BY,

PRERANA T H M

COMPUTER SCIENCE DEPARTMENT

MACHINE LEARNING PROJECT REPORT

**Introduction:**

In machine learning, classification refers to the problem of identifying to which class or category a new observation belongs to, on the basis of a training dataset containing observations whose class is already known. The individual observations are analyzed into a set of quantifiable properties, known as features or explanatory variables etc. We divide the dataset into train and test data to build our model and apply one of the classification algorithms like Support Vector Machines, k-Nearest Neighbour Classifier, Naive Bayes Classifier etc. to classify the data into one of the classes and finally calculate the accuracy based on the correctly classified instances.

**Problem Definition:**

I have chosen the 2nd dataset out of the given 4 datasets .In this particular dataset we are trying to classify various night sky scenes and detect if there is an aurora in the sky, is it cloudy or otherwise. Hence it is a 3-class classification problem. The data is features extracted from all-sky camera images.

**Algorithms:**

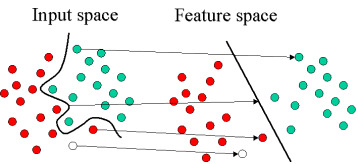
The algorithms used are SVM, kNN and Naive Bayes.

**SVM:**

In [machine learning](http://en.wikipedia.org/wiki/Machine_learning), support vector machines SVMs are [supervised learning](http://en.wikipedia.org/wiki/Supervised_learning) models that analyze data and recognize patterns, used for classification and regression analysis.

In our case we are using SVM for classification analysis. Given the training set, each marked as belonging to one of 3 classes, an SVM training algorithm builds a model that assigns new examples into one category or the other, making it a non-probabilistic binary linear classifier .The type of SVM we are using is a multiclass SVM, since it involves 3 classes. Multiclass SVM aims to assign labels to instances by using support vector machines, where the labels are drawn from a finite set of several elements. In multiclass SVM, we are using one-versus-the-rest approach to obtain classification. Classification of new instances for the one-versus-all case is done by a winner-takes-all strategy, in which the classifier with the highest output function assigns the class.

The illustration below shows the basic idea behind Support Vector Machines. Here we see the original objects (left side) are mapped using a set of mathematical functions, known as kernels. The process of rearranging the objects is known as mapping or transformation. In this new setting, the mapped objects (right side) are linearly separable and thus instead of constructing the complex curve (left schematic), all we have to do is to find an optimal line that can separate the GREEN and the RED objects.



**Implementation:**

Here we are using the C-SVM Classification. For this type of SVM, training involves the minimization of the error function:

https://www.statsoft.com/textbook/graphics/SVMIntro4.gif

subject to the constraints:

https://www.statsoft.com/textbook/graphics/SVMIntro5.gif

where C is the capacity constant, w is the vector of coefficients, b is a constant, and https://www.statsoft.com/textbook/image2.gif represents parameters for handling non separable data inputs. The index i labels the N training cases. Note that https://www.statsoft.com/textbook/graphics/SVMIntro7.gif represents the class labels and xi represents the independent variables. The kernel https://www.statsoft.com/textbook/graphics/SVMIntro15c.gif is used to transform data from the input (independent) to the feature space. It should be noted that the larger the C, more is the error. Thus, C should be carefully chosen to avoid over fitting.

The different types of kernels used are as follows:

* Linear: \langle x, x'\rangle.
* Polynomial: (\gamma \langle x, x'\rangle + r)^d. d is specified by keyword degree, r by coef0.
* RBF: \exp(-\gamma |x-x'|^2). \gamma is specified by keyword gamma, must be greater than 0.
* Sigmoid: (\tanh(\gamma \langle x,x'\rangle + r)), where r is specified by coef0.

The function used is svm(), which is used to train a support vector machine. Some import parameters include:

* data: an optional data frame containing the variables in the model. If this option is used, the parameters x and y described below, aren't necessary;
* x: a data matrix, a vector, or a sparse matrix that represents the instances of the dataset and their respective properties. Rows represent the instances and columns represent the properties;
* y: a response vector with one label for each row (instance) of x;
* type: sets how svm() will work. The possible values for classification are: C, nu and one (for novelty detection);
* kernel: defines the kernel used in training and prediction. The options are: linear, polynomial, radial basis and sigmoid;
* degree: parameter needed if the kernel is polynomial (default: 3);
* gamma: parameter needed for all types of kernels except linear (default: 1/(data dimension));
* coef0: parameter needed for polynomial and sigmoid kernels (default: 0);
* cost: cost of constraint violation (default: 1). This is the ‘C’-constant of the regularization term in the Lagrange formulation;

**Code and Experimental Results:**

Library used: e1071

**Dividing the dataset into training(80%) and testing(20%) datasets:**

library(e1071)

data=read.table("f2.txt")

labels=read.table("l2.txt")

dim(data)

10188 507

dim(labels)

10188 1

train=data[1:8150,]

test=data[8151:10188,]

trainlabels=labels[1:8150,]

testlabels=labels[8151:10188,]

dim(train)

8150 507

dim(test)

2038 507

**RBF Kernel:**

model<-svm(train,trainlabels,type="C-classification")

model

Call:

svm.default(x = train, y = trainlabels, type = "C-classification")

Parameters:

SVM-Type: C-classification

SVM-Kernel: radial

cost: 1

gamma: 0.001972387

Number of Support Vectors: 3063

pred<-predict(model,test)

