Stability of Feature Selection Algorithms

Sourabh Vartak
Department of Computer Science
Binghamton University
State University of New York

Termination Project
Submitted in partial fulfillment of requirements for the degree of
Master of Science in Computer Science
In the Graduate School of
Binghamton University
State University of New York
2010
Termination Project

Accepted in partial fulfillment of requirements for the degree of
Master of Science in Computer Science
In the Graduate School of
Binghamton University
State University of New York
2010

Asst. Prof. Lei Yu
(M.S. Project Advisor)
Department of Computer Science
July, 2010
Table of Contents

TABLE OF FIGURES ........................................................................................................ 3

ABSTRACT ...................................................................................................................... 4

1. INTRODUCTION ........................................................................................................ 5

   FEATURE (VARIABLE) SELECTION ............................................................................. 5
   STABILITY OF FEATURE SELECTION ...................................................................... 6

2. EXPERIMENTAL PLATFORMS .................................................................................... 9

   R STATISTICAL COMPUTING LANGUAGE ................................................................ 9
   R AND JAVA CONNECTIVITY .................................................................................... 9

3. DIFFERENT ALGORITHMS EVALUATED .................................................................. 10

4. EXPERIMENTAL SETUP ......................................................................................... 14

5. EXPERIMENTAL RESULTS ...................................................................................... 16

6. CONCLUSION ........................................................................................................... 23

7. ACKNOWLEDGEMENT .......................................................................................... 23

8. APPENDIX ............................................................................................................... 24

9. REFERENCES .......................................................................................................... 25

Table of Figures

FIGURE 1. STABILITY ON COLON DATASET ................................................................. 16
FIGURE 2. STABILITY ON LEUKEMIA DATASET ......................................................... 17
FIGURE 3. STABILITY ON LUNG DATASET .................................................................. 17
FIGURE 4. STABILITY ON PROSTATE DATASET .......................................................... 18
FIGURE 5. SMO CLASSIFICATION ACCURACY ON COLON DATASET ......................... 19
FIGURE 6. IB1 CLASSIFICATION ACCURACY ON COLON DATASET ............................. 19
FIGURE 7. SMO CLASSIFICATION ACCURACY ON LEUKEMIA DATASET ..................... 20
FIGURE 8. IB1 CLASSIFICATION ACCURACY ON LEUKEMIA DATASET ....................... 20
FIGURE 9. SMO CLASSIFICATION ACCURACY ON LUNG DATASET ............................ 21
FIGURE 10. IB1 CLASSIFICATION ACCURACY ON LUNG DATASET ............................ 21
FIGURE 11. SMO CLASSIFICATION ACCURACY ON PROSTATE DATASET ................. 22
FIGURE 12. IB1 CLASSIFICATION ACCURACY ON PROSTATE DATASET ...................... 22
FIGURE 13. CODE SNIPPET FOR STARTING AND STOPPING R ENGINE ..................... 24
FIGURE 14. CODE SNIPPET FOR CALLING ELASTIC-NET FUNCTION IN R FROM JAVA .............................................................. 24
Abstract

In gene microarray data analysis, typical number of features is thousands or tens of thousands, but the number of samples is often less than a hundred. Due to high-dimensionality of such data, feature selection is very important to improve classification accuracy. Stability of feature selection is the sensitivity of the algorithm to variations in training set. In biomedical domain, feature selection algorithms with both high classification accuracy and high stability are desired.

In my MS project, I have evaluated stability and classification accuracy of three different feature selection algorithms, namely, Elastic-net, Lasso and HHSVM. Empirical studies performed in my project conclude that Elastic-net is the most stable amongst the three algorithms while Lasso is least stable. All three algorithms show good classification accuracy especially for dataset with more number of samples.
1. Introduction

Feature (variable) selection

Feature selection is the problem of selecting a minimum subset of original features for best predictive accuracy (1). In biomedical domain, feature selection is very important since the typical number of features in microarray data is thousands or tens of thousands, but the number of samples is often less than a hundred. This is referred to as “curse of dimensionality” associated with high-dimensional data.

