq 0, i = 1, 2, \ldots, N$, the constraints of optimization problem (2.3) are modified as follows

$$\sum_{i=1}^{N} \alpha_i y_i = 0$$

$$0 \leq \alpha_i \leq C, i = 1, \ldots, N,$$

where α_i are Lagrange multipliers and C is the penalty parameter of soft margin to control the balance between the empirical error and margin.

For a nonlinear hyperplane in which the data cannot be separated linearly, a nonlinear mapping of data is transformed from input space $\mathbb{R}^d$ into a high dimensional feature space $\mathbb{F}$ to obtain the optimal hyperplane linearly in the feature space. The *kernel* trick (Cortes and Vapnik, 1995) is utilized to obtain the similarity between two instances by computing the dot product under a kernel function instead of mapping the data. The decision function

is modified as follows

$$f(\mathbf{x}^*) = \text{sign}\left(\sum_{j=1}^{n} \alpha_j y_j \mathbf{K}(\mathbf{x}^*, \mathbf{x}_j) + b\right) \tag{2.5}$$

where $\mathbf{K}(,)$ is the *kernel* function. The optimization problem in the dual form is as follows

$$\max_{\alpha_i} Q(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \mathbf{K}(\mathbf{x}_i, \mathbf{x}_j) \tag{2.6}$$

Subject to

$$\sum_{i=1}^{N} \alpha_i y_i = 0 \tag{2.7}$$

$$0 \leq \alpha_i \leq C, i = 1, \ldots, N \tag{2.8}$$

Solving the QP problem of the SVM optimization problem requires $O(N^3)$ time complexity and $O(N^2)$ space complexity, where N is the number of training set.

2.2.2 Instance Selection Methods for Machine Learning

Data have been grown rapidly with development of technology, and methods to deal with massive data have been developed. Instance selection (IS) is one of a preprocessing method that is used to reduce the size of massive data by constructing a subset contains relevant instances from a training set. IS reduces the execution time and memory of training data when working with the reduced dataset. Hence, it makes state-of-the-art methods like SVM is effective in massive data. Ideally, the performance of prediction of new instances based on the SVM classifier, which is trained on a selected set, should be as accurate as if the original data is used.

Some IS methods select instances directly from a training set. Others construct pseudo instances that represent the training set. These methods are considered replacement ap-

proaches because the selected instances do not belong directly to the dataset. Most replacement approaches are based on clustering (Liu and Motoda, 2002; Spillmann et al., 2006). The process begins by partitioning a dataset into clusters, then a prototype-centroid or medoid of each cluster-is constructed. The selected datapoints are the computed prototypes that represent the clusters.

Several methods for IS of SVM have been developed. Tran et al. (2003) and Yang et al. (2003) cluster each class in training data by using k-means, then centroids of each cluster are used as a reduced training set to train SVM. A prototype selection method based on clustering (CLU) is proposed in Lumini and Nanni (2006). This method applies fuzzy C-means to partition the templates into clusters, then centers of clusters are used as selected prototypes. In Cao and Boley (2006), an approximate SVM approach based on prototypes is proposed. The idea relies on clustering data, then constructing representatives from clusters to be utilized as a reduced set. Training SVM on this reduced set is faster than using the original dataset and preserves the accuracy. They argue that kernel k-means should be used to get prototypes similar to original data, experimentally, principal direction divisive partition (PDDP) (Boley, 1998) is used as approximation to kernel k-means because of its speed. In addition, Li and Fang (2008) applied the same idea by using DBSCAN clustering.

This type of prototype selection method is simple and effective, however, using clustering such as fuzzy C-means or k-means is not efficient, especially in big data settings. Additionally, specifying the number of clusters k may lead to poor results if k is improper. Also, when the value of k in k-means is large, the algorithm to obtain the centroids may be converged after several iterations, which consumes time and space. In general, using an effective clustering method for IS method can have some significant advantages, but current methods for clustering require prohibitive computation to be applied under big-to-massive data settings.

2.3 Support Vector Machines with Threshold Clustering

Threshold clustering (TC) (Higgins et al., 2016) is a recently-developed clustering method that is efficient in big data settings. In Luo et al. (2019), TC is used to propose a new instance selection method called iterative threshold instance selection (ITIS). Then, ITIS is used to create a novel clustering method for big data settings.

We propose a new instance selection (IS) method for SVM by using threshold clustering (TC) which is named as support vector machines with threshold clustering (SVMTC). Using TC as IS overcomes the limitation of using other clustering in selecting the number of clusters as occurs in standard k-means and kernel k-means. In general, the proposed method is based on constructing prototypes of TC from each class in a training dataset. TC can cluster big data quickly and accurately. Thus, applying the prototypes constructed from TC in training SVM can greatly accelerate the process while keeping classification accuracy as similar to that of the original dataset. Also, TC is designed to form many small clusters of units, making it ideal for IS.

Consider N d-dimensional training set T consists $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N), \mathbf{x}_i \in \mathbb{R}^d$, $\mathbf{y} = (y_1, y_2, \ldots, y_N)$ is a class label in which each element $y_i, (i = 1, 2, \ldots, N)$ represents only one class c_j^* among $j = 1, 2, \ldots, C$ classes.

The process of support vector machines with threshold clustering (SVMTC) is as follows

1. **(Clustering)** Given threshold parameter t^*, cluster instances of the training set T in each class $c_j^*, j = 1, 2, \ldots, C$, separately, by using TC.

2. **(Create prototypes)** For each class c_j^*, compute centroids of all clusters, and add them to the reduced set $T^{'}$.

3. **(Training SVM)** Given $T^{'}$ from Step 2, the cost parameter, and the kernel function, SVM is trained on $T^{'}$.

4. TC is iterated on the prototypes if data reduction after Step 1 is insufficient.

Several kernel functions can be used in training SVM. However, the most popular one is Gaussian radial basis function (RBF) that works efficiently. SVMTC can train SVM in a set that is reduced by factor at least $(t^*)^r$ in each class, where $r \geq 1$ is the number of implementations of Step 1 in SVMTC. The size of reduced set T' is approximated as $|T'| \leq \left\lfloor \frac{N}{(t^*)^r} \right\rfloor$, where $\lfloor\ \rfloor$ is the floor function. The following example shows the results of implementing the SVMTC procedure without repeating Step 1.

Example: We simulated a data of size 10^3 from a mixture distribution of weighted combinations of two bivariate Gaussian distributions with $\mu_1 = (-0.5, 1)^\top$ and $\mu_2 = (3, 4)^\top$, and Σ_1 and Σ_2 are 2×2 diagonal matrices where diagonal entries are $(1, 0.5)$ and $(2, 1)$, respectively. The sizes of classes are 491 and 509 instances for class 1 and 2, respectively. The reduced data after implementing SVMTC with $t^* = 4$ is 189 with size 93 and 96 for class 1 and 2, respectively. The original data in each class is approximately reduced by factor $5 > t^* = 4$. Figure 2.1 visualizes Example 1 after training SVM on the original data and on the reduced dataset.

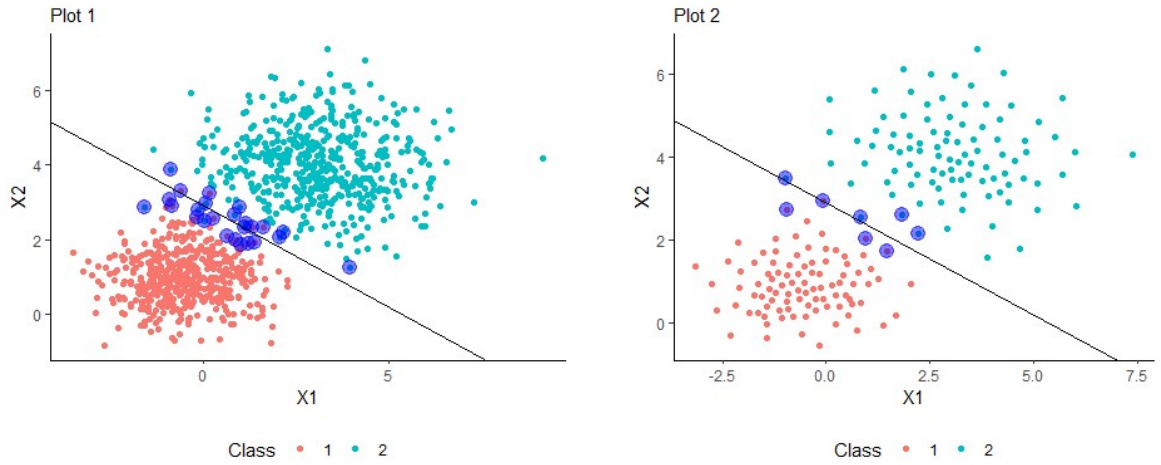

Figure 2.1: *Plot 1 shows the decision boundary after training SVM on the original data and plot 2 the decision boundary of training SVM on reduced data (prototypes of threshold clustering). Support vectors are surrounded by blue circles*

2.3.1 Theoretical Results of SVMTC

Evaluation of instance selection (IS) methods relies on minimizing the difference between the performance of using original dataset and reduced set. Most IS methods are evaluated by experimental study with lack of theory. However, Cao and Boley (2006) used the kernel matrix, which is the part of optimization problem that contains the training data, to compare between using the original data and the prototypes. It is proven that the difference between the kernel matrix that is used to solve the optimization problem for exact SVM based on the original data and kernel matrix for approximate SVM based on prototypes is minimized if kernel k-means is used to obtain the representatives for approximate SVM.

In this research, we consider to bound the maximum difference problem. Under the Gaussian radial basis kernel function, we prove that the maximum difference between kernel matrices of SVM optimization based on original data and based on prototypes can be bounded when threshold clustering (TC) is utilized to extract the prototypes. Equation 2.6 is rewritten as follows to facilitate the computations

$$\max_{\boldsymbol{\alpha}} Q(\boldsymbol{\alpha}) = \boldsymbol{\alpha}^\top \mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}^\top \mathbf{Y}^\top \mathcal{K} \mathbf{Y} \boldsymbol{\alpha}, \tag{2.9}$$

where $\mathbf{1}$ is all-ones vector, $\mathcal{K}$ is the kernel matrix, also called Gram matrix in which $\mathcal{K}_{ij} = \mathbf{K}(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)$, $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_N)$, and $\mathbf{Y}$ is $N \times N$ diagonal matrix in which the diagonal entries are $y_i, i = 1, 2, \ldots, N$.

We argue that the maximum distance between the kernel matrix under the original data $\mathcal{K}$ and the kernel matrix using the prototypes $\hat{\mathcal{K}}$ is bounded when the prototypes are extracted from threshold clustering under the Gaussian radial basis kernel function. As the objective for SVM (2.9) is a function of the kernel matrix $\mathcal{K}$, this result suggests a bound on the differences in support vectors when substituting the original data points with prototypes. The following lemma assists to prove our argument.

Lemma 2. *Suppose an N d-dimensional dataset, $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$. For any $\epsilon > 0$, and under*

Gaussian radial basis kernel function, if there exists a clustering C such that each cluster $c \in C$ contains at least t^* units and that

$$\max_{c \in C} \max_{ij \in c} \|\mathbf{x}_i - \mathbf{x}_j\| \leq \frac{\sqrt{2}}{4}\sigma \left(-\log\left(1 - \frac{\epsilon^2}{2}\right)\right)^{\frac{1}{2}}, \tag{2.10}$$

then, threshold clustering with parameter t^* produces a clustering such that

$$\max_i \|\phi(\mathbf{x}_i) - \phi(\hat{\mathbf{x}}_i)\| \leq \epsilon,$$

where $\hat{\mathbf{x}}_i$ is a prototype of the cluster that $\mathbf{x}_i$ belongs to, and σ is the parameter of Gaussian radial basis kernel function. The proof of Lemma 2 is in Appendix A.

Theorem 2. *Suppose prototypes are $\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, \ldots, \hat{\mathbf{x}}_N$ extracted from a threshold clustering, and suppose there is a clustering C such that each cluster $c \in C$ contains at least t^* units and the clustering satisfies (2.10). Then, under the Gaussian radial basis kernel function, the maximum distance between $\mathcal{K}$ and $\hat{\mathcal{K}}$ is bounded by $h(\epsilon) = 2\epsilon\left(1 + \frac{1}{2}\epsilon\right)$:*

$$\|\mathcal{K} - \hat{\mathcal{K}}\|_{max} \leq 2\epsilon\left(1 + \frac{1}{2}\epsilon\right),$$

if the prototypes are extracted from threshold clustering.

Proof. Under the Gaussian radial basis kernel function, $\mathbf{K}(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{x}_i - \mathbf{x}_j\|^2\right)$, we have $\|\phi(\mathbf{x}_i)\| = \sqrt{\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_i)} = \sqrt{\mathbf{K}(\mathbf{x}_i, \mathbf{x}_i)} = 1$. Also, $\|\mathcal{K} - \hat{\mathcal{K}}\|_{\max}$ can be written as

$\max_{ij}|\mathcal{K}_{ij} - \hat{\mathcal{K}}_{ij}|$. Then, $|\mathcal{K}_{ij} - \hat{\mathcal{K}}_{ij}|$ is simplified as follows

$$
\begin{aligned}
|\mathcal{K}_{ij} - \hat{\mathcal{K}}_{ij}| &= |\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j) - \phi(\hat{\mathbf{x}}_i).\phi(\hat{\mathbf{x}}_j)| \\
&= |\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j) - \phi(\hat{\mathbf{x}}_i) \cdot \phi(\mathbf{x}_j) + \phi(\hat{\mathbf{x}}_i) \cdot \phi(\mathbf{x}_j) - \phi(\hat{\mathbf{x}}_i) \cdot \phi(\hat{\mathbf{x}}_j)| \\
&= |\phi(\mathbf{x}_j) \cdot (\phi(\mathbf{x}_i) - \phi(\hat{\mathbf{x}}_i)) + \phi(\hat{\mathbf{x}}_i) \cdot (\phi(\mathbf{x}_j) - \phi(\hat{\mathbf{x}}_j))| \\
&\leq |\phi(\mathbf{x}_j).(\phi(\mathbf{x}_i) - \phi(\hat{\mathbf{x}}_i))| + |\phi(\hat{\mathbf{x}}_i) \cdot (\phi(\mathbf{x}_j) - \phi(\hat{\mathbf{x}}_j))| \\
&\leq \|\phi(\mathbf{x}_j)\|\|\phi(\mathbf{x}_i) - \phi(\hat{\mathbf{x}}_i)\| + \|\phi(\hat{\mathbf{x}}_i)\|\|\phi(\mathbf{x}_j) - \phi(\hat{\mathbf{x}}_j)\| \\
&\leq \|\phi(\mathbf{x}_j)\|\|\phi(\mathbf{x}_i) - \phi(\hat{\mathbf{x}}_i)\| + \|\phi(\mathbf{x}_i) + \phi(\hat{\mathbf{x}}_i) - \phi(\mathbf{x}_i)\|\|\phi(\mathbf{x}_j) - \phi(\hat{\mathbf{x}}_j)\| \\
&\leq \|\phi(\mathbf{x}_j)\|\|\phi(\mathbf{x}_i) - \phi(\hat{\mathbf{x}}_i)\| + \|\phi(\mathbf{x}_i)\|\|\phi(\mathbf{x}_j) - \phi(\hat{\mathbf{x}}_j)\| \\
&+ \|\phi(\hat{\mathbf{x}}_i) - \phi(\mathbf{x}_i)\|\|\phi(\mathbf{x}_j) - \phi(\hat{\mathbf{x}}_j)\| \\
&= \|\phi(\mathbf{x}_i) - \phi(\hat{\mathbf{x}}_i)\| + \|\phi(\mathbf{x}_j) - \phi(\hat{\mathbf{x}}_j)\| + \|\phi(\hat{\mathbf{x}}_i) - \phi(\mathbf{x}_i)\|\|\phi(\mathbf{x}_j) - \phi(\hat{\mathbf{x}}_j)\|
\end{aligned}
$$

Now, we find the maximum, then apply Lemma 2 as follows

$$\max_{ij} \|\phi(\mathbf{x}_i) - \phi(\hat{\mathbf{x}}_i)\| + \max_{ij} \|\phi(\mathbf{x}_j) - \phi(\hat{\mathbf{x}}_j)\| + \max_{ij} \|\phi(\hat{\mathbf{x}}_i) - \phi(\mathbf{x}_i)\|\|\phi(\mathbf{x}_j) - \phi(\hat{\mathbf{x}}_j)\| \leq 2\epsilon + \epsilon^2$$

□

Hence, the maximum distance between the kernel matrix used for training SVM under original data and the kernel matrix under prototypes is bounded when the prototypes are extracted from threshold clustering under Gaussian radial basis kernel function.

2.3.2 Time Complexity of SVMTC

We obtain time complexity of our proposed IS method to compare with the time complexity of training SVM by using the original training set. Time complexity of the SVMTC method is computed as follows. The time complexity of TC with r repetitions is $O(Nt^*r \log N)$,

where N is the training set size, t^* is the least number of instances in each cluster. The approximate complexity of SVM with the reduced dataset $T^{'}$ constructed from step 2 of SVMTC method is $O\left((N')^3\right)$, where N' is size of $T^{'}$ which is bounded as $N^{'} \leq \left\lfloor \frac{N}{(t^*)^r} \right\rfloor$. Hence, the total time complexity of SVMTC method is as follows

$$O\left(Nt^*r\log N + (N')^3\right)$$

where $N^{'}$ is the number of clusters. This is smaller than the time complexity of the regular SVM, $O(N^3)$. When $N^{'} \ll N$, the time complexity of SVMTC is much smaller than the time complexity of SVM by training the entire dataset.

2.4 Experimental Study

We evaluate SVMTC on real-world datasets in the UCI Machine Learning database (Merz, 1998) and on benchmark datasets. We compare our method to SVM-KM by clustering the entire training set by k-means. Then, centroids of one-class clusters are combined with data points of heterogenous clusters to form the reduced dataset. Also, we compare to CLU under k-means in which the reduced dataset is the prototype obtained from k-means clustering. Comparison between the proposed method and the other methods is carried out by several measurements. First, runtime, which is divided into two parts. One part computes the time (seconds) of reducing the data under IS methods and the other computes the time of training SVM. Memory space also is split into two parts. One is to compute the capacity (megabytes) of reducing the training set and the other is for the capacity of data training. Additionally, testing accuracy is utilized for comparisons. The accuracy is simply defined as proportion of correct predicted classes of a class label in a test set to all predictions. Finally, the reduction rate is used to obtain how much the IS method can reduce the data. The higher the value of reduction rate and testing classification, the more preferable the instance selection method. The superior IS method is the one that uses less runtime and

memory usage to reduce the data.

For an imbalanced dataset, it is more accurate to compare between methods using F1. Recall and precision measures are required to obtain F1. Recall is defined as proportion of correct classification of positive class to total of correct classification of positive class and incorrect classification of negative class. Precision is a proportion of correct classification of positive class to total of correct classification of positive class and incorrect classification of positive class. Hence, F1 can be computed as

$$\text{F1} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}.$$

The higher the value of F1, the more accurate classification.