Accuracy:

Formula: (count/total)\*100

count=0

for(i in 1:2038)

{

if(testlabels[i]==pred[i])

{

count=count+1;

}

}

count

1792

1792/2038\*100

87.92934

**Linear Kernel:**

model<-svm(train,trainlabels,type="C-classification",kernel="linear")

model

Call:

svm.default(x = train, y = trainlabels, type = "C-classification", kernel = "linear")

Parameters:

SVM-Type: C-classification

SVM-Kernel: linear

cost: 1

gamma: 0.001972387

Number of Support Vectors: 1161

pred<-predict(model,test)

Accuracy:

count

1634

1634/2038\*100

80.17664

**Polynomial Kernel:**

model<-svm(train,trainlabels,type="C-classification",kernel="polynomial")

model

Call:

svm.default(x = train, y = trainlabels, type = "C-classification", kernel = "polynomial")

Parameters:

SVM-Type: C-classification

SVM-Kernel: polynomial

cost: 1

degree: 3

gamma: 0.001972387

coef.0: 0

Number of Support Vectors: 4446

pred<-predict(model,test)

Accuracy:

count

1753

1753/2038\*100

86.0157

**Sigmoid Kernel:**

model<-svm(train,trainlabels,type="C-classification",kernel="sigmoid")

model

Call:

svm.default(x = train, y = trainlabels, type = "C-classification", kernel = "sigmoid")

Parameters:

SVM-Type: C-classification

SVM-Kernel: sigmoid

cost: 1

gamma: 0.001972387

coef.0: 0

Number of Support Vectors: 2125

pred<-predict(model,test)

Accuracy:

count

1648

1648/2038\*100

80.86359

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SVM Kernel | Radial | Linear | Polynomial | Sigmoid |
| Accuracy | 87.93% | 80.18% | 86.02% | 80.86% |

**GRAPH:**

Radial\_Linear\_Polynomial\_Sigmoid<-c(1,2,3,4)

Accuracy<-c(87.93,80.18,86.02,80.86)

plot( Radial\_Linear\_Polynomial\_Sigmoid, Accuracy,main="SVM - Kernel Vs Accuracy",col="purple",type="o")

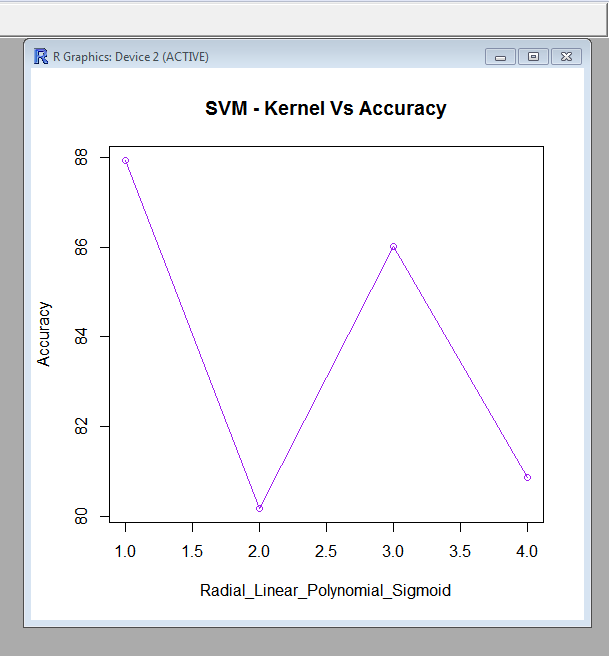


Fig 1. Plot of Different SVM Kernels Vs Accuracy

**kNN:**

In pattern recognition, the *k*-Nearest Neighbours algorithm (or *k*-NN) is a non-parametric method used for classification and regression. In both cases, the input consists of the *k* closest training examples in the feature space. The output depends on whether *k*-NN is used for classification or regression: In our case we are using *k-NN* classification where the output is a class membership. An object is classified by a majority vote of its neighbours, with the object being assigned to the class most common among its *k* nearest neighbours (*k* is a positive integer). If *k* = 1, then the object is simply assigned to the class of that single nearest neighbour.

**Implementation:**

The function used is Knn (train, test, cl, k = 1, l = 0, prob = FALSE, use.all = TRUE)

**Arguments**

|  |  |
| --- | --- |
| train | matrix or data frame of training set cases. |
| test | matrix or data frame of test set cases. A vector will be interpreted as a row vector for a single case. |
| Cl | factor of true classifications of training set |
| K | number of neighbours considered. |
| L | minimum vote for definite decision, otherwise doubt. (More precisely, less than k-l dissenting votes are allowed, even if k is increased by ties.) |
| Prob | If this is true, the proportion of the votes for the winning class are returned as attribute prob. |
| use.all | controls handling of ties. If true, all distances equal to the kth largest are included. If false, a random selection of distances equal to the kth is chosen to use exactly k neighbours. |

**Code and Experimental Results:**

Library used: class

**Using *k*=1:**

pred=knn(train,test, trainlabels,k=1)

Accuracy:

Formula: (count/total)\*100

count=0

for(i in 1:2038)

{

if(testlabels[i]==pred[i])

{

count=count+1;

}

}

count

1665

1665/2038\*100

81.69774

**Using *k*=10:**

pred=knn(train,test, trainlabels,k=10)

