B.Comp. Dissertatition

Classification using the Semi-lazy Mining Paradigm

By

Tang Yixuan

Department of Computer Science

School of Computing

National University of Singapore

2015/2016

B.Comp. Dissertatition

Classification using the Semi-lazy Mining Paradigm

By

Tang Yixuan

Department of Computer Science

School of Computing

National University of Singapore

2015/2016

Project No: H015600
Advisor: Prof Anthony K. H. TUNG
Deliverables:
Report: 1 Volume

Abstract

Building classifiers on high dimensional data is always challenging, especially when the dataset is large. The training process is time consuming while the accuracy is not satisfactory. To overcome such problems, feature selection is often adopted to reduce dimensionality before constructing the classifiers.

In this project, we propose and implement two novel dynamic feature selection algorithms for semi-lazy learning paradigm(LAMP). Instead of pre-selecting the features, we delay the feature selection process until the prediction query arrives. Firstly, kNNs of the prediction query are retrieved using generic inverted index on the GPU(GENIE). Then, wrapper-based feature selection is applied by constructing a large number of small SVM classifiers. And lastly, the query is classified based on selected features.

We conducted emprical evaluations on several high-dimensional datasets. The evaluation results show that our approach is able to achieve high classification accuracy while significantly reducing the number of features required for classification.

Subject Descriptors:
Machine Learning
Learning Paradigms
Instance-based Learning
Dimensionality Reduction

Keywords:
Machine learning, classification, feature selection, GPU processing

Implementation Software and Hardware:
Ubuntu Linux , CUDA 7.0, C++11, GPU Nvidia GTX Titan X

Acknowledgement

I would like to express my sincere gratitude to my supervisor, Associate Professor Anthony K. H. TUNG, for his guidance and support throughout this project. I would also like to thank Zhou Jingbo, Luan Wenhao, Yang Yueji for initial code of GENIE and Zhulei for initial code of GPU-Precompute-SVM. Furthermore, I would like to extend my appreciation to everyone in GENIE and LAMP team for their patient assistance and valuable advise.

List of Figures

3.1 System architecture 6
3.2 Example data distribution for lazy learning 7
3.3 Key steps of feature selection algorithm 1 10
3.4 Key steps of feature selection algorithm 2 14

4.1 Relation between accuracy and number of top features selected (algorithm 1) . . 17
4.2 Change in accuracy when disgarding one feature from data 17
4.3 Relation between accuracy and number of top features selected (algorithm 2) . . 18
4.4 Relation between accuracy and value of N (algorithm 2) 19

A.1 Relations between max number of training set, training time and number of points per set using GPU-Precompute-SVM A-1

List of Tables

4.1 Classification accuracy on 3 real-life datasets 20

Table of Contents

Title i

Abstract ii

Acknowledgement iii

List of Figures iv

List of Tables v

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Project Objective . . . 2
- 1.3 Report Organization . . . 2

2 Related Work 3
- 2.1 Filter Methods for Feature Selection . . . 3
- 2.2 Wrapper Methods for Feature Selection . . . 4

3 Methodology 6
- 3.1 System Overview . . . 6
- 3.2 kNN and GENIE . . . 7
- 3.3 GPU-Precompute-SVM . . . 8
- 3.4 Dynamic Feature Selection . . . 9
  - 3.4.1 Algorithm 1 . . . 10
  - 3.4.2 Algorithm 2 . . . 12
- 3.5 Classification . . . 14

4 Evaluation 15
- 4.1 Experimental Setup . . . 15
- 4.2 Datasets for Evaluation . . . 15
- 4.3 Experiments for Key Parameters . . . 16
  - 4.3.1 Experiments for Parameter $reduce - ratio$ . . . 16
  - 4.3.2 Experiments for Parameter N . . . 18
- 4.4 Classification Results and Discussion . . . 19

5 Conclusion 21
- 5.1 Contributions . . . 21
- 5.2 Future Work . . . 21

References **23**

A Performances for GPU-Precompute-SVM **A-1**

Chapter 1

Introduction

1.1 Motivation

Building classifiers on high dimensional data is always challenging, in terms of both accuracy and efficiency. In order to overcome such problems, feature selection is a common processing step adopted before constructing the classifiers. Feature selection refers to the problem of selecting features that are most discrimitive of a given outcome. By eliminating irrelavent features from data, feature selection can help learning algorithms to overcome the curse of the dimensionality and to make classification much faster. With appropriate features selected, classification could even achieve higher accuracy since some noises in the original data are removed.

So far, there have been lots of research efforts devoted to feature selection algorithms for high dimensional data. These unsupervised feature selection algorithms are usually the first step of data processing for classification. They run statistical analysis on the training data and return a smaller feature space. However, when applied on large scale dataset, the feature selection algorithm itself is time consuming. Also, these algorithms select the same important features for the whole dataset, while ignoring the fact that important features may vary from query to query.

1.2 Project Objective

In this project, we propose and implement a semi-lazy classification paradigm for high dimensional data. Instead of applying feature selection on the whole training dataset, our system firstly finds k nearest neighbors (kNN) for each query, and then it applys a novel dynamic feature selection algorithm for the query based on its kNNs. Our dynamic feature selection algorithm aims to find a new feature space of current query with significantly less number of features but contains most useful information of the original feature space. And lastly, our system builds a more accurate and efficient classifier based on selected features.

1.3 Report Organization

The remainder of this dissertation is organized as follows. Chapter 2 briefly introduces knowledge related to feature selection algorithms for high dimensional data. Chapter 3 firstly presents the overall system architecture and then explains each step in detail. Chapter 4 prescribes the experimental settings and presents evaluations of our system with three high dimensional datasets. Lastly, Chapter 5 concludes this dissertation with contributions of our work and some potential directions for future research.

