Problem Set 7 - Classification Contest

Nishanth Gandhidoss

10 March 2017

## About the data

The file masking.csv describes an experiment in which people saw different letter stimuli (A..Z) briefly, and had to make a forced choice identification, each time between the actual letter and an alternative. The data summarizes the median response time for each stimulus letter for each person. Two different masks were given, a @ in one condition (mask==0) and a # in a second condition (mask==1). A mask is a image that appears immediately before and after the stimulus to disrupt processing. Masks will have different impacts on different letters, and so we may be able to detect which mask a person had by their response patterns.

Lets identify which mask was given based only on the median response times.

train\_all <- read.csv("masking.csv")  
head(train\_all)

## A B C D E F G H I J K L M N O P Q R  
## 1 476 597 494 473 577 542 577 476 417 561 555 458 531 437 455 497 634 541  
## 2 744 804 629 871 647 749 588 461 548 666 469 585 543 487 690 725 682 749  
## 3 456 472 395 373 398 356 392 417 213 452 336 401 400 421 297 292 341 332  
## 4 472 437 177 401 448 444 469 370 493 431 18 350 457 448 433 10 491 402  
## 5 453 449 557 511 443 474 534 453 469 450 443 449 429 452 482 443 531 434  
## 6 615 523 632 658 511 631 710 498 455 598 571 478 513 522 673 498 797 631  
## S T U V W X Y Z mask  
## 1 534 436 538 472 561 434 542 597 0  
## 2 730 527 521 470 486 581 464 608 1  
## 3 356 356 457 455 393 314 313 193 0  
## 4 147 403 430 231 474 474 409 391 1  
## 5 492 439 511 450 448 449 471 451 1  
## 6 473 470 711 497 609 590 473 488 1

Before moving to buliding the model, let read the test file containing 12 data points.

test\_all <- read.csv("testcases-hidden.csv", header = FALSE)  
colnames(test\_all) <- c(LETTERS)  
head(test\_all)

## A B C D E F G H I J K L M N O P Q R  
## 1 681 713 595 634 758 673 501 676 858 617 713 580 594 694 755 612 693 651  
## 2 693 491 531 839 540 522 510 553 526 693 571 463 482 618 611 479 710 590  
## 3 393 417 395 377 417 401 400 515 541 420 381 374 374 495 415 373 424 414  
## 4 652 530 675 631 562 593 622 573 554 622 550 531 553 562 618 583 692 550  
## 5 418 402 378 441 407 444 427 375 399 517 434 397 389 398 469 487 471 417  
## 6 397 437 453 397 451 441 441 581 572 433 480 538 541 518 396 416 441 420  
## S T U V W X Y Z  
## 1 697 713 656 638 672 678 814 538  
## 2 664 441 612 514 550 578 428 579  
## 3 411 395 392 416 395 755 440 397  
## 4 538 599 689 572 589 575 536 631  
## 5 381 407 369 374 421 430 378 389  
## 6 453 418 415 475 454 513 512 434

Next, we are going to bulid some of the below classifcation Machinne learning algorithums to create a model to fit the data and also generalize for unseen data points.

* Logistic regression
* LDA
* QDA
* Naive Bayes
* K-nearest Neighbor
* Decision/partition Tree
* Random Forest
* SVM
* Neural Network

##### Data Preprocessing

Lets check for NA values in the dataset to ensure there will be no problem while buliding the models.

First, will check for the train data

# Installing the required package  
installNewPackage("Amelia")  
library(Amelia)

## Warning: package 'Amelia' was built under R version 3.3.3

## Loading required package: Rcpp

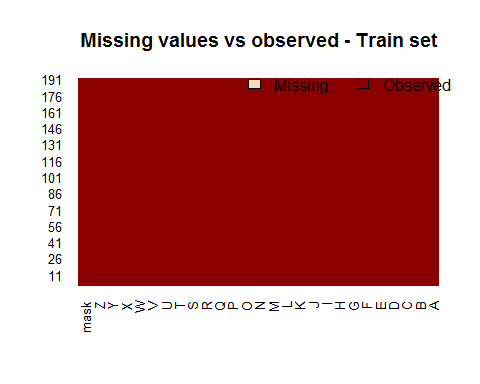
## Warning: package 'Rcpp' was built under R version 3.3.2

## ##   
## ## Amelia II: Multiple Imputation  
## ## (Version 1.7.4, built: 2015-12-05)  
## ## Copyright (C) 2005-2017 James Honaker, Gary King and Matthew Blackwell  
## ## Refer to http://gking.harvard.edu/amelia/ for more information  
## ##

sapply(train\_all, function(x) sum(is.na(x)))

## A B C D E F G H I J K L M N O   
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## P Q R S T U V W X Y Z mask   
## 0 0 0 0 0 0 0 0 0 0 0 0

# Plotting the data to see for NAs  
missmap(train\_all, main = "Missing values vs observed - Train set")



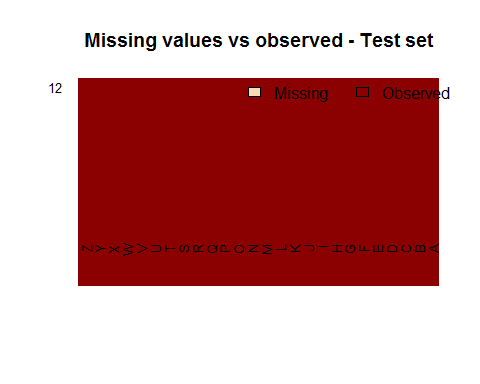
From this we can say that there is no NA or empty data in the train set.

Thus, lets check the same for the test set we have.

sapply(test\_all, function(x) sum(is.na(x)))

## A B C D E F G H I J K L M N O P Q R S T U V W X Y Z   
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

# Plotting the data to see for NAs  
missmap(test\_all, main = "Missing values vs observed - Test set")



As we can see that Test set also has no NA values. Now that we are sure the data is good, we can go to modelling part. But before that we must need to select feautures to use in our prediction. Thus we need to look for some feauture selection process.

#### Feature Selection

In this I am going to use three different methods of feature selection before finalizing the independent variables to use in the model. They are

* Correlation
* Rank by importance
* Recursive Feature Elimination

#### Correlation

I am going to use pearson correlation to see correlation between the variables with help of ggplot heatmaps. I am using pearson because it is much suitable for continuous variables.

