CS109B - Final Project

Group 26 - Nina Iftikhar

The glmnet package was used to construct logistic regression models for classification of movie genres represented in binary 0-1 values. The binary representation aided in dealing with the multi-label and imbalanced data issues within this data set; data was acquired from TBDM, with missing data supplemented from IMDB where appropriate. We became interested in using overview words for genre classification because of findings during our EDA which showed that this approach may be promising.

The cv.glm function was used because it allowed for selecting the optimal lambda value via built-in tuning and cross-validation, but the function also has settings that allow control over the model, like setting the tresh value; this defines the convergence threshold for coordinate descent, and is important in determining the limit of coefficient update for the algorithm.

Separate logistical regression models were constructed for each genre- based on the misclassification plots, different values of lambda are needed for higher accuracy by genre. Lambda, the regularization parameter, is important for this algorithm, and for other regression models, because it prevents overfitting on the training set by modifying the optimization problem to prefer small weights (source: https://justindomke.wordpress.com/2008/12/12/why-does-regularization-work/ (https://justindomke.wordpress.com/2008/12/12/why-does-regularization-work/)). Hence considerations for bias-variance trade off are built into the model.

Glmnet fits a generalized linear model via penalized maximum likelihood- a grid of values of lambda is used to compute the regularization path for the lasso or the elasticnet penalty (Glmnet Vignette). Therefore, the lambda and alpha parameters were tuned because they are central to this algorithm (see alpha tune chunk), and thus their values are expected to affect the classification accuracy.

Text2Vec was used to construct document-term-matrices from the Overview field in the data set- the Overview field contains descriptions of the plots of the movies. The objective of creating the DTMs is to use them as “bag-of-words” to drive genre classification via logistic regression. The DTMs were also normalized in order to suppress the overrepresentation of any particular words.

The training set for this model approach is a document-term-matrix of a vocabulary vector made from the overview words for each movie in the sampled train set. The vocabulary vector stores all the term counts and doc terms associated with the each keyword from the overview words fields, and the dtm assigns movie ids to the occurence of those words. The test set for this model approach is a dtm of the test set.

The optimal metric of this classification approach would be a confusion matrix that shows more true positives and negatives, than false positives and negatives, or an accuracy bar plot.

```
set.seed(200)
suppressWarnings(suppressMessages(library(text2vec)))
suppressWarnings(suppressMessages(library(data.table)))
suppressWarnings(suppressMessages(library(MLmetrics)))
suppressWarnings(suppressMessages(library(glmnet)))
suppressWarnings(suppressMessages(library(caret)))

#Note: I followed the directions here to construct a document-term matrix for the Overvi
ew field in the data set: http://text2vec.org/vectorization.html

movies <- read.csv("Meta_data_First_All_Cleaned.csv")

#the genre labels matrix attempts to override the imbalanced data and multi-label proble
ms in this data set
g.labels <- read.csv("Genres_labels_All copy.csv")

head(movies, 4)
```

```
##   X  budget          director
## 1 0       0 Richard Fleischer
## 2 1 6000000    Gregory Jacobs
## 3 3       0     David Silberg
## 4 4       0        Herb Freed
##
           genres
## 1                                                                      [{u'id': 80,
 u'name': u'Crime'}]
## 2 [{u'id': 18, u'name': u'Drama'}, {u'id': 27, u'name': u'Horror'}, {u'id': 53, u'nam
e': u'Thriller'}]
## 3                               [{u'id': 35, u'name': u'Comedy'}, {u'id': 10749, u'na
me': u'Romance'}]
## 4                                                                     [{u'id': 27, u'n
ame': u'Horror'}]
##      id
## 1 22924
## 2 14223
## 3 18307
## 4 27420
##                                                                               keyword
s
## 1                                                              ['armored car', 'film noi
r']
## 2 ['terror', 'winter', 'paranoia', 'cold', 'supernatural', 'snow', 'student', 'cras
h']
## 3

## 4                                                                             ['slashe
r']
##




                                                                      overview
## 1


                                              The film tells the story of a well-planned
robbery of an armored car when it stops at a sports stadium. Yet, the heist goes awry, a
nd a tough Los Angeles cop named Cordell (Charles McGraw) is in hot pursuit.
## 2


                                                                              Two coll
ege students share a ride home for the holidays. When they break down on a deserted stre
tch of road, they're preyed upon by the ghosts of people who have died there.
## 3
                                                        Three beautiful women (Electra,
 Dash, and Fox) who have had their share of men trouble enter into a game of fun in whic
h they choose a random guy and film each other seducing him so as to use the footage lat
er to humiliate him. But problems arise when the random man is in on the joke.
## 4 After a high school track runner, named Laura, suddenly dies from a heart attack af
ter finishing a 30-second 200-meter race, a killer wearing a sweat suit and a fencing ma
sk begins killing off her friends on the school track team one by one. The suspects incl
```

```
ude the track coach Michaels, Laura's sister Anne who arrives in town for the funeral, t
he creepy school principal Mr. Guglione, and Laura's strange boyfriend Kevin.
##   popularity                       poster_path releaseyear revenue runtime
## 1   0.122706 /6JdNN04zLhWrcz5rD3k9d4nDbM7.jpg        1950       0      67
## 2   0.249227 /pEVChTdLzgXO1fBvlSYFsukWlP4.jpg        2007       0      91
## 3   0.137653 /9u2f7mbgwJzSIcnOqfS5kOAELlH.jpg        2005       0      81
## 4   0.188098 /bxg98VKp0tUA2U3AUuM2Ej9Yc4S.jpg        1981       0      96
##                 title
## 1 Armored Car Robbery
## 2          Wind Chill
## 3      Getting Played
## 4      Graduation Day
```

```
head(g.labels, 4)
```

```
##   X Action Adventure Animation Comedy Crime Documentary Drama Family
## 1 0      0         0         0      0     1           0     0      0
## 2 1      0         0         0      0     0           0     1      0
## 3 2      0         0         0      1     0           0     0      0
## 4 3      0         0         0      0     0           0     0      0
##   Fantasy History Horror Music Mystery Romance Science.Fiction TV.Movie
## 1       0       0      0     0       0       0               0        0
## 2       0       0      1     0       0       0               0        0
## 3       0       0      0     0       0       1               0        0
## 4       0       0      1     0       0       0               0        0
##   Thriller War Western    ID
## 1        0   0       0 22924
## 2        1   0       0 14223
## 3        0   0       0 18307
## 4        0   0       0 27420
```

```
movies <- cbind(movies, g.labels)

#coerce data table
setDT(movies)

#create key on data table called id
setkey(movies, id)

#set all_id variable to easily refer to ids when splitting
all_ids = movies$id

#set train ids, difference is test set ids
train_ids = sample(all_ids, 7491)
test_ids = setdiff(all_ids, train_ids)

#subset by ids
train = movies[J(train_ids)]
test = movies[J(test_ids)]
```

```
#The objective is to create a document term matrix, this chunk of code creates a vocabul
ary vector.

prep_fun = tolower
tok_fun = word_tokenizer

#itoken() creates an iterator over tokens
it_train = itoken(as.character(train$overview),
             preprocessor = prep_fun,
             tokenizer = tok_fun,
             ids = train$id,
             progressbar = FALSE)

vocab = create_vocabulary(it_train)
head(vocab,10)
```

```
## $vocab
##            terms terms_counts doc_counts
##     1: mikhalych            1          1
##     2:  crowther            1          1
##     3: naysayers            1          1
##     4:   hibbert            1          1
##     5: concocted            1          1
##    ---
## 30445:      rant            1          1
## 30446:    полоцк            1          1
## 30447:    doctor          143        119
## 30448:      cage            9          7
## 30449:     kōchi            1          1
##
## $ngram
## ngram_min ngram_max
##         1         1
##
## $document_count
## [1] 7491
##
## $stopwords
## character(0)
##
## $sep_ngram
## [1] "_"
```

```
#Document term matrix is created from vocab vector
vectorizer = vocab_vectorizer(vocab)
dtm_train = create_dtm(it_train, vectorizer)

#check the dimensions of the DTM to ensure same # of rows as train set, this will be imp
ortant when running the model
dim(dtm_train)
```

```
## [1]  7491 30449
```

```
vocab
```

```
## Number of docs: 7491
## 0 stopwords:  ...
## ngram_min = 1; ngram_max = 1
## Vocabulary:
##             terms terms_counts doc_counts
##      1: mikhalych            1          1
##      2:  crowther            1          1
##      3: naysayers            1          1
##      4:   hibbert            1          1
##      5: concocted            1          1
##    ---
## 30445:      rant             1          1
## 30446:    полоцк             1          1
## 30447:    doctor           143        119
## 30448:      cage             9          7
## 30449:     kōchi             1          1
```

```
#repeat above steps for test set
ts.prep_fun = tolower
ts.tok_fun = word_tokenizer

it_test = itoken(as.character(test$overview),
             preprocessor = ts.prep_fun,
             tokenizer = ts.tok_fun,
             ids = test$id,
             progressbar = FALSE)

ts.vocab = create_vocabulary(it_test)

ts.vocab = create_vocabulary(it_test)
head(ts.vocab,10)
```

```
## $vocab
##              terms terms_counts doc_counts
##      1:    submits            1          1
##      2:       bari            1          1
##      3:   fleecing            1          1
##      4:        bey            1          1
##      5:       juli            1          1
##    ---
## 17377: shopkeeper            1          1
## 17378:    persian            2          2
## 17379:        dom            2          2
## 17380:   peacable            1          1
## 17381:       turn           57         56
##
## $ngram
## ngram_min ngram_max
##         1         1
##
## $document_count
## [1] 2497
##
## $stopwords
## character(0)
##
## $sep_ngram
## [1] "_"
```

```
ts.vectorizer = vocab_vectorizer(ts.vocab)
dtm_test = create_dtm(it_test, ts.vectorizer)
dim(dtm_test)
```

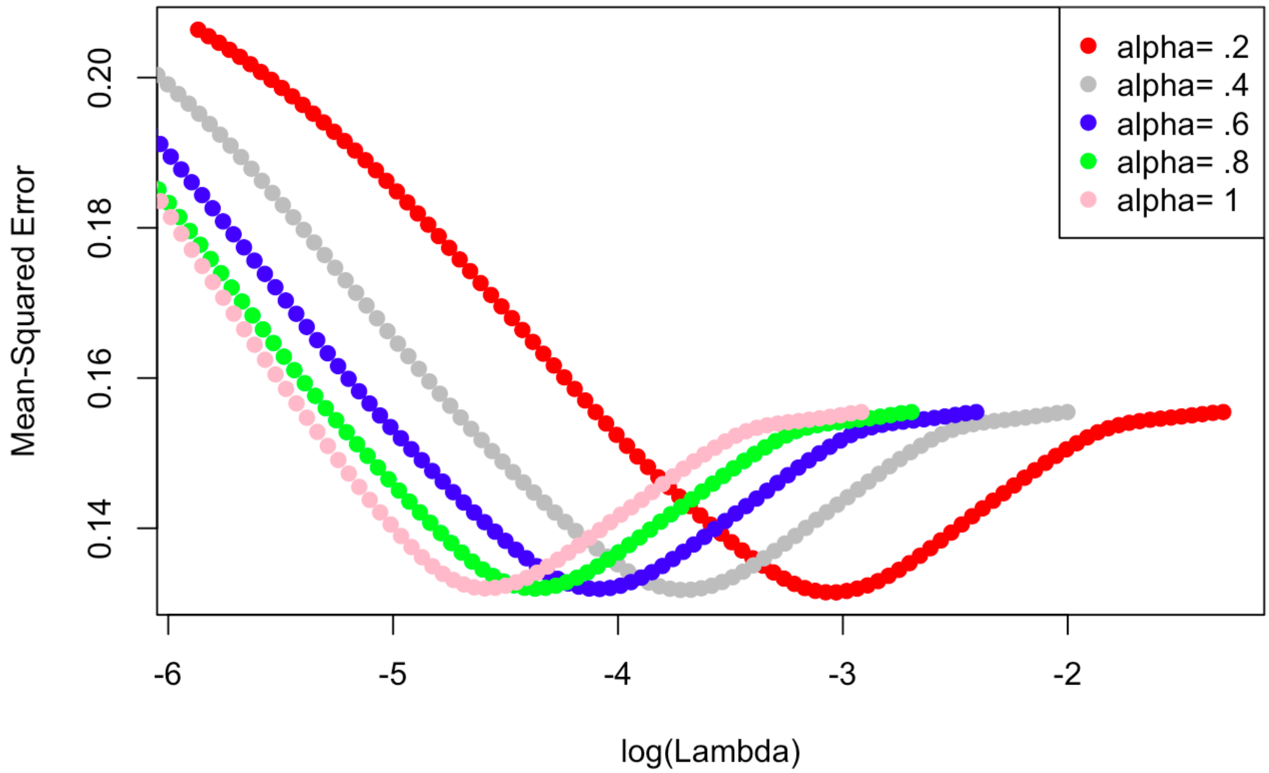
```
## [1]  2497 17381
```

```
#normalize DTMs to remove unequal weight from overrepresentation of certain words
tfidf = TfIdf$new()

dtm_train_tfidf = fit_transform(dtm_train, tfidf)

dtm_test_tfidf  = create_dtm(it_test, vectorizer) %>%
  transform(tfidf)
```

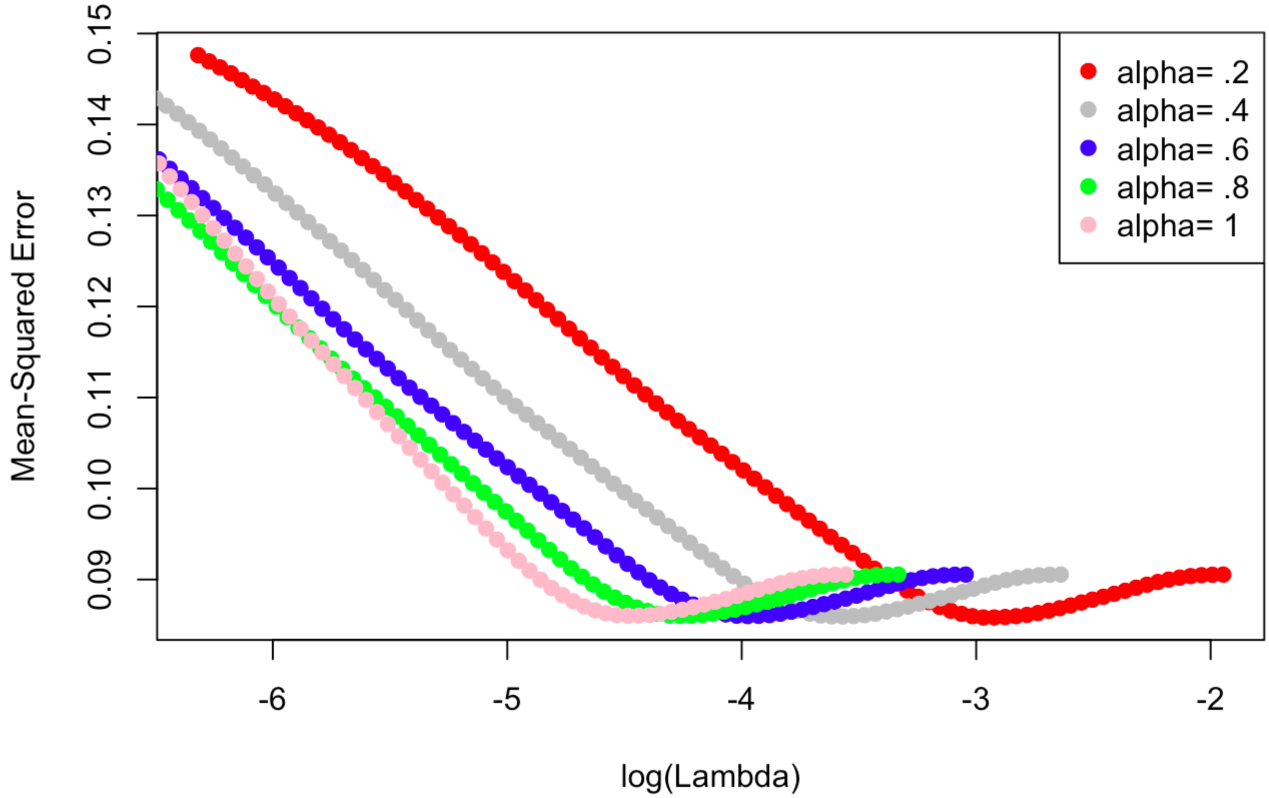
Alpha values were plotted to determine the best alpha for each one of the 19 models; best value of alpha gives the lowest MSE when plotted over lambda. Alpha is the elastic-net mixing parameter, combines lasso and ridge regression, therefore it must be in the range $\alpha \in [0,1]$. For ease of running the plots function over 19 separate models, alpha was stepped by 0.2.