Accuracy:

count

1700

1700/2038\*100

83.41511

**Using *k*=20:**

pred=knn(train,test, trainlabels,k=20)

Accuracy:

count

1690

1690/2038\*100

82.92444

**Using *k*=50:**

pred=knn(train,test, trainlabels,k=50)

Accuracy:

count

1653

1653/2038\*100

81.10893

**Using *k*=100:**

pred=knn(train,test, trainlabels,k=100)

Accuracy:

count

1590

1590/2038\*100

78.01766

**Using *k*=200:**

pred=knn(train,test, trainlabels,k=100)

Accuracy:

count

1522

1522/2038\*100

74.68106

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| K –value | K=1 | K=10 | K=20 | K=50 | K=100 | K=200 |
| Accuracy | 81.69% | 83.24% | 82.92% | 81.10% | 78.02% | 74.68% |

**GRAPH:**

k<-c(1,10,20,50,100,200)

Accuracy<-c(81.69,83.24,82.92,81.10,78.02,74.68)

plot( k, Accuracy,main="k Vs Accuracy",col="red",type="o")

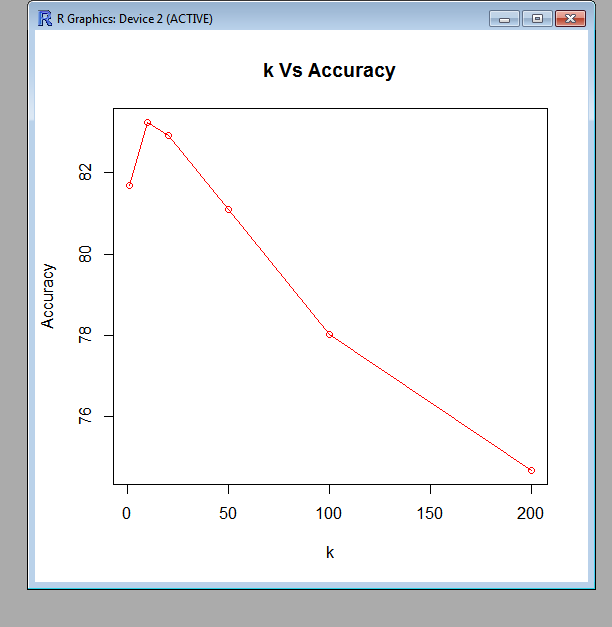


Fig 2. Plot of Different k values Vs Accuracy

Naive Bayes:

A naive Bayes classifier is a simple [probabilistic classifier](http://en.wikipedia.org/wiki/Probabilistic_classifier) based on applying [Bayes' theorem](http://en.wikipedia.org/wiki/Bayes%27_theorem) with strong [independence](http://en.wikipedia.org/wiki/Statistical_independence) assumptions. A more descriptive term for the underlying probability model would be [independent](http://en.wikipedia.org/wiki/Statistical_independence) feature model. An overview of statistical classifiers is given in the article on [pattern recognition](http://en.wikipedia.org/wiki/Pattern_recognition). In this classification we are dividing the data set into training dataset and testing data set. The testing data set is classified based on the training data set and accuracy is also determined.

Abstractly, the probability model for a classifier is a conditional model

p(C \vert F_1,\dots,F_n)\,

Over a dependent class variable C with a small number of outcomes or classes, conditional on several feature variables F_1 through F_n.

In this classification we calculate the posterior probability using

p(C \vert F_1,\dots,F_n) = \frac{p(C) \ p(F_1,\dots,F_n\vert C)}{p(F_1,\dots,F_n)}. \,

We construct a classifier from the probability model using

\mathrm{classify}(f_1,\dots,f_n) = \underset{c}{\operatorname{argmax}} \ p(C=c) \displaystyle\prod_{i=1}^n p(F_i=f_i\vert C=c).

**Implementation:**

The function used is predict.naiveBayes(object, newdata, type = c("class", "raw"), threshold = 0.001, ...)

Arguments:

|  |  |
| --- | --- |
| Object | An object of class "naiveBayes". |
| Newdata | A dataframe with new predictors. |
| Type | see value. |
| Threshold | Value replacing cells with 0 probabilities. |
|  | Currently not used. |

**Code and Experimental Results:**

Library used: class

classifier<-naiveBayes(train,as.factor(trainlabels))

pred<-predict(classifier,test)

Accuracy:

count

1150

1150/2038\*100

56.42787

**Comparison of Algorithms:**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Algorithm | SVM Radial | SVM  Linear | SVM Polynomial | SVM Sigmoid | k-NN  k=1 | k-NN  k=10 | k-NN  k=20 | k-NN  k=50 | k-NN  k=100 | k-NN  k=200 | Naïve Bayes |
| x-axis | X=1 | X=2 | X=3 | X=4 | X=5 | X=6 | X=7 | X=9 | X=10 | X=11 | X=12 |
| Accuracy | 87.93 % | 80.18  % | 86.02  % | 80.86  % | 81.69 % | 83.24% | 82.92% | 81.10% | 78.02% | 74.68% | 56.42% |

**Creation of H-Graph:**

Algorithm<-c(1,2,3,4,5,6,7,8,9,10,11)