All algorithms are implemented in the R programming language. The `e1071` package (Meyer et al., 2020) is used to perform SVM by using `svm` function which implements the normal SMO algorithm. The `e1071` package interfaces with `libsvm` in C++. The `libsvm` supports the multiclass problems by using one-against-one approach. For all experiments, the Gaussian radial basis function is used, $\mathbf{K}(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2\right)$, where $\gamma = 1/(2\sigma^2)$. Threshold clustering is executed by using `sc_clustering` function in `scclust` package (Savje et al., 2018). Some important functions we made are shown in Appendix C. Some experiments are implemented on Intel(R) Core(TM) i7-7500 CPU at 2.7 GHz processor. For big data simulation, experiments are applied in Intel(R) Core(TM) i7-7700 CPU at 3.6 GHz processor.

2.4.1 Experimental Datasets

Datasets used in the experiments are described in detail:

- ***Simulated*** data is a two-dimensional data with binary-class variable. The training set is sampled from a mixture distribution of weighted combinations of two bivariate

Gaussian distributions with pdf as follows

$$f(\mathbf{x}) = 0.5p(\mathbf{x}|\mu_1, \mathbf{\Sigma}_1) + 0.5p(\mathbf{x}|\mu_2, \mathbf{\Sigma}_2) \tag{2.11}$$

for $j = 1, 2$, $p(\mathbf{x}|\mu_j, \mathbf{\Sigma}_j)$ is the pdf of Gaussian distribution with parameters $\mu_1 = (-0.5, 1)^\top$, $\mu_2 = (2.5, 4)^\top$, $\mathbf{\Sigma}_1$ and $\mathbf{\Sigma}_2$ are 2×2 diagonal matrices with diagonal entries $(1, 0.5)$ and $(2, 1)$, respectively. The category of the class variable is labeled as 0, if the data point is sampled from $p(\mathbf{x}|\mu_1, \mathbf{\Sigma}_1)$ in equation 2.11, otherwise, it is 1. The test set is sampled in the same way as the training dataset. The sizes of the training sets varied between 5×10^4 and 10^6, while the test set is 5×10^4.

- ***Checkerboard*** is two-dimensional 4×4 checkerboard dataset and consists of $50,000$ data points, two attributes and one class variable, see Figure 2.2.

- ***Covertype*** dataset has $580,012$ instances, and 55 attributes. The class variable has seven categories which are the types of forest cover. The dataset is preprocessed so that the class variable becomes binary (Baker et al., 2019).

- ***Credit Card*** dataset has $30,000$ instances, and 24 attributes. The class label is to determine if a customer uses a default payment or not.

- ***HTRU2*** dataset has $17,898$ instances, and 9 attributes. The class label has two categories; positive and negative.

- ***Magic*** dataset has $19,020$ instances, and 11 attributes. The class label has two categories; gamma (signal) and hadron (background).

- ***Mnist*** handwritten (0-9) digits dataset contains $70,000$ gray scale images in which each image is formed of 28×28 pixels of handwritten digits, totaling 784 attributes besides the class label, which takes values $0, 1, \ldots, 9$. To facilitate the training process, Cao and Boley (2006) transformed the class variable into two classes in which digits

$(1, 2, 3, 4, 5)$ belong to class 1, and digits $(0, 6, 7, 8, 9)$ are transformed into class -1. In the experiment, we implement our method in binary and multi-class MNIST dataset. The original MNIST with 10 classes and the prototypes obtained from threshold clustering with $t^* = 3$ are shown in Figure 2.3.

- ***Shuttle*** dataset has $57,999$ instances, and 9 attributes. The class label has seven categories; Rad Flow, Fpv Close, Fpv Open, High, Bypass, Bpv Close, and Bpv Open.

- ***Skin*** dataset has $245,057$ instances, and 4 attributes. The class label has two categories; skin and non-skin. All datasets used in the experiments are summarized in Table 2.1.

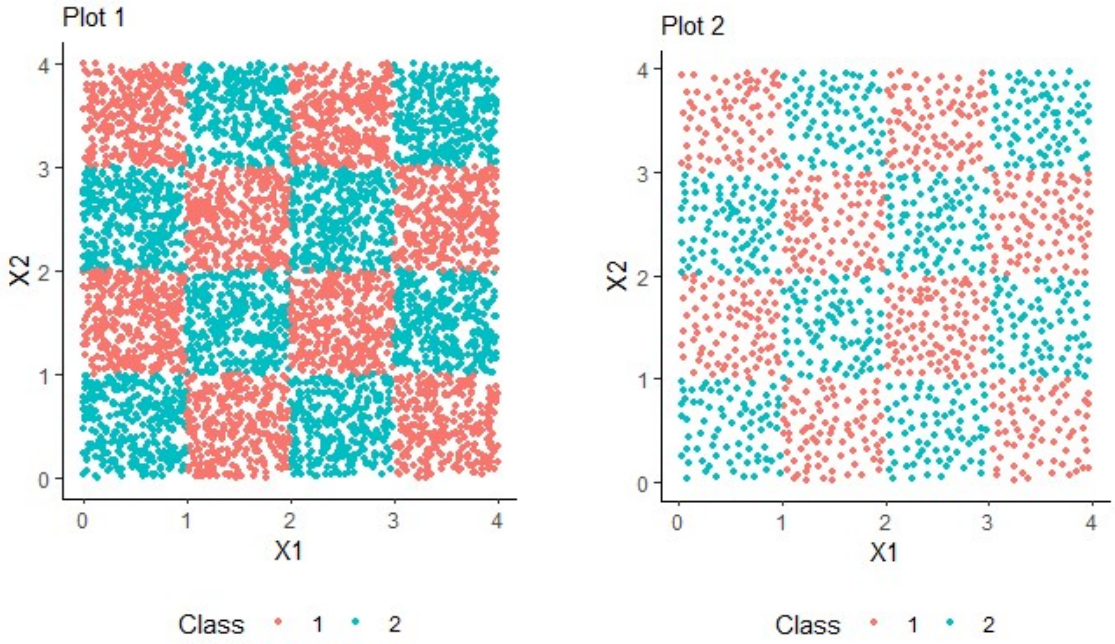

Figure 2.2: *A two-dimensional* 4×4 *Checkerboard dataset: Plot* 1 *is the original dataset. Plot* 2 *is prototypes obtained from step* 2 *in SVMTC method.*

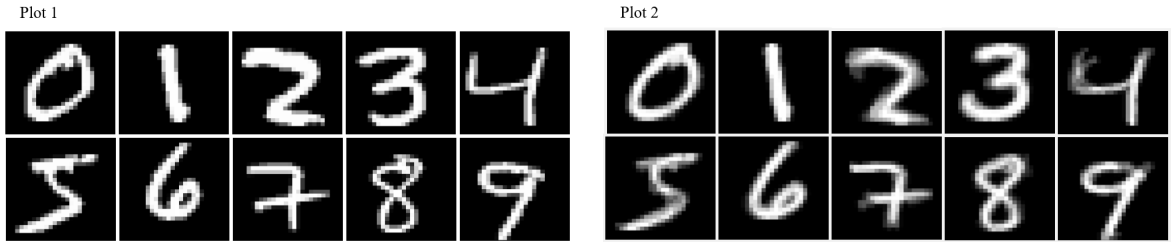

Figure 2.3: *Plot 1 illustrates* 10 *labels* $0-9$ *in* 10 *images from original MNIST dataset. Plot 2 shows the images of labels* $0-9$ *from prototypes of threshold clustering extracted from the original MNIST dataset. For each label, the original data point in plot 1 contributes in making the corresponding prototype in Plot 2.*

Dataset	Instances	Features	Classes	Reference
Covertype-7	581,012	55	7	Blackard and Dean (1999)
CreditCard	30,000	24	2	Yeh and Lien (2009)
HTRU2	17,898	9	2	Lyon et al. (2015)
Magic	19,020	11	2	Bock et al. (2004)
Mnist	70,000	785	10	LeCun et al. (1998)
Shuttle	58,000	9	7	Michie et al. (1994)
Skin	245,057	4	2	Bhatt and Dhall (2010)

Table 2.1: *Datasets Description*

2.4.2 Experimental Setup

To make unbiased comparison between instance selection methods, a stratified 10-fold cross-validation method is used. In this method, a dataset is split into 10 folds, then for 10 times, one fold is used for test set, and the remaining sets are utilized as a training set. This full process is repeated 10 times to decrease the impact of randomization of splitting datasets. Based on a class label, stratified splitting contributes to sampling from each class proportionally. The final results of the evaluation measures are averaged over both folds and repetitions, which is 100 in most of our experiments.

We evaluate the performance of SVMTC by comparing it with the performance of applying the original data on SVM when data scales up. Covertype dataset is used with training data sizes $(50, 100, \ldots, 500)$ thousands. With scaling the data, the accuracy and training time are examined see Figure 2.4.

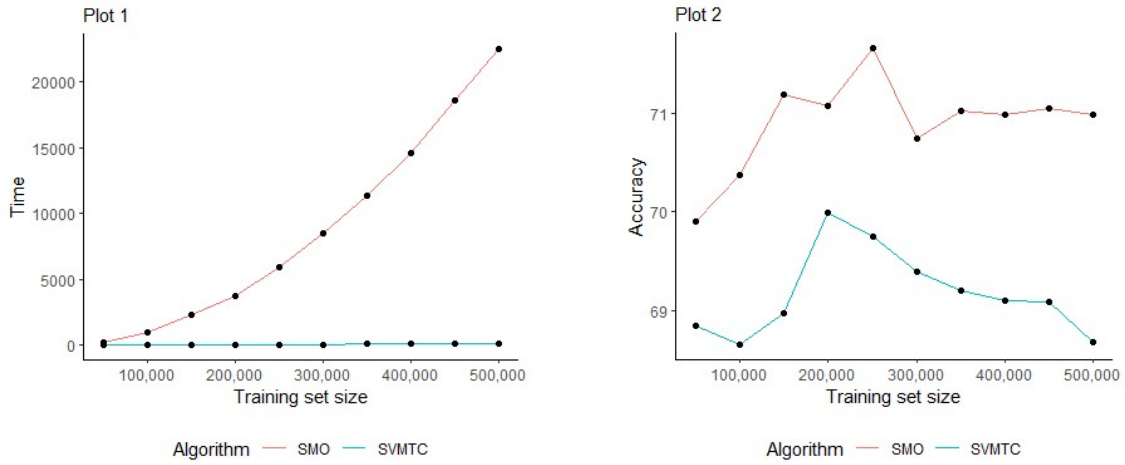

Figure 2.4: *Scaling performance of the proposed method SVMTC and using the entire data (SMO) on Covertype dataset with sizes from* $50,000$ *to* $500,000$ *varied on the interval* $50,000$. *Plot* 1 *illustrates the comparison of runtime of SVMTC and of using the original data in seconds. Plot* 2 *shows the comparison of classification accuracy*

In terms of scaling, Figure 2.4 compares the performance of our proposed method SVMTC and SVM applied on raw data without IS (SMO). Figure 2.4 illustrates that classification accuracy by SVMTC fluctuates with training size increases. However, comparing with SMO, it works effectively in terms of classification accuracy and training time, especially when data size is greater than $400,000$. Also, training time under SVMTC is linearly increased. Under SMO, training time grows rapidly.

Big Data Simulation

To show the efficiency of SVMTC in a big data, we compare SVMTC with a well-known prototypes selection method that applies k-means to obtain the prototypes (CLU). We simulated data of sizes $10^6, 10^7$, and 10^8 from a mixture distribution of weighted combinations

of two bivariate Gaussian distributions with parameters $\mu_1 = (-0.5, 1)^\top$, $\mu_2 = (2.5, 4)^\top$, $\mathbf{\Sigma}_1$ and $\mathbf{\Sigma}_2$ are 2×2 diagonal matrices with diagonal entries $(1, 0.5)$ and $(2, 1)$, respectively. We also simulated data of sizes 10^6 and 10^7 from a mixture distribution of weighted combinations of two 5-dimensional Gaussian distributions with parameters $\mu_1 = (-0.5, 1, 1.5, 1.7, 1.9)^\top$, $\mu_2 = (1.8, 2.1, 2.5, 2.8, 3)^\top$, $\mathbf{\Sigma}_1$ and $\mathbf{\Sigma}_2$ are 5×5 diagonal matrices with diagonal entries $(3, 2, 2, 1, 1)$ and $(1, 3, 2, 1, 2)$, respectively.

For 2-dimensional and 5-dimensional simulated data, when training size is 10^6, we picked $t^* = 3$. To obtain sufficient reduction we iterate TC twice; the second iteration is done by clustering the prototypes obtained from the first iteration, while for k-means, $k = 50,000$ is chosen to balance with number of clusters constructed from TC. With scaling the training size up, some parameters are scaled up little to obtain an adequate reduced size for training SVM. When sample size is 10^7, $t^* = 5$ and $r = 2$ are chosen for SVMTC, and $k = 100,000$ for k-means. For sample size 10^7, we tried $t^* = 7$ and $r = 2$ for SVMTC, and $k = 50,000$ for k-means. After training SVM on reduced datasets, we test the accuracy on simulated test set of size 10^4 that is sampled by the same way as training set for all experiments.

Table 2.2 and Table 2.3 illustrate the results of comparisons. When training set size is 10^8, it is impossible to implement k-means because of memory space limitation. In contrast, our method spends less than 20 minutes to reduce data and less than 10 minutes for training SVM on this reduced data with a high accuracy while training SVM on the original data could takes days or it could be sometimes impossible. Results in Table 2.2 and Table 2.3 show that our proposed method SVMTC outperforms k-means in performing high accuracy in quite short clustering and training time. It could be impossible to train SVM in the original data when data is 10^8, however, SVMTC allows training SVM while keeping classification accuracy high.

Results of SVMTC on the simulated data size 10^7 with $t^* = 2$ and varying the number of repetitions r are shown in Figure B.3 and Figure B.4 in Appendix B.

Feature Size		2				
t^*		3	5	7	6	8
Measurement	Method	10^6	10^7	10^7	10^8	10^8
Pre Time	k-means	402.511	11,831.439	12,123.843	-	-
	SVMTC	5.653	51.322	63.200	751.969	782.028
Train Time	k-means	57.226	229.087	66.008	-	-
	SVMTC	18.967	376.5780	80.818	15,427.06	4,106.26
Pre Mem	k-means	345.87	2,322.96	2,907.345	-	-
	SVMTC	210.845	1,736.145	1,747.245	18,293.67	18,315.68
Train Mem	k-means	87.03	192.315	93.580	-	-
	SVMTC	62.115	186.7550	96.965	1,434.405	772.200
SV	k-means	3,868.25	7,698.5	4,086.7	-	-
	SVMTC	2,602.05	7,728.60	4,188.45	51,667.65	30,301.60
RR	k-means	90.00	98.00	99.00	-	-
	SVMTC	93.29	97.94	98.90	98.57	99.17
Accuracy	k-means	98.490	98.504	98.499	-	-
	SVMTC	98.500	98.502	98.505	98.500	98.478

Table 2.2: *Comparisons between SVMTC and k-means in big simulated datasets with two features. RR indicates reduction rate of a dataset; pre is for preprocessing (clustering); time and mem indicate runtime and memory usage, respectively.*

Parameters Analysis

Threshold clustering (TC) is based on only one threshold parameter t^*. Choosing a value of t^* for TC in our proposed method SVMTC affects the results remarkably. Figures 2.5 and 2.6 illustrate the performance of SVMTC with parameter change t^*. The runtime and memory usage of obtaining the prototypes are approximately constant when t^* increases; time and memory for training SVM in the reduced set increases slightly. Although classification accuracy reduces with increasing t^*, it is still high enough to be used. Selecting a small value of t^* for SVMTC provides accurate results approximately equals to training SVM on the original dataset. The results may not be affected greatly when training set size is massive.

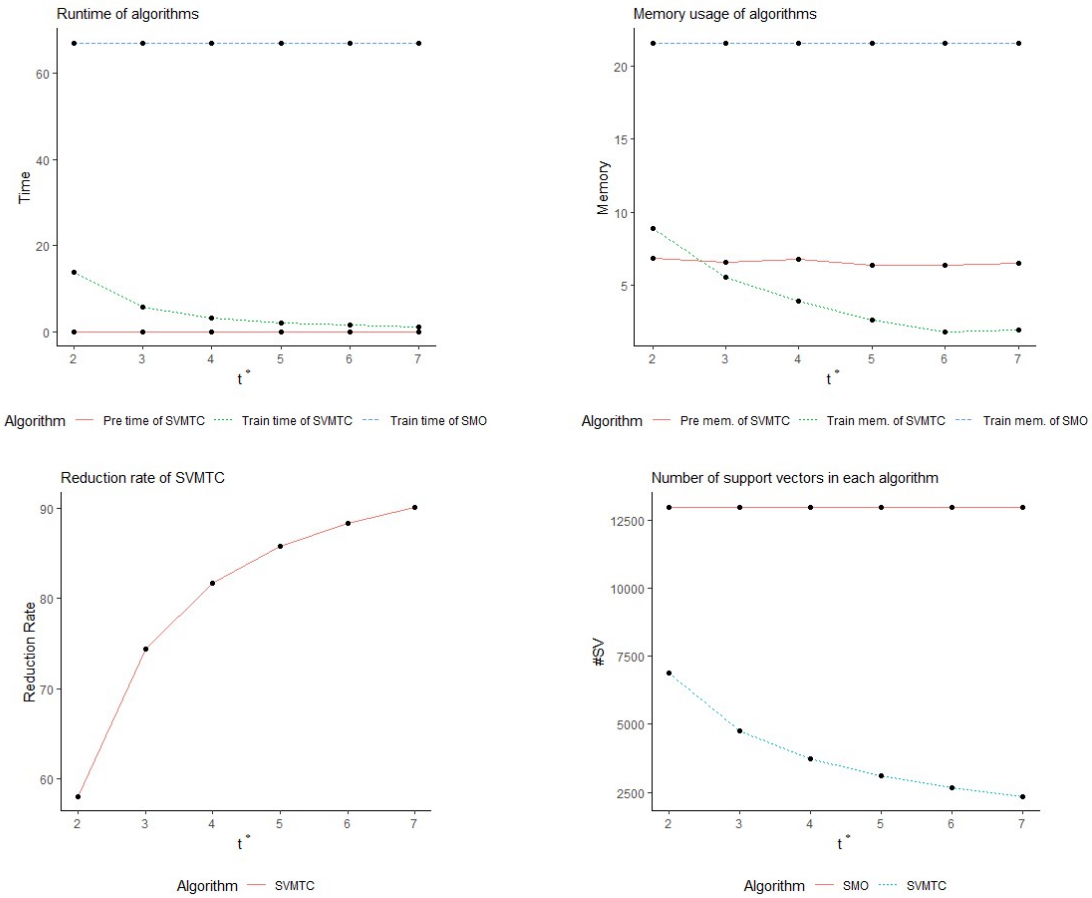

Figure 2.5: *Runtime, memory usage, reduction rate, and number of support vectors of SVMTC and SMO in* 4×4 *checkerboard dataset of size 50,000*