Chapter 2

Related Work

In this section, we briefly discuss relevant knowledge in the area of feature selection algorithms. Feature selection methods are usually divided into two types: filter methods and wrapper methods (Das, 2001). In filter methods, feature selection procedure is independent of the learning algorithm and it usually works as a pre-processing step on training data. While in wrapper methods, features are selected using estimated accuracy from the learning algorithm as the measure of importance (Kohavi & John, 1997). We introduce some famous algorithms for both filter and wrapper methods below.

2.1 Filter Methods for Feature Selection

Filter methods select a feature subset for any learning algorithms, e.g (Hall, 1999). Many of them are based on statistical distribution of training data. We introduce two most famous ones in this section, namely principal component analysis (PCA) and linear discriminant analysis (LDA).

PCA is a statistical procedure that converts a set of correlated features to linearly uncorrelated features via an orthogonal transformation (Kambhatla & Leen, 1997). Basicly, it finds a subspace of features that maximizes the variance. By applying PCA to feature selection, it can remove correlated features while retaining as much of the variance in the dataset as possible.

LDA is a generalization of Fishers linear discriminant method. It finds a subspace of features

that maximizes the inter-class variance and minimizes the intra-class variance (McLachlan, 1992). By doing so, LDA is able to find a linear combination of features that characterizes each class.

One disadvantage of filter methods is that it is possible for the criterion used by filters to result in a feature subspace that doesn't work well for the learning algorithm. Moreover, filter methods perform on the whole training dataset and the combination of features selected is applied to all coming queries. But in some dataset, different class of data may have different discriminant features. In this case, a general feature selection algorithm is no longer suitable. Besides, another limitation of filter methods is that they can hardly guarantee the number of features they can reduce.

2.2 Wrapper Methods for Feature Selection

Wrapper methods search through the space of feature subsets using a learning algorithm to inform search, e.g (John, Kohavi, & Pfleger, 1994). They calculate the estimated accuracy of the learning algorithm for each feature that can be added to or removed from the feature subset. Accuracy is estimated using cross validation on the training set. In forward selection, a wrapper estimates the accuracy of adding each unselected feature to the feature subset and chooses the best feature to add according to this criterion. This type of methods typically terminates when the estimated accuracy of adding any feature is less than the estimated accuracy of the feature set already selected (Das, 2001).

Genetic algorithms, population-based learning, and related Bayesian methods have been commonly used as search engines for wrappers (Inza, Merino, Larraaga, Quiroga, Sierra, & Girala, 1999; Kudo & all, 2000). Particularly for SVMs, one wrapper method, VS-SSVM, attracts our interests (Bi, Breneman, & Song, 2003). The basic idea of this algorithm is to repeatedly train linear SVM models on some validation dataset and to discard certain number of least significant features by interpreting SVM models trained each round. They believe the distribution of the linear model weights provides a mechanism for ranking and interpreting the importance of features. The feature selection algorithm proposed in our project is similar to this one. But

instead of interpreting the weights of features in training model, we rank the importance of features by SVM accuracy results on validation data directly.

Chapter 3

Methodology

In this chapter, we introduce the novel algorithm we proposed and all the decisions we made in the process of implementation. In section 3.1, we first present the overview of our program. Then in section 3.2 to section 3.5, we explain each step of the program in detail.

3.1 System Overview

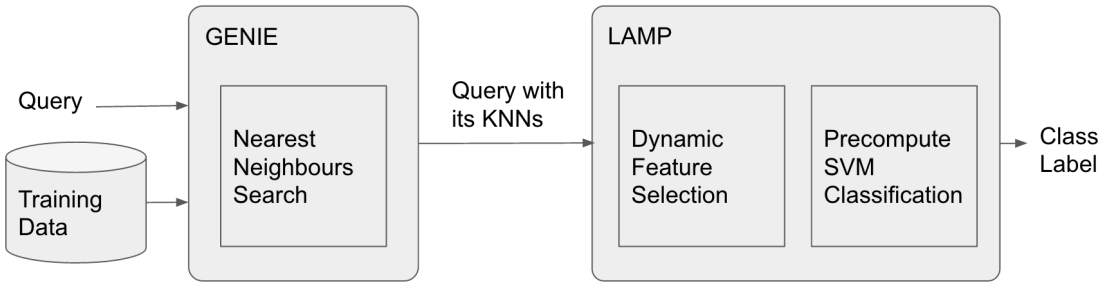


Figure 3.1: System architecture

Figure 3.1 shows the overview of our semi-lazy classification paradigm. This system accepts training and testing data as input, and it classifies each query in three steps.

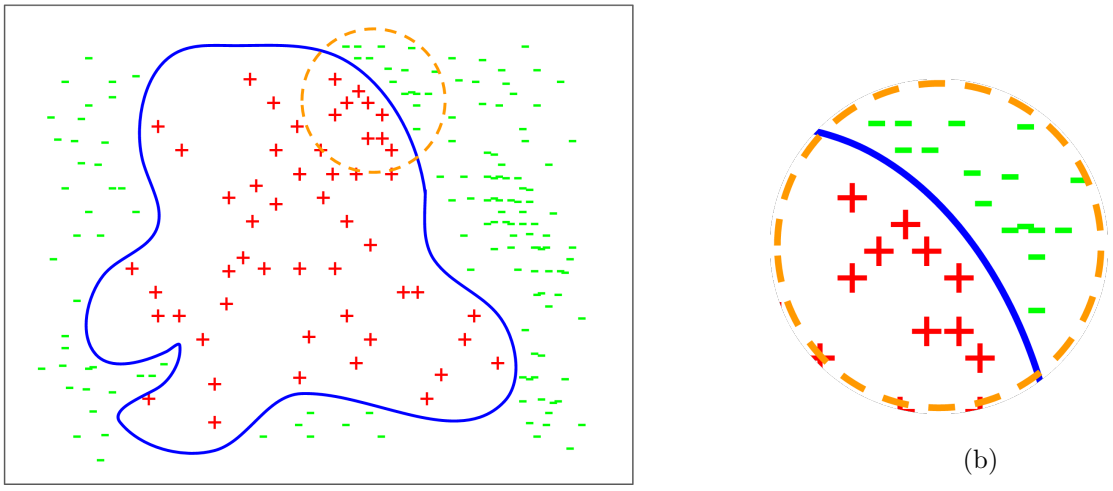
- GENIE picks kNNs for the query in training data.
- LAMP runs dynamic feature selection algorithm for the query based on its kNNs.