Accuracy<-c(87.93,80.18,86.02,80.86,81.69,83.24,82.92,81.10,78.02,74.68,56.42)

par(pch=22, col="blue")

par(mfrow=c(1,1)) # all plots on one page

opts = c("h")

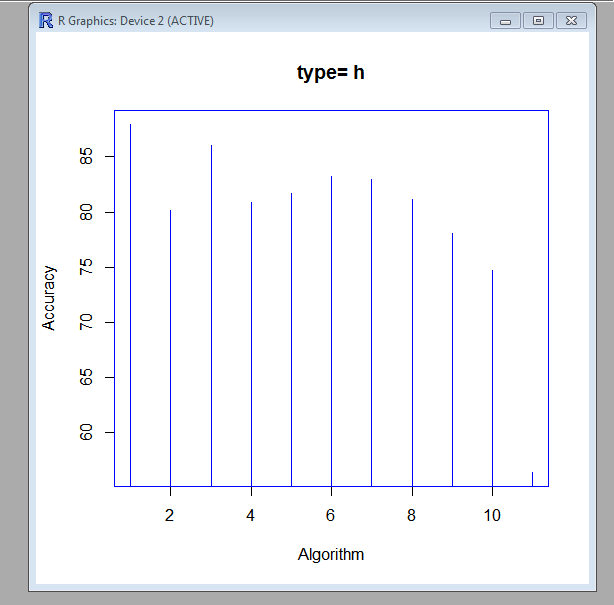
for(i in 1:length(opts)){

heading = paste("type=",opts[i])

plot(Algorithm, Accuracy, type="n", main=heading)

lines(Algorithm, Accuracy, type=opts[i])

}



**CONCLUSION:**

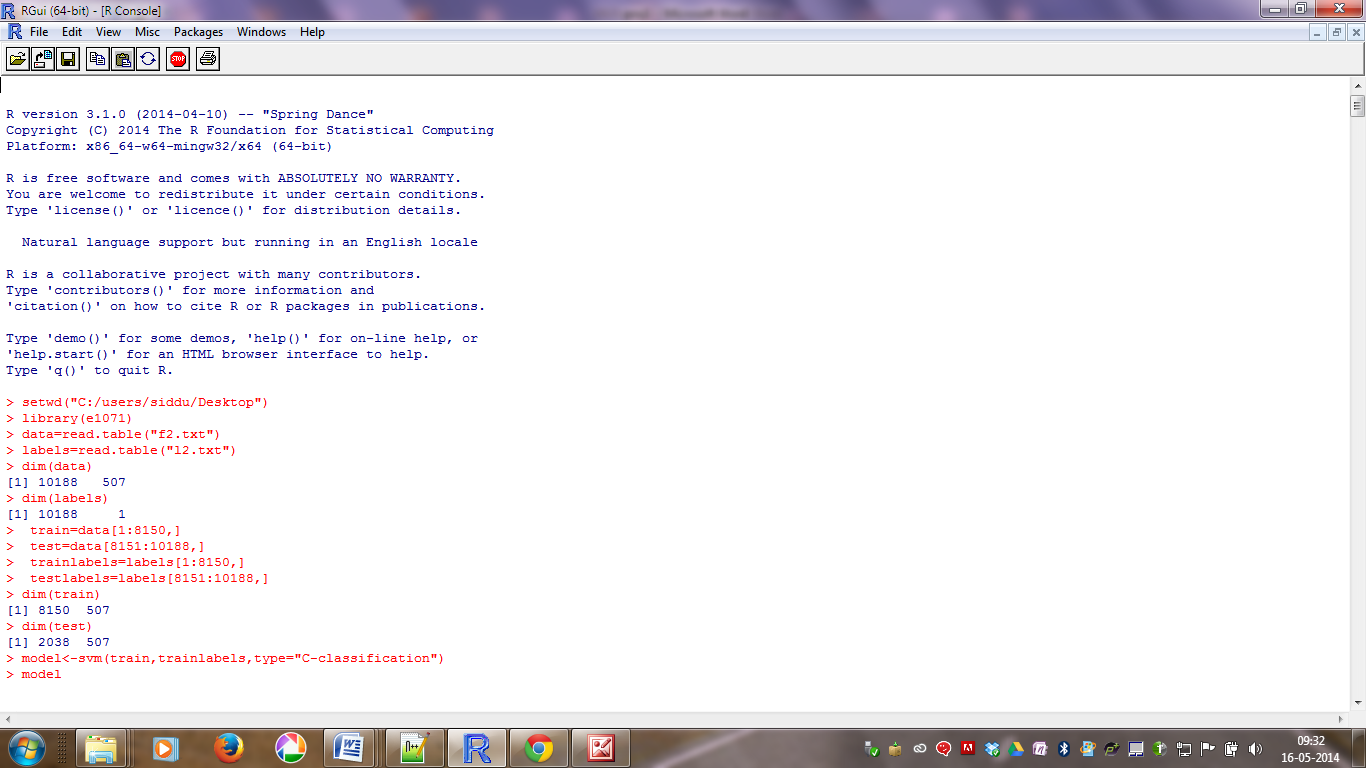
In this project the aurora detection dataset has been classified using the three Machine Learning Algorithms i.e. Support Vector Machine, k-Nearest Neighbour Algorithm and Naïve Bayes.