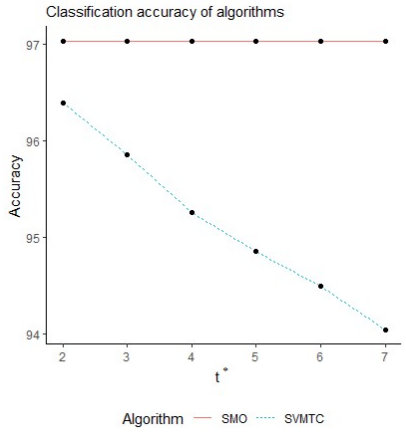

Figure 2.6: *Classification accuracy of SVMTC and SMO in two-dimensional* 4×4 *checkerboard dataset of size 50,000 based on changing* t^*

Feature Size		5		
t^*		3	5	7
Measurement	Method	10^6	10^7	10^7
Pre Time	k-means	1,401.903	44,857.904	22,444.860
	SVMTC	23.870	337.198	408.525
Train Time	k-means	521.427	1,659.400	538.838
	SVMTC	172.279	1,016.635	237.642
Pre Mem	k-means	461.560	3,521.850	3,284.200
	SVMTC	272.770	2807.240	2,802.975
Train Mem	k-means	167.150	392.360	194.895
	SVMTC	94.840	311.050	127.695
SV	k-means	27,702.9	55,582.500	29,500.9
	SVMTC	16,555.1	43,491.250	19,645.8
RR	k-means	90.00	98.00	99.00
	SVMTC	93.92	98.35	99.29
Accuracy	k-means	89.183	89.097	89.095
	SVMTC	89.166	89.106	89.104

Table 2.3: *Comparisons between SVMTC and k-means in big simulated datasets with five features. RR indicates reduction rate of a dataset; pre is for preprocessing (clustering); time and mem indicate runtime and memory usage, respectively.*

2.4.3 Experimental Results and Analysis

The results of applying our proposed method, SVMTC on simulated and real datasets are discussed in this section. Based on Tables 2.4 and 2.5, runtime of reducing a dataset by using SVMTC and training is rapid compared to other methods. In addition, SVMTC uses slight memory storage for reducing and training a dataset, see Tables 2.6 and 2.7. The instance selection algorithm for SVM that uses few support vectors and high reduction is superior. In Tables 2.8 and 2.9, SVMTC uses less support vectors in all datasets than the other methods; some methods outperform SVMTC in reducing a data. Classification accuracy of predicting new instances for different datasets is also shown in Tables 2.8 and 2.9. The accuracy of our proposed method SVMTC is higher than other methods in most datasets, and is almost identical to the accuracy of using the original dataset. We compute recall, precision, and F1 measures for imbalanced datasets HTRU2, magic, and skin. F1 results under SMO and SVMTC are shown in Figure 2.7. Results of applying SVMTC on

the real datasets with varying values of t^* and $r = 1$ are shown in Figure B.1 and Figure B.2 in Appendix B. In general, SVMTC accelerates training SVM without sacrificing the performance. Also, SVMTC outperforms other instance selection methods in term of accuracy, time and memory.

Dataset	Time	Credit Card	HTRU2	Magic	Mnist	Shuttle	Skin
SVMTC	Pre	0.3788	0.058	0.103	754.90	0.4392	0.299
	Train	32.6698	0.491	2.783	157.56	2.5132	1.541
k-means	Pre	9.9096	0.5542	1.9526	981.16	-	19.750
	Train	72.3584	0.1726	3.9172	214.06	-	1.348
SVM-KM	Pre	13.1386	2.726	2.031	2,139.80	83.8998	44.875
	Train	143.2354	0.860	15.535	355.76	0.5478	1.940
SMO	Pre	-	-	-	-	-	-
	Train	294.9776	3.321	23.713	2,454.18	9.8914	44.256

Table 2.4: *Execution time for preprocessing (pre) and training (Train) of data constructed from algorithms for different real datasets. Runtime of SVMTC is less than the time of other methods in most datasets.*

Dataset	Time	Checker50K	Checker100K	Sim50K	Sim100K	Sim500K	Sim1M
SVMTC	Pre	0.1200	0.2638	0.0918	0.2042	1.9710	5.6530
	Train	1.0948	2.2698	0.3786	0.9578	24.1428	18.9760
k-means	Pre	1.7066	6.5016	2.2058	6.6072	181.4016	402.5110
	Train	1.275	4.0792	0.7810	1.9098	73.1770	57.2260
SVM-KM	Pre	3.4244	13.919	5.0202	13.7784	334.9844	823.0390
	Train	1.2298	4.1570	2.5608	8.4420	304.0714	988.8140
SMO	Pre	-	-	-	-	-	-
	Train	20.1414	68.1524	10.4996	62.1834	2,584.2988	-

Table 2.5: *Execution time for preprocessing (pre) and training (Train) of data constructed from algorithms for simulated datasets of different sizes. Runtime of SVMTC is less than the time of other methods in most datasets.*

Dataset	Memory	Credit Card	HTRU2	Magic	Mnist	Shuttle	skin
SVMTC	Pre	30.778	4.940	4.314	2,693.80	24.979	43.39
	Train	44.212	18.299	10.677	1,676.90	40.064	23.31
k-means	Pre	124.936	10.311	28.688	2,087.40	-	423.28
	Train	54.534	5.194	11.98	1,736.20	-	23.16
SVM-KM	Pre	1,821.518	913.663	277.994	90,494.50	847.413	2,556.11
	Train	77.643	17.991	18.564	2,123.10	10.0580	23.77
SMO	Pre	-	-	-	-	-	-
	Train	109.235	42.229	23.106	6,891.90	97.787	198.74

Table 2.6: *Memory usage for preprocessing (pre) and training (Train) of data constructed from algorithms for different real datasets. Memory usage of SVMTC is less than the time of other methods in most datasets.*

Dataset	Memory	Checker50K	Checker100K	Sim50K	Sim100K	Sim500K	Sim1M
SVMTC	Pre	6.3170	13.7940	7.806	17.061	106.404	210.845
	Train	1.7000	3.3340	6.798	12.166	60.197	62.115
k-means	Pre	25.3030	49.6040	28.872	47.985	274.846	345.87
	Train	1.5690	6.224	10.105	17.242	84.395	87.03
SVM-KM	Pre	647.1570	1,626.89	1,098.936	1,691.084	9,207.345	11,500.89
	Train	1.6140	5,2770	10.388	18.663	97.515	140.315
SMO	Pre	-	-	-	-	-	-
	Train	18.3460	36.7510	45.253	71.974	366.279	-

Table 2.7: *Memory usage for preprocessing (pre) and training (Train) of data constructed from algorithms for simulated datasets of different sizes. Memory usage of SVMTC is less than the time of other methods in most datasets.*

Dataset		Credit Card	HTRU2	Magic	Mnist	Shuttle	skin
SVMTC	SV	8,517.67	537.09	2,766.88	3,180	582.32	618.63
	RR	58.7164	58.409	58.79	76.99	58.14	90.10
k-means	SV	11,847.15	513.38	3,651	4,125	-	581.9
	RR	48.15	86.34	59.11	77.00	-	90.93
SVM-KM	SV	15,032.99	909.61	5,963.82	5,436	527.53	802.51
	RR	29.7763	58.793	27.06	72.73	89.35	90.85
SMO	SV	18,776.02	1,115.99	6,508.18	12,720	851.90	1,718.64
	RR	-	-	-	-	-	-

Table 2.8: *Number of support vectors that are used (SV) and the reduction rate of instance selection methods (RR) for different real datasets.*

Dataset		Checker50K	Checker100K	Sim50K	Sim100K	Sim500K	Sim1M
SVMTC	SV	4,046.30	5,594.60	400.74	591.58	2,726.25	2,602.05
	RR	81.66	85.79	81.65	85.80	85.80	93.29
k-means	SV	4,305.22	7,173.76	527.53	834.06	3,831.77	3,868.25
	RR	80.00	80.00	76.00	80.00	80.00	90.00
SVM-KM	SV	5,125.53	8,303.75	1,808.67	3,576.77	17,538.53	35,325.50
	RR	82.93	82.15	80.50	79.00	78.99	83.24
SMO	SV	14,026.87	23,191.42	1,924.77	3,753.96	18,345.07	-
	RR	-	-	-	-	-	-

Table 2.9: *Number of support vectors that are used (SV) and the reduction rate of instance selection methods (RR) for simulated datasets.*

Dataset	Credit Card	HTRU2	Magic	Mnist	Shuttle	skin
SVMTC	79.77	97.96	86.25	98.47	99.71	99.65
k-means	80.10	96.92	82.39	98.49	-	99.53
SVM-KM	80.13	97.95	86.31	98.48	99.19	99.59
SMO	80.37	97.98	87.25	98.76	99.75	99.86

Table 2.10: *Prediction accuracy of instance selection methods for different real datasets. Accuracy of SVMTC is higher than the accuracy of other methods in most datasets.*

Dataset	Checker50K	Checker100K	Sim50K	Sim100K	Sim500K	Sim1M
SVMTC	95.83	96.52	98.48	98.51	98.50	98.50
k-means	95.11	96.53	98.47	98.50	98.50	98.49
SVM-KM	90.81	94.46	98.47	98.49	98.50	98.50
SMO	97.60	98.07	98.47	98.50	98.50	-

Table 2.11: *Prediction accuracy of instance selection methods for simulated datasets. Accuracy of SVMTC is higher than the accuracy of other methods in most datasets.*

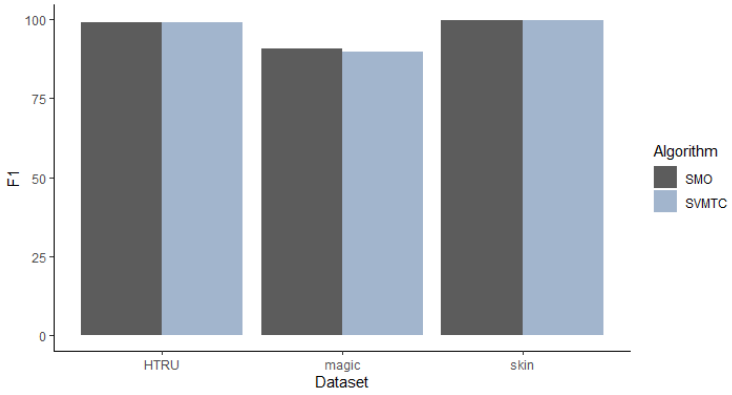

Figure 2.7: *F*1 *for imbalanced datasets under SMO and SVMTC algorithms.*

2.5 Discussion

Many instance selection (IS) methods for support vector machines (SVM) have been developed; however, most methods decelerate when training set size is very large. Specifically, approaches based on clustering typically utilize traditional clustering methods which are not effective in big data settings. Recently, threshold clustering (TC) has been discovered for clustering in which only the minimum number of instances in a cluster is assigned. The objective of TC is to form clusters where maximum within-cluster distance is minimized. Two main characteristics distinguish TC from other traditional clustering methods. First, implementing TC reduces runtime and memory usage, especially in a massive dataset. Second, TC is designed to form many clusters in which each cluster contains few instances which makes it ideal for IS.

In this chapter, an IS method using TC is proposed for SVM. The method, support vector machines with threshold clustering (SVMTC), is based on constructing prototypes from TC, based on the parameter t^*, for each class in training set. Then, prototypes are used as a reduced set for training SVM. If the reduction is not sufficient, TC is iterated on the prototypes r times. We then prove that constructing prototypes by using TC can approximately minimizes the maximum difference between the kernel matrix of using original data and the kernel matrix of using an approximate data based on prototypes under radial basis function (RBF) kernel function.

Simulations and experiments are applied to compare our method with other instance selection methods. In big data settings, SVMTC shows its ability to train SVM using less time and memory while keeping the accuracy high. In real data application, performance of SVMTC is almost identical to the performance of using an original data where execution time and memory usage of SVMTC is much smaller. With increasing the value of t^*, there is a trade-off between classification accuracy and preprocessing time. Hence, to get more accurate classification of SVMTC, small values of t^* could be required, but time of clustering could be longer.

Chapter 3

SVMTC under Feature Reduction Techniques

3.1 Introduction

In machine learning applications, large number of features, and massive data reduce the performance of trained models and increase the time of training process. Many methods have been developed to deal with data that has a large number of variables. Feature selection (FS) and feature extraction (FE) are feature reduction techniques. FS methods select a subset of original features. The selected features aim to improve the performance of classification by minimizing the redundancy. Least absolute shrinkage and selection operator (LASSO) and Fisher scores (F-scores) are examples of FS. On the other hand, FE techniques aim to extract new features by projecting the data onto a new feature space.

Threshold clustering (TC) (Higgins et al., 2016) is a recent clustering method that works well under massive data. Recently, TC is utilized to improve traditional clustering methods such as k-means, and it is used as an instance selection (IS) method. In Chapter 1.5, we used TC to accelerate training support vector machines (SVM). However, TC might work poorly in data with a large number of features. Thus, we propose using several feature

reduction methods for TC to reduce the number of features so that TC works ideally to accelerate training SVM.

In this chapter, we combine feature reduction and IS to improve SVM performance. We utilize different feature reduction methods as a preprocessing step to our proposed IS method, SVMTC. We proposed two methods to reduce the dimensions prior to SVMTC implementation. One method is based on using the extracted features; the other aims to use the reduced original features. Using real data applications, we find that LASSO tends to be an effective feature selection method, and overall, show that SVMTC is improved significantly under the proposed methods.

The rest of this chapter is organized as follows. In section 3.2, we describe principal components analysis (PCA), linear discriminate analysis (LDA), LASSO, and F-scores. The proposed procedures are discussed in section 3.3. Experimental studies are presented in section 3.4. Finally, we present discussion about the proposed methods and the results of applications in section 3.5.

3.2 Feature Reduction Methods

A brief review of feature reduction methods is presented.

3.2.1 Principal Components Analysis

Principal components analysis (PCA) is an unsupervised feature extraction method that was proposed by Karl Pearson in 1901. This method aims to transform the correlated features into uncorrelated features called principal components that represent data with the least loss of information. PCA aims to preserve the variation of the data.

PCA extracts the first principal component that represents the highest variance of data. The following component is orthogonal on the first one, and so forth. Consequently, the new components are uncorrelated.

Consider a dataset with d features, $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_d$, principal components are defined by the linear combination of all features as follows

$$\begin{aligned}
\mathbf{x}_1^* &= a_{11}\mathbf{x}_1 + a_{12}\mathbf{x}_2 + \ldots + a_{1d}\mathbf{x}_d \\
\mathbf{x}_2^* &= a_{21}\mathbf{x}_1 + a_{22}\mathbf{x}_2 + \ldots + a_{2d}\mathbf{x}_d \\
&\vdots \\
\mathbf{x}_d^* &= a_{d1}\mathbf{x}_1 + a_{d2}\mathbf{x}_2 + \ldots + a_{dd}\mathbf{x}_d
\end{aligned}$$

where $a's$ are principal components coefficients. Principal components $\mathbf{x}_j^*$, $j = 1, 2, \ldots, d$ are extracted in which $\mathbf{x}_j^*$ is the eigenvector of the sample covariance matrix $\mathbf{\Sigma}$ with the largest eigenvalue λ_j. To determine the most important principal components to use, the components that explain the large ratio of variance are selected. A scree plot also can be used to identify the useable principal components.

3.2.2 Linear Discriminant Analysis

Linear discriminant analysis (LDA) is a supervised feature extraction approach that aims to extract lower dimensions of the original data. The objective is to minimize the ratio of within-class variance to between-class variance. Consider a training dataset $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ consisting of two classes C_1 and C_2, and let a unit vector $\mathbf{w} \in \mathbb{R}^d$, the optimization problem to obtain the best $\mathbf{w}$ that discriminates the two classes is

$$\max_{\mathbf{w}} \frac{\mathbf{w}^\top \mathbf{S}_b \mathbf{w}}{\mathbf{w}^\top \mathbf{S}_w \mathbf{w}}$$

where, $\mathbf{S}_b = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^\top$ is the between-class scatter matrix. The within-class scatter matrix is

$$\mathbf{S}_w = \sum_{\mathbf{x}_i \in C_1} (\mathbf{x}_i - \boldsymbol{\mu}_1)(\mathbf{x}_i - \boldsymbol{\mu}_1)^\top + \sum_{\mathbf{x}_i \in C_2} (\mathbf{x}_i - \boldsymbol{\mu}_2)(\mathbf{x}_i - \boldsymbol{\mu}_2)^\top$$

and $\boldsymbol{\mu}_j = \frac{1}{N_j} \sum_{\mathbf{x}_i \in C_j} \mathbf{x}_i$, $j = 1, 2$.

3.2.3 Least Absolute Shrinkage and Selection Operator

Least absolute shrinkage and selection operator (LASSO) is a regression analysis approach designed by Tibshirani (1996). LASSO is a robust method that is used for regularization and feature selection tasks. The main goal of LASSO is to reduce the prediction error by keeping only the features correspond to non-zero coefficients. The process penalizes coefficients of features and shrinks some to zero. One tuning parameter of LASSO controls regularization process.

Consider a response variable $\mathbf{y} \in \mathbb{R}^N$ and a matrix of predictor variables $\mathbf{X} \in \mathbb{R}^{N \times d+1}$. To estimate LASSO parameters ($\boldsymbol{\beta}$), the optimization problem is

$$\min_{\beta_0, \boldsymbol{\beta} \in \mathbb{R}^{d+1}} \|\mathbf{y} - \beta_0 - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda \|\boldsymbol{\beta}\|_1$$

where $\lambda > 0$ is a tuning parameter. The L_1 penalty, $\lambda \|\boldsymbol{\beta}\|_1$, generates sparse solutions of the previous optimization problem. Thus, the selected features are the ones that correspond to non-zero coefficients.

When a response variable $\mathbf{y} \in \{0, 1\}$ is binary, the corresponding optimization problem is

$$\min_{\beta_0, \boldsymbol{\beta} \in \mathbb{R}^{d+1}} - \left\{ \frac{1}{N} \sum_{i=1}^{N} y_i (\beta_0 + \mathbf{x}_i^\top \boldsymbol{\beta}) - \log \left(1 + \exp(\beta_0 + \mathbf{x}_i^\top \boldsymbol{\beta})\right) \right\} + \frac{\lambda}{2} \left\{ 2\alpha \|\boldsymbol{\beta}\|_1 + (1 - \alpha) \|\boldsymbol{\beta}\|_2^2 \right\}$$

where $0 \leq \alpha \leq 1$, $\alpha = 1$ for LASSO.