# Correlations between variables found to get to know   
# the variables to be added for the buliding the model.  
cor\_mat <- round(cor(train\_all), 3)  
  
cor\_mat[round(abs(cor\_mat["mask", ]), 1) > .2, "mask"]

## A C D G H I O Q Y mask   
## 0.266 0.315 0.272 0.287 -0.288 -0.254 0.448 0.259 -0.276 1.000

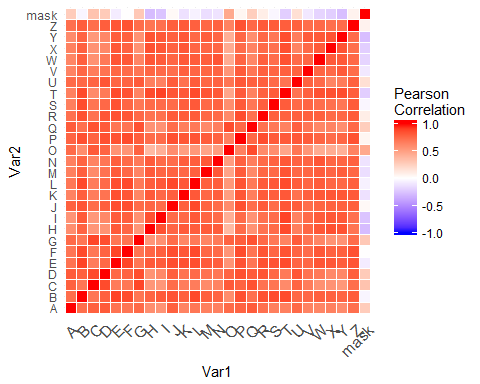
# Create the correlation heatmap with ggplot2  
installNewPackage("reshape2")  
library(reshape2)

## Warning: package 'reshape2' was built under R version 3.3.2

melted\_cor\_mat <- melt(cor\_mat)  
  
# Plotting the heatmaps for correlations  
installNewPackage("ggplot2")  
library(ggplot2)

## Warning: package 'ggplot2' was built under R version 3.3.2

ggplot(data = melted\_cor\_mat, aes(x=Var1, y=Var2, fill=value)) +  
 geom\_tile(color = "white")+  
 scale\_fill\_gradient2(low = "blue", high = "red", mid = "white",  
 midpoint = 0, limit = c(-1,1), space = "Lab",  
 name="Pearson\nCorrelation") +  
 theme\_minimal()+  
 theme(axis.text.x = element\_text(angle = 45, vjust = 1,  
 size = 12, hjust = 1))+  
 coord\_fixed()



So here we have the following variables selected from the correlation method. They are

* A
* C
* D
* G
* H
* I
* O
* Q
* Y

But is it reliable to go with just correlation for buliding the model. Lets look for some more methods.

#### Rank by Importance

The importance of features can be estimated from data by building a model. Some methods like decision trees have a built in mechanism to report on variable importance. For other algorithms, the importance can be estimated using a ROC curve analysis conducted for each attribute.

installNewPackage("caret")  
library(caret)

## Warning: package 'caret' was built under R version 3.3.2

## Loading required package: lattice

installNewPackage("mlbench")  
library(mlbench)

## Warning: package 'mlbench' was built under R version 3.3.3

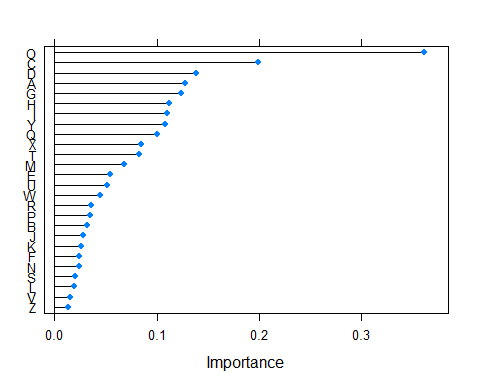
set.seed(690)  
  
# prepare training scheme  
control <- trainControl(method = "repeatedcv", number = 10, repeats = 3)  
# train the model  
model <- train(as.factor(mask) ~ ., data = train\_all, method = "lvq", preProcess = "scale",  
 trControl = control)

## Loading required package: class

# estimate variable importance  
importance <- varImp(model, scale=FALSE)  
# summarize importance  
print(importance)

## loess r-squared variable importance  
##   
## only 20 most important variables shown (out of 26)  
##   
## Overall  
## O 0.36126  
## C 0.19897  
## D 0.13837  
## A 0.12816  
## G 0.12367  
## H 0.11203  
## I 0.11015  
## Y 0.10837  
## Q 0.10036  
## X 0.08491  
## T 0.08282  
## M 0.06830  
## E 0.05398  
## U 0.05141  
## W 0.04454  
## R 0.03628  
## P 0.03487  
## B 0.03152  
## J 0.02847  
## K 0.02618

# plot importance  
plot(importance)



The varImp is then used to estimate the variable importance, which is printed and plotted. It shows that the variables O, C, D, A, G, H, I, Y ,Q, S, t and M are the top 12 most important attributes in the dataset. But the problem here is this method does not give us the cut off. The cut off is decided by the me here. So, I thought of looking at a one more feature selection wrapper method.

#### Recursive Feature Elimination

Recursive Feature selection Elimination method gives us the advantage in which it will approximately gives us the cut off value we can use to minimize the generalization error that arise while predicting with the model.

set.seed(100)  
# define the control using a random forest selection function  
control <- rfeControl(functions=rfFuncs, method="cv", number=10)  
# run the RFE algorithm  
results <- rfe(train\_all[, - 27], as.factor(train\_all[, 27]), sizes=c(1:8), rfeControl=control)

## Loading required package: randomForest

## Warning: package 'randomForest' was built under R version 3.3.2

## randomForest 4.6-12

## Type rfNews() to see new features/changes/bug fixes.

##   
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':  
##   
## margin

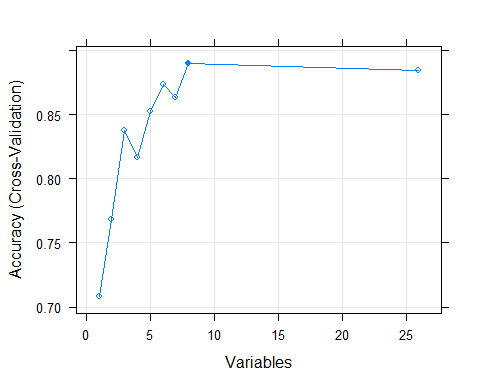
# summarize the results  
print(results)

##   
## Recursive feature selection  
##   
## Outer resampling method: Cross-Validated (10 fold)   
##   
## Resampling performance over subset size:  
##   
## Variables Accuracy Kappa AccuracySD KappaSD Selected  
## 1 0.7079 0.4090 0.07028 0.14460   
## 2 0.7684 0.5298 0.05656 0.11684   
## 3 0.8380 0.6728 0.03637 0.07468   
## 4 0.8166 0.6302 0.05541 0.11614   
## 5 0.8527 0.7042 0.05213 0.10059   
## 6 0.8737 0.7456 0.07364 0.14711   
## 7 0.8632 0.7238 0.04679 0.09489   
## 8 0.8904 0.7787 0.03685 0.07379 \*  
## 26 0.8843 0.7628 0.06489 0.13521   
##   
## The top 5 variables (out of 8):  
## O, C, D, Y, I

# list the chosen features  
predictors(results)

## [1] "O" "C" "D" "Y" "I" "H" "A" "T"

# plot the results  
plot(results, type=c("g", "o"))



Thus, from this method we have the following variables that could possibly explain with 0.89 accuracy for the model generated with the 10 cross validation using the caret package.

