**STATISTICAL DATA MINING 1**

**Homework 4**

**Ashwin Vijayakumar (50249042)**

**Class Number 3**

1)

Using the prostate data set , we try to perform linear regression on the lspa values by selecting the best subset and using methods such as k fold cross validation and 0.632 bootstrap .

The Coefficients obtained from the best subset selection are :

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 1.52629 1.01147 1.509 0.1361

lcavol 0.58264 0.08722 6.680 6.35e-09 \*\*\*

lweight 0.45208 0.22767 1.986 0.0513 .

age -0.02414 0.01191 -2.027 0.0468 \*

lbph 0.12588 0.06783 1.856 0.0680 .

svi 0.62146 0.26265 2.366 0.0210 \*

trainTRUE -0.06347 0.19070 -0.333 0.7403

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.6979 on 65 degrees of freedom

Multiple R-squared: 0.6651, Adjusted R-squared: 0.6341

F-statistic: 21.51 on 6 and 65 DF, p-value: 9.599e-14

Hence we can see that lcavol , age and svi have a pretty good influence on the target value .

We remove all the other predictors from our data set and use only these predictors .

We then fit a linear model using the “lm” function . After prediction on the test data , we obtain the mean squared test error to be :

[1] 0.5720821

The calculated AIC and BIC parameters are :

AIC(linear\_model)

[1] 164.267

BIC(linear\_model)

[1] 189.3103

We perform 5 fold cross validation to get the following mean squared error :

[1] 4.298105

We perform 10 fold cross validation to get the following mean squared error :

attr(cvResults, 'ms')

[1] 2.14853

We see that 10 fold CV has a lower error than the 5 fold case .

The following are the prediction errors for the 5 fold and the 10 fold CV case :

mean((data.test$lpsa - cvResults) ^ 2)

[1] 639.446

mean((data.test$lpsa - cvResults) ^ 2)

[1] 639.0952

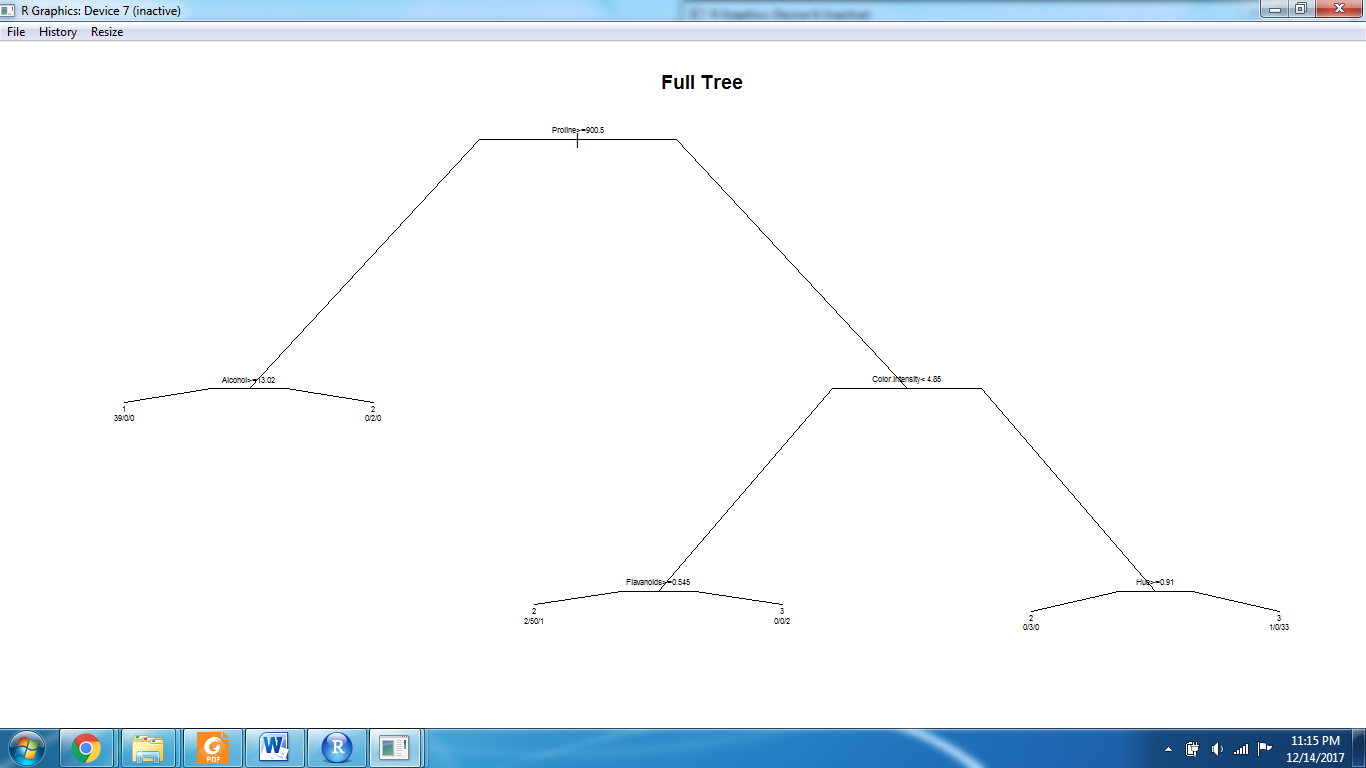
These errors seem to be on the higher side . They should ideally be lesser than the prediction error we obtained from the simple linear model without cross validation .

