Assignment 3

Meghashree Madhava Rao Ramachandrahosur (**21200301**)

Building a supervised learning model

In the below code, we use hate speech data set which contains 41 variables and 10703 observations. Seed is set to reproduce the same train, test, validation data sets and output. The first column of the data set contains the sentence using which we will predict if it is a hate speech or not. There are 39 numerical features which have been extracted from the text, which are related to frequencies of certain characters, numerical aspects of the sentence, coordinates in a latent embedding representation of the text, and sentiment scores. We use 'Class' column as a target variable which has hate or no_hate factor levels. The rest of the columns are used as predictor variables.

```
# Set path for data and load data set
setwd("C:/Users/DELL/OneDrive/Documents/SEM-2/ML/asss3")
hate <- read.csv("data_hate_speech.csv")

# to obtain same splits
set.seed(21200301)

#Load the required libraries
library(rpart)
library(partykit)
library(rpart)
library(glmnet)
library(rpart.plot)
library(randomForest)
library(kernlab)
library(adabag)
library(ROCR)

# Remove first column
hate = subset(hate, select = -c(text) )

hate$class <- as.factor(hate$class)
#dat$class <- ifelse(dat$class == "hate", 1,0)
```

The 3 supervised classification methods chosen are *Logistic Regression*, *Classification Tree* and *Random Forest*.