In the gene selection problem, features are gene expression coefficients corresponding to mRNA in a tissue sample for a number of patients. (2) A typical classification task is to separate healthy patients from cancer patients, based on their gene expression profile. Usually fewer than 100 samples (patients) data are available overall for training and testing. But, the number of variables ranges from 6000 to 60,000. Some initial pre-processing usually reduces the number of features to a few thousand. Many of these features are redundant or irrelevant to class label which leads to very poor classification accuracy. Hence, feature selection is performed to select only those features which can improve predictive accuracy.

Potential benefits of feature selection are facilitating data visualization and data understanding, reducing the measurement and storage requirements, reducing training and utilization times, and defying the curse of dimensionality to improve prediction performance. (2)

Feature selection algorithms are categorized as feature ranking and subset selection. Feature ranking ranks the features by a metric and eliminates all features that do not achieve an adequate score while subset selection searches the set of possible features for the optimal subset that can be used to build a good predictor. (2)

Feature subset selection algorithms are divided into wrappers, filters and embedded methods. Wrappers utilize the learning machine of interest as a black box to score subsets of features according to their predictive power. Filters select subsets of features as a pre-processing step, independently of the chosen predictor. Embedded methods perform feature selection in the process of training and are usually specific to given learning machines. (2)
**Stability of Feature Selection**

Stability is defined as the sensitivity of an algorithm to variations in the training set (3). Stability of feature selection algorithm measures how different sub-samples of a training set affect the algorithm’s assessment of a particular feature's importance and consequently the final set of features selected by the algorithm. The motivation for investigating the stability of feature selection algorithms comes from the need to provide application domain experts with quantified evidence that the selected features are relatively robust to variations in the training data. (3)

The generic model of classification includes following:

i) A generator of random vectors $x$, drawn according to an unknown but fixed probability distribution $P(X)$

ii) A supervisor that assigns class labels $x$, to the $x$ random vectors, according to an unknown but fixed conditional probability distribution $P(C|X)$

iii) A learning space populated by pairs $(x, c)$ drawn from the joint probability distribution $P(X|C) = P(C|X) P(X)$. (3)

Now, the stability of a feature selection algorithm as the sensitivity of the feature preferences it produces to differences in training sets drawn from the same generating distribution $P(X|C)$. (3)

Stability measurement requires a similarity measure for feature preferences. Three different types of representation languages for feature preferences are weighting each feature according to importance, ranking features and lastly, selecting only a set of features without weighting or ranking. Any weighting schema can be cast as a ranking schema, which in turn can be cast as a set of features by setting a threshold on the ranks or requesting for a fixed number of features. (3) Interestingly, the most important information is delivered when we are examining the stability of the methods for sets of selected features of given cardinality, followed by ranking schema and lastly, least information is provided by weighting schema. (3)
The stability of a feature selection algorithm can be measured with respect to the selected subsets using some similarity measure. In the similarity based approach, the stability of a feature selection algorithm is measured by the average over all pair-wise similarity comparisons among all feature subsets obtained by the same algorithm from different sub-samplings of a data set. (3)

Let \( \{D_i\} \) where \( i=1 \) to \( q \) be a set of sub-samplings of a dataset of the same size, and \( S_i \) be the subset selected by a feature selection algorithm \( F \) on the sub-sampling \( D_i \).

The stability of \( F \) is given as follows:

\[
\overline{Sim} = \frac{2 \sum_{i=1}^{q} \sum_{j=i+1}^{q} Sim(S_i, S_j)}{q(q-1)}
\]

where \( Sim(S_i, S_j) \) represents a similarity measure between two subsets.

The similarity measure used in my MS project to measure stability of feature selection algorithms is as follows:

Let \( R_1 \) and \( R_2 \) be two sets of feature selection results, then similarity between \( R_1 \) and \( R_2 \) can be represented as

\[
Sim_{ID}(R_1, R_2) = \frac{2|R_1 \cap R_2|}{|R_1| + |R_2|}
\]

where the subscript ID indicates that the similarity is decided by matching feature indexes between the two subsets. (1)

The more complex similarity measures based on maximum weighted bipartite matching proposed in research paper referenced in (1) are not used since they are useful when feature selection algorithm returns groups (clusters) of selected features rather than individual selected features. But, the algorithms evaluated in my MS project do not return feature groups.
The main contribution of my MS project is evaluating three different feature selection algorithms for their stability based on above similarity measure and also compare their classification accuracy on two Weka classifiers, namely IB1 (1-nearest neighbor) and SMO (Sequential Minimal Optimization).