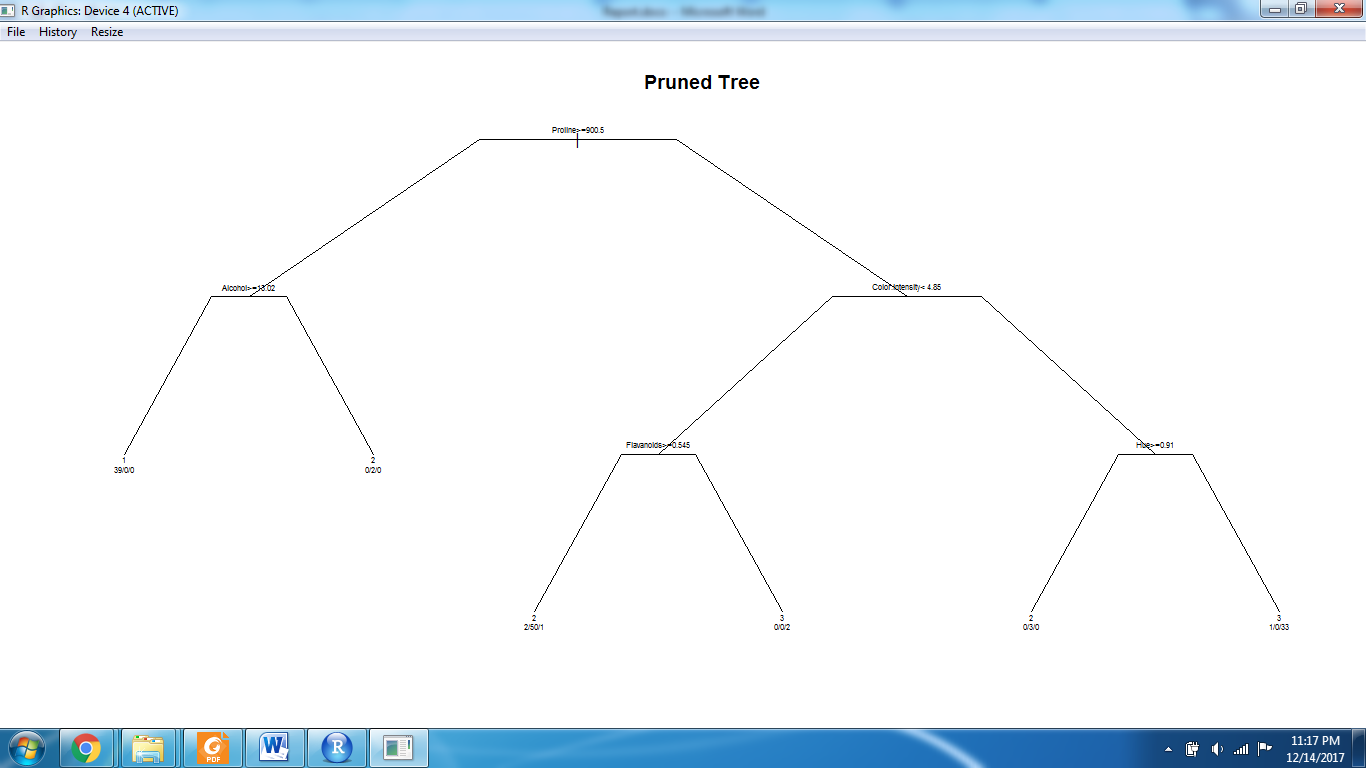
2) The following question deals with the data obtained from a chemical analysis of 178 wines grown over the decade 1970-1979 in the same region of Italy, but derived from three different cultivars (Barolo, Grignolino, Barbera). The Babera wines were predominately from a period that was much later than that of the Barolo and Grignolino wines. The analysis determined the quantities MalicAcid, Ash, AlcAsh, Mg, Phenols, Proa, Color, Hue, OD, and Proline. There are 50 Barolo wines, 71 Grignolino wines, and 48 Barbera wines.

