

COMPARATIVE CLASSIFICATION OF PROSTATE CANCER DATA USING THE
SUPPORT VECTOR MACHINE, RANDOM FOREST, DUALKS AND K-NEAREST
NEIGHBOURS

A Thesis
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Kekoura Sakouvogui

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Statistics

July 2015

Fargo, North Dakota

North Dakota State University
Graduate School

Title

COMPARATIVE CLASSIFICATION OF PROSTATE CANCER DATA
USING THE SUPPORT VECTOR MACHINE, RANDOM FOREST,
DUALKS AND K-NEAREST NEIGHBOURS

By

Kekoura Sakouvogui

The Supervisory Committee certifies that this *disquisition* complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Yarong Yang

Chair

Dr. Rhonda Magel

Dr. Ying Huang

Approved:

7/10/2015

Date

Dr. Rhonda Magel

Department Chair

ABSTRACT

This paper compares four classifications tools, Support Vector Machine (SVM), Random Forest (RF), DualKS and the k -Nearest Neighbors (k NN) that are based on different statistical learning theories. The dataset used is a microarray gene expression of 596 male patients with prostate cancer. After treatment, the patients were classified into one group of phenotype with three levels: PSA (Prostate-Specific Antigen), Systematic and NED (No Evidence of Disease). The purpose of this research is to determine the performance rate of each classifier by selecting the optimal kernels and parameters that give the best prediction rate of the phenotype. The paper begins with the discussion of previous implementations of the tools and their mathematical theories. The results showed that three classifiers achieved a comparable performance that was above the average while DualKS did not. We also observed that SVM outperformed the k NN, RF and DualKS classifiers.

ACKNOWLEDGMENTS

I would like to express my special gratitude and appreciation to Dr. Yarong Yang for her time, advice, support and most importantly for being able to work with me. I am greatly thankful to her for giving me the idea of the thesis.

Special thanks and appreciation go to Aurora Manley for her feedback. I am more than appreciative for her comment and her help.

I would like to thank my family. Without their help, I would not be where I am today. Their support and encouragement is making a huge difference. Special thanks go to God for protecting and always being with me.

TABLE OF CONTENTS

ABSTRACT.....	iii
ACKNOWLEDGMENTS.....	iv
LIST OF TABLES.....	viii
LIST OF FIGURES.....	ix
CHAPTER 1. INTRODUCTION AND LITERATURE REVIEW.....	1
CHAPTER 2. SUPPORT VECTOR MACHINE CLASSIFICATION.....	3
2.1. Introduction of the SVM.....	3
2.2. Support Vector Machine.....	3
2.3. Separation of the Hyperplanes.....	4
2.4. Optimal Separation of the SVM.....	6
2.5. SVM Software.....	8
2.6. Brief Introduction of SVM to the Multiclass.....	8
2.6.1. One-versus-The Rest Classification.....	8
CHAPTER 3. k - NEAREST NEIGHBOURS.....	10
3.1. Introduction of the k NN.....	10
3.2. Distance or Similarity of k NN.....	10
3.3. Classification of the k NN Algorithm.....	11
3.4. Estimating the number of neighbours k of the k NN.....	12
3.5. k NN Software.....	12
CHAPTER 4. RANDOM FOREST.....	13
4.1. Introduction of the Random Forest.....	13
4.2. Classification of RF.....	13

4.3. Estimating the Error Rate of RF.....	14
4.4. RF Software.....	14
CHAPTER 5. DUALKS.....	16
5.1. Introduction of DualKS.....	16
5.2. Classification of DualKS.....	16
5.3. Identification of the Discriminant Genes.....	17
5.4. Limitation of the Scoring Functions.....	18
5.4.1. Rescaling DualKS Score.....	18
5.4.2. Weight of DualKS Score.....	18
5.5. DualKS Software.....	19
CHAPTER 6. METHODOLOGY AND DATASET.....	20
6.1. Methods for the DualKS.....	21
6.2. Methods for the SVM.....	21
6.3 Methods for the <i>k</i> NN.....	22
6.4. Methods for the RF.....	23
CHAPTER 7. RESULTS.....	24
7.1. Results using the DualKS Classification.....	24
7.2. Results using the SVM Classification	28
7.3. Results using the <i>k</i> NN Classification.....	29
7.4. Results using the RF Classification	32
CHAPTER 8. MISCLASSIFICATION ASSESSMENT OF DUALKS.....	35
8.1 0.632 plus Bootstrap Method	35
CHAPTER 9. DISCUSSION	37

CHAPTER 10. CONCLUSION	38
REFERENCES.....	39
APPENDIX A. TRAINING THE ALGORITHMS.....	42
APPENDIX B. MISCLASSIFICATION ALGORITHM.....	47

LIST OF TABLES

<u>Table</u>	<u>Page</u>
1. Prediction using DualKS.....	24
2. Predicted class frequencies.....	25
3. P-value of the first nine genes.....	27
4. Performance of SVM kernels.....	28
5. Prediction of the phenotype using SVM.....	29
6. Determination of the optimal kernels of k NN.....	30
7. Finding the optimal k values with $k_{max}=24$ of k NN.....	30
8. Prediction using k NN.....	31
9. Prediction using RF.....	32
10. Out-of-bag error rate for each class.....	33
11. Bootstrap method.....	36

LIST OF FIGURES

<u>Figure</u>	<u>Page</u>
1. Linear separation of two classes -1 and +1 with SVM classifier.....	6
2. Plot of samples of upregulated genes.....	26
3. Plot of samples of downregulated genes	27
4. Finding the optimal k -value.....	31
5. Variables of importance.....	33
6. Plot of important genes per phenotype.....	34

CHAPTER 1. INTRODUCTION AND LITERATURE REVIEW

Gene expression is a series of step by which information from gene is used to unify gene products. Nowadays, the challenge is no longer obtaining the profiles of gene expressions but in somehow being able to explain the important results in order for society to gain some insights. Scientists are only able to perform genetic analyses not only on a few genes and cells in the human brains, but on thousand to billions of cells or genes that make the structure of our internal biological system by using DNA Microarray.

DNA Microarray can be used to measure the changes in genes expression's levels. Some techniques, such as microarrays, allow us to study genomes and their wide associations of gene expression with diseases, such as prostate cancer. This important change has been motivated biologically, as many diseases such as prostate cancer are believed to be associated with modest regulation in a set of related genes rather than a strong increase in a single gene (Subramanian et al., 2005).

Cancer cells exhibit a common phenotype of uncontrolled cell growth, but this phenotype may arise from many different combinations of mutation by inferring how cells evolve in individual tumors, a process called cancer progression (Park et al., 2009). We may then be able to identify important mutational events for different tumor types, which could potentially lead to new therapeutics and diagnostics (Kumar et al., 2012). It is possible to infer frequent progression pathways by using gene expression profiles to estimate "distances" between any kinds of tumors (Park et al., 2009).

In the United States, prostate cancer remains the principal malignancy in African American men and the leading cause of cancer-related death in this group (Ries et al., 2000). Cancer has now become the second leading cause of death in the U.S, and it accounts for 1 in 4

deaths (Jemal et al, 2009). Racial disparities in cancer mortality persist in the U.S, but however the survival has improved in virtually all ethnic groups (Ries et al., 2000). As such, survival after a cancer diagnosis still remains poorer among African Americans than White Americans (Clegg et al, 2002).

Prostate cancer mortality has been approximately twice higher among Black Americans than White Americans in recent decades (Bach et al., 2002), and current global comparisons confirm worse mortality outcomes in African American men (Delancey et al., 2008). Vital statistics from the United Kingdom briefly state that prostate cancer mortality rates among men born in West Africa and the West Indies are two to three times higher than overall rates in the worldwide population (Wild et al., 2006). The limitation of the data also confirm that there is a lower 5-year survival among African American than White men following a prostate cancer diagnosis (Coleman et al., 2008). In 2009, approximately 192,000 men from the USA were diagnosed with prostate cancer, with an estimate of 27,000 of those men dying from the same disease (Jemal et al, 2009).

This paper focuses on the classification of patients with prostate cancer using three different phenotypes: PSA, Systematic and NED. The classification was done using the Support Vector Machine, the k -Nearest Neighbour, Random Forest and the DualKS. The goal of the classification is to sort the 596 patients by the three levels of phenotype. The paper is organized as follows: Chapter 2 deals with the Support Vector Machine classification. Chapter 3 gives the k -Nearest-Neighbours classification. Chapter 4 talks about the DualKS classification, and chapter 5 discusses the Random Forest classification. A description of the methodology and dataset is given in Chapter 6 and the results in Chapter 7. Chapter 8 deals with the performance of the DualKS. A brief discussion and a conclusion are given in Chapter 9 and 10 respectively.

CHAPTER 2. SUPPORT VECTOR MACHINE CLASSIFICATION

2.1. Introduction of the SVM

The Support Vector Machine (SVM) is a biologically inspired method used to model information processing. It is a relatively young classification technique that was previously proposed by Vapnick (Xu et al., 2006) which has become more popular nowadays since its introduction in the late 1990s. During that time, SVM was largely unnoticed due to its non-relevance in the practical applications. Since then, SVM has been developed and used in many practical applications such as in bioinformatics, chemistry and other applications and fields.

SVM, a supervised machine learning technique, is shown to perform well in multiple areas of biological analysis including evaluating microarrays expression data (Terrence et al., 2006). SVM was the first mathematical models that does not assume any specific probability distribution but it tends to learn the distribution from the experimental data (Vojislav, 2001). The simplest and first introduced SVM model is the Maximum Margin classifier. This model is only suitable for data that are linearly separable without any overlapping samples.

2.2. Support Vector Machine

SVM is a supervised learning algorithm capable of solving linear and nonlinear binary classification problems (Stefan et al., 2006). In the binary classification, let's f defines a real valued function such as $f: X \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$ and the input $x = (x_1, x_2, \dots, x_n)^T$. For now, let's consider the case where $f(x)$ is a linear function of $x \in X$ and $x_i \in \mathbb{R}^n$ then:

$$\begin{aligned} f(x) &= \langle w \cdot x \rangle + b \\ &= \sum_{i=1}^n w_i x_i + b, \end{aligned}$$

where $(w, b) \in \mathbb{R}^n \rightarrow \mathbb{R}$ are the parameters and the decision rule is given by sign of $(f(x))$ with w being the weight that defines a direction that is perpendicular to the hyperplane, b is the bias term that moves the hyperplane parallel to itself and x is the support of the support machine.

In the binary classification, we then define $x_i \in \mathbb{R}^n$ as the input and $y_i \in \{-1,+1\}$ being the class label of x_i . In that case, the function margin that is defined as the margin measured by the function output of $f(x)$ implies that:

$$\langle w \cdot x^+ \rangle + b = +1$$

$$\langle w \cdot x^- \rangle + b = -1,$$

with x^+ being the positive point and x^- being the negative point. The positive example points (+1) should lie on or above the first supporting hyperplane. The negative example points (-1) should lie on or below the second supporting hyperplane.

2.3. Separation of the Hyperplanes

The goal of SVM classification is to separate examples by means of a maximal margin hyperplane (Nello and John, 2000). A training set, S , is defined to be a collection of training examples and it is denoted by:

$$S = ((x_1, y_1), \dots, (x_l, y_l)) \subseteq (X \times Y)^l,$$

where l is the number of examples. The goal of the algorithm is to maximize the distance between examples that are closest to the decision boundary. The margin of separation is related to the so called Vapnik-Chervonenkis dimension (VC dim) which measures how complex the learning machine is (Vapnick, 1998).

