Predicting Breast Cancer Survivability Using Data Mining Techniques

Islamic University of Technology (IUT)

Organization of Islamic cooperation (OIC)

SUBMITTED BY

MOHAMMAD ABID (STUDENT NO: 134447)

NJAYOU YOUSSOUF (STUDENT NO: 134448)

UNDER THE SUPERVISION OF

TAREQUE MOHMUD CHOWDHURY, ASSISTAN PROFESSOR

DEPARMENT OF COMPUTER SCIENCE & ENGINEERING

ISLAMIC UNIVERSITY OF TECHNOLOGY (IUT)

Predicting Breast Cancer Survivability Using Data Mining Techniques

Islamic University of Technology (IUT)

Organization of Islamic cooperation (OIC)

SUBMITTED BY

MOHAMMAD ABID (STUDENT NO: 134447)

NJAYOU YOUSSOUF (STUDENT NO: 134448)

UNDER THE SUPERVISION OF

TAREQUE MOHMUD CHOWDHURY, ASSISTAN PROFESSOR

DEPARMENT OF COMPUTER SCIENCE & ENGINEERING

ISLAMIC UNIVERSITY OF TECHNOLOGY (IUT)

CO-SUPERVISED BY:

REDWAN KARIM SONY

Lecturer, Computer Science and Engineering

November - ©2016

CERTIFICATION

This is to certify that the work presented in this thesis paper is the outcome of the investigation carried out by the candidates under the supervision of Mr Tareque Mohmud Chowdhury Associate Professor, Department of Computer Science and Engineering (CSE) Islamic University of Technology (IUT), Boardbazar, Gazipur. It is also declared that neither of this thesis nor any part therefore has been submitted anywhere else for the award of any degree or diploma or for any publication.

Authors:

_____ _____

Njayou Youssouf Mohammad Abid

ID:134448 ID:134447

Head of CSE Department:

Pro. Muhammad Muhbob Alam

Signature:_____

Date

Approved by

Tareque Mohmud Chowdhury

Supervision and Assistant professor,
Department of computer science & Engineering,
Islamic university of technology(IUT),

Boardbazar, Gazipur-1704

Data:……..

Acknowledgement

First we wish to acknowledge our profound gratitude to Allah (SWA), the gracious and the merciful who has granted us the ability to start and finish this thesis. We also plead with him to give us more ability to continue serving Muslim Ummah.

Our most great thankfulness goes to our supervisor Mr. Tareque Mohmud Chowdhury for his endless patience, scholarly guidance, continuous encouragement, constant and energetic supervision, valuable advice that made our thesis possible to arrive at this stage. In addition, we wish to thank our dearest family members, friends, lecturers and the department of Computer Science and Engineering of Islamic University of Technology (IUT) for their support and understanding in difficult times.

We would like to convey our deep gratitude and appreciation to the Bachelor of Science students of Islamic University of Technology (IUT) for their guidance and valuable suggestion regarding this study.

Finally, our special thanks to everyone who contributed in one way or the other towards the successful completion of our thesis.

Thank you.

Abstract

Breast cancer is one of the major causes of death in women when compared to all other cancers. Breast cancer has become the most hazardous types of cancer among women in the world. In this paper we present an analysis of the prediction of survivability rate of breast cancer patients using data mining techniques. The collection of large volumes of medical data has offered an opportunity to develop prediction models for survival by the medical research community. The data used is the SEER Public-Use Data. The preprocessed data set consists of 262,423 records, which have all the available72 fields from the SEER database. After cleaning of the data set, 106,237 records were put under analysis, then we have investigated five data mining techniques: the Naïve Bayes, the back-propagated neural network, logistic regression, support vector machine and the J48 decision tree algorithms. Comparison of the performance of all these different techniques shows that the Logistic regression has a better performance of 93.07% accuracy. Afterwards, three feature reduction methods, Attribute correlation, Information gain and factor analysis for mixed dataset, were used to reduce data dimension. The result of these methods showed a better performance in time for all the above mentioned data mining techniques. It had a fluctuating accuracy in case of other methods but showed and increase to 94.35% accuracy in case of Logistic regression when factor analysis for mixed dataset was used.

Contents

Chapter 1

Introduction

1.1 Overview

Data mining, also known as knowledge discovery in databases is defined as "the extraction of implicit, previously unknown, and potentially useful information from data" [8]. It encompasses a set of processes performed automatically, whose task is to discover and extract hidden features (such as: various patterns, regularities and anomalies) from large data sets [18]. Classification is one of the most studied problems in machine learning and data mining [8]. Predicting the outcome of a disease is one of the most interesting and challenging tasks in which to develop data mining applications. An analysis of the most recent data has shown that the survival rate is 88% after 5 years of diagnosis and 80% after 10 years of diagnosis [1].

In this paper we analyze the breast cancer data available from the SEER program with the aim of developing accurate survival prediction models for breast cancer. The data analyzed in this study is from the surveillance, epidemiology and end results (SEER) breast cancer incidence data in the years of 2004-2013.

1.2 Research challenges

How to analyze the data, the data set at first was not organized in the proper way and then we clean the data then the records 106,237 were put for the analysis.

How to extract the knowledge from the data related to the diseases. One of those technique is SEER to (Surveillance Epidemiology and End Results), which is a unique, reliable and essential resource for investigating the different aspects of cancer.

Data variety, trying to accommodate the data comes from different sources and in variety different forms like (images, data text, and social numeric). The data may come in different form we should analysis in form of understanding then we can have the meaningful information to use those data for a particular task.

Dealing with huge data sets, as we know the technique of data mining is built for dealing with huge amount of data sets there are several technique to analyze those data we have used Weka software tool to process the data using different methodology.

1.2 Motivation

- Predicting the outcome of a disease is one of the most interesting and challenging tasks in which to develop data mining applications. Because
- The classification of Breast Cancer data can be useful to predict the outcome of some diseases or discover the genetic behavior of tumors.
- Predicating cancer outcome may help researchers who are working in the cure for cancer.

Chapter 2

Literature Review

2.1 Feature Selection

Feature selection, also known as variable selection, feature reduction, attribute selection or variable subset selection, is the technique of selecting a subset of relevant features for building robust learning models.

We have preprocessed the SEER data of 2004-2013 which contain 262,423 records, which have all the available72 fields for breast cancer to remove redundancies and missing information.

The resulting data set had 106,237 records, which then pre-classified into two groups of "survived" and "not survived".

- **Supervised learning** generates a function that maps inputs to desired outputs.
- **Unsupervised learning** models a set of inputs.
- **Semi-supervised learning** combines both labeled and unlabeled examples to generate an appropriate function or classifier.
- **Reinforcement learning** learns how to act given an observation of the world. Every action has some impact in the environment, and the environment provides feedback in the form of rewards that guides the learning algorithm.
- **Transduction** tries to predict new outputs based on training inputs, training outputs, and test inputs.
- **Learning to learn** learns its own inductive bias based on previous experience

2.2 Feature Selection Techniques

From a theoretical perspective, it can be shown that optimal feature selection for supervised learning problems requires an exhaustive search of all possible subsets of features of the

Chosen cardinality. If large numbers of features are available, this is impractical. For practical supervised learning algorithms, the search is for a satisfactory set of features instead of an optimal set. Feature selection algorithms can be classified into two broad categories:

Feature ranking (FR) also called feature weighting [2], assesses individual features and assigns them weights according to their degrees of relevance. Many researches have been done with feature ranking as the base meth.

2.3 Related Works

Bellachia et al [2] uses the SEER data to compare three prediction models for detecting breast cancer. They have reported that C4.5 algorithm gave the best performance of 86.7% accuracy.

Delen et al [6] in their work preprocessed the SEER data for to remove redundancies and missing information. They have compared the predictive accuracy of the SEER data on three prediction models indicated that the decision tree (C5) is the best predictor with 93.6% accuracy on the holdout sample.

Endo et al [3] implemented common machine learning algorithms to predict survival rate of breast cancer patient. This study is based upon data of the SEER program with high rate of positive examples (18.5 %). Logistic regression had the highest accuracy, artificial neural network showed the highest specificity and J48 decision trees model had the best sensitivity

Kotsiantis et.al. [13] did a work on Bagging, Boosting and Combination of Bagging and Boosting as a single ensemble using different base learners such as C4.5, Naïve Bayes, OneR and Decision Stump. These were experimented on several benchmark datasets of UCI Machine Learning Repository

Chapter 3

Proposed method

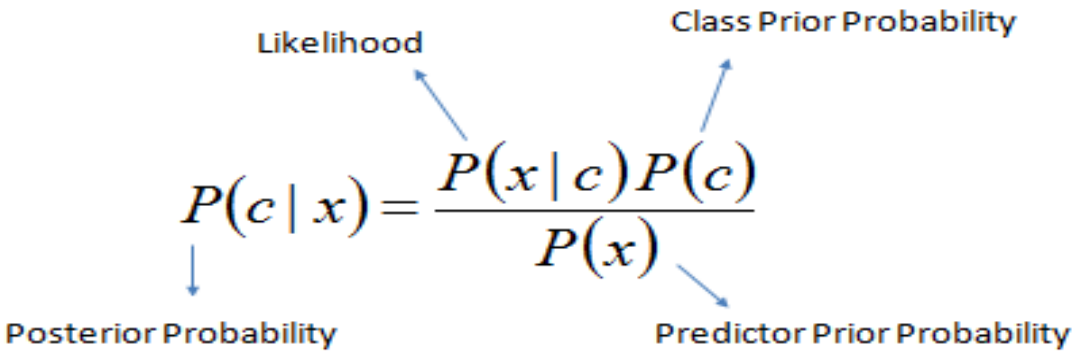
3.1 Overall Concept

We have used five methods in our experiments which are Naïve bays, Artificial Neural Network ,logistic regression, support vector machine and Decision tree J48 we have compared the results between them among these logistic regression has better performance.

Building accurate and efficient classifiers for large databases is one of the essential tasks of data mining and machine learning research. Building effective classification systems is one of the central tasks of data mining.