When a response variable $\mathbf{y}$ has C classes in which $\mathbf{y} \in \{1, 2, \ldots, c\}$, the corresponding

optimization problem is

$$-\left\{\frac{1}{N}\sum_{i=1}^{N}\left[\sum_{k=1}^{c} y_i\left(\beta_0^{(k)}+\mathbf{x}_i^{\top}\boldsymbol{\beta}^{(k)}\right)-\log\sum_{j=1}^{c}\exp\left(\beta_0^{(j)}+\mathbf{x}_i^{\top}\boldsymbol{\beta}^{(j)}\right)\right]\right\}$$
$$+\frac{\lambda}{2}\left\{2\alpha\sum_{l=1}^{d}\|\boldsymbol{\beta}_l\|_q+(1-\alpha)\|\boldsymbol{\beta}\|_F^2\right\}$$

Where $\boldsymbol{\beta}$ is $d \times C$ matrix of coefficients.

3.2.4 Fisher Scores

Fisher scores (F-scores) is a statistical feature selection technique that computes the importance of each feature in the dataset. The discriminative power of each feature is measured. F-scores measure how a feature can assign values that minimize the distance of instances in the same class whereas maximize the distance of instances from different classes. Based on F-scores, the features are ranked. The higher F-scores, the most important the features are.

To define the F-score of a feature, suppose a training dataset $\mathbf{X} \in \mathbf{R}^{N\times d}$ with a class variable that contains c categories, F-score of j^{th} feature ($\mathbf{x}_j$) is defined as follows

$$FS(\mathbf{x}_j)=\frac{\sum_{k=1}^{c} N_k(\mu_{j(k)}-\mu_j)^2}{\sum_{k=1}^{c}\sum_{i=1}^{N_k}(x_{ij(k)}-\mu_{ij(k)})^2}$$

where N_k denotes the sample size of class $k = 1, 2, \ldots, c$, μ_j is the mean of feature j, $\mu_{j(k)}$ is the mean of feature j in class k, $x_{ij(k)}$ is the sample i in feature j of the class k.

3.3 SVMTC with Feature Selection and Feature Extraction

Many methods that combine instance selection and feature reduction have been developed to accelerate the machine learning algorithms. Generally, instance selection and feature reduc-

tion are implemented separately. Suganthi and Karunakaran (2019) combined a cuttlefish optimization algorithm and principal component analysis for instance selection and feature reduction, respectively, to reduce the data prior the decision tree. In Malekipirbazari et al. (2021), random instance selection is combined with some feature selection methods such as F-scores and ReliefF. The performance of some classifiers after scaling the data down and reducing the features is computed.

On the other hand, some researcher integrates instance selection and feature reduction, simultaneously. Fragoudis et al. (2002) proposed an algorithm that combines feature and instance selection, simultaneously, for text classification. De Souza et al. (2008) developed an algorithm based on simulated annealing to combine instance and feature selection process.

In this section, we combine feature reduction methods with SVMTC in two methods. The first method is the popular way in which feature reduction is first applied then instance selection follows. We also propose another method in which feature selection is applied first to reduce the features for TC, then the original data points are used in training SVM. This method is used for feature extraction methods, PCA and LDA.

The process of Method 1 is shown as follows:

1. **(Feature reduction)** Apply the feature reduction algorithm on the entire dataset to extract the subset selected features.

2. **(SVMTC)** Apply SVMTC directly on the reduced features dataset.

   (a) **(Clustering)** Given threshold parameter t^*, cluster instances of the reduced features dataset $T^{'}$ in each class $c_j^*, j = 1, 2, \ldots, C$, separately, by using TC.

   (b) **(Create prototypes)** For each class c_j^*, compute centroids of all clusters, and add them to the reduced set $T^{''}$.

   (c) **(Training SVM)** Given $T^{''}$ from Step 2b, the cost parameter, and the kernel function, SVM is trained on $T^{''}$.

   (d) TC is repeated on the prototypes if data reduction after Step 2a is insufficient.

The process of Method 2 is shown as follows:

1. **(Feature extraction)** Apply the feature extraction (PCA or LDA) algorithm in the entire dataset to extract the subset selected features.

2. **(SVMTC)** Apply SVMTC with some changes as follows

   (a) **(Clustering)** Apply TC in the features-reduced dataset to cluster each class, separately.

   (b) **(Repetition)** If the reduction is insufficient, iterate TC on the prototypes obtained from Step 2a.

   (c) **(Create prototypes)** For each cluster from a, compute the prototypes of the entire features of datapoints that correspond to the reduced set in that cluster.

   (d) **(Training SVM)** Given the reduced data from Step 2c, the cost parameter, and the kernel function, SVM is trained on the reduced dataset of the entire features.

In Method 2, if repetition is required, prototypes obtained in 2a are computed from the features-reduced dataset which are not the same as the prototypes obtained from 2c.

We apply two methods on different real and simulated data using several feature reduction methods.

3.4 Experiments

We evaluate SVMTC under feature reduction methods using real-world datasets in the UCI Machine Learning database (Merz, 1998) and simulated data. Comparison between performance of SVMTC under the two proposed methods of feature reduction methods is carried out using several measurements. First, we evaluate runtime, which is divided into three parts. One part computes the time (seconds) of reducing the features under feature

reduction methods. The second part computes the time (seconds) of reducing the feature-reduced data under IS methods. The last part computes the time of training SVM. Memory space is used to compute the capacity (mega bytes) of reducing features, reducing data and training SVM. In addition, testing accuracy, which is the proportion of correct predicted classes of a class label in a test set to all predictions, is utilized for comparisons.

All algorithms are implemented using R programming language. The `e1071` package (Meyer et al., 2020) is used to perform SVM by using the `svm` function, which implements the normal sequential minimal optimization (SMO) algorithm. TC is executed by using the `sc_clustering` function in `scclust` package (Savje et al., 2018). We use `prcomp` function in the `stats` package to perform principal components analysis. For linear discriminate analysis, we use the `lda` function in the `MASS` package (Ripley et al., 2022). The `glmnet` function in the `glmnet` package (Friedman et al., 2022) is used to perform LASSO. For F-scores, we use the `do.fscore` function in the `Rdimtools` package (You, 2018). Some experiments are implemented on an Intel(R) Core(TM) i7-7500 CPU at 2.7 GHz processor. For big datasets, experiments are applied on an Intel(R) Core(TM) i7-7700 CPU at 3.6 GHz processor.

A stratified 5-fold cross-validation method is implemented on the most datasets to make an unbiased comparison between feature reduction methods for SVMTC. In this method, a dataset is split into five folds. then repeated five times, one fold is used for the test set; and the remaining four sets are utilized as a training set. This full process is repeated many times to decrease the impact of randomization of splitting datasets. Based on a class label, stratified splitting contributes to sampling from each class proportionally. The final results of the evaluation measures are averaged over both folds and repetitions. For datasets with a small number of instances, in each fold of 5-fold cross-validation, we perform 10-cross-validation to choose the optimal λ for LASSO. For big datasets, we select the features of the entire dataset based on the optimal λ, then these features are used in each fold of 5-fold cross-validation.

Datasets used in the experiments are described as follows:

- ***Covertype*** dataset has $580,012$ instances, and 55 attributes. The class variable has seven categories that are the types of forest cover. The dataset is also pre-processed so that the class variable becomes binary (Baker et al., 2019).

- ***Credit Card*** dataset has $30,000$ instances, and 24 attributes. The class label is to determine if a customer uses a default payment or not.

- ***HIGGS*** dataset has 11 million instances and 29 attributes. The dataset has been generated using Monte Carlo simulations, in which physicists discriminate between signal and background processes. The signal process produces Higgs particles; the background process does not. The first 21 attributes are kinematic properties. The rest are functions of the first 21 attributes.

- ***Mnist*** handwritten (0-9) digit dataset contains $70,000$ grayscale images in which each image is formed of 28×28 pixels of handwritten digit, totally 784 attributes besides the class label, which takes values $0, 1, \ldots, 9$. Cao and Boley (2006) transformed the class variable into two classes in which digits $(1, 2, 3, 4, 5)$ belong to class 1, and digits $(0, 6, 7, 8, 9)$ are transformed into class -1. In the experiment, we implemented our methods in binary and multi-class MNIST dataset.

- ***SUSY*** dataset has 5 million instances and 19 attributes. The dataset has been generated using Monte Carlo simulations, in which physicists discriminate between signal and background processes. The signal process produces supersymmetric particles, the background process does not. The first eight attributes are kinematic properties; the rest are functions of the first eight attributes.

The summary of datasets is shown in Table 3.1.

Dataset	Instances	Features	Classes	Reference
CreditCard	30,000	24	2	Yeh and Lien (2009)
Covertype-7	581,012	55	7	Blackard and Dean (1999)
HIGGS	11M	29	2	Baldi et al. (2014)
Mnist-2	70,000	785	2	LeCun et al. (1998)
Mnist-10	70,000	785	10	LeCun et al. (1998)
SUSY	5M	19	2	Baldi et al. (2014)

Table 3.1: *Datasets description*

3.4.1 Results

The results of applying our proposed methods on real datasets are discussed in this section. Based on Tables 3.2 and 3.3, the results show that the accuracy of SVMTC under feature reduction methods outperform SVMTC without feature reduction. However, SVMTC under LASSO takes less total time and memory than other feature reduction methods. Tables 3.4 and 3.5 demonstrate that our second proposed method (Method 2) outperforms Method 1. In Tables 3.6 and 3.7, several feature reduction methods improve SVMTC in term of accuracy. For the SUSY dataset, Table 3.10 shows that SVMTC under LASSO outperforms other feature reduction methods in term of accuracy, time, and memory. Tables 3.11 and 3.12 illustrate that the performance of SVMTC under LASSO is higher than other methods in terms of time and memory. In general, LASSO tends to be an effective feature selection method, and overall, it is shown that SVMTC is improved significantly under the proposed methods.

3.5 Discussion

Many methods have been developed to deal with data with large number of features. Feature reduction is one popular preprocessing method to extract the most relevant features and remove redundant ones. Feature selection (FS) and feature extraction (FE) are the most popular techniques to reduce the features. FS methods tend to select a subset of original features to improve the performance of classification by minimizing the redundant features;

FR method	PCA-1			PCA-2			LDA-1	LDA-2
Feature size	5	10	15	5	10	15	1	1
FR Time	0.0976	0.0976	0.0976	0.0976	0.0976	0.0976	0.1376	0.1376
FR Mem.	47.928	47.928	47.928	47.928	47.928	47.928	76.432	76.432
Clust. Time	0.1176	0.3408	1.032	0.2008	0.456	1.06	0.052	0.0896
Clust. Mem.	8.036	11.624	17.988	20.98	34.232	49.276	3.928	20.036
Train. Time	11.6216	17.5256	28.3064	31.0464	31.6464	33.2264	7.9384	38.0168
Train. Mem.	10.82	17.864	25.132	39.768	37.896	35.8	3.976	39.224
Total Time	11.8368	17.964	29.436	31.3448	32.2	34.384	8.128	38.244
Total Mem..	66.784	77.416	91.048	108.676	120.056	133.004	84.336	135.692
Accuracy	80.36	80.28	80.59	79.85	79.90	79.93	81.62	79.45
Red. Rate	58.51	58.63	58.70	58.51	58.63	58.70	56.72	56.72
SV	5,274	6,326	7,324	7,547	7,679	7,771	3,988	8,635

Table 3.2: *Performance of SVM for Credit Card dataset using SVMTC with PCA and LDA under methods 1 and 2.*

FR method	FS-1	LASSO-1	SVMTC	SMO
Feature size	10	15	23	23
FR Time	0.1968	0.0592	-	-
FR Mem.	42.376	14.332	-	-
Clust. Time	0.1632	0.0928	0.3648	-
Clust. Mem.	16.00	12.892	22.6	-
Train. Time	20.2016	14.9064	30.9848	258.7864
Train. Mem.	23.152	19.916	34.72	82.136
Total Time	20.5616	15.0584	31.3496	258.7864
Total Mem..	81.528	47.14	57.32	82.136
Accuracy	81.11	81.12	79.69	80.25
Red. Rate	58.69	58.33	58.63	-
SV	6,156	5,503	7,679	16,837

Table 3.3: *Performance of SVM for Credit Card dataset using SVMTC with Fisher Scores and LASSO, SVMTC without feature reduction, and SVM on original data.*

FR method	PCA-1				PCA-2			
Feature size	5	10	15	20	5	10	15	20
FR Time	8.6616	8.6616	8.6616	8.6616	8.6616	8.6616	8.6616	8.6616
FR Mem.	2,999.092	2,999.092	2,999.092	2,999.092	2,999.092	2,999.092	2,999.092	2,999.092
Clust. Time	3.2496	5.1688	6.6168	7.1648	10.9968	9.072	13.2168	13.3584
Clust. Mem.	287.7	489.756	689.648	884.7	4,526.844	4,191.556	4,690.904	4,865.884
Train. Time	2,069.8504	3,405.1176	3,972.9704	4,345.212	11,623.9792	10,292.9712	10,853.9536	10,883.396
Train. Mem.	204.78	296.504	396.296	497.904	1,143.172	1,092.544	1,131.664	1,144.996
Total Time	2,081.7616	3,418.948	3,988.2488	4,361.0384	11,635.0736	10,302.1408	10,867.268	10,896.852
Total Mem.	3,491.572	3,785.352	4,085.036	4,381.696	5,717.944	5,332.028	5,870.496	6,058.808
Accuracy	79.70	89.12	89.41	89.56	88.92	92.51	92.55	92.58
Red. Rate	75.01	74.92	74.91	74.92	75.01	74.92	74.91	74.92
SV	63,147	54,560	54,356	54,226	63,637	61,745	61,869	61,933

Table 3.4: *Performance of SVM for Covertype dataset using SVMTC with PCA under methods 1 and 2.*

FR method	LDA-1	LDA-2	FS-1	LASSO-1	SVMTC
Feature Size	6	6	30		54
FR Time	11.1972	11.1972	6.5704	533.8	-
FR Mem.	4,510.396	4,510.396	3,084.06	4,726.02	-
Clust. Time	4.132	6,497.2804	2.1512	4.1216	6.2848
Clust. Mem.	334.264	1,906.4	1,822.5	1,314.696	2,306.044
Train. Time	1,579.9656	12,400.5552	7,269.4576	13,201.1136	14,897.2984
Train. Mem.	220.04	1,142.376	712.396	947.148	1,203.972
Total Time	1,595.2948	18,909.0326	7,278.1792	13,739.0352	14,903.5832
Total Mem.	5,064.7	7,559.172	5,111.152	6,987.864	3,510.016
Accuracy	79.67	90.43	89.25	85.45	89.61
Red. Rate	75.08	75.08	74.02	74.44	74.09
SV	57,010	66,947	58,0223	72,176	71,407

Table 3.5: *Performance of SVM for Covertype dataset using SVMTC with LDA under methods 1 and 2, Fisher Scores and LASSO and SVMTC without feature reduction.*

FR method	PCA-1						PCA-2					
Feature size	5	10	15	30	100	150	5	10	15	30	100	150
FR Time	59.52	59.52	59.52	59.52	59.52	59.52	59.52	59.52	59.52	59.52	59.52	59.52
FR Mem.	2,151.1	2,151.1	2,151.1	2,151.1	2,151.1	2,151.1	2,151.1	2,151.1	2,151.1	2,151.1	2,151.1	2,151.1
Clust. Time	0.28	1.3	10.62	18.72	34.38	49.76	15.76	19.08	63.88	7.56	65.72	68.7
Clust. Mem.	17.1	58.8	100.10	173.30	595.80	919.00	4,002.3	3,836.3	4,228.7	4,141.5	4,382.6	4,557.9
Train. Time	18.3	7.36	9.38	11.40	32.52	45.32	154.22	173.32	207.90	158.18	159.18	157.48
Train. Mem.	28.9	42.2	50.50	70.70	200.40	365.80	1,870.80	1,897.00	1,928.10	1,824.2	1,833.1	1,817.3
Total Time	78.1	68.18	203.68	89.64	132.6	172.66	229.5	251.92	331.3	225.26	284.42	285.7
Total Mem.	2,197.1	2,252.1	2,691.1	2,395.1	3,497.5	4,004.7	8,024.2	7,884.4	8,307.9	4,116.8	8,366.8	8,526.3
Accuracy	65.81	92.23	95.98	97.60	98.15	98.02	96.77	97.68	97.91	97.77	97.89	97.9
Red. Rate	75.1	75.91	76.11	76.30	76.56	76.69	75.1	75.91	76.11	76.30	76.56	76.69
SV	9,935	3,700	2,943	3,511	4,803	5,018	4,117	4,843	5,024	5,304	5,337	5,307

Table 3.6: *Performance of SVM for Mnist-10 dataset using SVMTC with PCA under methods 1 and 2.*

FR method	LDA-1	LDA-2	FS-1			LASSO-1	SVMTC	SMO
Feature Size	9	9	50	100	150		784	784
FR Time	260.76	260.76	4.92	6.52	9	4,362.92	-	-
FR Mem.	5,836.70	5,836.70	4,794	5,080.10	5,429.10	1,114.8	-	-
Clust. Time	8.70	74.74	9.92	35.46	50.72	228.02	357.56	-
Clust. Mem.	66.10	3079.1	247.20	574.10	897	3,122.3	4,805.10	-
Train. Time	10.56	192.30	46.64	54.12	78.34	154.86	233.36	5,000.30
Train. Mem.	38.70	2,367.00	144.30	248.30	346.10	1,248.3	2,262.60	7,666.00
Total Time	280.02	527.8	61.48	96.1	138.06	4,641.96	590.92	5,000.3
Total Mem.	5,941.5	11,282.8	5,188.5	5,902.5	6,672.2	5,485.4	7,067.7	7,666
Accuracy	91.92	97.29	83.88	93.23	95.27	97.96	97.87	98.51
Red. Rate	75.91	75.91	76.18	76.56	76.76	76.98	76.96	-
SV	3,110	4,453	6,555	4,205	4,009	5,297	5,362	19,134

Table 3.7: *Performance of SVM for Mnist-10 dataset using SVMTC with LDA under methods 1 and 2, Fisher Scores and LASSO, SVMTC without feature reduction, and SVM on original data.*