At last a comparative study and analysis of algorithm accuracies for all the three algorithms has been done. This comparison contains 3 classes 0,1,2. All the three algorithms have provided successful classification results. First two algorithms have shown nearly the same accuracy of around 80% whereas Naïve Bayes shows very less accuracy of about 56.42% when compared to the other 2 algorithms.

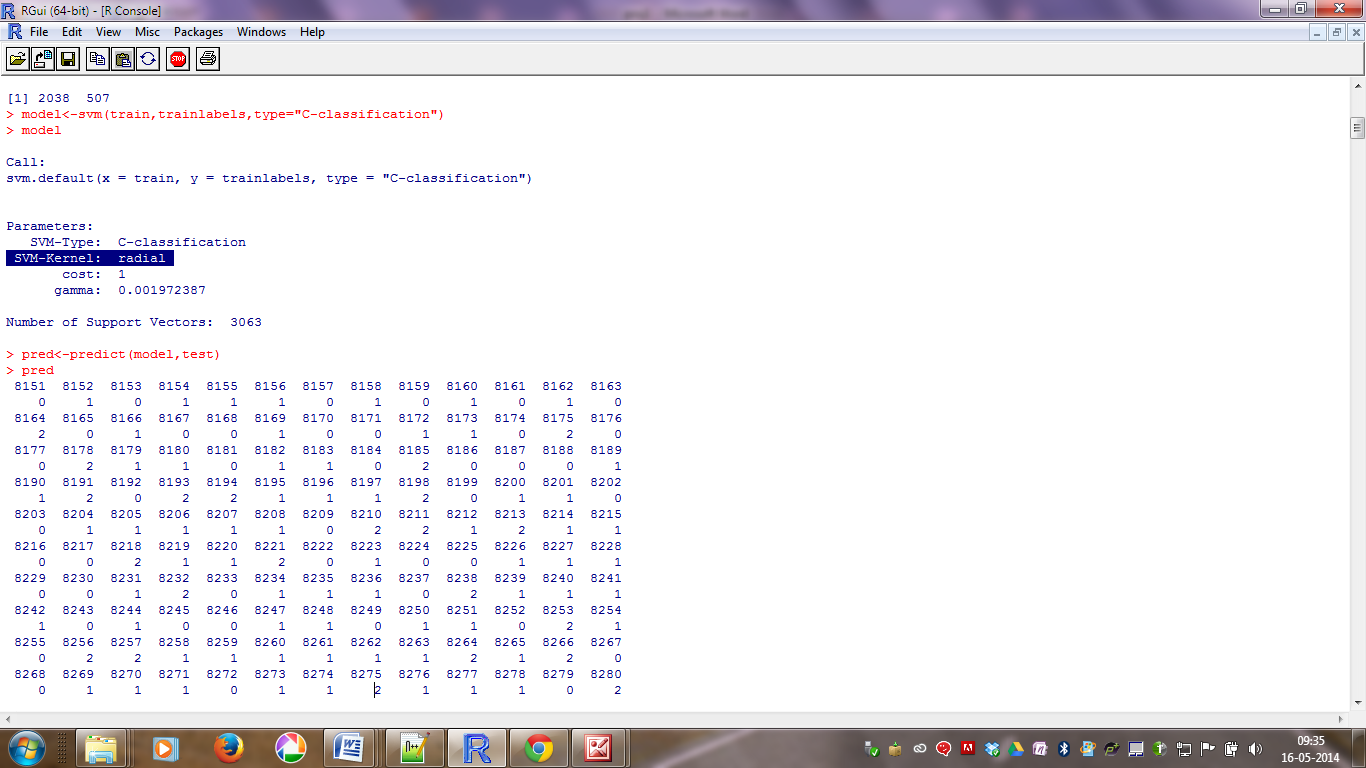
This shows that all the three algorithms best fit the given dataset and provides appreciable results in classification of the given aurora dataset.