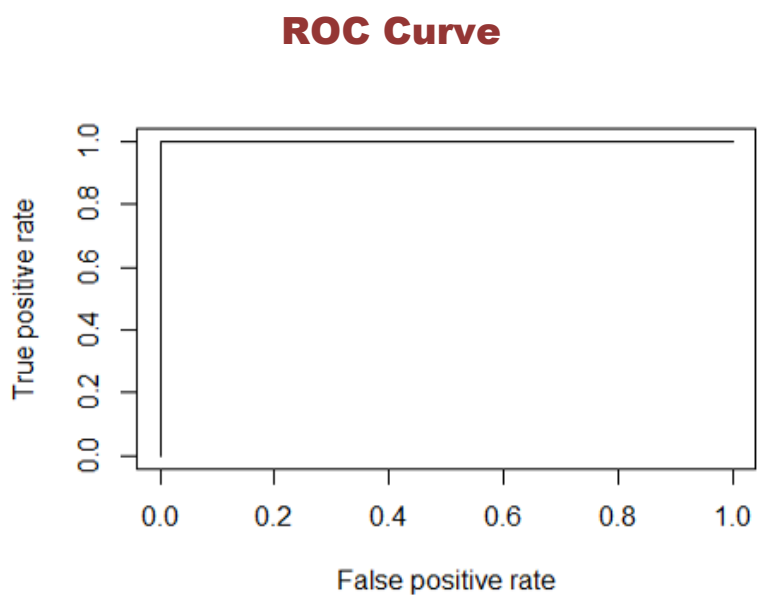
Logistic Regression

```
# to obtain same splits
set.seed(21200301)

# split into training and test
N <- nrow(hate)
```

```
train <- sample(1:nrow(hate), N*0.8)
test <- setdiff(1:N, train)

fit <- glm(class ~ ., data = hate[train,], family = "binomial")
summary(fit)

##
## Call:
## glm(formula = class ~ ., family = "binomial", data = hate[train,
##     ])
##
## Deviance Residuals:
##        Min          1Q      Median          3Q         Max
## -1.032e-04   2.100e-08   2.100e-08   2.100e-08   1.409e-04
##
## Coefficients: (1 not defined because of singularities)
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -5.014e+01  1.460e+04  -0.003    0.997
## n_urls         -2.322e+00  7.911e+03   0.000    1.000
## n_chars         1.928e-01  7.071e+02   0.000    1.000
## n_uq_chars     -1.250e+00  5.650e+02  -0.002    0.998
## n_commas        5.393e-01  3.649e+03   0.000    1.000
## n_digits        5.355e-01  1.337e+03   0.000    1.000
## n_exclaims      2.769e-01  6.254e+03   0.000    1.000
## n_extraspaces   6.540e+00  2.043e+04   0.000    1.000
## n_lowers       -6.036e-02  7.336e+02   0.000    1.000
## n_periods      -2.343e-01  8.593e+02   0.000    1.000
## n_words        -2.945e-01  1.152e+03   0.000    1.000
## n_uq_words     -3.802e-02  1.207e+03   0.000    1.000
## n_caps         -4.720e-02  9.479e+02   0.000    1.000
## n_charsperword  7.432e-01  2.674e+03   0.000    1.000
## n_prepositions -1.628e+00  5.349e+03   0.000    1.000
## w1              1.700e+01  7.356e+04   0.000    1.000
## w2              4.980e+01  5.095e+04   0.001    0.999
## w3              5.515e+01  7.017e+04   0.001    0.999
## w4              4.764e+01  5.086e+04   0.001    0.999
## w5              4.550e+01  7.201e+04   0.001    0.999
## w6              2.645e+01  7.464e+04   0.000    1.000
## w7              3.944e+01  7.843e+04   0.001    1.000
## w8              2.608e+01  6.965e+04   0.000    1.000
## w9              6.380e+01  5.566e+04   0.001    0.999
## w10             3.314e+01  6.965e+04   0.000    1.000
## w11             6.166e+01  6.897e+04   0.001    0.999
## w12             5.564e+01  6.340e+04   0.001    0.999
## w13             3.328e+01  8.213e+04   0.000    1.000
## w14             3.773e+01  7.221e+04   0.001    1.000
## w15             4.840e+01  7.642e+04   0.001    0.999
## w16             3.123e+01  6.430e+04   0.000    1.000
## w17             2.033e+01  7.417e+04   0.000    1.000
## w18             5.094e+01  7.635e+04   0.001    0.999
```

```
## w19              2.918e+01  6.375e+04   0.000    1.000
## w20              5.052e+01  5.405e+04   0.001    0.999
## sent_syuzhet     1.454e-01  9.099e+03   0.000    1.000
## sent_bing        5.120e-01  7.594e+03   0.000    1.000
## sent_afinn       3.325e-02  2.780e+03   0.000    1.000
## sent_nrc        -5.323e-01  3.673e+03   0.000    1.000
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 5.9177e+03  on 8561  degrees of freedom
## Residual deviance: 2.0399e-07  on 8523  degrees of freedom
## AIC: 78
##
## Number of Fisher Scoring iterations: 25
```

```
fitted.results <- predict(fit, newdata=hate[test,], type='response')
pr <- prediction(fitted.results, hate$class[test])

# Plot of ROC curve
prf <- performance(pr, measure = "tpr", x.measure = "fpr")
plot(prf)
```




```
# Prediction Accuracy
auc <- performance(pr, measure = "auc")
auc <- auc@y.values[[1]]
auc
```

```
## [1] 1
```