The Vapnik-Chervonenkis dimension, used in several bounds for the generalization error of a learner and known as the margin maximization is beneficial for the generalization ability of

the resulting classifier (Vapnik, 1995). Given a linearly separable training sample S , with $S = ((x_1, y_1), \dots, (x_l, y_l))$, the hyperplane (w, b) that solves the optimization problem

$$\text{minimise}_{w,b} \langle w \cdot w \rangle$$

$$\text{Subject to } y_i(\langle w \cdot x_i \rangle + b) \geq 1 \text{ for } \forall_i,$$

realizes the maximal margin hyperplane with a geometric margin $\gamma = \frac{1}{\|w\|_2}$ which is the minimal distance between two classes. The transforming of the optimization problem above into the corresponding dual problem give us the primal Lagrangian:

$$L(w, b, \alpha) = \frac{1}{2} \langle w \cdot w \rangle - \sum_{i=1}^l \alpha_i [y_i(\langle w_i \cdot x_i \rangle + b) - 1].$$

This dual is found by differentiation with respect to the weight w and the bias b , and it is only dependable on the Lagrange multipliers α_i ,

$$\sum_{i=1}^l \alpha_i - \frac{1}{2} \sum_{i,j=1}^l y_i y_j \alpha_i \alpha_j \langle x_i \cdot x_j \rangle,$$

where $\alpha_i \geq 0 \forall_i$ and the one linear constraint $\sum_{i=1}^l y_i \alpha_i = 0$, and l is the number of training and y_i is the correct output for the i^{th} training examples. Once the Lagrange multipliers are determined, the normal weight vector w and the bias b can be derived from the Lagrange multipliers (John, 1998) where:

$$w = \sum_{i=1}^l y_i \alpha_i x_i$$

$$b = w_k \cdot x_k - y_k, \text{ for some } \alpha_k \geq 0 \forall_k.$$

Therefore, the amount of computation required to evaluate a linear SVMs is constant in the number of non-zero support vectors. The construction of the SVM classifier is done by minimizing the norm of the weight vector w under the constraint that the training patterns of each class reside on opposite sides of the separating surface (Stefan et al., 2006). An example is illustrated below in Figure 1.

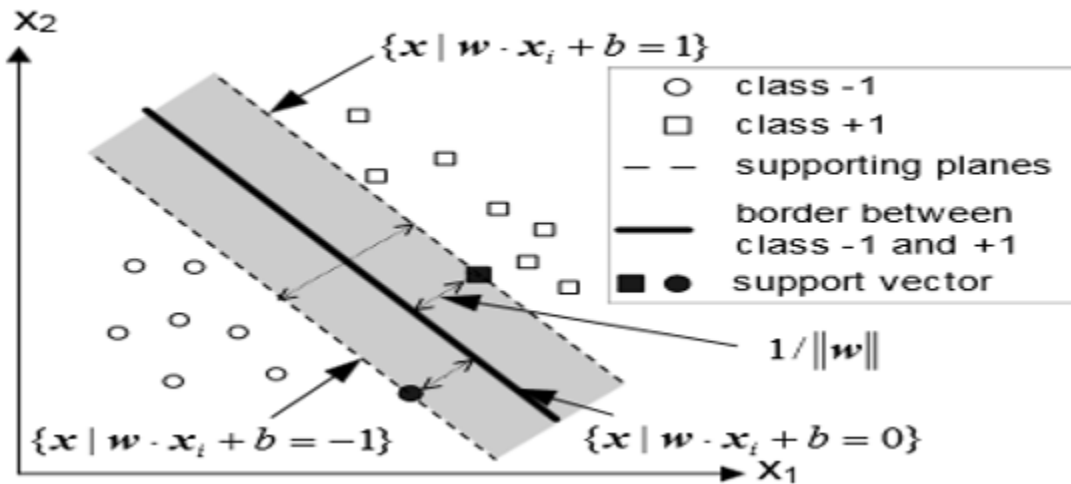


Figure 1. Linear separation of two classes -1 and +1 with SVM classifier

Figure 1 is the easiest classification problem in which the data are linearly separable without any overlapping samples (Chang and Lin, 2001). The maximal distance is highlighted. Lagrange multipliers are calculated from the weight vector w . The points that are not on the support vector have no influence at all.

2.4. Optimal Separation of the SVM

The first concept of an optimal hyperplane regarding the support vector machine was first proposed by Vapnick (Karatzoglou and Meyer, 2006). A decision surface for a binary classification is optimal if the separation of the hyperplane is done without an error but it also has to maximize the distance between the hyperplane and the decision surface. Because the decision surface $f(x)$ has a maximum margin's distance, then we have two supporting hyperplanes equidistant satisfying:

$$\langle w \cdot x \rangle + b = +1 \text{ for the positives}$$

$$\langle w \cdot x^- \rangle + b = -1 \text{ for the negatives,}$$

with w being the weight vector of the data points.

So far, we have assumed the training data is linearly separable. However, in real world applications studies, many datasets are not linearly separable and sometimes there might not be a hyperplane that splits the positive examples from negative examples in the binary cases. In the formulation above, the non-separable case would correspond to an infinite solution (John, 1998). However, in 1995, Cortes and Vapnick suggested a modification to the original optimization statement that will penalize the failure of an example to reach the correct margin. The proposed modification is done by introducing the slack variable that is defined to be the “Soft Margin”:

$$\text{minimize}_{\varepsilon, w, b} \langle w \cdot w \rangle + C \sum_{i=1}^l \varepsilon_i$$

$$\text{Subject to } y_i(\langle w \cdot x_i \rangle + b) \geq 1 - \varepsilon_i \text{ for } i=1, \dots, l \text{ and } \varepsilon_i \geq 0, i = 1, \dots, l \text{ and } C > 0.$$

Any point x_i can satisfy the constraint even if it is located on the wrong side of the decision surface as long as the slack variable (ε_i) is large enough. C is cost. The introduction of the slack variables will account for any examples that were wrongly misclassified. If the slack variable $\varepsilon_i \rightarrow \infty$ then the classification of SVM becomes the nonlinear. In that case, the algorithm could be generalized to a nonlinear classification by mapping the input data into a high-dimensional feature space by the chosen nonlinear mapping function ϕ (Stefan et al., 2006). Therefore, SVM can be even further generalized to nonlinear classifiers (Vapnik, 1982). The output of a non-linear SVM is explicitly computed from the Lagrange multipliers:

$$\sum_{i=1}^l \alpha_i y_i K(x, z) + b,$$

$$K(x, z) = \langle \phi(x), \phi(z) \rangle = \langle \phi(z), \phi(x) \rangle = K(z, x),$$

where K a kernel function, such that for all $x, z \in X$. The functions associated with the output of the nonlinear SVM transformations are called *kernel functions* and the process of moving these function move from linear to a nonlinear Support Vector Machines is referred to as: “*kernel trick*”.

2.5. SVM Software

The first implementation of SVM using the software R was developed by “R Development Core Team in 2005” and was introduced in the package e1071 (Dimitriadou et al., 2005). The SVM function in the package e1071 provides a rigid interface to the library svm (libsvm) along with visualization and parameter tuning methods (Alexandros et al., 2006). The libsvm is faster and easier to use for most popular SVM formulations featuring one of the following:

- *one* class classification: the idea behind this model is that it tries to find the support of a distribution that is resulted by allowing for outlier/novelty detection (David, 2014).

2.6. Brief Introduction of SVM to the Multiclass

Previously, we talked about the Support Vector Machine only supporting the binary classification. Meaning, we have presented and discussed only two classes since the introduction of this paper. However, as we all know, the real world problems deal generally with classifying objects in more than two classes. The same idea will still work as in the binary classification with some minor modifications.

2.6.1. One-versus-The Rest Classification

The binary classification of SVM can be extended into multiclass classifications. The most popular techniques for multiclass classification, using the binary Support Vector Machine, is referred as “*The-one-versus-The Rest classification*”, also known as “*The- one-versus-all*”. To illustrate this, let’s consider the training set:

$$S = ((x_1, y_1), \dots, (x_l, y_l)) \subset \mathbb{R}^n \times \{1, 2, 3, \dots, F\},$$

where the label y_i for each observation can take on any value *in* $\{1, 2, 3, \dots, F\}$ with F being the number of classes greater than two. In “*one-versus-all*”, the construction of F binary support

vector is based on decision surfaces (k^1, k^2, \dots, k^M) . Each individual decision surface is trained to separate one class from the rest. The classification of an unknown sample is done using the voting scheme that is based on the largest value of the F decision surfaces for this unknown sample. As a result, this unknown sample is assigned to the class that return the largest value of the decision surface.

CHAPTER 3. *k*-NEAREST NEIGHBOURS

3.1. Introduction of the *k*NN

The *k*-Nearest Neighbours machine, one of the oldest machine algorithm, is a nonparametric classification method which does not rely on any assumption concerning the structure of the density function and is one of the most commonly used methods for pattern recognition (Moreno et al., 2002). Previously applied in a variety of cases (Khan et al., 2002), *k*NN has been used in statistical estimation and pattern recognition since in the beginning of the 1970s. A *k*-Nearest Neighbours (*k*NN) classifier (Belur, 1991) is a typical example of the latter category. As a lazy learning, *k*NN's algorithm is instance-based and used in many applications for statistical pattern recognition, data mining, image processing and many others (Quansheng and Lei, 2009)

3.2. Distance or Similarity of *k*NN

The behavior of *k*-Nearest Neighbours depends mainly on the definition of similarity or distance. There are many ways to define the distance between two points p_i and q_i in a multidimensional space. One such way is the L^n Norm distance that is also defined as:

$$L^n \text{ Norm} = \sqrt[n]{\sum_{i=1}^{dim} |p_i - q_i|^n}.$$

The Manhattan distance is a special case of the L^n norm distance when n is 1 and when n is equal to 2, the L^n norm becomes the Euclidean distance.

If all the features are numeric, the Manhattan and Euclidean distance could be used. The Manhattan Distance is a simple similarity measure compared to the Euclidean and square-Euclidean distance measure, which takes the summation of the absolute difference among individual elements of the vector and it is defined as:

$$\text{Manhattan distance} = \sum_{i=1}^{dim} |p_i - q_i|^1.$$

The Euclidean distance on the other hand is defined as the length of the line segment between p_i and q_i in a multidimensional space

$$\text{Euclidean distance} = \sqrt{\sum_{i=1}^{\text{dim}} |p_i - q_i|^{n=2}}.$$

However if the features are not numeric, then the Hamming distance should be used. The Hamming distance between two vector points p_i and q_i in a multidimensional space is defined as the number of places where the two elements of a vector differ. The hamming distance is also used for symbolic feature and it is defined as:

$$\text{Hamming distance} = \sum_{i=1}^{\text{dim}} |p_i - q_i|^1.$$

3.3. Classification of the k NN Algorithm

One important aspect regarding the Manhattan, Euclidean and L^n norm distances above is the fact that these models are only valid for continuous variables. In order to classify an unknown dataset point to a new class, we must first calculate either the Euclidean or Manhattan distance for numeric feature, dataset, or the Hamming distance for categorical or symbolic feature but now the question still remains: How to choose the best value for the parameter k ? A case is classified by a majority vote of its neighbours. Consequently, if we have equal closest samples from each class, then the unknown sample will be classified randomly.