In Section 2, we look at the experimental platforms used. The feature selection algorithms are implemented in R language and the stability and classification accuracy experimenter is written in Java language. The classifiers used are from the open-source Weka data mining software.

In Section 3, I describe the various feature selection algorithms evaluated, namely, Lasso, Elastic-net and HHSVM, in detail.

In Section 4, we look at the experimental setup and the different datasets used to perform the stability evaluation and comparison.

In Section 5, we observe the various results generated over different experimental settings and across different datasets. We study various graphs which illustrate the trends of various stability and classification accuracy results obtained.

In Section 6, we finally conclude the findings of the project.
2. Experimental Platforms

*R statistical computing language*

R is a programming language and software environment for statistical computing and graphics. (4) The R language has become a *de facto* standard among statisticians for the development of statistical software, and is widely used for statistical software development and data analysis. Algorithms written in R are often bundled in packages which can be loaded in memory and accessed by other R programs. Comprehensive R Archive Network (CRAN) is the central repository for all the available R packages. When R package is not available, we can directly include R source files in our program similar to the #include directive in C or import statement in Java.

*R and Java connectivity*

rJava is a simple R-to-Java interface. It is comparable to the .C/.Call C interface. rJava provides a low-level bridge between R and Java (via JNI). It allows us to create objects, call methods and access fields of Java objects from R. (5) JRI, the inverse of rJava, is a Java/R Interface, which allows us to run R inside Java applications as a single thread. Basically it loads R dynamic library into Java and provides a Java API to R functionality. It supports both simple calls to R functions and a full running REPL. JRI is now shipped as a part of the rJava package, although it still can be used as a separate entity (especially for development).

In my MS project, I have used JRI to call R algorithms from Java code.

Code snippets for starting and stopping R engine and for calling R algorithm from Java are shown in Appendix.


### 3. Different algorithms evaluated

I evaluated various algorithms implemented in R language for their stability over variations in training data. These are LASSO (Least Absolute Shrinkage and Selection Operator), Elastic-net and Hybrid Huberized Support Vector Machines (HHSVM).

The Lasso is a penalized least squares regression method imposing an L1-penalty on the regression coefficients. Owing to the nature of the L1-penalty, the lasso does both continuous shrinkage and automatic variable (feature) selection simultaneously. (6)

**Definition of Lasso estimate:**

Suppose that we have data \((x^i, y^i), i = 1, 2, \ldots, N\) where \(x^i = (x_{i1}, \ldots, x_{ip})^T\) are the predictor variables and \(y^i\) are the responses. Similar to usual regression set-up, it is assumed that either the observations are independent or that the \(y^i\)s are conditionally independent given the \(x_{ij}\)s. Also, it is assumed that the \(x_{ij}\) are standardized so that \(\sum_i x_{ij} / N = 0\) and \(\sum_i x_{ij}^2 / N = 1\). (7)

Letting \(\hat{\beta} = (\hat{\beta}_1, \ldots, \hat{\beta}_p)^T\), the lasso estimate \((\hat{\alpha}, \hat{\beta})\) is defined by,

\[
(\hat{\alpha}, \hat{\beta}) = \arg \min \left\{ \sum_{i=1}^{N} \left( y^i - \alpha - \sum_j \beta_j x_{ij} \right)^2 \right\} \quad \text{subject to } \sum_j |\beta_j| \leq t.
\]

Here, \(t \geq 0\) is the tuning parameter.

For all \(t\), the solution for \(\alpha\) is \(\hat{\alpha} = \bar{y}\). (7)

Elastic-net is a regularization and feature selection method which was proposed as an improvement over the earlier Lasso method, while maintaining similar sparsity of representation. Elastic-net is like a stretchable fishing net that retains ‘all the big fish’. (6) The elastic-net also encourages a grouping effect, where strongly correlated predictors (features) tend to be in or out of the model together. The elastic net is particularly useful when the number of predictors (\(p\)) is much bigger than the number of observations (\(n\)). An algorithm called LARS-EN is proposed by the authors for computing elastic net regularization paths efficiently, much like Least Angle Regression (LARS) algorithm does for the Lasso. (6)
**Definition of naïve elastic-net:**