- LAMP trains a SVM classifier on kNNs' selected features and classifies the query.

3.2 kNN and GENIE

In contrast to eager learning methods that construct a general, explicit description of the target function on the whole training dataset, lazy learning methods store the training examples and postpone the training process until a new query must be classified. Each time a new query comes, its relationship with training data is evaluated and a local classifier function for the query can be built only on those points close to the query. A key advantage of lazy learning is that instead of estimating the target function on the entire instance space, it can estimate it locally and differently for each query (Mitchell, 1997).

There are two reasons that we choose to adopt lazy learning for large scale high dimensional data classification tasks. First of all, target function for high dimensional data on the whole training space can be very complex. By limiting the training examples to a smaller scale, the target function can be much simpler. Figure 3.2a demonstrates such example for 2D points. Although the class boundary is complex in whole space, it's simple in the circle as shown in Figure 3.2b. The other reason to support lazy learning is that with less training data, the computational complexity can be largely reduced by pruning training data.



(a) (b)

Figure 3.2: Example data distribution for lazy learning

KNN is one of the most basic lazy learning methods. This algorithm assumes each data corresponds to a point in the n dimensional space (Mitchell, 1997). Given an instance x, its kNNs are found in terms of the standard Euclidean distance. More precisely, let an arbitrary instance x be described by the feature vector

$$(a_1(x), a_2(x), ...a_n(x))$$

where $a_r(x)$ denotes the value of the r_{th} attribute of instance x. Then the distance between two instances x_i and x_j is defined to be $d(x_i, x_j)$, where

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^{n}(a_r(x_i) - a_r(x_j))^2}$$

Then kNNs of instance x refers to the k points in the training data with smallest $d(x, x_i)$. In this project, picking kNNs program is the first step to prune the training data for each query and it's achieved by GENIE (Zhou, Guo, Jagadish, Luan, Tung, & Zheng, 2016).

GENIE refers to generic inverted index on the GPU. It's a program that can effectively support approximate nearest neighbor search for high dimensional data through exerting Locality Sensitive Hashing (LSH) schemes. Moreover, GENIE is implemented on the Graphics Processing Units (GPU) to parallelize the computations and thus to speed up the whole process.

3.3 GPU-Precompute-SVM

Before proceeding to the dynamic feature selection, we need to introduce another program, GPU-Precompute-SVM, which is used as a tool for feature selection.

Support Vector Machines (SVMs) have been recognized as one of the most successful classification methods for many applications(Mitchell, 1997). The basic idea of SVM learning is to find a hyperplane which separates the d-dimensional data into two classes; and the hyperplane should be as far away from the data of both classes as possible. Let $x_1, x_2, , x_n$ be our dataset and let y_i be the class label of x_i. The decision boundary is described by function

$$w^T x + b = 0$$

Such decision boundary can be found by solving the following constrained optimization problem:

$$\text{minimize } \frac{1}{2}||w||^2$$

$$\text{subject to } y_i(w^T x_i + b) \leq 1 \text{ for all } i$$

However, the classification method described above requires data to be linearly separable. As for non-linearly separable dataset, SVMs will first use kernel function to cast the data into a higher dimension space where it is separable and then solve the classification there. GPU-Precompute-SVM is an implementation of SVM used in this project. It exploits the computation power of GPU to speed up the training process of a public library LIBSVM. And it modifies the library to allow concurrent training of large number of SVMs. The speed of GPU-Precompute-SVM satisfies the prerequisite of all dynamic feature selection algorithm, and its performance is shown the Appendix A.

3.4 Dynamic Feature Selection

Having a query and its nearest neighbors, now we proceed to select important features for classification. Two algorithms are proposed and implemented for feature selection, which are suitable for different types of data. Algorithm 1 follows our first intuition and is more suitable for data whose features are independent with each other; while algorithm 2 is an attempt to fix shortcomings of algorithm 1.

Two algorithms share several common ideas. First of all, both of them are based on constructing a large number of small SVM classifiers on kNNs. GPU-Precompute-SVM mentioned above allow us to compute these small SVM classifiers concurrently and efficiently. Secondly, both algorithms use cross validation accuracy on kNNs to estimate the possibility of correctly classifying the query during the process. A strong argument for wrapper methods is that the estimated accuracy of the learning algorithm is the best available heuristic for measuring the values of features. Different learning algorithms may perform better with different features, even if they are using the same training data. Since kNNs forms a local area near the query, and they will be used as training examples for the query. We believe that the SVMs trained during cross validation on kNNs are similar to SVMs trained using all kNNs to classify the query. Both of them are classifiers for this local area. Therefore, its rational to use cross validation accuracy

for kNNs to estimate the possibility of correctly classifying the query. Thirdly, both algorithms rank the features by importance and follow the same steps to decide the final number of features used for classification.

3.4.1 Algorithm 1

The first intuitive assumption we had to evaluate the importance of a feature is that: the more dropping in accuracy it can cause to remove a feature, the more important the feature is. Thus, following algorithm is proposed. The pseudo code is shown below and figure 3.3 demonstrates several key steps for easy understanding.