A training dataset with accurate classification labels should be known at the beginning of the algorithm. Then for a query data M_i , whose label is not known and is presented by a vector in the feature space, the k NN will calculate the similarities between the unknown label and every point in the training data set (Quansheng and Lei, 2009). After sorting, the distances calculation results in a decision of the given class labeling of the test point m_i can be made according to the label of the k nearest points in the examples (Quansheng and Lei, 2009).

3.4. Estimating the number of neighbours k of the k NN

The optimal choice of the parameter k can be estimated using the bootstrap method and also by cross-validation technique. The quality of the training examples affects directly the prediction rate of the classification. At the same time, the choice of parameter k is also very important, for different k could result in different classification labels (Quansheng and Lei, 2009). The aim is to provide an optimal estimator of the parameter k that is less variable. It has also been said the estimate of the parameter k could be chosen by taking the square root of the sample which also brings completely different problems. Therefore for a small value of the parameter of k , the k NN algorithm becomes more sensitive to noise points and lead to a small bias. For a large value of the parameter k , the k NN algorithm includes some points from different classes.

3.5. k NN Software

The implementation of the k NN package using the software R was developed by Klaus Schliep and Klaus Hechenbichler in 2004. The link of the package is available on: <http://cran.r-project.org/web/packages/kknn/kknn.pdf>. The package contains many functions. The package “*kknn*” is used for k NN classification, regression and clustering. The function “*kknn*” performs the k NN classification of the training set. For each individual row of the testing set, the k nearest training set vectors (according to Minkowski distance) are calculated, and the classification is done by using the maximum of summed kernel densities. By using the Minkowski distance, ordinal and continuous variables could also be predicted.

CHAPTER 4. RANDOM FOREST

4.1. Introduction of the Random Forest

The Random Forest (RF) is a popular machine-learning algorithm that has recently been successfully used when dealing with various biological prediction problems (Zhang et al., 2012). RF was developed by Loe Breiman and Adele Cutler (Breiman, 2001). Random Forest is a classifier consisting of an ensemble of classification and regression tree-structured classifiers (Jian-Hua et al., 2013). All trees in the forest are unpruned. The algorithm has two powerful advantages: bagging and random feature selection (Jian-Hua et al., 2013).

4.2. Classification of RF

The techniques of the RF are based on a combination of a set of decision trees. There are three parameters that need to be defined in the Random Forest algorithm: N_{tree} , which is defined as the number of bootstrap samples for the original data; M_{try} , is the number of different predictors; and *node size* is the minimal size of the terminal nodes of the trees. Each tree is constructed using the following algorithm (Breiman, 2001):

- We define the number of bootstrap samples cases N_{tree} to be N , and the number of variables or predictors in the classifier M_{try} , to be M .
- We will let m be the number of input variables to be used and m should be much less than M
- Decide a training set for this tree by choosing n_i times with replacement from all N available training cases containing two-third of the data. The one-third will use to estimate the error of the tree, by predicting their classes. The elements not present in n are referred to as “out-of-bag” data (*oob*) for that bootstrap sample.

- For each individual node of the tree, we randomly choose m variables then the best split based on these m variables in the training set is calculated.
- Individual tree is fully grown and not pruned.

4.3. Estimating the Error Rate of RF

In random forests, there is no point for a cross-validation technique to get an unbiased estimate prediction rate (Breiman, 2001). The process of estimation is done internally, during the run, as follows (Breiman, 2001):

1. Each individual tree is constructed using a different bootstrap sample from the training N and about one-third of the data are left out of the bootstrap and they would not be used in the construction of the k^{th} tree.
2. For prediction, a new sample is pushed down the tree and is assigned the label of the training sample in the terminal node it ends up in. This procedure is done all over all trees in the ensemble, and the average vote of all trees is reported as random forest prediction. For the k^{th} element (y_i) of the training set, all the trees are taking into account in which the i^{th} element is out-of-bag (Simone et al., 2011). On the basis of the random trees, an aggregated prediction is developed on average and it is defined as : g_{oob} . And each element of N is out-of-bag in one-third of N_{tree} iterations (Simone et al., 2011). The out-of-bag estimate is computed as the following:

$$Error_{oob} = \left(\frac{1}{N_{tree}}\right) \sum_{i=1}^{N_{tree}} [y_i - g_{oob}]^2.$$

4.4. RF Software

The package “Random Forest” was implemented by Breiman’s random forest algorithm (based on Breiman and Cutler’s original FORTRAN code) for classification and regression. It can also be used in unsupervised mode for assessing proximities among data points. The link of

the package is available on: <http://statwww.berkeley.edu/users/breiman/RandomForests>. In the package, we have a formula interface and a prediction of the Random Forest that could be specified using the structure of a matrix or a data frame via an x argument. As long as the response variable is a factor, then the Random Forest will perform the classification. If the response variable is a continuous one, the Random Forest will perform the regression.

CHAPTER 5. DUALKS

5.1. Introduction of DuaKS

Another kind of nonparametric classification is the Dual Kolmogorov –Smirnov classification also called the DualKS. The DualKS was developed by Dr. Yarong Yang and her collaborators: Eric J. Kort, Ebarhim Nader, Zhang Zhongfa and Bin T in 2014 (Kort and Yang, 2014). This test is based on the KS statistic that measure the greatest distance between the empirical distribution function of a univariate data and the comparison step function of the second dataset (Feigelson and G.Joseph, 2012).

5.2. Classification of DualKS

In order for a classification to occur using the DualKS algorithm, a gene signature which matches the gene expression must primarily be found (Kort et al., 2010). Then, the sample from the gene signature will be classified based on their unique instance of representation values z with $(z_1, z_2, z_3, \dots, z_n)$ defined as the instance representation values of the upregulated gene signatures and the downgraded gene signatures are defined to be the instances representation values of $(z_{n+1}, z_{n+2}, z_{n+3}, \dots, z_{2n})$. The sample instance representation value of z is then denoted by $((z_1, z_2, z_3, \dots, z_n); (z_{n+1}, z_{n+2}, z_{n+3}, \dots, z_{2n}))$. The enrichment score of each specific class is then calculated and the sample is will belong to the class whose signature achieves the highest score (Kort et al., 2008). The n upregulated and downregulated genes expression are defined in decreasing and increasing order respectively such as:

$$a'_{il} = \begin{cases} \frac{n}{n_l} & \text{if gene } i \in \text{up regulated genes} \\ \frac{-n}{n-n_l} & \text{Otherwise} \end{cases},$$

where n_l is the number of upregulated genes in decreasing order in the signature of class l .

$$b'_{il} = \begin{cases} \frac{n}{n_l} & \text{if gene } i \in \text{down regulated genes} \\ \frac{-n}{n-n_l} & \text{Otherwise} \end{cases},$$

where n_l is the number of downregulated genes in increasing order in the signature of class l , with $n=t * K$, where t is found empirically .

5.3. Identification of the Discriminant Genes

The package DualKS is applied to perform the discriminant analysis of the training set and classification analysis. Therefore, given a $G \times N$ expression matrix X for G genes and N being the total sample size and a classification vector $y_1 \dots y_N$ with y_j being the classification for the sample j , then for each individual gene we sort its N expressions in a decreasing order to identify the degree of increasing cellular component in each of the individual genes (Kort et al., 2010). After, for each of the sample N ordered from the highest to lowest based on their expression values in each row of individual genes (Kort et al., 2010). The scoring function is:

$$u_{ik} = \max \sum_{j=1}^N a_{ikj},$$

$$\text{where } a_{ijl} = \begin{cases} \frac{N}{N_k} & \text{if sample } j \text{ is of class } k \\ \frac{-N}{N-N_k} & \text{Otherwise,} \end{cases}$$

and j is the index of the ordered list of the N expressions values for gene i , and k is the class among the K unique classes in Y . N_k is defined to be the number of samples of class k in the complete set of N samples. By sorting the genes based on decreasing u_{ik} for a given class, we are capable of detecting the most upwardly biased in a given class in terms of their ordered expression levels (Kort et al., 2010). On the other hand, by sorting the genes based on decreasing order d_{ik} for a given class, we could target those genes that are downwardly biased in a specific class (Kort et al., 2010).

$$d_{ik} = \max \sum_{j=1}^N b_{ikj},$$

$$\text{where } b_{ijl} = \begin{cases} \frac{N}{N_k} & \text{if sample } j \text{ is of class } k \\ \frac{-N}{N-N_k} & \text{Otherwise} \end{cases}.$$

5.4. Limitation of the Scoring Functions

Scoring functions, a class of computational methods, are widely applied in computational statistics and learning methodology. Even though scoring functions are important in statistical principal, it has some limitations in its application of the DualKS algorithm (Kort et al., 2010). One of those limitations is when the scoring function gets an elevated value for both certain samples of a given class ordered early and late in the ordered lists which lead to an elevated value of the scoring function u and an elevated value for the scoring function d for the late samples (Kort et al., 2010). The two scoring functions are defined to be:

$$a'_i = \max \sum_{i=1}^n a'_{il} \text{ for Upregulated genes}$$

$$d'_i = \max \sum_{i=1}^{2n} b'_{il} \text{ for Downregulated genes.}$$

5.4.1. Rescaling DualKS Score

The enrichment (E) for each class l is defined as:

$$E_l = a'_i + b'_i$$

$$E_l = \frac{a'_i + b'_i}{r_l}, \text{ for the rescaled case,}$$

where r_l is the scaling factor of that class.

5.4.2. Weighted DualKS Score

The genes are weighted according to each gene's average expression in a given class.

The weight of gene i in class l is defined as:

$$W_{il} = -\log \frac{\bar{R}_{il}}{G},$$

where G is the total numbers of genes and \bar{R}_{il} is the numerical rank of gene i 's average expression in class l among the G genes' average values in class l .

5.5. DualKS Software

The implementation of the DualKS package using the software R was developed by Dr. Yarong Yang and Eric in 2008 (Kort and Yarong, 2008) and used for classification and discrimination analysis. The link of the package is available on: <http://www.bioconductor.org/packages/release/bioc/manuals/dualKS/man/dualKS.pdf>. The package contains many functions. In the package, we have a formula interface and a prediction of the DualKS that could be specified using the structure of a matrix or a data frame containing the gene expression. As long as the response variable is a factor, then the DualKS will perform the classification. One of the algorithm's features is the function "*type*" which must be as one of "*up*", "*down*", or "*both*" indicating whether one wants to analyze and classify based on upregulated or downregulated genes, or both. The package contains a function "*dksTrain*" that returns on object of class "*DKSGeneScore*" to hold the analysis results for classifier extraction and classification.

CHAPTER 6. METHODOLOGY AND DATASET

The cancer dataset used for this experiment is downloaded from Gene Expression Omnibus (GEO) website (Edgar et al., 2002). GES10645 was used for this research and it is a microarray genes expressions consisting of 596 men patients with prostate cancer. After treatment, the 596 men patients were classified into one class of phenotype with three different levels: Systematic, Prostate-Specific Antigen (PSA), and No Evidence of Disease (NED).