The foldid parameter allows setting up the cross validation folds before running the models for tuning alpha.

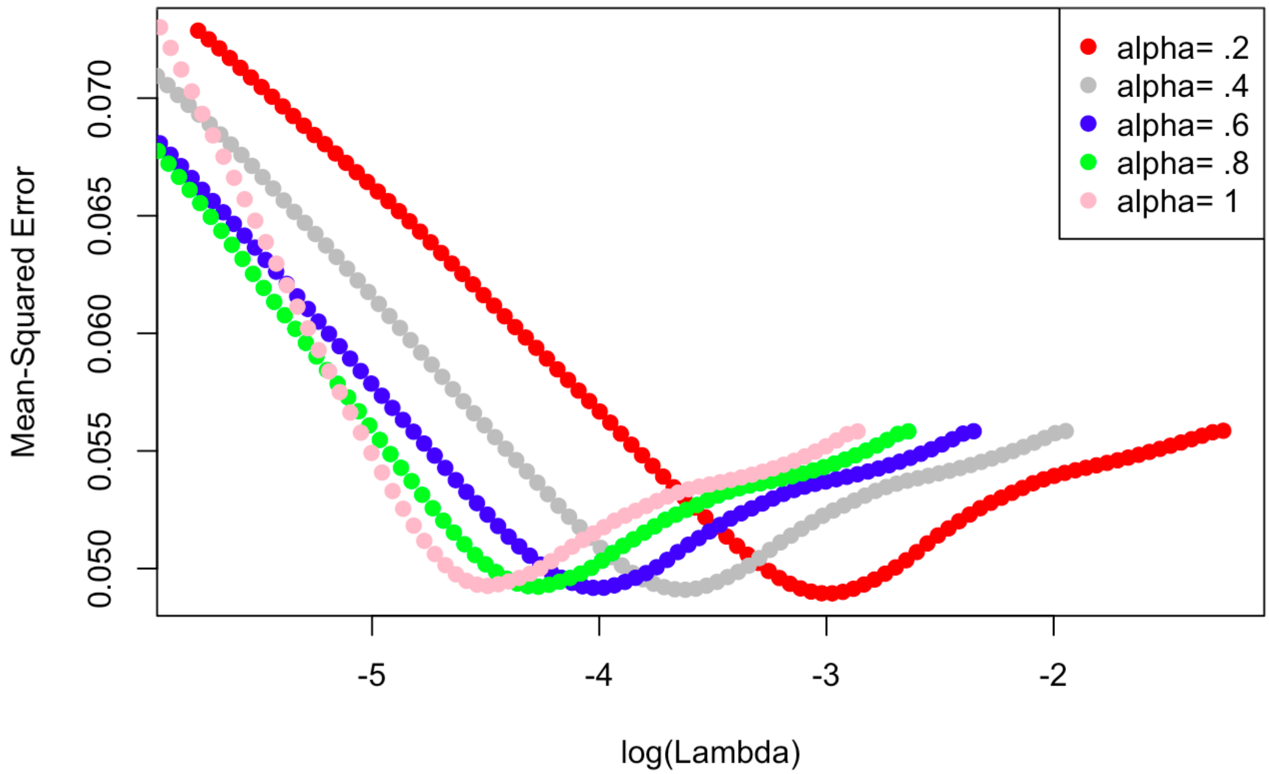
```
x = dtm_train_tfidf

plots <- function(y){

foldid=sample(1:10,size=length(y),replace=TRUE)
cv1=cv.glmnet(x,y,foldid=foldid,alpha=0.2)
cv2=cv.glmnet(x,y,foldid=foldid,alpha=0.4)
cv3=cv.glmnet(x,y,foldid=foldid,alpha=0.6)
cv4=cv.glmnet(x,y,foldid=foldid,alpha=0.8)
cv5=cv.glmnet(x,y,foldid=foldid,alpha=1)

plot(log(cv1$lambda),cv1$cvm,pch=19,col="red",xlab="log(Lambda)",ylab=cv1$name)
points(log(cv2$lambda),cv2$cvm,pch=19,col="grey")
points(log(cv3$lambda),cv3$cvm,pch=19,col="blue")
points(log(cv4$lambda),cv4$cvm,pch=19,col="green")
points(log(cv5$lambda),cv5$cvm,pch=19,col="pink")
legend("topright",legend=c("alpha= .2","alpha= .4", "alpha= .6", "alpha= .8", "alpha=
1"),pch=19,col=c("red","grey", "blue", "green", "pink"))

mse.min1 <- cv1$cvm[cv1$lambda == cv1$lambda.min]
mse.min2 <- cv2$cvm[cv2$lambda == cv2$lambda.min]
mse.min3 <- cv3$cvm[cv3$lambda == cv3$lambda.min]
mse.min4 <- cv4$cvm[cv4$lambda == cv4$lambda.min]
mse.min5 <- cv5$cvm[cv5$lambda == cv5$lambda.min]

   mses <- cbind(mse.min1, mse.min2, mse.min3, mse.min4, mse.min5)
   mses <- as.data.frame(mses)
   return(mses)

}

apply(train[,15:33],2,plots)
```

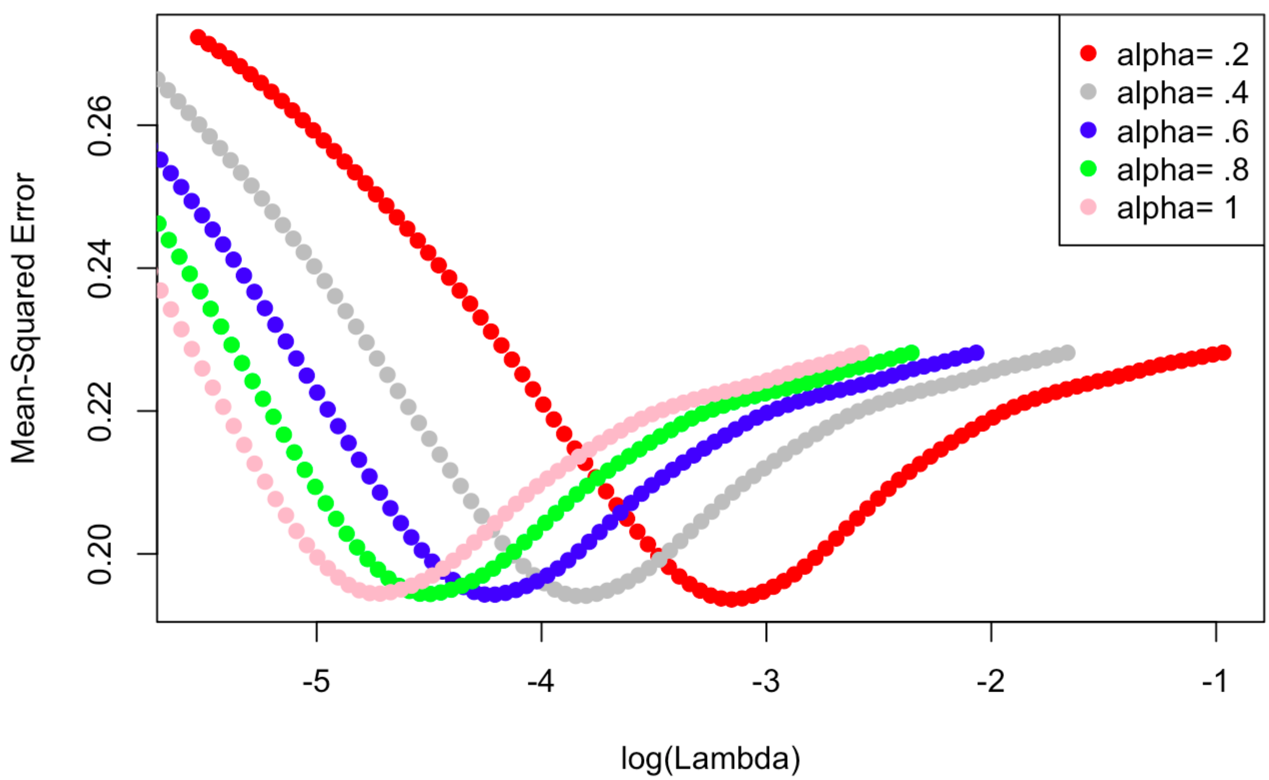


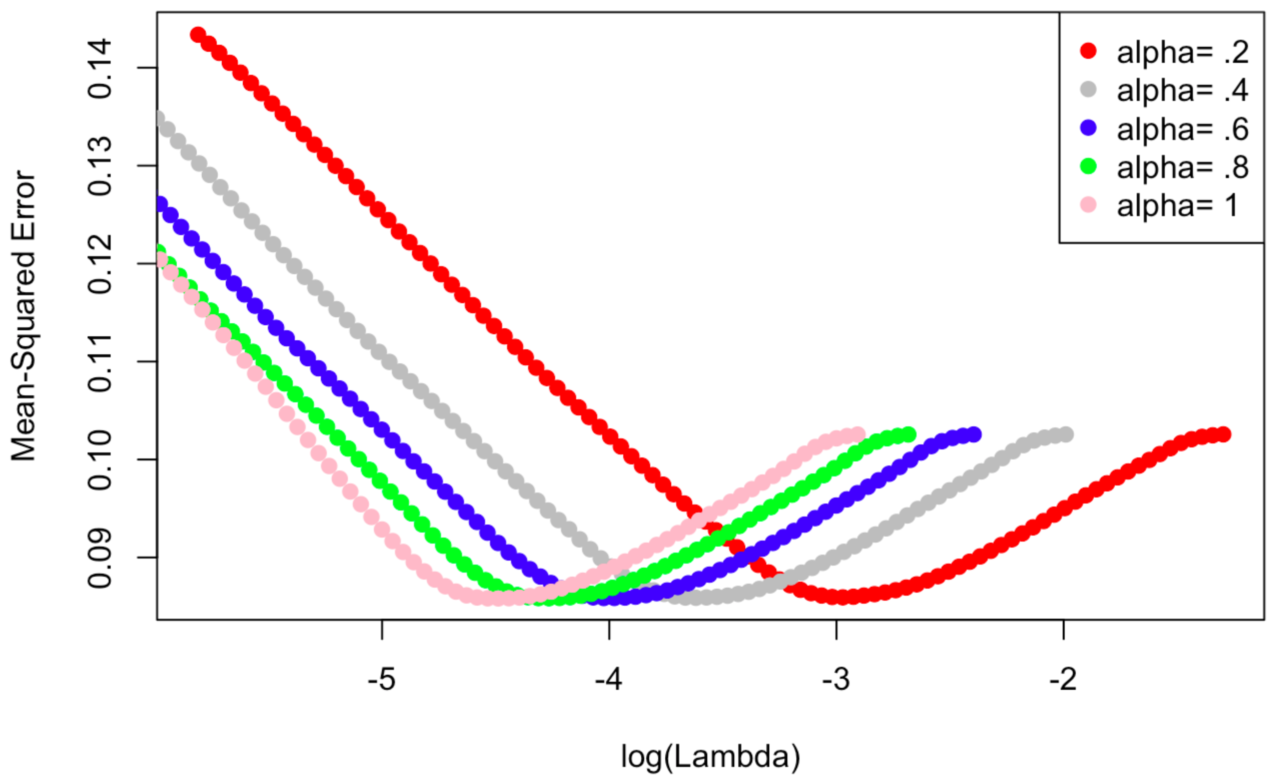
alpha= .2
alpha= .4
alpha= .6
alpha= .8
alpha= 1
Mean-Squared Error
0.14
0.16
0.18
0.20
-6
-5
-4
-3
-2
log(Lambda)



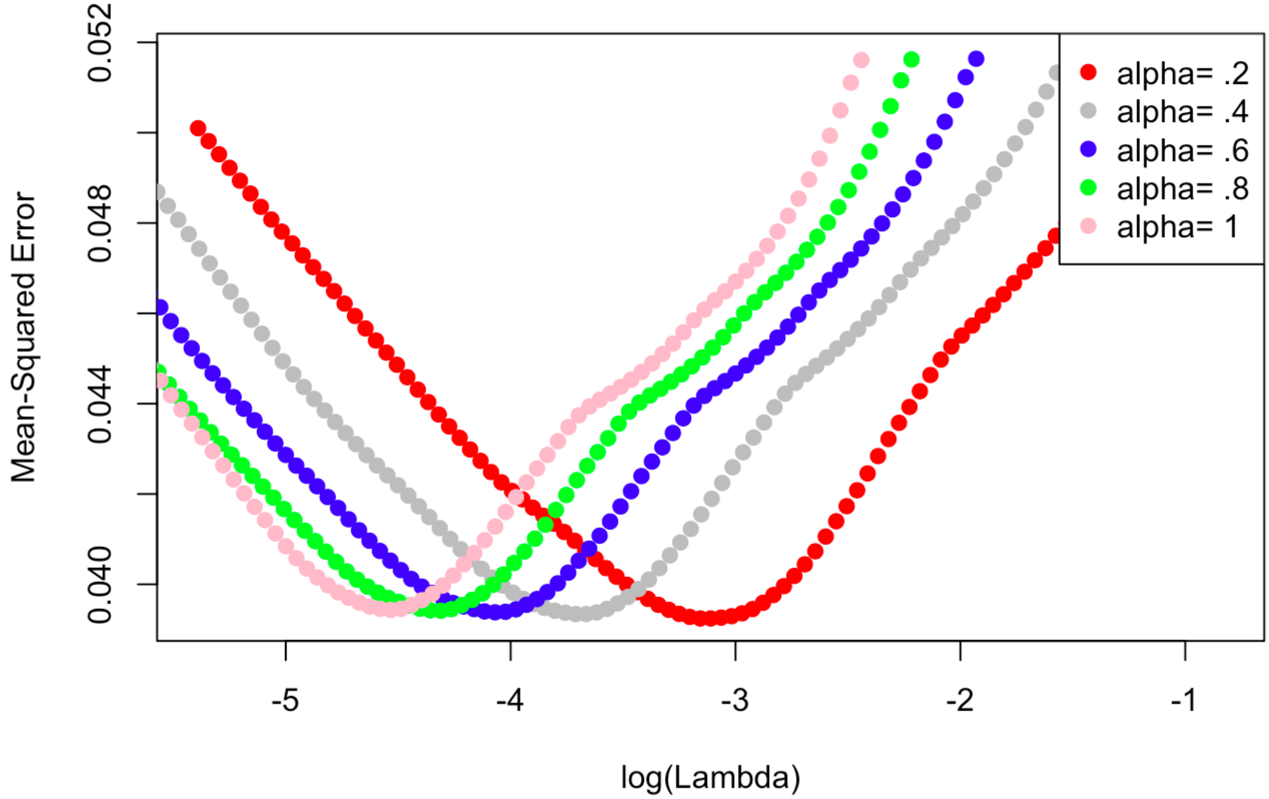
alpha= .2
alpha= .4
alpha= .6
alpha= .8
alpha= 1
Mean-Squared Error
0.09
0.10
0.11
0.12
0.13
0.14
0.15
-6
-5
-4
-3
-2
log(Lambda)


alpha= .2
alpha= .4
alpha= .6
alpha= .8
alpha= 1
Mean-Squared Error
0.050
0.055
0.060
0.065
0.070
-5
-4
-3
-2
log(Lambda)