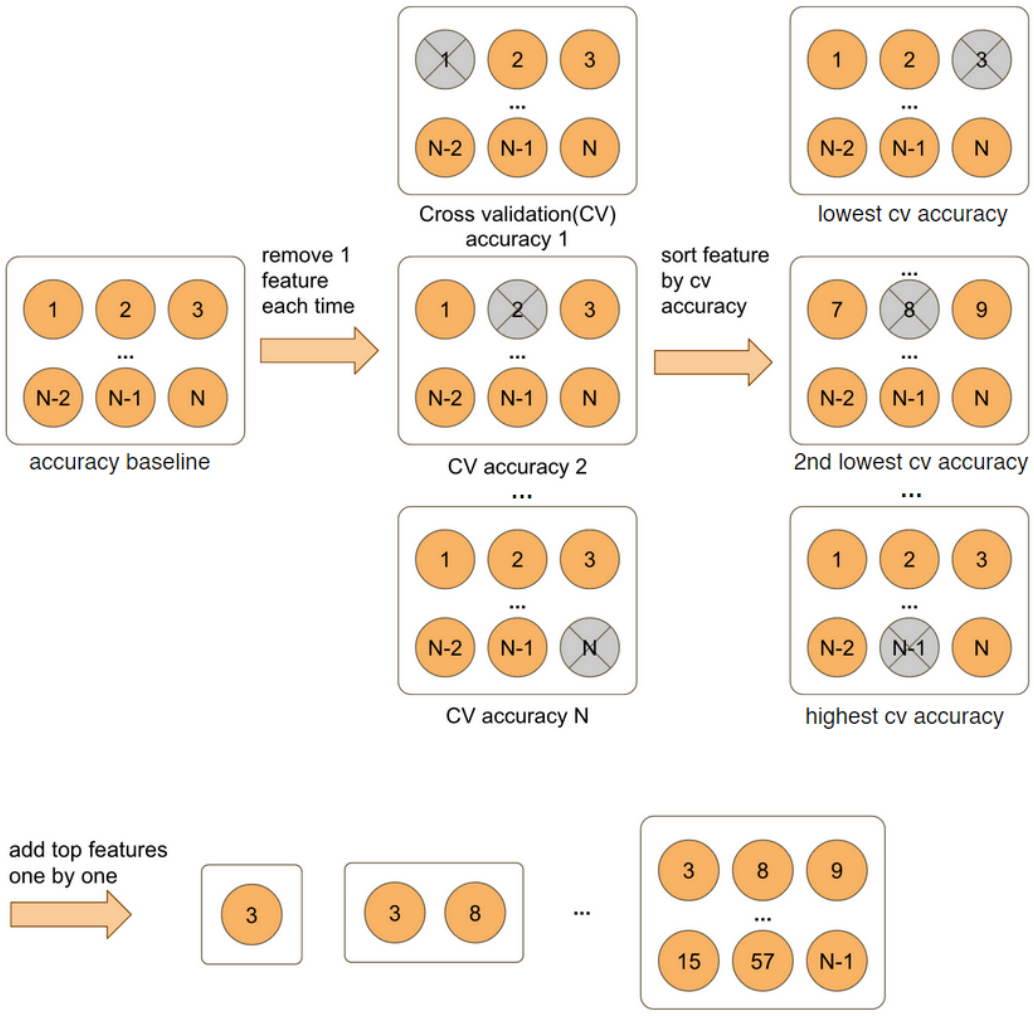


Figure 3.3: Key steps of feature selection algorithm 1

Algorithm 1 Dynamic Feature Selection Algorithm 1

$accuracy_{full}$ = CV–ACCURACY($kNNs$)
for each feature $i \in Features$ **do**
    $Neighbours_i = kNNs$ with feature i removed
    $accuracy_i$ = CV–ACCURACY($Neighbours_i$)
end for
$improvement_i = accuracy_{full} - accuracy_i$
$feature\text{–}order$ = SORT–FEATURES($improvements$)
$max\text{–}num\text{–}features = reduce\text{–}ratio * original\text{–}num\text{–}features$
for i from 1 to $max\text{–}num\text{–}features$ **do**
    $Neighbours2_i = kNNs$ with top i features selected
    $accuracy2_i$ = CV–ACCURACY($Neighbours2_i$)
end for
$features_{max}$ = features with highest $accuracy2_i$
classify the query with $features_{max}$ on $kNNs$

procedure CV–ACCURACY($data$)
    return cross validation accuracy of SVM on data
end procedure

procedure SORT–FEATURES(improvements)
    return order of features sorted by improvements
end procedure

Given a query and its kNNs, we begin with applying SVM cross validation on kNNs. This accuracy is used as a baseline for later comparison. Then, we remove each feature individually from data points in kNNs and performs SVM cross validation on the new datasets. The new cross validation accuracies are used as a measurement of how important a feature is. After that, we sort the features according to the cross validation accuracies we gained in the last step in ascending order.

Based on our assumption, the features are sorted in decreasing importance by this step. And now we need to determine the number of features required to classify the query. We start with the most important feature, then we add in features one by one, and see how cross validation accuracy changes with more features added. We choose the features with highest cross validation accuracy to build the SVM classifier for the query. In the pseudo code, there is an parameter, $reduce - ratio$. It refers to the upper limit for the ratio of final number of features selected versus original number of features. Since in later experiments, we found that the highest accuracy is usually achieved with small number of features, we add this upper limit to prune the computation. This processs of determining the final set of features is also different with existing wrapper methods.