Suppose that the data set has n observations with p predictors. Let \( y = (y_1, ..., y_n)^T \) be the response and \( X = (x_{11}, ..., x_{p1})^T \) be the model matrix, where \( x_j = (x_{1j}, ..., x_{nj})^T, j = 1, ..., p, \) are the predictors. After a location and scale transformation, we can assume that the response is centered and the predictors are standardized,

\[
\sum_{i=1}^{n} y_i = 0, \quad \sum_{i=1}^{n} x_{ij} = 0 \quad \text{and} \quad \sum_{i=1}^{n} x_{ij}^2 = 1, \quad \text{for } j = 1, 2, \ldots, p.
\]

For any fixed non-negative \( \lambda_1 \) and \( \lambda_2 \), the naïve elastic-net criterion is defined as follows:

\[
L(\lambda_1, \lambda_2, \beta) = |y - X\beta|^2 + \lambda_2 |\beta|^2 + \lambda_1 |\beta|_1 \quad [1]
\]

where

\[
|\beta|^2 = \sum_{j=1}^{p} \beta_j^2, \quad |\beta|_1 = \sum_{j=1}^{p} |\beta_j|.
\]

The naïve elastic-net estimator \( \hat{\beta} \) is the minimizer of equation [1]

\[
\hat{\beta} = \text{arg min}_{\beta} \{ L(\lambda_1, \lambda_2, \beta) \}
\]

This procedure can be viewed as a penalized least squares method. Let \( \alpha = \lambda_2 / (\lambda_1 + \lambda_2) \) then solving \( \hat{\beta} \) in equation [1] is equivalent to the optimization problem

\[
\hat{\beta} = \text{arg min}_{\beta} |y - X\beta|^2, \quad \text{subject to } (1 - \alpha) |\beta|_1 + \alpha |\beta|^2 \leq t \text{ for some } t.
\]

The function \( (1 - \alpha) |\beta|_1 + \alpha |\beta|^2 \) is called the elastic-net penalty, which is a convex combination of the lasso and ridge penalty. When \( \alpha=1 \), the naïve elastic-net becomes simple ridge regression.

For all \( \alpha \in [0, 1) \), the elastic net penalty function is singular (without first derivative) at 0 and it is strictly convex for all \( \alpha>0 \), thus having the characteristics of both the lasso and ridge regression. Note that the lasso penalty \( (\alpha=0) \) is convex but not strictly convex. (6)
**Deficiency of the naïve elastic net:**

As an automatic variable selection method, the naïve elastic net overcomes the limitations of the lasso. However, empirical evidence shows that the naïve elastic net does not perform satisfactorily unless it is very close to either ridge regression or the lasso. This is why we call it naïve. (6)

In the regression prediction setting, an accurate penalization method achieves good prediction performance through the bias–variance trade-off. The naïve elastic net estimator is a two-stage procedure: for each fixed $\lambda_2$ we first find the ridge regression coefficients, and then we do the lasso-type shrinkage along the lasso coefficient solution paths. It appears to incur a double amount of shrinkage. Double shrinkage does not help to reduce the variances much and introduces unnecessary extra bias, compared with pure lasso or ridge shrinkage. The elastic-net estimator improves the prediction performance of the naïve elastic-net by correcting this double shrinkage. (6)

**Definition of elastic-net estimate:**

Given data $(y, X)$, penalty parameter $(\lambda_1, \lambda_2)$ and augmented data $(y^*, X^*)$, the naïve elastic-net solves a lasso-type problem

$$\hat{\beta}^* = \arg \min_{\beta^*} \|y^* - X^* \beta^*\|^2 + \lambda_1 \frac{1}{\sqrt{1 + \lambda_2}} |\beta^*|_1$$

The elastic net (corrected) estimates $\hat{\beta}$ are defined by,

$$\hat{\beta}(\text{elastic net}) = \sqrt{(1 + \lambda_2)} \hat{\beta}^*$$

Since, $\hat{\beta}(\text{naïve elastic net}) = \{1/\sqrt{(1 + \lambda_2)}\} \hat{\beta}^*$, therefore,

$$\hat{\beta}(\text{elastic net}) = (1 + \lambda_2) \hat{\beta}(\text{naïve elastic net})$$