3.2 Naïve Bayes

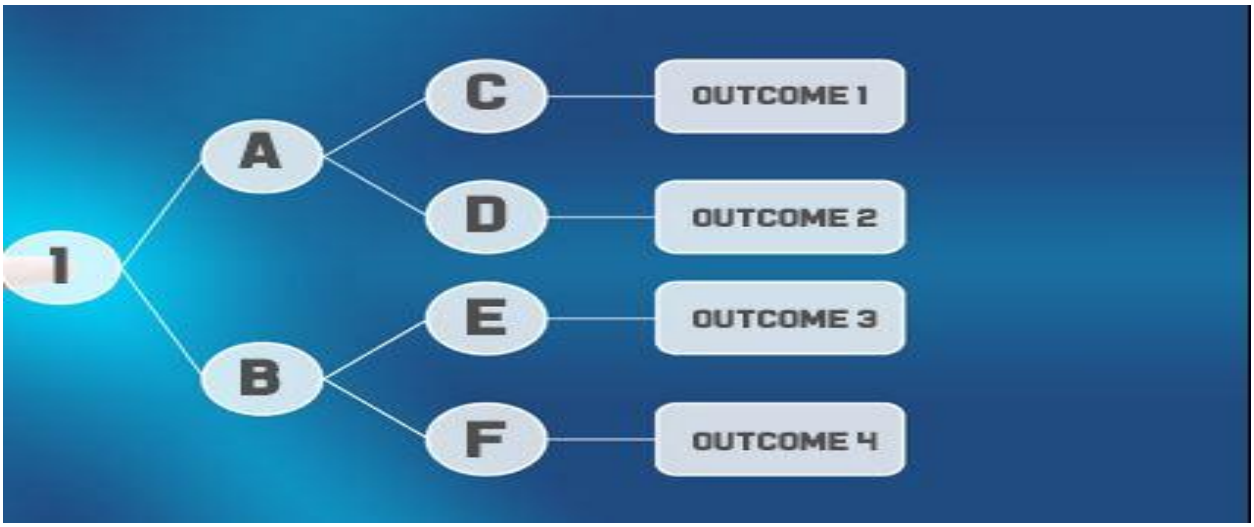
The Naive Bayes is a quick method for creation of statistical predictive models [16]. NB is based on the Bayesian theorem. This classification technique analyses the relationship between each attribute and the class for each instance to derive a conditional probability for the relationships between the attribute values and the class. During training, the probability of each class is computed by counting how many times it occurs in the training dataset. This is called the "prior probability" P(C=c). In addition to the prior probability, the algorithm also computes the probability for the instance x given c with the assumption that the attributes are independent. This probability becomes the product of the probabilities of each single attribute. The probabilities can then be estimated from the frequencies of the instances in the training set.



$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

3.4 Decision Tree J48

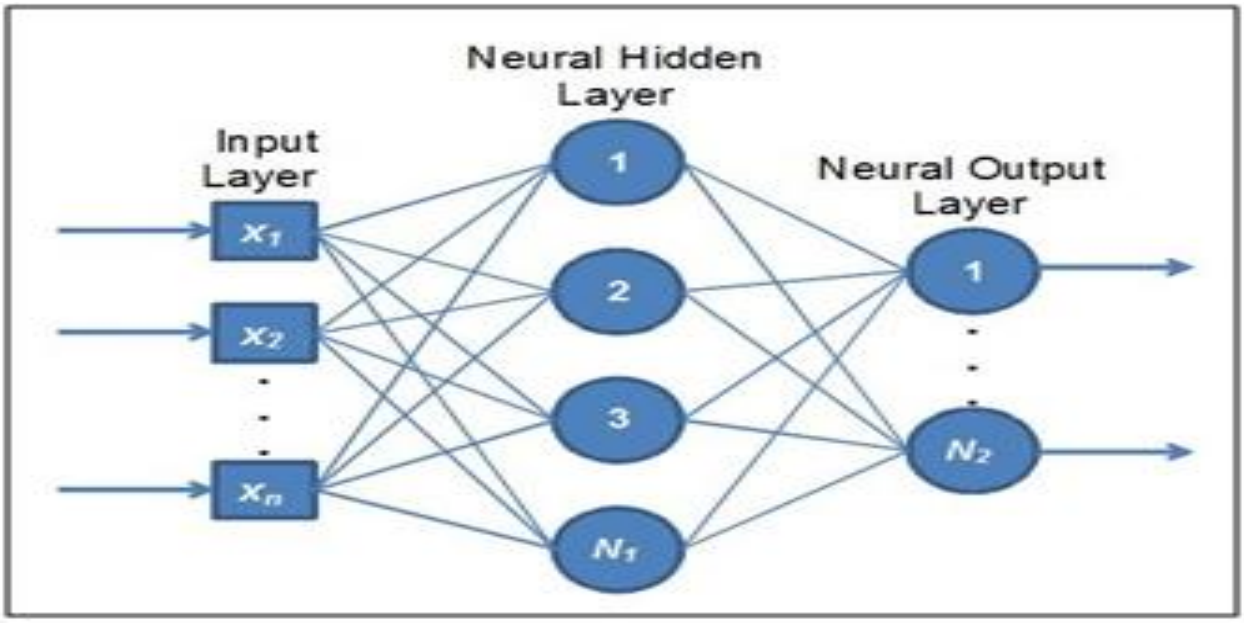
Decision tree is a classifier in the form of a tree structure where each node is either a leaf node, indicating the value of the target attribute or class of the examples, or a decision node, specifying some test to be carried out on a single attribute-value, with one branch and sub-tree for each possible outcome of the test. A decision tree can be used to classify an example by starting at the root of the tree and moving through it until a leaf node is reached, which provides the classification of the instance.



3.5 Artificial Neural Network

Neural networks are capable of modeling extremely complex, typically non-linear functions [10]. It is made up of structure or a network of numerous interconnected units (artificial neurons).

Each of these units consists of input/output characteristics that implement a local computation or function. The function could be a computation of weighted sums of inputs which produces an output if it exceeds a given threshold. The output (whatever the result), could serve as an input to other neurons in the network. This process iterates until a final output is produced.



3.6 Logistic Regression

Logistic regression is considered as the standard statistical approach to modeling binary data [16]. It is a better alternative for a linear regression which assigns a linear model to each of the class and predicts unseen instances basing on majority vote of the models. During prediction,

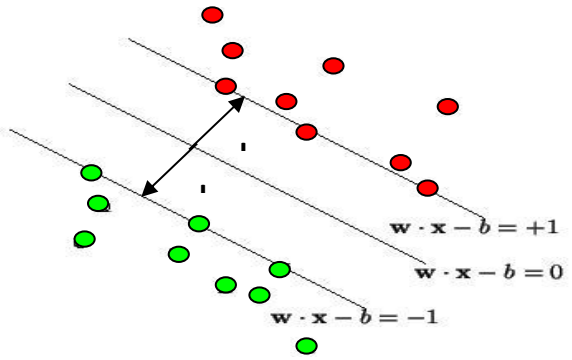
instead of predicting the point estimate of the event itself, it builds a model to predict the odds of its occurrence. In two class problem for example, when the odds are greater than 50%, then the case is assigned to the class designated as “1” for YES and “0” for “YES” and “NO” instead. The given equations, equ1 and equ2, are the linear regression and logistic regression respectively.

$$u = A + B_1 X_1 + B_2 X_2 + \cdots + B_K X_K \quad \text{..... equ1}$$

$$\hat{\pi}_i = \frac{e^{b_0 + b_1 X_1}}{1 + e^{b_0 + b_1 X_1}} \quad \text{..... equ2}$$

3.7 SUPPORT VECTOR MACHINE (SVM)

Support Vector Machine are a set of related supervised learning methods that analyze data and recognize patterns, used for classification and regression analysis. Support Vector Machine is an algorithm that attempts to find a linear separator (hyper-plane) between the data points of two classes in multidimensional space. Support Vector Machine represents a learning technique which follows principles of statistical learning theory [14]. Generally, the main idea of Support Vector Machine comes from binary classification, namely to find a hyperplane as a segmentation of the two classes to minimize the classification error. The Support Vector Machine finds the hyperplane using support vectors (training tuples) and margins (support vectors). The Sequential Minimal Optimization (SMO) algorithm is a simple and fast method for training a Support Vector Machine.



Chapter 4:

Experimental Analysis

We have used java programming language to extract and clean the data from SEER Public-Use data. Weka (3.8) tool was used to apply all the classifiers algorithms I this paper.

4.1 Dataset Details

The data set on breast cancer on which we have applied the mentioned classification algorithms was from the SEER, a unique and reliable data source on cancer. The data set consisted of records from 2004-20013, it consisted of 262,423 cancer incidence. The 16 important fields were extracted from the 72 field that the SEER data set provides. This fields were considered the important fields that are affecting and is concerned with breast cancer. The fields are as follows:

Nominal variable name	Number of distinct values
Race	29
Marital status	6
Primary site code	9
Histologic type	104
Behavior code	2
Grade	4

Extension of tumor	39
Lymph node involvement	38
Site specific surgery code	47
Radiation	8
Stage of cancer	13

Table1: Nominal attributes of dataset

Numeric variable name	**Mean**	**Std. Dev.**	**Range**
Age	59.209	13.228	12-107
Tumor size	54.185	176.324	0-998
Number of positive nodes	26.504	42.641	0-98
Number of nodes	6.581	12.879	0-98
Number of primaries	0.433	0.804	0-7

Table2: Numeric attributes of dataset.

The survivability attribute was added in the pre-classified data. To consider a cancer patient record as survived or not survived, a record was classified as survived if the survival time record was greater than or equal to 60 months and the vital status record was alive. Then if the survival time record was less than 60 months and the cause of death was breast cancer, the patient instance was classified as not survived. Otherwise, the record was ignored.

// Setting the survivability dependent variable for 60

// months threshold

If STR ≥60 months and VSR is alive then

the record is pre-classified as “survived”

else if STR < 60 months and COD is breast cancer, then

the record is pre-classified as “not survived”

else

Ignore the record

end if

After the pre-classification, 106,237 records were left. Of this number of records, 92,812 patient records were classified as survived and 13,425 were classified as not survived.

Class	Number of instances	Percentage
0 Not survived	13,425	12.6
1 survived	92,812	87.4
Total	106,237	100

Table3: Dataset composition

This shows 87.4 % survival rate in breast cancer, which is approximately equal to the survival rate of 89 % from 2006 to 2013 according to cancer association.

After the preprocessing step, a common analysis would be determining the effect of the attributes on the prediction, or attribute selection. We used the information gain measure to rank the attributes due to the fact that it is a common method and the C4.5 decision tree technique utilizes this measure.

The figure bellow shows the ranked survivability attributes of our data as calculated by the Weka toolkit. It clearly shows that Extension of Tumor and cancer stage has a higher rank than the Tumor Size.