**Screenshots:**

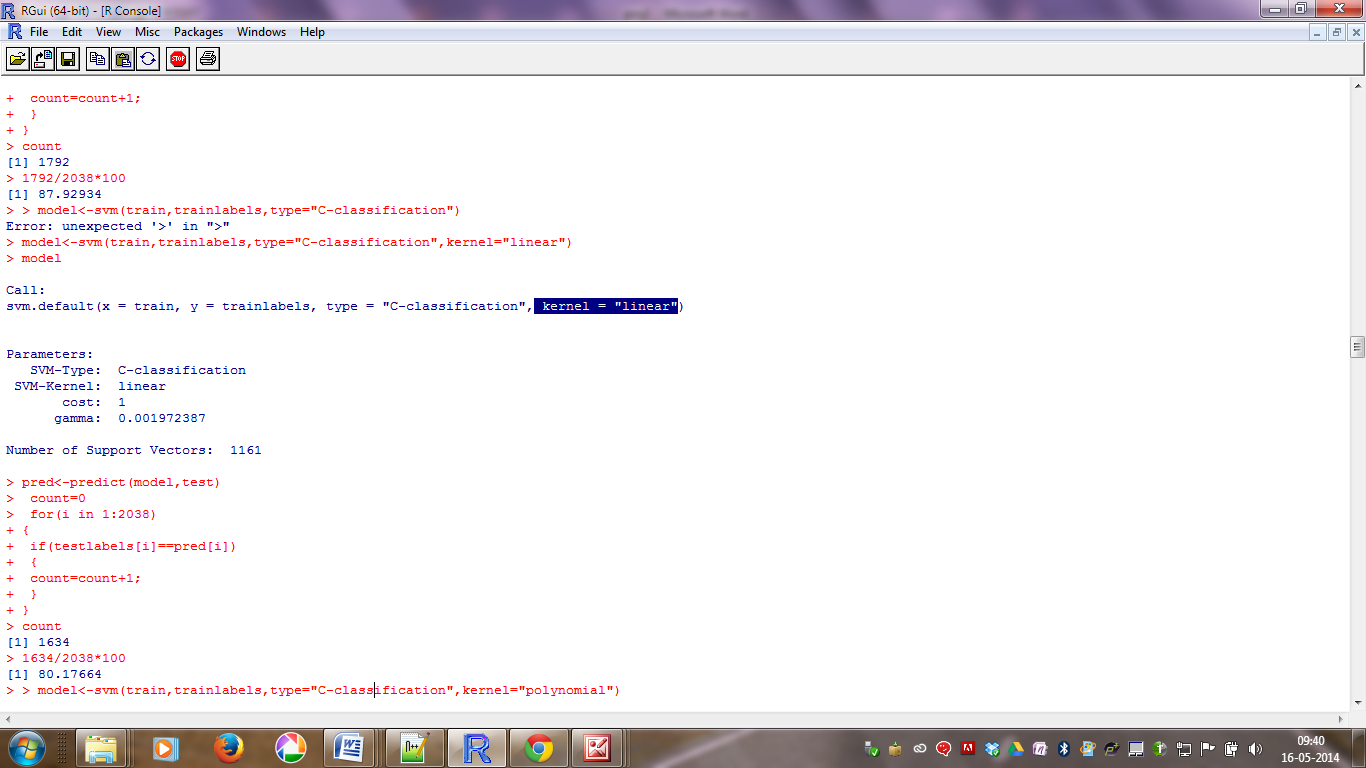
* Dataset division into training(80%) and testing(20%) datasets