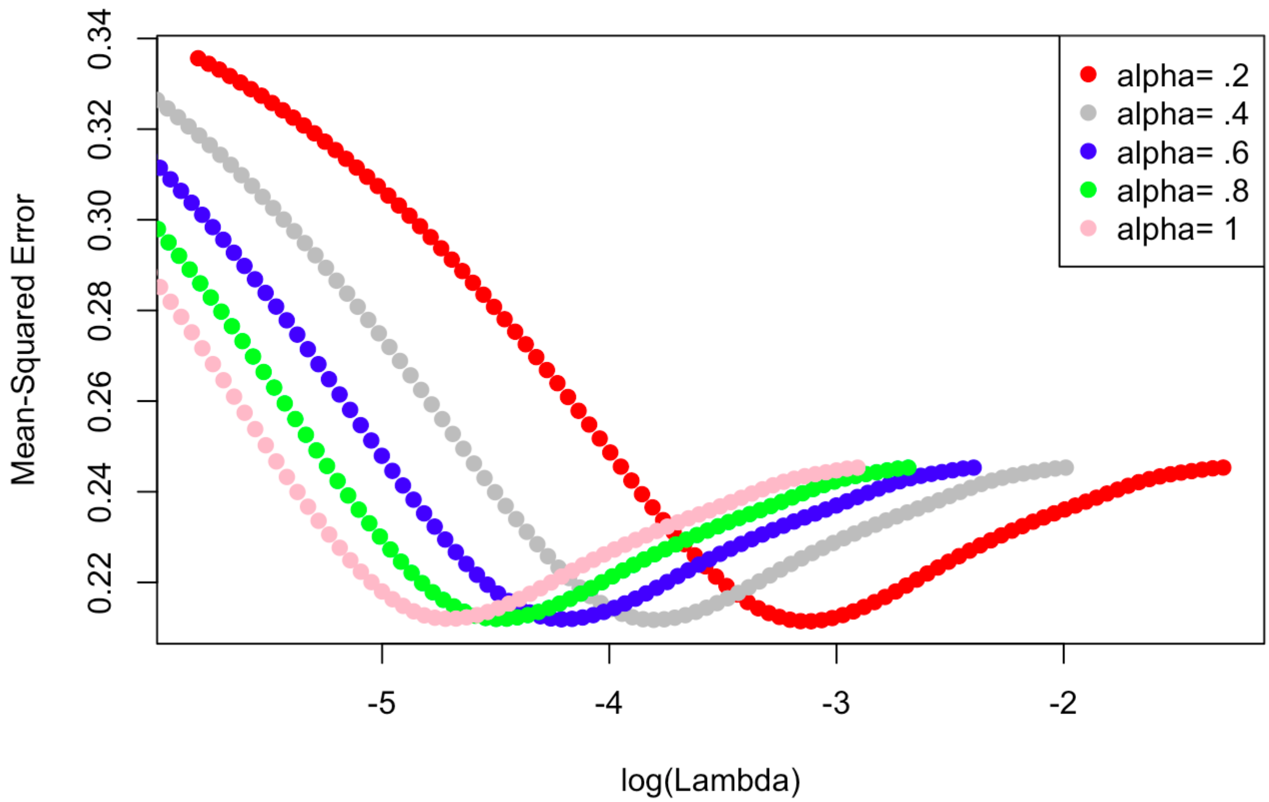

alpha= .2
alpha= .4
alpha= .6
alpha= .8
alpha= 1
Mean-Squared Error
0.20
0.22
0.24
0.26
-5
-4
-3
-2
-1
log(Lambda)

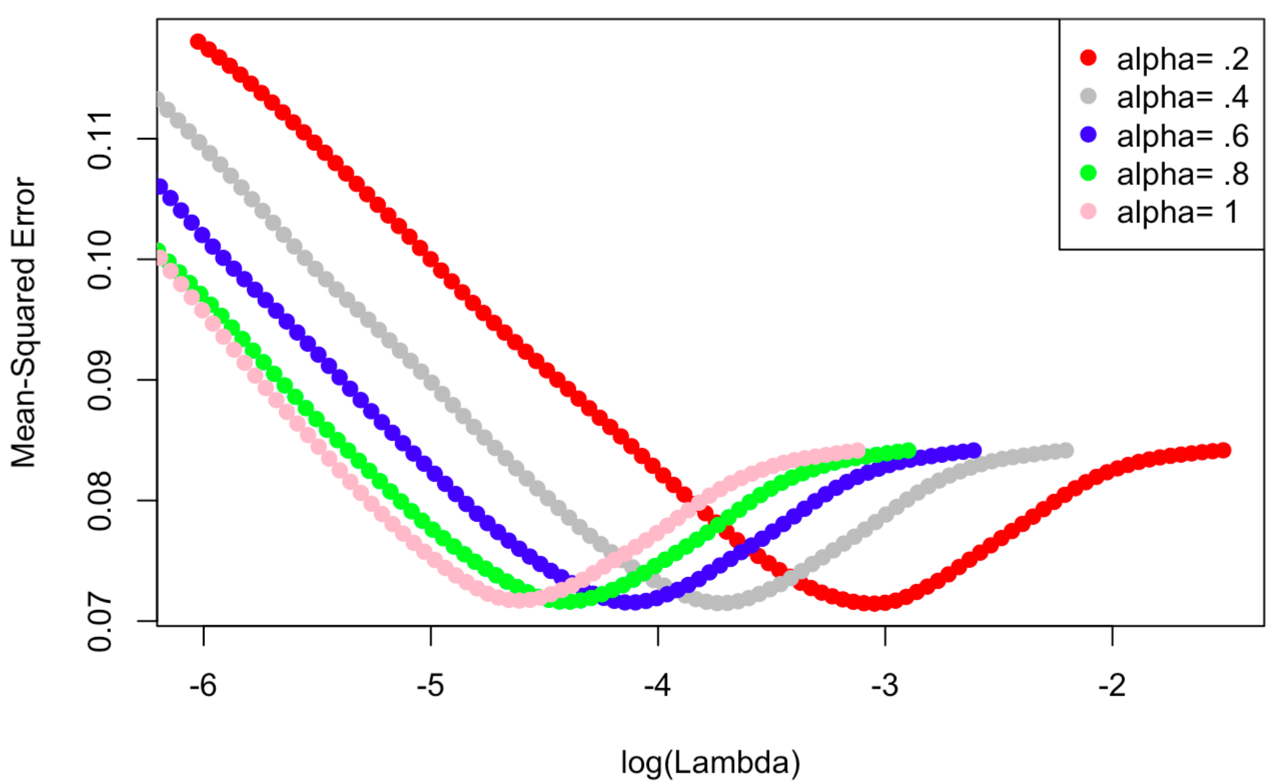


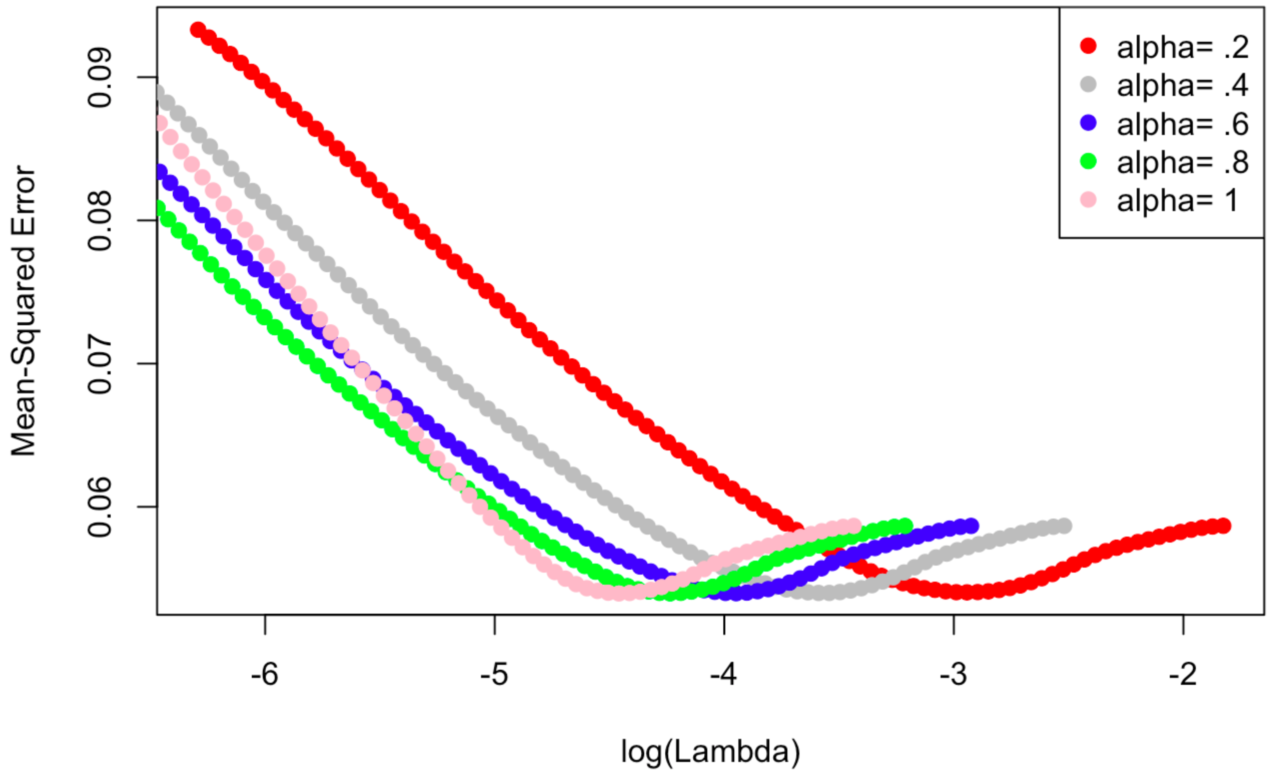
alpha= .2
alpha= .4
alpha= .6
alpha= .8
alpha= 1
Mean-Squared Error
0.09
0.10
0.11
0.12
0.13
0.14
-5
-4
-3
-2
log(Lambda)

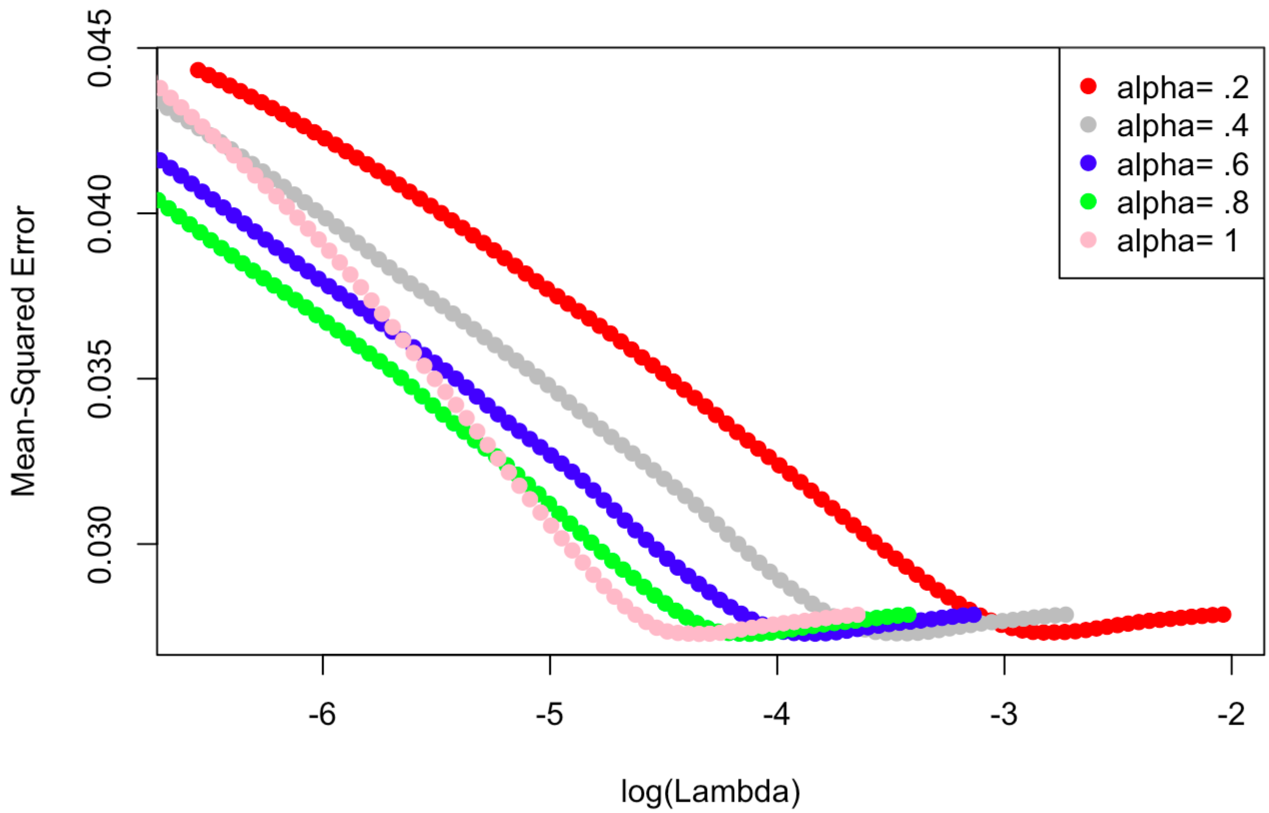


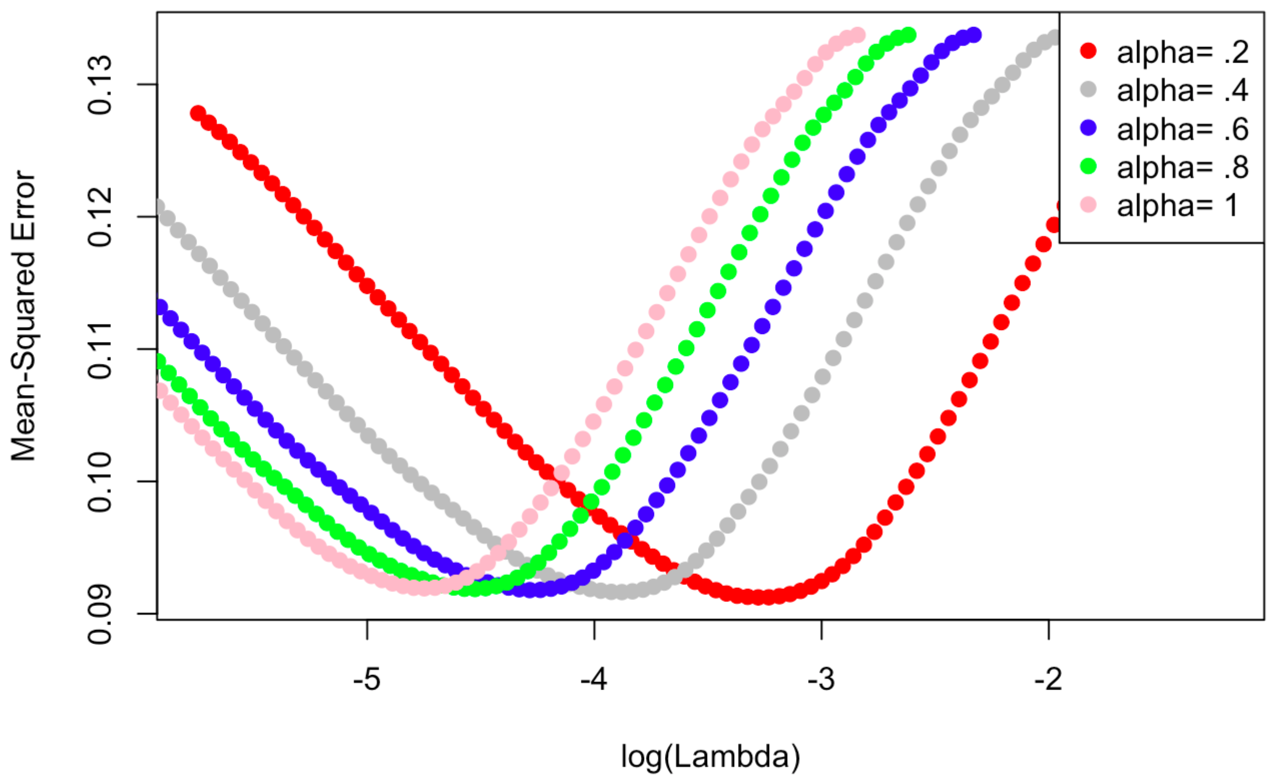
alpha= .2
alpha= .4
alpha= .6
alpha= .8
alpha= 1
Mean-Squared Error
0.040
0.044
0.048
0.052
-5
-4
-3
-2
-1
log(Lambda)