We first aim to split the data into training and test sets in the ratio 3:1

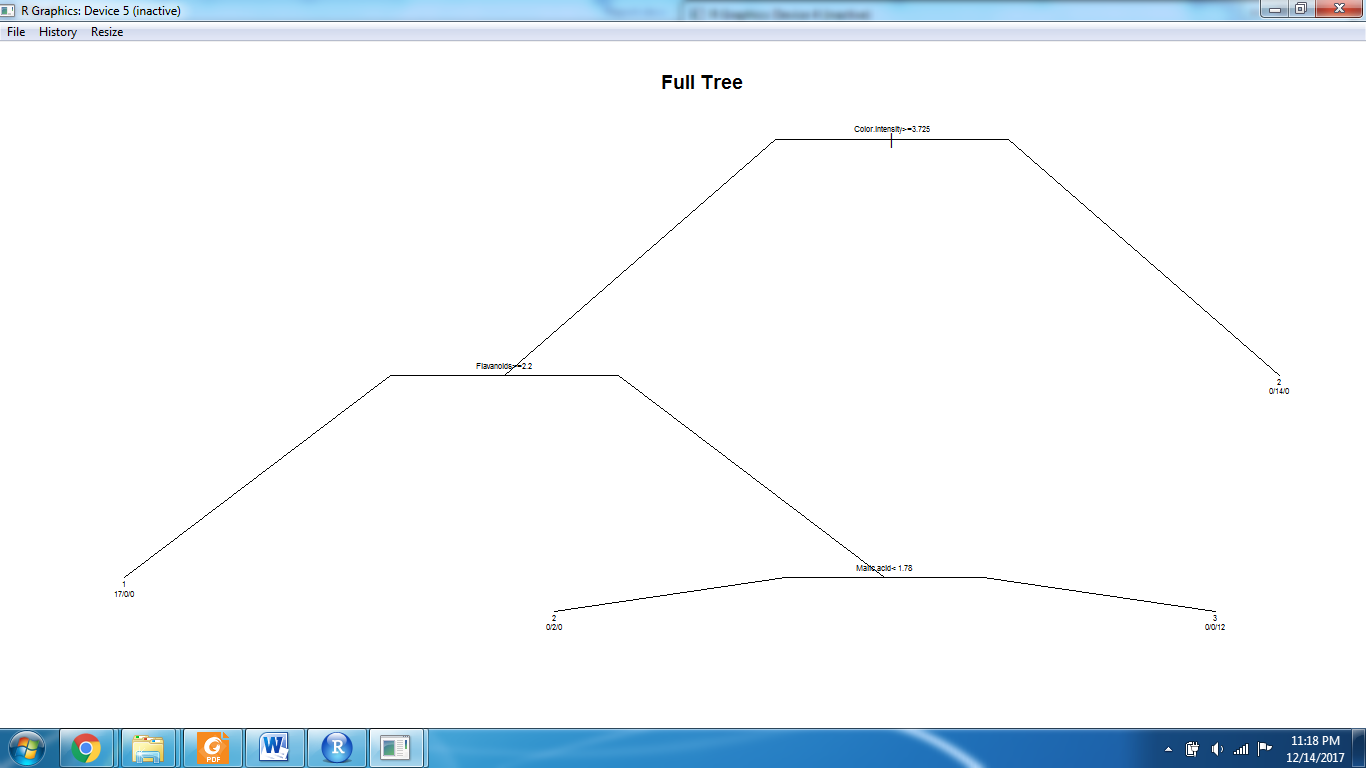
We then aim to use decision trees in order to classify these training and test samples into the three classes (Barolo , Gringolino and Barbera) . We first obtain the full decision tree and then prune it to a smaller tree . The following are the results for the test and training set .