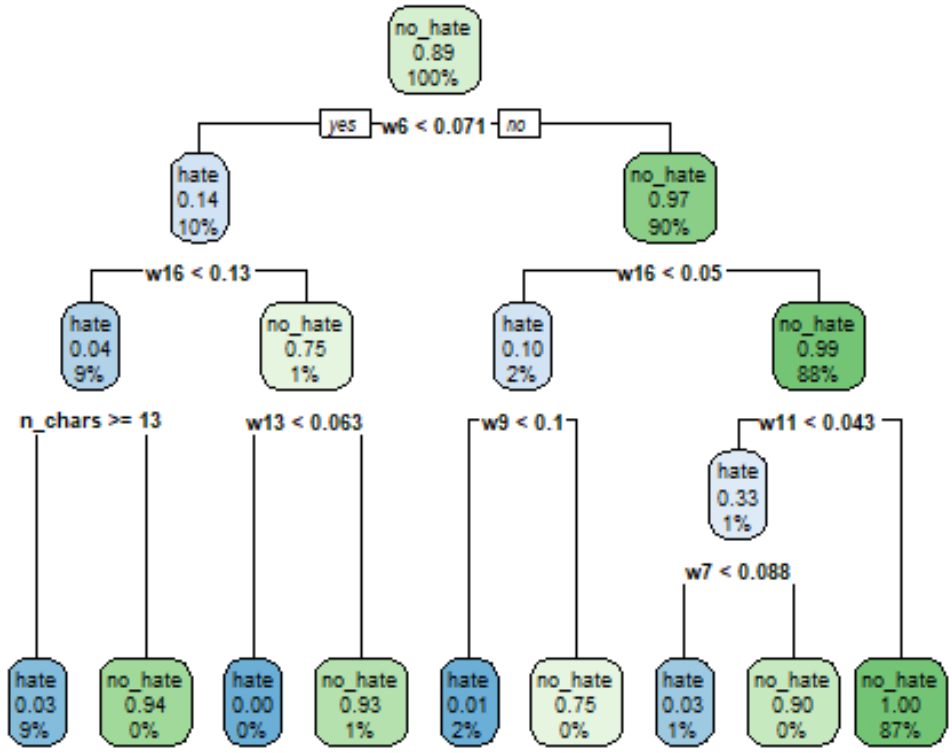
The data is split into two sets, one for training and one for testing, and fitted using Logistic Regression. In the z-scores-based model, none of the variables appear to be significant. A low AIC implies a poor fit between the model and the data. However, the model's forecast for the test data

set appears to be 100 percent accurate, indicating a good predictive model. The ROC curve shown above also supports the prediction showing a perfect fit.

When there is near-multicollinearity, this can easily happen. We have a good model, but none of the variables adds anything considering the contribution of all the other variables. It implies that at least one variable could be omitted without causing your model to fail.

Classification Trees

```
# Fit the model
ct <- rpart(class ~ ., data = hate) # Accuracy 0.985
rpart.plot(ct, extra = 106)
```




```
# Fit the model with hyperparameters for better prediction
ct <- rpart(class ~ ., data = hate, cp = 0.01/10, minsplit = 4)

# Prediction
phat <- predict(ct)
class <- predict(ct, type = "class")
table(hate$class, class)

##          class
##           hate no_hate
```

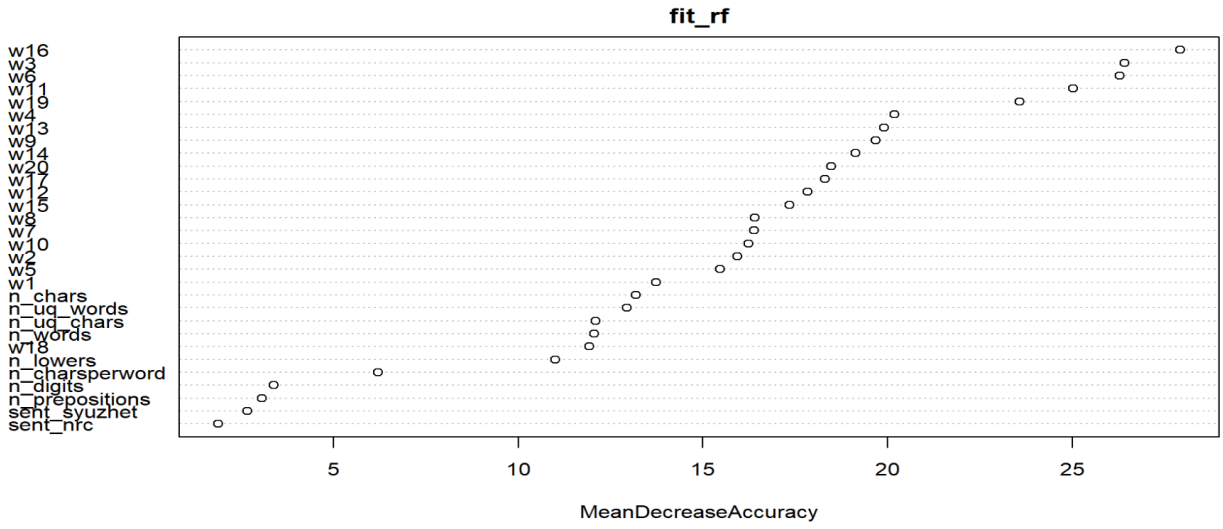
```
##   hate    1190       6
##   no_hate    5    9502

pred_obj <- prediction(phat[,2], hate$class)
auc <- performance(pred_obj, "auc")
auc@y.values

## [[1]]
## [1] 0.9977921
```