A systematic cancer is an initial cancer, called primitive, which developed metastases that migrated to different parts of the body. NED is the term most often used to describe a patient's status after treatment. Unfortunately, we can never truly say a patient is cured of cancer - the best we can do is to say that we find no evidence of disease. Prostate-Specific Antigen, or PSA, is a protein produced by cells of the prostate gland. Of the 596 men patients, 200 of them were patients with systemic disease progression, 201 patients had PSA recurrence and 195 patients had no evidence of disease groups.

Four widely methods were used for prediction, aiming to obtain high quality of genes in each class. In recent years, many methods have been provided to perform the gene expression analysis. The measurements for these experiments could provide genes expression's levels. For classification, we need two types of datasets: the training dataset and the testing datasets. Besides dividing the datasets into half or using a cross-validation technique to split the dataset, the whole dataset was used as a training set and testing set for classification for the four different algorithms. The training set was prepared by selecting every other points from the genes expressions of 596 men patients other than the phenotype.

6.1. Methods for the DualKS

The “DualKS” package is applied to the KS algorithm in different ways: First, it can be used to perform the discriminant analysis then second, it is used for the classification analysis. The first step is to rank each gene based on how biased its expression is in each class. We then have the option on focusing on the upregulated or downregulated genes in each class by using the function “*dksTrain*”.

Later, we extracted a classifier of five genes per class from the training data by using the function “*dksSelectGenes*”. We then apply this classifier to a testing data by running it against the training set to check for the internal consistency using the function “*dksClassify*”. After the classification, we then used the function “*summary*” by defining the actual classes of samples. By doing so, the percent correspondence rate will be calculated and displayed along with the summary and keep the correspondence predicted sample. After trying to classify the training set, we computed the classification rate of the DualKS algorithm. In order to do this, we had to slightly modify the function “*dksPerm*” of the package DualKS.

6.2. Methods for the SVM

In order to test the SVM algorithm, the training was done using the function “*svm*” in R. This function also provided the choice of many kernel functions. Since the kernel is the most important feature of the algorithm, then we decided to get a generic function which tunes hyper-parameters of statistical methods using a grid search over supplied parameter ranges to get the classification rate of each kernel.

One of the features of the function “*svm*” is the ability to define the margin failures or cost parameter. Different cost values were tuned in because determining the best cost C correctly is a vital step for best practice in the use of SVM, as structural risk minimization. By

running the function “*svm*” without defining any parameters such as the cost and distribution of the kernel, the “*svm*” outputs are the number of support vectors, the support vector type, the number of classes and their respective levels, the estimate of the cost and gamma and the type of kernel that best fit the dataset. These outputs defined the classifier at the earlier stage of the analysis. After determining the optimal kernels over a given set of kernels and the best cost, it was imperative to estimate the prediction of that kernel. After, we then computed its classification rate.

6.3. Methods for the *k*NN

The training of the prostate cancer was done using the function “*knn*” in R. *k*NN has many useful characteristics, one of which being its non-sensitivity to outliers making it resilient to any errors in the classification dataset. The *knn* function is housed in the *class* package and it neighbours classification for test set from training set. For each row of the test set, the *k* nearest training set vectors are found, and the classification is decided by majority vote, with ties broken at random. If there are ties for the k^{th} nearest vectors, all candidates are included in the vote.

This function also provided the choice of many kernels functions. When using *k*NN, it is very necessary to find the optimal kernel over different sets of kernels by using the function “*tune.kknn*”. As one of the main features of the algorithm, it is really important to determine the best value of *k*. We then trained the data by using the function “*train.kknn*” over different set of kernels and by letting the maximum *k* to be the square root of the total sample size and in our case $k_{max}=24$. Training the dataset in such a way, does not just only gives us the optimal kernel but also provides the best value for *k*. After that, we plot a graphical optimal *k* value by using the bootstrapping method of the function “*tune.knn*” and *k* ranging from 1 to 24. We, then computed the classification rate using the *k*NN algorithm.

6.4. Methods for the RF

The Random Forest algorithm is a very unique machine learning of its genre. It is very simple and it does not need any cross-validation technique to determine its respective parameters since it is done internally. In order to test for the classification of the prostate cancer data using the RF algorithm, the package “*Random Forest*” in R was used which had many options. The function random forest is unique in its genre because it has the option of selecting the variables that are mostly important. The two main parameters in this packages are the number of input variables randomly chosen at each split “*mtry*” and the number of trees in the random forest “*ntree*”. After, we then computed its prediction rate.

CHAPTER 7. RESULTS

7.1. Results using the DualKS Classification

The DualKS is an algorithm that performs discriminant analysis based on the tissue to gene set enrichments that are used for the classification analysis. The most important step for using this algorithm is to rank each of the 596 patients and 53 genes based on its biasness expression in each of the three classes of phenotype. Table 1 summarizes the percent prediction rate calculated for the scoring options of genes.

Table 1. Prediction using DualKS

Scoring options of gene	Prediction rate
Upregulated	40.10 percent
Downregulated	39.10 percent
Both	40.10 percent

In Table 1, the prediction rate for the upregulated genes is 40.10 percent. DualKS successfully predicted the up regulated genes 40.10 percent of patients, whereas, it successfully predicted the downregulated 39.10 percent of the patients.

Table 2 summarizes the frequency of genes that were displayed from the three classes of phenotype. In the scoring option of the upregulated genes, 186 men patients were classified having no evidence of disease (NED), whereas 231 patients still had the protein produced by cells of the prostate glands (PSA) and 179 patients had the presence of the initial cancer cells. Furthermore, in the scoring option of the downregulated genes, 245 patients were classified in the phenotype of patients presenting no evidence of disease, 168 patients with proteins produced by cells of the prostate gland and 183 patients had the presence of cancer in their bodies. In the scoring option of upregulated genes, the patients with the proteins produced by cells of the

prostate gland account for 63.3 percent of the total patients whereas in the downregulated option, they only account for 39.3 percent of the total patients.

Table 2. Predicted class frequencies

Class of Genes	Scoring Option of Genes		
	Upregulated	Downregulated	Without an option
NED	186	245	186
PSA	231	168	231
SYSTEMATIC	179	183	179

As we have seen from Table 1, the classification for any of the option was about 40 percent; which was low for any classification algorithm. In order to gain insights, the prediction object was plotted to visualize how the samples of each phenotype class compared to the other samples in terms of the KS scores for the individual signature. To visualize this distinction, a separate panel is created for each signature.

From Figures 2 and 3, a color bar is plotted below each sample that indicates its predicted and actual class for ready identification of outliers and misclassified samples. The data is plotted by using the three scoring options and sorted by each class signature so that the relationship between the upregulated and downregulated in each class could be observed. The plot does not tell us the reason our prediction rate is low. The first, second, and third panel in Figure 2 correspond, respectively, to the signatures of the patients with no evidence of disease, initial cancer patients and the patients with the protein produced by cells of the prostate gland. In Figure 2, the first panel shows the sorted patients according to the upregulated score. The patients with no evidence of disease (red bar) are clustered to the left with almost the same

scores as the patients with the initial prostate cancer and the patients that had protein produced by cells of the prostate gland. As we move to the right, the red line of the first panel decreases and the lines of the other signatures (blue line=Systematic signature score, green line=PSA signature score) increase following no discernable pattern. The other two panels in the figure behave in the same manner with the systematic signature and PSA signature sorted in a decreasing manner.

In order to correct this problem of low prediction rate, we slightly looked at three different methods: scaling the score of each of the signature so that they fall between 0 and 1, ranking the signatures by weighting the genes, and by converting the samples using a ratio. None of these three methods did improve our classification rate.

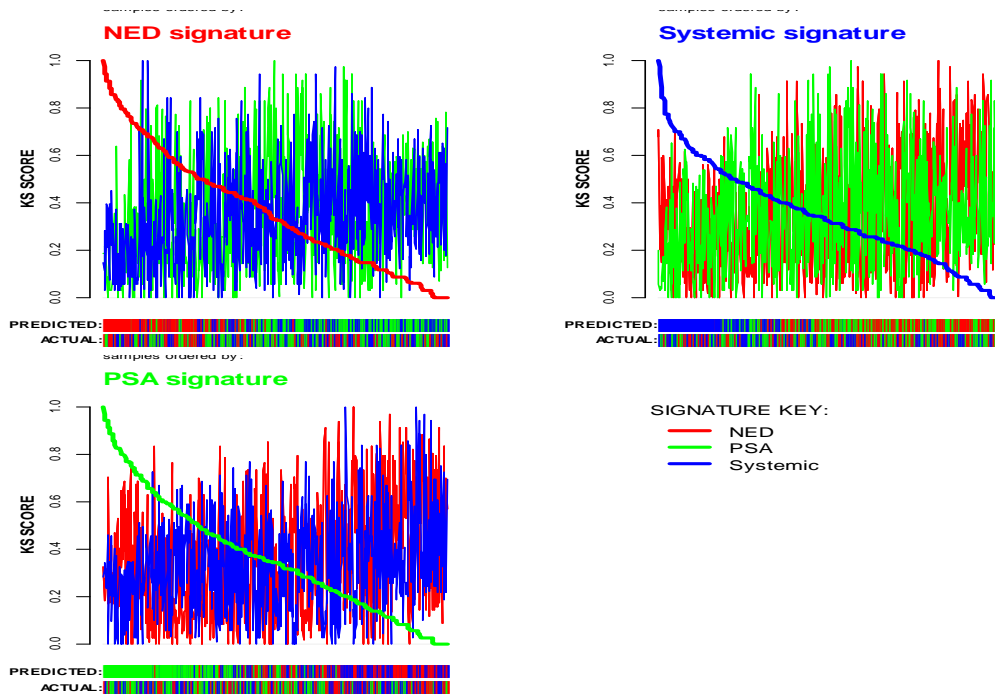


Figure 2. Plot of samples of upregulated genes

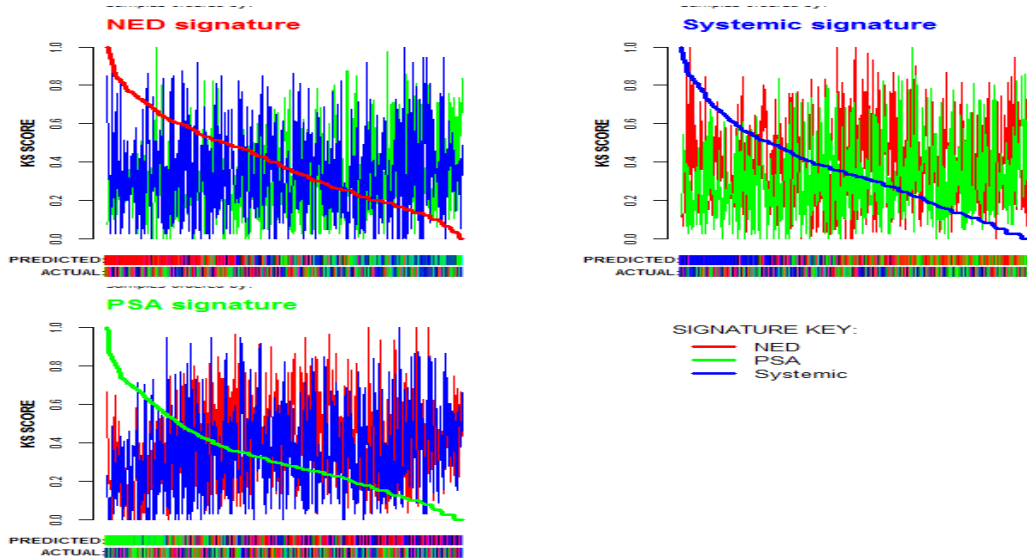


Figure 3. Plot of samples of downregulated genes

The best prediction rate for this dataset is approximately 40 percent, so it is important to localize which signature has the maximum score. The distribution of KS scores using this package follows a gamma distribution with shape =3.2041 and rate=0.11935. Both shape and rate are parameters of the gamma distribution, and they depend on the signature size and the total number of genes. The first nine estimated p-values for our predicted classes are given in the Table 3. Table 3 summarizes the random variation that resulted from the predicted classes. These high p-values were expected since the number of genes from each phenotype was low.