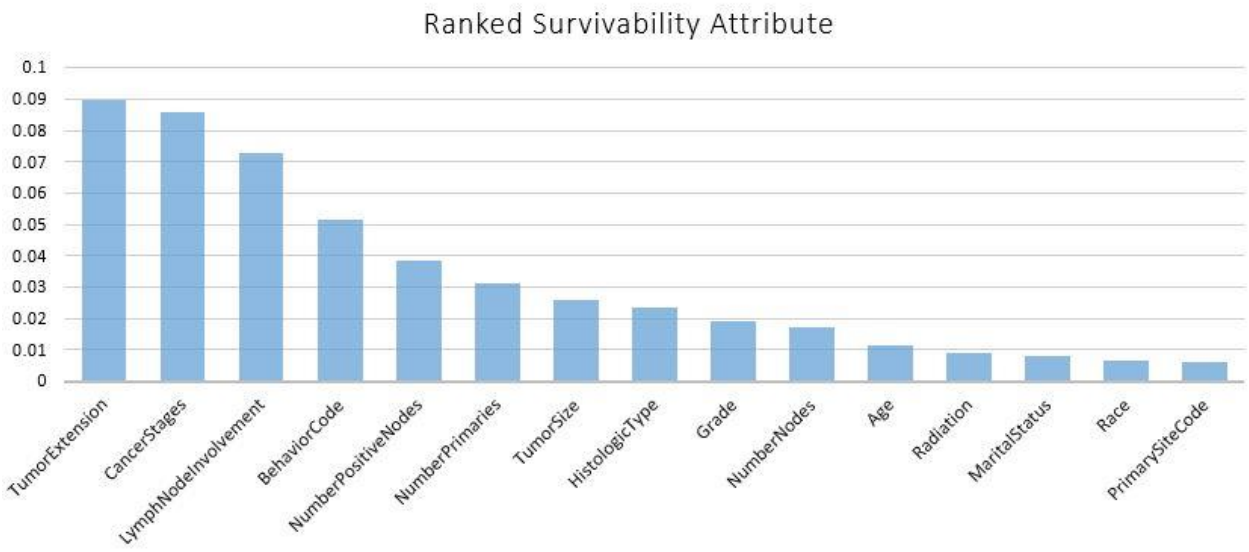


Figure 1: Rank of attributes

4.2 Performance Analysis

We will use the performance metrics of accuracy, precision (or Specificity), recall (or Sensitivity) ROC (receiver oriented characteristic) curve and time taken to build the model to compare the three techniques. In order to have a fair measure of the performance of the classifier; we used a cross validation with 10 folds. In its most elementary form, cross-validation consists of dividing the data into k subgroups. Each subgroup is predicted via the classification rule constructed from the remaining (k-1) subgroups, and the estimated error rate is the average error rate from these k subgroups. This is to avoid overfitting and also, the error rate is estimated in an unbiased way. The final classifier rule is calculated from the entire data set.

4.2.1 Decision Tree Performance Analysis

For each of the mentioned classification algorithms, a performance analysis was carried out.

Decision Tree

The J48 which is a java implementation of the c4.5 algorithm was carried out. The performance and confusion matrix was as given below.

Actual	Predicted	
	Survived	Not Survived
Survived	91251	1561
Not Survived	6043	7382

Time taken to build model: **10.3 seconds**

Correctly Classified Instances	98633	**92.8424 %**
Incorrectly Classified Instances	7604	7.1576 %
Kappa statistic	0.6218	
Mean absolute error	0.1141	
Root mean squared error	0.2391	
Relative absolute error	51.6621 %	
Root relative squared error	71.9661 %	
Total Number of Instances	106237	

Table 4: Decision Tree confusion matrix.

4.2.2 Naïve Bayes Performance Analysis

The Naïve Bayes algorithm showed a less performance accuracy but the time taken to build the model was considerably faster compared to all the other algorithms.

Time taken to build model: **0.32 seconds**

Correctly Classified Instances	96554	**90.8855 %**

Incorrectly Classified Instances	9683	9.1145 %
Kappa statistic	0.615	
Mean absolute error	0.1018	
Root mean squared error	0.2733	
Relative absolute error	46.123 %	
Root relative squared error	82.2391 %	
Total Number of Instances	106237	

Actual	**Predicted**	
	Survived	Not Survived
Survived	86836	5976
Not Survived	3707	9718

Table 5: Naïve Bayes Confusion Matrix.

4.2.3 Artificial Neural Network Performance Analysis

The Artificial Neural Network with back propagation showed a better accuracy prediction but the time taken to build the model was way too high.

Time taken to build model**: 69995.93 seconds**

Correctly Classified Instances	97842	**92.0979 %**
Incorrectly Classified Instances	8395	7.9021 %
Kappa statistic	0.6002	
Mean absolute error	0.081	
Root mean squared error	0.2803	
Relative absolute error	36.6756 %	
Root relative squared error	84.3513 %	
Total Number of Instances	106237	

Actual	Predicted	
	Survived	Not Survived
Survived	90260	2552
Not Survived	5843	7582

Table 6: Artificial Neural Network Confusion Matrix

4.2.4 Logistic Regression Performance Analysis

Logistic regression was more accurate among all methods that we have used.

Time taken to build model: **52.96 seconds**

Correctly Classified Instances 99029 **93.2152 %**

Incorrectly Classified Instances 7208 6.7848 %

Kappa statistic 0.6516

Mean absolute error 0.1047

Root mean squared error 0.2286

Relative absolute error 47.4294 %

Root relative squared error 68.799 %

Total Number of Instances 106237

Actual	Predicted	
	Survived	Not Survived
Survived	91059	1753
Not Survived	5455	7970

Table 7: Logistic Regression Confusion Matrix

4.2.5 Support Vector Machine Performance Analysis

Support vector machine doesn't has good accuracy.

Time taken to build model: 5729.55 seconds

Correctly Classified Instances 98314 92.5421 %

Incorrectly Classified Instances 7923 7.4579 %

Kappa statistic 0.6114

Mean absolute error 0.0746

Root mean squared error 0.2731

Relative absolute error 33.7758 %

Root relative squared error 82.1909 %

Total Number of Instances 106237

Actual	Predicted	
	Survived	Not Survived
Survived	7412	6013
Not Survived	1910	90902

Table 8: Support vector machine Confusion Matrix

4.3 Comparative Analysis

The comparative analysis on the different algorithm was performed and demonstrated in the figure below. The performance criterion was accuracy, precision, recall, and time taken to build the model.

Algorithms	**Results(106,237)**				
	Accuracy	Sensitivity	Specificity	ROC Area	Time(s)
Decision Tree	92.84 %	**0.983**	0.938	0.914	10.3
Naïve Bayes	90.88 %	0.936	**0.959**	**0.926**	**0.32**
ANN	92.08 %	0.973	0.939	0.797	69998.78

Logistic regression	**93.07%**	0.981	0.942	0.935	191.25
Support Vector Machine	92.35%	0.983	0.933	0.748	2223.75

Table 9: comparison of all algorithms

4.3.1 Accuracy:

The accuracy performance criterion measures how much of the instances were correctly classified by the classification technique used.

$$\text{Acc} = \frac{TP+TN}{P+N} \quad \text{.....} \quad \text{eq3}$$

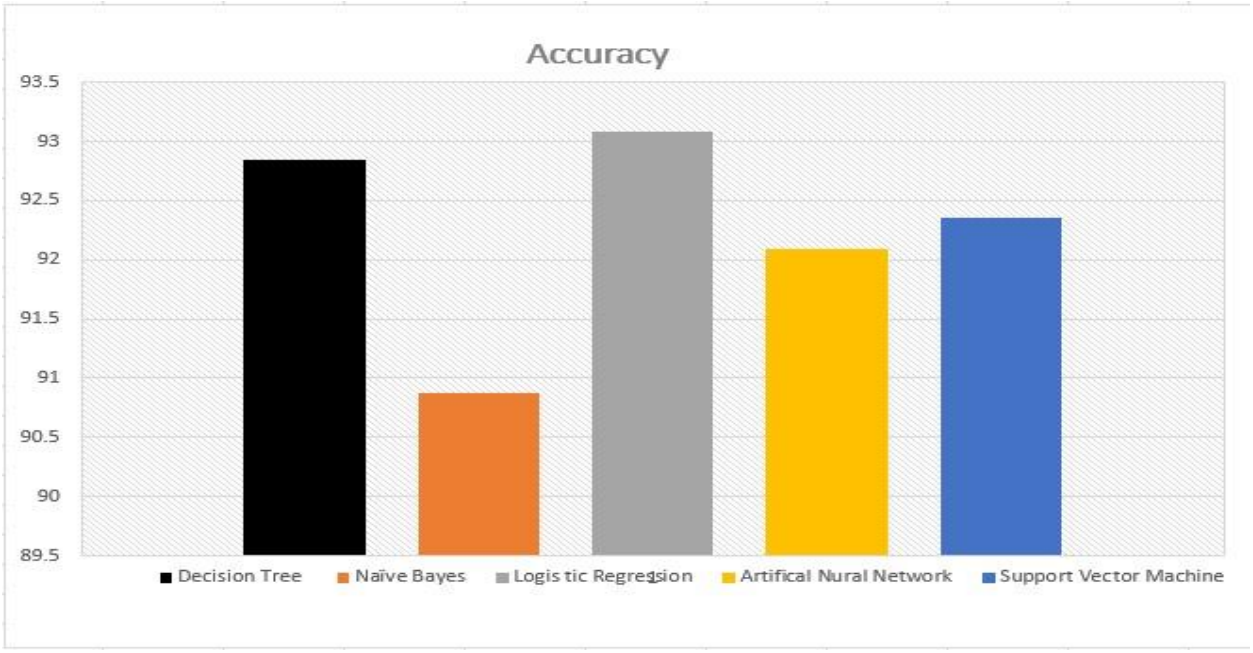


Figure 3: Accuracy of algorithms

4.3.2 Recall or Sensitivity:

The fraction of those instances that are actually positive were predicted positive.

$$\text{Recall} = \frac{TP}{TP+FN} \quad \text{..... eq4}$$

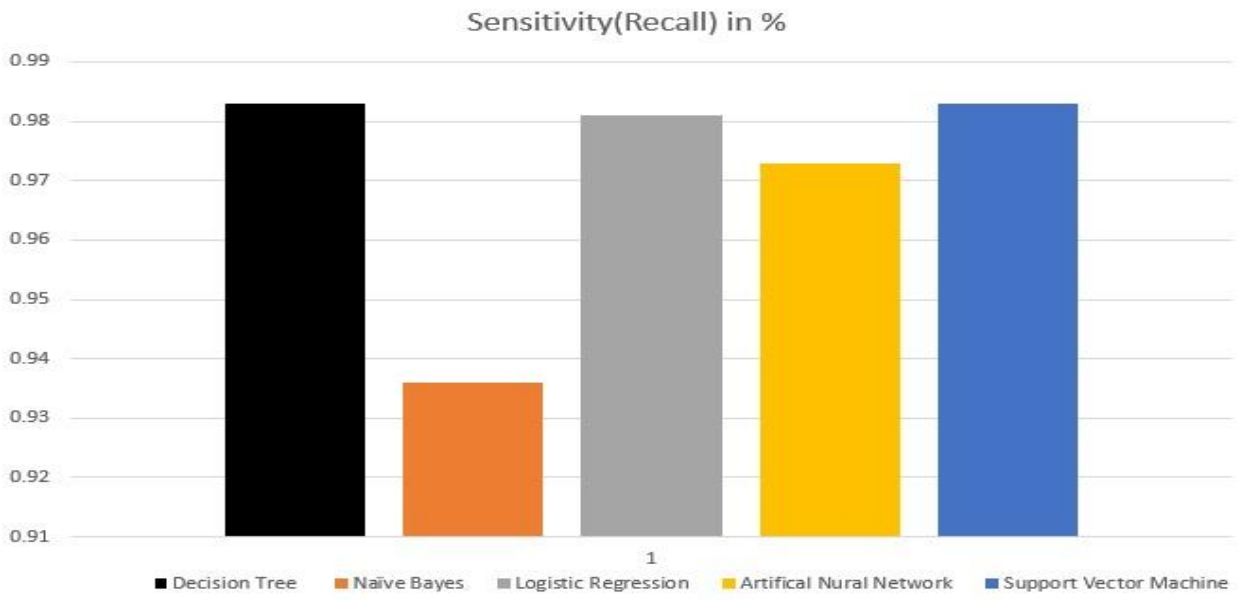


Figure 4: Sensitivity of algorithms

4.3.3 Precision or Specificity:

What fraction of those predicted positive are actually positive.