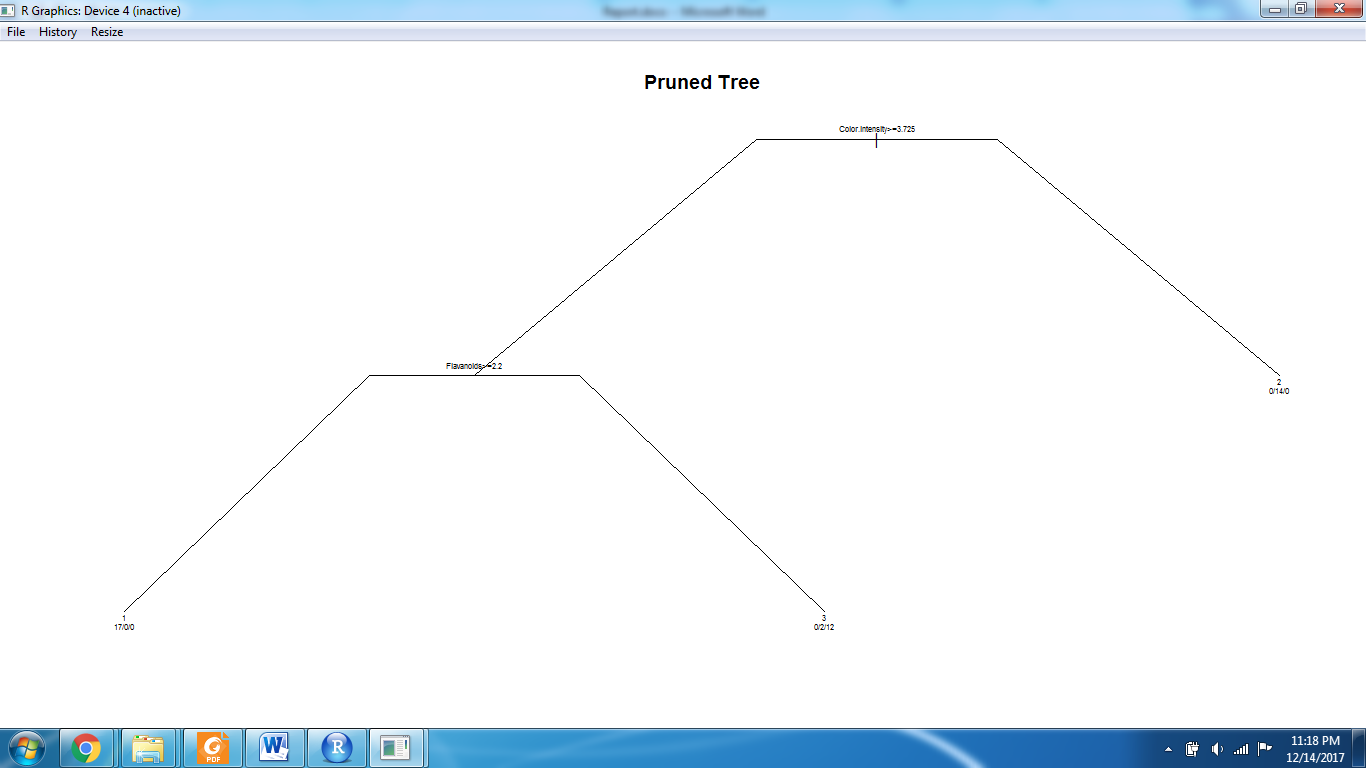
Classification over Train Data Set :





Classification over test data set :





The above is the tree created using Rplot on both the training and test data set . We can see the number of samples classified under each class by looking at the leaf values. Essentially all the samples (in both the training and test set) will be sampled into one of the many lead nodes of the decision tree . The sum of all the values will be equal to 178 .

**For the Training Set we see that 42 fall into category 1 , 55 fall into category 2 and 36 fall into category 3.**

**For the Test Set we see that 18 fall into category 1 , 14 fall into category 2 and 12 fall into category 3.**

We can see that these results agree as 42+55+36 = 133 , which is the size of our training set. Similarly we can see the same for the test set . Hence our model accurately classifies the wine samples into the above three categories.