The next method used to try to fit the data is a decision tree. When the complexity parameter 'cp' is altered and tweaked, it delivers an approximate accuracy of 99.78 percent. The fit with default parameters has a 98.5 percent accuracy.

The visualization of the tree aids in recognizing the significant variables in the data set. *'w6'* and *'w16'* are the two most important variables used for classification followed by *'n_class', 'w13', 'w9'* and *'w11'.*

Random Forest

```
# function to compute classification accuracy
class_acc <- function(y, yhat) {
tab <- table(y, yhat)
return( sum(diag(tab))/sum(tab) )
}

# implement the random forest algorithm
fit_rf <- randomForest(class ~ ., data = hate, subset = train, importance = T
RUE)

# examine the results
fit_rf

##
## Call:
##  randomForest(formula = class ~ ., data = hate, importance = TRUE,      su
bset = train)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 6
##
##         OOB estimate of  error rate: 0%
## Confusion matrix:
##         hate no_hate class.error
## hate     938       0           0
## no_hate    0    7624           0

# look at variable importance
varImpPlot(fit_rf, type = 1)
```




```
# check predictions
pred_rf <- predict(fit_rf, type = "class", newdata = hate[test,])
table(hate$class[test], pred_rf)

##          pred_rf
##           hate no_hate
##   hate     258       0
##   no_hate    0    1883

class_acc(hate$class[test], pred_rf) # Perfect Fit

## [1] 1
```

The dataset is now fit using random forest method. We obtain a perfect fit with prediction accuracy of 1 to a 80 – 20 split data. This can also be seen from confusion matrix produced by the output of random forest with 0 class error.

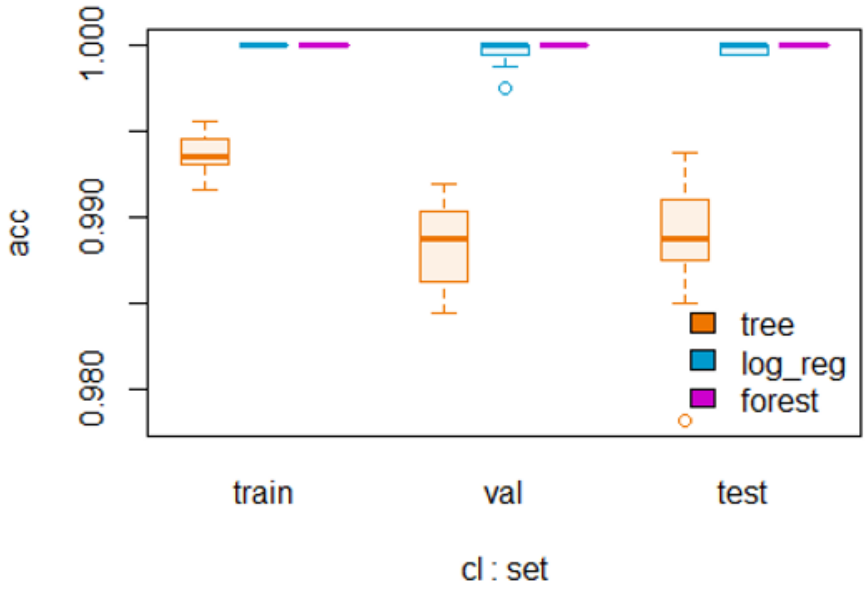
Cross-validation of the 3 models using train, test and validation data set

