# A Comparative Machine Learning Study in Bladder Cancer Diagnosis

Canada, Memorial University of Newfoundland

F. Dorani F. M. Kazemi S. S. Mirhendi M. H. Sheykholeslam

April 12, 2016

**Abstract**

The aim of this research is to find the best model that can fit to **Dataset 3** which is a data-set from bladder cancer patients that contains their gender, smoke status and genotypic data.

At first we will try to limit features based on two methods: Generalized Low Rank Models(GLRM) and Gradient Boosted Feature Selection (GBFS). Then, two methods of classification including Support Vector Machine (SVM) and Gradient Boosting Machine (GBM) will be applied to analyze purified data. Finally, two performance criteria including accuracy value and area under the ROC curve (AUC) are used to evaluate output results.

The results showed that GLRM feature extraction method performed well by imputing missing values and producing low dimensional features vector. On the other hand, GBFS feature selecting method could efficiently find the most important features that may affect the patient cancer status.

## 1 Introduction

Genome-wide association studies (GWAS) is commonly used for discovering complex disease genes. These discoveries may lead to improve diagnostic accuracy and to make new drugs that may hopefully lead to more efficient treatment and disease prevention options.

In this article, we would like to apply Machine Learning methods to recognize bladder cancer affected people. To get this done, we suggested the idea of applying two different classifiers (SVM and GBM) on output data of two different feature selection/extraction methods (GBFS and GLRM) to find out the best model that can predict this cancer based on our input data-set.

On the rest of this article, firstly we will describe our dataset properties and what we have done to clean this data and to overcome data heterogeneity and missing values. In the second section, we briefly described principles of four methods (two for feature extraction and two for classification) that we used. In the third part, we have our methods output results and in the fourth section, we discuss the results and their interpretations as well. Finally, we have a conclusion and what could be done in the future works.

## 1.1 Dealing With Dataset

The dataset consists of both phenotypic and genotypic data from people who have been diagnosed with bladder cancer (cases) and people who have normal health status (controls). The goal of our analysis is to highlight a test that can recognize a healthy person from one who is dealing with bladder cancer.

The input dataset has 1694 categorical and binary features and 2048 samples while a huge number of feature values are missing and we don't have any information about them. On the other hand, in order to use the classification and feature selection methods we need a clean dataset with no missing values. To overcome data imperfection barriers, we chose two strategies:

1. **Generalized Low Rank Models:** We used this method for imputing missing data and extracting features on raw data (2048 sample with 1694 features). More details are available in section 2.

2. **Statistical Methods:** Firstly, we performed some statistical analysis in our data to find out raw data structure, number of its missing values, mean and standard deviation for each feature. Based on these analyses, we set some thresholds for both samples and features to decide if they have to remain or to be removed.

It is clear that the dataset has around 30% of missing values. Thus, applying just data imputation methods on them is not the right option because

it causes bias in the results. The better choice is to remove some samples and features that have more (based on a threshold) missing values. By threshold 5, it means for every row, if it has more than 5 percent missing values in all features, it would be considered for removal. The same scenario applied for features. If a feature has missing values more than a threshold (e.g. 5 percent) in all samples, it would be considered for removal.

The order of imputation does matter regarding rows and columns. It means, it's better to consider removing rows first and then columns in subject of imputation. For the quality control, samples that have more than 5% missing data are removed. Counting wise, if a row has more than 85 "NA" is considered for elimination from dataset. This methodology could remove 925 rows from dataset and remained us with only 1123 samples. Next step is the quality control of features or columns. The same as rows, we have to remove columns that have more than 5% missing data for all samples. In other words, columns that have more than 56 missing samples will be removed. After applying this method, it removed 108 columns and remained use 1587 features.

After pruning the dataset, the missing values have to be filled out with proper data. The most frequent value of each column would be considered for replacing the missing value for that column. Note that, if we decided to remove small amount of rows, then we had to remove more columns in order to keep more useful data. That's just the matter of number of "NA"s in the original dataset.

| Dataset | No. of Samples | No. of Features |
|---|---|---|
| Dataset1 | 2048 | 1694 |
| Dataset2 | 1123 | 1694 |
| Dataset3 | 1123 | 1587 |
| Dataset4 | 1123 | 1634 |
| Dataset5 | 1123 | 1673 |
| Dataset6 | 1123 | 1679 |

Table 1: Datasets names, number of samples and number of features of each one. Dataset1 represents the raw data and Dataset2 represents trimmed data without any feature removal

Because of our time limit and for avoiding further complexity, we only considered 4 datasets which are based on 5 percent threshold on rows and 5,

10, 50, and 90 percent thresholds on features and also one trimmed dataset which nothing was eliminated and another dataset which only samples were removed. It means we have datasets with 1123 samples and 1694, 1694, 1587, 1634, 1637, and 1679 features respectively. Table 1.1 shows the datasets and their number of samples and number of features. We will continue our further analysis based on mentioned strategies.

# 2 Methods

In this section, we will describe principles of methods that we used for analyzing data. We will explain a feature extraction method (GLRM) as well as a feature selection approach (GBFS). Furthermore, we will describe classification methods that we used for analyzing extracted/selected features. Finally, in the next section, we will apply our methods into data and find out how well our algorithms will perform.

## 2.1 Feature Selection/Extraction

### 2.1.1 Generalized Low Rank Models (GLRM)

Udell et. al. [9] extended the idea behind Principal components analysis (PCA) into a generalized method that can handle different types of data sets consisting numerical, Boolean, categorical, ordinal, and other data types. This approach, named Generalized Low-Rank Models (GLRM), handles heterogeneous data sets while it can compress and denoise data as well as imputing missing records. We will use this method for extracting the huge number of input data features while we use all of observed samples even if they have many missing features values.

Firstly, we will describe principles of this approach and its parameters quickly, then we will try to find appropriate model parameters for our dataset. Finally, in the results section, we will classify our date based on features that extracted here.

GLRM represents high dimensional massive data in a lower-dimensional space. Consider we have a matrix $A$ that has $n$ rows representing samples and $m$ columns that represent $m$ features. Additionally, these features may take different types of date, for instance, one column may take float values while others have categorical values. By solving the optimization problem, we can approximate $A$ by $X$ as a "tall and skinny" matrix and $Y$ as a "short

and wide" matrix (figure 1 shows this concept). $X$ represents the $k$ new latent features for $m$ samples, and $Y$ encodes the transformation of matrix $A$ features into the $k$ new latent features.

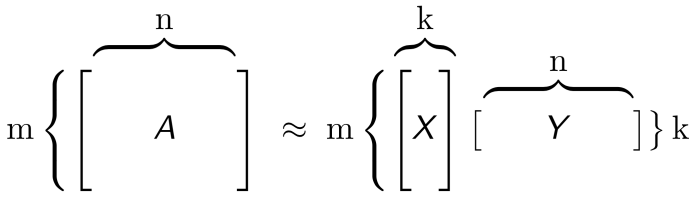



Figure 1: Approximate $A$ by $X$ and $Y$

To find $X$ and $Y$ the following optimization problem should be solved:

$$minimize \quad \sum_{i,i\in\Omega}(A_{ij} - x_i y_j)^2 + \sum_{i=1}^{m} r_i(x_i) + \sum_{j=1}^{n} \tilde{r}_j(y_j) \tag{1}$$

The loss function (sigma part) will measure the accuracy of data approximation and the problem-solving algorithm will try to minimize this part. Different loss functions are appropriate for different types of inputs and GLRM gives the flexibility to define different functions for each column of data (features) based on their values. This loss should be only calculated over the set $\Omega$ which represents non-missing entries.

On the other hand, the regularizers $r_x$ and $r_y$ limits latent feature values. Choosing appropriate regularization can improve the model and prevent it from over-fitting.

**Implementing Model:** To find our data set model, we used the $H_2O$ package in R which implements the GLRM algorithm illustrated by Udell et. al [9]. This package use a very fast and powerful algorithm as well as parallel processing to solve mentioned optimization problem.