$$\text{Precision} = \frac{TP}{TP+FP} \quad \text{..... eq5}$$

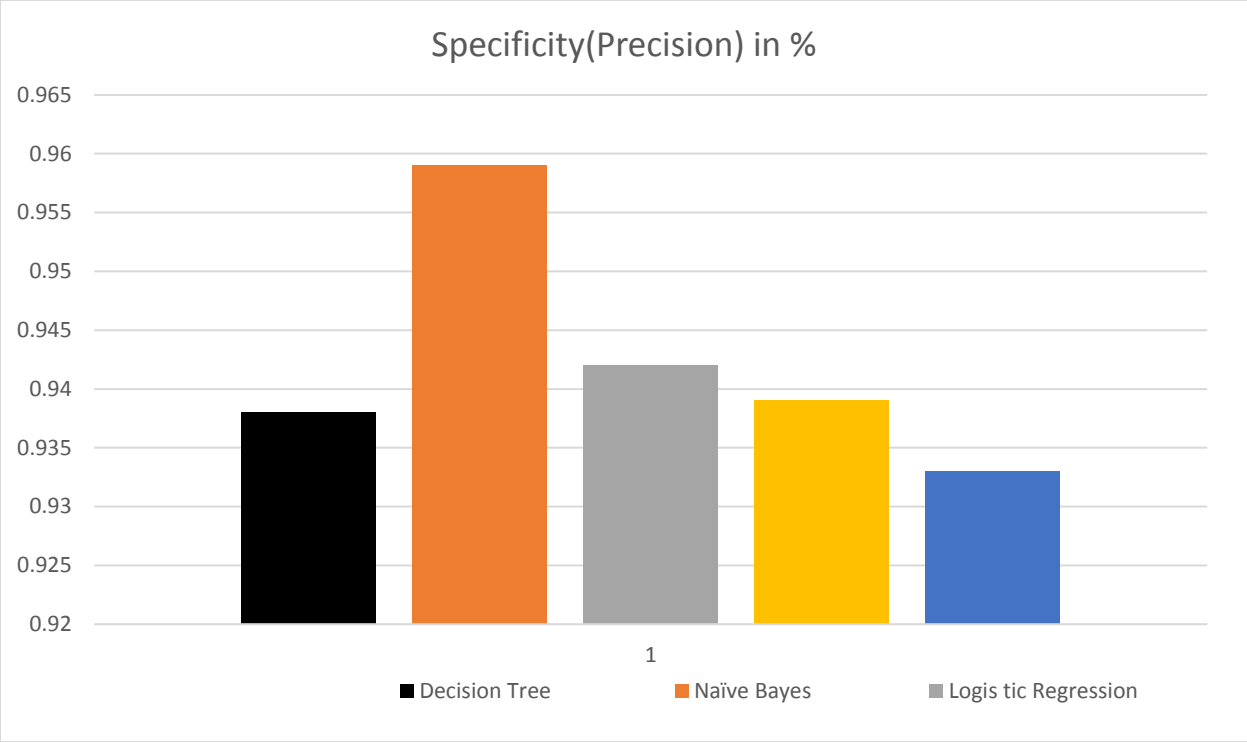


Figure 5: Specificity of algorithms

4.3.4 Time:

In this performance comparison time was a criterion and the time an algorithm takes to build its model is important.

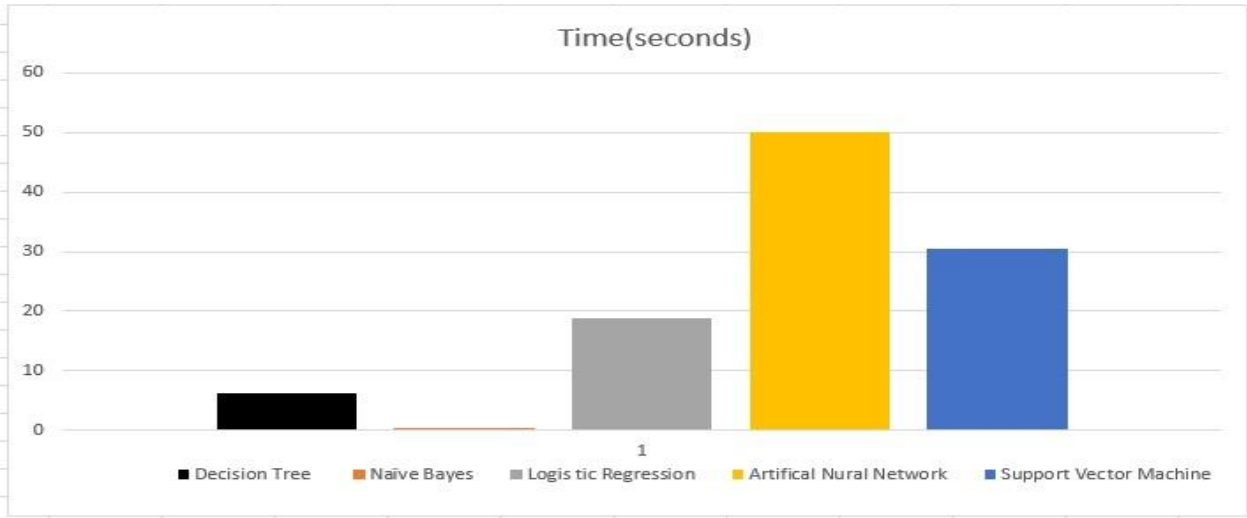


Figure 6: Time to build each model.

4.4 ROC CURVE

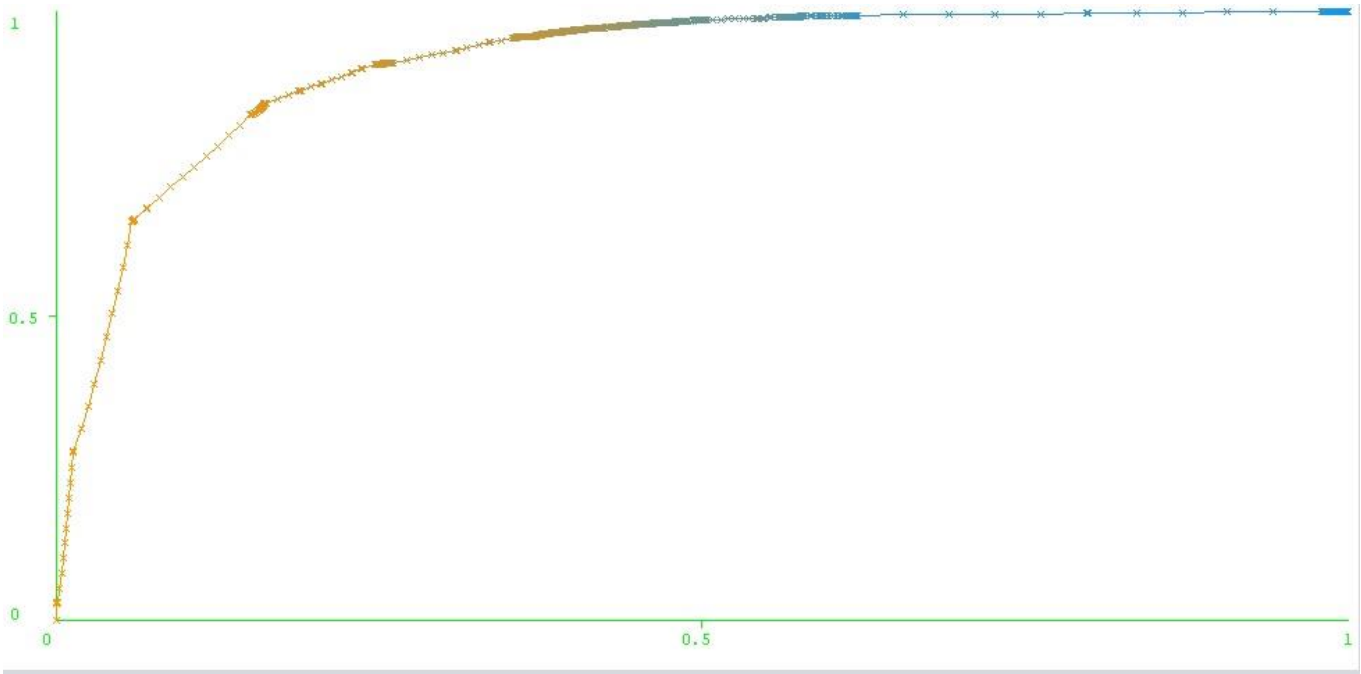
The **ROC curve** is a fundamental tool for diagnostic test evaluation. In a **ROC curve** the true positive rate (Sensitivity) is plotted in function of the false positive rate (100-Specificity) for different cut-off points of a parameter.