Table 3. P-values of the first nine genes

predicted	p-value
041	0.2909539
058	0.8881518
067	0.4082878
077	0.3005362
085	0.6032476
017	0.5019659
024	0.7794827
032	0.6378167
041	0.6032476

7.2. Results using the SVM Classification

The classification using the Support Vector Machine requires the determination of the optimal kernel over the given set of kernels and its optimal cost. The effect of the cost C on the classification error was done by permitting C to range from 10 to 1000. A 10 fold cross-validation technique was done on our training set using five different kernels including the polynomial kernel, the linear kernel, the radial basic function kernel, the gamma kernel, and the sigmoid kernel in order to determine the optimal kernel.

The 10 fold cross-validation involves for every set of 11 samples of the dataset, use 10 fold for the training and the remaining one will be used for testing. This technique is performed until all of the examples in the dataset are used for both training and testing. The results are shown in the Table 4.

Table 4. Performance of SVM kernels

Kernels	Cost	Sampling method	Best Performance	Best Value
Polynomial	100	10 fold cross validation	0.683	
Radial basic Function	100	10 fold cross validation	0.692	gamma=0.012
Gamma	100	10 fold cross validation	0.66	0.01
Linear	100	10 fold cross validation	0.67	
Sigmoid	100	10 fold cross validation	0.68	

After tuning each kernel to the training set using the 10 fold cross-validation technique, the radial basic function kernel was selected because of its high accuracy of 69.2 percent and a cost of 100. Model selection is also an important aspect in the Support Vector Machine. Its

success depends on the tuning of several parameters such as cost and kernels that affect the generalization error.

After using the grid-search method in k -fold cross-validation to select the best kernel and cost, we applied this parameter set to the training dataset and then obtained a classifier. Then, the obtained classifier was used to classify from the training by allocating the testing dataset in order to get the generalization accuracy. The prediction model from the radial basic function kernel is given the Table 5.

Table 5. Prediction of the phenotype using SVM

NED	PSA	Systematic
195	201	200

After inputting 596 patients into the classifier, the number of supports set was 588 and 195 patients were predicted to have no evidence of disease (NED), whereas 201 patients had protein produced by cells of the prostate gland (PSA) and 200 patients had the presence of the initial cancer in their cells (Systematic). The accuracy rate of the prediction using SVM is 100 percent which is statistically significant. It perfectly classified the genome prostate cancer dataset.

7.3. Results using the k NN Classification

The aim of the k NN classification algorithm is to predict the testing sample's category according to the k training samples which are the nearest to the testing sample. As it is with the SVM, the determination of the best kernels is one of the main features of the algorithm in addition to finding the optimal k values. In Table 6, we have a list of different kernels and their respective optimal minimal misclassification. This classification problem seems to be too simple

to produce significant differences between the different kernels and their corresponding kernel values.

The optimal kernel is determined to be the rectangular kernels which is defined as the standard unweighted of the k NN since it has the smallest minimal misclassification. Its misclassification rate is 66.9 percent.

Table 6. Determination of the optimal kernels of k NN

optimal kernels	Best k values	Minimal misclassification
rectangular	6	0.669
triangular	10	0.672
Gaussian	4	0.6812
rank	6	0.6812
optimal	12	0.676
epanechnikov	10	0.673

Table 7. Finding the best optimal k values with $k_{max}=24$ for k NN

k values	Best k values	Minimal misclassification
5	3	0.676
6	6	0.669
7	6	0.669
8	6	0.669
9	6	0.669
10	6	0.669
11	6	0.669
12	6	0.669
13	6	0.669
24	6	0.669

A small kernel value and a small misclassification seems is the best choice. As the k values increase, the misclassification increases by a small margin of error. No matter which k value is used in the rectangular kernel, the results for a higher k reach the optimal results of 6. To avoid problems with the choice of the parameter k values which is a key feature of the k NN,

a bootstrap sampling method was used in order to check if the optimal k value found from the Table 7 was exactly the correct k value.

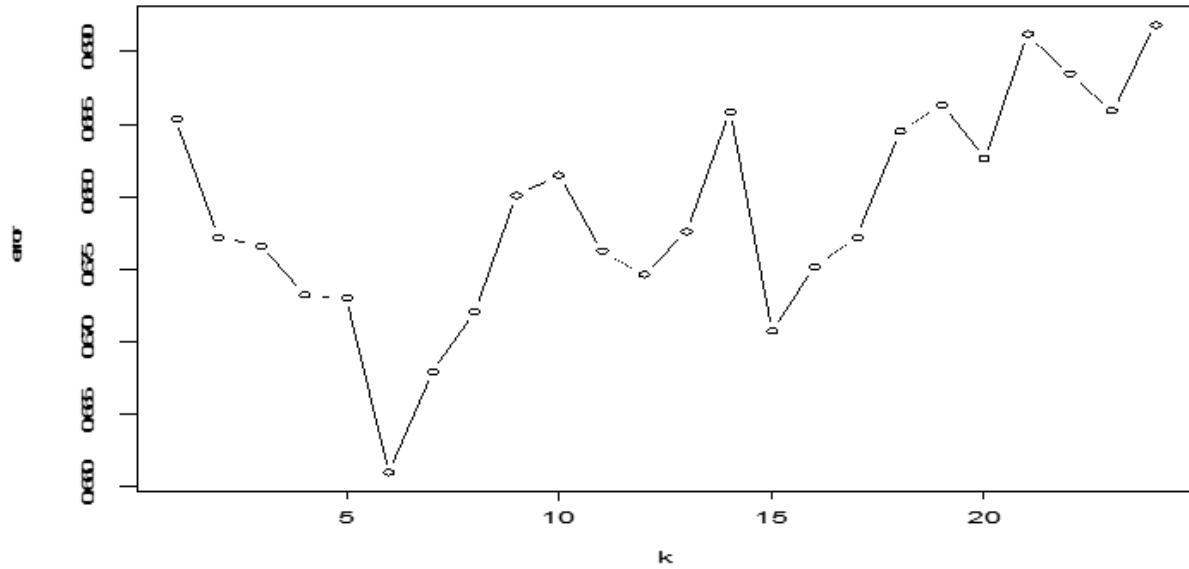


Figure 4. Finding the optimal k -value

We choose $k=6$ as the best number of neighbors for k NN in this sample since it yields the smallest test error rates. The classification result based on $k = 6$ is shown in the scatter plot of Figure 4. After selecting the best value of k , one could make prediction based on this algorithm. In our case, the prediction rate is 54.4 percent.

Table 8. Prediction using k NN

NED	PSA	SYSTEMATIC
202	171	223

In Table 8, using the k NN algorithm, 202 men patients were classified to have no evidence of disease (NED), whereas 171 patients had protein produced by cells of the prostate gland (PSA) and 223 patients had the presence of the initial cancer in their cells (Systematic).

7.4. Results using the RF Classification

One of the key features of the Random Forest is the idea that there is no need to do a cross-validation to get an unbiased estimator of the testing set since it is estimated internally. As a result, after running the RF package, 500 trees were constructed from three phenotypes of the patient conditions from seven parameters with their class errors. The error rate for the algorithm with the out-of-bag data is 69.3 percent. The result is displayed in the Table 9.

Table 9. Prediction using RF

Random Forest	Record
Number of Tree	500
Number of variables at each split	7
OOB prediction accuracy	69.3percent
Type of Random Forest	Classification

The score of the out-of-bag error rate is high. This score means that the cluster of the random forest calculated with the classifier did score 69.3 percent by using the original dataset as the training set and testing set. It is important to note that RF does not just waste those “out-of-bag” observations, it uses them to see how well each tree performs on unseen data. It is used as a testing set to determine the performance on the model. The Random Forest produces the three different classes of phenotype when asked too. The conditions of each patient are decided by the type of phenotype he or she processes. Table 10 indicates the out-of- bag error rate for each class of phenotype.

Table 10. Out-of-bag error rate for each class

ntree	oob	NED	PSA	SYSTEMATIC
100:	71.98%	72.31%	72.14%	71.50%
200:	71.31%	72.82%	72.14%	69.00%
300:	69.46%	71.28%	71.64%	65.50%
400:	69.13%	67.69%	72.14%	67.50%
500:	69.63%	70.26%	71.14%	67.50%

Another feature of the RF is the importance that it plays with the variable. Using 500 trees, the prediction rate is 68.3 percent. We then wanted to see how the important variables affected the out-of-bag error. The variable importance by the Random Forest can be useful in order to reduce the number of variable. There are two types of importance measures in Figure 5. The accuracy one tests to see how worse the model performs without each variable, so a high decrease in accuracy would be expected for very predictive variables. The Gini one digs into the mathematics behind decision trees, but essentially measures how pure the nodes are at the end of the tree. Again it tests to see the result if each variable is taken out and a high score means the variable was important.