To form a GLRM, we need to specify the loss functions, regularizers and a rank $k$ (number of extracted latent features). Due to the fact that all of our features values are categorical we will use Categorical PCA loss function based on Udell et. al. [9]. Additionally we used no regularizers because our input date is too spars to over-fit and we don't need regularizers here. We also used cross-validation to find an appropriate rank for our model.

By using the term cross-validation here we mean re-sampling the data to evaluate the model's performance. We will omit a portion of observed values from input data and fit a model to this incomplete data-set. Then we will predict all missing values by multiplying the output $X$ and $Y$ matrices. By comparing these predicted values to omitted values we can find how well our model is fitted.

By choosing $k = 1$ the model predicted about 55% of omitted values correctly. When $k = 10$ this portion increased to about 62% and by choosing much higher values for $k$, ($k = 100$ and $k = 150$ we couldn't achieve more that 66% validity. Thus we set $k$ to it's moderate value $k = 10$ for later analysis. It must be mentioned that we also test regularizers effect by this approach and they didn't improved anything thus we didn't use them and set them to zero.

All in all, by using GLRM we could compress the enormous number of features into a few number of latent features. On the other hand we didn't throw away any samples even if they had lots of missing values. These two points made this method very powerful for further analyzing data. In results section we will apply classification algorithms to the extracted latent features ($X$ matrix) concatenated by patient cancer status to find out how well this method could be helpful.

### 2.1.2 Gradient Boosted Feature Selection (GBFS)

Feature selection methods provides us a way of reducing computation time, improving prediction performance, and a better understanding of the data in machine learning or pattern recognition applications.

A feature selection algorithm should ideally satisfy four conditions. First of all, it should extract related features. The second condition is that feature selection algorithm should identify non-linear feature relations. Third, the method should scale linearly with the number of features and dimensions. The last but not least is that allow the combination of known sparsity structure. Gradient Boosted Feature Selection [10] satisfy all of this conditions.

GBFS has motivated by the (NP-Hard) feature selection problem and use gradient boosted regression trees [4] for which greedy approximations [1] exist. Following the gradient boosting framework, trees are built with the greedy CART algorithm [1]. Features are selected sparsely following an important change in the impurity function:

- Splitting on new features is penalized by a cost $\lambda > 0$.

- Re-use of previously selected features incurs no additional penalty.

GBFS has several compelling properties:

1. Instead of using a linear classifier like $l_1$-norm, GBFS follow [11] and use gradient boosted regression trees so it naturally discover nonlinear interactions between features while learning an ensemble of regression trees.

2. In contrast to other feature selection methods like Random Forests [6], it unifies feature selection and classification into a single optimization.

3. In contrast to existing nonlinear feature selection algorithms, its time and memory complexity scales as $O(dn)$, where d denotes the number of features and n the number of samples.

4. GBFS can naturally incorporate pre-specified feature cost structures or side-information, e.g., select bags of features or focus on regions of interest, similar to generalized lasso in linear feature selection [8].

**Algorithm 1** GBFS in pseudo-code.

Input: data$\{x_i, y_i\}$, learning rate $\epsilon$, iteration $T$.
Initialize predictions $H = 0$ and selected feature set $\Omega = 0$.
**for** $t = 1$ **to** $T$ **do**
    Learn $h_t$ using greedy CART to minimize the Eq.3.
    Update $H = H + \epsilon h_t$
    **for** each feature $f$ used in $h_t$ **do**
        set $\phi_f = 0$ and $\Omega = \Omega \cup f$
Return $H$ and $\Omega$

The approach that HSIC Lasso [12] uses for incorporating non-linear feature selection but it has order of $n^4$ memory and time complexity. But, GBFS uses boosting which has more efficient performance. Following equation motivates GBFS:

$$\min_{\beta} \sum_{(\phi(x_i), y_i)} \ell(\phi(x_i), y_i, \beta) + \lambda|\beta|_1 + \mu q_\epsilon(\beta) \tag{2}$$

Here, $\phi(x)$ is vector of all possible regression trees that obtain identical values on entire training set and $\beta$ is a sparse linear vector that selects trees. It

should mentioned that Eq. (2) applied two penalties. The first one reduces overfitting while the second penalize selecting new feature. The algorithm finally uses following equation:

$$h_t = \underset{h_t \in \mathcal{H}}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^{n} (g_i - h_t(x_i))^2 + \mu \sum_{f=1}^{d} \phi_f F_{ft} \tag{3}$$

The equation 3 is a good solutions can be found efficiently via the greedy CART algorithm for learning regression trees [4]. The Eq. 3 encourages feature splits to best match the negative gradient of the loss function, and the second term rewards splitting on features which have already been used in previous iterations. Algorithm 1 summarized the overall algorithm in pseudo-code.

## 2.2 Classification

### 2.2.1 Support Vector Machines (SVM)

Support vector machine method is one of the most popular ones for pattern recognition problems.Generally, a Support Vector makes a hyperplane or more hyperplanes in a high-dimensional space, which can be used for pattern recognition. By using these hyperplanes, good separation is attained, which has the largest distance from the nearest training data points of the classes. Thus, the larger the margin provides the lower the error of the classification. Also, the original dimensional space can be mapped into a multidimensional space to be able to classify the not linearly separable classes in the original space, and the discrimination is easier in the space. The basic idea is that a nonlinear separating surface in the low-dimensional space can be mapped into a linear hyperplane in the high-dimensional space.Also, a kernel function $K(x, y)$ is designed for the mapping used by Support Vectors to support computing the points in terms of the variables in the original space. The use of a kernel function provides the linear separation in non-linear classification problems [2].

Generally, the idea behind the SVM classifier can be explained based on four basic notions: (i) the separating hyperplane, (ii) the maximum margin hyperplane, (ii) the soft margin, and (iv) the kernel function. In general, a SVM looks for a separating hyperplane in the data that produces the largest separation margin between two classes [2, 3].

### 2.2.2 Gradient Boosting Machine (GBM)

Gradient Boosting is a powerful machine learning algorithm that can do regression, classification and ranking. Literally, **Gradient Boosting** is combination of **Gradient Descent** and **Boosting**. In other words, formulating a boosting approach (like Adaboost[1]) as gradient descent with a special loss function then generalizing it to Gradient Boosting in order to handle a variety of loss functions. Explaining gradient boosting is possible by saying that boosting shares a close connection to forward stage-wise modeling. Considering Adaboost, it can be viewed as a forward step-wise model where the base learners are trees, and the loss is exponential. So, by applying other loss functions we can generalize it to gradient boosting. Moreover, there are two key points in gradient boosting. First, function estimation and second numerical optimization. What actually being done in gradient boosting is that each learner estimates the gradient of loss function and then applies a numerical optimization like steepest descent to take sequence of steps to reduce the loss.

**Learning Task:** In GBM, the learning procedure consecutively fits new models to provide a more accurate estimate of the response variable. The main idea of this algorithm is to create the new base-learners to be maximally correlated with the negative gradient of the loss function, in the whole ensemble[5]. The loss functions applied can be arbitrary, which can be selected from both a rich variety of loss functions derived so far and the possibility of implementing one's own task-specific loss. Given the $dataset(x, y)$ for $i = 1, ..., N$ where $x = (x_1, x_2, ..., x_d)$ refers to the explanatory input variables and $y$ to the corresponding labels of the response variable and $f(x)$ is the prediction of response variable y and also $L(y, f(x))$ would be the loss function. The loss in using $f(x)$ to predict y on the training data is:

$$L(f) = \sum i = 1^N L(y_i, f(x_i)) \tag{4}$$

The goal is to minimize $L(f)$ with respect to f. In contrast to SGD, f is not a numerical but a function or tree. Then, minimizing the $L(f)$ can be viewed as numerical optimization:

$$\hat{f} = \underset{f}{\operatorname{argmin}} L(f) \tag{5}$$

[1]Adaboost is powerful boosting algorithm

where $f = f(x_1), f(x_2), ..., f(x_N)$ are the values of approximating $f$ at each of the $N$ data points. Numerical optimization procedures solve the above equation as a sum of component vectors:

$$f_M = \sum_{m=0}^{M} h_m, h_m \in \mathbb{R} \tag{6}$$

where $f_0 = h_0$ is an initial guess and each successive $f_m$ is computed based on the current parameter vector $f_{m-1}$, which is the sum of the previously induced updates. Then we use Steepest Descent for optimizing the function. Steepest descent chooses negative gradient $h_m = \rho_m g_m$ where $\rho_m$ is a scalar and $g_m$ is the gradient of $L(f)$ evaluated at $f = f_{m-1}$. The step length $\rho_m$ is the solution to:

$$\rho_m = \underset{\rho}{\operatorname{argmin}}\, L(f_{m-1} - \rho g_m) \tag{7}$$