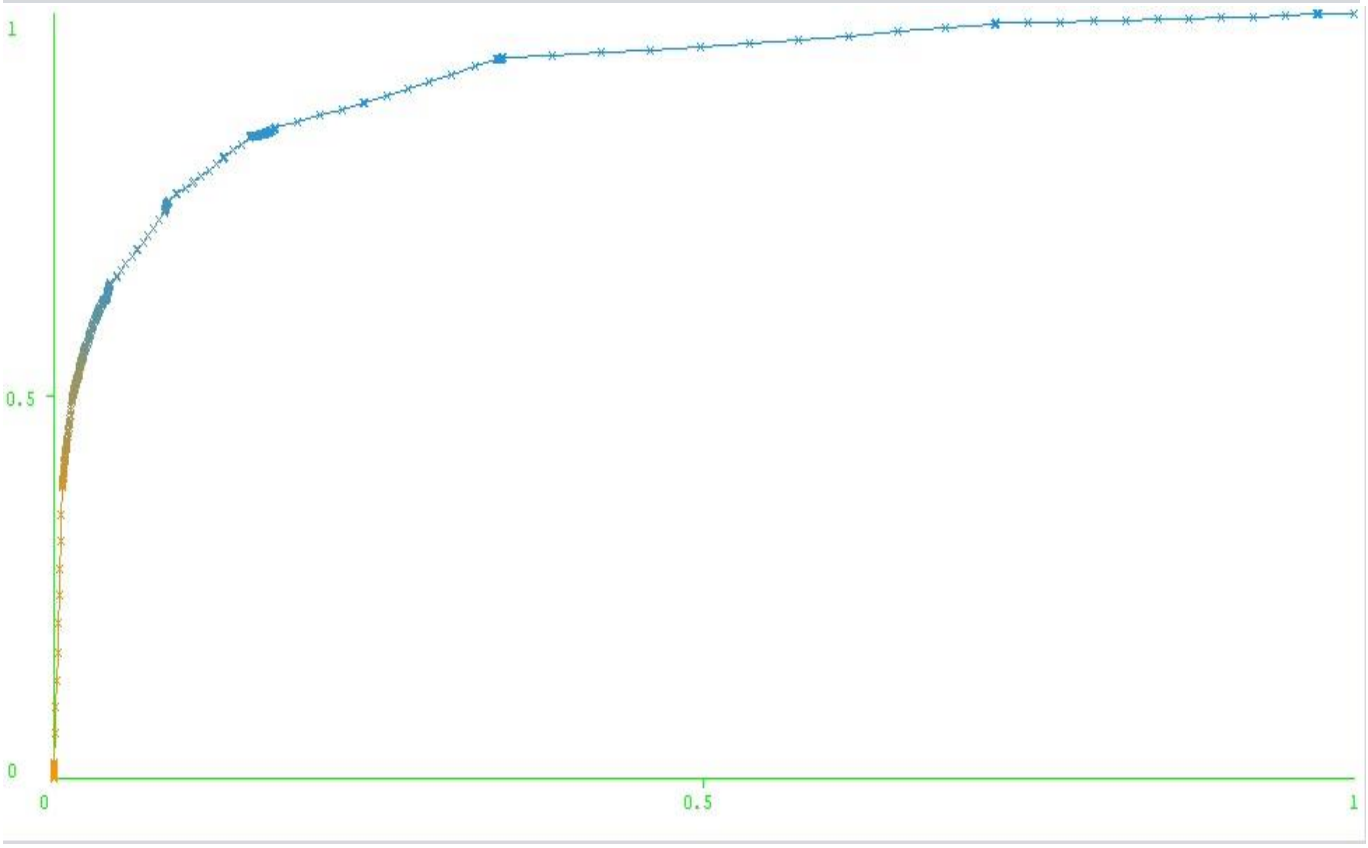
This type of graph is called a **Receiver Operating Characteristic curve** (or ROC curve.) It is a plot of the true positive rate against the false positive rate for the different possible cut points of a diagnostic test.

An ROC curve demonstrates several things:

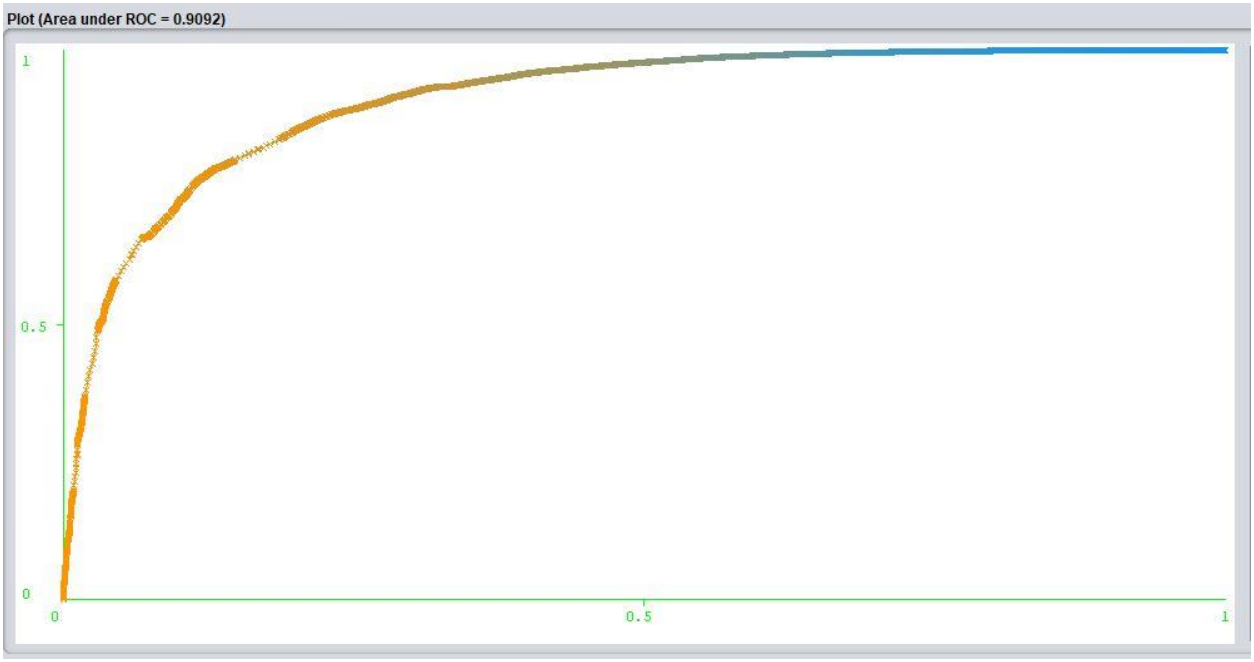
1. It shows the tradeoff between sensitivity and specificity (any increase in sensitivity will be accompanied by a decrease in specificity).
2. The closer the curve follows the left-hand border and then the top border of the ROC space, the more accurate the test.
3. The closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test.
4. The slope of the tangent line at a cut point gives the likelihood ratio (LR) for that value of the test. You can check this out on the graph above. Recall that the LR for T4 < 5 is 52. This corresponds to the far left, steep portion of the curve. The LR for T4 > 9 is 0.2. This corresponds to the far right, nearly horizontal portion of the curve.
5. The area under the curve is a measure of text accuracy.

DECISION TREE ROC

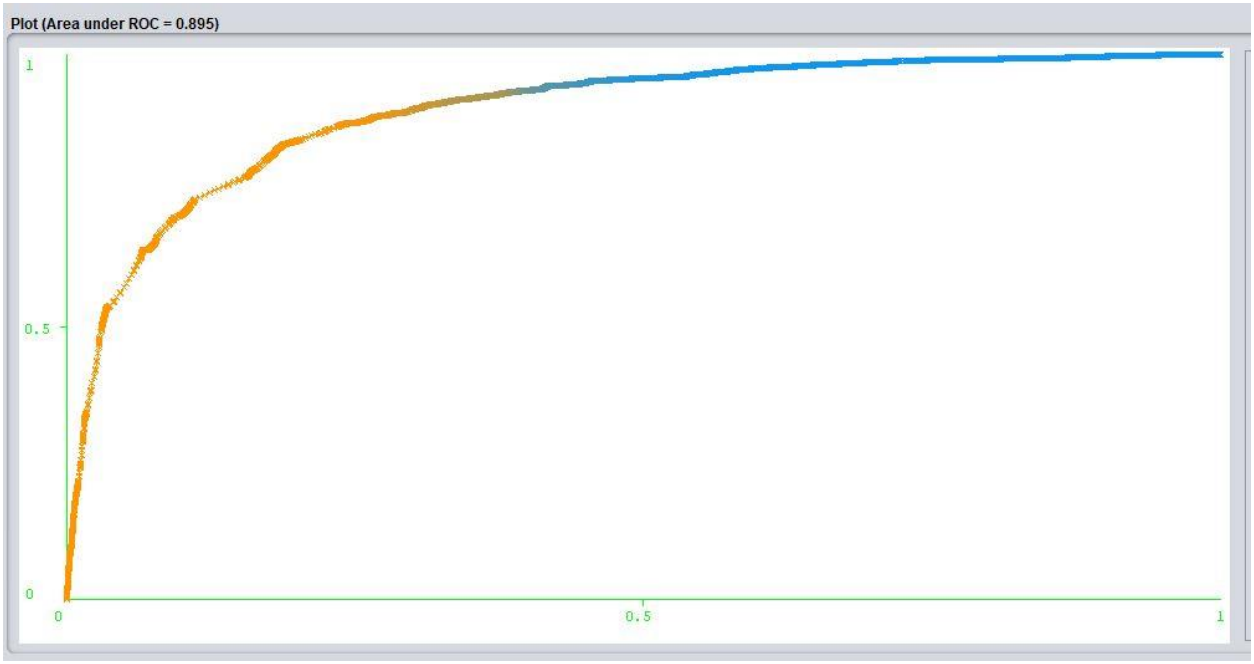




LOGISTIC REGRESSION ROC



NAÏVE BAYES



Chapter 5

5.1 Feature Reduction

Feature selection is critical to building a good model for several reasons. One is that feature selection implies some degree of *cardinality reduction*, to impose a cutoff on the number of attributes that can be considered when building a model. Data almost always contains more information than is needed to build the model, or the wrong kind of information. For example, you might have a dataset with 500 columns that describe the characteristics of customers; however, if the data in some of the columns is very sparse you would gain very little benefit from adding them to the model, and if some of the columns duplicate each other, using both columns could affect the model.

Not only does feature selection improve the quality of the model, it also makes the process of modeling more efficient. If you use unneeded columns while building a model, more CPU and memory are required during the training process, and more storage space is required for the completed model. Even if resources were not an issue, you would still want to perform feature selection and identify the best columns, because unneeded columns can degrade the quality of the model in several ways:

- Noisy or redundant data makes it more difficult to discover meaningful patterns.
- If the data set is high-dimensional, most data mining algorithms require a much larger training data set.

During the process of feature selection, either the analyst or the modeling tool or algorithm actively selects or discards attributes based on their usefulness for analysis. The analyst might perform feature engineering to add features, and remove or modify existing data, while the machine learning algorithm typically scores columns and validates their usefulness in the model.

In short, feature selection helps solve two problems: having too much data that is of little value, or having too little data that is of high value. Your goal in feature selection should be to identify the minimum number of columns from the data source that are significant in building a model.

5.2 Analysis of different algorithms after feature reduction

Correlation attribute

Actual	Predicted	
	Survived	Not Survived
Survived	2033	90779
Not Survived	7336	6089

For 8 attributes of Decision tree j48 confusion matrix.

Actual	Predicted	
	Survived	Not Survived
Survived	1432	91380
Not Survived	7136	6289

For 10 attributes of Decision tree j48 confusion matrix.

Actual	Predicted	
	Survived	Not Survived
Survived	1564	91248
Not Survived	7355	6070

For 13 attributes of Decision tree j48 confusion matrix.

Actual	Predicted	
	Survived	Not Survived
Survived	6185	86627
Not Survived	9109	4316

For 8 attributes of Naïve Bayes confusion matrix.

Actual	Predicted	
	Survived	Not Survived
Survived	6047	86765
Not Survived	9438	3987

For 10 attributes of Naïve Bayes confusion matrix.

Actual	Predicted	
	Survived	Not Survived
Survived	6007	86805
Not Survived	9610	3815

For 13 attributes of Naïve Bayes confusion matrix.

Gain Ratio Attributes

Actual	Predicted	
	Survived	Not Survived
Survived	1532	91280
Not Survived	6855	6570

For 8 attributes of Decision tree j48 confusion matrix.