****

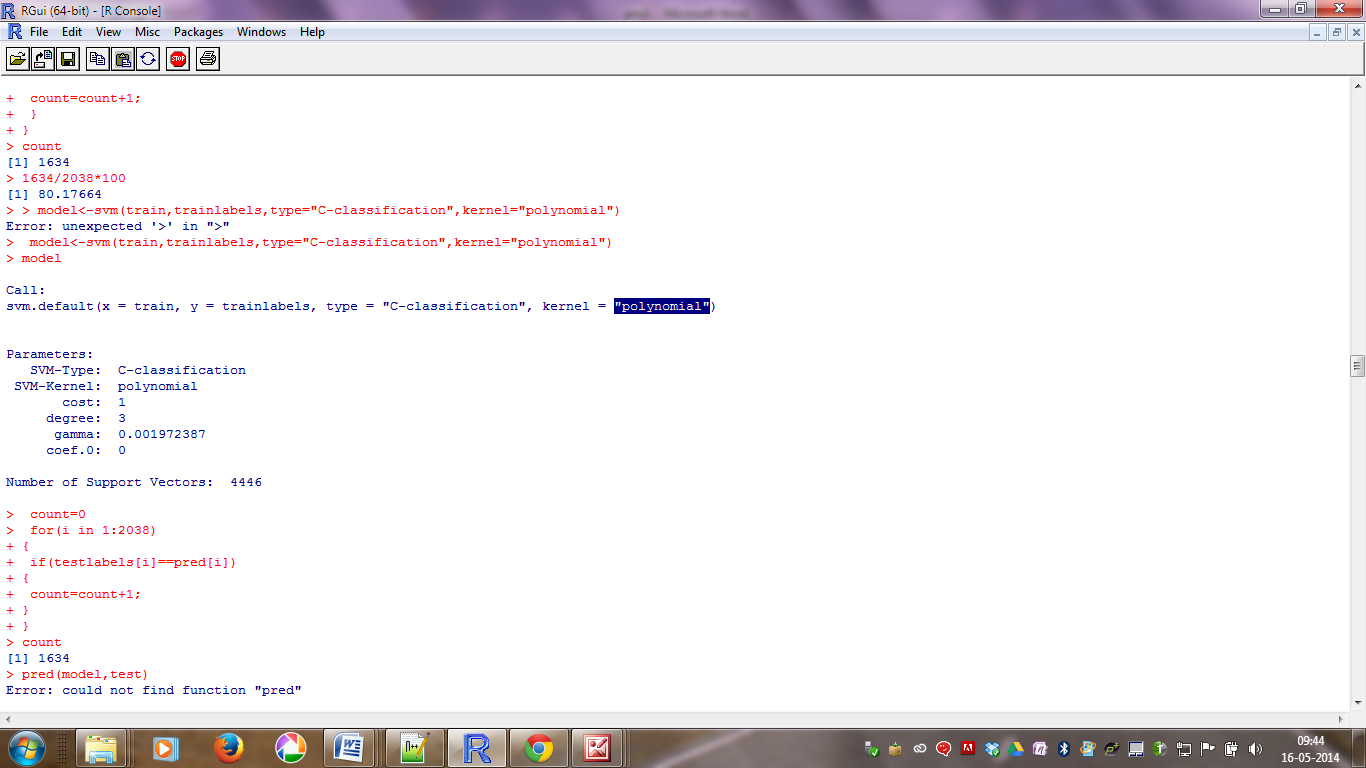
* SVM-Radial Kernel



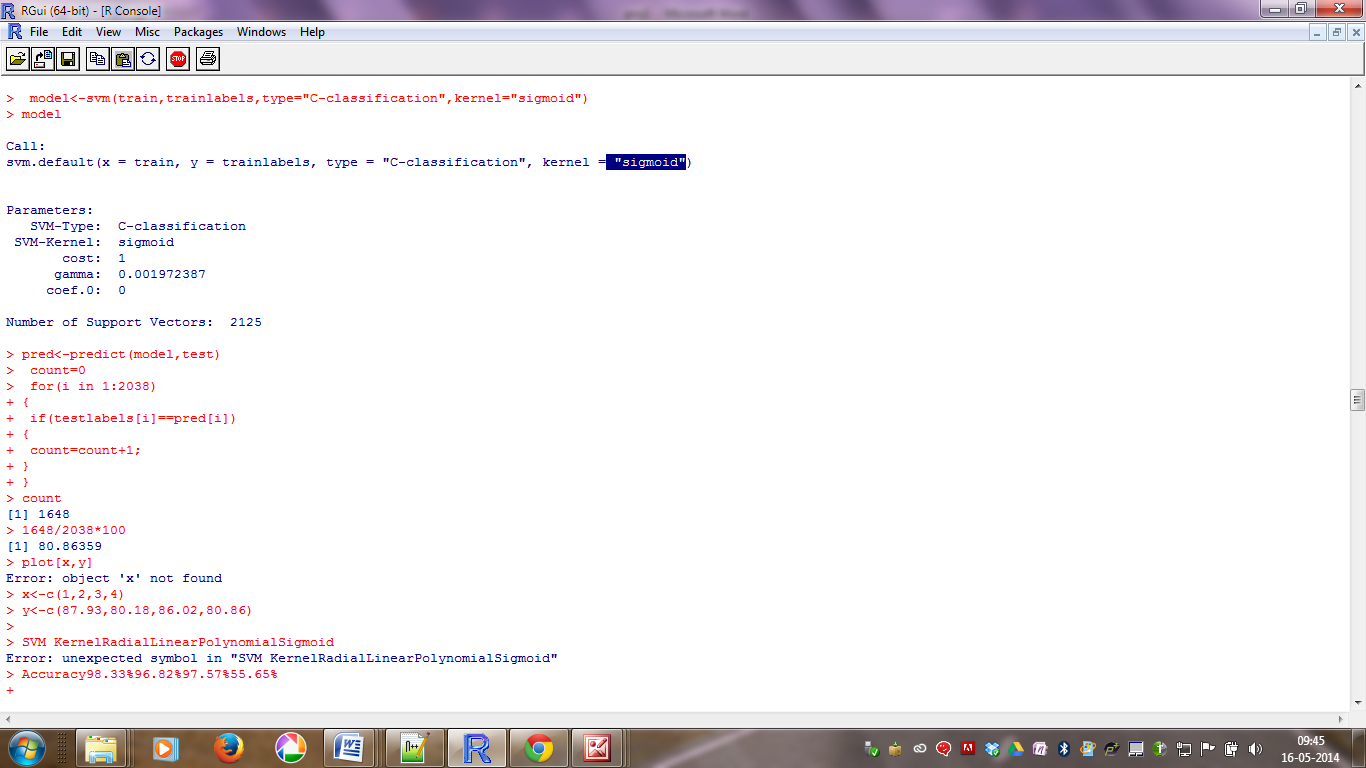
* SVM-Linear Kernel