The current solution is then updated

$$f_m = f_{m-1} - \rho g_m \tag{8}$$

and the process repeated at the next iteration. Steepest descent is very greedy, since $-g_m$ is the local direction in $\mathbb{R}^N$ for which $L(f)$ is rapidly decreasing at $f = f_{m-1}$. At the end, the algorithm of GBM from Friedman Gradient Boosting Approach paper:

**Algorithm 1: Gradient_Boost**

1. $F_0(\mathbf{x}) = \arg\min_\rho \sum_{i=1}^{N} L(y_i, \rho)$
2. For $m = 1$ to $M$ do:
3. $\quad \tilde{y}_i = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}, \; i = 1, N$
4. $\quad \mathbf{a}_m = \arg\min_{\mathbf{a},\beta} \sum_{i=1}^{N} [\tilde{y}_i - \beta h(\mathbf{x}_i; \mathbf{a})]^2$
5. $\quad \rho_m = \arg\min_\rho \sum_{i=1}^{N} L(y_i, F_{m-1}(\mathbf{x}_i) + \rho h(\mathbf{x}_i; \mathbf{a}_m))$
6. $\quad F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h(\mathbf{x}; \mathbf{a}_m)$
7. endFor

end Algorithm

Figure 2: Algorithm of GBM from The Elements of Statistical Learning

Simply put, gradient boosting works as:

- Using a simple (regression) model to start.
- Subsequent models predict (or fit) the loss (or negative gradient) of the previous predictions.
- Overall prediction is given by a weighted sum of the collection

Overfitting in GBM is possible with different types of base-learners with very different loss-functions. To avoid it, number of different regularization procedures are introduced including subsampling, shrinkage and early stopping. **Relative Influence:** Another good point about GBM is that it also develops an extension of a variable's "relative influence" for boosted estimates. For tree based methods the approximate relative influence of a variable $x_j$ is

$$\hat{J}^2 = \sum splits\ on\ x_j I_t^2 \tag{9}$$

where $I_t^2$ is the empirical improvement by splitting on $x_j$ at that point. Friedman's extension to boosted models is to average the relative influence of variable $x_j$ across all the trees generated by the boosting algorithm.

**Implementation:** In the GBM there are some parameters that have to be specified before running the algorithm. Having the best combination of these parameters would have great effect on the result. The first and foremost choice is distribution. By distribution it means the loss function being used in the GBM algorithm. For most classification problems either bernoulli or adaboost will be appropriate, but the former being recommended. The distribution for two-class classification problems is bernoulli and for multiclass classification problems is multinomial, and when the distribution is not specified the gaussian is determined by default. As it is clear in our dataset, there is only two classes ("0" and "1") as target variables. Thus, we used the bernoulli distribution which is actually a logistic regression loss because we have 0–1 outcomes. In the bernoulli loss the returned value is on the log odds scale.

Three other main parameters to be tuned in the GBM are 1) number of trees, also known as number of iterations of the algorithm 2) shrinkage parameter which prevent the algorithm from over-fitting 3) number of splits or size of each constituent tree.

There is always an issue when choosing n.trees (number of trees) and shrinkage (learning rate) parameters. It is important to know that smaller

values of shrinkage (almost) always give improved predictive performance. That is, setting shrinkage=0.001 will almost certainly result in a model with better out-of-sample predictive performance than setting shrinkage=0.01. However, there are computational costs, both storage and CPU time, associated with setting shrinkage to be low. The model with shrinkage=0.001 will likely require ten times as many iterations as the model with shrinkage=0.01, increasing storage and computation time by a factor of 10. Most figures below demonstrate the accuracy of model on different shrinkage values and number of iterations. For the number of iterations, there have to be an optimal number for different values of shrinkage. Note that as number of iterations increases between different values of shrinkage its storage and computational complexity increases by the factor roughly equal to that of shrinkage parameter. As indicated in [7], the rule of thumb is to set shrinkage as small as possible while still being able to fit the model in a reasonable amount of time and storage. It usually aims for 3,000 to 10,000 iterations with shrinkage rates between 0.01 and 0.001.

The *interaction.depth* or the depth of each tree is another important parameter that should be considered for tuning. Interaction depth of 1 (which also called as stump) adds more both time and cost complexity to the algorithm in comparison to higher values like 5 or 9. In our implementation, the interaction depths are tuned between 1, 5, and 9.

The *bag.fraction* or the subsampling rate is the fraction of the training set observations randomly selected to propose the next tree in the expansion. This introduces randomness into the model fit. If *bag.fraction* is set to be greater than 0 (0.5 is recommended), gbm computes an out-of-bag estimate of the improvement in predictive performance. It evaluates the reduction in deviance on those observations not used in selecting the next regression tree.

The *n.minobsinnode* is another parameter which is the number of observations in trees terminal nodes. This is actual number of observations not the total weight. In our implementation, we used the *n.minobsinnode* of 10 for the algorithm. This is a kind of parameter that could also be considered for tuning but it is not that influential like other parameters.

For implementation the R *gbm* (Generalized Boosted Regression Modeling) package was used. The gbm package contains different boosting algorithms such as Adaboost, Gradient Boosting Machine, and etc which in our case just the Gradient Boosting Machine was the main and target algorithm. Different run of the algorithm containing combination of the GBM parameters with different values will ends up in different results. Thus, it

is important to tune these parameters so that the algorithm gives better results. The *caret* package in R is a general purpose tool which is used for the tuning intentions. It can tune parameters for any algorithm given different values of parameters. Because of high computation time of the GBM and our limited amount of time, we only considered very simple configuration for the model. As an example, one test of model with shrinkage of 0.01 took around 75 minutes to finish. However, lower values of shrinkage give results with high accuracy. For this reason, we only used the shrinkage of 0.1. The configuration of GBM is as follow:

| GBM Parameters | Values to tune |
|---|---|
| n.trees | {50, 100, 150} |
| Interaction.depth | {1, 2} |
| Shrinkage | {0.1} |
| n.monobsinnode | {10} |
| bag.fraction | 0.5 (default value) |

Table 2: Setup values for GBM classifier.

### 2.2.3 k-Nearest Neighbor (K-NN)

Nearest neighbor method is based on the distance determination between the samples. Many different distance calculation criteria are available. The Euclidean distance was used to determine distances between an unknown sample and each of the samples of the training dataset. After this stage, the smallest distance is selected for the classification of the class. In the KNN, the k-nearest samples to the unknown sample are selected, and the unknown sample is classified into the group to which the majority of the k samples belong to. The k values are determined by calculating the prediction ability. Regarding the advantages of this method, we should say that this method has mathematical simplicity, and its effectiveness does not depend on the space distribution of the classes [3].

# 3 Experimental Results

Before the model building, the raw data analyzed. To visualize the structure of the raw data we obtained the number of missing values, mean and standard deviation for each feature. For details you can refer to statistics file that attached to email.

We compared the discriminatory ability of two popular classifiers, which are SVM and KNN classifiers and a new classifier GBM. As an objective measure for estimating the discriminatory ability, we determined the accuracy, confusion matrix and area under the ROC curve (AUC).

In our experiments, we used a 10-fold cross-validation technique by 10 times repetitions. It means the cross-validation is repeated 10 times for 10 different folds of data.