Actual	Predicted	
	Survived	Not Survived
Survived	1909	90903
Not Survived	7476	5949

For 10 attributes of Decision tree j48 confusion matrix.

Actual	Predicted	
	Survived	Not Survived
Survived	1589	91223
Not Survived	7392	6033

For 13 attributes of Decision tree j48 confusion matrix.

Actual	Predicted	
	Survived	Not Survived
Survived	7018	85794
Not Survived	9754	3671

Naïve Bayes confusion matrix For 8 attributes.

Actual	Predicted	
	Survived	Not Survived
Survived	6846	85966
Not Survived	9799	3626

Naïve Bayes confusion matrix For 10 attributes.

Actual	Predicted	
	Survived	Not Survived
Survived	6174	86638
Not Survived	9655	3770

Naïve Bayes confusion matrix For 13 attributes.

Actual	Predicted	
	Survived	Not Survived
Survived	2023	90789
Not Survived	6847	6578

Logistic regression confusion matrix For 8 attributes.

Actual	Predicted	
	Survived	Not Survived
Survived	1859	90953
Not Survived	6992	6433

Logistic regression confusion matrix For 10 attributes.

Actual	Predicted	
	Survived	Not Survived
Survived	1697	91115
Not Survived	7513	5912

Logistic regression confusion matrix For 13 attributes.

Multiple factor analysis

Actual	Predicted	
	Survived	Not Survived
Survived	91018	1794
Not Survived	6453	6972

Multiple factor analysis confusion matrix For J48 with 8 attributes.

Actual	Predicted	
	Survived	Not Survived
Survived	90949	1863
Not Survived	7029	6396

Multiple factor analysis confusion matrix For Logistic with 8 attributes.

Actual	Predicted	
	Survived	Not Survived
Survived	86408	6404
Not Survived	5211	8214

Multiple factor analysis confusion matrix For Naive with 8 attributes.

Actual	Predicted	
	Survived	Not Survived
Survived	90952	1860
Not Survived	5999	7426

Multiple factor analysis confusion matrix For J48 with 10 attributes.

Actual	Predicted	
	Survived	Not Survived
Survived	91084	1728
Not Survived	6791	6634

Multiple factor analysis confusion matrix For Logistic with 10 attributes.

Actual	Predicted	
	Survived	Not Survived
Survived	86088	6724
Not Survived	4955	8470

Multiple factor analysis confusion matrix For Naive with 10 attributes.

Actual	Predicted	
	Survived	Not Survived
Survived	90617	2195
Not Survived	5430	7995

Multiple factor analysis confusion matrix For J48 with 13 attributes.

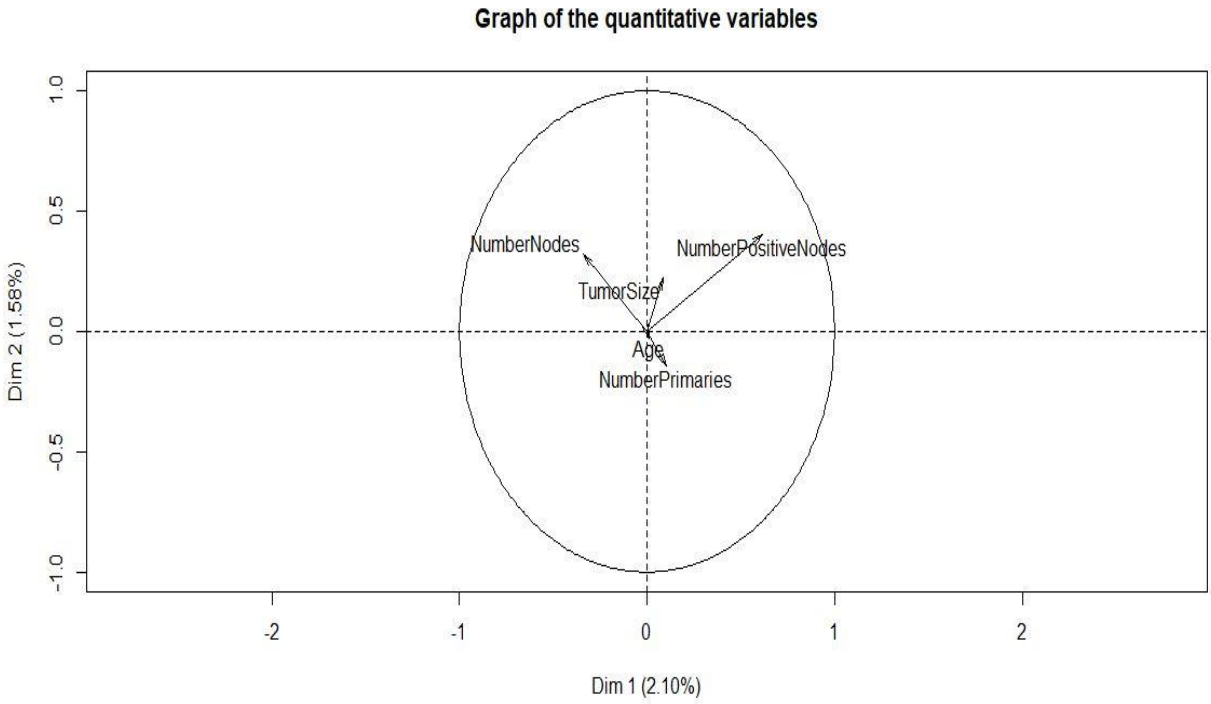
Actual	Predicted	
	Survived	Not Survived
Survived	91125	1687
Not Survived	6747	6678

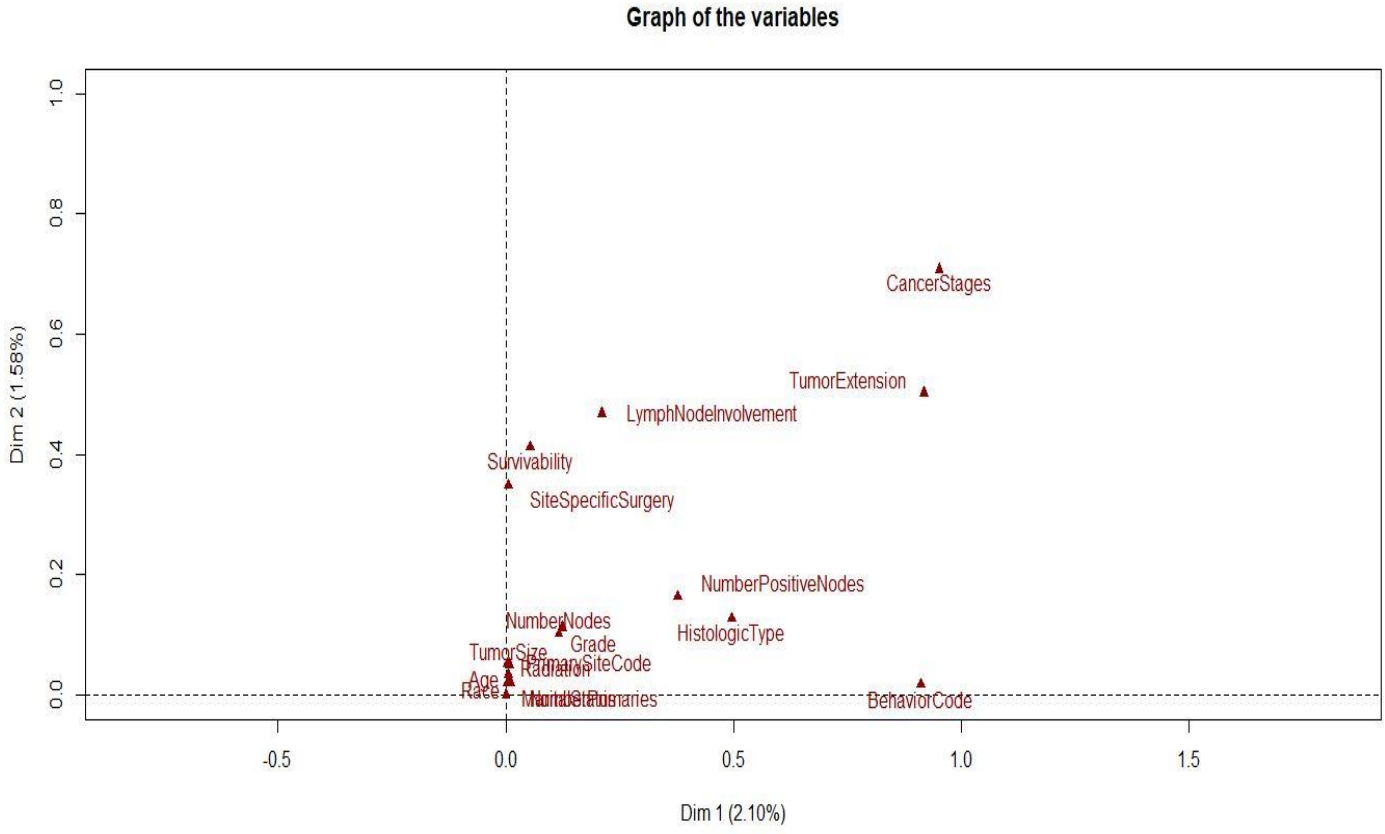
Multiple factor analysis confusion matrix For Logistic with 13 attributes.

Actual	Predicted	
	Survived	Not Survived
Survived	86128	6684
Not Survived	4797	8628

Multiple factor analysis confusion matrix For Naive with 13 attributes

Factor Analysis for Mixed Data





Comparison of algorithms

Algorithms	Results(106,237)				
	Accuracy	**Sensitivity**	**Specificity**	**ROC Area**	**Time(s)**
Decision Tree	92.84 %	0.983	0.938	0.914	10.3
Naïve Bayes	90.88 %	0.936	0.959	0.926	0.32
ANN	92.08 %	0.973	0.939	0.797	69998.78
Logistic regression	93.07%	0.981	0.942	0.935	191.25
Support Vector Machine	92.35%	0.983	0.933	0.748	2223.75

Table 10: Comparison of different Algorithms after feature reduction.