FR method	PCA-1						PCA-2					
Feature size	5	10	15	30	100	150	5	10	15	30	100	150
FR Time	286.48	286.48	286.48	286.48	286.48	286.48	286.48	286.48	286.48	286.48	286.48	286.48
FR Mem.	2,327.7	2,327.7	2,327.7	2,327.7	2,327.7	2,327.7	2,327.7	2,327.7	2,327.7	2,327.7	2,327.7	2,327.7
Clust. Time	0.4	2.44	15.20	49.20	132.70	151.84	144.38	88.38	183.74	140.78	84.9	76.88
Clust. Mem.	20.9	38.9	59.20	110.10	336.90	549.00	2,134.8	1,693.7	1,246	1,754.9	2,066.1	2,382.1
Train. Time	16.42	6.86	12.18	13.06	48.04	96.86	194.82	256.06	271.20	211.84	195.28	195.96
Train. Mem.	11.4	27.5	36.9	67.1	222.1	315.3	1,849.8	1,850.4	1,763.4	1,238.1	1,807.6	1,710.7
Total Time	303.3	295.78	313.86	348.74	467.22	535.18	625.68	630.92	741.42	639.1	566.66	559.32
Total Mem.	2,360.0	2,394.1	2,423.8	2,504.9	2,886.7	3,192	6,312.3	5,871.8	5,337.1	5,320.7	6,201.4	6,420.7
Accuracy	79.95	93.85	96.88	98.00	98.48	98.42	97.23	98.20	98.46	98.44	98.43	98.41
Red. Rate	75.13	75.82	76.07	76.34	76.57	76.66	75.13	75.82	76.07	76.34	76.57	76.66
SV	7,414	2,789	1,775	1,732	2,718	2,921	2,027	2,771	2,974	3,140	3,228	3,231

Table 3.8: *Performance of SVM for Mnist-2 dataset using SVMTC with PCA under methods 1 and 2.*

FR method	LDA-1	LDA-2	FS-1			LASSO-1	SVMTC	SMO
Feature size	1	1	50	100	150		784	784
FR Time	281.66	281.66	4.22	5.96	9.96	13.42	-	-
FR Mem.	5,492.10	5,492.10	2,156.20	1,963.90	2,240.20	729.10	-	-
Clust. Time	0.44	144.14	54.16	169.12	241.22	459.78	2,506.64	-
Clust. Mem.	11.30	2,499.8	189.40	343.60	519.40	1,261.00	2,893.80	-
Train. Time	23.26	416.82	54.70	161.80	227.30	108.24	322.42	4,906.40
Train. Mem.	5.90	2,570.20	125.00	239.10	370.60	798.70	2,197.90	7,443.20
Total Time	305.36	842.62	113.08	336.88	478.48	581.44	2,829.06	4,906.4
Total Mem.	5,509.3	10,562.1	2,470.6	2,546.6	3,130.2	2,788.8	5,091.7	7,443.2
Accuracy	86.34	92.02	92.35	95.05	94.70	98.28	98.39	98.77
Red. Rate	73.05	73.05	76.17	76.56	76.78	77.06	76.97	-
SV	5,281	3,154	4,546	6,751	9,687	2,467.00	3,209	12,656

Table 3.9: *Performance of SVM for Mnist-2 dataset using SVMTC with LDA under methods 1 and 2, Fisher Scores and LASSO, SVMTC without feature reduction, and SVM on original data.*

FR method	PCA-1		PCA-2		LDA-1	LDA-2	FS-1	LASSO-1	SVMTC
Feature size	5	10	5	10	1	1	10	–	18
FR Time	4.9376	4.9376	4.9376	4.9376	15.9160	15.9160	17.7464	111.876	-
FR Mem.	3,983.3040	3,983.3040	3,983.3040	3,983.3040	14,195.9200	14,195.9200	6,033.6920	1,473.112	-
Clust. Time	105.4792	999.4376	120.5288	1,016.8004	16.9696	18.6168	398.2392	372.092	2,268.901
Clust. Mem.	1,223.3680	1,947.8320	5,004.912	6,686.096	625.6480	3,365.884	1,953.9920	1,786.944	3,105.064
Train. Time	5,764.5776	6,589.9808	1,321.8968	5,050.002	3,045.4424	1,509.4704	1,959.7200	2,520.934	7,341.221
Train. Mem.	133.0760	220.9000	330.3880	324.596	80.5640	525.6240	234.2240	218.288	378.720
Total Time	5,874.9944	7,594.3560	1,447.3632	6,071.74	3,078.3280	1,544.0032	2,375.7056	3,004.902	9,610.122
Total Mem.	5,339.7480	6,152.0360	9,318.604	10,993.996	14,902.1320	18,087.428	8,221.9080	3,478.344	3,483.784
Accuracy	75.68	77.27	72.20	76.05	76.81	51.69	78.79	79.26	77.12
Red. Rate	97.06	97.24	97.06	97.24	96.03	95.26	97.13	97.14	97.26
SV	66,499	70,019	24,904	65,224	76,278	26,731	52,081	54,508	74,004

Table 3.10: *Performance of SVM for SUSY dataset using SVMTC with PCA and LDA under methods 1 and 2, Fisher Scores, LASSO, and SVMTC without feature reduction*

FR method	PCA-1		PCA-2		LDA-1	LDA-2	FS-1	LASSO-1	SVMTC
Feature size	5	10	5	10	1	1	10	–	28
FR Time	25.98	25.98	25.98	25.98	153.98	153.98	332.64	240.92	-
FR Mem.	14,634.5	14,634.5	14,634.5	14,634.5	105,407.7	105,407.7	51,255.6	3,487.5	-
Clust. Time	1,958.82	2,062.36	290.34	2,642.22	50.22	601.28	2,387.86	7,157.18	172,983.9
Clust. Mem.	6,225.1	4,358.9	44,120.5	110,127.2	1,442.9	101,921.9	4,372.4	6,267.3	10,298.6
Train. Time	345.02	691.26	10.46	121.8	564.52	58.14	593.02	545.5	438.92
Train. Mem.	214.5	161.9	198.8	197.8	53.2	418.1	160.1	202.3	350.6
Total Time	2,329.82	2,779.6	326.78	2,790.0	768.72	813.4	3,313.52	7,943.6	173,421.92
Total Mem.	21,074.1	19,155.3	58,953.8	124,959.5	106,903.8	207,747.7	55,788.1	9,957.1	10,649.2
Accuracy	54.28	56.11	52.99	53.00	64.07	52.99	67.08	64.97	52.99
Red. Rate	99.59	99.55	99.49	99.55	99.19	99.19	99.55	99.58	99.63
SV	36,096	39,228	21,476	17,894	51,889	45,437	28,823	34,572	32,952

Table 3.11: *Performance of SVM for HIGGS dataset using SVMTC with PCA and LDA under methods 1 and 2, Fisher Scores, LASSO, and SVMTC without feature reduction*

FR method	PCA-1		PCA-2		LDA-1	LDA-2	FS-1	LASSO-1	SVMTC
Feature size	12	15	12	15	1	1	10	–	29
FR Time	25.82	25.82	25.82	25.82	122.2	122.2	90.92	244.0	-
FR Mem.	14,902.8	14,902.8	14,902.8	14,902.8	138,039	138,039	85,299.4	3,488	-
Clust. Time	2,065.86	6,533	2,225.84	6,660.86	50.88	168.82	2,425.16	7,233.2	170,396.7
Clust. Mem.	4,405.2	5,005.1	109,116.1	141,172.4	1,570.9	74,880.8	4,338.8	6,367.8	10,160.8
Train. Time	697.92	394.46	504.64	26.16	27.88	97.8	589.26	541.88	437.36
Train. Mem.	161.9	176.7	423.9	119.6	14.7	653.2	165.3	240.8	350.6
Total Time	2,789.6	6,920.28	2,756.28	6,712.84	200.96	388.82	3,105.34	8,019.08	170,834.06
Total Mem.	19,469.9	20,084.6	124,442.8	156,194.8	139,624.6	213,573.0	89,803.5	10,096.6	10,511.4
Accuracy	56.11	57.34	52.99	53.01	64.07	52.99	67.08	64.97	52.99
Red. Rate	99.55	99.56	99.55	99.56	99.19	99.19	99.55	99.58	99.63
SV	39,228	38,330	39,958	38,356	51,889	70,921	28,823	34,572	32,952

Table 3.12: *Performance of SVM for HIGGS dataset using SVMTC with PCA and LDA under methods 1 and 2, Fisher Scores, LASSO, and SVMTC without feature reduction*

FE methods aim to extract new features by projecting the data onto a new feature space.

In Chapter 1.5, we proposed an IS method using TC to accelerate training SVM. However, TC works poorly in data with large number of features. Thus, we proposed adding a step of feature reduction methods in SVMTC algorithm so that TC works ideally to accelerate training SVM. In this chapter, we combined feature reduction and IS to improve SVM performance. We utilize different feature reduction methods as a preprocessing step to our proposed IS method, SVMTC. We propose two ways to reduce the dimensions before IS. One way is based on using the extracted features from FE methods; the other aims to use the reduced original features based on FS methods. we show, via application to datasets that reducing features by using feature reduction methods can improve the performance of SVMTC.

Chapter 4

Comparative Study of Iterative Threshold Instance Selection and K-means

4.1 Introduction

Instance selection (IS) is a common data mining process that is used with large amount of data to reduce the size (Liu and Motoda, 2002). It plays an important role in data mining by providing relevant data and discarding superfluous ones. Applying selected data on machine learning algorithms reduces runtime without losing integrity of data (Olvera-López et al., 2010a). IS can be applied by sampling, classification, or clustering. In real applications, the majority of data are without class values. IS methods that are based on classification are extremely beneficial but cannot be applied to unlabled data directly. In contrast, methods associated with clustering are useful for unlabled data.

Liu and Motoda (2002) review some methods relies on clustering. One generates pseudo points from clusters, called prototypes, and then uses these prototypes instead of working with all data points. One well-known example is the k-means clustering algorithm. Another

method is data description in a hierarchy. One example of this is COBWEB which improves the idea of conceptual clustering by applying an incremental method (Fisher, 1987). Also, there are IS methods based on squashing data. Squashing compresses an original dataset without losing any statistical information. The process passes through grouping, momentizing and generating (GMG) sequentially (DuMouchel et al., 1999). In this chapter, we focus on the method that relies on constructing prototypes.

In this chapter, we use k-means and some of its variants as IS methods and compare with iterative threshold instance selection (ITIS) that is proposed by (Luo et al., 2019). Then, we analyze the performance of k-means clustering, HAC, and DBSCAN after implementing the reduction methods. By using k-means, hierarchical agglomerative clustering (HAC), and density based spatial clustering of applications with noise (DBSCAN), we cluster the prototypes obtained from the IS methods. Then for each original unit, we assign it back to the cluster that belongs to the prototype of the original unit that is used to obtain that prototype Some performance measurements are computed for comparisons. Additionally, we compare k-means after ITIS with stabilized hybrid clustering (SHC) (Amiri et al., 2019), and hybrid hierarchical method (Chipman and Tibshirani, 2006) with HAC after ITIS.

By using simulations, we illustrate that the performance of clustering after ITIS is more accurate, with reducing runtime and memory usage, than other competing methods in most cases. By increasing sample size, ITIS works more effectively than other methods. Under clustering by DBSCAN, ITIS and k-means work effectively.

The rest of this chapter is organized as follows. Section 4.2 reviews ITIS, k-means, HAC, DBSCAN, SHC, and hybrid hierarchical clustering. Simulation and results are covered in section 4.3.1. The last section 4.4 includes general discussion.

4.2 Preliminaries

In this section, a brief review of ITIS, k-means and its variations, and hybrid clustering methods is presented.

4.2.1 Iterative Threshold Instance Selection

Iterative Threshold Instance Selection (ITIS) (Luo et al., 2019) is a new method of instance selection (IS) based on recent efficient clustering called threshold clustering (TC) (Higgins et al., 2016). The main advantage of TC is its ability to cluster the data a prespecified number of data points in each cluster. In addition, TC allows clustering with small maximum within-cluster dissimilarity to an average within-cluster dissimilarity. The idea of ITIS is to decrease the data size by a factor of α. The algorithm performs in the following steps:

1. Proceed threshold clustering on data of size n to make n^* clusters each one has t^* or more, where t^* is a small size threshold.

2. For each cluster, compute a center point to form n^* prototypes.

3. Terminate if the data is reduced by a factor of α or go to Step 1 after replacing data of size n with n^* prototypes.

ITIS works effectively in a large dataset with runtime $O(t^*mn \log n)$, where m is the number of iterations. However, it has a limitation when sample size is small becasue the similarity between each prototype and its data points becomes small.

4.2.2 Iterative Hybridized Threshold Clustering

Luo et al. (2019) proposed a novel clustering method called iterative hybridized threshold clustering (IHTC) based on ITIS. This method allows traditional clustering such as k-means, HAC and DBSCAN to work effectively in massive data. The idea is to use ITIS as a

preprocessing step before applying traditional clustering methods. IHTC goes through the following steps:

1. Based on threshold parameter t^* and the number of iterations m, ITIS is applied on the dataset of size n to obtain the prototypes.

2. By using a clustering method such as k-means, cluster the prototypes computed from Step 1.

3. For every prototype, assign the original instances that are a part of computing a prototype to the cluster belonging to that prototype.

IHTC can reduce the size of data and can eliminate overfitting of instances so that the performance of clustering algorithm is improved.

4.2.3 K-means Clustering

The k-means clustering (Lloyd, 1982) is Lloyd's method in which initial values of centers are chosen randomly. The method is based on partitioning data points into prespecified k clusters. The k-means algorithm is performed as follows:

1. Select k data points uniformly at random from the dataset and set them as centers.

2. Compute the squared Euclidean distance between all units and centers. Assign each unit to the nearest center to perform k clusters.

3. Recompute the average of each cluster and replace centers with them.

4. Repeat Steps 2 and 3, then terminate when there are no changes for the centers.

The k-means clustering can be utilized as an IS method (MacQueen, 1967) by partitioning the data into k clusters, then the centroids of clusters are dealt as a prototype. One drawback of k-means is the sensitivity to the initialization. One way to improve the speed of k-means is to apply it in a parallel way.

4.2.4 K-means++ Clustering

The k-means++ clustering (Arthur and Vassilvitskii, 2006) overcomes the drawback of initializing the centers randomly in k-means by specifying a way of choosing them. It is performed as k-means except the initialization in Step 1 is developed as follows

1. Select one center uniformly at random, say c_1, from data points.

2. For $i = 1, 2, .., k$, select a new center c_i, choosing x from data points with probability $\frac{D(x)^2}{\sum_x D(x)^2}$, where $D(x)$ is the shortest distance from a record to the closest center.

The remaining steps are proceed as k-means Steps 2-4.

4.2.5 Fuzzy C-means Clustering

The fuzzy C-means clustering (Bezdek, 1981) is an extension of k-means to overcome its limitations. The basic idea is that each data point belongs to many clusters with a different degree of membership between 0 and 1. The fuzzy C-means algorithm steps are as follows

1. Initialize a fuzzy partition matrix randomly.

2. Calculate c centers.

3. Calculate fuzzy partition values based on distances.

4. Compare the matrices of iteration t and $t+1$, stop if there are no differences, otherwise go to Step 2.

One drawback of the fuzzy C-means is that it takes a long computational time.

4.2.6 Stabilized Hybrid Clustering

Stabilized Hybrid Clustering (SHC) is a hybrid clustering method proposed by Amiri et al. (2019). For a given k, the clusters are obtained in the following way.

1. K-means is applied to obtain much more (small) clusters than k. Then by using single linkage clustering, these small clusters are joined.

2. Recluster the results from the first stage to obtain a dendrogram. This stage is called the stabilization stage.

3. The dendrogram that is obtained from Step 2 is cut to get clusters greater than or equal to k. Then, the clusters are merged to obtain exactly k clusters.

SHC is able to cluster even with non-convex clusters.

4.2.7 Hybrid Hierarchical Clustering

Hybrid hierarchical clustering (Chipman and Tibshirani, 2006) combines agglomerative (bottom-up) hierarchical clustering and top-down clustering. The method based on making a group of objects that is closer to each other than to any other object; this group is called a mutual cluster. This type of cluster does not break by bottom-up clustering. The steps of clustering data are as follows.

1. Compute mutual clusters.

2. Tree structured vector quantization (TSVQ) the top-down clustering is performed with keeping each mutual cluster right. This step is completed by replacing the mutual clusters by their centroid.

3. Execute a top-down clustering within each mutual cluster.

This method overcomes the weaknesses of top-down clustering and bottom-up by combining advantages of both. As data size increases, mutual clusters can be affected especially with low dimensions data.

4.3 Process of Comparisons of ITIS and K-means

To evaluate ITIS, we compare ITIS to other IS methods under traditional clustering methods based on runtime, memory usage, and prediction accuracy. The general process is shown as follows.

1. Apply an IS method on the simulated data to obtain the reduced set.
2. Implement the clustering method on the reduced set from Step 1.
3. For each data point $\mathbf{x}^{'}$ of the reduced set, assign the original instances that are a part of computing $\mathbf{x}^{'}$ to the cluster belonging to $\mathbf{x}^{'}$.
4. Obtain the clustering performance on the ultimate clusters from Step 3.

We simulated data with different sizes to compare between ITIS and the different competitive IS methods.

4.3.1 Simulation

We compare ITIS with k-means and its variations using simulated data. These methods are applied to samples of sizes $(10^4, 10^5, 10^6)$. The data is sampled from a mixture distribution of weighted combinations of three bivariate Normal distributions. The pdf is as follows.

$$f(\mathbf{x}) = 0.3p(\mathbf{x}|\mu_1, \mathbf{\Sigma}_1) + 0.6p(\mathbf{x}|\mu_2, \mathbf{\Sigma}_2) + 0.2p(\mathbf{x}|\mu_3, \mathbf{\Sigma}_3)$$

for $j = 1, 2, 3$, $p(\mathbf{x}|\mu_j, \mathbf{\Sigma}_j)$ is the pdf of Normal distribution with parameters μ_j and $\mathbf{\Sigma}_j$.

$$\mu_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \mu_2 = \begin{bmatrix} 7 \\ 8 \end{bmatrix}, \mu_3 = \begin{bmatrix} 3 \\ 5 \end{bmatrix}$$

$$\boldsymbol{\Sigma}_1 = \begin{bmatrix} 1 & 0 \\ 0 & 0.5 \end{bmatrix}, \boldsymbol{\Sigma}_2 = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}, \boldsymbol{\Sigma}_3 = \begin{bmatrix} 3 & 0 \\ 0 & 4 \end{bmatrix}$$

For sample sizes of $10,000$ and $100,000$, each case is replicated $1,000$ times; with $1,000,000$ it is replicated 100 times becasue of lengthy time to execute k-means and its variations. In each replicate, the execution time in seconds, memory in megabytes, prediction accuracy and clustering validation methods are computed. All implementations are executed in the R programming language. The `Itis` package is used to perform ITIS. We use the `clusternor` package to perform k-means++ and fuzzy C-means (Mhembere, 2020). The default `kmeans`, `hclust`, and `dbscan` functions are used to cluster prototypes by k-means, HAC, and DBSCAN, respectively. Additionally, an R library, `GHC`, in `GitHub` is used to implement Stabilized Hybrid Clustering (Saeid et al., 2019). To implement the hybrid hierarchical clustering, we use R source codes which are available in Chipman and Tibshirani (2015). For computing `Silhouette` measurement, the `clusterCrit` package is used (Desgraupes, 2018). Some parts of simulation are performed on the Beocat Research Cluster at Kansas State University. The Beocat is funded in part by NSF grants CNS-1006860, EPS-1006860, and EPS-0919443 (University, 2018). All simulations are implemented on Intel(R) Core(TM) i7-7500 CPU at 2.7 GHz processor.