## 3.1 SVM Classifier Output Results

Parameter settings listed in table 3 was used for exploring the performance of SVM classifier on our datasets. For SVM, we used linear and quadratic kernels. In this project we reported the results of applying linear kernel. The results of using quadratic kernel was better that we will report them in the future reports.

Table 3. SVM parameters on the models.

| | |
|---|---|
| Type of SVM | One versus One |
| KernelFunction | Linear |
| DeltaGradientTolerance | 1.0000e-3 |
| IterationLimit | 1000000 |
| Kernel Scale | 53.65 |
| Nu | 0.5 |
| ShrinkagePeriod | 0 |
| Reason for convergence | Delta gradient |

### 3.1.1 Without Applying Feature Selection/Extraction

Figures 3 and 4 show the Accuracy, ROC curves, AUC values and confusion matrix of the two models.

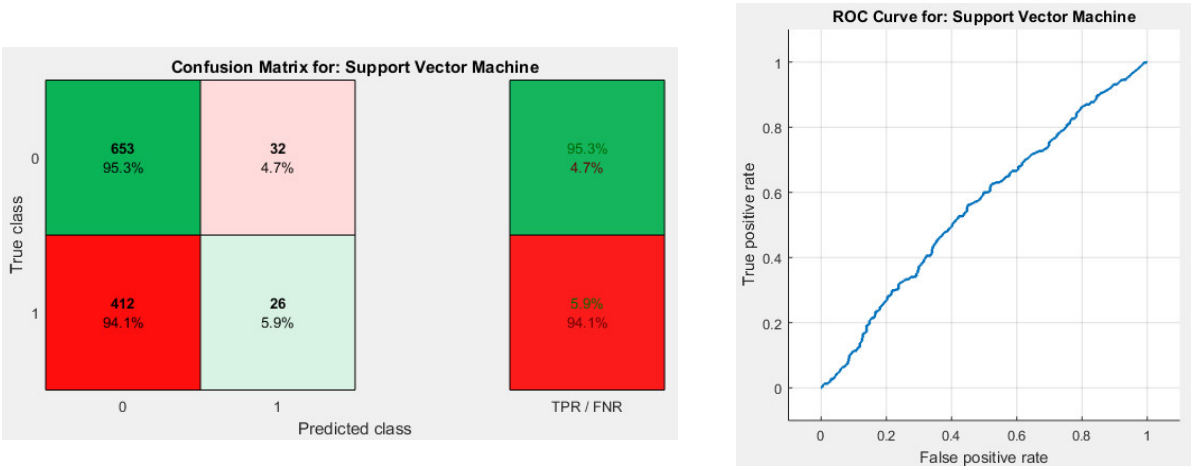



Figure 3. a) Confusion matrix of SVM classifier without feature selection on dataset 3 b) ROC of SVM classifier without feature selection. (Accuracy=60.5%, AUC=0.555601)

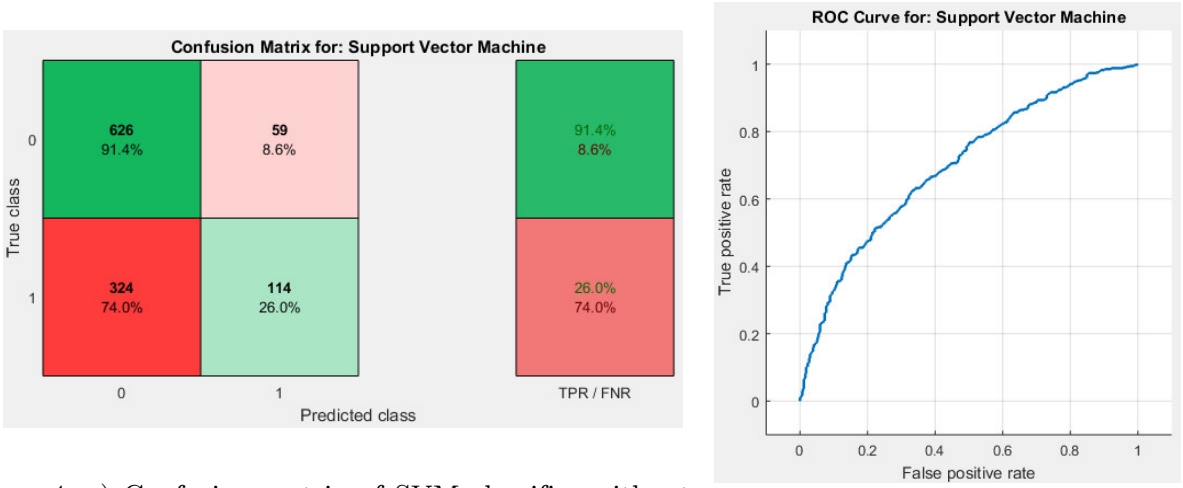



Figure 4. a) Confusion matrix of SVM classifier without feature selection on dataset 6 b) ROC of SVM classifier without feature selection. (Accuracy=65.9%, AUC=0.697957)

### 3.1.2 With Applying Feature Selection/Extraction

After using feature extraction methods, we encountered with two kind of feature vectors. 1- Feature vectors by GLRM (10 features) on raw dataset including missing values 2- feature vectors by GBFS (6 features)

**Feature vectors by GLRM:** The small dimensional feature vectors GLRM comprises 2048 samples with 10 features after performing Generalized Low Rank Model (GLRM) on dataset 1 (without removing samples that their missing entries were high). As you know the process of collecting medical data is hard and the advantage of this method is that we used all of raw data.

**Feature vectors by GBFS:** The small dimensional feature vectors GBFS includes 1123 samples with 6 features after performing GBFS on dataset 2.

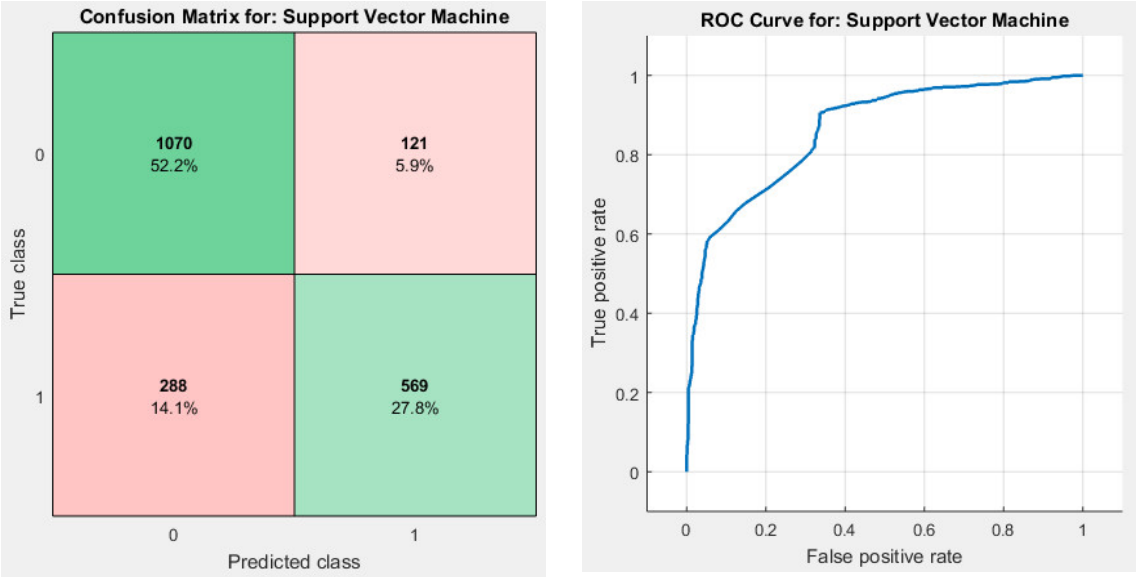
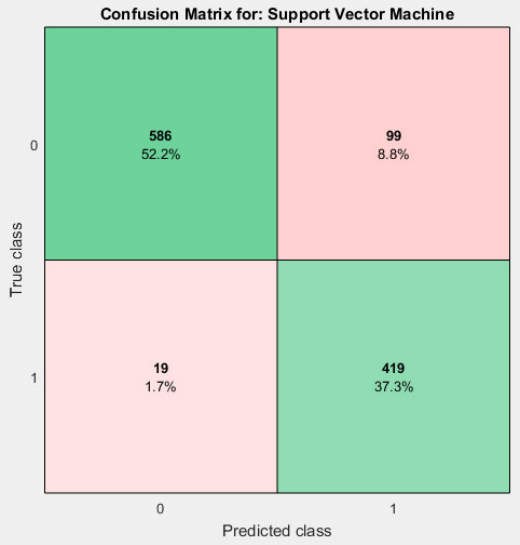