Implementation feature reduction part

The constants class

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package featureselectionapp;
```

```
/**
 *
 * @author abid&youssouf
 */
public class Constants {
    public static final int THREADS_MAX_COUNT = 64;
    public static final long MIN_DATA_SET_BYTES_THREAD = 4096;
}
```

The CustomLogger Class

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package featureselectionapp;

import java.io.PrintStream;

/**
 *
 * @author abid & youssouf
 */
public class CustomLogger {

    private static PrintStream ps = System.err;

    private static void printUsage() {
```

```
    log("Usage: ./feature_selection_app [options] data_set_file [output_file]\n"
            + "options:\n"
            + "-h : to show this help\n"
            + "-m metric_type : feature selection metric\n"
            + "     pmi  -- PMI (Pointwise Mutual Information) feature selection metric\n"
            + "     chi2 -- Chi^2 (X^2) feature selection metric\n"
            + "-n number : number of features to select and return\n"
            + "-t number : number of threads\n");
}

public static void exitWithUsage(){
    printUsage();
    System.exit(1);
}

public static void logAndExit(String txt) {
    log(txt);
    System.exit(2);
}

public static void logAndExit(Throwable ex, String txt) {
    log(ex, txt);
    System.exit(2);
}

public static void logAndExitWithUsage(String txt) {
    log(txt);
```

```
        exitWithUsage();
    }

    public static void logAndExitWithUsage(Throwable ex, String txt) {
        log(ex, txt);
        exitWithUsage();
    }

    public static void log(String txt){
        ps.println(txt);
    }

    public static void log(Throwable t) {
        ps.println(t.getMessage());
    }

    public static void log(Throwable t, String txt) {
        ps.println(txt);
        ps.println(t.getMessage());
        t.printStackTrace(ps);
    }

}
```

The CustomStringIntHashMap Class

```
/*
```

```
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package featureselectionapp;
import gnu.trove.map.hash.THashMap;
/**
 *
 * @author abid & youssouf */
public class CustomStringIntHashMap extends THashMap<String, Integer> {

    public void increment(String key, int inc) {
        if (this.containsKey(key)) {
            inc += this.get(key).intValue();
        }
        this.put(key, inc);
    }

    public void decrement(String key, int dec) {
        if (this.containsKey(key)) {
            dec = this.get(key).intValue() - dec;
        }
        this.put(key, dec);
    }
}
```

The DataSetFileEntry Class

```
/*
```

```
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package featureselectionapp;
import gnu.trove.map.TMap;
import gnu.trove.map.hash.THashMap;
/**
 *
 * @author abid & youssouf
 */
public class DataSetFileEntry {
   public String class_name;
   public TMap<String, Double> features;

   public static DataSetFileEntry getInstanceByLineString(String _line) {
      DataSetFileEntry entry = new DataSetFileEntry();
      String[] parts = _line.split("\\s+");
      entry.class_name = parts[0];
      entry.features = new THashMap(parts.length - 1);
      for (int i = parts.length - 1; i >= 1; i--) {
         String[] feature_and_value = parts[i].split(":");
         entry.features.put(feature_and_value[0], Double.valueOf(feature_and_value[1]));
      }
      return entry;
   }
}
```

The FeatureSelectionApp Class

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package featureselectionapp;
import gnu.trove.map.TMap;
import gnu.trove.map.hash.THashMap;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.util.Comparator;
import java.util.Iterator;
import java.util.List;
import java.util.SortedMap;
import java.util.TreeMap;
import java.util.concurrent.atomic.AtomicInteger;
import java.util.logging.Level;
import java.util.logging.Logger;
/**
 *
 * @author abid & youssouf
```

```
 */
public class FeatureSelectionApp extends ThreadObserver {


   /**
    * Map<feature_name, Map<class_name, frequency_of_feature_for_this_class>>
    */
   private static TMap<String, CustomStringIntHashMap> features_frequencies_per_class;
   private static CustomStringIntHashMap classes_frequencies = new
CustomStringIntHashMap();
   private static CustomStringIntHashMap features_frequencies = new
CustomStringIntHashMap();
   private static int all_classes_count = 0;
   private static int all_features_count = 0;
   private static int records_count = 0;
   private static String data_set_file_path = null;
   private static File output_file = null;
   private static FeatureSelectionMetricEnum metric = null;
   private static int selected_features_count = 0;
   private static int threads_count = 1;
   private static final Object mutex = new Object();
   private static volatile boolean processing_done = false;


   private static void parse_command_line(String argv[]) {
      // parse options
      int i;
      for (i = 0; i < argv.length; i++) {
         if (argv[i].charAt(0) != '-') {
            // read data set file path
```

```
    if (data_set_file_path == null) {
        data_set_file_path = argv[i];

    } else if (output_file == null) {   // read output file path
        output_file = new File(argv[i]);
        if (!output_file.exists()) {
            try {
                output_file.createNewFile();
            } catch (IOException ex) {
                CustomLogger.logAndExit(ex, "Could not create output file: " + argv[i]);
            }
        }
        if (!output_file.canWrite()) {
            CustomLogger.logAndExit("Output file " + argv[i] + " is not writable");
        }
    } else {    // else this is the end of the params
        break;
    }
} else {
    if (++i >= argv.length) {
        CustomLogger.exitWithUsage();
    }
    char option_char = argv[i - 1].charAt(1);
    switch (option_char) {
        case 'h':
            CustomLogger.exitWithUsage();
            break;
```

```
            case 'm':
              try {
                metric = FeatureSelectionMetricEnum.getMetricByKey(argv[i]);
              } catch (IllegalArgumentException ex) {
                CustomLogger.logAndExit(ex.getMessage());
              }
              break;
            case 'n':
              selected_features_count = Integer.valueOf(argv[i]).intValue();
              break;
            case 't':
              threads_count = Integer.valueOf(argv[i]).intValue();
              if (threads_count > Constants.THREADS_MAX_COUNT) {
                threads_count = Constants.THREADS_MAX_COUNT;
              }
              break;
            default:
              CustomLogger.logAndExit("Invalid option: -" + option_char);
          }
        }
      }

      if (data_set_file_path == null || selected_features_count < 1 || metric == null) {
        CustomLogger.logAndExitWithUsage("Data set file, metric and number of selected
features are mandatory paramters");
      }
    }
```

```
  private static void readDataSetFile(List<DataSetFileEntry> _data) {
    features_frequencies_per_class = new THashMap();
    records_count = _data.size();
    for (DataSetFileEntry line_entry : _data) {
      classes_frequencies.increment(line_entry.class_name, 1);
      for (String feature_name : line_entry.features.keySet()) {
        // the value of the feature is either 1 or nothing ////FIXME this part I am not sure
about
        int feature_val = line_entry.features.get(feature_name).intValue();
        CustomStringIntHashMap feature_map;
        if (features_frequencies_per_class.containsKey(feature_name)) {
          feature_map = features_frequencies_per_class.get(feature_name);
        } else {
          feature_map = new CustomStringIntHashMap();
          features_frequencies_per_class.put(feature_name, feature_map);
        }
        feature_map.increment(line_entry.class_name, feature_val);
        features_frequencies.increment(feature_name, feature_val);
      }
    }
    all_features_count = features_frequencies.size();
    all_classes_count = classes_frequencies.size();
  }

  /**
   * @param args the command line arguments
```

```
 */
public static void main(String[] argv) {
    // parse the command options and read data set file into data structures
    parse_command_line(argv);

    // create an instance of this class to act as an observer for threads
    FeatureSelectionApp app = new FeatureSelectionApp();
    app.startThreads(threads_count, data_set_file_path);

    app.waitForAllThreads();
}

private void printOutput(final TMap<String, Double> all_features_scores) {
    PrintStream os = null;
    if (output_file != null) {
        try {
            os = new PrintStream(output_file);
        } catch (FileNotFoundException ex) {
            CustomLogger.log(ex, "Cannot write to output file");
        }
    } else {
        os = System.out;
    }

    // sort the features
    SortedMap<String, Double> all_features_sorted_scores = new TreeMap(new
Comparator<String>() {
```

```
        @Override
        public int compare(String o1, String o2) {
            double diff = all_features_scores.get(o2) - all_features_scores.get(o1);    //
descendingly
            if (diff > 0) {
                return 1;
            } else {
                return -1;
            }
        }
    });
    all_features_sorted_scores.putAll(all_features_scores);

    // put the top features in the outputfile
    Iterator<String> features_names = all_features_sorted_scores.keySet().iterator();
    int i = 0;
    while (features_names.hasNext() && i < selected_features_count) {
        String fname = features_names.next();
        os.println(fname + "\t" + all_features_scores.get(fname));
        i++;
    }
    if (output_file != null) {
        os.close();
    }
  }

  @Override
```

```
public void allThreadsFinished(List<DataSetFileEntry> _data) {
   readDataSetFile(_data);
   // execute feature selection
   TMap<String, Double> all_features_scores = FeatureSelectionMetric.getInstance(
         metric,
         features_frequencies_per_class,
         classes_frequencies,
         features_frequencies,
         records_count).execute();

   // print the output
   this.printOutput(all_features_scores);
   synchronized (mutex) {
      processing_done = true;
   }
}

private void waitForAllThreads() {
   while (true) {
      try {
         synchronized (mutex) {
            if (processing_done) {
               return;
            }
            mutex.wait(250);
         }
      } catch (InterruptedException ex) {
```