* O
* C
* D
* Y
* I
* H
* A
* T

significant\_features <- c("O", "C", "D", "Y", "I", "H", "A", "T", "mask")  
train\_after\_fs <- train\_all[, significant\_features]  
test\_after\_fs <- test\_all[, significant\_features[-9]]

#### Preparing the data for the 10 fold cross validation

Although, while doing the feauture selection we had the cross validation. I am creating the 10 fold of the train data so that I can use it if required while developing the models like KNN. I am going to perform 10 - fold cross validation to ensure the generalization of the model. First lets prepare the data for the cross validation with maximum possiblity of equal distribution of class labels in each folds.

# Data preprocessing for 10 fold cross validation  
  
# Setting the seed for reproduciablility  
set.seed(690)  
  
# NO of fold  
k\_fold <- 10  
  
# Creating the folds variable in the data frame  
# music$folds <- createFolds(music$Top10, k = n\_fold, list = FALSE)  
cv\_index\_list <- createFolds(train\_all$mask, k = k\_fold, list = TRUE, returnTrain = FALSE)  
  
# Checking the class label proportions  
for(i in 1:k\_fold) {  
 fold <- train\_all$mask[cv\_index\_list[[i]]]  
 fold\_length <- length(fold)  
 class0\_prop <- round(length(fold[fold == 0]) / fold\_length, 2)  
 class1\_prop <- round(length(fold[fold == 1]) / fold\_length, 2)  
 print(paste("Fold" , i , "length", fold\_length, sep = " "))  
 print(paste("Class Label(0)", round(class0\_prop, 2), sep = " "))  
 print(paste("Class Label(1)", round(class1\_prop, 2), sep = " "))  
}

## [1] "Fold 1 length 19"  
## [1] "Class Label(0) 0.42"  
## [1] "Class Label(1) 0.58"  
## [1] "Fold 2 length 19"  
## [1] "Class Label(0) 0.53"  
## [1] "Class Label(1) 0.47"  
## [1] "Fold 3 length 19"  
## [1] "Class Label(0) 0.53"  
## [1] "Class Label(1) 0.47"  
## [1] "Fold 4 length 19"  
## [1] "Class Label(0) 0.53"  
## [1] "Class Label(1) 0.47"  
## [1] "Fold 5 length 20"  
## [1] "Class Label(0) 0.2"  
## [1] "Class Label(1) 0.8"  
## [1] "Fold 6 length 19"  
## [1] "Class Label(0) 0.53"  
## [1] "Class Label(1) 0.47"  
## [1] "Fold 7 length 19"  
## [1] "Class Label(0) 0.42"  
## [1] "Class Label(1) 0.58"  
## [1] "Fold 8 length 19"  
## [1] "Class Label(0) 0.42"  
## [1] "Class Label(1) 0.58"  
## [1] "Fold 9 length 19"  
## [1] "Class Label(0) 0.42"  
## [1] "Class Label(1) 0.58"  
## [1] "Fold 10 length 19"  
## [1] "Class Label(0) 0.53"  
## [1] "Class Label(1) 0.47"

#### Buliding the models

As we have completed the so called tough part of data science that is the data preprocessing and feauture selection, lets dive into buliding the models. Here I am going to bulid the following seven classification models.

* Logistic Regression
* Naive Bayes
* K-nearest Neighbor
* Decision/partition Tree
* Random Forest
* SVM
* Neural Network

#### Logistic Regression

Logistic Regression is a classification not a regression algorithm. It is used to estimate discrete values (Binary values like 0/1, yes/no, true/false) based on given set of independent variable(s). In simple words, it predicts the probability of occurrence of an event by fitting data to a logit function. Hence, it is also known as logit regression. Since, it predicts the probability, its output values lies between 0 and 1 (as expected).

installNewPackage("MLmetrics")  
library(MLmetrics)

##   
## Attaching package: 'MLmetrics'

## The following object is masked from 'package:base':  
##   
## Recall

# Setting the seed  
set.seed(1)  
  
for(data in c("train\_all", "train\_after\_fs")) {  
   
 # Intailize variables  
 total\_acc <- 0  
 total\_error\_rate <- 0  
 total\_AUC <- 0  
   
 # To differentiate full and feature subset model  
 if(data == "train\_all") {  
 msg <- "With the full model"  
 data <- train\_all  
 } else {  
 msg <- "After feature selection"  
 data <- train\_after\_fs  
 }  
   
 # Running the Logistic Regression with 10 - fold cross validation  
 # Accuracy, error, AUC are rounded off to two decimal points  
 for(fold in 1:k\_fold) {  
 train\_set <- data[-cv\_index\_list[[fold]], ]   
 test\_set <- data[cv\_index\_list[[fold]], ]   
   
 # Fitting the Logistic Regression model  
 log\_fit <- glm(mask ~ ., train\_set, family = binomial(), maxit = 100)  
   
 # Calculating Accuracy. Error, AUC  
 test\_prediction <- predict(log\_fit, test\_set, type = "response")  
 test\_prediction <- ifelse(test\_prediction > 0.5,1,0)  
 accuracy <- round(Accuracy(test\_prediction, test\_set$mask), 2)  
 error\_rate <- 1 - accuracy  
 AUC <- round(AUC(test\_prediction, test\_set$mask), 2)  
 total\_acc <- total\_acc + accuracy  
 total\_error\_rate <- total\_error\_rate + error\_rate  
 total\_AUC <- total\_AUC + AUC  
 }  
 cat("==========================================================\n\n")  
 cat(msg, "\n\n")  
 cat("==========================================================\n\n")  
 cat(paste("Overall accuracy is", round(total\_acc / k\_fold, 2), sep = " "), "\n\n")  
 cat(paste("Overall error is", round(total\_error\_rate / k\_fold, 2), sep =" "), "\n\n")  
 cat(paste("Overall area under the curve(AUC) is", round(total\_AUC / k\_fold, 2), sep = " "), "\n")  
 cat("\n==========================================================\n\n")  
}

## ==========================================================  
##   
## With the full model   
##   
## ==========================================================  
##   
## Overall accuracy is 0.89   
##   
## Overall error is 0.11   
##   
## Overall area under the curve(AUC) is 0.88   
##   
## ==========================================================  
##   
## ==========================================================  
##   
## After feature selection   
##   
## ==========================================================  
##   
## Overall accuracy is 0.87   
##   
## Overall error is 0.13   
##   
## Overall area under the curve(AUC) is 0.86   
##   
## ==========================================================

So looking at the overall accuracy from both the model, it is better to go with smaller model that has almost equal accuracy to the full model.