3.4.2 Algorithm 2

Algorithm 1 works fine for the dataset we first evaluate. It's a dataset with 60 dimensions. However, when we start to work on data with hundreds of dimensions, the algorithm's performance is not satisfactory, which leads us to analyze the shortcomings of algorithm 1. Firstly, algorithm 1 remove one feature from data each time. For a high dimensional data, the effect caused by discarding one feature sometimes is too small to be detected by cross validation on kNNs. Secondly, algorithm 1 treats each feature independently, while some features may have correlations between each other. To overcome these problems, a different algorithm is proposed. The pseudo code is shown below and Figure 3.4 demonstrates several key steps for easy understanding.

Given a query and its kNNs, instead of removing features one by one, now we randomly divide all the features into 2 equal halves. For each feature, there is one half that contains it and the

Algorithm 2 Dynamic Feature Selection Algorithm 2

for i from 1 to N **do**

  $Neighbours_{i1}$, $Neighbours_{i2}$= randomly divide $kNNs$ into 2 equal halves

  $accuracy_{i1}$ = CV–ACCURACY($Neighbours_{i1}$)

  $accuracy_{i2}$ = CV–ACCURACY($Neighbours_{i2}$)

end for

for each feature $i \in Features$ **do**

  $$importance_i = \frac{sum\ of\ accuracies\ of\ N\ halves\ that\ contain\ feature_i}{sum\ of\ accuracies\ of\ N\ halves\ that\ don't\ contain\ feature_i}$$

end for

$feature\text{–}order$ = SORT–FEATURES($importances$)

$max\text{–}num\text{–}features = reduce\text{–}ratio * original\text{–}num\text{–}features$

for i from 1 to $max\text{–}num\text{–}features$ **do**

  $Neighbours2_i = kNNs$ with top i features selected

  $accuracy2_i$ = CV–ACCURACY($Neighbours2_i$)

end for

$features_{max}$ = features with highest $accuracy2_i$

classify the query with $features_{max}$ on $kNNs$

procedure CV–ACCURACY(data)

return cross validation accuracy of SVM on data

end procedure

procedure SORT–FEATURES(importances)

return order of features sorted by importances

end procedure

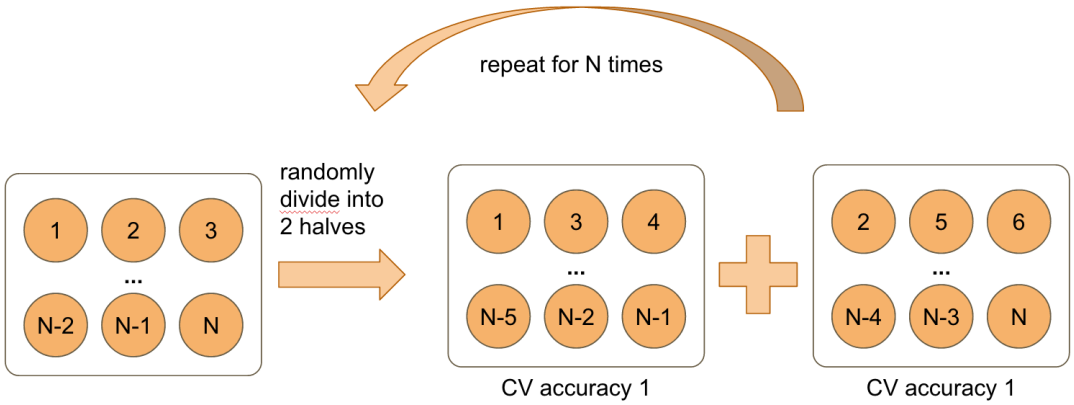


Figure 3.4: Key steps of feature selection algorithm 2

other half that doesn't contain it. We perform SVM cross validation on both halves of KNNs. If we repeat this random division for N times, there will be N halves that contain feature i and N halves that don't contain feature i. And a feature's importance is calculated with the formula below.

$$Importance_i = \frac{sum\ of\ accuracies\ of\ N\ halves\ that\ contain\ feature_i}{sum\ of\ accuracies\ of\ N\ halves\ that\ don't\ contain\ feature_i}$$

The rest of the process is the same as algorithm 1. We rank the features by their importances. Then we add top features one by one to settle the number of features needed.

3.5 Classification

Building a SVM classifier on the features selected is the last step of our system. We use the selected features of kNNs to train a SVM classifier and use it to classify the query. In addition, we also use the GPU-Precompute-SVM program for this step.

Chapter 4

Evaluation

In this chapter, we first introduce experimental settings and datasets we use to evaluate our program. Then, we present two sets of experiments to determine the appropriate value for some key parameters. And lastly, we show the classification results of our program.

4.1 Experimental Setup

This project is implemented in $C++$, using CUDA interface. The program runs on GPU Nvidia GTX Titan X to speed up the kNN and SVM process.

4.2 Datasets for Evaluation

We use three high-dimensional datasets to evaluate our system.

[Splice] This dataset is to classify whether a DNA sequence contains exon/intron boundaries (EI sites) or intron/exon boundaries (IE sites). Each sequence starts at position -30 and ends at position +30 of the junction. The data is available in the UCI Machine Learning Repository (Towell, Noordewier, & Shavlik, 1991). It contains 3175 instances (1000 for training and 2175 for testing) with 60 features. All features are categorical, each feature represents one of the A,G,C,T in that position. This is a relatively small dataset. We use it in the initial phase of this project to evaluate the algorithms and determine suitable values for parameters.

[MNIST] This dataset contains image of handwritten digits, available at MNIST webpage

(LeCun & Cortes, 2012). It has a training set of 60,000 examples, and a test set of 10,000 examples. The digits have been size-normalized and centered in a 28x28 image. Thus, the data is a 784 dimensional. Each dimension represents the grey value at one pixel.