Hence, elastic-net coefficient is a rescaled naïve elastic-net coefficient. Such a scaling transformation preserves the variable selection property of the naïve elastic-net and is the simplest way to undo shrinkage. Thus, all the good properties of naïve elastic-net hold for elastic-net. (6)
The L1-norm SVM can automatically select genes via the L1-norm regularization. However, it has following two limitations:
a) The number of selected genes is upper-bounded by the sample size. Therefore, when the number of relevant genes exceeds the sample size, the L1-norm SVM can only discover a portion of them.
b) For the highly correlated and relevant genes, the L1-norm SVM tends to pick only one or a few of them. (8)

The HHSVM has the form of “loss” + “penalty”, where it uses the huberized hinge loss function to measure the loss and the elastic-net penalty to control the complexity of the model. (8) The HHSVM algorithm overcomes the above mentioned drawbacks of L1-norm SVM by encouraging grouping effect similar to elastic-net.

The elastic-net penalty given by,
\[ \lambda_1 \| \beta \|_1 + \frac{\lambda_2}{2} \| \beta \|_2^2 \]
is a hybrid of the L1-norm and the L2-norm penalties. (8)

The hybrid huberized support vector machine (HHSVM) applies the elastic-net penalty to the SVM as follows:
\[
\min_{\beta, \alpha} \sum_{i=1}^{n} \phi(y_i (\beta_0 + x_i^T \beta)) + \lambda_1 \| \beta \|_1 + \frac{\lambda_2}{2} \| \beta \|_2^2
\]
where \( \lambda_1, \lambda_2 \geq 0 \) are regularization parameters. Increasing \( \lambda_1 \) tends to eliminate more irrelevant variables; and increasing \( \lambda_2 \) makes the “grouping effect” more prominent. (8)

Note that instead of using the standard hinge loss function of the SVM, HHSVM uses the huberized hinge loss function to measure “badness-of-fit” as shown below:
\[
\phi(y_f) = \begin{cases} 
0, & \text{for } y_f > 1, \\
(1 - y_f)^2 / 2\delta, & \text{for } 1 - \delta < y_f \leq 1, \\
1 - y_f - \delta / 2, & \text{for } y_f \leq 1 - \delta,
\end{cases}
\]
where \( \delta \geq 0 \) is a pre-defined constant. (8)
4. Experimental Setup

I evaluated stability of various feature selection algorithms implemented in R language using a Java stability experimenter developed by Steven Loscalzo.

I experimented with four public frequently studied microarray datasets having following characteristics:

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Number of Genes</th>
<th>Number of samples</th>
<th>Number of class labels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Colon</td>
<td>2000</td>
<td>62</td>
<td>2</td>
</tr>
<tr>
<td>Leukemia</td>
<td>7129</td>
<td>72</td>
<td>2</td>
</tr>
<tr>
<td>Lung</td>
<td>5000</td>
<td>181</td>
<td>2</td>
</tr>
<tr>
<td>Prostate</td>
<td>6033</td>
<td>102</td>
<td>2</td>
</tr>
</tbody>
</table>

The experimenter has the several nested loops as shown in the following pseudo code:

Loop 1:
For each dataset, read Arff file to create a Weka Instances object from the data

Loop 2:
Do 10 random shuffles of the original dataset

Loop 3:
Do 10 fold cross validation for each shuffle.
Pass training data to R algorithm like HHSVM, Elastic-net or Lasso.
Get the list of selected feature groups from the R algorithm.
Calculate the stability of feature selection algorithm for each fold.
Build a classifier using the testing data.
Calculate the classification accuracy of the classifier.

The above 10 fold X 10 shuffle experimenter is ran for Lasso, Elastic-net and HHSVM 10 different times, each time varying the number of steps taken by the algorithm from 10 to 100. Such experiments are performed on each of the 4 datasets.