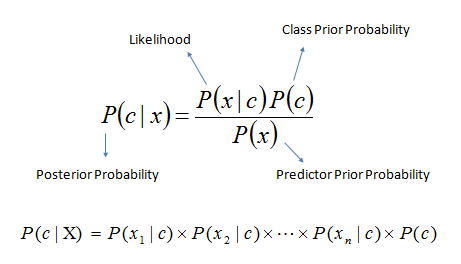
# Training the model again  
log\_fit <- glm(mask ~ ., data = train\_after\_fs, family = binomial(), maxit = 100)  
# Predicting the test set on the model  
log\_pred <- predict(log\_fit, test\_after\_fs, type = "response")  
log\_pred <- ifelse(log\_pred >= 0.5, 1, 0)  
log\_pred

## 1 2 3 4 5 6 7 8 9 10 11 12   
## 0 1 0 1 1 0 0 1 0 1 0 1

Thus, the above we have the prediction for the test set using Logistic Regression classifier with smaller model.

#### Naive Bayes

Naive Bayes is a classification technique based on Bayes' theorem with an assumption of independence between predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.



installNewPackage("e1071")  
library(e1071)

## Warning: package 'e1071' was built under R version 3.3.2

for(data in c("train\_all", "train\_after\_fs")) {  
   
 # Intailize variables  
 total\_acc <- 0  
 total\_error\_rate <- 0  
 total\_AUC <- 0  
   
 # To differentiate full and feature subset model  
 if(data == "train\_all") {  
 msg <- "With the full model"  
 data <- train\_all  
 } else {  
 msg <- "After feature selection"  
 data <- train\_after\_fs  
 }  
   
 # Running the Naive Bayes with 10 - fold cross validation  
 # Accuracy, error, AUC are rounded off to two decimal points  
 for(fold in 1:k\_fold) {  
 train\_set <- data[-cv\_index\_list[[fold]], ]   
 test\_set <- data[cv\_index\_list[[fold]], ]   
   
 # Fitting the Naive bayes model  
 nb\_fit <- naiveBayes(as.factor(mask) ~ ., data = train\_set)  
   
 # Calculating Accuracy. Error, AUC  
 test\_prediction <- predict(nb\_fit, test\_set, type = "class")  
   
 accuracy <- round(Accuracy(test\_prediction, test\_set$mask), 2)  
 error\_rate <- 1 - accuracy  
 AUC <- round(AUC(test\_prediction, test\_set$mask), 2)  
 # print(paste("Test set fold #", fold, sep = " "))  
 # cat("\n")  
 # print(paste("Accuracy is", accuracy, sep = " "))  
 # print(paste("Error is", error\_rate, sep =" "))  
 # print(paste("Area under the curve(AUC) is", AUC, sep = " "))  
 # cat("\n")  
 total\_acc <- total\_acc + accuracy  
 total\_error\_rate <- total\_error\_rate + error\_rate  
 total\_AUC <- total\_AUC + AUC  
 }  
 cat("==========================================================\n\n")  
 cat(msg, "\n\n")  
 cat("==========================================================\n\n")  
 cat(paste("Overall accuracy is", round(total\_acc / k\_fold, 2), sep = " "), "\n\n")  
 cat(paste("Overall error is", round(total\_error\_rate / k\_fold, 2), sep =" "), "\n\n")  
 cat(paste("Overall area under the curve(AUC) is", round(total\_AUC / k\_fold, 2), sep = " "), "\n")  
 cat("\n==========================================================\n\n")  
}

## ==========================================================  
##   
## With the full model   
##   
## ==========================================================  
##   
## Overall accuracy is 0.74   
##   
## Overall error is 0.26   
##   
## Overall area under the curve(AUC) is 0.74   
##   
## ==========================================================  
##   
## ==========================================================  
##   
## After feature selection   
##   
## ==========================================================  
##   
## Overall accuracy is 0.82   
##   
## Overall error is 0.18   
##   
## Overall area under the curve(AUC) is 0.83   
##   
## ==========================================================

Naive Bayesian model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Here we can see that the feature selection model with less number of predictor outperforms the full model by nearly 8 percent so lets predict the Althogh we have much smaller datset. Thus, lets predict the given test set with the smaller model.

# Training the model again  
nb\_fit <- naiveBayes(as.factor(mask) ~ ., data = train\_after\_fs)  
# Predicting the test set on the model  
nb\_pred <- predict(nb\_fit, test\_after\_fs, type = "class")  
nb\_pred

## [1] 1 1 0 1 1 0 0 1 0 1 0 1  
## Levels: 0 1

Thus, the above we have the prediction for the test set using naive bayes classifier with smaller model.

#### K - Nearest Neighbour

It can be used for both classification and regression problems. However, it is more widely used in classification problems in the industry. K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases by a majority vote of its k neighbors. The case being assigned to the class is most common amongst its K nearest neighbors measured by a distance function.

These distance functions can be Euclidean, Manhattan, Minkowski and Hamming distance. First three functions are used for continuous function and fourth one (Hamming) for categorical variables. If K = 1, then the case is simply assigned to the class of its nearest neighbor. At times, choosing K turns out to be a challenge while performing KNN modeling.

k\_values <- c(1, 3, 5, 7, 9)  
  
for(k in k\_values) {   
 for(data in c("train\_all", "train\_after\_fs")) {  
 # Intailize variables  
 total\_acc <- 0  
 total\_error\_rate <- 0  
 total\_AUC <- 0  
   
 # To differentiate full and feature subset model  
 if(data == "train\_all") {  
 msg <- "With the full model"  
 data <- train\_all  
 } else {  
 msg <- "After feature selection"  
 data <- train\_after\_fs  
 }  
   
 # Running the KNN with 10 - fold cross validation  
 # Accuracy, error, AUC are rounded off to two decimal points  
 for(fold in 1:k\_fold) {  
 train\_set <- data[-cv\_index\_list[[fold]], ]   
 test\_set <- data[cv\_index\_list[[fold]], ]   
   
 # Fitting the Naive bayes model  
 knn\_predicted\_values <- knn(train = train\_set, test = test\_set, cl = train\_set$mask, k = k)  
   