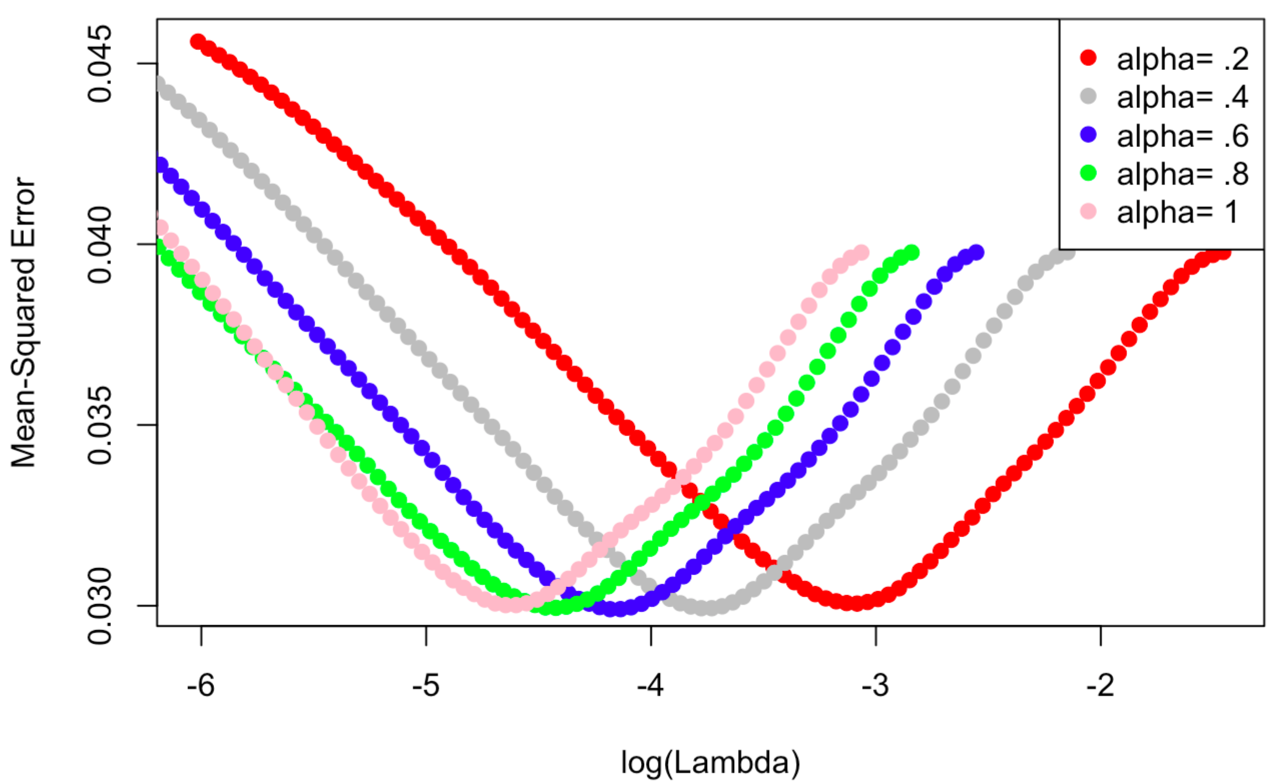

alpha= .2
alpha= .4
alpha= .6
alpha= .8
alpha= 1
Mean-Squared Error
0.22
0.24
0.26
0.28
0.30
0.32
0.34
-5
-4
-3
-2
log(Lambda)

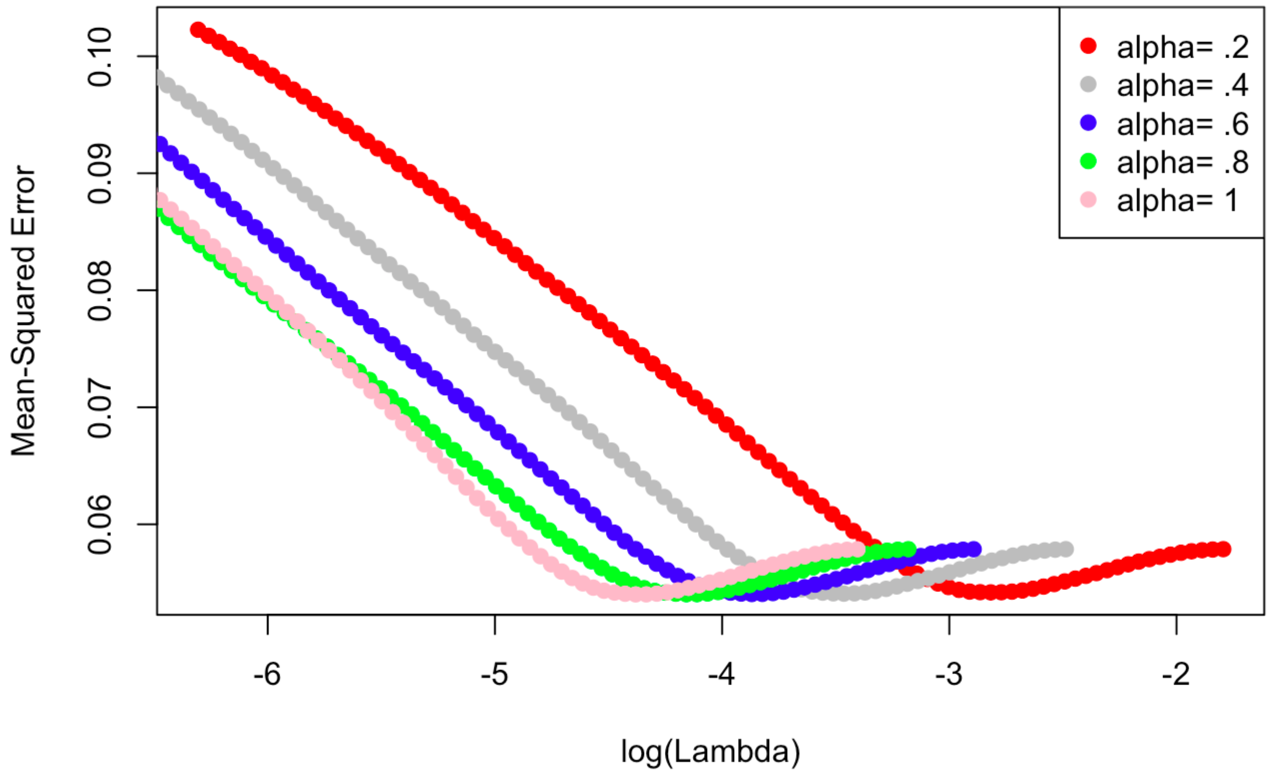

alpha= .2
alpha= .4
alpha= .6
alpha= .8
alpha= 1
Mean-Squared Error
0.07
0.08
0.09
0.10
0.11
-6
-5
-4
-3
-2
log(Lambda)

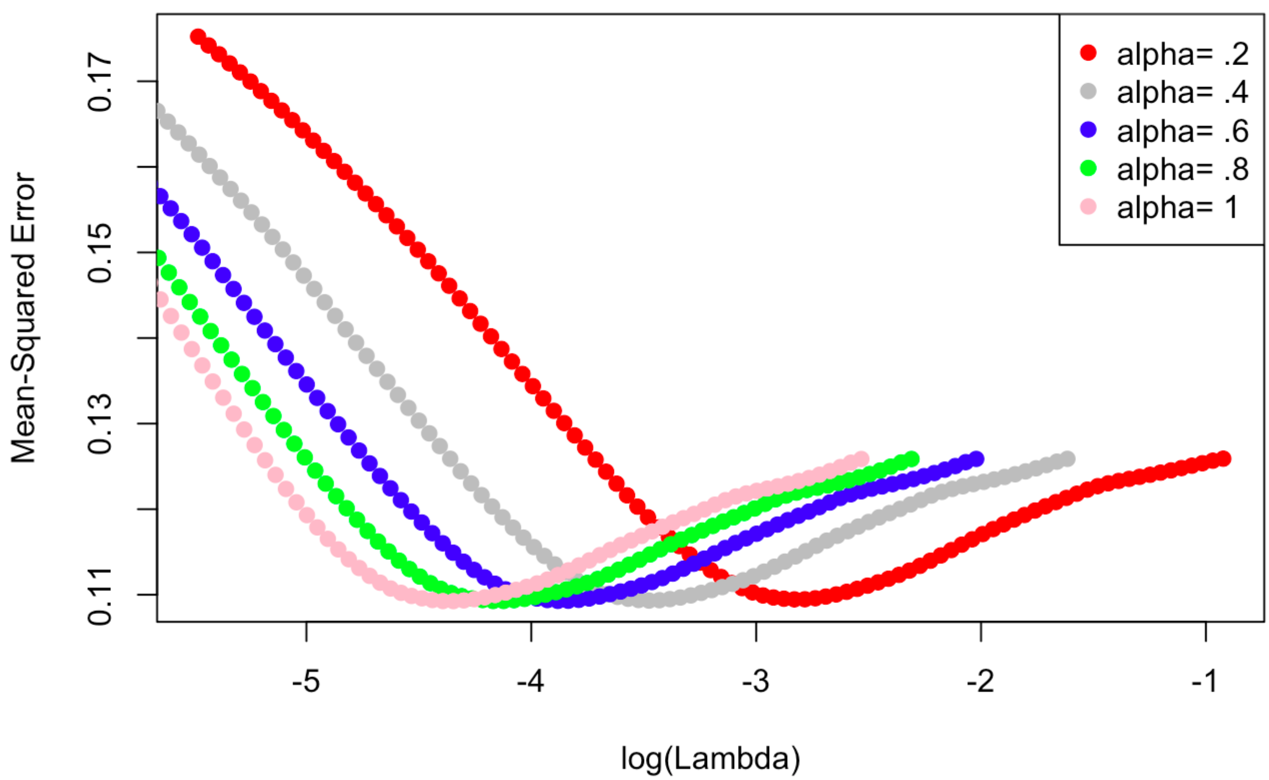

alpha= .2
alpha= .4
alpha= .6
alpha= .8
alpha= 1
Mean-Squared Error
0.06
0.07
0.08
0.09
log(Lambda)
-6
-5
-4
-3
-2

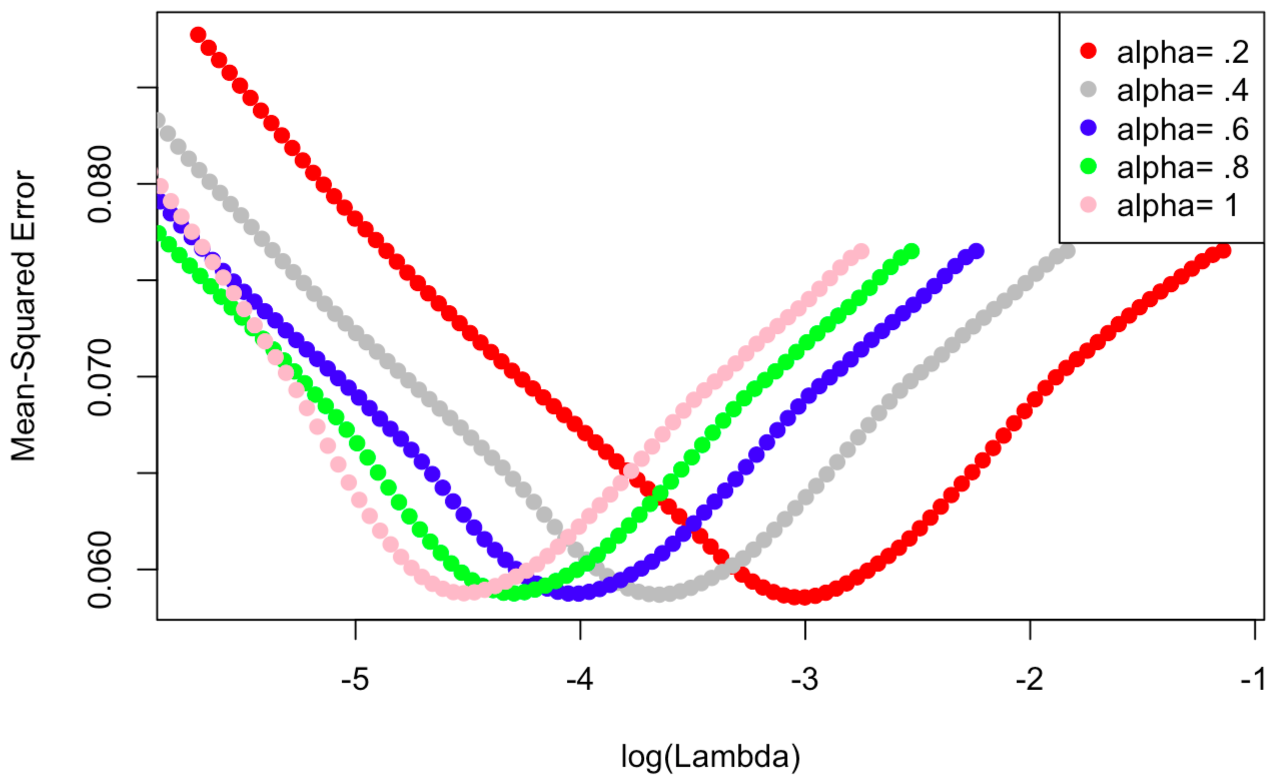

alpha= .2
alpha= .4
alpha= .6
alpha= .8
alpha= 1
Mean-Squared Error
0.030
0.035
0.040
0.045
log(Lambda)
-6
-5
-4
-3
-2

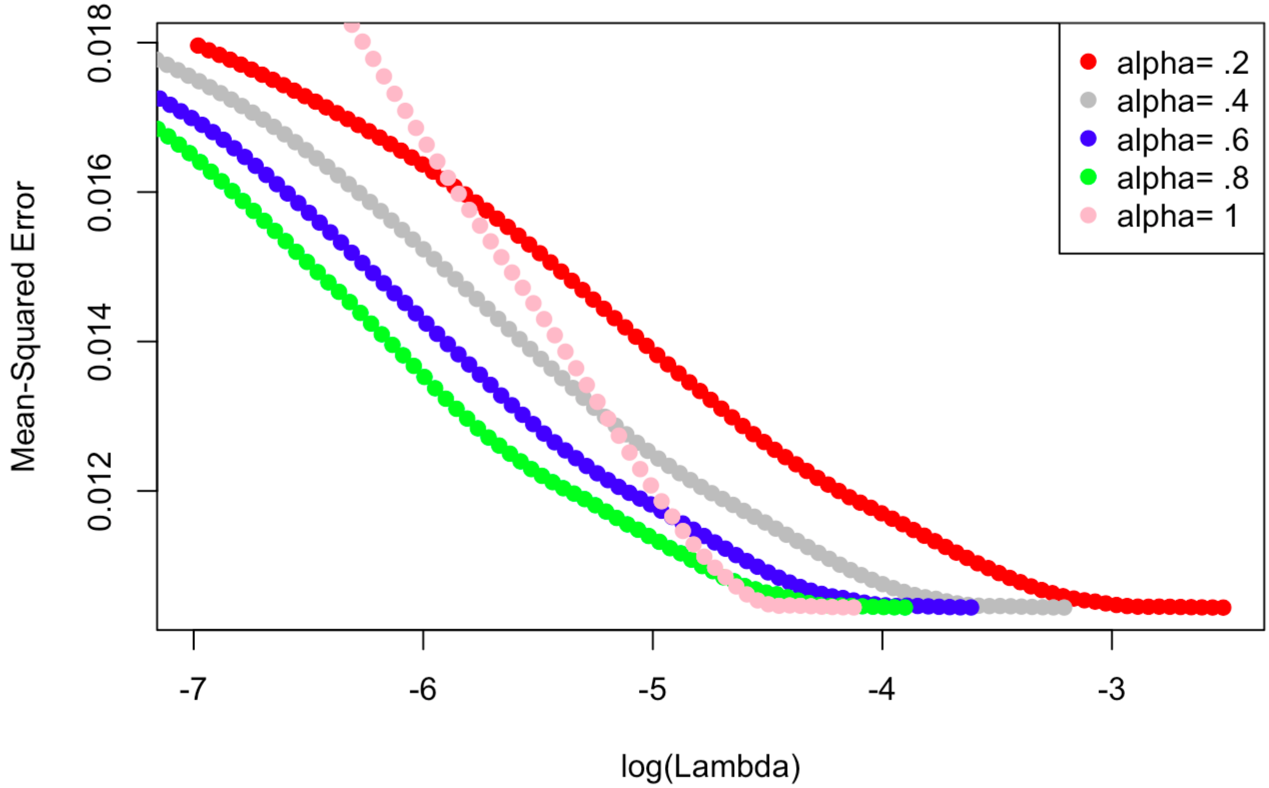

Mean-Squared Error
0.09
0.10
0.11
0.12
0.13
-5
-4
-3
-2
log(Lambda)
alpha= .2
alpha= .4
alpha= .6
alpha= .8
alpha= 1