3) In this exercise , we try to fit decision trees making use of ensemble techniques such as bagging , boosting and random forests . We also compare this to logistic regression and knn to show how ensemble techniques perform better than some non ensemble techniques .

The data set we use is the the IBM Watson data set that is used for prediction attrition rate based on several factors such as time spent with the company , time spent under the current manager , salary , age , etc .

The following are the results that we obtained from the confusion matrix on the different models :

Logistic Regression : Accuracy = 0.8156

Sensitivity : 0.9416

Specificity : 0.4068

KNN : Accuracy : 0.8283

Sensitivity : 0.97727

Specificity : 0.05085

Decision Tree : Accuracy : 0.8131

Sensitivity : 0.9708

Specificity : 0.1017

Random Forest : Accuracy : 0.8202

Sensitivity : 0.9351

Specificity : 0.2203

Bagged Decision Tree : Accuracy : 0.8256

Sensitivity : 0.9286

Specificity : 0.2881

Boosted Decision Tree : Accuracy : 0.8638

Sensitivity : 0.9448

Specificity : 0.4407

Our observations were that Tree based methods in general performed better in this example because most of our variables could be partitioned to . Had we had many variables with continuous values with very minimal influence on the final prediction , then our tree models may not have performed as well .

Our random forest model performed better than our simple decision tree . This makes sense because random forest is able to discover more complex dependencies at the cost of more time for fitting. Also our bagged and boosted version of the forest worked marginally better . This also makes sense as using these techniques can further improve our variance , but at the cost of a slightly higher computational time .

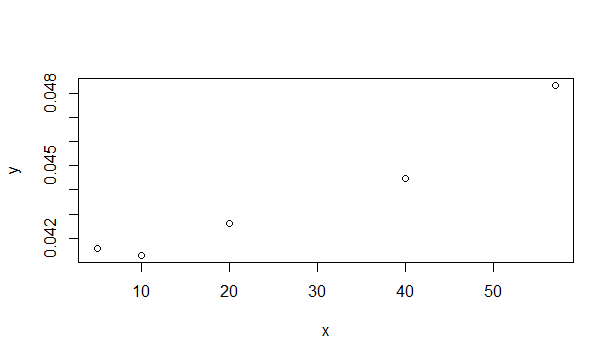
In general , an ensemble method is a technique that combines the predictions from multiple machine learning algorithms together make more accurate predictions than any individual model. Decision trees are sensitive to the specific data on which they are trained. If the training data is changed (e.g. a tree is trained on a subset of the training data) the resulting decision tree can be quite different and in turn the predictions can be quite different.

4)Fit a series of random-forest classifiers to the SPAM data, to explore the sensitivity to m (the number of randomly selected inputs for each tree). Plot both the OOB error as well as the test error against a suitably chosen range of values for m.

In order to do this, we start with a value of 57 , indicating that all 57 predictors should be considered for each split of each tree . Since we have only 57 overall parameters , there’s really no “randomness” here while constructing our forest.

Then we construct 5 more trees with values of mtry = 40 ,30 ,20, 10 and 5 . We also calculate the respective test error for each model and plot it against the mtry values . We observe that the selection of the number of randomly selected inputs for each tree plays a part in the performance of the model .

The following is the graph for test errors.



We see that the error is the least for m try value of around 10 . The value increases upon increasing mtry to 50+ . The graph for OOB errors will also be similar . I ran into some trouble in getting it to work .

5) This exercise deals with making use of a neural network classifier that is trained to predict whether a particular data point belongs to the label “spam” or “email” .

We first numerize all the values in the data set and create an equivalent column whose values represent either a spam or a non spam email (0 means email and 1 means spam ) . We split the data set to test and train using a 75-25 split .

We initially feed in parameters for the neural network such as network size , training epochs size and the learning rate . The train function makes use of cross validation to come up with the best set of hyperparameters for the model . The following is the output of our model .

Training misclassification error:

sum(train.label.nn!=y)/length(y) # 39

[1] **0.6098551**

Test misclassification error :

sum(test.label.nn!=tclass)/length(tclass) # 0.5952381

[1] **0.5942659**

We then run the trained model on the test data set and calculate the misclassification rates . In our case the training and test misclassification rates were almost identical . This points to the fact that the model that we trained has a pretty respectable bias and a really low variance because it avoided over fitting on the training samples . But being a neural network , it has a very low interpretability . It makes up for the lack of interpretability in its ability to learn a wide variety of problems and its inherent flexibility .