 # Calculating Accuracy. Error, AUC  
 accuracy <- round(Accuracy(knn\_predicted\_values, test\_set$mask), 2)  
 error\_rate <- 1 - accuracy  
 AUC <- round(AUC(knn\_predicted\_values, test\_set$mask), 2)  
 total\_acc <- total\_acc + accuracy  
 total\_error\_rate <- total\_error\_rate + error\_rate  
 total\_AUC <- total\_AUC + AUC  
 }  
 cat("==========================================================\n\n")  
 cat("K value = ", k, "\n\n", msg, "\n\n")  
 cat("==========================================================\n\n")  
 cat(paste("Overall accuracy is", round(total\_acc / k\_fold, 2), sep = " "), "\n\n")  
 cat(paste("Overall error is", round(total\_error\_rate / k\_fold, 2), sep =" "), "\n\n")  
 cat(paste("Overall area under the curve(AUC) is", round(total\_AUC / k\_fold, 2), sep = " "), "\n")  
 cat("\n==========================================================\n\n")  
 }  
}

## ==========================================================  
##   
## K value = 1   
##   
## With the full model   
##   
## ==========================================================  
##   
## Overall accuracy is 0.83   
##   
## Overall error is 0.17   
##   
## Overall area under the curve(AUC) is 0.83   
##   
## ==========================================================  
##   
## ==========================================================  
##   
## K value = 1   
##   
## After feature selection   
##   
## ==========================================================  
##   
## Overall accuracy is 0.88   
##   
## Overall error is 0.12   
##   
## Overall area under the curve(AUC) is 0.88   
##   
## ==========================================================  
##   
## ==========================================================  
##   
## K value = 3   
##   
## With the full model   
##   
## ==========================================================  
##   
## Overall accuracy is 0.85   
##   
## Overall error is 0.15   
##   
## Overall area under the curve(AUC) is 0.85   
##   
## ==========================================================  
##   
## ==========================================================  
##   
## K value = 3   
##   
## After feature selection   
##   
## ==========================================================  
##   
## Overall accuracy is 0.85   
##   
## Overall error is 0.15   
##   
## Overall area under the curve(AUC) is 0.84   
##   
## ==========================================================  
##   
## ==========================================================  
##   
## K value = 5   
##   
## With the full model   
##   
## ==========================================================  
##   
## Overall accuracy is 0.83   
##   
## Overall error is 0.17   
##   
## Overall area under the curve(AUC) is 0.83   
##   
## ==========================================================  
##   
## ==========================================================  
##   
## K value = 5   
##   
## After feature selection   
##   
## ==========================================================  
##   
## Overall accuracy is 0.85   
##   
## Overall error is 0.15   
##   
## Overall area under the curve(AUC) is 0.84   
##   
## ==========================================================  
##   
## ==========================================================  
##   
## K value = 7   
##   
## With the full model   
##   
## ==========================================================  
##   
## Overall accuracy is 0.81   
##   
## Overall error is 0.19   
##   
## Overall area under the curve(AUC) is 0.81   
##   
## ==========================================================  
##   
## ==========================================================  
##   
## K value = 7   
##   
## After feature selection   
##   
## ==========================================================  
##   
## Overall accuracy is 0.85   
##   
## Overall error is 0.15   
##   
## Overall area under the curve(AUC) is 0.84   
##   
## ==========================================================  
##   
## ==========================================================  
##   
## K value = 9   
##   
## With the full model   
##   
## ==========================================================  
##   
## Overall accuracy is 0.81   
##   
## Overall error is 0.19   
##   
## Overall area under the curve(AUC) is 0.82   
##   
## ==========================================================  
##   
## ==========================================================  
##   
## K value = 9   
##   
## After feature selection   
##   
## ==========================================================  
##   
## Overall accuracy is 0.86   
##   
## Overall error is 0.14   
##   
## Overall area under the curve(AUC) is 0.86   
##   
## ==========================================================

Here we have feature subset model (smaller model) has better performance in k values. With KNN, it looks like the we have best accuracy from k value = 1 ie) 0.88. But in genreal using k = 1 wont be a good option considering the genralization in consideration. Thus its better to go with K = 9. KNN is computaionally intense since we need to calculate the distance at each step and higher value of K will make it much more difficult only. However here we have a much smaller dataset. Thus, its okay to go with k = 9. So we we will use k = 9 and smaller model with the features selected from feauture selection methods.

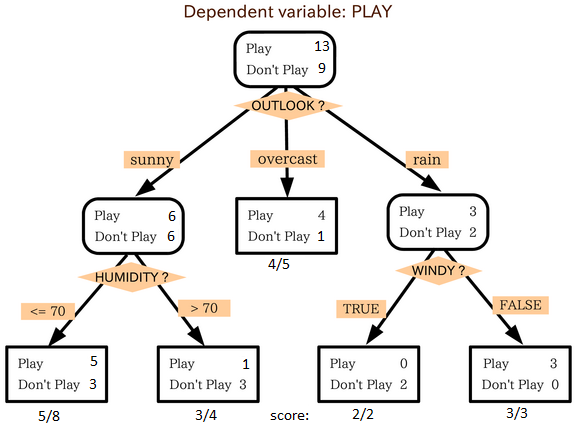
test <- test\_after\_fs  
set.seed(100)  
test$mask <- sample(c(0, 1), 12, replace = TRUE)  
# Predicting the test set on the model  
knn\_pred <- knn(train = train\_after\_fs, test = test, cl = train\_after\_fs$mask, k = 9)  
knn\_pred

## [1] 0 1 0 1 1 0 0 1 0 1 0 1  
## Levels: 0 1

Thus, the above we have the prediction for the test set using K nearesh neighbour classifier with the smaller model an K value = 9.