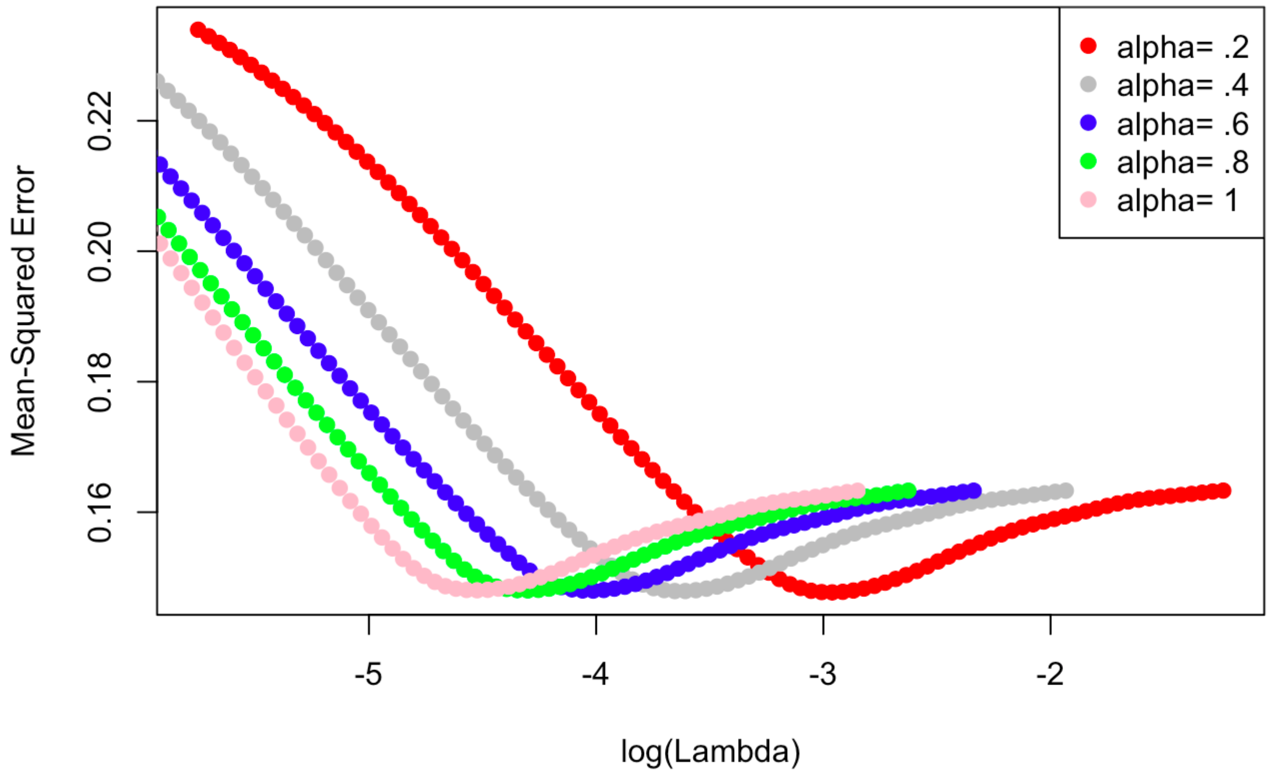

Mean-Squared Error
0.030
0.035
0.040
0.045
-6
-5
-4
-3
-2
log(Lambda)
alpha= .2
alpha= .4
alpha= .6
alpha= .8
alpha= 1

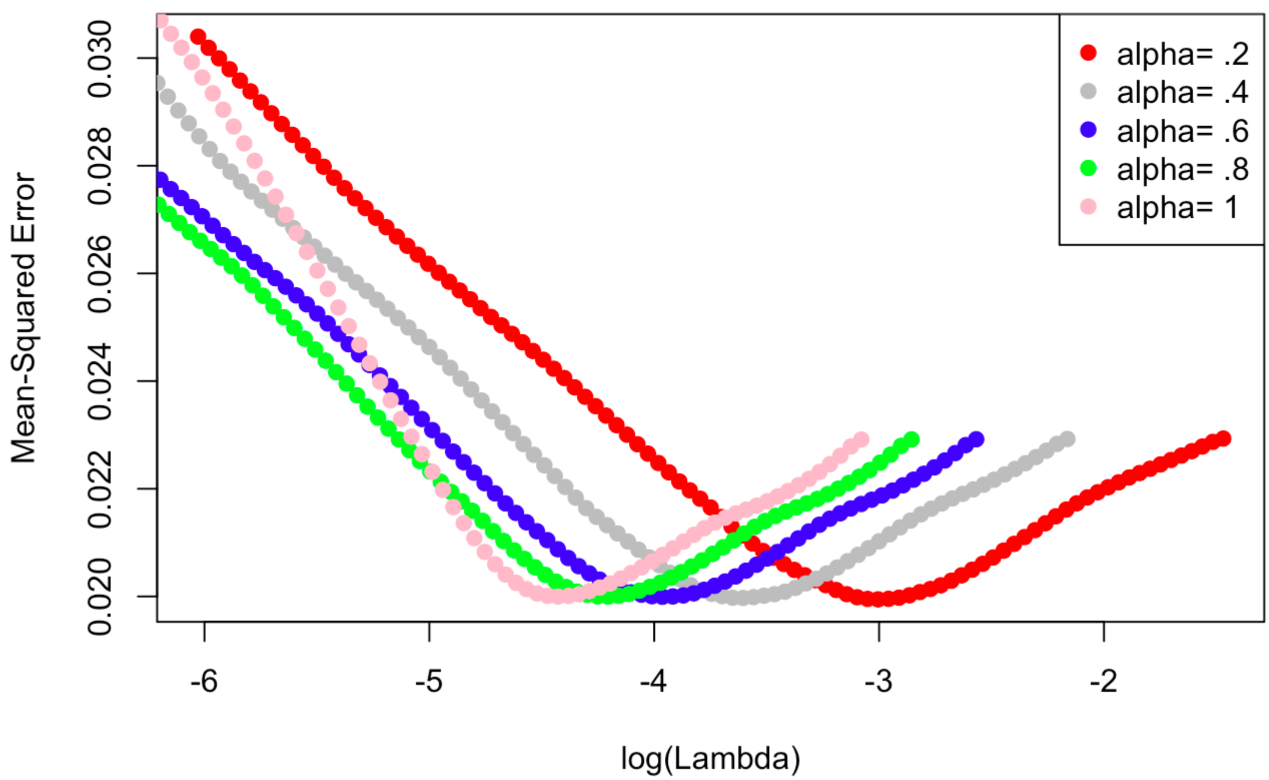

alpha= .2
alpha= .4
alpha= .6
alpha= .8
alpha= 1
Mean-Squared Error
0.06
0.07
0.08
0.09
0.10
-6
-5
-4
-3
-2
log(Lambda)

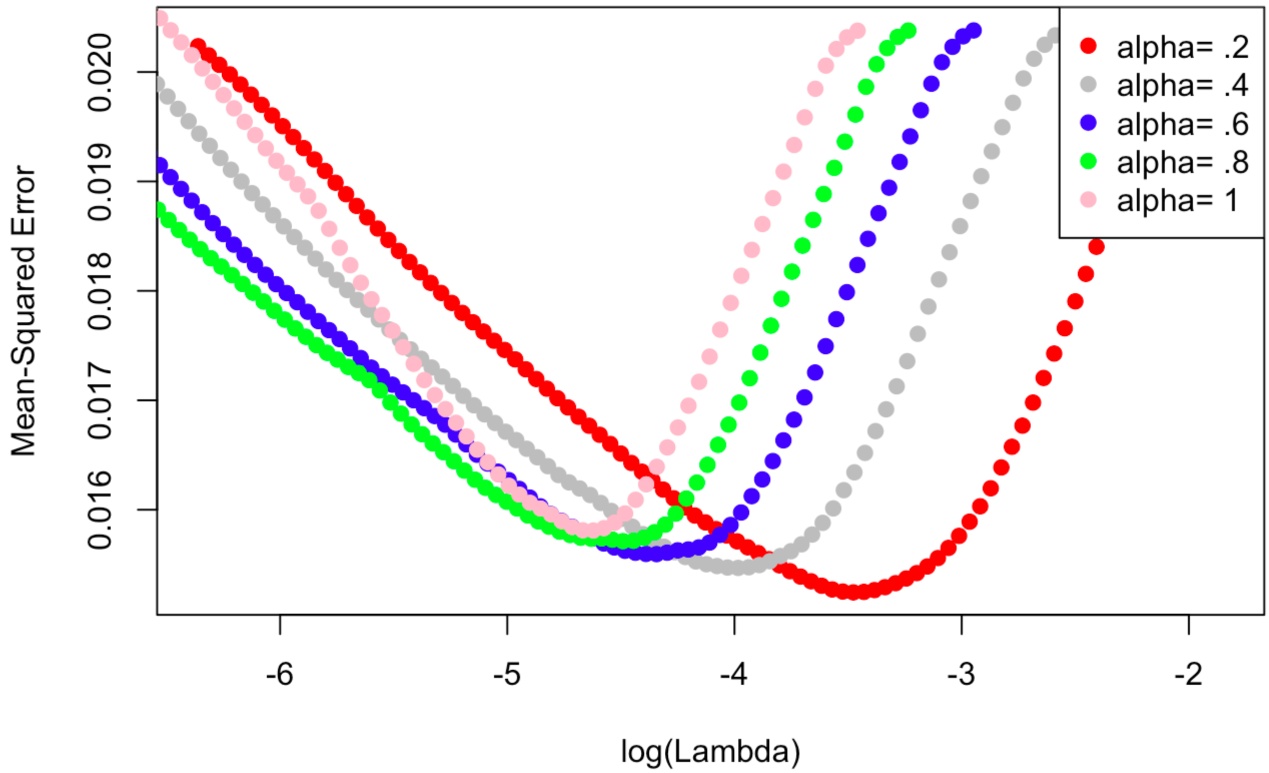

alpha= .2
alpha= .4
alpha= .6
alpha= .8
alpha= 1
Mean-Squared Error
0.11
0.13
0.15
0.17
-5
-4
-3
-2
-1
log(Lambda)


alpha= .2
alpha= .4
alpha= .6
alpha= .8
alpha= 1
Mean-Squared Error
0.060
0.070
0.080
-5
-4
-3
-2
-1
log(Lambda)


alpha= .2
alpha= .4
alpha= .6
alpha= .8
alpha= 1
Mean-Squared Error
0.012
0.014
0.016
0.018
-7
-6
-5
-4
-3
log(Lambda)


alpha= .2
alpha= .4
alpha= .6
alpha= .8
alpha= 1
Mean-Squared Error
0.16
0.18
0.20
0.22
-5
-4
-3
-2
log(Lambda)


alpha= .2
alpha= .4
alpha= .6
alpha= .8
alpha= 1
Mean-Squared Error
0.020
0.022
0.024
0.026
0.028
0.030
-6
-5
-4
-3
-2
log(Lambda)


alpha= .2
alpha= .4
alpha= .6
alpha= .8
alpha= 1
Mean-Squared Error
0.016
0.017
0.018
0.019
0.020
-6
-5
-4
-3
-2
log(Lambda)

```
## $Action
##    mse.min1  mse.min2  mse.min3 mse.min4 mse.min5
## 1 0.1314703 0.1317915 0.1319145 0.131965 0.131992
##
## $Adventure
##     mse.min1   mse.min2   mse.min3   mse.min4   mse.min5
## 1 0.08582935 0.08594284 0.08598045 0.08600346 0.08601903
##
## $Animation
##     mse.min1  mse.min2   mse.min3   mse.min4   mse.min5
## 1 0.04894493 0.0490979 0.04917851 0.04922532 0.04926092
##
## $Comedy
##    mse.min1  mse.min2  mse.min3  mse.min4  mse.min5
## 1 0.1936264 0.1941154 0.1943044 0.1943899 0.1944444
##
## $Crime
##     mse.min1   mse.min2  mse.min3   mse.min4   mse.min5
## 1 0.08592985 0.08588375 0.0858618 0.08584053 0.08581989
##
## $Documentary
##     mse.min1   mse.min2  mse.min3   mse.min4   mse.min5
## 1 0.03924444 0.03934108 0.0393809 0.03941531 0.03943377
##
## $Drama
##    mse.min1  mse.min2  mse.min3  mse.min4  mse.min5
## 1 0.2114327 0.2117375 0.2118668 0.2119374 0.2119824
##
## $Family
##     mse.min1   mse.min2   mse.min3 mse.min4   mse.min5
## 1 0.07145284 0.07149263 0.07154895 0.071615 0.07170642
##
## $Fantasy
##     mse.min1   mse.min2   mse.min3   mse.min4   mse.min5
## 1 0.05402974 0.05398871 0.05396757 0.05396086 0.05395526
##
## $History
##     mse.min1   mse.min2   mse.min3   mse.min4   mse.min5
## 1 0.02732669 0.02730202 0.02729196 0.02728698 0.02728417
##
## $Horror
##     mse.min1   mse.min2   mse.min3   mse.min4   mse.min5
## 1 0.09123909 0.09163592 0.09178674 0.09186869 0.09191986
##
## $Music
##     mse.min1   mse.min2  mse.min3   mse.min4  mse.min5
## 1 0.03006008 0.02993145 0.0299041 0.02994221 0.0300176
##
## $Mystery
##     mse.min1   mse.min2   mse.min3   mse.min4   mse.min5
## 1 0.05418517 0.05407481 0.05404158 0.05402472 0.05401504
##
## $Romance
```

```
##    mse.min1  mse.min2  mse.min3  mse.min4  mse.min5
## 1 0.1094639 0.1093188 0.1092847 0.1092715 0.1092596
##
## $Science.Fiction
##     mse.min1   mse.min2  mse.min3   mse.min4  mse.min5
## 1 0.05855349 0.05869238 0.0587488 0.05875921 0.0587681
##
## $TV.Movie
##     mse.min1   mse.min2   mse.min3   mse.min4   mse.min5
## 1 0.01043936 0.01044018 0.01044065 0.01044093 0.01044113
##
## $Thriller
##    mse.min1  mse.min2  mse.min3  mse.min4  mse.min5
## 1 0.1477111 0.1479129 0.1479786 0.1480137 0.1480309
##
## $War
##     mse.min1   mse.min2   mse.min3   mse.min4   mse.min5
## 1 0.01994616 0.01997206 0.01999398 0.01999767 0.01999765
##
## $Western
##     mse.min1   mse.min2   mse.min3   mse.min4   mse.min5
## 1 0.01524505 0.01546854 0.01559427 0.01571254 0.01581002
```

```
    Action  Adventure  Animation    Comedy      Crime Documentary     Drama     Family
  Fantasy
```

[1,] 0.1325201 0.08906477 0.04864058 0.1941113 0.08426901 0.04064171 0.2127845 0.07218937 0.05319670
[2,] 0.1328166 0.08916923 0.04876307 0.1944108 0.08424354 0.04065542 0.2131184 0.07212872 0.05325774
[3,] 0.1329427 0.08919738 0.04881739 0.1945339 0.08425693 0.04067155 0.2132548 0.07209167 0.05328253
[4,] 0.1330485 0.08921478 0.04884216 0.1946057 0.08427830 0.04069139 0.2133211 0.07207209 0.05329398
[5,] 0.1331501 0.08922616 0.04884504 0.1946527 0.08429573 0.04070574 0.2133567 0.07205487 0.05330376
History Horror Music Mystery Romance Science.Fiction TV.Movie Thriller [1,] 0.02709648 0.09559806 0.03181055 0.05397884 0.1119673 0.05942226 0.01004372 0.1471946 [2,] 0.02721936 0.09574700 0.03175188 0.05395156 0.1117565 0.05945573 0.01004375 0.1473630 [3,] 0.02725629 0.09581951 0.03177520 0.05395159 0.1116973 0.05946177 0.01004377 0.1474183 [4,] 0.02726468 0.09586636 0.03183213 0.05395328 0.1116662 0.05947532 0.01004370 0.1474496 [5,] 0.02726783 0.09589498 0.03187170 0.05395127 0.1116450 0.05947457 0.01004367 0.1474710 War Western [1,] 0.01942773 0.01520812 [2,] 0.01941971 0.01523102 [3,] 0.01940142 0.01526231 [4,] 0.01943716 0.01538277 [5,] 0.01954477 0.01549498