For each data set, each feature vector is normalized so that the mean over its components is zero and the standard deviation is one. (1)
To evaluate the stability of feature selection algorithms under sample hold out, each data set was randomly partitioned into 10 folds, with each fold containing 1/10 of all the samples. The feature selection algorithm was repeatedly applied to 9 out the 10 folds, while a different fold was hold out each time. This process was repeated 10 times for different partitions of the data set. Overall, a total of 10 X 10 different subsets of samples were used to generate different sets of features by the feature selection algorithm. (1)

For evaluating the generalization ability and stability of feature selection algorithm, each of the 100 subsets of samples in the previous study was used as the training set to select relevant features produced by feature selection algorithm, and then train classifiers based on selected features. The corresponding hold-out fold was used as the test set. (1)

Both sophisticated Sequential Minimal Optimization (SMO) and simple KNN (K=1) called IB1 classification algorithms were used to evaluate the generalization ability of the features selected by feature selection algorithm. The classifiers used are Weka software implementations. (9) The average predictive accuracy over the 100 folds was used as the measure for generalization ability.
5. Experimental Results

<table>
<thead>
<tr>
<th>#Steps</th>
<th>Elastic-net</th>
<th>HHSVM</th>
<th>Lasso</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.77</td>
<td>0.66</td>
<td>0.67</td>
</tr>
<tr>
<td>20</td>
<td>0.72</td>
<td>0.63</td>
<td>0.55</td>
</tr>
<tr>
<td>30</td>
<td>0.68</td>
<td>0.59</td>
<td>0.53</td>
</tr>
<tr>
<td>40</td>
<td>0.65</td>
<td>0.58</td>
<td>0.53</td>
</tr>
<tr>
<td>50</td>
<td>0.64</td>
<td>0.58</td>
<td>0.53</td>
</tr>
<tr>
<td>60</td>
<td>0.64</td>
<td>0.58</td>
<td>0.52</td>
</tr>
<tr>
<td>70</td>
<td>0.65</td>
<td>0.59</td>
<td>0.51</td>
</tr>
<tr>
<td>80</td>
<td>0.66</td>
<td>0.6</td>
<td>0.51</td>
</tr>
<tr>
<td>90</td>
<td>0.67</td>
<td>0.6</td>
<td>0.5</td>
</tr>
<tr>
<td>100</td>
<td>0.67</td>
<td>0.6</td>
<td>0.49</td>
</tr>
</tbody>
</table>

The above table shows the comparison of the stability of Elastic-net, HHSVM and Lasso on colon dataset. Let us plot a graph of the above results.

![Feature Selection Stability](Image)

*Figure 1. Stability on colon dataset*

We can see that the stability of all the three algorithms goes of decreasing as the number of steps for which the algorithm is run is increased. Elastic-net has better stability on colon dataset than HHSVM and Lasso is the least stable.
Now, let us compare the stability of the three algorithms on leukemia dataset.

![Figure 2. Stability on leukemia dataset](image)

As we can see the trend is similar to colon dataset but here Lasso is performing even worse as the number of steps taken reach 100.

Let us compare stability on lung cancer dataset.

![Figure 3. Stability on lung dataset](image)
Here, Lasso performed better than HHSVM till 30 steps were taken but then onwards the typical trend of Lasso being the least stable continued. Elastic-net has been most stable on all three datasets so far.

Finally, we take a look at the prostate dataset.

![Feature Selection Stability](image)

*Figure 4. Stability on prostate dataset*

Here, trend is similar to other three datasets, Elastic-net being most stable, followed by HHSVM and then Lasso being least stable.

Now we will show the graph of classification accuracy obtained on the testing folds of colon dataset by the three algorithms for Weka classifier called Sequential Minimal Optimization (SMO).
As we can see the classification accuracy of all the three algorithms lies between 80% and 90% though it goes on decreasing with increase in the number of steps. Again, Lasso has the least SMO classification accuracy on colon dataset.

Below is the graph of IB1 classification accuracy on colon dataset.

We can observe that for IB1 classifier, accuracy of the algorithms is low ranging from 78% to 83%.
Now, we see SMO classification accuracy trends for leukemia dataset.

![Classification Accuracy - SMO](image)

**Figure 7. SMO classification accuracy on leukemia dataset**

For leukemia dataset the range of classification accuracy is 92% to 97% which is higher than colon dataset. Also, accuracy doesn’t decrease with the number of steps for which algorithm is ran.

IB1 classification accuracy on leukemia dataset is as follows, which is slightly lower than SMO accuracy but shows similar trend.