For evaluation of the 3 classifiers, we use K – fold cross validation method. The data is split into train, test and validation data set to obtain accuracy of the model at each stage.

```
# to obtain same splits
set.seed(21200301)

# set number of replicates and containers
R <- 20
acc_test <- acc_val <- acc_train <- matrix(NA, R, 3)
for ( r in 1:R ) {
  # a 70-15-15 split
  train <- sample(1:N, 0.7*N)
  val <- sample(setdiff(1:N, train), (N - 0.7*N)*0.5)
```

```r
test <- setdiff(1:N, c(train, val))

# fit on training data

# classification tree
fit_ct <- rpart(class ~ ., data = hate, subset = train)

# logistic regression
fit_lr <- glm(class ~ ., data = hate, subset = train, family = "binomial")

# random forest
fit_rf <- randomForest(class ~ ., data = hate, subset = train)

# predict and assess on training data

pred_ct <- predict(fit_ct, newdata = hate[train,], type = "class")
acc_train[r,1] <- class_acc(hate$class[train], pred_ct)

pred_lr <- predict(fit_lr, newdata = hate[train,], type = "response")
pred_lr <- ifelse(pred_lr > 0.5, 1, 0)
acc_train[r,2] <- class_acc(hate$class[train], pred_lr)

pred_rf <- predict(fit_rf, newdata = hate[train,], type = "class")
acc_train[r,3] <- class_acc(hate$class[train], pred_rf)

# predict and assess on validation data

pred_ct <- predict(fit_ct, newdata = hate[val,], type = "class")
acc_val[r,1] <- class_acc(hate$class[val], pred_ct)

pred_lr <- predict(fit_lr, newdata = hate[val,], type = "response")
pred_lr <- ifelse(pred_lr > 0.5, 1, 0)
acc_val[r,2] <- class_acc(hate$class[val], pred_lr)

pred_rf <- predict(fit_rf, newdata = hate[val,], type = "class")
acc_val[r,3] <- class_acc(hate$class[val], pred_rf)

# predict and assess on test data

pred_ct <- predict(fit_ct, newdata = hate[test,], type = "class")
acc_test[r,1] <- class_acc(hate$class[test], pred_ct)

pred_lr <- predict(fit_lr, newdata = hate[test,], type = "response")
pred_lr <- ifelse(pred_lr > 0.5, 1, 0)
acc_test[r,2] <- class_acc(hate$class[test], pred_lr)

pred_rf <- predict(fit_rf, newdata = hate[test,], type = "class")
acc_test[r,3] <- class_acc(hate$class[test], pred_rf)
```

```
}
# accuracy values in a single data frame for visualization
acc <- c( rbind(acc_train, acc_val, acc_test))
cl <- factor(rep(1:3, each = R*3), labels = c("tree", "log_reg", "forest"))
set <- rep(factor(rep(1:3, each = R), labels = c("train", "val", "test")), 3)
acc <- data.frame(acc = acc, cl = cl, set = set)

# colors denote classifiers
cols <- c("darkorange2", "deepskyblue3", "magenta3")

# boxplot to show all results in a single panel
boxplot(acc ~ cl + set, data = acc, border = cols, col = adjustcolor(cols, 0.
1), at = c(1:3, 5:7, 9:11), xaxt = "n")
mtext(side = 1, line = 1, at = c(2, 6, 10), text = levels(set))
legend("bottomright", legend = levels(cl), bty = "n", fill = cols)
```



We can see from the above graph that Logistic regression and Random Forest have very good and consistent accuracy for all 3 data sets. Logistic regression has variable accuracy and hence performing differently for test and validation sets. We can say that the model is overfit , hence concluding that Random Forest produces the best fit for the hate speech dataset.

Merits of each classification methods:

Logistic regression is a simple and more efficient method for binary and linear classification problems. It is a classification model, which is very easy to realize and achieves very good performance with linearly separable classes. It is an extensively employed algorithm for classification in industry.