[**SUN**] This dataset contains images for scene recognition task. Its available in MIT SUN Database website (Xiao & all, 2012). It contains 397 classes, we use the first 10 classes to test our program. There 3663 images for training and 500 images for testing. We only extract the GIST (Oliva & Torralba, 2006) information from the image. Therefore, each image is represented in 512 features.

4.3 Experiments for Key Parameters

There are many parameters to be determined for the whole program, such as k and selectivity for nearest neighbors search by GENIE, $reduce - ratio$ and N for the dynamic feature selection algorithm, and so on. Values of these parameters vary from data to data. We need to run sets of experiments to determine suitable values for each parameter when dataset changes. Out of all the experiments, result of choosing values for $reduce - ratio$ and N is quite interesting. We present them in this section.

4.3.1 Experiments for Parameter $reduce - ratio$

As mentioned in Section 3.4, after sorting the features by their importance, we need to determine the number of top features to classify the query. It's determined by adding features one by one according to their importance, and the features with the highest cross validation accuracy is chosen. However, for high dimensional data, this process itself is time consuming. Thus, we decide to set a upper limit of number of features we need to test on.

Figure 4.1 presents how accuracy changes when we add number of features selected. This experiment uses feature selection algorithm 1 on splice dataset. The x-axis of the figure represents the number of features used in cross validation test; while the y-axis represents the accuracy. The horizontal line is the cross validation accuracy gained with all features. It's a baseline to compare with. As figure 4.1 shows, the accuracy increases surprisingly fast for the first few

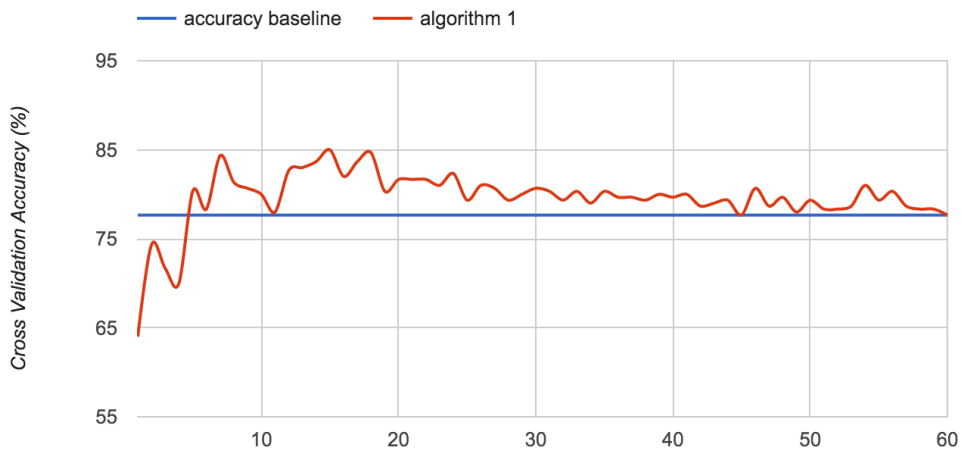


Figure 4.1: Relation between accuracy and number of top features selected (algorithm 1)

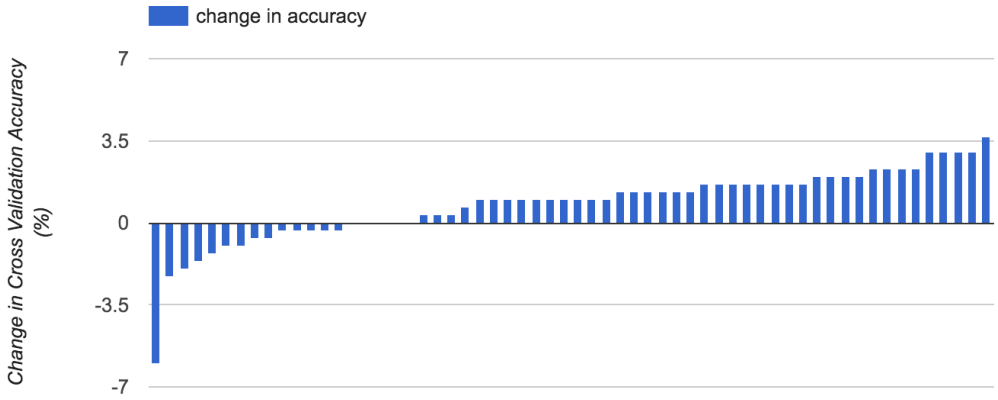


Figure 4.2: Change in accuracy when disgarding one feature from data

features selected. Then it exceeds the baseline using only a few features, which means it is possible to extract important information for classification on high dimensional data with small number of features. With more features added, the accuracy fluctuates and remains above the baseline. After the maximum is reached, the accuracy starts to decrease when more features are added for classification. Until all the features are added, the accuracy falls to the baseline. Figure 4.2 shows the change in accuracy when one feature is removed from the dataset. The features are already sorted by their importance. Thus, the feature removed in Figure 4.2 corresponds to the feature added in Figure 4.1. As we can see, if removing a feature causes

decreasing in accuracy, including it in classification can increase the accuracy. On the other hand, if removing a feature causes increasing in accuracy, then including it in classification can decrease the accuracy. These features may introduce noises to the classification task. This result supports our first intuitive assumption mentioned in section 3.4.

Although Figure 4.1 and 4.2 only demonstrate the feature selection algorithm for one query, we conducted this experiments on all the queries of Splice dataset. The result shows similar patterns, on both algorithm 1 and algorithm 2. We found that the maximum accuracy usually can be found within 12 features. Thus, we set $reduce - ratio$ to 0.2 for this dataset. Figure 4.3 shows a typical result for algorithm 1 and algorithm 2.