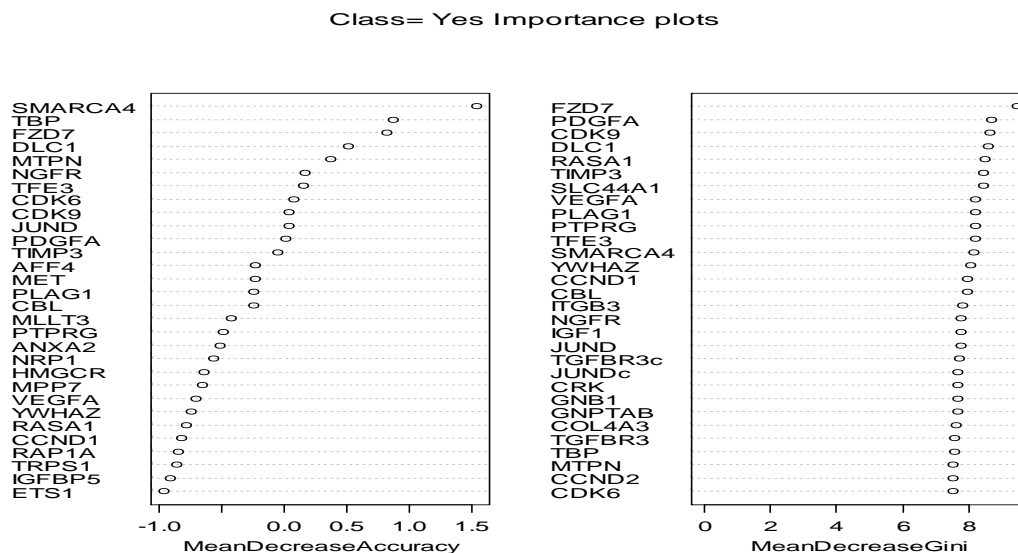


Figure 5. Variables of importance

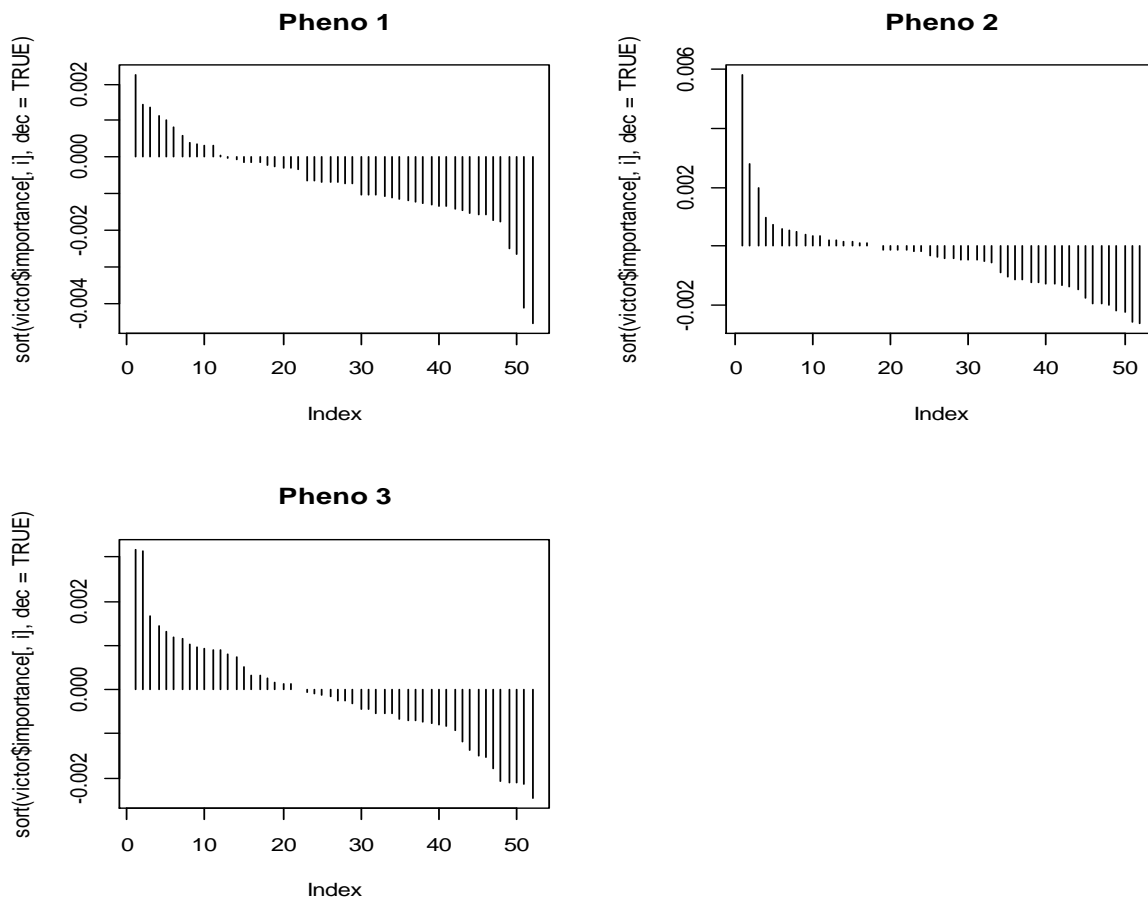


Figure 6. Plot of important genes per phenotype

CHAPTER 8. MISCLASSIFICATION ASSESSMENT OF DUALKS

8.1. 0.632 plus Bootstrap Method

Bootstrap resampling is used to estimate the sampling distribution of any statistic. It is an alternative cross-validation that generates new samples by drawing instances from the original samples with replacement. If we define the training data as $x = (x_1, x_2, x_3, \dots, x_N)$, then B bootstrap samples from the set of $z_1, z_2, z_3, \dots, z_B$ with z_i being a set of samples N . The estimated extra sample prediction error for the bootstrap is (Bradley & Robert, 1997):

$$Err_{boot} = \frac{1}{B} \sum_{b=1}^B \frac{1}{N} \sum_{i=1}^N L(y_i, f^b(x_i)),$$

where $f^b(x_i)$ is the predicted value at x_i from the model fit to the b^{th} bootstrap. This estimate by itself is not very accurate because the bootstrap samples used $f^b(x_i)$ may have contained x_i .

The leave-one-out bootstrap estimator improves the estimate by mimicking cross-validation and is defined as (Efron and Tibshirani, 1997):

$$Err_{boot(1)} = \frac{1}{N} \sum_{i=1}^B \frac{1}{|C^{-i}|} \sum_{b \in C^{-i}} L(y_i, f^b(x_i)),$$

where C^{-i} the set of indices is for the bootstrap samples that do not contain observation i , and $|C^{-i}|$ is the number of such samples. $Err_{boot(1)}$ solves the overfitting problem, but is still

biased. The average number of distinct observations in each sample is about $0.632N$. To solve the bias problem, Efron and Tibshirani proposed the 0.632 estimator in 1997:

$$Err_{0.632} = 0.368 \overline{error} + 0.632 Err_{boot(1)}$$

$$\overline{error} = \frac{1}{N} \sum_{i=1}^N L(y_i, f^b(x_i))$$

is the naïve estimate of prediction error often called training error. This bootstrap technique will be used to estimate the extra-sample prediction error. In order to gain more insights about the

poor performance of the DualKS classifier, we calculated the misclassification rate using the 0.632 plus estimators.

Table 11. Bootstrap method

Algorithms	Parameters		Misclassification rate using 0.632plus bootstrap
DualKS	B=100	n=4	0.6666
		n=7	0.6589
		n=2	0.6631
		n=1	0.6665

Table 11 summarizes the misclassification rate using the 0.632 plus bootstrap method. As we see, the number of boot is constant whereas the number of selected genes from each class of phenotype associated with the DualKS classifier varies. The average performance is about 66 percent.

CHAPTER 9. DISCUSSION

The aim of this research was to examine the different classifier learning abilities and also to compare the prediction rate of each algorithm. We observed that SVM classification outperformed k NN classification, RF classification, and DualKS classification. Although the performance of DualKS was low compared to SVM, k NN, RF, this was because the DualKS classifier needs more genes per class in order for its prediction to improve and it is a new classifier that needs more implementations, whereas SVM, k NN, RF are somehow effective in classifying small data. If the number of genes per class is not large enough, the prediction rate of the DualKS tends to be low. By comparing each classifier, we saw that SVM outperformed k NN by 45.6 percent, and 30.7 percent for RF and about 60 percent for the DualKS.

Furthermore, the average misclassification rate of the DualKS using the 0.632 plus bootstrap is about 66 percent which is very high but in this case it is acceptable since its classification was about 40.10 percent. With regards to the features of the DualKS algorithm and its low prediction rate, we could foresee that the misclassification rate will be very high. Therefore for every classification, this classifier will only classify about one-third of time and it will wrongly classify about two-third.

Trying to improve the performance of the DualKS, we investigated the reason of the low performance compared to the other three classifiers. We concluded that the size of the data and the number of genes per phenotype play an important role in the performance of the package. Since every dataset has its own features and characteristics, it is consistently impossible to foresee which algorithm will perform best. For future work, we could create a new version that combines both DualKS and k NN and compare the combination to the performance of SVM using the same dataset.

CHAPTER 10. CONCLUSION

The goal of this research was to evaluate the performance of SVM, k NN, RF and DualKS as learners and to better understand the idea behind each algorithm in order to determine the best classifier for the prostate cancer data. In order to get the optimal classification rate for each algorithm, the prostate cancer data has been analyzed using different parameters over the ranges and their associated plots were created in order to get a better understanding of the individual effect of each algorithm.

We observed that SVM classification outperformed k NN classification, RF classification, DualKS classification. It is important to note that all four algorithms represent different approaches of machine learning. The Random Forest, DualKS, and k -Nearest Neighbours all assume the underlying distribution of the prostate cancer dataset in a nonparametric manner, whereas SVM assumes that there is a hyperplane separating the three classes of phenotype.

Therefore, the performance of an algorithm and learning machine such as k NN, RF, SVM, DualKS depends on the characteristics or features of the dataset. Since every dataset has its own features, it is important to note that some classifiers might fall below the prediction rate of the DualKS whereas, another one might perform well in the case of SVM and RF. As a result, it is important to understand the features of the data before applying a classifier. Some classifiers might do well while others might do poorly.

REFERENCES

- Alexandros Karatzoglou, David Meyer, and Kurt Hornik, "Support Vector Machines in R," *Journal of Statistical Software* 15.9, Vol. 15, Iss 9, 2006.
- Bach Peter B, Deborah Schrag, Otis W Brawley, Aaron Galaznik, Sofia Yakren, and Colin B Begg, "Survival of Blacks and Whites after a Cancer Diagnosis" *Journal of the American Medical Association* 287.16, pages 2106-2113, 2002.
- Efron Bradley, and Robert Tibshirani, "Improvements on Cross-Validation: The 632+ Bootstrap Method," *Journal of the American Statistical Association* 92.438, pages 548-60, 1997.
- Breiman Leo, "Random Forests," *Machine Learning* 45.1, pages 5-32, 2001.
- Belur.V. Dasarathy, "Nearest Neighbor Norms: NN Pattern Classification Techniques," IEEE Computer Society Press, 1991.
- Chang Chih-chung, and Lin Chih-jen, "libsvm: A Library for Support Vector Machines," 2001.
- Clegg Limin X, Frederick P Li, Benjamin F Hankey, Kenneth Chu, and Brenda K Edwards, "Cancer survival among US whites and minorities: a SEER (surveillance, epidemiology, and end results) program population-based study," *Archives of Internal Medicine* 162(17), pages 1985–1993, 2002.
- Coleman MP, Quaresma M, Berrino F, et al. "Cancer survival in five continents: a worldwide population-based study (CONCORD)," *The Lancet Oncology* 9(8), pages 730–756, 2008.
- Cortes C., Vapnik V., "Support Vector Networks," *Machine Learning* 20, pages 273-297, 1995.
- David Meyer, "Support Vector Machines _The Interface to libsvm in package e1071," September 1, 2014.
- Delancey John Oliver L, Michael J Thun, Ahmedin Jemal, and Elizabeth M Ward, "Recent Trends in Black-White Disparities in Cancer Mortality," *Cancer Epidemiology, Biomarkers & Prevention : A Publication of the American Association for Cancer Research, Cosponsored by the American Society of Preventive Oncology* 17.11 , pages 2908-2912, 2008.
- Dimitriadou E, Hornik K, Leisch F, Meyer D, Weingessel A, "e1071: Misc Functions of the Department of Statistics (e1071)," TU Wien, Version 1.5-11, 2005.
- Edgar Ron, Michael Domrachev, and Alex E Lash, "Gene Expression Omnibus: NCBI Gene Expression and Hybridization Array Data Repository," *Nucleic Acids Research* 30.1, pages 207-10, 2002.