Similarly, there are several advantages and disadvantages to Classification trees. Classification trees are straightforward to comprehend and interpret. It is easier to discern important variables when trees are visualized. It works well with missing data and does not require data

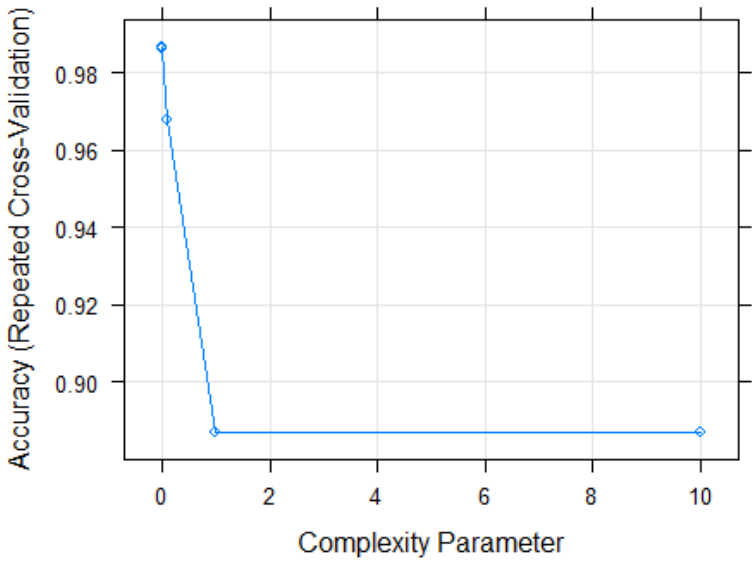
standardization. It is capable of dealing with both numerical and categorical data. On the other side, using decision trees, it is very easy to overfit the data. If some classes dominate, decision tree learners build biased trees. It is consequently advised that the dataset be balanced before using the decision tree to fit it.

When we have a huge dataset and interpretability isn't a key concern, Random Forest is a good choice. Using several trees, the technique avoids and inhibits overfitting. This produces precise and accurate results. Because decision trees need little processing, they are quick to implement and have poor accuracy.

Model Tuning

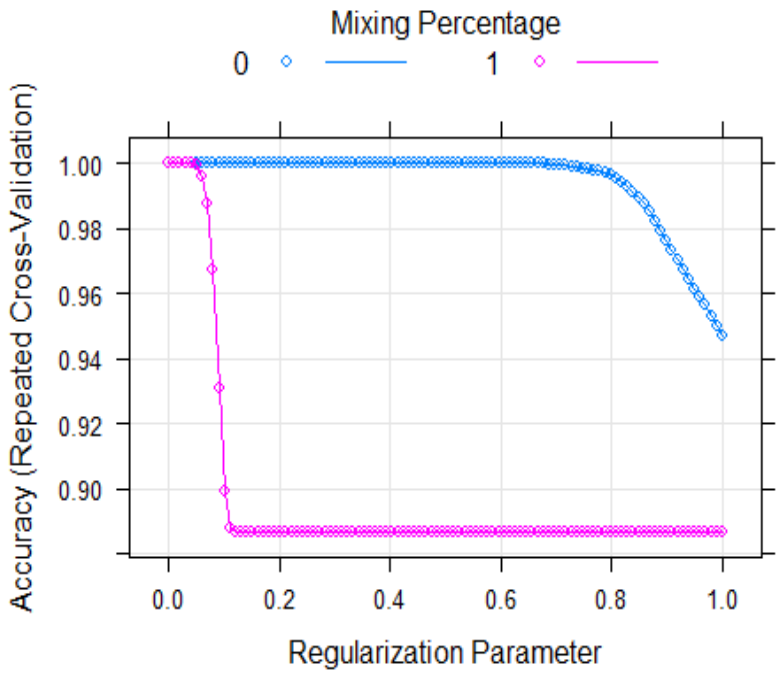
Hyperparameter optimization is another term for model tuning. The training process is controlled by hyperparameters, which are variables. During a model training job, these are configuration variables that do not change. Model tweaking gives ideal settings for hyperparameters, increasing the predicted accuracy of your model.

```
library(doParallel)
cl <- makeCluster(6)
registerDoParallel(cl) # start parallel computing back-end

# just to implement two-fold, i.e. held out sample cross validation
train_ctrl <- trainControl(method = "repeatedcv", number = 2, repeats = 10)

# Classification Tree

tune_grid <- expand.grid(cp = c(10, 1.0, 0.1, 0.01, 0.001))
fit_ct <- train(class ~ ., data = hate[train,], method = "rpart", trControl =
train_ctrl, tuneGrid = tune_grid)
plot(fit_ct)
```