Figure 5. a) Confusion matrix of SVM classifier with applying GLRM on raw data (dataset 1), b) ROC of SVM classifier with applying GLRM (Accuracy=80% AUC=0.861902)



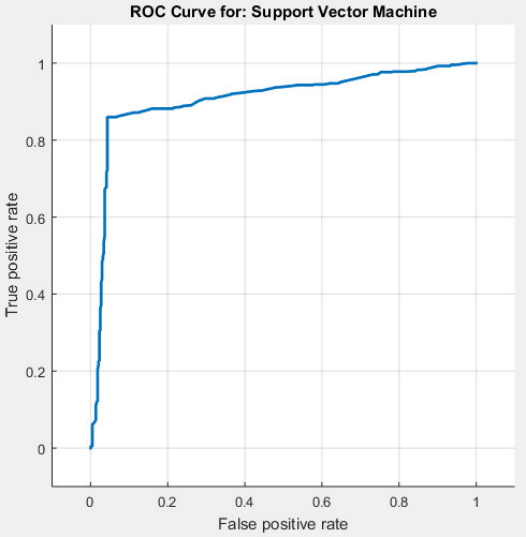




Figure 6. a) Confusion matrix of SVM classifier with applying GBFS on dataset2 b) ROC of SVM classifier with applying GBFS (Accuracy=89.5%, AUC=0.907964)

## 3.2 GBM Classifier Output Results

GBM classifier was also applied to the datasets both on before feature selection and after feature selection.

### 3.2.1 Without Applying Feature Selection/Extraction

At the first step, we only considered testing the GBM model on six datasets on which no feature selection has been applied.

As We stated before, different configuration of GBM may give different results. Figure 7 clearly explains this statement. As the number of iterations increases for the shrinkage 0.01 the accuracy goes up and for the shrinkage 0.1 the accuracy goes down. The GBM parameters are tuned between different values. The n.trees is chosen from {50, 100, 150, 200}, interaction.depth is chosen from {1, 5, 10}, and shrinkage is chosen from {0.1, 0.01}. By tuning, it means all of the combination of these parameters which make it 24 tuples, are tested and the combination with the best accuracy is chosen for final values for these parameters. The accuracy and ROC AUC of the best model is also shown in the table 4.

Table 4. Values of the GBM parameters on dataset Dataset3, accuracy and AUC of the best model.

| n.trees | interaction.depth | shrinkage | n.minobsinnode | accuracy | AUC |
|---|---|---|---|---|---|
| 200 | 5 | 0.01 | 10 | 0.6331254 | 0.8736 |

Diagram of boosting iterations versus accuracy for different shrinkages on dataset Dataset3.

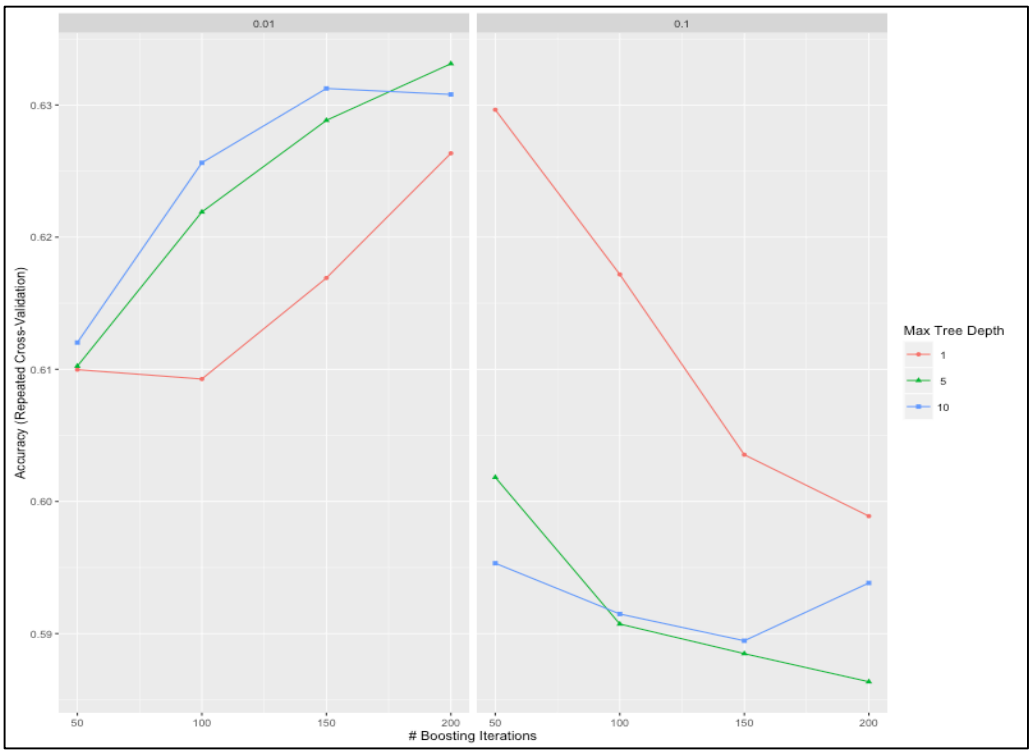



Figure 7. Diagram of boosting iterations vs accuracy for different shrinkages on dataset Dataset3. As shown, shrinkage of 0.01 has higher accuracy than shrinkage 0.1.

The main drawback of tuning with lots of parameters is the computational time. It takes a long time to even complete one epoch of GBM. It's obvious that lower values of shrinkage (i.e. 0.01, 0.001) and higher values of n.trees (i.e. 1000, 10,000) gives higher accuracy but its time complexity is huge. With this in mind, we narrowed down our parameters range to a limited one.

With the simple tuning configuration for the GBM, the results on different datasets is as follow.

Table 5: Value of GBM parameters, accuracy and AUC on different datasets,

| Parameters / datasets | n.trees | interaction.depth | shrinkage | n.minobsinnode | accuracy | ROC AUC |
|---|---|---|---|---|---|---|
| Dataset1 | 100 | 1 | 0.1 | 10 | 0.8020576 | 0.883 |
| Dataset2 | 100 | 1 | 0.1 | 10 | 0.8822131 | 0.9291 |
| Dataset3 | 50 | 1 | 0.1 | 10 | 0.6291833 | 0.5934 |
| Dataset4 | 50 | 1 | 0.1 | 10 | 0.6352301 | 0.6079 |
| Dataset5 | 50 | 2 | 0.1 | 10 | 0.8806826 | 0.9138 |
| Dataset6 | 50 | 2 | 0.1 | 10 | 0.8805024 | 0.9196 |

As shown in the above table, as we increase the threshold for features (columns) that actually remain us with more features than lower thresholds (that are stricter), the model gives high accuracy. The reason is that by taking a high threshold for features, there would be features with lots of missing values which are imputed with the same value (the most frequent one) for entire column. Thus, we have a dataset with lots of same values in lots of columns. This is a kind of bias that lead the algorithm to give high accuracy.
For example, consider comparing the result of dataset Dataset6 with dataset Dataset6. Both have the same number of samples but Dataset6 has 1679 features which is 92 more than Dataset1 1587 features. These 92 features have improved 30 percent of accuracy which is strange. For threshold 5 percent, features that have more than 56 missing values are removed and for threshold 90 percent, features that have more than 1010 missing values are removed. Thus, there are 92 features that have more than 56 and lesser than 1010 missing values which have high impact on result. It shows that how important it is to have better understanding of data and impute the data correctly.