- Feigelson Eric and G. Jogesh Babu, “Beware the Kolmogorov-Smirnov test!,” Center for Astrostatistics, Penn State University, USA 2012.
- Hechenbichler K. and Schliep K.P, “Weighted k-Nearest-Neighbor Techniques and Ordinal Classification,” Discussion Paper 399, SFB 386, Ludwig-Maximilians University Munich, 2004.
- Jemal A, Siegel R, Ward E, Hao Y, Xu J, Thun MJ, “Cancer statistics,” CA Cancer Journal for Clinicians 59(4), pages 225–249, 2009.
- Jian-Hua Huang, Hua-Lin Xie, Jun Yan, Hong-Mei Lu, Qing-Song Xu, Yi-Zeng Liang, “Using random forest to classify T-cell epitopes based on amino acid properties and molecular features,” Analytica Chimica Acta, Volume 804, pages 70-75, December 4th, 2013.
- John C. Platt, “Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines,” April 21st, 1998.
- Khan, M., Ding, Q. and Perrizo, W, “K-Nearest Neighbors Classification of Spatial Data Streams using P-trees,” Proceedings of the PAKDD, pages 517-528, 2002.
- Kort Eric.J., Yarong Yang, Z. Zhang, B.T. Teh, and N. Ebrahimi, “Gene selection and classification of microarray data by twins kolmogorov-smirnov analysis,” Technical Report, 2008.
- Kort Eric.J., Yarong Yang, “Using DualKS,” October 13 2014.
- Kort Eric.J., Yarong Yang, Z. Zhang, B.T. Teh, and N. Ebrahimi, “Dual KS: Defining Gene Sets with Tissue Set Enrichment Analysis,” Cancer Informatics, Vol. 2010(9), 2010.
- Kumar, Rajnish, Anju Sharma, and Rajesh Kumar Tiwari, “Application of Microarray in Breast Cancer: An Overview,” Journal of Pharmacy & Bioallied Sciences 4.1, pages 21-26, 2012.
- Moreno-Seco, F., Mico, L. and Oncina, J, “A Modification of the LAESA Algorithm for Approximated k-NN Classification,” Pattern Recognition Letters 24, pages 47–53, 2003.
- Nello Cristianini and John. Shawe-Taylor, “An introduction to support vector machines and other kernel-based learning methods,” Cambridge University Press, 2000.
- Park Y, Shackney S, Schwartz R, “Network-based inference of cancer progression from microarray Data,” Computational Biology and Bioinformatics, IEEE/ACM Transactions on 6.2, pages 200-212, 2009.
- Quansheng Kuang, and Lei Zhao, “A Practical GPU Based KNN Algorithm,” School of Computer Science and Technology, Soochow University, Suzhou 215006, Huangshan, P. R. China, pages 151-155, December. 2009.

- R Development Core Team, "R: A Language and Environment for Statistical Computing," 2005.
- Ries Lag, Eisner MP, Kosary CL, Hankey BF "SEER Cancer Statistics Review," Bethesda, Md, USA: National Cancer Institute, pages 1973-1997, 2000.
- Simone Vincenzi, Matteo Zucchetto, Piero Franzoi, Michele Pellizzato, Fabio Pranovi, Giulio A. De Leo, Patrizia Torricelli, "Application of a Random Forest algorithm to predict spatial distribution of the potential yield of *Ruditapes philippinarum* in the Venice lagoon, Italy, Ecological Modelling," Volume 222, Issue 8, pages 1471-1478, April 24th, 2011.
- Subramanian A, Tamayo P, Mootha VK, et al., "Gene set enrichment analysis: a knowledge-based approach for interpreting genome-wide expression profiles," Proc Natl Acad Sci USA 102(43), pages 15545-15550, 2005.
- Stefan Lessmann, Robert Stahlbock, Sven F. Crone, "Genetic Algorithms for Support Vector Machine Model Selection," International Joint Conference on Neural Networks Sheraton Vancouver Wall Centre Hotel, Vancouver, Canada, pages 16-21, July 2006.
- Vapnik Vladimir N., "Estimation of Dependences Based on Empirical Data," New York: Springer, 1982.
- Vapnik Vladimir N., "The Nature of Statistical Learning Theory," New York: Springer, 1995.
- Vojislav .Kecman, "Learning and soft computing," The MIT PRESS, Cambridge, MA, 2001.
- Wild SH, Fischbacher CM, Brock A, Griffiths C, Bhopal R, "Mortality from all cancers and lung, colorectal, breast and prostate cancer by country of birth in England and Wales 2001–2003," British Journal of Cancer 94(7), pages 1079-1085, 2006.
- Xu, Yun, Simeone Zomer, and Richard G. Brereton. "Support Vector Machines: A Recent Method For Classification In Chemometrics," Critical Reviews In Analytical Chemistry 36.3/4, pages 177-188, 2006.

APPENDIX A. TRAINING THE ALGORITHMS

```
### Preparing the dataset

Load("Expression.RData")
source("dksPerm.r.R")
names(Exp)
str(Exp) # getting to know the data very well
Exp_new=Exp[,-which(colnames(Exp)=="MET")[2]] # eliminate the 2nd duplicate variable
class(Exp_new)
length(Exp_new)
names(Exp_new)
str(Exp_new)
ks<-as.matrix(Exp_new) # changing it to a matrix
ks
ks<-t(ks)
ks
Pheno<-Exp_new$Pheno # creating the classes of the phenotypes
duplicated(ks) # checking for duplicate
duplicated(names(Exp_new)) # checking for duplicate

###DUALKS algorithm

source("http://bioconductor.org/biocLite.R") # installing the package into R
biocLite("dualKS") # installing the package into R.
library(dualKS)
library("BiocInstaller")
ks<-as.matrix(Exp_new) # changing it to a matrix
ks<-t(ks)
ks
Pheno<-Exp_new$Pheno
tr<-dksTrain(ks,class=Pheno, type="up") # training for the upregulated genes
cl<-dksSelectGenes(tr,n=5) # extract 5 genes per class from the training data
pr<- dksClassify(ks, cl)
summary(pr, actual=Pheno) # getting the classification rate of the training set
show(pr)
plot(pr, actual=Pheno, main=" upregulated")
dv<- dksClassify(ks, cl, rescale=TRUE)
summary(dv, actual=Pheno)
plot(dv, actual=Pheno, main=" upregulated")

tr<-dksTrain(ks,class=Pheno, type="down")# training for the downregulated genes
cl<-dksSelectGenes(tr,n=5) # extract 5 genes per class from the training data
pr<- dksClassify(ks, cl)
summary(pr, actual=Pheno) # getting the classification rate of the training set
plot(pr, actual=Pheno)
```

```

dv<- dksClassify(ks, cl, rescale=TRUE)
summary(dv, actual=Pheno)
plot(dv, actual=Pheno)

tr<-dksTrain(ks,class=Pheno) # training for the genes
cl<-dksSelectGenes(tr,n=5) # extract 5 genes per class from the training data
pr<- dksClassify(ks, cl)
summary(pr, actual=Pheno) # getting the classification rate of the training set
show(pr)
plot(pr, actual=Pheno)
dv<- dksClassify(ks, cl, rescale=TRUE)
summary(dv, actual=Pheno)
source("dksPerm.r.R")
p.value <- dksPerm.r(ks, type="both", Pheno,m=5)
p.value
b=p.value(pr@predictedScore) # estimated p-values
b
a= data.frame(pr@predictedScore, pr@scoreMatrix) # constructing the table
a
summary(a)
plot(b)
plot(a)

```

###SVM algorithm

```

install.packages("e1071", dep=T)
install.packages("kernlab", dependencies = TRUE)
install.packages("rpart", dependencies = TRUE)
install.packages("RColorBrewer", dependencies = TRUE)
library(rpart) # loading packages from R
library(kernlab) # loading packages from R
library(e1071) # loading packages from R

x <- subset(Exp_new, select = -Pheno)
y <- Pheno # y is factor for the classification task

# finding the best kernels for SVM.
tuned <- tune.svm(x,y, polynomial = 10^(-6:-1), cost = 10^(1:3)) # give the best performance
plot(tuned, data=x, main=" polynomial performance")
v <- tune.svm(x,y, rbf = 10^(-6:-1), cost = 10^(1:3)) # give the best performance
plot(v, data=x, main=" rbf performance")
tunedS <- tune.svm(x,y, gamma = 10^(-6:-1), cost = 10^(1:3)) # give the best performance
plot(tunedS, main="gamma performance")
obj <- tune.svm(x,y, linear = 10^(-6:1), cost = 10^(1:3))
plot(obj, data=x, main="linear performance")
objS <- tune.svm(x,y, sigmoid = 10^(-6:1), cost = 10^(1:3))
plot(objS, data=x, main="linear performance")

```

```

# training the best kernel to find the cost and the prediction rate
model <- svm(x, y, probability = TRUE, kernel="radial", cost = 100, data= Exp_new)
model
summary(model)
pred <- predict(model, x) # Getting the prediction
pred
summary(pred)
summary(table(pred, y)) # check for accuracy
r=table(y, pred) # check for accuracy
classAgreement(r) #check for accuracy

```

```

#### K-Nearest Neighbour algorithm

```

```

install.packages("RWeka", dependencies = TRUE) # loading
install.packages("kknn", dependencies = TRUE)
library(class)
library(kknn)
library(RWeka)
library(rpart)
library(kernlab)
library(e1071)
x <- subset(Exp_new, select = -Pheno)
y <- Pheno # y is factor for the classification task
train=Exp_new # training data is the whole dataset
test= Exp_new # test data is the whole dataset

```

```

# determining the best kernels

```

```

(j<- train.kknn(Pheno~., train,test, kmax = 24, kernel= "rectangular", distance=2))
(j<- train.kknn(Pheno~., train,test, kmax = 24, kernel= "triangular", distance=2))
(j<- train.kknn(Pheno~., train,test, kmax = 24, kernel= "gaussian", distance=2))
(j<- train.kknn(Pheno~., train, test, kmax = 24, kernel= "rank", distance=2))
(j<- train.kknn(Pheno~., train, test, kmax = 24, kernel= "optimal", distance=2))
(j<- train.kknn(Pheno~., train,test, kmax = 24, kernel= "epanechnikov", distance=2))

```

```

#Using the best kernel ( rectangular kernel) to find the best k value

```

```

(j<- train.kknn(Pheno~., train,test, kmax = 5, kernel= "rectangular", distance=2))
(j<- train.kknn(Pheno~., train,test, kmax = 24, kernel= "rectangular", distance=2) )
(j<- train.kknn(Pheno~., train,test, k= 6, kernel= "rectangular", distance=2))
(j<- train.kknn(Pheno~., train,test, kmax = 7, kernel= "rectangular", distance=2) )
(j<- train.kknn(Pheno~., train,test, kmax = 8, kernel= "rectangular", distance=2))
(j<- train.kknn(Pheno~., train,test, kmax = 9, kernel= "rectangular", distance=2))
(j<- train.kknn(Pheno~., train,test, kmax = 10, kernel= "rectangular", distance=2))
(j<- train.kknn(Pheno~., train,test, kmax = 11, kernel= "rectangular", distance=2))
(j<- train.kknn(Pheno~., train,test, kmax = 12, kernel= "rectangular", distance=2))