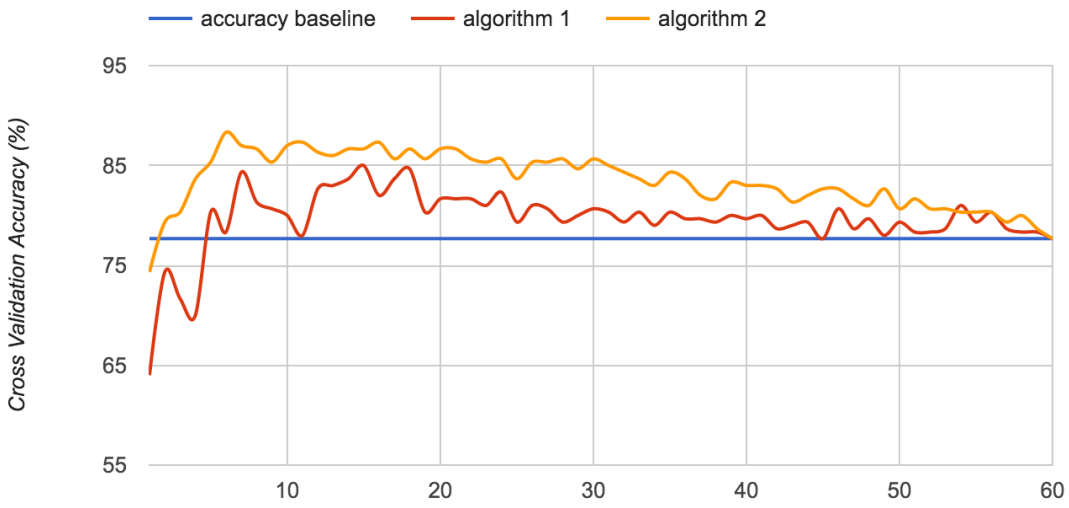


Figure 4.3: Relation between accuracy and number of top features selected (algorithm 2)

In addition, the $reduce - ratio$ may vary for different dataset. For a new dataset, we need to run a few queries to determine the value of $reduce - ratio$ in the beginning.

4.3.2 Experiments for Parameter N

N is an important parameter in algorithm 2. It's the number of times we randomly divide the features into two equal haves. Since we want to simulate the random combination of features, N is supposed to be a large number. To determine the number of N, we also use cross validation accuracy to estimate the possibility that a query can be correctly classified. Fiture 4.4 shows

the effect of different N values on SUN dataset.

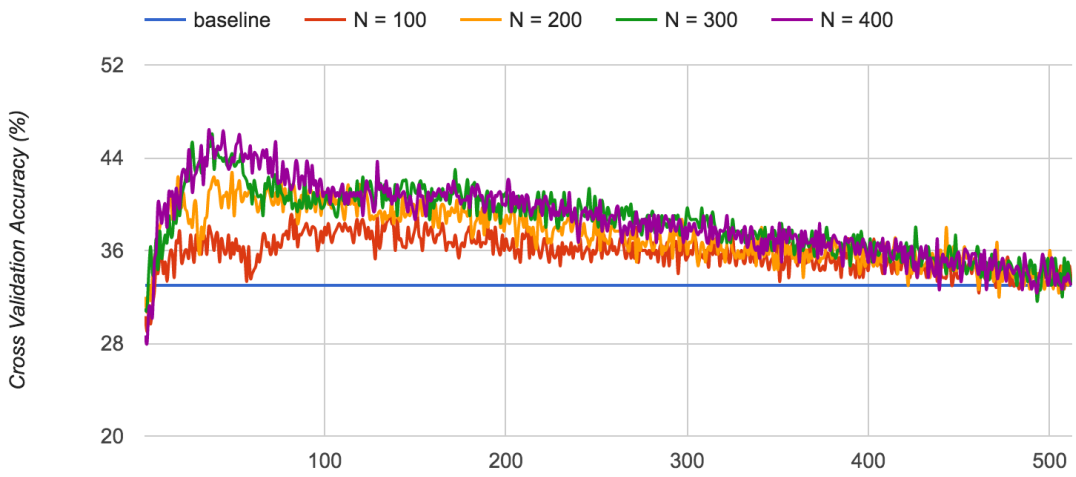


Figure 4.4: Relation between accuracy and value of N (algorithm 2)

Its not hard to notice that the pattern in Figure 4.4 is quite similar to the pattern in Figure 4.1. As we can see, when $N = 200$, the accuracy increases fast at the beginning, and the maximum accuracy is higher then that when $N = 100$. However, when we further increase N, there is no obvious improvement. Therefore, N is set to 200 as default value.

4.4 Classification Results and Discussion

After we determine the values of parameters through a series of experiments, we evaluate our classification paradigm and feature selection algorithms on training and testing data. For data Splice, both feature selection algorithms are applied. But for data MNIST and SUN, only algorithm 2 are applied due to their high dimensionality (>100). We classify each dataset with three methods using our system. Firstly, we use GENIE only to perform nearest neighbours search and classify the queries using ensemble method. Secondly, we classify the data by building SVMs on kNNs. And lastly, we classify the data with our feature selection approaches.

Table 4.1 shows our evaluation resullts. First of all, both of our feature selection algorithms can largely reduce the number of features used for classification. For dataset Splice, two algorithms select 12.8% (7.7/60) and 6.9% (5.54/60) of original features to classify the query. For dataset

MNIST, the ratio is as low as 4.5% (35.47/784). And the ratio is 9.4% (48.3/512) for data SUN. The algorithms select less than 15% of the features on average to classify the query for all datasets, but the classification results is still accurate. Compared with results of KNN and KNN+SVM, both feature selection algorithms help to increase the classification accuracy on all the three datasets we evaluated.