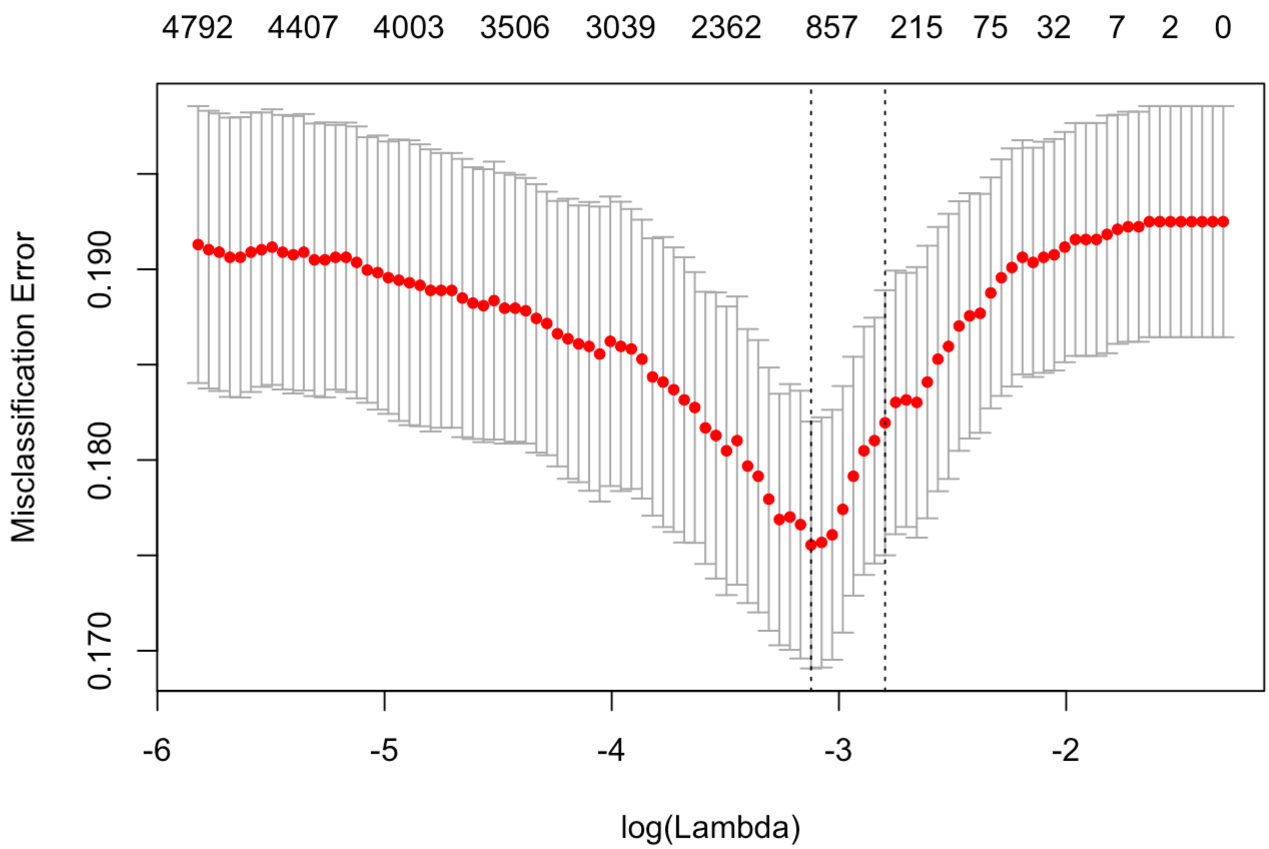
Even though cv.glmnet auto-tunes for the best value of lambda, and consequently alpha, experimenting with the model showed that setting an alpha value resulted in a slight increase in accuracy. For example for the Action genre, accuracy was 82.319 without a user-set alpha value, and 82.339 with an alpha value of 0.2. For the 19 models tested below, the optimal value of alpha was chosen based on the previous output.

```
NFOLDS = 5
thresh = 1e-5 #low thresh value chosen for improving accuracy
maxit = 1e3 #1000 maximum iterations

#set to family= binomial due to 0-1 labeling in the genre lables matrix.

#action
glmnet_classifierA = cv.glmnet(x = dtm_train_tfidf, y = as.matrix(train[,15]),
                              family = 'binomial',
                              type.measure = "class",
                              alpha = 0.2,
                              nfolds = NFOLDS,
                              thresh = thresh,
                              maxit = maxit)
#adv.
glmnet_classifierB = cv.glmnet(x = dtm_train_tfidf, y = as.matrix(train[,16]),
                              family = 'binomial',
                              alpha = 0.2,
                              type.measure = "class",
                              nfolds = NFOLDS,
                              thresh = thresh,
                              maxit = maxit)
#animation
glmnet_classifierC = cv.glmnet(x = dtm_train_tfidf, y = as.matrix(train[,17]),
                              family = 'binomial',
                              alpha = 0.2,
                              type.measure = "class",
                              nfolds = NFOLDS,
                              thresh = thresh,
                              maxit = maxit)
#comedy
glmnet_classifierD = cv.glmnet(x = dtm_train_tfidf, y = as.matrix(train[,18]),
                              family = 'binomial',
                              alpha = 0.2,
                              type.measure = "class",
                              nfolds = NFOLDS,
                              thresh = thresh,
                              maxit = maxit)
#crime
glmnet_classifierE = cv.glmnet(x = dtm_train_tfidf, y = as.matrix(train[,19]),
                              family = 'binomial',
                              alpha = 0.4,
                              type.measure = "class",
                              nfolds = NFOLDS,
                              thresh = thresh,
                              maxit = maxit)
#docs
glmnet_classifierF = cv.glmnet(x = dtm_train_tfidf, y = as.matrix(train[,20]),
                              family = 'binomial',
                              alpha = 0.2,
                              type.measure = "class",
                              nfolds = NFOLDS,
                              thresh = thresh,
```

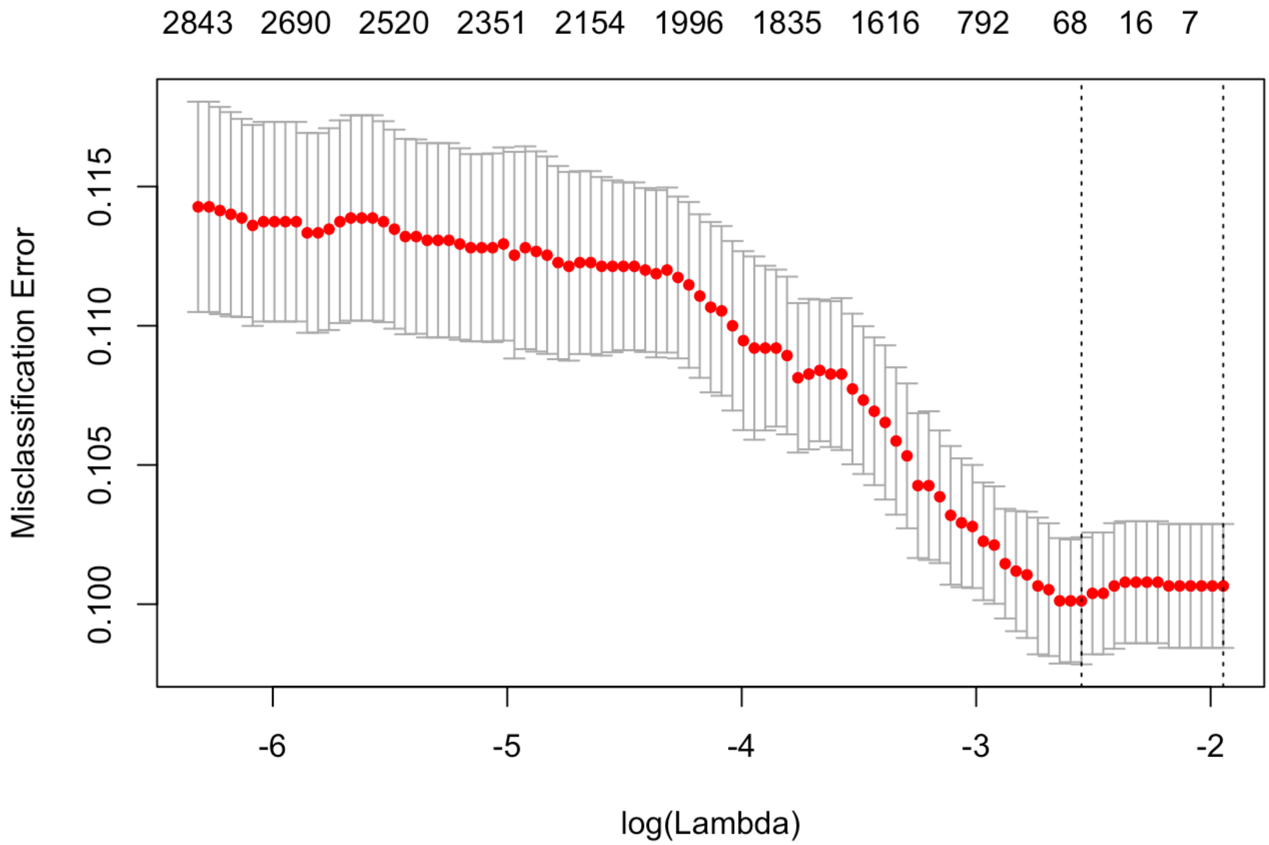
```
                              maxit = maxit)
#drama
glmnet_classifierG = cv.glmnet(x = dtm_train_tfidf, y = as.matrix(train[,21]),
                              family = 'binomial',
                              alpha = 0.2,
                              type.measure = "class",
                              nfolds = NFOLDS,
                              thresh = thresh,
                              maxit = maxit)
#family
glmnet_classifierH = cv.glmnet(x = dtm_train_tfidf, y = as.matrix(train[,22]),
                              family = 'binomial',
                              alpha = 1,
                              type.measure = "class",
                              nfolds = NFOLDS,
                              thresh = thresh,
                              maxit = maxit)
#fantasy
glmnet_classifierI = cv.glmnet(x = dtm_train_tfidf, y = as.matrix(train[,23]),
                              family = 'binomial',
                              alpha = 0.2,
                              type.measure = "class",
                              nfolds = NFOLDS,
                              thresh = thresh,
                              maxit = maxit)

#hist
glmnet_classifierJ = cv.glmnet(x = dtm_train_tfidf, y = as.matrix(train[,24]),
                              family = 'binomial',
                              alpha = 0.2,
                              type.measure = "class",
                              nfolds = NFOLDS,
                              thresh = thresh,
                              maxit = maxit)

#horror
glmnet_classifierK = cv.glmnet(x = dtm_train_tfidf, y = as.matrix(train[,25]),
                              family = 'binomial',
                              alpha = 0.2,
                              type.measure = "class",
                              nfolds = NFOLDS,
                              thresh = thresh,
                              maxit = maxit)

#music
glmnet_classifierL = cv.glmnet(x = dtm_train_tfidf, y = as.matrix(train[,26]),
                              family = 'binomial',
                              alpha = 0.4,
                              type.measure = "class",
                              nfolds = NFOLDS,
                              thresh = thresh,
                              maxit = maxit)

#mystery
```

```
glmnet_classifierM = cv.glmnet(x = dtm_train_tfidf, y = as.matrix(train[,27]),
                               family = 'binomial',
                               alpha = 1,
                               type.measure = "class",
                               nfolds = NFOLDS,
                               thresh = thresh,
                               maxit = maxit)

#romance
glmnet_classifierN = cv.glmnet(x = dtm_train_tfidf, y = as.matrix(train[,28]),
                               family = 'binomial',
                               alpha = 1,
                               type.measure = "class",
                               nfolds = NFOLDS,
                               thresh = thresh,
                               maxit = maxit)

#sci-fi
glmnet_classifierO = cv.glmnet(x = dtm_train_tfidf, y = as.matrix(train[,29]),
                               family = 'binomial',
                               alpha = 0.2,
                               type.measure = "class",
                               nfolds = NFOLDS,
                               thresh = thresh,
                               maxit = maxit)

#tv-movie
glmnet_classifierP = cv.glmnet(x = dtm_train_tfidf, y = as.matrix(train[,30]),
                               family = 'binomial',
                               alpha = 1,
                               type.measure = "class",
                               nfolds = NFOLDS,
                               thresh = thresh,
                               maxit = maxit)

#thriller
glmnet_classifierQ = cv.glmnet(x = dtm_train_tfidf, y = as.matrix(train[,31]),
                               family = 'binomial',
                               alpha = 0.2,
                               type.measure = "class",
                               nfolds = NFOLDS,
                               thresh = thresh,
                               maxit = maxit)

#war
glmnet_classifierR = cv.glmnet(x = dtm_train_tfidf, y = as.matrix(train[,32]),
                               family = 'binomial',
                               alpha = 0.6,
                               type.measure = "class",
                               nfolds = NFOLDS,
                               thresh = thresh,
                               maxit = maxit)

#western
```

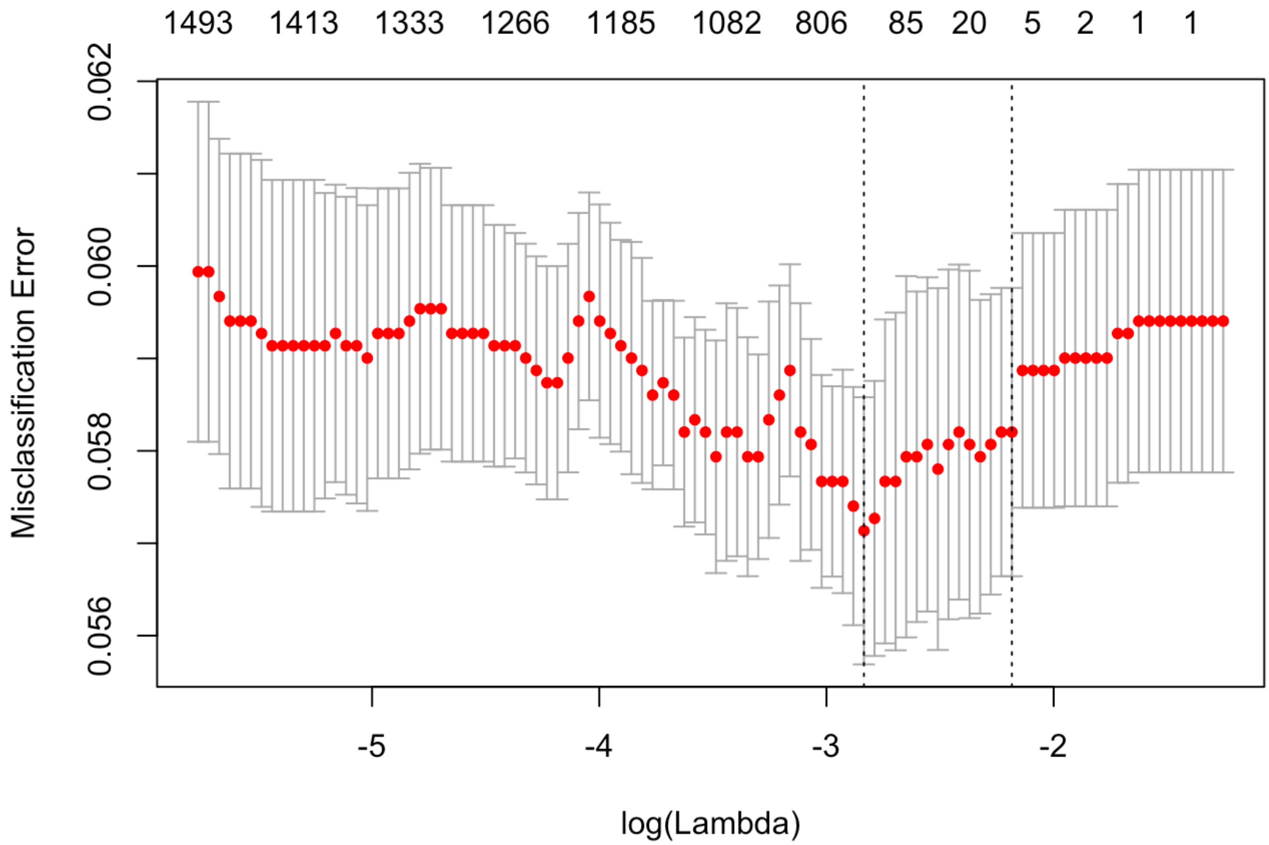
```
glmnet_classifierS = cv.glmnet(x = dtm_train_tfidf, y = as.matrix(train[,33]),
                              family = 'binomial',
                              alpha = 0.2,
                              type.measure = "class",
                              nfolds = NFOLDS,
                              thresh = thresh,
                              maxit = maxit)

#Missclassification error plots for different values of lambda
plot(glmnet_classifierA)
```