4.3.2 Clustering Based on K-means

By using simulated data of sizes $(10^4, 10^5, 10^6)$, we obtain prototypes from each of ITIS with $m = 3$ and $t = 2$, k-means, k-means++, parallel k-means, and fuzzy C-means with k values $(10^3, 10^4, 10^5)$, respectively. Then, we cluster prototypes from each method by k-means with $k = 3$ because the data is sampled from the mixture with three bivariate normal distributions. We assign each original unit back to the cluster that belongs to the prototype from which the original unit is used to obtain that prototype. To reduce the runtime, we

exclude fuzzy C-means when sample size is more than 10^4, and k-means++ is discarded when $N = 10^6$. The runtime and memory usage are computed for each method. Additionally, to measure prediction accuracy of clustering, we considered that each data comes from independent Normal distribution in a cluster. The prediction accuracy computes the data points correctly clustered divided by the sample size. Cluster validation methods, Silhouette and R^2, are also computed to measure the performance of clustering. Computing Silhouette was excluded when sample size increases. Silhouette (Rousseeuw, 1987) is used to test the consistency of the clustering with range $(-1, 1)$, in which a large value shows that each unit matches its cluster correctly. R^2 is a ratio of sum of squares between clusters to total sum of squares with range $(0, 1)$ in which a large value indicates small variance of clusters. Table 1 displays the results. We notice from the results that clustering after ITIS works more efficiently as data size increases. In general, using ITIS consumes less time and memory while preserving prediction accuracy in most cases.

4.3.3 Clustering Based on HAC

HAC, proposed by Ward Jr (1963) is a well-known bottom-up clustering method in which each data point is considered a cluster and then every two clusters are merged. The process ends up with only one cluster. In this part of simulation, we compute prototypes by using ITIS with $t = 2$, and each of k-means, k-means++, parallel k-means, and fuzzy C-means with k values $(10^3, 10^4, 10^4)$; from dataset of sizes $(10^4, 10^5, 10^6)$, respectively. For ITIS, we use $m = 3$ when the data size 10^4 and 10^5, and we increase the number of iterations to $m = 6$, with data size 10^6. Then, we cluster the prototypes from each method by HAC. We assign each original unit back to the cluster that belongs to its prototype. Again, to reduce the runtime, we exclude fuzzy C-means when sample size is more than 10^4, and k-means++ is discarded when $N = 10^6$. The runtime and memory usage are computed for each method. In addition, prediction accuracy and cluster validation methods (Silhouette

Performance	Memory Usage (Mb)			Runtime (second)			Accuracy (%)		
	10^4	10^5	10^6	10^4	10^5	10^6	10^4	10^5	10^6
K-means	0.8772	36.7094	402.3580	1.1062	15.52238	1,296.618	85.72280	85.98123	85.99308
K-means++	0.3254	19.929	-	1.03534	8.59728	-	85.94125	85.99829	-
Parallel K-means	10.4371	66.5162	703.5770	0.99148	1.31884	14.34380	84.26776	84.73853	85.82207
Fuzzy C-means	0.4448	-	-	167.54508	-	-	85.87573	-	-
ITIS	0.9483	17.7762	197.324	0.99182	1.43424	6.5514	85.85908	85.99926	86.00827

Table 4.1: *Clustering performance using (runtime, memory usage and accuracy) based on k-means.*

Performance	Silhouette			R^2			Number of Prototypes		
	10^4	10^5	10^6	10^4	10^5	10^6	10^4	10^5	10^6
K-means	0.47051	-	-	0.78846	0.78886	0.788882	1,000	10,000	100,000
K-means++	0.46987	-	-	0.78792	0.78880	-	1,000	10,000	-
Parallel K-means	0.44031	-	-	0.77276	0.78588	0.78853	1,000	10,000	100,000
Fuzzy C-means	0.47011	-	-	0.78805	-	-	1,000	-	-
ITIS	0.47033	-	-	0.78839	0.78885	0.788881	741	7,368	73,593

Table 4.2: *Clustering performance using (Silhouette, R^2 and the size of reduced data) based on k-means.*

and R^2) are computed. We exclude computing Silhouette when sample size increases. The results are shown in Table 2. It is clear that IHTC with HAC is more accurate than other methods in all cases while using the least time and memory.

4.3.4 Clustering Based on DBSCAN

DBSCAN was first proposed by Ester et al. (1996) and is based on density-based of clusters. It can cluster data with arbitrary shape. The DBSCAN algorithm relies on determining two parameters; ϵ which is a radius of neighborhoods of each unit, and minimum amounts of units to form a cluster. In this case the simulated data is constructed of two rings, the small ring is inside the large one, see Figure 4.1. From data of sizes ($10^4, 10^5$,10^6), we obtain prototypes by ITIS with $t = 2$ and $m = 3$, and with k-means and parallel k-means with k values ($10^3, 10^4$,10^5). Then, we cluster prototypes from each method by DBSCAN with $\epsilon = 1.5$ and default minimum amounts 5. We assign each original data point back to the cluster that belongs to its prototype. The runtime and memory usage are computed for each method. In addition, prediction accuracy and cluster validation methods are computed. The results are shown in Table 3. The results show that clustering after ITIS consumes short time with less memory usage. It is clear that all methods are accurate, and parallel k-means consumes the same time as ITIS; however, ITIS uses less memory than others.

4.3.5 IHTC with K-means Versus SHC

We also implement IHTC with k-means and SHC on the simulated data of size 10^3 only because SHC is running slowly with big data. We compute runtime and memory usage for each method along with prediction accuracy and cluster validation methods. The results of this simulation are shown in Table 4. The results demonstrate that the performance of IHTC with k-means is superior to SHC.

Performance	Memory Usage (Mb)			Runtime (second)			Accuracy (%)		
	10^4	10^5	10^6	10^4	10^5	10^6	10^4	10^5	10^6
K-means	12.0332	1,158.42	1,231.39	0.48426	46.39738	770.749	85.53338	86.36167	85.48064
K-means++	11.1426	994.4577	-	0.40888	25.80094	-	85.75031	86.5734	-
Parallel K-means	21.0693	1,189.962	1,208.019	0.14778	9.98114	19.9554	85.05894	84.80141	82.67686
Fuzzy C-means	11.0221	-	-	300.8313	-	-	85.87119	-	-
ITIS	6.7651	575.8104	402.486	0.07472	5.3326	12.9896	86.1677	86.4951	86.19625

Table 4.3: *Clustering performance using (runtime, memory usage and accuracy) based on HAC.*

Performance	Silhouette			R^2			Number of Prototypes		
	10^4	10^5	10^6	10^4	10^5	10^6	10^4	10^5	10^6
K-means	0.44549	-	-	0.76385	0.76503	0.76293	1,000	10,000	10,000
K-means++	0.44635	-	-	0.76441	0.76577	-	1,000	10,000	-
Parallel K-means	0.42700	-	-	0.75544	0.75889	0.75323	1,000	10,000	10,000
Fuzzy C-means	0.44726	-	-	0.76483	-	-	1,000	-	-
ITIS	0.44738	-	-	0.76492	0.76602	0.76654	742	7,368	5,454

Table 4.4: *Clustering performance using (Silhouette, R^2 and the size of reduced data) based on HAC.*

Performance	Memory Usage (Mb)			Runtime (second)			Accuracy (%)		
	10^4	10^5	10^6	10^4	10^5	10^6	10^4	10^5	10^6
K-means	1.122	16.1505	251.0076	0.14138	8.88678	954.1226	99.84207	99.9974	99.99816
Parallel K-means	10.1971	48.05100	594.8377	0.05304	0.19402	9.12592	99.97291	99.9953	99.99852
ITIS	1.3929	6.0004	64.3945	0.03744	0.24464	4.14304	99.92118	99.99998	99.99984
Raw Data	0.0342	0.4399	-	0.04566	1.74306	-	99.9828	99.9994	-

Table 4.5: *Clustering performance using (runtime, memory usage and accuracy) based on DBSCAN.*

Performance	Silhouette			Number of Prototypes		
	10^4	10^5	10^6	10^4	10^5	10^6
K-means	0.251318	0.251225	-	1,000	10,000	100,000
Parallel K-means	0.251240	0.251226	-	1,000	10,000	100,000
ITIS	0.251239	0.251222	-	748	7,401	73,659.67
Raw Data	0.251241	0.251223	-	-	-	-

Table 4.6: *Clustering performance using (Silhouette, and the size of reduced data) based on DBSCAN*

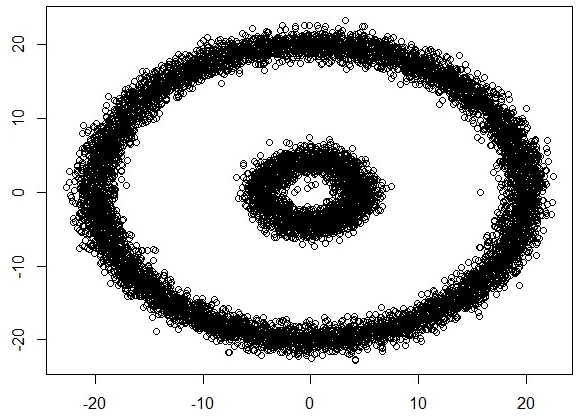

Figure 4.1: *Data is formed of outer ring with* 7700 *units and inner ring of* 2300 *units.*

	SHC	**IHTC with K-means**
Memory (Mb)	10,047.15	0.2098
Time(Second)	655.1712	4.6749
Accuracy(%)	61.6048	85.2516
Silhouette	0.4250	0.4656
R^2	0.5131	0.7859

Table 4.7: *Comparison between SHC and IHTC with k-means* ($N = 10^3$)

4.3.6 IHTC with HAC Versus Hybrid Hierarchical Clustering

We implement IHTC with HAC; and Hybrid Hierarchical Clustering (hybridHclust) on data that is sampled from a mixture distribution of size 10^4. Then, we compute runtime memory usage, prediction accuracy and cluster validation for each method. Table 5 represents the results of this simulation. The results illustrate that IHTC with HAC uses significantly less time and memory than the hybrid hierarchical clustering; with 6% more accuracy.

	hybridHclust	**IHTC with HAC**
Memory (Mb)	59,475.9	6.9816
Time(Second)	239.6496	0.0763
Accuracy(%)	80.4631	86.2058
Silhouette	0.4137	0.4473
R^2	0.7490	0.7650

Table 4.8: *Comparison between hybridHclust and IHTC with HAC ($N = 10^4$)*

4.4 Discussion

Clustering is an unsupervised learning method that can be performed on unlabled data. There are a minority of instance selection (IS) methods that are based on clustering. K-means clustering is a well-known method that can be used as IS by keeping its centers as selected prototypes. However, the number of selected prototypes should be prespecified. Luo et al. (2019) proposed an IS method based on threshold clustering. This type of clustering ensures clustering the data has a prespecified number of points in each cluster. Simulations and experiments on real data proved the efficiency of ITIS to reduce the massive dataset without consuming much time and memory, and while preserving the accuracy.

In this chapter, we validate the efficiency of ITIS by comparing it with IS methods based on clustering. The comparisons are obtained after clustering the prototypes by ITIS and well-known clustering methods. We also compared IHTC with k-means and IHTC with HAC with some hybrid clustering methods based on k-means and HAC. Several performance measurements are used for comparisons.

The results from simulation illustrate that clustering the prototypes that are created by ITIS reduce runtime and memory usage with retaining the clustering accuracy. Clustering by HAC records superior results in most cases compared to standard k-means. Comparing IHTC with HAC and IHTC with k-means with hybrid clustering methods demonstrates a significant difference favor ITIS.

Chapter 5

Conclusion

5.1 Introduction

Support vector machines (SVM) is a powerful supervised learning method for classification. However, training SVM would be computationally impossible, especially when data is massive. Instance selection (IS) methods for SVM have been developed to counteract this. In this dissertation, we propose the use of threshold clustering (TC) as IS to accelerate training SVM. TC is a recently-developed efficient clustering method that is designed to split data into many small clusters, making it ideal for IS. Given a threshold t^*, TC partitions data into t or more units while ensuring that the maximum within-cluster dissimilarity is small. The proposed method begins by applying TC on each class of the training dataset. Centroids of all clusters are computed to create the refined training set, then SVM is trained on this set. If data reduction is insufficient, TC may be applied on the centroids to obtain a new reduced training set. The entire algorithm is named support vector machines with threshold clustering (SVMTC). Under Gaussian radial bases kernel function, we prove that the maximum distance between kernel matrix, which is used in solving SVM, by using original data and kernel matrix that uses prototypes, is bounded when TC is used to extract these prototypes.

TC works effectively in big data, but it suffers from the curse of dimensions. For data with a large number of dimensions, TC might work slowly. We propose using feature reduction methods before applying TC. Two methods are considered to integrate feature reduction with SVMTC.

Iterative threshold instance selection (ITIS) is a recent extension of TC that is used in a massive data. ITIS begins by partitioning unlabeled data into clusters, then prototypes are computed of each cluster. If the reduction is not enough, TC is applied on the prototypes. Iterative hyperdized threshold clustering (IHTC) is a novel clustering method that is formed by using ITIS. In this dissertation, we use simulation to compare between ITIS and other IS methods in a massive dataset. Also, we compare between IHTC and some hybrid clustering methods. Results show that ITIS and IHTC outperform the competitive methods.

5.2 Future Work

Using threshold clustering as instance selection to accelerate SVM training shows it is effective via simulation and experiments in real datasets and big data. In addition, we also show the effectiveness of using the prototypes extracting from threshold clustering for SVM, theoretically. In general, we may apply our method in other classification methods such as random forest. However, it is insufficient to prove that experimentally. future work could show the effectiveness of our method for other classification methods such as random forest experimentally and theoretically.

Most feature selection methods take a long time, and few methods utilize clustering methods for feature selection. Future research could exploit the efficiency of threshold clustering to propose a new feature selection methods that could work faster than other feature selection method. Future work may also integrate TC for IS and FS tasks, simultaneously to accelerate SVM.

Some clustering methods such as k-means clustering has a kernel version that can cluster

data of any shape. We obtained the kernel version of TC in R by making some changes in the `distance` package. In the future, work can improve the kernel version of TC. Then, we utilize kernel TC as IS to accelerate training SVM.

Additional work can build an R package for SVMTC; and for SVMTC for datasets with a large number of features.

Bibliography

Doug Beaver, Sanjeev Kumar, Harry C Li, Jason Sobel, Peter Vajgel, et al. Finding a needle in haystack: Facebook's photo storage. In *OSDI*, volume 10, pages 1–8, 2010.

Usama M Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, et al. Knowledge discovery and data mining: Towards a unifying framework. In *KDD*, volume 96, pages 82–88, 1996.

Huan Liu and Hiroshi Motoda. On issues of instance selection. *Data Mining and Knowledge Discovery*, 6(2):115, 2002.

Reshma Khemchandani, Suresh Chandra, et al. Twin support vector machines for pattern classification. *IEEE Transactions on pattern analysis and machine intelligence*, 29(5): 905–910, 2007.

Johan AK Suykens and Joos Vandewalle. Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300, 1999.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3): 273–297, 1995.

Edgar Osuna, Robert Freund, and Federico Girosi. An improved training algorithm for support vector machines. In *Neural networks for signal processing VII. Proceedings of the 1997 IEEE signal processing society workshop*, pages 276–285. IEEE, 1997a.

John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. *Microsoft*, 1998.

Michael J Higgins, Fredrik Sävje, and Jasjeet S Sekhon. Improving massive experiments with

threshold blocking. *Proceedings of the National Academy of Sciences*, 113(27):7369–7376, 2016.

J Arturo Olvera-López, J Ariel Carrasco-Ochoa, J Francisco Martínez-Trinidad, and Josef Kittler. A review of instance selection methods. *Artificial Intelligence Review*, 34(2): 133–143, 2010a.

M Grochowski and N Jankowski. Comparison of instance selection algorithms i. algorithms survey, volume 3070 of lncs, 2004.

Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

Peter Hart. The condensed nearest neighbor rule (corresp.). *IEEE transactions on information theory*, 14(3):515–516, 1968.

Chien-Hsing Chou, Bo-Han Kuo, and Fu Chang. The generalized condensed nearest neighbor rule as a data reduction method. In *18th international conference on pattern recognition (ICPR'06)*, volume 2, pages 556–559. IEEE, 2006.

G Ritter, H Woodruff, S Lowry, and T Isenhour. An algorithm for a selective nearest neighbor decision rule (corresp.). *IEEE Transactions on Information Theory*, 21(6):665–669, 1975.

Dennis L Wilson. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics*, 3:408–421, 1972.

David W Aha, Dennis Kibler, and Marc K Albert. Instance-based learning algorithms. *Machine learning*, 6(1):37–66, 1991.

D Randall Wilson and Tony R Martinez. Reduction techniques for instance-based learning algorithms. *Machine learning*, 38(3):257–286, 2000.

Henry Brighton and Chris Mellish. Advances in instance selection for instance-based learning algorithms. *Data mining and knowledge discovery*, 6(2):153–172, 2002.

D Randall Wilson and Tony R Martinez. Instance pruning techniques. In *ICML*, volume 97, pages 400–411, 1997.

Barbara Spillmann, Michel Neuhaus, Horst Bunke, Elżbieta Pekalska, and Robert PW Duin. Transforming strings to vector spaces using prototype selection. In *Joint IAPR international workshops on statistical techniques in pattern recognition (SPR) and structural and syntactic pattern recognition (SSPR)*, pages 287–296. Springer, 2006.

Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

Paul S Bradley, Usama M Fayyad, Cory A Reina, Fayyad Reina Bradley, PS Bradley, Usama Fayyad, and Cory Reina. Scaling clustering algorithms to large databases", microsoft research report. *Microsoft*, 1998.

Alessandra Lumini and Loris Nanni. A clustering method for automatic biometric template selection. *Pattern Recognition*, 39(3):495–497, 2006.

J Arturo Olvera-López, J Ariel Carrasco-Ochoa, and J Francisco Martínez-Trinidad. A new fast prototype selection method based on clustering. *Pattern Analysis and Applications*, 13(2):131–141, 2010b.

William DuMouchel, Chris Volinsky, Theodore Johnson, Corinna Cortes, and Daryl Pregibon. Squashing flat files flatter. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 6–15, 1999.

Chaoyu Gong, Zhi-gang Su, Pei-hong Wang, Qian Wang, and Yang You. Evidential instance selection for k-nearest neighbor classification of big data. *International Journal of Approximate Reasoning*, 138:123–144, 2021.