For the comparison, ROC curve for GBM on model on Dataset3 and Dataset 6. ROC curve of the model on dataset Dataset1 has shown. The AUC is 0.5934.

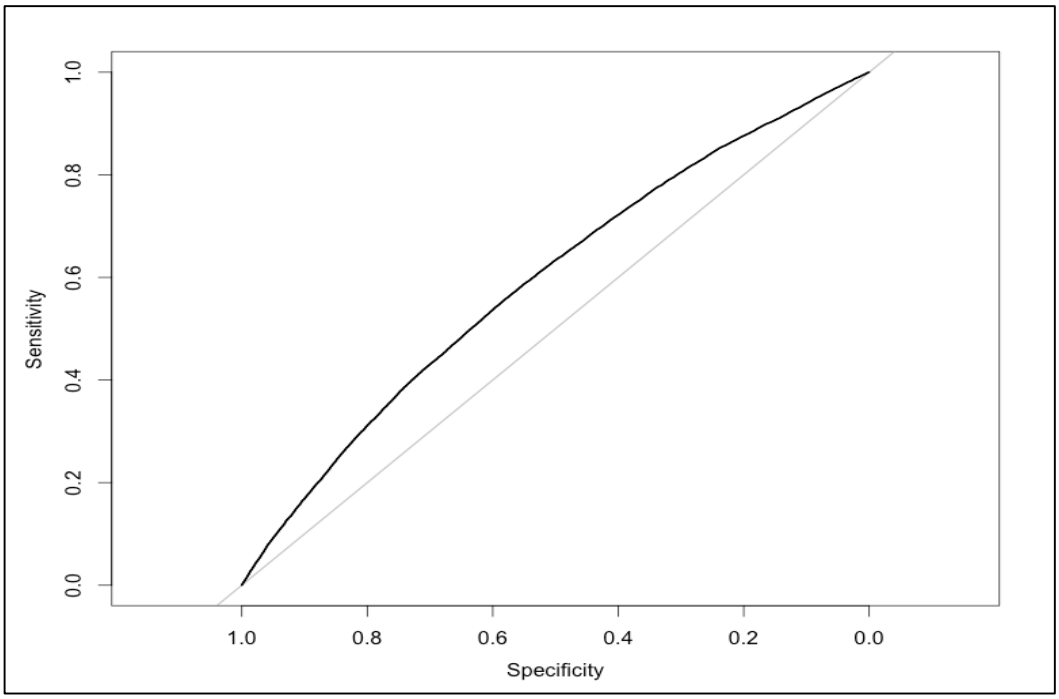



Figure 8. ROC curve of GBM on Dataset3

ROC curve of the model on dataset Dataset6 has shown. The AUC is 0.9196.

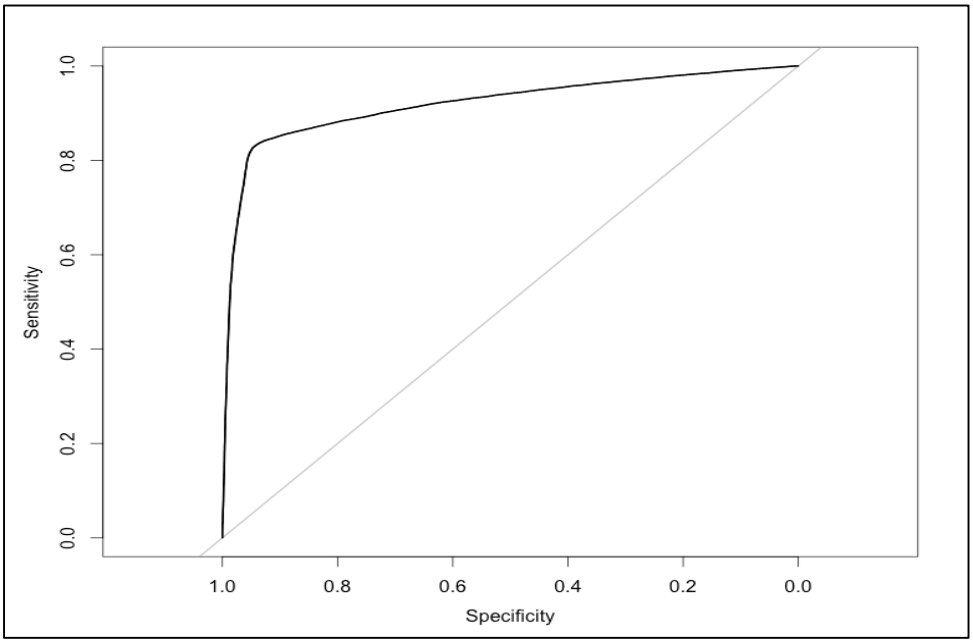



Figure 9. ROC curve of GBM on dataset Dataset6

**Features relative Influence:**

The GBM has the ability to rank the features based on their relative influence on the result and also assign an impact value to each feature. For example, for the dataset Dataset3, top 10 important features are shown in table 3.

Table 6: relative influence of top 10 important features by GBM model on dataset3

| | |
|---|---|
| smkstat | 33.381672 |
| sex | 7.317514 |
| TERT_02 | 3.918678 |
| PIN1_21 | 3.910972 |
| MTHFD2_01 | 3.762963 |
| rs12801239 | 3.709356 |
| UGT1A1_24 | 2.602280 |
| CBR1_01 | 2.447950 |
| GATA3_76 | 2.066658 |
| CCR5_04 | 1.926689 |

This table says that the feature smkstat (which is the smoke status) has higher influence on the output in comparison to other features.
For the dataset Dataset6, the top 10 important features are somehow different from those of dataset Dataset3.

Table 7: relative influence of top 10 important features by GBM model on dataset6

| | |
|---|---|
| PTCH_1315 | 23.1944431 |
| xrcc1_632 | 17.3339562 |
| XPA_6 | 17.2486417 |
| xrcc1_1678 | 11.1048828 |
| PTCH_2560 | 7.3378576 |
| rs7208422 | 6.9519078 |
| smkstat | 3.1583817 |
| rs2572023 | 2.3119687 |
| rs1867380 | 1.2317619 |
| rs4399592 | 0.6176645 |

The only common features between their top 10 most influential features is the smkstat. Even though the GBM might be biased on dataset Dataset6, the relative influence of features indicates that the smkstat is a very important and influential feature.
The relative influence of features for datasets Dataset3 and Dataset4 are almost the same. The relative influence is also the same for datasets Dataset5 and Dataset6.

### 3.2.2 With Applying Feature Selection/Extraction

The two feature selection methods have been applied datasets. After applying those methods which give datasets with lesser features, the algorithm performs faster and with different accuracy in comparison to before feature selection.

#### Result on GLRM feature extraction

The GLRM method was applied to the very first original dataset. It gave a dataset with 2048 rows and only 10 variables. First, the GBM was applied to this dataset with the same configuration that was being used for previous datasets. It was completed at only 20 seconds to model the GBM on this dataset. It had a huge difference with the previous datasets. The GBM parameters are as follow:

Table 8: The GBM parameters for GLRM generated dataset

| n.trees | interaction.depth | shrinkage | n.minobsinnode | accuracy | AUC |
|---|---|---|---|---|---|
| 50 | 1 | 0.1 | 10 | 0.8020576 | 0.883 |

The accuracy of model using this method is almost close to that of previous datasets. It actually highlights the GLRM power so that with only 10 features it could give the accuracy of almost close to that of 1587 features. Figure 10 shows the ROC curve for this model. The AUC of curve is 0.883 which is very high and almost equal to AUC of GBM on dataset with only samples removed.