#### Decision/partition Tree

After seeing the some simple machine learning classification algorithm, Now let make a step further by looking into Decision Tree algorithm. This is one of the favorite algorithm and I use it quite frequently. It is a type of supervised learning algorithm that is mostly used for classification problems. Surprisingly, it works for both categorical and continuous dependent variables. In this algorithm, we split the population into two or more homogeneous sets. This is done based on most significant attributes/ independent variables to make as distinct groups as possible. Below some example that I found from internet



installNewPackage("rpart")  
library(rpart)  
installNewPackage("rpart.plot")  
library(rpart.plot)

## Warning: package 'rpart.plot' was built under R version 3.3.2

installNewPackage("rattle")  
library(rattle)

## Warning: package 'rattle' was built under R version 3.3.3

## Rattle: A free graphical interface for data mining with R.  
## Version 4.1.0 Copyright (c) 2006-2015 Togaware Pty Ltd.  
## Type 'rattle()' to shake, rattle, and roll your data.

for(data in c("train\_all", "train\_after\_fs")) {  
   
 # Intailize variables  
 total\_acc <- 0  
 total\_error\_rate <- 0  
 total\_AUC <- 0  
   
 # To differentiate full and feature subset model  
 if(data == "train\_all") {  
 msg <- "With the full model"  
 data <- train\_all  
 } else {  
 msg <- "After feature selection"  
 data <- train\_after\_fs  
 }  
   
 # Running the Decision Tree with 10 - fold cross validation  
 # Accuracy, error, AUC are rounded off to two decimal points  
 for(fold in 1:k\_fold) {  
 train\_set <- train\_all[-cv\_index\_list[[fold]], ]   
 test\_set <- train\_all[cv\_index\_list[[fold]], ]   
   
 tree\_fit <- rpart(mask ~ ., method="class", data = train\_set)  
 # fancyRpartPlot(tree\_fit, main = paste("Before Pruning - Fold #", fold, sep = " "))  
   
 # Find cp with minimum Cross validate error from rpart results  
 min\_error\_cp <- tree\_fit$cptable[which.min(tree\_fit$cptable[,"xerror"]), "CP"]  
   
 # pruning the tree with minimum cross validate error  
 ptree\_fit <- prune(tree\_fit, cp = min\_error\_cp)  
 # fancyRpartPlot(ptree\_fit, main = paste("After Pruning - Fold #", fold, sep = " "))  
   
 # Calculating Accuracy. Error, AUC  
 test\_prediction <- predict(ptree\_fit, test\_set, type = "class")  
 accuracy <- round(Accuracy(test\_prediction, test\_set$mask), 2)  
 error\_rate <- 1 - accuracy  
   
 AUC <- round(AUC(test\_prediction, test\_set$mask), 2)  
 total\_acc <- total\_acc + accuracy  
 total\_error\_rate <- total\_error\_rate + error\_rate  
 total\_AUC <- total\_AUC + AUC  
 }  
 cat("==========================================================\n\n")  
 cat(msg, "\n\n")  
 cat("==========================================================\n\n")  
 cat(paste("Overall accuracy is", round(total\_acc / k\_fold, 2), sep = " "), "\n\n")  
 cat(paste("Overall error is", round(total\_error\_rate / k\_fold, 2), sep =" "), "\n\n")  
 cat(paste("Overall area under the curve(AUC) is", round(total\_AUC / k\_fold, 2), sep = " "), "\n")  
 cat("\n==========================================================\n\n")  
}

## ==========================================================  
##   
## With the full model   
##   
## ==========================================================  
##   
## Overall accuracy is 0.78   
##   
## Overall error is 0.22   
##   
## Overall area under the curve(AUC) is 0.78   
##   
## ==========================================================  
##   
## ==========================================================  
##   
## After feature selection   
##   
## ==========================================================  
##   
## Overall accuracy is 0.78   
##   
## Overall error is 0.22   
##   
## Overall area under the curve(AUC) is 0.77   
##   
## ==========================================================

Here we can see that the feature selection model with less number of predictor is better compared to the full model. So here lets use the smaller which got after pruning with lowest cross validation. Lets predict the given test set with the smaller model.

# Training the model again  
tree\_fit <- rpart(mask ~ ., method="class", data = train\_after\_fs)  
# Find cp with minimum Cross validate error from rpart results  
min\_error\_cp <- tree\_fit$cptable[which.min(tree\_fit$cptable[,"xerror"]), "CP"]  
# pruning the tree with minimum cross validate error  
ptree\_fit <- prune(tree\_fit, cp = min\_error\_cp)  
  
# Predicting the test set on the model  
dt\_pred <- predict(ptree\_fit, test\_after\_fs, type = "class")  
dt\_pred

## 1 2 3 4 5 6 7 8 9 10 11 12   
## 1 1 0 1 1 0 0 1 0 1 0 1   
## Levels: 0 1

Thus, the above we have the prediction for the test set using Decision tree classifier with smaller model.

### Random Forest

Random Forest is a trademark term for an ensemble of decision trees. In Random Forest, we've collection of decision trees (so known as "Forest"). To classify a new object based on attributes, each tree gives a classification and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest). here I am using a random forest model with 100 trees.

installNewPackage("randomForest")  
library(randomForest)  
  
for(data in c("train\_all", "train\_after\_fs")) {  
   
 # Intailize variables  
 total\_acc <- 0  
 total\_error\_rate <- 0  
 total\_AUC <- 0  
   
 # To differentiate full and feature subset model  
 if(data == "train\_all") {  
 msg <- "With the full model"  
 data <- train\_all  
 } else {  
 msg <- "After feature selection"  
 data <- train\_after\_fs  
 }  
   
 # Running the Random Forest with 10 - fold cross validation  
 # Accuracy, error, AUC are rounded off to two decimal points  
 for(fold in 1:k\_fold) {  
 train\_set <- data[-cv\_index\_list[[fold]], ]   
 test\_set <- data[cv\_index\_list[[fold]], ]   
   
 # Fitting the Random Forest model  
 rf\_fit <- randomForest(as.factor(mask) ~ ., data = train\_set, importance = TRUE, ntree = 100)  
   
 # Calculating Accuracy. Error, AUC  
 test\_prediction <- predict(nb\_fit, test\_set, type = "class")  
   
 accuracy <- round(Accuracy(test\_prediction, test\_set$mask), 2)  
 error\_rate <- 1 - accuracy  
 AUC <- round(AUC(test\_prediction, test\_set$mask), 2)  
 total\_acc <- total\_acc + accuracy  
 total\_error\_rate <- total\_error\_rate + error\_rate  
 total\_AUC <- total\_AUC + AUC  
 }  
 cat("==========================================================\n\n")  
 cat(msg, "\n\n")  
 cat("==========================================================\n\n")  
 cat(paste("Overall accuracy is", round(total\_acc / k\_fold, 2), sep = " "), "\n\n")  
 cat(paste("Overall error is", round(total\_error\_rate / k\_fold, 2), sep =" "), "\n\n")  
 cat(paste("Overall area under the curve(AUC) is", round(total\_AUC / k\_fold, 2), sep = " "), "\n")  
 cat("\n==========================================================\n\n")  
}

## ==========================================================  
##   
## With the full model   
##   
## ==========================================================  
##   
## Overall accuracy is 0.88   
##   
## Overall error is 0.12   
##   
## Overall area under the curve(AUC) is 0.88   
##   
## ==========================================================  
##   
## ==========================================================  
##   
## After feature selection   
##   
## ==========================================================  
##   
## Overall accuracy is 0.88   
##   
## Overall error is 0.12   
##   
## Overall area under the curve(AUC) is 0.88   
##   
## ==========================================================

So for the random forest model we have something interesting, that is, both the full model and smaller model have same accuracy. Since similar the better, I am predicting the test set with the smaller feature subset model itself.