The additive model example used in the ESL text book performs better than our neural network model with a misclassification rate of about 5.5 % . This clearly highlights the fact that our model with 10 units of hidden layer , epoch size of 1000 and learning rate of 0.001 is a highly rudimentary implementation. Perhaps this may not be the case if we use neural networks with an increased number of layers and different neural architectures such as Deep Learning , etc .

7) The following exercise reflects upon the use of several kernels and parameter tuning to fit an SVM classifier on the OJ data set and study the effect of these on the training and test errors .

A)0.01 to 10 parameter tuning for linear SVM :

The summary of the tuning is as follows :

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:

cost

**10**

- best performance: **0.1706667**

- Detailed performance results:

**cost error dispersion**

1 0.01000000 0.1853333 0.04046642

2 0.01778279 0.1880000 0.04046642

3 0.03162278 0.1826667 0.04356916

4 0.05623413 0.1840000 0.04159772

5 0.10000000 0.1826667 0.04402020

6 0.17782794 0.1813333 0.04363711

7 0.31622777 0.1760000 0.04435547

8 0.56234133 0.1773333 0.04663491

9 1.00000000 0.1773333 0.04446666

10 1.77827941 0.1773333 0.04124004

11 3.16227766 0.1760000 0.04206990

12 5.62341325 0.1733333 0.03664983

**13 10.00000000 0.1706667 0.03544427**

The best error is 0.1706667 and the parameter is 10 . We use this value to fit a new svm on our training set . We then perform the classification on the test set and obtain the test error as : **14.48**

B) We then use a radial kernel to fit the svm on our training set.

We identify the best parameter selection from the following table :

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:

cost

1

- best performance: 0.184

- Detailed performance results:

cost error dispersion

1 0.01000000 0.3933333 0.03220306

2 0.01778279 0.3933333 0.03220306

3 0.03162278 0.3866667 0.03442652

4 0.05623413 0.2053333 0.03781697

5 0.10000000 0.1920000 0.03781697

6 0.17782794 0.1866667 0.04024616

7 0.31622777 0.1853333 0.04190524

8 0.56234133 0.1866667 0.04024616

**9 1.00000000 0.1840000 0.03965281**

10 1.77827941 0.1866667 0.03718489

11 3.16227766 0.1853333 0.03947808

12 5.62341325 0.1906667 0.03977716

13 10.00000000 0.1920000 0.03985158

The confusion matrix for train and test set is as follows :

train.pred

CH MM

CH 412 43

MM 77 218

test.pred

CH MM

CH 177 21

MM 33 89

We then calculate the training and test error for the best selection to be :

**Training Error : 16.11**

**Test error : 16.87**

We then use a polynomial kernel (with k=2) and carry out the same set of tasks we did for the previous case .

The results are as follows .

- sampling method: 10-fold cross validation

- best parameters:

cost

10

- best performance: **0.1853333**

- Detailed performance results:

cost error dispersion

1 0.01000000 0.3933333 0.03220306

2 0.01778279 0.3746667 0.03689157

3 0.03162278 0.3666667 0.03513642

4 0.05623413 0.3280000 0.04408746

5 0.10000000 0.3173333 0.04523737

6 0.17782794 0.2520000 0.04764063

7 0.31622777 0.2053333 0.03278060

8 0.56234133 0.2040000 0.03208015

9 1.00000000 0.2026667 0.03544427

10 1.77827941 0.1973333 0.03314018

11 3.16227766 0.1893333 0.04014787

12 5.62341325 0.1880000 0.03689157

**13 10.00000000 0.1853333 0.02699794**

The confusion matrix for train and test set is as follows :

train.pred

CH MM

CH 413 42

MM 69 226

test.pred

CH MM

CH 182 16

MM 34 88

**Training Error : 14.8**

**Test error : 15.625**

We observe that for different parameter selections between 0.01 to 10 , we obtain different error rates. Our selection of the best parameter allows us to minimize our future test errors . We also observe that fitting different kernels have different effects . In this case , the polynomial kernel seems to have lower training and test errors as compared to the radial and linear ones . This may have a lot to do with the nature of the true decision boundary of our data set , which most certainly is not linear and maybe some complex function of the predictor variables. In our case, a polynomial estimation of the function performs better than a gaussian one .