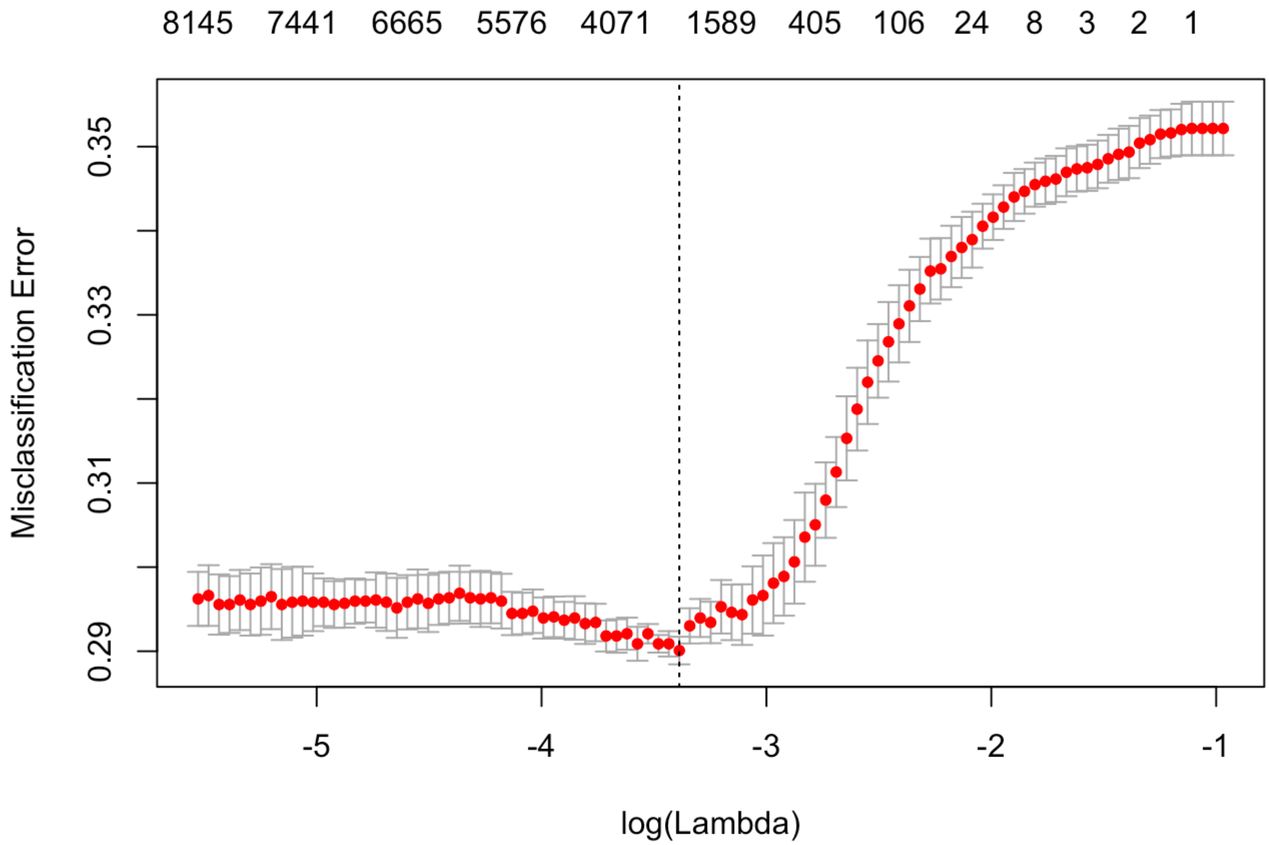

```
plot(glmnet_classifierB)
```



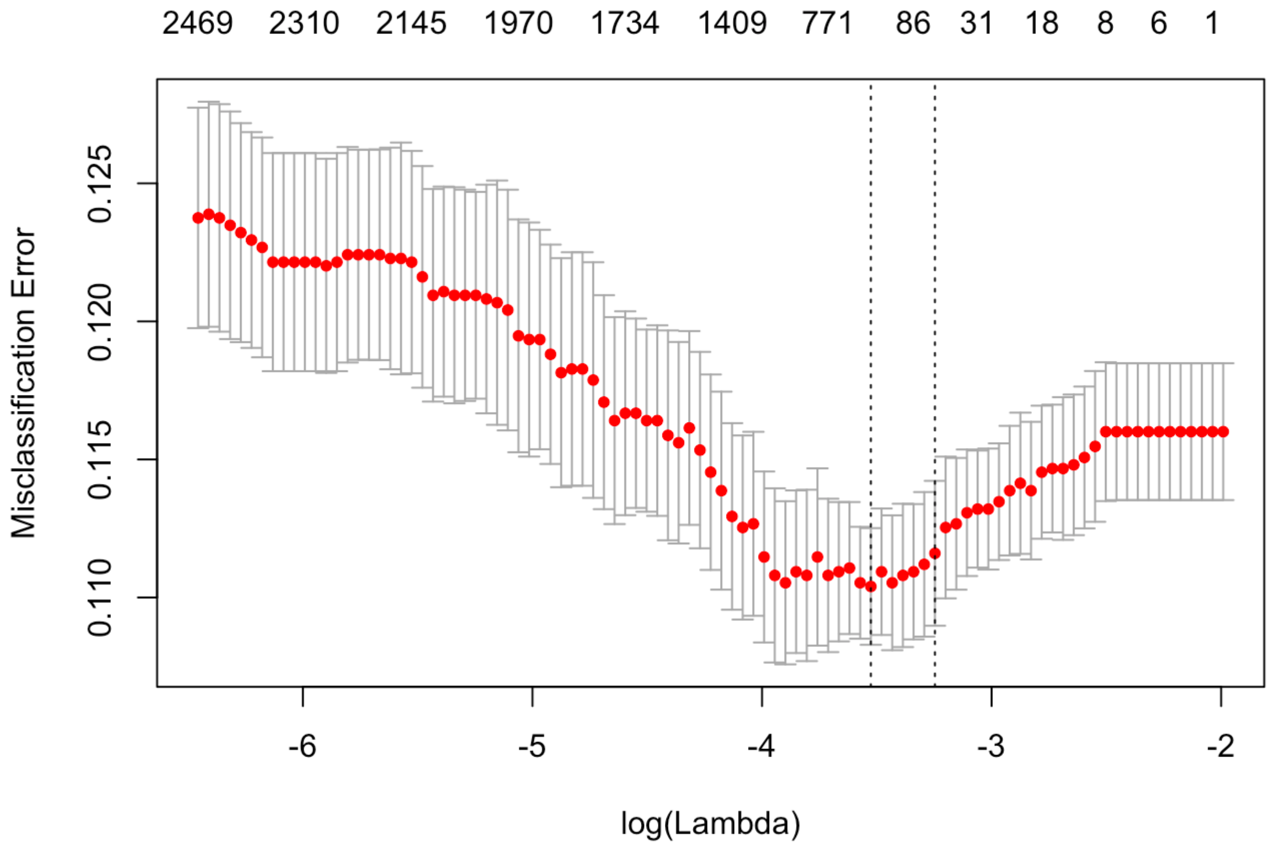

```
plot(glmnet_classifierC)
```



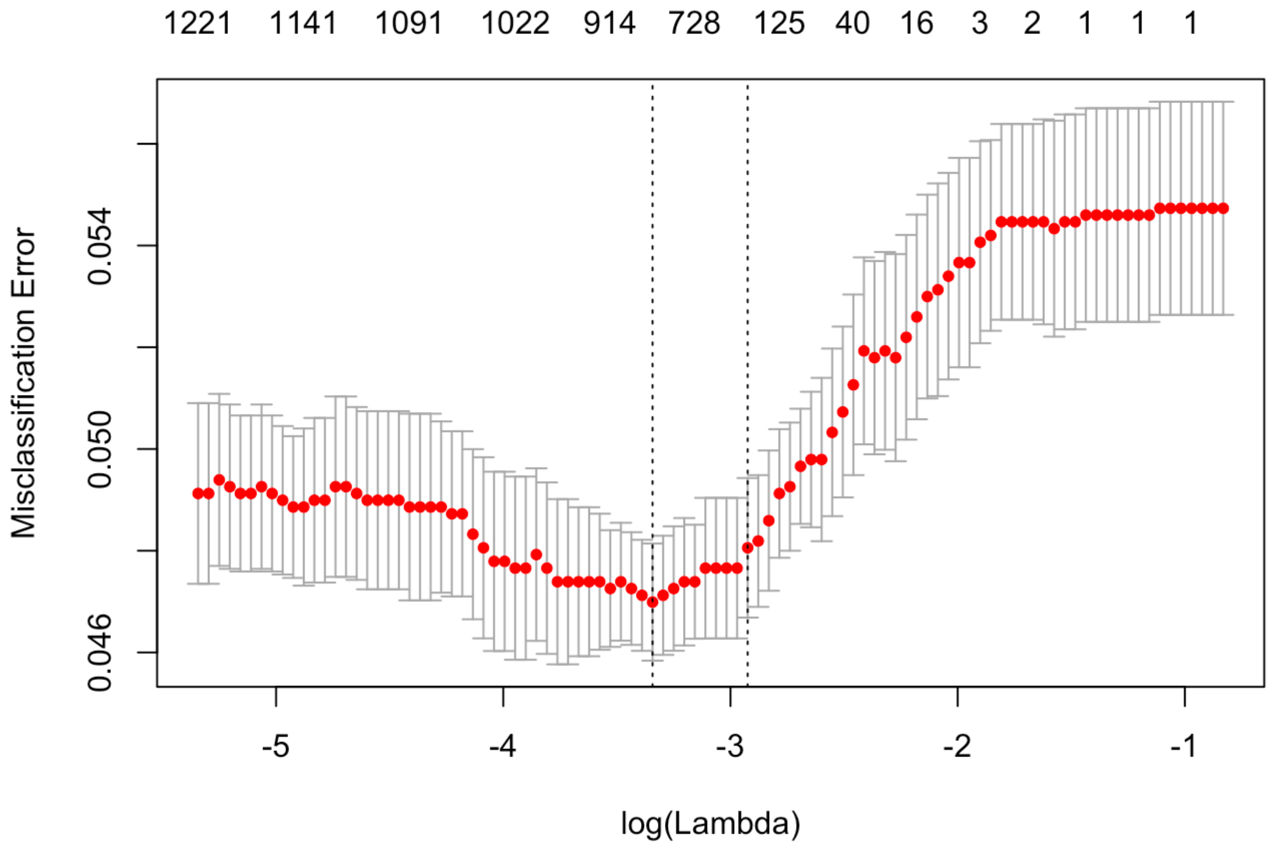

```
plot(glmnet_classifierD)
```



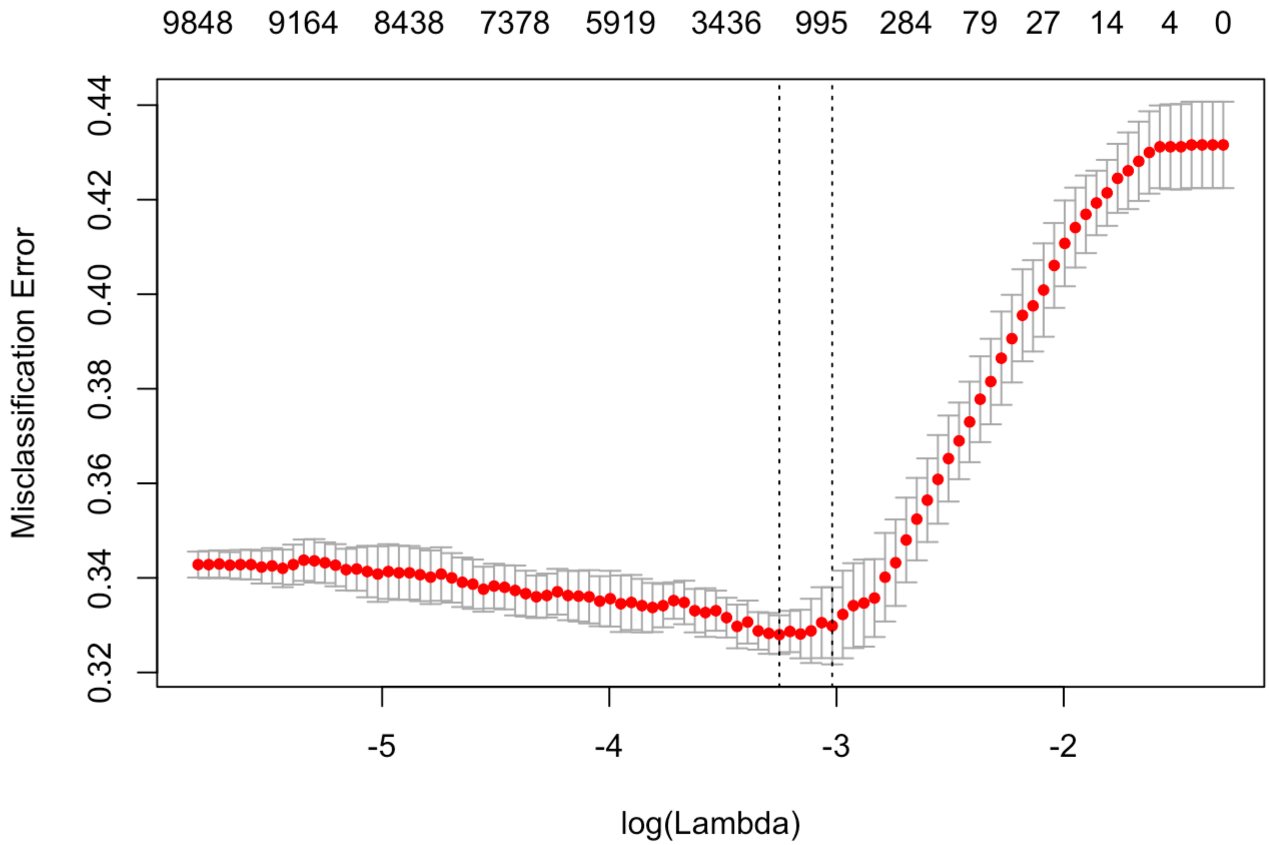

```
plot(glmnet_classifierE)
```



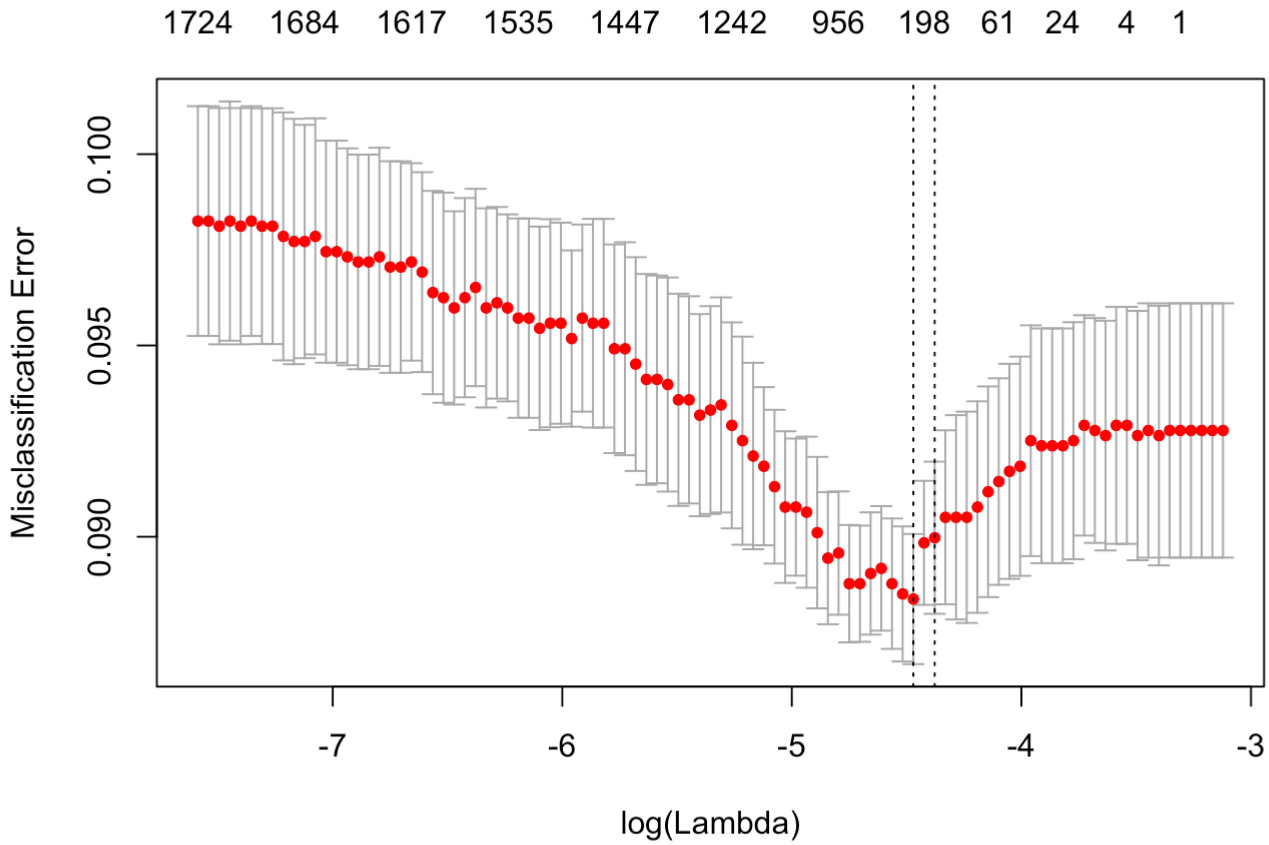

```
plot(glmnet_classifierF)
```