# Training the model again  
rf\_fit <- randomForest(as.factor(mask) ~ ., data = train\_after\_fs, importance = TRUE, ntree = 100)  
# Predicting the test set on the model  
rf\_pred <- predict(rf\_fit, test\_after\_fs, type = "class")  
rf\_pred

## 1 2 3 4 5 6 7 8 9 10 11 12   
## 0 1 0 1 1 0 0 1 0 1 0 1   
## Levels: 0 1

Thus, the above we have the prediction for the test set using Random Forest classifier with smaller model.

#### Support Vector Machine

Support Vector Machine" (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiate the two classes very well.

# Intializing the cost variables  
cost\_values <- c(0.01, 0.1, 1, 10, 100)  
  
for(data in c("train\_all", "train\_after\_fs")) {  
   
 total\_acc\_list <- c()  
 total\_error\_rate\_list <- c()  
 total\_AUC\_list <- c()  
   
 # To differentiate full and feature subset model  
 if(data == "train\_all") {  
 msg <- "With the full model"  
 data <- train\_all  
 } else {  
 msg <- "After feature selection"  
 data <- train\_after\_fs  
 }  
   
 # Running the Support Vector machine for cost values   
 # 0.01, 0.1, 1, 10, 10, 100 with 10 - fold cross validation  
 # Accuracy, error, AUC are rounded off to two decimal points  
 for(cost in cost\_values) {  
   
 # Intailize variables  
 total\_acc <- 0  
 total\_error\_rate <- 0  
 total\_AUC <- 0  
   
 for(fold in 1:k\_fold) {  
 train\_set <- data[-cv\_index\_list[[fold]], ]   
 test\_set <- data[cv\_index\_list[[fold]], ]   
   
 # Fitting the model  
 svm\_fit <- svm(as.factor(mask) ~ ., data = train\_set, kernel = "radial", cost = cost)  
   
 # Calculating Accuracy. Error, AUC  
 test\_prediction <- predict(svm\_fit, test\_set, type = 'class')  
 accuracy <- round(Accuracy(test\_prediction, test\_set$mask), 2)  
 error\_rate <- 1 - accuracy  
 AUC <- round(AUC(test\_prediction, test\_set$mask), 2)  
 total\_acc <- total\_acc + accuracy  
 total\_error\_rate <- total\_error\_rate + error\_rate  
 total\_AUC <- total\_AUC + AUC  
 }  
 total\_acc\_list <- c(total\_acc\_list, total\_acc / k\_fold)  
 total\_error\_rate\_list <- c(total\_error\_rate\_list, total\_error\_rate / k\_fold)  
 total\_AUC\_list <- c(total\_AUC\_list, total\_AUC / k\_fold)  
 }  
   
 max\_acc\_index <- which.max(total\_acc\_list)  
 best\_cost <- cost\_values[max\_acc\_index]  
 best\_acc <- total\_acc\_list[max\_acc\_index]  
 best\_error\_rate <- total\_error\_rate\_list[max\_acc\_index]  
 best\_AUC <- total\_AUC\_list[max\_acc\_index]  
   
 cat("==========================================================\n\n")  
 cat(msg, "\n\n")  
 cat("==========================================================\n\n")  
 cat(paste("Best model is one with cost as", best\_cost, "\n\n", sep = " "))  
 cat(paste("Overall accuracy is", round(total\_acc / k\_fold, 2), sep = " "), "\n\n")  
 cat(paste("Overall error is", round(total\_error\_rate / k\_fold, 2), sep =" "), "\n\n")  
 cat(paste("Overall area under the curve(AUC) is", round(total\_AUC / k\_fold, 2), sep = " "), "\n")  
 cat("\n==========================================================\n\n")  
}

## ==========================================================  
##   
## With the full model   
##   
## ==========================================================  
##   
## Best model is one with cost as 100   
##   
## Overall accuracy is 0.88   
##   
## Overall error is 0.12   
##   
## Overall area under the curve(AUC) is 0.88   
##   
## ==========================================================  
##   
## ==========================================================  
##   
## After feature selection   
##   
## ==========================================================  
##   
## Best model is one with cost as 0.1   
##   
## Overall accuracy is 0.85   
##   
## Overall error is 0.15   
##   
## Overall area under the curve(AUC) is 0.85   
##   
## ==========================================================

Here we are provided two models. The full model is one with a high cost function of 100 which has a accracy of 0.88. And on the otherhand, it is a small model with cost as 0.1 and providing a overall accuracy od 0,85. When we take factor of generalize the model to test data, I feel the model will do better job.

# Training the model again  
svm\_fit <- svm(as.factor(mask) ~ ., data = train\_after\_fs, kernel = "radial", cost = .1)  
# Predicting the test set on the model  
svm\_pred <- predict(svm\_fit, test\_after\_fs, type = "class")  
svm\_pred

## 1 2 3 4 5 6 7 8 9 10 11 12   
## 0 1 0 1 1 0 0 1 0 1 0 1   
## Levels: 0 1

Thus, the above we have the prediction for the test set using Support vector machine classifier with radial kernel and cost of 0.1 on the smaller model.