```

```

(j<- train.kknn(Pheno~., train,test, kmax = 13, kernel= "rectangular", distance=2))

# using the sampling method to find the best k values

set.seed(26) # reproduces the same graph
tunes <- tune.knn(x,y,k=1:24, tunecontrol = tune.control(sampling = "boot")) # give the best
performance
summary(tunes)
plot(tunes, main=" finding best k values")

# training the knn with best parameters

v<- train.kknn(Pheno~., train,test, k= 24, kernel= "rectangular", distance=2) # provides best k
summary(v)
par(mfrow=c(1,1))
plot(v, type="l")
# computing the prediction accuracy
pred <- predict(v, x)
summary(pred)
sum(pred==y)/length(y) # check for accuracy

###Random Forest algorithm

install.packages("randomForest", dependencies = TRUE) # loading
install.packages("rpart", dependencies = TRUE) # loading
install.packages("RColorBrewer", dependencies = TRUE) # loading
library(randomForest)
library(rpart)
library(RColorBrewer)
library(randomForest)
x <- subset(Exp_new, select = -Pheno)
y <- Pheno # y is factor for the classification task

#training the rf

p3 <- randomForest(Pheno ~ ., data = Exp_new) # CLASSIFICATION ERROR RATE
print(p3)
summary(p3)
plot(p3)
pred <- predict(p3, x, type="prob") # prediction
summary(pred)
victor <- randomForest(Pheno ~ ., Exp_new, proximity=TRUE,keep.forest=FALSE,
na.action=na.omit, rsq=TRUE, mtry=7, do.trace=100, ntree=500,importance = TRUE)
victor
summary(victor)
plot(victor, main="Error rate over trees") # plotting the regression

```

```

# looking at important variables

importance(victor) # important variable
l=varImpPlot(victor) # important variable
varImpPlot(victor,class="Yes",main=" Class= Yes Importance plots") # plot of important
variable
par(mfrow = c(2, 2))
for (i in 1:3)
  plot(sort(victor$importance[,i], dec = TRUE), type = "h", main = paste("Pheno" , i))
plot(outlier(victor), type="h",col=c("red", "green", "blue")[as.numeric(Pheno)])
v=MDSplot(victor, Pheno) #Plot the scaling coordinates of the proximity matrix from
randomForest
kl=MDSplot(victor, Pheno, palette=rep(1, 3), pch=as.numeric(Pheno)) # Using different symbols
for the classes
margins.rf=margin(victor,Pheno)
plot(margins.rf)
hist(margins.rf,main="Margins of Random Forest for churn dataset")
boxplot(margins.rf~Pheno,
main="Margins of Random Forest for churn dataset by class")

```

APPENDIX B. MISCLASSIFICATION ALGORITHM

```
### DUALKS

library(dualKS)
library(bootstrap)
load("Exp.Rdata")
CL<-Exp[,1]
data<-t(Exp[,2:54])
B=100 # ADDED B=100
      # selecting four samples
boot_dualKS<-function(x=data,y=CL,nboot=B) {

  n<-length(y)
  saveii<-NULL
  miss.clas<-function(y,yhat) { 1*(yhat!=y)}
  tr<-dksTrain(eset=x,class=y,type="up",method="yang")
  GS<-dksSelectGenes(data=tr,n=4)
  CL_pred<-dksClassify(eset=data, classifier=GS)@predictedClass
  app.err<-mean(miss.clas(y,CL_pred))
  err1<-matrix(0,nrow=nboot,ncol=n)
  err2<-rep(0,nboot)
  for(b in 1:nboot) {
    ii<-sample(1:n,replace=TRUE)
    saveii<-cbind(saveii,ii)
    tr_fit<-dksTrain(eset=x[,ii],class=y[ii],type="up",method="yang")
    GS_fit<-dksSelectGenes(data=tr_fit,4)
    CL_pred_1<-dksClassify(eset=x[,ii],classifier=GS_fit)@predictedClass
    CL_pred_2<-dksClassify(eset=x[,ii],classifier=GS_fit)@predictedClass
    err1[b,]<-miss.clas(y,CL_pred_2)
    err2[b]<-miss.clas(y[ii],CL_pred_1)
  }

  optim<-mean(apply(err1,1,mean)-err2)
  junk<-function(x,i) sum(x==i)

  e0 <- 0
  for (i in 1:n) {
    o <- apply(saveii, 2, junk, i)
    if (sum(o == 0) == 0)
      cat("increase nboot for computation of the .632 estimator",
          fill = TRUE)
    e0 <- e0 + (1/n) * sum(err1[o == 0, i])/sum(o == 0)
  }

  err.632 <- 0.368 * app.err + 0.632 * e0
```



```

gamma <- sum(outer(y, CL_pred, function(x1, y) ifelse(x1==y, 0, 1)))/
  (length(y)^2)
r <- (e0 - app.err)/(gamma - app.err)
r <- ifelse(e0 > app.err & gamma > app.err, r, 0)
errprime <- min(e0, gamma)
err <- err.632 + (errprime - app.err)*(0.368*0.632*r)/(1-0.368*r)
return(list(err632=err.632,err632plus=err))
}

kk=boot_dualKS(x=data,y=CL,nboot=B)
kk
`      # selecting seven samples
boot_dualKS<-function(x=data,y=CL,nboot=B) {

  n<-length(y)
  saveii<-NULL
  miss.clas<-function(y,yhat) { 1*(yhat!=y)}
  tr<-dksTrain(eset=x,class=y,type="up",method="yang")
  GS<-dksSelectGenes(data=tr,n=7)
  CL_pred<-dksClassify(eset=data, classifier=GS>@predictedClass
  app.err<-mean(miss.clas(y,CL_pred))
  err1<-matrix(0,nrow=nboot,ncol=n)
  err2<-rep(0,nboot)
  for(b in 1:nboot) {
    ii<-sample(1:n,replace=TRUE)
    saveii<-cbind(saveii,ii)
    tr_fit<-dksTrain(eset=x[,ii],class=y[ii],type="up",method="yang")
    GS_fit<-dksSelectGenes(data=tr_fit,4)
    CL_pred_1<-dksClassify(eset=x[,ii],classifier=GS_fit>@predictedClass
    CL_pred_2<-dksClassify(eset=x[,ii],classifier=GS_fit>@predictedClass
    err1[b,]<-miss.clas(y,CL_pred_2)
    err2[b]<-miss.clas(y[ii],CL_pred_1)
  }

  optim<-mean(apply(err1,1,mean)-err2)
  junk<-function(x,i) sum(x==i)

  e0 <- 0
  for (i in 1:n) {
    o <- apply(saveii, 2, junk, i)
    if (sum(o == 0) == 0)
      cat("increase nboot for computation of the .632 estimator",
        fill = TRUE)
    e0 <- e0 + (1/n) * sum(err1[o == 0, i])/sum(o == 0)
  }
}

```

```

err.632 <- 0.368 * app.err + 0.632 * e0

gamma <- sum(outer(y, CL_pred, function(x1, y) ifelse(x1==y, 0, 1)))/
  (length(y)^2)
r <- (e0 - app.err)/(gamma - app.err)
r <- ifelse(e0 > app.err & gamma > app.err, r, 0)
errprime <- min(e0, gamma)
err <- err.632 + (errprime - app.err)*(0.368*0.632*r)/(1-0.368*r)
return(list(err632=err.632,err632plus=err))
}

kk=boot_dualKS(x=data,y=CL,nboot=B)
kk

# selecting two samples
boot_dualKS<-function(x=data,y=CL,nboot=B) {

  n<-length(y)
  saveii<-NULL
  miss.clas<-function(y,yhat) {1*(yhat!=y)}
  tr<-dksTrain(eset=x,class=y,type="up",method="yang")
  GS<-dksSelectGenes(data=tr,n=2)
  CL_pred<-dksClassify(eset=data, classifier=GS>@predictedClass
  app.err<-mean(miss.clas(y,CL_pred))
  err1<-matrix(0,nrow=nboot,ncol=n)
  err2<-rep(0,nboot)
  for(b in 1:nboot) {
    ii<-sample(1:n,replace=TRUE)
    saveii<-cbind(saveii,ii)
    tr_fit<-dksTrain(eset=x[,ii],class=y[ii],type="up",method="yang")
    GS_fit<-dksSelectGenes(data=tr_fit,4)
    CL_pred_1<-dksClassify(eset=x[,ii],classifier=GS_fit>@predictedClass
    CL_pred_2<-dksClassify(eset=x[,ii],classifier=GS_fit>@predictedClass
    err1[b,]<-miss.clas(y,CL_pred_2)
    err2[b]<-miss.clas(y[ii],CL_pred_1)
  }

  optim<-mean(apply(err1,1,mean)-err2)
  junk<-function(x,i) sum(x==i)

  e0 <- 0
  for (i in 1:n) {
    o <- apply(saveii, 2, junk, i)
    if (sum(o == 0) == 0)
      cat("increase nboot for computation of the .632 estimator",
        fill = TRUE)
    e0 <- e0 + (1/n) * sum(err1[o == 0, i])/sum(o == 0)
  }
}

```

```

}

err.632 <- 0.368 * app.err + 0.632 * e0

gamma <- sum(outer(y, CL_pred, function(x1, y) ifelse(x1==y, 0, 1)))/
  (length(y)^2)
r <- (e0 - app.err)/(gamma - app.err)
r <- ifelse(e0 > app.err & gamma > app.err, r, 0)
errprime <- min(e0, gamma)
err <- err.632 + (errprime - app.err)*(0.368*0.632*r)/(1-0.368*r)
return(list(err632=err.632,err632plus=err))
}

kk=boot_dualKS(x=data,y=CL,nboot=B)
kk

# selecting one sample
boot_dualKS<-function(x=data,y=CL,nboot=B) {

n<-length(y)
saveii<-NULL
miss.clas<-function(y,yhat) { 1*(yhat!=y)}
tr<-dksTrain(eset=x,class=y,type="up",method="yang")
GS<-dksSelectGenes(data=tr,n=1)
CL_pred<-dksClassify(eset=data, classifier=GS>@predictedClass
app.err<-mean(miss.clas(y,CL_pred))
err1<-matrix(0,nrow=nboot,ncol=n)
err2<-rep(0,nboot)
for(b in 1:nboot) {
  ii<-sample(1:n,replace=TRUE)
  saveii<-cbind(saveii,ii)
  tr_fit<-dksTrain(eset=x[,ii],class=y[ii],type="up",method="yang")
  GS_fit<-dksSelectGenes(data=tr_fit,4)
  CL_pred_1<-dksClassify(eset=x[,ii],classifier=GS_fit>@predictedClass
  CL_pred_2<-dksClassify(eset=x[,ii],classifier=GS_fit>@predictedClass
  err1[b,]<-miss.clas(y,CL_pred_2)
  err2[b]<-miss.clas(y[ii],CL_pred_1)
}

optim<-mean(apply(err1,1,mean)-err2)
junk<-function(x,i) sum(x==i)

e0 <- 0
for (i in 1:n) {
  o <- apply(saveii, 2, junk, i)
  if (sum(o == 0) == 0)
    cat("increase nboot for computation of the .632 estimator",

```

```

      fill = TRUE)
    e0 <- e0 + (1/n) * sum(err1[o == 0, i])/sum(o == 0)
  }

  err.632 <- 0.368 * app.err + 0.632 * e0

  gamma <- sum(outer(y, CL_pred, function(x1, y) ifelse(x1==y, 0, 1) ))/
    (length(y)^2)
  r <- (e0 - app.err)/(gamma - app.err)
  r <- ifelse(e0 > app.err & gamma > app.err, r, 0)
  errprime <- min(e0, gamma)
  err <- err.632 + (errprime - app.err)*(0.368*0.632*r)/(1-0.368*r)
  return(list(err632=err.632,err632plus=err))
}

kk=boot_dualKS(x=data,y=CL,nboot=B)
kk

```