Jerome H Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)*, 3(3):209–226, 1977.

Dorit S Hochbaum and David B Shmoys. A unified approach to approximation algorithms for bottleneck problems. *Journal of the ACM (JACM)*, 33(3):533–550, 1986.

Jianmei Luo, ChandraVyas Annakula, Aruna Sai Kannamareddy, Jasjeet S Sekhon, William Henry Hsu, and Michael Higgins. Hybridized threshold clustering for massive data. *arXiv preprint arXiv:1907.02907*, 2019.

V Vapnik. Statistical learning theory new york. *NY: Wiley*, 1998.

Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning*, pages 137–142. Springer, 1998.

Edgar Osuna, Robert Freund, and Federico Girosi. Training support vector machines: an application to face detection. In *Proceedings of IEEE computer society conference on computer vision and pattern recognition*, pages 130–136. IEEE, 1997b.

Sumit Bhatia, Praveen Prakash, and GN Pillai. Svm based decision support system for heart disease classification with integer-coded genetic algorithm to select critical features. In *Proceedings of the world congress on engineering and computer science*, pages 34–38, 2008.

Harold W Kuhn and Albert W Tucker. Nonlinear programming. In *Traces and emergence of nonlinear programming*, pages 247–258. Springer, 2014.

Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 1999.

Stefan Knerr, Léon Personnaz, and Gérard Dreyfus. Single-layer learning revisited: a stepwise procedure for building and training a neural network. In *Neurocomputing*, pages 41–50. Springer, 1990.

JC Platt. Advances in kernel methods: Support vector learning, ch. fast training of support vector machines using sequential minimal optimization, 1999.

Abdelouahid Lyhyaoui, Manel Martinez, Inma Mora, Maryan Vaquez, J-L Sancho, and Anibal R Figueiras-Vidal. Sample selection via clustering to construct support vector-like classifiers. *IEEE Transactions on neural networks*, 10(6):1474–1481, 1999.

M Barros De Almeida, Antônio de Pádua Braga, and João Pedro Braga. Svm-km: speeding svms learning with a priori cluster selection and k-means. In *Proceedings. Vol. 1. Sixth Brazilian Symposium on Neural Networks*, pages 162–167. IEEE, 2000.

Ravindra Koggalage and Saman Halgamuge. Reducing the number of training samples for fast support vector machine classification. *Neural Information Processing-Letters and Reviews*, 2(3):57–65, 2004.

Zheng Songfeng, Lu Xiaofeng, Zheng Nanning, and Xu Weipu. Unsupervised clustering based reduced support vector machines. In *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03).*, volume 2, pages II–821. IEEE, 2003.

Yuh-Jye Lee and Olvi L Mangasarian. Rsvm: Reduced support vector machines. In *Proceedings of the 2001 SIAM International Conference on Data Mining*, pages 1–17. SIAM, 2001.

Xiao-Li Li, Ji-Min Liu, and Zhong-Zhi Shi. A chinese web page classifier based on support vector machine and unsupervised clustering. *CHINESE JOURNAL OF COMPUTERS-CHINESE EDITION-*, 24(1):62–68, 2001.

Quang-Anh Tran, Qian-Li Zhang, and Xing Li. Reduce the number of support vectors by using clustering techniques. In *Proceedings of the 2003 International Conference on Machine Learning and Cybernetics (IEEE Cat. No. 03EX693)*, volume 2, pages 1245–1248. IEEE, 2003.

Wenjian Wang and Zongben Xu. A heuristic training for support vector regression. *Neurocomputing*, 61:259–275, 2004.

Dongwei Cao and Daniel Boley. On approximate solutions to support vector machines. In *Proceedings of the 2006 SIAM International Conference on Data Mining*, pages 534–538. SIAM, 2006.

Zhi-Qiang Zeng, Hua-Rong Xu, Yan-Qi Xie, and Ji Gao. A geometric approach to train svm on very large data sets. In *2008 3rd International Conference on Intelligent System and Knowledge Engineering*, volume 1, pages 991–996. IEEE, 2008.

Asdrúbal López Chau, Xiaoou Li, and Wen Yu. Convex and concave hulls for classification with support vector machine. *Neurocomputing*, 122:198–209, 2013.

Xiang-Jun Shen, Lei Mu, Zhen Li, Hao-Xiang Wu, Jian-Ping Gou, and Xin Chen. Large-scale support vector machine classification with redundant data reduction. *Neurocomputing*, 172:189–197, 2016.

Li Zhang, Ning Ye, Weida Zhou, and Licheng Jiao. Support vectors pre-extracting for support vector machine based on k nearest neighbour method. In *2008 International Conference on Information and Automation*, pages 1353–1358. IEEE, 2008.

Li Guo and Samia Boukir. Fast data selection for svm training using ensemble margin. *Pattern Recognition Letters*, 51:112–119, 2015.

Li Guo, Samia Boukir, and Nesrine Chehata. Support vectors selection for supervised

learning using an ensemble approach. In *2010 20th International Conference on Pattern Recognition*, pages 37–40. IEEE, 2010.

Li Guo and Samia Boukir. Margin-based ordered aggregation for ensemble pruning. *Pattern Recognition Letters*, 34(6):603–609, 2013.

Mohammad Aslani and Stefan Seipel. Efficient and decision boundary aware instance selection for support vector machines. *Information Sciences*, 577:579–598, 2021.

Marko Robnik-Šikonja and Igor Kononenko. Theoretical and empirical analysis of relieff and rrelieff. *Machine learning*, 53(1):23–69, 2003.

Xiaofei He, Deng Cai, and Partha Niyogi. Laplacian score for feature selection. *Advances in neural information processing systems*, 18, 2005.

Feiping Nie, Shiming Xiang, Yangqing Jia, Changshui Zhang, and Shuicheng Yan. Trace ratio criterion for feature selection. In *AAAI*, volume 2, pages 671–676, 2008.

Richard O Duda, Peter E Hart, and David G Stork. Pattern classification. 605 third avenue, 2012.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

John C Davis and Robert J Sampson. *Statistics and data analysis in geology*, volume 646. Wiley New York, 1986.

Huan Liu and Rudy Setiono. Chi2: Feature selection and discretization of numeric attributes. In *Proceedings of 7th IEEE international conference on tools with artificial intelligence*, pages 388–391. IEEE, 1995.

Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 2(11):559–572, 1901.

Ali Ghodsi. Dimensionality reduction a short tutorial. *Department of Statistics and Actuarial Science, Univ. of Waterloo, Ontario, Canada*, 37(38):2006, 2006.

Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In *International conference on artificial neural networks*, pages 583–588. Springer, 1997.

Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.

Sebastian Mika, Gunnar Ratsch, Jason Weston, Bernhard Scholkopf, and Klaus-Robert Mullers. Fisher discriminant analysis with kernels. In *Neural networks for signal processing IX: Proceedings of the 1999 IEEE signal processing society workshop (cat. no. 98th8468)*, pages 41–48. Ieee, 1999.

Manmohan Singh, Rajendra Pamula, et al. Email spam classification by support vector machine. In *2018 International Conference on Computing, Power and Communication Technologies (GUCON)*, pages 878–882. IEEE, 2018.

Hwanjo Yu, Jiong Yang, and Jiawei Han. Classifying large data sets using svms with hierarchical clusters. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 306–315, 2003.

Shigeo Abe and Takuya Inoue. Fast training of support vector machines by extracting boundary data. In *International Conference on Artificial Neural Networks*, pages 308–313. Springer, 2001.

Wan Zhang and Irwin King. Locating support vectors via/spl beta/-skeleton technique. In *Proceedings of the 9th International Conference on Neural Information Processing, 2002. ICONIP'02.*, volume 3, pages 1423–1427. IEEE, 2002.

Xiaowei Yang, Daying Lin, Zhifeng Hao, Yanchun Liang, Guirong Liu, and Xu Han. A fast svm training algorithm based on the set segmentation and k-means clustering. *Progress in Natural Science*, 13(10):750–755, 2003.

Daniel Boley. Principal direction divisive partitioning. *Data mining and knowledge discovery*, 2(4):325–344, 1998.

Der-Chiang Li and Yao-Hwei Fang. An algorithm to cluster data for efficient classification of support vector machines. *Expert Systems with Applications*, 34(3):2013–2018, 2008.

Christopher J Merz. Uci repository of machine learning databases. *http://www. ics. uci. edu/˜ mlearn/MLRepository. html*, 1998.

David Meyer, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel, Friedrich Leisch, Chih-Chung Chang (libsvm C++-code), and Chih-Chen Lin (libsvm C++-code). *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*, 2020. URL https://CRAN.R-project.org/package=e1071. R package version 1.7-9.

Fredrik Savje, Michael Higgins, and Jasjeet Sekhon. *scclust: Size-Constrained Clustering*, 2018. URL https://CRAN.R-project.org/package=scclust. R package version 0.2.2.

Jack Baker, Christopher Nemeth, Fearnhead, and Emily B. Fox. *sgmcmc: Stochastic Gradient Markov Chain Monte Carlo*, 2019. URL https://rdrr.io/cran/sgmcmc/man/getDataset.html.

Jock A Blackard and Denis J Dean. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and electronics in agriculture*, 24(3):131–151, 1999.

I-Cheng Yeh and Che-hui Lien. The comparisons of data mining techniques for the predictive

accuracy of probability of default of credit card clients. *Expert systems with applications*, 36(2):2473–2480, 2009.

RJ Lyon, BW Stappers, S Cooper, JD Brooke, and JM Knowles. Fifty years of pulsar candidate selection: From simple filters to a new principled real-time classification approach. *MNRAS*, pages 000–000, 2015.

RK Bock, A Chilingarian, M Gaug, F Hakl, Th Hengstebeck, M Jiřina, J Klaschka, E Kotrč, P Savickỳ, S Towers, et al. Methods for multidimensional event classification: a case study using images from a cherenkov gamma-ray telescope. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 516(2-3):511–528, 2004.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Donald Michie, David J Spiegelhalter, and Charles C Taylor. *Machine learning, neural and statistical classification*. Citeseer, 1994.

Rajen Bhatt and Abhinav Dhall. Skin segmentation dataset. *UCI Machine Learning Repository*, 2010.

M Suganthi and V Karunakaran. Instance selection and feature extraction using cuttlefish optimization algorithm and principal component analysis using decision tree. *Cluster Computing*, 22(1):89–101, 2019.

Milad Malekipirbazari, Vural Aksakalli, Waleed Shafqat, and Andrew Eberhard. Performance comparison of feature selection and extraction methods with random instance selection. *Expert Systems with Applications*, 179:115072, 2021.

Dimitris Fragoudis, Dimitris Meretakis, and Spiros Likothanassis. Integrating feature and

instance selection for text classification. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 501–506, 2002.

Jerffeson Teixeira De Souza, Rafael Augusto Ferreira Do Carmo, and Gustavo Augusto Lima De Campos. A novel approach for integrating feature and instance selection. In *2008 International Conference on Machine Learning and Cybernetics*, volume 1, pages 374–379. IEEE, 2008.

Brian Ripley, Bill Venables, Douglas M. Bates, Kurt Hornik, Albrecht Gebhardt, and David Firth. *MASS: Support Functions and Datasets for Venables and Ripley's MASS*, 2022. URL https://CRAN.R-project.org/package=MASS. R package version 7.3-58.1.

Jerome Friedman, Trevor Hastie, Balasubramanian Narasimhan Rob Tibshirani, Kenneth Tay, Noah Simon, Junyang Qian, and James Yang. *glmnet: Lasso and Elastic-Net Regularized Generalized Linear Models*, 2022. URL https://CRAN.R-project.org/package=glmnet. R package version 4.1-4.

Kisung You. *Rdimtools: Dimension Reduction and Estimation Methods*, 2018. URL https://CRAN.R-project.org/package=Rdimtools. R package version 0.3.1.

Pierre Baldi, Peter Sadowski, and Daniel Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, 5(1):1–9, 2014.

Douglas H Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine learning*, 2(2):139–172, 1987.

Saeid Amiri, Bertrand S Clarke, Jennifer L Clarke, and Hoyt Koepke. A general hybrid clustering technique. *Journal of Computational and Graphical Statistics*, 28(3):540–551, 2019.

Hugh Chipman and Robert Tibshirani. Hybrid hierarchical clustering with applications to microarray data. *Biostatistics*, 7(2):286–301, 2006.

J MacQueen. Classification and analysis of multivariate observations. In *5th Berkeley Symp. Math. Statist. Probability*, pages 281–297, 1967.

David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. Technical report, Stanford, 2006.

James C Bezdek. Objective function clustering. In *Pattern recognition with fuzzy objective function algorithms*, pages 43–93. Springer, 1981.

Disa Mhembere. *clusternor: A Parallel Clustering Non-Uniform Memory Access ('NUMA') Optimized*, 2020. URL https://CRAN.R-project.org/package=clusternor. R package version 0.0-4.

Amiri Saeid, Clarke Bertrand S, and Clarke Jennifer L. *saeidamiri1/GHC: General Hybrid Clustering Technique*, 2019. URL https://github.com/saeidamiri1/GHC/wiki.

Hugh Chipman and Rob Tibshirani. *hybridHclust: Hybrid Hierarchical Clustering*, 2015. URL https://github.com/cran/hybridHclust.

Bernard Desgraupes. *clusterCrit: Clustering Indices*, 2018. URL https://CRAN.R-project.org/package=clusterCrit. R package version 1.2.8.

Kansas State University. *Acknowledging use of Beocat resources and/or personnel in publications*, 2018. URL https://support.beocat.ksu.edu/BeocatDocs/index.php/Policy.

Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

Joe H Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, 58(301):236–244, 1963.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996.

Appendix A

Proof of Lemma 2

A.1 Lemma 3

Lemma 3. *If $\hat{\mathbf{x}}$ is a prototype of a convex hull of a cluster c, and $i, j \in c$, then*

$$\|\mathbf{x}_i - \hat{\mathbf{x}}\| \leq \max_{ij \in c} \|\mathbf{x}_i - \mathbf{x}_j\|$$

where convex hull $H(c)$ of cluster c is defined as the set of all convex combinations of finite number of datapoints in c:

$$\mathbf{x} \in H(c) \quad \textit{if and only if there exists } \beta_j \textit{ such that } \quad \mathbf{x} = \sum_{j \in c} \beta_j \mathbf{x}_j \quad \textit{and} \quad \sum_{j \in c} \beta_j = 1.$$

A.1.1 Proof of Lemma 3

Suppose that $i \in c$ and define $\lambda_i = \max_{ij \in c} \|\mathbf{x}_i - \mathbf{x}_j\|$. By definition, we can write $\hat{\mathbf{x}}_i$ as convex linear combination of datapoints $j \in c$. Thus,

$$\|\mathbf{x}_i - \hat{\mathbf{x}}\| = \left\| \mathbf{x}_i - \sum_{l \in c} \beta_l \mathbf{x}_l \right\| = \left\| \sum_{l \in c} \beta_l \left(\mathbf{x}_i - \mathbf{x}_l \right) \right\| \leq \sum_{l \in c} \beta_l \|\mathbf{x}_i - \mathbf{x}_l\| \leq \lambda_i \left(\sum_{l \in c} \beta_l \right) = \lambda_i.$$

A.2 Proof of Lemma 2

Proof. Let $\epsilon > 0$, σ^2 is the parameter of Gaussian radial basis function. Since threshold clustering guarantees producing clustering such that the maximum within-cluster weight is no larger than 4λ, where λ is an optimal value.

$$\max_{ij \in c} \|\mathbf{x}_i - \mathbf{x}_j\| \leq 4\lambda$$

Since $\lambda = \frac{\sqrt{2}}{4}\sigma \left(-\log\left(1 - \frac{\epsilon^2}{2}\right)\right)^{\frac{1}{2}}$, and $\log\left(1 - \frac{\epsilon^2}{2}\right) < 0$ for $0 < \epsilon < 1$, hence we have

$$\max_{ij \in c} \|\mathbf{x}_i - \mathbf{x}_j\| \leq 4\left[\frac{\sqrt{2}}{4}\sigma \left(-\log\left(1 - \frac{\epsilon^2}{2}\right)\right)^{\frac{1}{2}}\right]$$

$$\max_{ij \in c} \|\mathbf{x}_i - \mathbf{x}_j\| \leq \sqrt{2}\sigma \left(-\log\left(1 - \frac{\epsilon^2}{2}\right)\right)^{\frac{1}{2}}$$

$$\max_{ij \in c} \|\mathbf{x}_i - \mathbf{x}_j\|^2 \leq -2\sigma^2 \log\left(1 - \frac{\epsilon^2}{2}\right)$$

Using Lemma 3, we have $\max_i \|\mathbf{x}_i - \hat{\mathbf{x}}_i\| \leq \max_{ij \in c} \|\mathbf{x}_i - \mathbf{x}_j\|$, where $\hat{\mathbf{x}}_i$ is the prototype of

the cluster that $\mathbf{x}_i$ belongs to. Hence,

$$\begin{aligned}
\frac{-1}{2\sigma^2}\max_i\|\mathbf{x}_i-\hat{\mathbf{x}}_i\|^2 &\geq \log\left(1-\frac{\epsilon^2}{2}\right)\\
\exp\left(\frac{-1}{2\sigma^2}\max_i\|\mathbf{x}_i-\hat{\mathbf{x}}_i\|^2\right) &\geq \left(1-\frac{\epsilon^2}{2}\right)\\
-\exp\left(\frac{-1}{2\sigma^2}\max_i\|\mathbf{x}_i-\hat{\mathbf{x}}_i\|^2\right) &\leq \left(\frac{\epsilon^2}{2}-1\right)\\
2-2\exp\left(\frac{-1}{2\sigma^2}\max_i\|\mathbf{x}_i-\hat{\mathbf{x}}_i\|^2\right) &\leq \epsilon^2\\
\sqrt{2-2\exp\left(\frac{-1}{2\sigma^2}\max_i\|\mathbf{x}_i-\hat{\mathbf{x}}_i\|^2\right)} &\leq \epsilon\\
\max_i\sqrt{2-2\exp\left(\frac{-1}{2\sigma^2}\|\mathbf{x}_i-\hat{\mathbf{x}}_i\|^2\right)} &\leq \epsilon\\
\max_i\sqrt{2-2\mathbf{K}(\mathbf{x}_i,\hat{\mathbf{x}}_i)} &\leq \epsilon
\end{aligned}$$

Under Gaussian radial basis kernel function, we can write

$$\begin{aligned}
\max_i\sqrt{\mathbf{K}(\mathbf{x}_i,\mathbf{x}_i)+\mathbf{K}(\hat{\mathbf{x}}_i,\hat{\mathbf{x}}_i)-2\mathbf{K}(\mathbf{x}_i,\hat{\mathbf{x}}_i)} &\leq \epsilon\\
\max_i\sqrt{\phi(\mathbf{x}_i)\cdot\phi(\mathbf{x}_i)+\phi(\hat{\mathbf{x}}_i)\cdot\phi(\hat{\mathbf{x}}_i)-2\phi(\mathbf{x}_i)\cdot\phi(\hat{\mathbf{x}}_i)} &\leq \epsilon\\
\max_i\sqrt{(\phi(\mathbf{x}_i)-\phi(\hat{\mathbf{x}}_i))\cdot(\phi(\mathbf{x}_i)-\phi(\hat{\mathbf{x}}_i))} &\leq \epsilon\\
\max_i\|\phi(\mathbf{x}_i)-\phi(\hat{\mathbf{x}}_i)\| &\leq \epsilon
\end{aligned}$$

□

Appendix B

SVMTC Performance

B.1 SVMTC Performance with Varying Threshold Size of TC

By varying threshold size t^* of threshold clustering, we compare the performance of SVMTC for different datasets. We use $t^* = 2, 3, 4, 5, 6, 7, 8, 9$, then we evaluate SVMTC by using runtime and memory usage of pre-processing, runtime and memory usage of training SVM on the reduced dataset, reduction rate, number of support vectors used and accuracy of SVM. Figure B.1 shows runtime and memory usage for SVMTC divided into pre-processing and SVM training. The other evaluation measurements are presented in Figure B.2.