#### Neural Network

Deep Learning and Neural Network lies in the heart of products such as self driving cars, image recognition software, recommender systems etc. Evidently, being a powerful algorithm, it is highly adaptive to various data types as well. Neural Networks (NN), also called as Artificial Neural Network is named after its artificial representation of working of a human being's nervous system.

installNewPackage("nnet")  
library(nnet)  
  
# Setting the seed for same intial weights  
set.seed(28)  
  
for(data in c("train\_all", "train\_after\_fs")) {  
   
 # Intailize variables  
 total\_acc <- 0  
 total\_error\_rate <- 0  
 total\_AUC <- 0  
   
 # To differentiate full and feature subset model  
 if(data == "train\_all") {  
 msg <- "With the full model"  
 data <- train\_all  
 } else {  
 msg <- "After feature selection"  
 data <- train\_after\_fs  
 }  
   
 # Running the Neural Network with 10 - fold cross validation  
 # Accuracy, error, AUC are rounded off to two decimal points  
 for(fold in 1:k\_fold) {  
 train\_set <- data[-cv\_index\_list[[fold]], ]   
 test\_set <- data[cv\_index\_list[[fold]], ]   
   
 # Fitting the Neural Network model  
 nn\_fit <- nnet(as.factor(mask) ~ ., data = train\_set, size = 10, decay=5e-4, rang = .1,  
 maxit = 200, trace = FALSE)  
  
 # Calculating Accuracy. Error, AUC  
 test\_prediction <- predict(nn\_fit, test\_set, type = "class")  
 accuracy <- round(Accuracy(test\_prediction, test\_set$mask), 2)  
 error\_rate <- 1 - accuracy  
 AUC <- round(AUC(test\_prediction, test\_set$mask), 2)  
 total\_acc <- total\_acc + accuracy  
 total\_error\_rate <- total\_error\_rate + error\_rate  
 total\_AUC <- total\_AUC + AUC  
 }  
 cat("==========================================================\n\n")  
 cat(msg, "\n\n")  
 cat("==========================================================\n\n")  
 cat(paste("Overall accuracy is", round(total\_acc / k\_fold, 2), sep = " "), "\n\n")  
 cat(paste("Overall error is", round(total\_error\_rate / k\_fold, 2), sep =" "), "\n\n")  
 cat(paste("Overall area under the curve(AUC) is", round(total\_AUC / k\_fold, 2), sep = " "), "\n")  
 cat("\n==========================================================\n\n")  
}

## ==========================================================  
##   
## With the full model   
##   
## ==========================================================  
##   
## Overall accuracy is 0.9   
##   
## Overall error is 0.1   
##   
## Overall area under the curve(AUC) is 0.9   
##   
## ==========================================================  
##   
## ==========================================================  
##   
## After feature selection   
##   
## ==========================================================  
##   
## Overall accuracy is 0.86   
##   
## Overall error is 0.14   
##   
## Overall area under the curve(AUC) is 0.86   
##   
## ==========================================================

Compared to other algorithm I used before, this is something that stands out say we have nearly .4 increase in the accuracy for the full model. But However , it important to consider the fact that the full model consists of nearly 26 features whereas the smaller model has only 8 features. So for the Neural network algorithm also I am going with small model for predicting my test set.

# Training the model again  
nn\_fit <- nnet(as.factor(mask) ~ ., data = train\_after\_fs, size = 10, decay=5e-4, rang = .1,  
 maxit = 200, trace = FALSE)  
# Predicting the test set on the model  
nn\_pred <- predict(nn\_fit, test\_after\_fs, type = "class")  
nn\_pred

## [1] "0" "1" "0" "1" "1" "0" "0" "1" "0" "1" "0" "1"

Thus, the above we have the prediction for the test set using Support vector machine classifier with radial kernel and cost of 0.1 on the smaller model.

**So we have completed building our model for the six classifcation algorithms. Now we have to decide what is going to be the output for the test set.**

#### Comparing Models - Define test set labels

In order to decide the final labels for the test set from the results of the previous algorithm, I am going to decide by taking the average of all the prediction got from the 6 classification models respectively to come up with my final prediction mask for the test set.

# Creating a data frame of all prediction  
final\_pred <- data.frame(Logistic = log\_pred, Naive.Bayes = nb\_pred, Knn = knn\_pred,   
 Decision.Tree = dt\_pred, Random.Forest = rf\_pred, svm = svm\_pred,   
 Nnet = nn\_pred)  
mask <- rowMeans(apply(final\_pred, 2, function(x) as.numeric(x)))  
final\_pred$mask <- ifelse(mask >= 0.5, 1, 0)  
final\_pred

## Logistic Naive.Bayes Knn Decision.Tree Random.Forest svm Nnet mask  
## 1 0 1 0 1 0 0 0 0  
## 2 1 1 1 1 1 1 1 1  
## 3 0 0 0 0 0 0 0 0  
## 4 1 1 1 1 1 1 1 1  
## 5 1 1 1 1 1 1 1 1  
## 6 0 0 0 0 0 0 0 0  
## 7 0 0 0 0 0 0 0 0  
## 8 1 1 1 1 1 1 1 1  
## 9 0 0 0 0 0 0 0 0  
## 10 1 1 1 1 1 1 1 1  
## 11 0 0 0 0 0 0 0 0  
## 12 1 1 1 1 1 1 1 1

test\_all$mask <- final\_pred$mask  
test\_all

## A B C D E F G H I J K L M N O P Q R  
## 1 681 713 595 634 758 673 501 676 858 617 713 580 594 694 755 612 693 651  
## 2 693 491 531 839 540 522 510 553 526 693 571 463 482 618 611 479 710 590  
## 3 393 417 395 377 417 401 400 515 541 420 381 374 374 495 415 373 424 414  
## 4 652 530 675 631 562 593 622 573 554 622 550 531 553 562 618 583 692 550  
## 5 418 402 378 441 407 444 427 375 399 517 434 397 389 398 469 487 471 417  
## 6 397 437 453 397 451 441 441 581 572 433 480 538 541 518 396 416 441 420  
## 7 504 487 461 449 489 486 488 590 790 462 505 460 490 487 481 461 505 524  
## 8 437 493 574 514 492 437 609 574 427 472 538 491 431 509 615 512 589 511  
## 9 448 554 516 498 617 561 555 676 533 516 597 534 498 635 414 514 554 496  
## 10 489 498 498 487 484 473 551 468 459 478 471 439 476 472 509 473 518 435  
## 11 473 453 420 456 440 535 432 594 574 453 474 417 513 433 417 473 451 453  
## 12 413 414 398 440 471 450 444 450 421 450 450 411 429 450 493 443 490 455  
## S T U V W X Y Z mask  
## 1 697 713 656 638 672 678 814 538 0  
## 2 664 441 612 514 550 578 428 579 1  
## 3 411 395 392 416 395 755 440 397 0  
## 4 538 599 689 572 589 575 536 631 1  
## 5 381 407 369 374 421 430 378 389 1  
## 6 453 418 415 475 454 513 512 434 0  
## 7 526 541 484 490 567 627 729 509 0  
## 8 559 479 514 517 497 570 512 529 1  
## 9 481 576 532 437 555 672 597 531 0  
## 10 470 462 470 449 431 478 449 555 1  
## 11 421 573 417 417 517 512 456 482 0  
## 12 423 414 431 409 432 453 417 473 1

Thus, that is final prediction label for the given test set.

\*\* End of the assignment \*\*