```
            CustomLogger.log(ex);
        }
      }
   }
}
```

The FeatureSelectionMetric Class

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package featureselectionapp;
import gnu.trove.map.TMap;
import gnu.trove.map.hash.THashMap;
import java.util.Map;
import java.util.Set;

/**
 *
 * @author abid & youssouf
 */
public abstract class FeatureSelectionMetric {

   protected TMap<String, CustomStringIntHashMap> features_frequencies_per_class;
   protected CustomStringIntHashMap classes_frequencies;
   protected CustomStringIntHashMap features_frequencies;
```

```
protected int all_classes_count;
protected int all_features_count;
protected int all_data_set_records_count;

protected void takeInput(
        TMap<String, CustomStringIntHashMap> _features_frequencies_per_class,
        CustomStringIntHashMap _classes_frequencies,
        CustomStringIntHashMap _features_frequencies,
        int _all_data_set_records_count) {
    features_frequencies_per_class = _features_frequencies_per_class;
    classes_frequencies = _classes_frequencies;
    features_frequencies = _features_frequencies;
    all_classes_count = _classes_frequencies.size();
    all_features_count = _features_frequencies.size();
    all_data_set_records_count = _all_data_set_records_count;
}

public static FeatureSelectionMetric getInstance(
        FeatureSelectionMetricEnum _type,
        TMap<String, CustomStringIntHashMap> _features_frequencies_per_class,
        CustomStringIntHashMap _classes_frequencies,
        CustomStringIntHashMap _features_frequencies,
        int _all_data_set_records_count) {

    FeatureSelectionMetric metric = null;
    if (_type == FeatureSelectionMetricEnum.PMI) {
        metric = new MetricPMI();
```

```
    } else if (_type == FeatureSelectionMetricEnum.CHI2) {
        metric = new MetricChi2();
    } else {
        throw new IllegalArgumentException("Invalid FeatureSelectionMetric type");
    }
    metric.takeInput(
            _features_frequencies_per_class,
            _classes_frequencies,
            _features_frequencies,
            _all_data_set_records_count);
    return metric;
  }

  public abstract TMap<String, Double> execute();

  ///// List of Metrics //////////////////////
  private static class MetricPMI extends FeatureSelectionMetric {

    private double p_of_f(String _feature) {
        return (double) features_frequencies.get(_feature) / (double)
all_data_set_records_count;
    }

    private double p_of_c(String _class) {
        return (double) classes_frequencies.get(_class) / (double) all_data_set_records_count;
    }
```

```
private double p_of_f_intersect_c(String _class, String _feature) {
        CustomStringIntHashMap map = features_frequencies_per_class.get(_feature);
        if (map == null) return 0;
        Integer i = map.get(_class);
        if (i == null) i = 0;
        return (double)i / (double) all_data_set_records_count;
}

private double p_of_c_given_f(String _class, String _feature) {
   return p_of_f_intersect_c(_class, _feature) / p_of_f(_feature);
}

private double p_of_f_given_c(String _feature, String _class) {
   return p_of_f_intersect_c(_class, _feature) / p_of_c(_class);
}

private double log2(double _num) {
   if (_num == 0.0) return 0.0;
   return Math.log(_num) / Math.log(2);
}

private double pmiOfFeatureForAllClasses(String _feature) {
   // PMI(f) = sum<all_classes>(P(f,c) * Log2(P(f|c) / P(f)))
   double sum = 0.0;
   for (String class_name : classes_frequencies.keySet()) {
      // P(f, c)
      sum += p_of_f_intersect_c(class_name, _feature)
```

```
                * log2(p_of_f_given_c(_feature, class_name) / p_of_f(_feature));
        }
        return sum;
    }

    @Override
    public TMap<String, Double> execute() {
        TMap<String, Double> scores = new THashMap(features_frequencies.size());
        for (String feature_name : features_frequencies.keySet()) {
            scores.put(feature_name, pmiOfFeatureForAllClasses(feature_name));
        }
        return scores;
    }
}

private static class MetricChi2 extends FeatureSelectionMetric {

        // class frequency
        private double Nc1(String _class){
                Integer i = classes_frequencies.get(_class);
                if (i != null) return (double)i.intValue();
                return 0;
        }

        // not class frequency
        private double Nc0(String _class, int _records_count){
                return _records_count - Nc1(_class);
```

```
    }

    // feature frequency
    private double Nf1(String _feature){
            Integer i = features_frequencies.get(_feature);
            if (i != null) return i;
            return 0;
    }

    // not feature frequency
    private double Nf0(String _feature, int _records_count){
            return _records_count - Nf1(_feature);
    }


private double Nc1f1(String _class, String _feature) {
        CustomStringIntHashMap map = features_frequencies_per_class.get(_feature);
        if (map == null) return 0;
            Integer i = map.get(_class);
            if (i != null) return (double)i.intValue();
            return 0;
}

// class With Out Feature Frequency
private double Nc1f0(String _class, String _feature) {
            return Nc1(_class) - Nc1f1(_class, _feature);
}
```

```
    // feature With Out Class Frequency
        private double Nc0f1(String _class, String _feature) {
                return Nf1(_feature) - Nc1f1(_class, _feature);
    }

    // frequency With out Class And Feature
    private double Nc0f0(String _class, String _feature, int _records_count) {
                return _records_count - ((Nc1(_class) + Nf1(_feature)) - Nc1f1(_class, _feature));
    }

    private double chi2OfFeatureForAllClasses(String _feature) {
                double sum = 0.0;
                double nc1f1, nc0f0, nc1f0, nc0f1, nc1, nf1, nf0, nc0;
                int n = all_data_set_records_count;
                nf1 = Nf1(_feature);
                //nf0 = Nf0(_feature, n);
                for (String _class : classes_frequencies.keySet()) {
                        nc1   = Nc1(_class);
                        nc0   = Nc0(_class, n);
                        nc1f1 = Nc1f1(_class, _feature);
                        nc0f0 = Nc0f0(_class, _feature, n);
                        nc1f0 = Nc1f0(_class, _feature);
                        nc0f1 = Nc0f1(_class, _feature);
                        //http://blog.datumbox.com/using-feature-selection-methods-in-text-
classification/
                        double d = (
```

```
                        (nc1f1 + nc0f1) *
                        (nc1f1 + nc1f0) *
                        (nc1f0 + nc0f0) *
                        (nc0f1 + nc0f0));
                        double val = 0.0;
                        if (d != 0.0) val = (n * Math.pow((nc1f1 * nc0f0) - (nc1f0 * nc0f1), 2.0)) / d;
                        sum += val;
                }
                return sum / (double)all_classes_count;
        }

        @Override
        public TMap<String, Double> execute() {
           TMap<String, Double> scores = new THashMap(features_frequencies.size());
           for (String feature_name : features_frequencies.keySet()) {
              scores.put(feature_name, chi2OfFeatureForAllClasses(feature_name));
           }
           return scores;
        }
     }
}
```

The FeatureSelectionMetricEnum

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package featureselectionapp;
```

```
/**
 *
 * @author abid & youssouf
 */
public enum FeatureSelectionMetricEnum {
   PMI("pmi"),
   CHI2("chi2");

   private String key = null;
   FeatureSelectionMetricEnum(String _key) {
      key = _key;
   }

   public static FeatureSelectionMetricEnum getMetricByKey(String _key) {
      for (FeatureSelectionMetricEnum f : FeatureSelectionMetricEnum.values()) {
         if (f.key.equals(_key)) return f;
      }
      throw new IllegalArgumentException("Invalid FeatureSelectionMetricEnum: " + _key);
   }
}
```

The FeatureSelectionObserver Interface

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package featureselectionapp;
```

```
import gnu.trove.map.TMap;

/**
 *
 * @author abid & youssouf
 */
public interface FeatureSelectionObserver {

    public void selectedFeatures(TMap<String, Double> _features_scores);

}
```

The ReaderThread Class

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package featureselectionapp;

import gnu.trove.map.TMap;
import gnu.trove.map.hash.THashMap;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
```

```
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author abid & youssouf
 */
public class ReaderThread {

    private ThreadObserver observer = null;
    private int thread_index = 0;
    private String file_path;
    private int thread_count = 0;

    public ReaderThread(String _file_path, int _thread_index, int _thread_count, ThreadObserver
_observer){
        file_path = _file_path;
        thread_index = _thread_index;
        thread_count = _thread_count;
        observer = _observer;
    }

    public void execute(){
        Thread th = new Thread(new Runnable() {
```

```
        @Override
        public void run() {
            List<DataSetFileEntry> class_features = new ArrayList();
            BufferedReader br = null;
            try {
                br = new BufferedReader(new InputStreamReader(new FileInputStream(file_path)));
            } catch (FileNotFoundException ex) {
                CustomLogger.logAndExit(ex, "Could not open Data set file for reading");
            }



            String line = null;
            try {
                int line_number = 0;
                while ((line = br.readLine()) != null) {
                    if ((line_number % thread_count) == thread_index)
class_features.add(DataSetFileEntry.getInstanceByLineString(line));
                    line_number++;
                }
            } catch (IOException ex) {
                CustomLogger.log("Error while reading from data set file");
            }
            try {
                if (br != null) br.close();
            } catch (IOException ex) {
                CustomLogger.log("Error while closing data set file");
```

```
            }
            observer.threadFinished(thread_index, class_features);
          }
        });
        th.start();
    }
}
```

The ThreadObserver Abstract Class

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package featureselectionapp;

import gnu.trove.map.TMap;
import gnu.trove.map.hash.THashMap;
import java.io.File;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.concurrent.atomic.AtomicInteger;

/**
 *
 * @author abid & youssouf
```

```
 */
public abstract class ThreadObserver {

    protected final List<DataSetFileEntry> data = new ArrayList();
    private AtomicInteger working_threads;
    private int threads_count = 0;

    public void startThreads(int _thread_count, String _file_path) {
        // adjust threads count and features per thread
        File data_set_file = new File(_file_path);
        if (!data_set_file.exists()) {
            CustomLogger.logAndExit("Dataset file " + _file_path + "does not exist");
        }
        threads_count = _thread_count;

        working_threads = new AtomicInteger(threads_count);

        for (int i = 0; i < threads_count; i++) {
            ReaderThread th = new ReaderThread(_file_path, i, threads_count, this);
            th.execute();
        }
    }

    public void threadFinished(int _thread_index, List<DataSetFileEntry> _class_features) {
        synchronized(data) {
            data.addAll(_class_features);
        }
```

```
        int count = working_threads.decrementAndGet();
        if (count <= 0) allThreadsFinished(data);
    }

    public abstract void allThreadsFinished(List<DataSetFileEntry> _data);

}
```

Chapter 6:

Conclusion and Future Work.

6.1 Conclusion

From the above performance evaluation it is seen that the **Logistic Regression has** the best accuracy of **93.07%** before any feature reduction and time to build the model was **191sec**.

The accuracy of logistic regression increased to **94.35%**, for factor analysis for mixed data feature reduction method.

Time to build the model also decreased to **102sec**. So from this study, logistic regression showed the best result for breast cancer survivability prediction.

6.2 Future work

This paper has outlined, discussed and resolved the issues, algorithms, and techniques for the problem of breast cancer survivability prediction in SEER database.

Increasing the performance of all the mention algorithms, removing some attributes that were ranked last in their contribution to the prediction of the model.

6.3 References

[1] American Cancer Society. Breast Cancer Facts& Figures 2005-2006. Atlanta: American Cancer Society, Inc. (http://www.cancer.org/).

[2] Surveillance, Epidemiology, and End Results (SEER) Program (www.seer.cancer.gov) Public-Use Data (1973-2002), National Cancer Institute, DCCPS, Surveillance Research Program, Cancer Statistics Branch, released April 2005, based on the November 2004 submission.

[3] Ian H. Witten and Eibe Frank. Data Mining: Practical machine learning tools and techniques, 2nd Edition. San Fransisco:Morgan Kaufmann;2005

[4]. Jyoti Soni, Ujma Ansari, Dipesh Sharma, Sunita Soni "Predictive Data Mining for Medical Diagnosis: An Overview of Heart Disease Prediction" IJCSE Vol. 3 No. 6 June 2011

[5] D. Delen, G. Walker and A. Kadam (2005), Predicting breast cancer survivability: a comparison of three data mining methods, Artificial Intelligence in Medicine.

[6] A.Bellachia and E.Guvan,"Predicting breast cancer survivability using data mining techniques", Scientific Data Mining Workshop, inconjunction with the 2006 SIAM Conference on Data Mining,2006

[7] Ian H. Witten and Eibe Frank. Data Mining: Practical machine learning tools and techniques, 2nd Edition. San Fransisco:Morgan Kaufmann;2005.

[8] American Cancer Society. Breast Cancer Facts& Figures 2005-2006. Atlanta: American Cancer Society, Inc. (http://www.cancer.org/).

[9] J. R. Quinlan, C4.5: Programs for Machine Learning. San Mateo, CA:Morgan Kaufmann; 1993.

[10] P.-N. Tan, M. Steinbach, and V. Kumar, Introduction to Data Mining. Reading, MA: Addison-Wesley, 2005.

[11] Razavi, A. R., Gill, H., Ahlfeldt, H., and Shahsavar, N., Predicting metastasis in breast cancer: comparing a decision tree with domain,2011

[10]. V. Chauraisa and S. Pal, "Early Prediction of Heart Diseases Using Data Mining Techniques", Carib.j.SciTech,Vol.1, pp. 208-217, 2013

[11] Weka: Data Mining Software in Java,