We conclude that increasing t^* improves runtime and memory usage for SVM training while we could use more time and memory for pre-processing. In addition, reduction rate is increased and number of support vectors reduces when t^* increases whereas SVM accuracy is slightly reduced for some datasets.

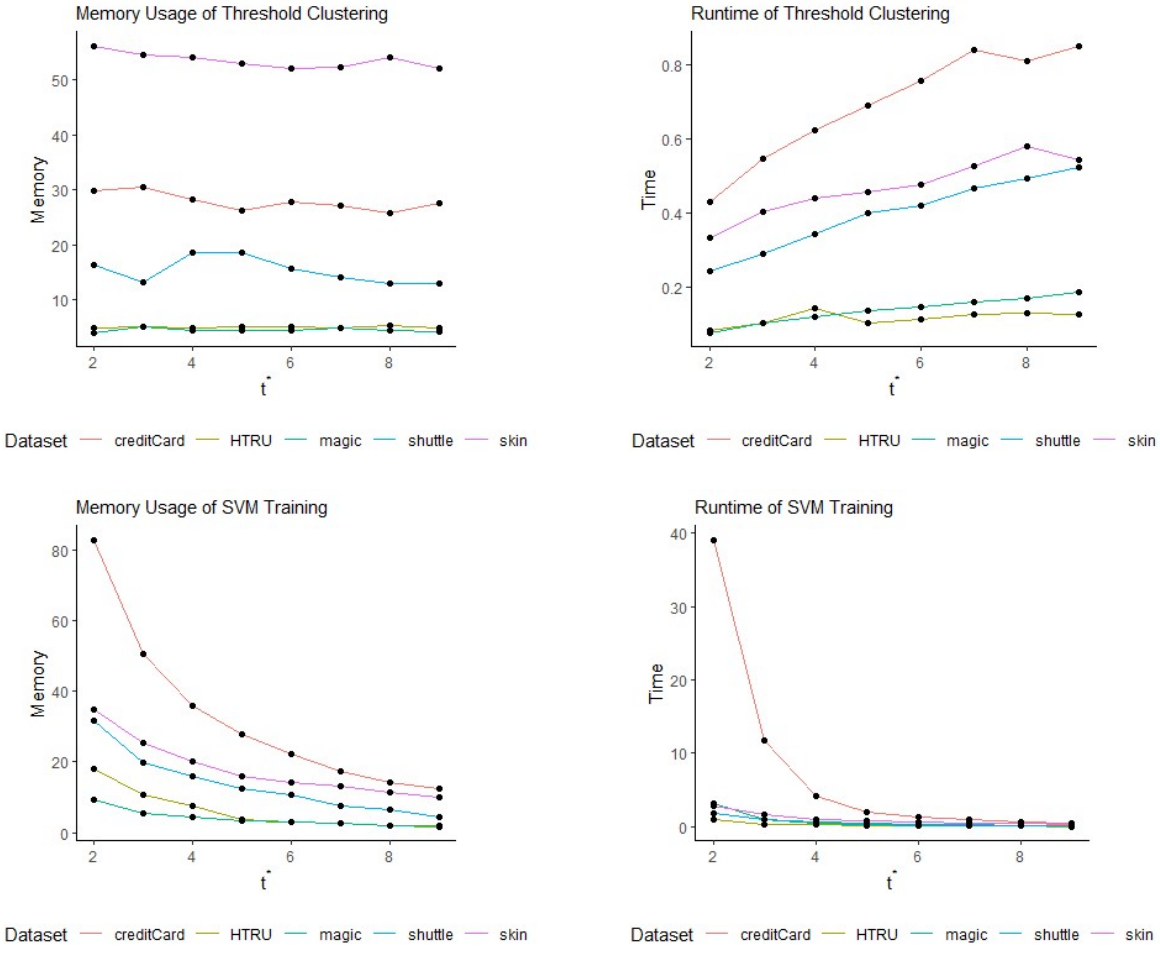

Figure B.1: *Runtime and memory usage of reduction and SVM training for different datasets with changing of* t^*.

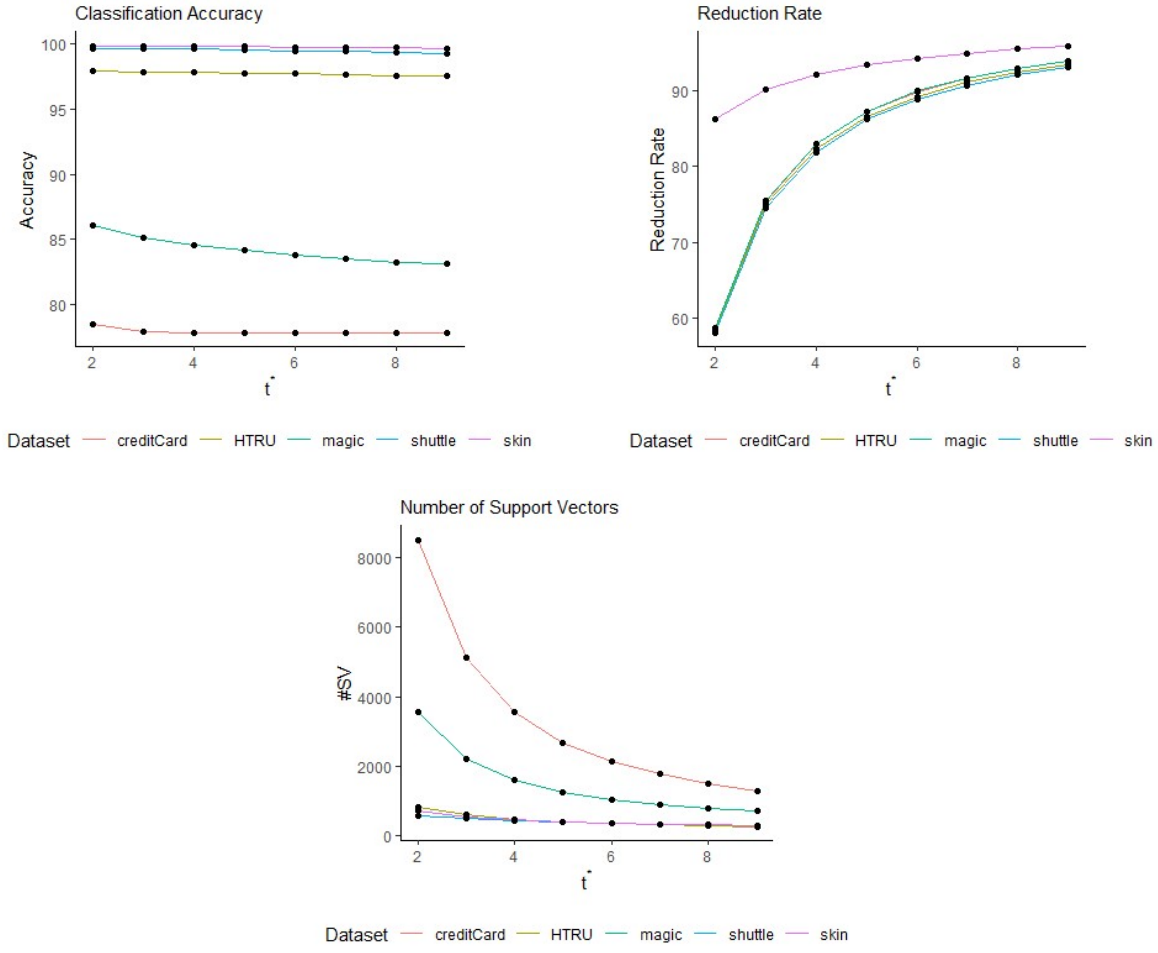

Figure B.2: *Classification accuracy, reduction rate of SVMTC and number of suport vectors used for different datasets with changing of t^*.*

B.2 SVMTC Performance with Varying Number of Iterations

By using threshold size $t^* = 2$ of threshold clustering and by varying the number of iterations r, we compare the performance of SVMTC for a simulated data of size 10^7 with two features. We use $m = 3, 4, 5, 6, 7, 8$. Then we evaluate SVMTC by using runtime and memory usage of pre-processing, runtime and memory usage of training SVM on the reduced dataset, reduction rate, number of support vectors used and accuracy of SVM. Figure B.3 shows runtime, memory for SVMTC divided into pre-processing and SVM training, reduction rate and number of support vectors. The classification accuracy is presented in Figure B.4.

We conclude that increasing r improves runtime for SVM training. We may use large amount of memory for preprocessing; however, less memory is used for SVM training. In addition, reduction rate is increased and number of support vectors reduces when r increases, whereas SVM accuracy is fluctuated.

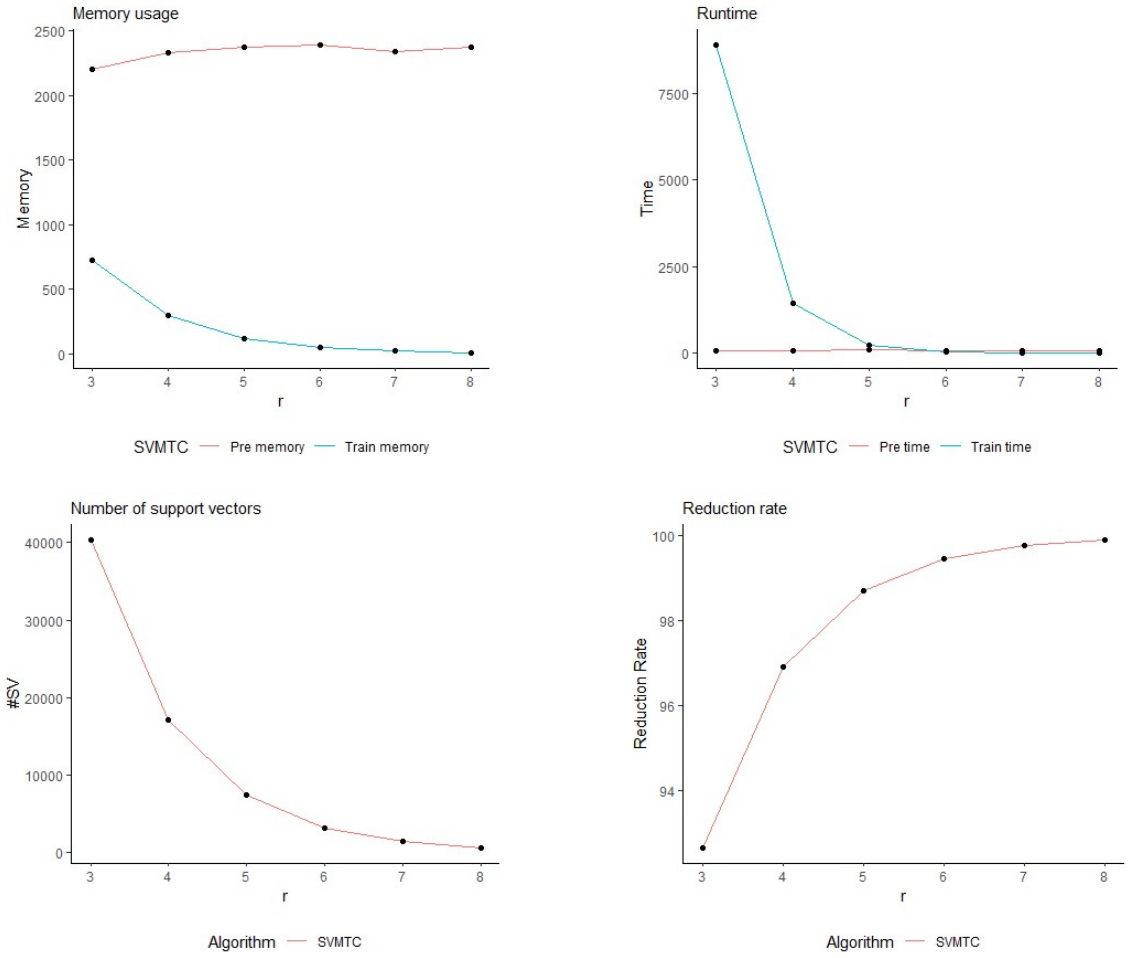

Figure B.3: *Runtime, memory usage, reduction rate and number of support vectors of SVMTC with* $t^* = 2$ *and with changing of* r *for simulated data of size* 10^7 *and* 2 *features.*

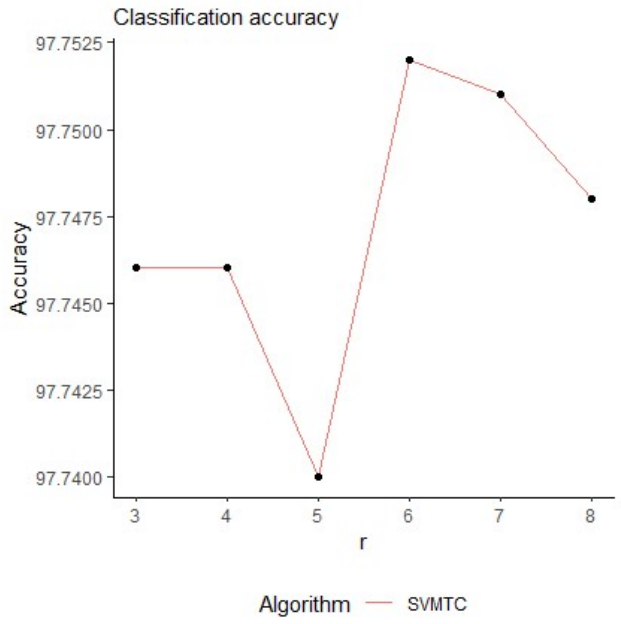

Figure B.4: *Classification accuracy of SVMTC with* $t^* = 2$ *and with changing of* r *for simulated data of size* 10^7 *and* 2 *features.*

Appendix C

R Functions

In this section, functions that we coded in R are showed. The functions that implement SVMTC in small and big data are given included the supplementary functions.

C.1 R Function of SVMTC

This function is used to obtain the reduced dataset, the first two steps of SVMTC. For small datasets, this function can be implemented with one repetition $r = 1$.

```
# tr is the training set.
# cl is the column number of class label.
# t is the parameter of threshold clustering.
# nc is the number of categories of class label.
# The output is the reduced dataset.
SVMTC_1_2 <- function(tr,cl,t,nc){
  r <- by(tr,tr[,cl],function(x){sc_clustering(distances(x),t)})
  ns <- matrix(0,byrow <- FALSE, ncol <- cl)
  for(i in 1:nc){
```

```
  ns <- rbind(ns, fastAgg(as.matrix(tr[tr[,cl]==i,]), r[[i]]))}
  assign("TrainData1",data.frame(ns),envir = globalenv())}
```

C.2 R Function of SVMTC for Big Datasets

For a big dataset, this function can be used to obtain the reduced dataset with r repetitions.

```
# tr is the training set.
# cl is the column number of class label.
# t is the parameter of threshold clustering.
# nc is the number of categories of class label.
# r is the number of repetition.
# The output is the reduced dataset.
SVMTC2_1_2 <- function(tr,cl,t,nc,r){
  fun <- function(tr){Itis::Itis_fct_nomid(tr,t,r,cl-1,dim(tr)[1])}
  r <- by(tr,tr[,cl],fun)
  b <- matrix(0,ncol=cl)
  for(i in 1:nc) {b <- rbind(b,r[[i]][[1]])}
  assign("TrainData1", b[-1,],envir = globalenv())}
```

C.3 Supplementary Functions

C.3.1 Function 1

The following function is fastAgg function that is used from C++ in our R codes. We use this function to compute the centroids of clusters fast.

```
NumericMatrix fastAgg(NumericMatrix orgMeans, IntegerVector cats) {
```

```
//orgMeans is the original dataset
//cats is the categories, numbered 0 to n-1
//Store dimensons of the for loop
long catLeng = cats.length();
int numCols = orgMeans.ncol();
//initialize number of categories,
//initialize number of observations in each category,
//initialize aggregated means
long numCats = max(cats);
IntegerVector catSize(numCats+1);
NumericMatrix aggMeans(numCats+1,numCols);
 for(int j = 0; j < numCols; j++){
 for(long i = 0; i < catLeng; i++ ){
  //Different instructions if j = 0 and if j notequal zero
  if(j == 0){
  //Increase the category total
   catSize[cats[i]]++;
  //Update the means
   aggMeans(cats[i],j)
   = (double)(catSize[cats[i]]-1)/catSize[cats[i]]*aggMeans(cats[i],j)
   +(double)1/catSize[cats[i]]*orgMeans(i,j);
            }
  else{
  aggMeans(cats[i],j) = (double)aggMeans(cats[i],j)
  +(double)1/catSize[cats[i]]*orgMeans(i,j);
     }
```

```
  }
 } return aggMeans;
}
```

C.3.2 Function 2

We use this function to iterate threshold clustering $m > 0$ times.

```
# Itis_fct_nomid
# iterated threshold instance selection
# dat is dataset, it should be a n*d matrix
# t is threshold size, it should be pre-specify
# m is number of iteration for threshold clustering (m > 0)
# d is dimension of dataset (eg: d = 2)
# n is datasize
# The output is a list of size 2.
# The first list is proto center for last step,
# The second list includes cluster label for last step
Itis_fct_nomid <- function(dat, t, m, d, n){
  clusterlabel <- rep(NA, n)
  for(i in 1:m){
    my_dist <- distances(dat)
    my_clustering_new <- sc_clustering(my_dist, t)
    aggdata_old <- fastAgg(as.matrix(dat), my_clustering_new)
    if(i == 1){
      clusterlabel <- as.integer(my_clustering_new)
    }
```

```
    else{
      clusterlabel <- fastJoin2(my_clustering_old, my_clustering_new)
    }
    dat <- aggdata_old
    my_clustering_old <- clusterlabel
  }
  return(list(dat, clusterlabel))
}
```