	Splice	MNIST	SUN
k for GENIE	100	100	100
N (for FS2)	100	100	200
Original features	60	784	512
Number of selected features with FS1	**7.73**	–	–
Number of selected features with FS2	**5.54**	**35.47**	**48.3**
KNN	79.59%	86.91%	38.2%
KNN+SVM	91.49%	92.32%	42.6%
KNN+FS1+SVM	93.61%	–	–
KNN+FS2+SVM	**94.12%**	**94.33%**	**44.2%**

Table 4.1: Classification accuracy on 3 real-life datasets

Chapter 5

Conclusion

5.1 Contributions

- Implemented a semi-lazy learning system based on GENIE and GPU-Precompute-SVM. It speeds up the process of classification on large training dataset and allows local estimation of target function.
- Proposed and implemented two dynamic feature selection algorithms for high dimensional dataset, which largely reduce the features used for classification and increase the classification accuracy.
- Evaluated the performances of the program using three real-world datasets.

5.2 Future Work

Some possible directions for future research work of this project are listed as following.

- The present approaches of dynamic feature selection both consider features independently when evaluate each feature's importance. However, features might be correlated with each other in some datasets. The dynamic feature selection algorithm can be improved by taking importance of combination of features into consideration.
- It is possible to apply this program on large scale image classification tasks. One of the

most important reasons that large scale image classification remains a difficult problem is the computational complexity brought by high dimensionality; while this program supports semi-lazy learning with dynamic feature selection. Actually, we did some research about image classification and found that complex pre-processing is usually required. The image needs to go through steps such as feature extraction, coding, pooling and concatenating before they can be used for classification. This is beyond the scope of this project, however, it is possible to apply this program on processed image datasets.

References

Bi, J., Breneman, M. E. C. M., & Song, M. (2003). Dimensionality reduction via sparse support vector machines. *Journal of Machine Learning Research*, *3*, March, 2003, 1229–1243.

Das, S. (2001). *Filters, wrappers and a boosting-based hybrid for feature selection* (Technical report). Harvard University.

Hall, M. (1999). *Correlation based feature selection for machine learning. doctoral dissertation* (Technical report). The University of Waikato, Dept of Comp. Sci.

Inza, I., Merino, M., Larraaga, P., Quiroga, J., Sierra, B., & Girala, M. (1999). *Feature subset selection by population-based incremental learninge* (Technical Report EHU-KZAA-IK-1/99). University of the Basque Country, Spain.

John, G. H., Kohavi, R., & Pfleger, K. (1994). Irrelevant features and the subset selection problem. *Proceedings of the 11th international conference*, , December, 1994, 121–129.

Kambhatla, N., & Leen, T. (1997). Dimension reduction by local principal component analysis. *Neutral Computation*, *9*(7), October, 1997, 1493–1516.

Kohavi, R., & John, G. H. (1997). Wrappers for feature subset selection. *Artificial Intelligence*, *1*, May, 1997, 273–304.

Kudo, M., & all (2000). Comparison of classifier-specific feature selection algorithms. *SSPR/SPR*, , December, 2000, 677–686.

LeCun, Y., & Cortes, C. (2012). The mnist database of handwritten digits.

McLachlan, G. J. (1992). *Selection of feature variables in discriminant analysis.*

Mitchell, T. M. (1997). *Machine learning.* McGraw-Hill.

Oliva, A., & Torralba, A. (2006). Building the gist of a scene: the role of global image features in recognition. *Progress in Brain Research*, *155*, December, 2006, 23–36.

Towell, G., Noordewier, M., & Shavlik, J. (1991). Molecular biology (splice-junction gene sequences) data set.

Xiao, J., & all (2012). Sun database.

Zhou, J., Guo, Q., Jagadish, H. V., Luan, W., Tung, A. K. H., & Zheng, Y. (2016). Generic inverted index on the gpu. , March, 2016.

Appendix A

Performances for GPU-Precompute-SVM

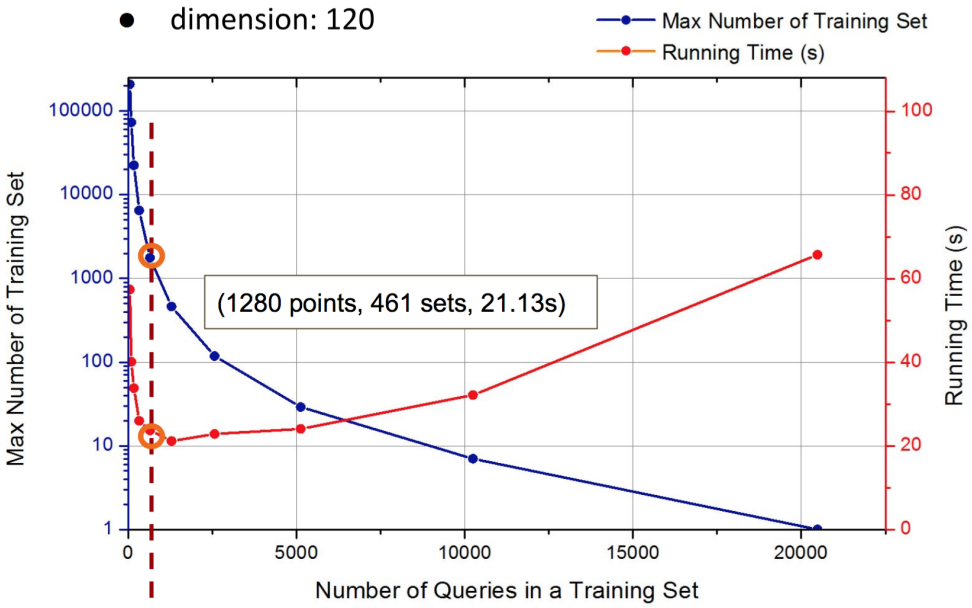


Figure A.1: Relations between max number of training set, training time and number of points per set using GPU-Precompute-SVM