![Classification Accuracy - IB1](image)

**Figure 8. IB1 classification accuracy on leukemia dataset**
Now, we will take a look at SMO classification accuracy for lung cancer dataset.

As we can see from the graph, SMO classification accuracy for lung cancer dataset is really impressive at 99%. This is most probably because the number of samples for lung dataset is more than double that of colon and leukemia datasets.

Also, IB1 classification accuracy is very impressive at 98% to 99% on lung dataset as shown below.

---

**Figure 9. SMO classification accuracy on lung dataset**

**Figure 10. IB1 classification accuracy on lung dataset**
Finally, below is the trend for prostate dataset. All three algorithms show good accuracy between 89% and 93% for SMO classifier.

![SMO classification accuracy on prostate dataset](image1)

*Figure 11. SMO classification accuracy on prostate dataset*

Similarly, all three algorithms show good accuracy ranging from 86% to 92% for IB1 classifier for prostate dataset.

![IB1 classification accuracy on prostate dataset](image2)

*Figure 12. IB1 classification accuracy on prostate dataset*
6. Conclusion

In my MS project, I have evaluated and compared the stability and classification accuracy of Elastic-net, Lasso, and HHSVM algorithms for four different datasets using Java experimenter. Since the feature selection algorithms evaluated by me are implemented in R language and stability experimenter is in Java language, I had to connect Java and R using JRI interface.

From the experimental results, it is found that Elastic-net is most stable, followed by HHSVM and then Lasso. The stability of the algorithms varies with different datasets but the general trend of decreasing stability with increasing number of steps is common.

All algorithms show good classification accuracy ranging from above 80% to 97% for colon, leukemia and prostate datasets. Classification accuracy for lung cancer dataset is really impressive at almost 99%, most probably due to double number of samples than the other three datasets. Also, IB1 classification accuracy is slightly lower than SMO classifier for all datasets for all feature selection algorithms.

7. Acknowledgement

I would like to thank my advisor Prof. Lei Yu for his helpful discussions and comments on my MS project. I would also like to thank Steven Loscalzo for the stability and classification experimenter written by him in Java which I have used in my MS project.
8. Appendix

```java
package research.util;

import org.rosuda.REngine.REngine;

public class R {
    public static REngine re;

    public static void startEngine() {
        // just making sure we have the right version of everything
        System.err.println("** Version mismatch - Java files don't match library version.*");
        System.exit(1);
    }

    String[] args = new String[]{};
    re = new REngine(args, false, new TextConsole());

    // the engine creates R as a new thread, so we should wait until it's ready
    if (!re.waitForR()) {
        System.err.println("Cannot load R");
    }

    public static void endEngine() {
        re.end();
        System.out.println("R engine shutdown successfully!!!");
    }
}
```

**Figure 13. Code snippet for starting and stopping R engine**

```java
public List<FeatureGroup> getFeatureGroups(Instances data) {
    super.convertToR(data);

    R.re.eval("library(elasticnet)");
    R.re.eval("output <-enet(dataTrain, dataTest, lambda=" + lambda + ",
            max.steps=" + maxSteps + ", silent=TRUE, 
            eps=1e-6)");

    REP allset = R.re.eval("outputAllset");

    System.out.println("1-based indexes of features selected by R algorithm: ");
    R.re.eval("print(outputAllset)");

    int[] selFeatures = allset.toIntArray();
    List<FeatureGroup> groups = new ArrayList<FeatureGroup>();

    for (int featureIndex : selFeatures) {
        Feature feat = new Feature();

        // REMINDER: Convert the 1-based R features to 0-based Java features
        feat.setIndex(featureIndex - 1);

        double[] instance = new double[data.numInstances.length];
        for (int instanceNum = 0; instanceNum < instance.length; ++instanceNum)
            instance[instanceNum] = data.instance(instanceNum).value(featureIndex - 1);
        feat.setInstance(instance);

        groups.add(FeatureGroup.createSingletonGroup(feat));
    }

    return groups;
}
```

**Figure 14. Code snippet for calling Elastic-net function in R from Java**
9. References

1. *Stable Feature Selection via Dense Feature Groups.* Yu, Lei, Loscalzo, Steven and Ding, Chris. 2008. KDD.