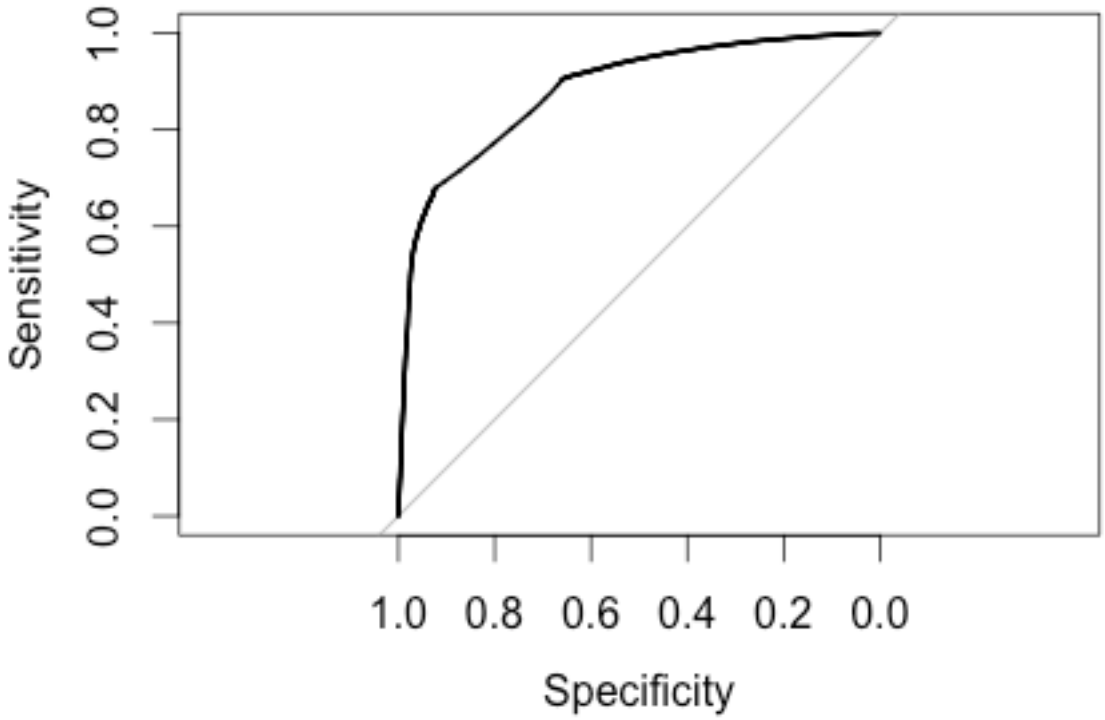



Figure 10. ROC curve of GBM model on dataset1

**Result on GBFS feature selection**

The GBFS feature selection approach was applied to the imputed datasets. The result is shown in table of accuracy and AUC in the next pages of this report.

**Between Models Comparison**

It is important that compare this model (GBM) with other models to show how different they are on these datasets. The figure 11 illustrate the ROC of GBM and SVM on dataset Dataset3. The GBM has higher ROC in comparison to SVM, although their difference is not that much. Moreover, specificity of GBM is bigger than SVM but the SVM has higher sensitivity on dataset.

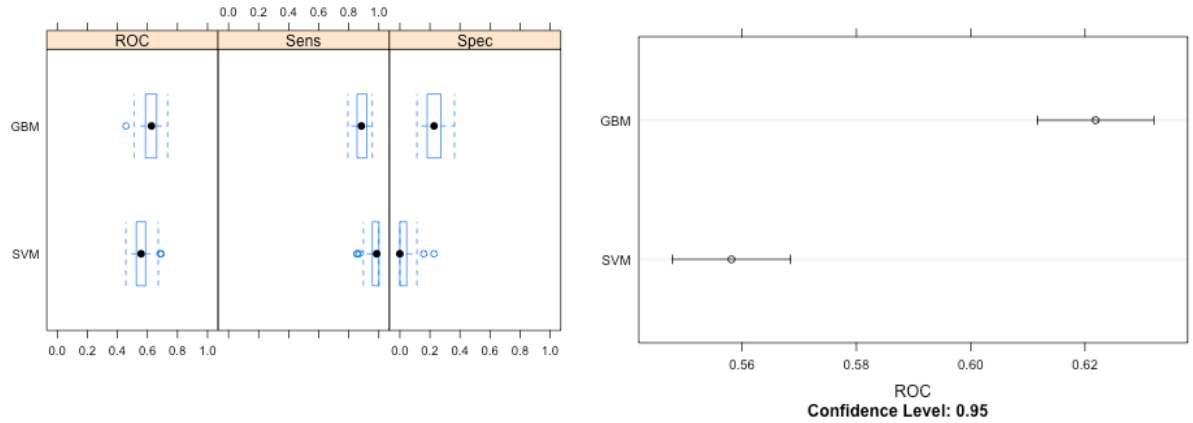



Figure 11. Boxplot and dotplot of GBM and SVM based on ROC. Test results are on dataset Dataset3

To further investigate their prediction power, their accuracy was also tested. Figure 12 displays the both boxplot and dotplot of accuracy of the GBM and SVM models on dataset Dataset3. The GBM has the accuracy of 0.6291833 and SVM has the accuracy of 0.5667.

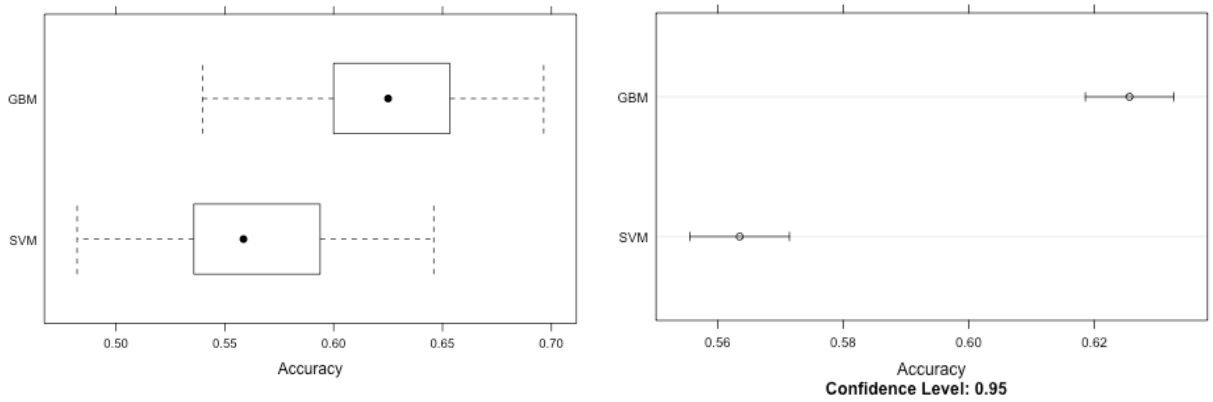



Figure 12. Boxplot and dotplot of GBM and SVM based on Accuracy. Test results are on dataset Dataset3

The results on dataset Dataset6 is shown on figure 10. On this dataset the GBM has the accuracy of 0.88 and SVM has the accuracy of 0.68. It shows that how GBM can bias in situations where data is biased or repetitive. Point is that the SVM couldn't understand the bias in data.

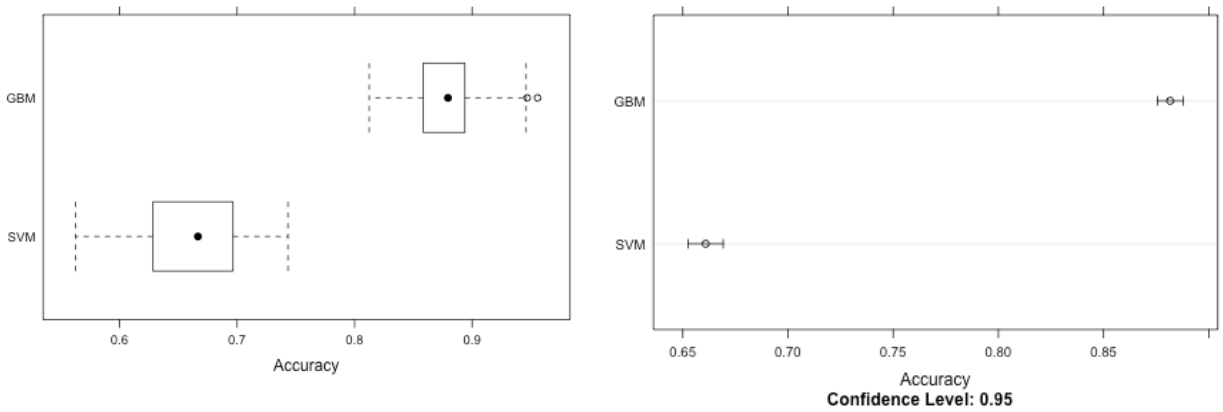



Figure 13. Boxplot and dotplot of GBM and SVM based on Accuracy. Test results are on dataset Dataset6

This diagrams are the same for datasets that have the same accuracy both in the GBM and SVM models. For example, boxplot of GBM on dataset4 is the same as that of Dataset3.
The figure 14 shows the boxplot and dotplot of these two models.