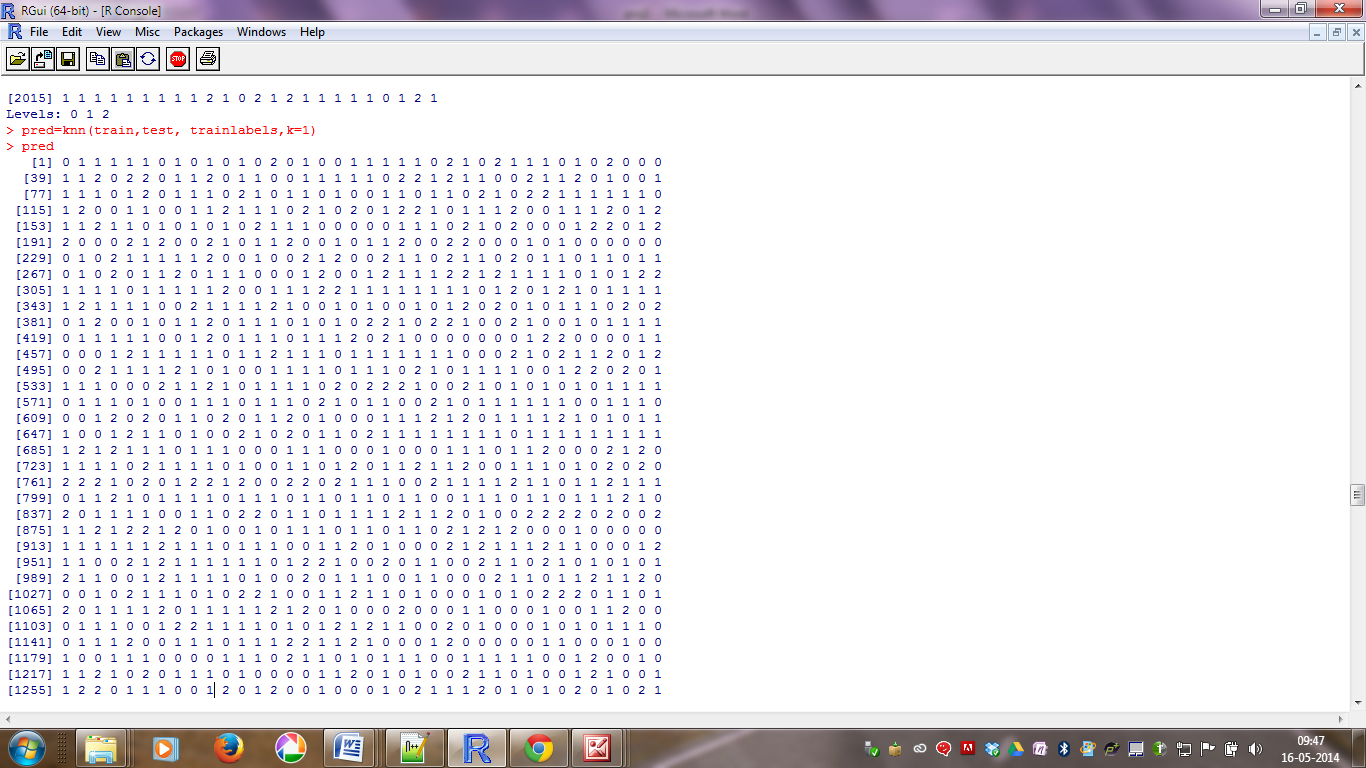
* SVM-Polynomial Kernel

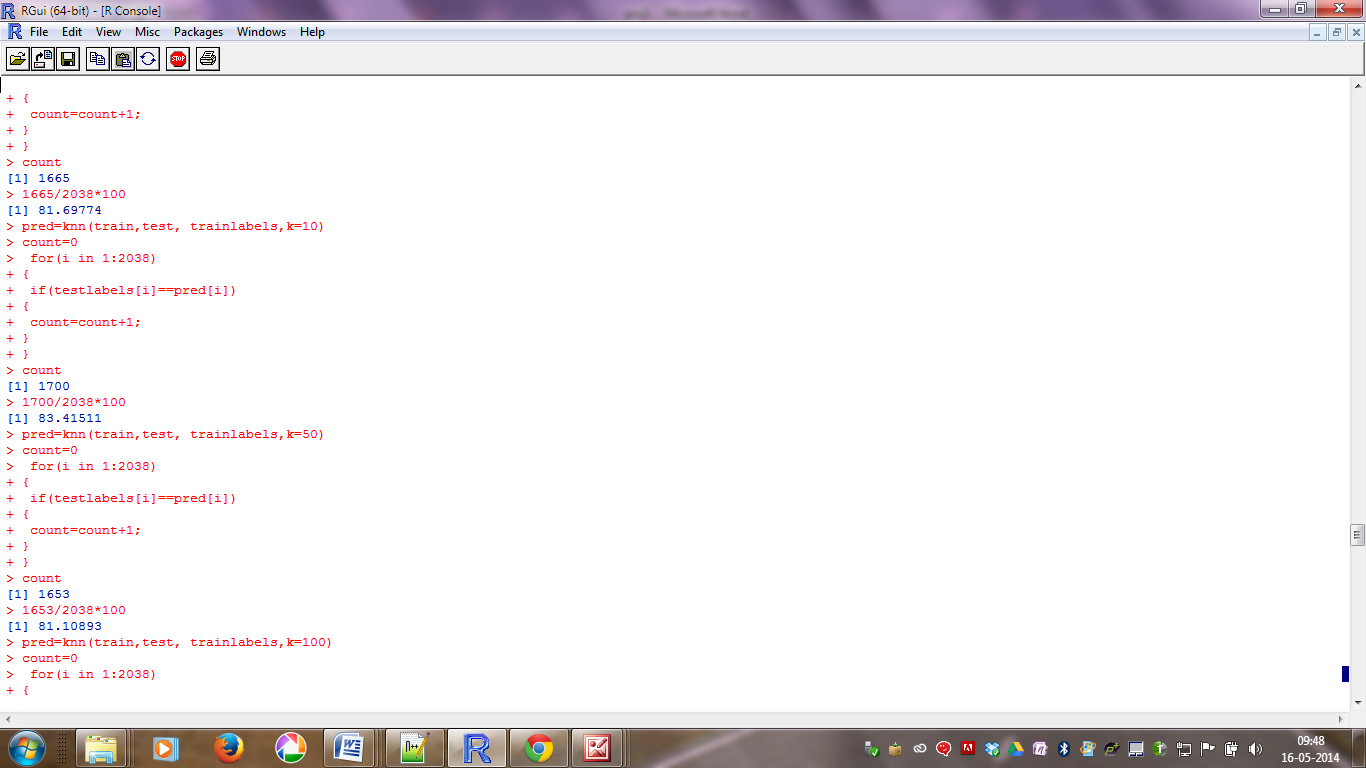


* SVM-Sigmoid Kernel

****

* kNN Classifiers





* Naïve Bayes