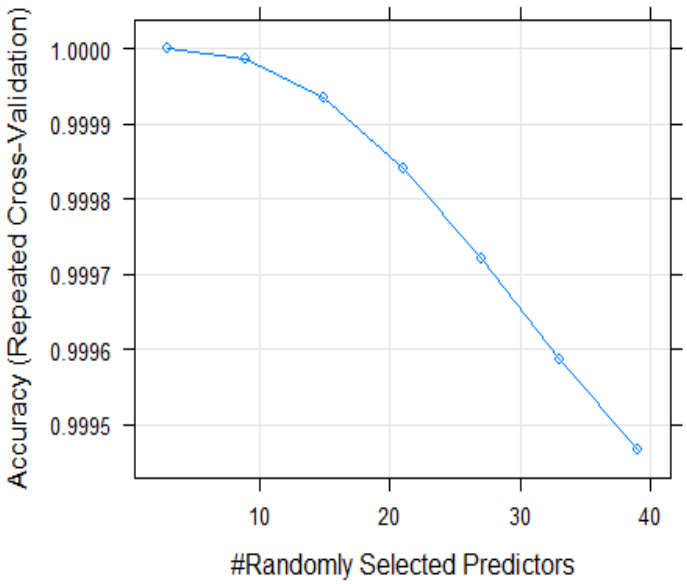


Classification Tree performs best at complex parameter ***cp = 0.01***

```
# Logistic Regression

tune_grid <- expand.grid(alpha = 0:1, lambda = seq(0.0001, 1, length = 100))
fit_lr <- train(class ~ ., data = hate[train,], method = "glmnet", trControl
= train_ctrl, tuneGrid = tune_grid)

plot(fit_lr)
```




```
# Random Forest
# note that we have 39 predictors
tune_grid <- expand.grid( mtry = c(3,9,15,21,27,33,39))

fit_rf <- train(class ~ ., data = hate[train,], method = "rf", trControl = tr
ain_ctrl, tuneGrid = tune_grid)

plot(fit_rf)
```



Random forest method fits the data best at its hyperparameter at 1

```
stopCluster(cl)

comp <- resamples(list(ct = fit_ct, lr = fit_lr, rf = fit_rf))
summary(comp)

##
## Call:
## summary.resamples(object = comp)
##
## Models: ct, lr, rf
## Number of resamples: 20
##
## Accuracy
##         Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## ct 0.9829151 0.9851842 0.9870529 0.9866391 0.9883209 0.9895889    0
## lr 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000    0
## rf 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000    0
##
## Kappa
##         Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## ct 0.9148956 0.9263479 0.9348712 0.9331876 0.9423144 0.9478712    0
## lr 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000    0
## rf 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000    0
```

We can see accuracy of both Logistic Regression and Random Forest to be the highest of 1. But due to spread out prediction accuracy for test and validation data set for Logistic Regression, ***we consider Random Forest to be the best model fit for the data set considered.***

Prediction with best model

```
# extract estimated class labels
class_hat <- predict(fit_rf, newdata = hate[test,])

# compute metrics
confusionMatrix(class_hat, hate$class[test])

## Confusion Matrix and Statistics
##
##           Reference
## Prediction hate no_hate
##    hate     173       0
##    no_hate    0    1433
##
##                Accuracy : 1
##                  95% CI : (0.9977, 1)
##     No Information Rate : 0.8923
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.1077
##          Detection Rate : 0.1077
##    Detection Prevalence : 0.1077
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : hate
##
```