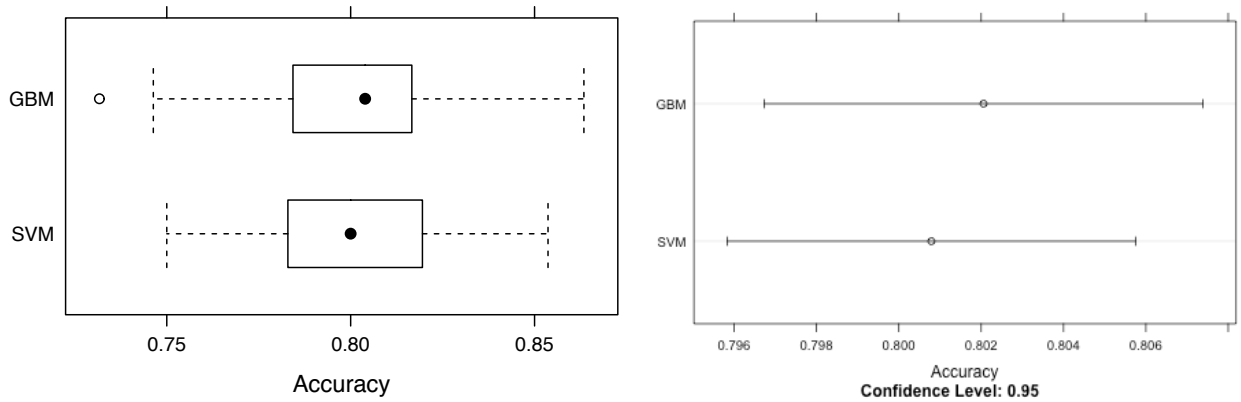



Figure 14. Boxplot and dotplot of GBM and SVM on GLRM dataset

# 4 Discussion

Three classification methods were used: KNN, SVM and GBM. The model calculations for GLRM feature extraction, GBM classifiers were carried out in R and for GBFS feature selection and SVM and KNN classifiers, we used MATLAB.

Accuracy, AUC of classifiers, ROC curves of classifies on datasets 1-6, small dimensional feature vectors GLRM and GBFS are shown in tables 9 and 10 and previous figures, respectively. In another words, the evaluation of classifiers in variation of size of feature vectors was performed. In this case, the results of the applying these methods are shown in tables 9 and 10.

As indicated in table 9 the combination of GLRM and Linear SVM or GBM classifier gives the highest overall accuracy and AUC on raw data (dataset 1) and also, the combination of GLRM and KNN classifier with k=10 shows the lowest accuracy and AUC on the raw data (2048 samples with 1694 features).
Nevertheless, KNN classifier did not perform well in our experimental setting when compared to other classifiers like SVM and GBM.

Table 9. Accuracy of different Models

| Type of input data | dataset 3 | dataset 4 | dataset 5 | dataset 6 | dataset 2 | dataset 2 | dataset 1 |
|---|---|---|---|---|---|---|---|
| Feature Selector/Extractor | | - | - | - | - | GBFS | GLRM (k=10) |
| Number of features | 1587 | 1634 | 1673 | 1679 | 1694 | 6 | 10 |
| GBM classifier | 62.9 | 63.5 | 88.06 | 88.05 | 88.22 | 89.8 | 80 |
| Linear SVM | 60.5 | 59.9 | 65.4 | 65.9 | 66.8 | 89.5 | 80 |
| KNN Classifier K=1 | 54.5 | 55.5 | 55.7 | 55.7 | 56.5 | 78.7 | 73.3 |
| KNN Classifier K=10 | 58.9 | 58.7 | 58.6 | 58.7 | 60.4 | 88.8 | 79.5 |

Table 10. AUC (Area Under Curve) of different Models

| Type of input data | dataset 3 | dataset 4 | dataset 5 | dataset 6 | dataset 2 | dataset 2 | dataset 1 |
|---|---|---|---|---|---|---|---|
| Feature Selector/Extractor | - | - | - | - | - | GBFS | GLRM (k=10) |
| Number of features | 1587 | 1634 | 1673 | 1679 | 1694 | 6 | 10 |
| GBM classifier | 0.5934 | 0.6079 | 0.9138 | 0.9196 | 0.9291 | 0.932 | 0.882 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Linear SVM** | 0.555601 | 0.546555 | 0.674759 | 0.697957 | 0.720468 | 0.907964 | 0.861902 |
| **KNN Classifier K=1** | 0.506401 | 0.51443 | 0.519183 | 0.519501 | 0.528129 | 0.754748 | 0.707696 |
| **KNN Classifier K=10** | 0.537555 | 0.532268 | 0.530632 | 0.554786 | 0.554248 | 0.900717 | 0.76408 |

# 5 Conclusion

In this work we compared the discriminatory ability of the different classifiers including SVM, KNN and GBM. This work was a challenging and yet instructive task for us because of the dataset used in this project. The missing data led us to use some approaches to deal with this kind of data. Two different classifiers were considered for the learning and prediction purpose.

In order to be able to train our classifiers in this dataset, some feature selection approaches have to be applied to this dataset. We realized that we can take advantage of GLRM at the same time for imputing missing values and producing low dimensional data. Moreover, GBFS could provide low dimensional feature vector through the imputed data.

The results show that SVM and GBM performs quite stable and reaches a high accuracy and AUC compared to KNN classifier. This may be explained by SVMs ability to minimize the structural risk when finding a unique hyper-plane with maximum margin to separate data from two classes. We think that this characteristic allows SVM the best generalization ability on unseen data compared with the other classifiers. The GBM classifier outperformed the other classifiers. Its accuracy on most of the datasets was higher than the other classifiers. Feature selection approaches gave us the best small subset of dataset which gave close results to that of full dataset. The key point about the feature selection methods was that with just subset of data we could create a predictor as close in accuracy to the predictor of original dataset. With this method we can save lots of time and still create a good predictor.

# References


[1] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

[2] Christopher JC Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998.

[3] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. John Wiley & Sons, 2012.

[4] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[5] R. Hastie, T. Tibshirani and J Friedman. *The Elements of Statistical Learning; Data Mining, Inference, and Prediction, Second edition*. Springer series in statistics, 2008.

[6] Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[7] Greg Ridgeway. Generalized boosted models: A guide to the gbm package. *Update*, 1(1):2007, 2007.

[8] Volker Roth. The generalized lasso. *Neural Networks, IEEE Transactions on*, 15(1):16–28, 2004.

[9] Madeleine Udell, Corinne Horn, Reza Zadeh, and Stephen Boyd. Generalized low rank models. *arXiv preprint arXiv:1410.0342*, 2014.

[10] Zhixiang Xu, Gao Huang, Kilian Q Weinberger, and Alice X Zheng. Gradient boosted feature selection. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 522–531. ACM, 2014.

[11] Zhixiang Xu, Kilian Weinberger, and Olivier Chapelle. The greedy miser: Learning under test-time budgets. *arXiv preprint arXiv:1206.6451*, 2012.

[12] Makoto Yamada, Leonid Sigal, Masashi Sugiyama, and Wittawat Jitkrittum. High-dimensional feature selection by feature-wise non-linear lasso. Technical report, 2012.