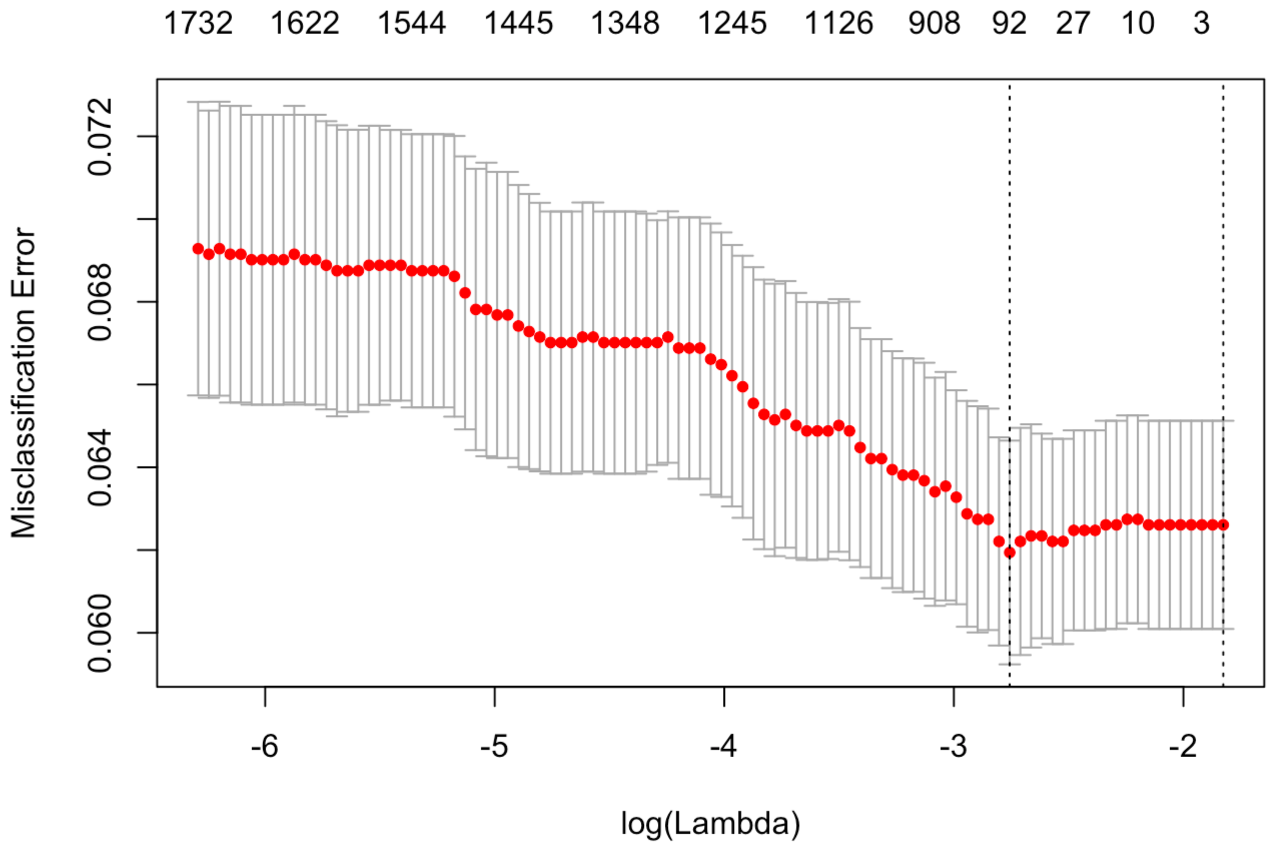

```
plot(glmnet_classifierG)
```



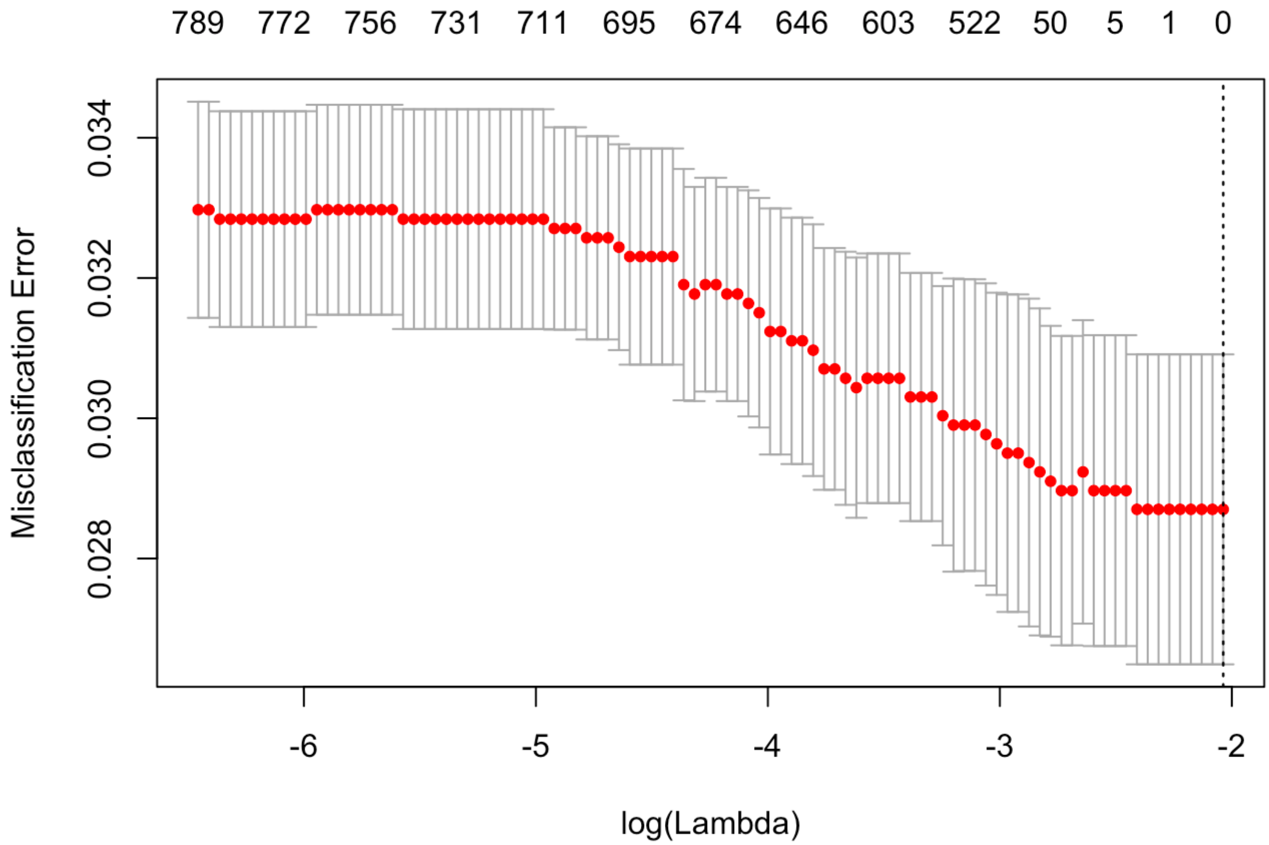

```
plot(glmnet_classifierH)
```



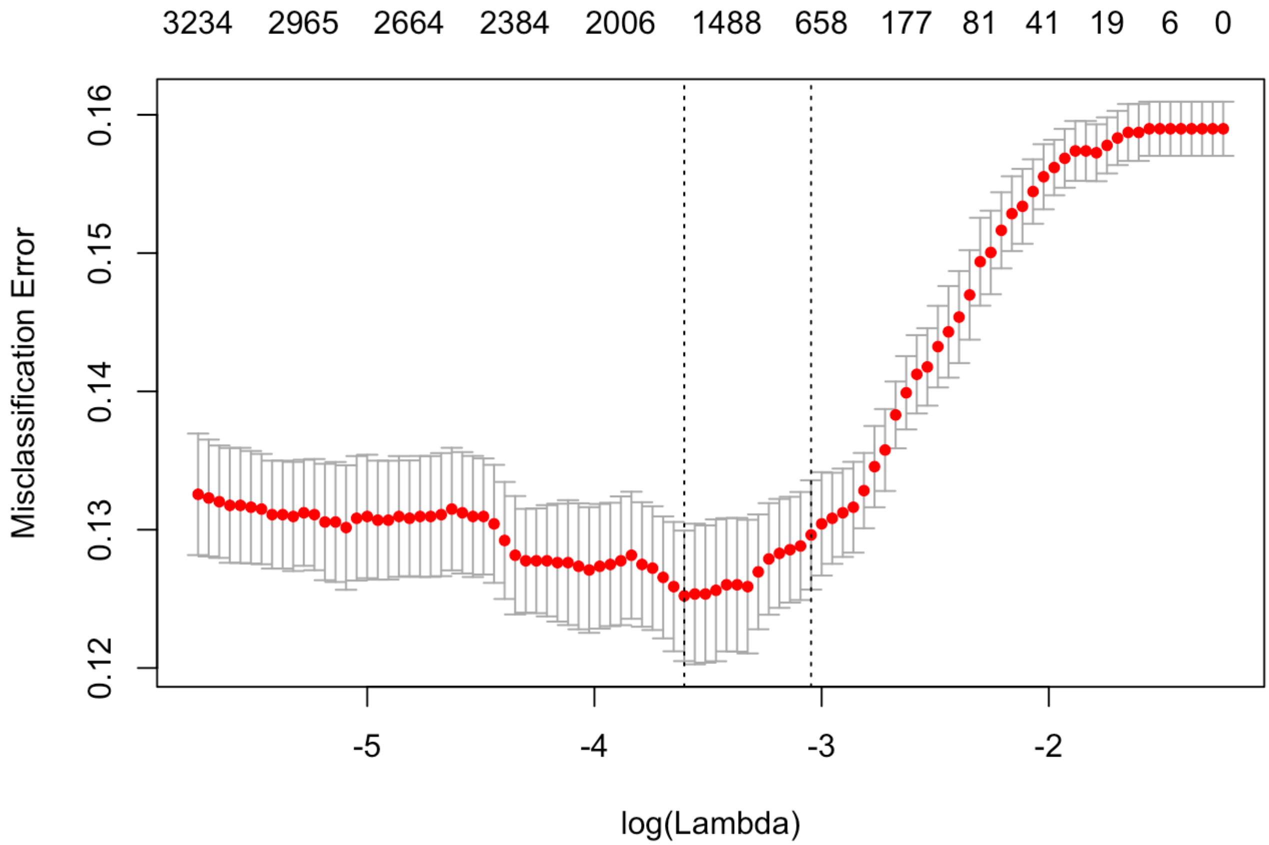

```
plot(glmnet_classifierI)
```



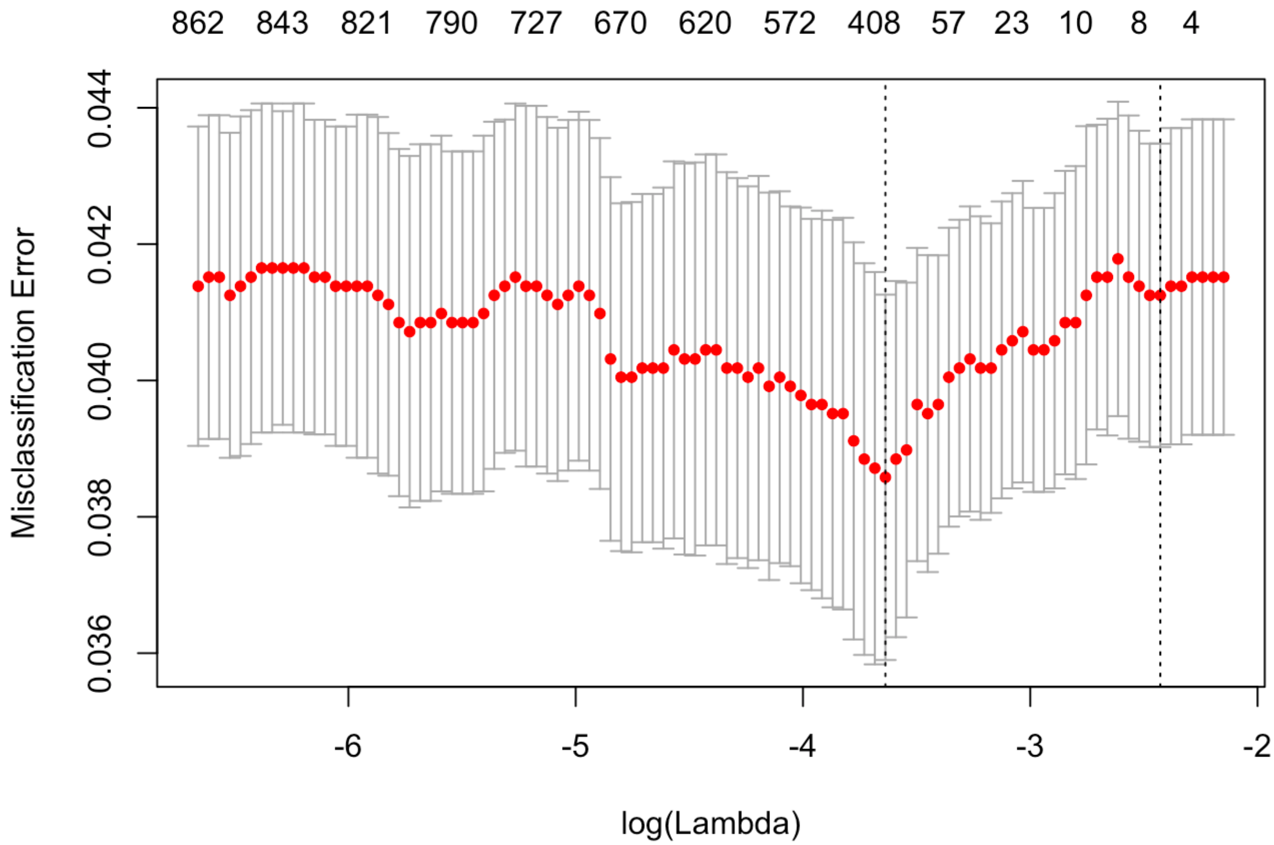

```
plot(glmnet_classifierJ)
```



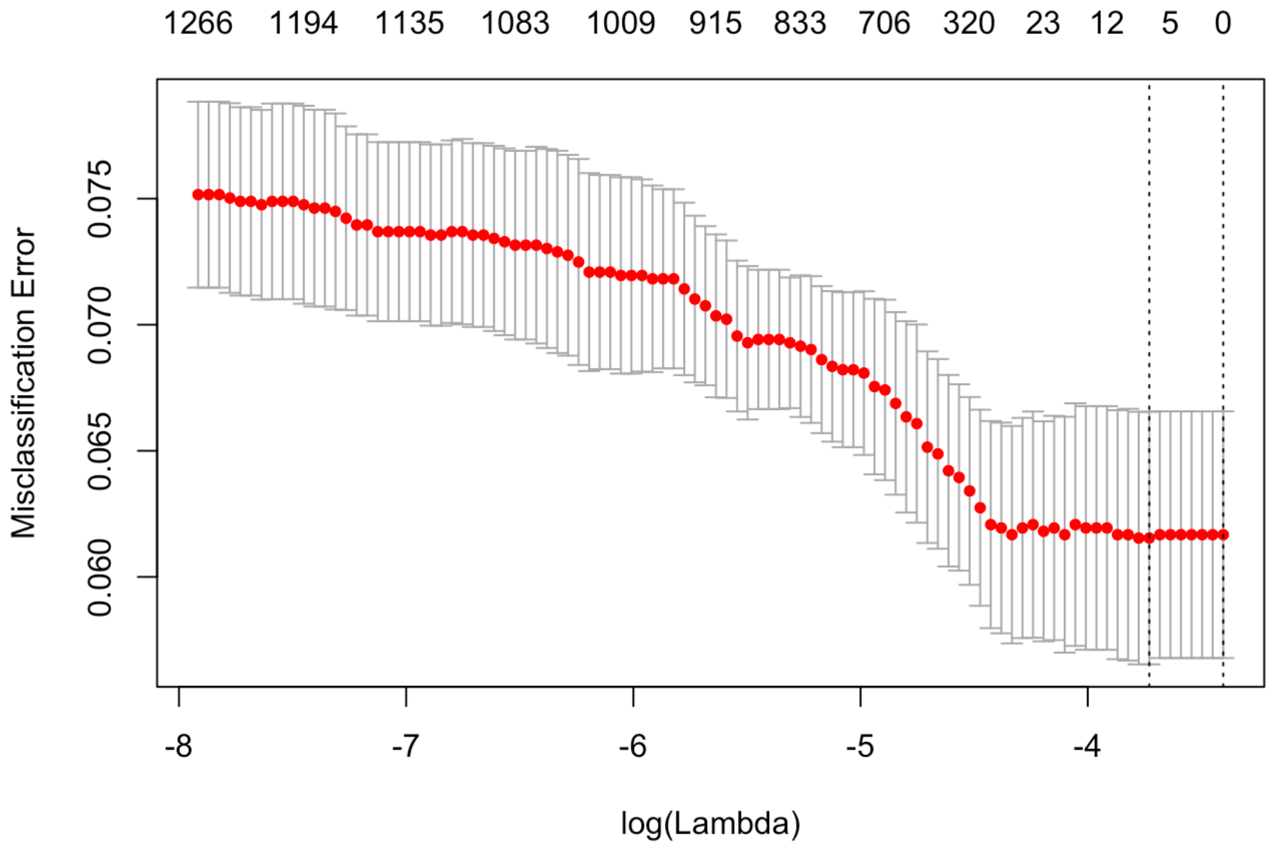

```
plot(glmnet_classifierK)
```



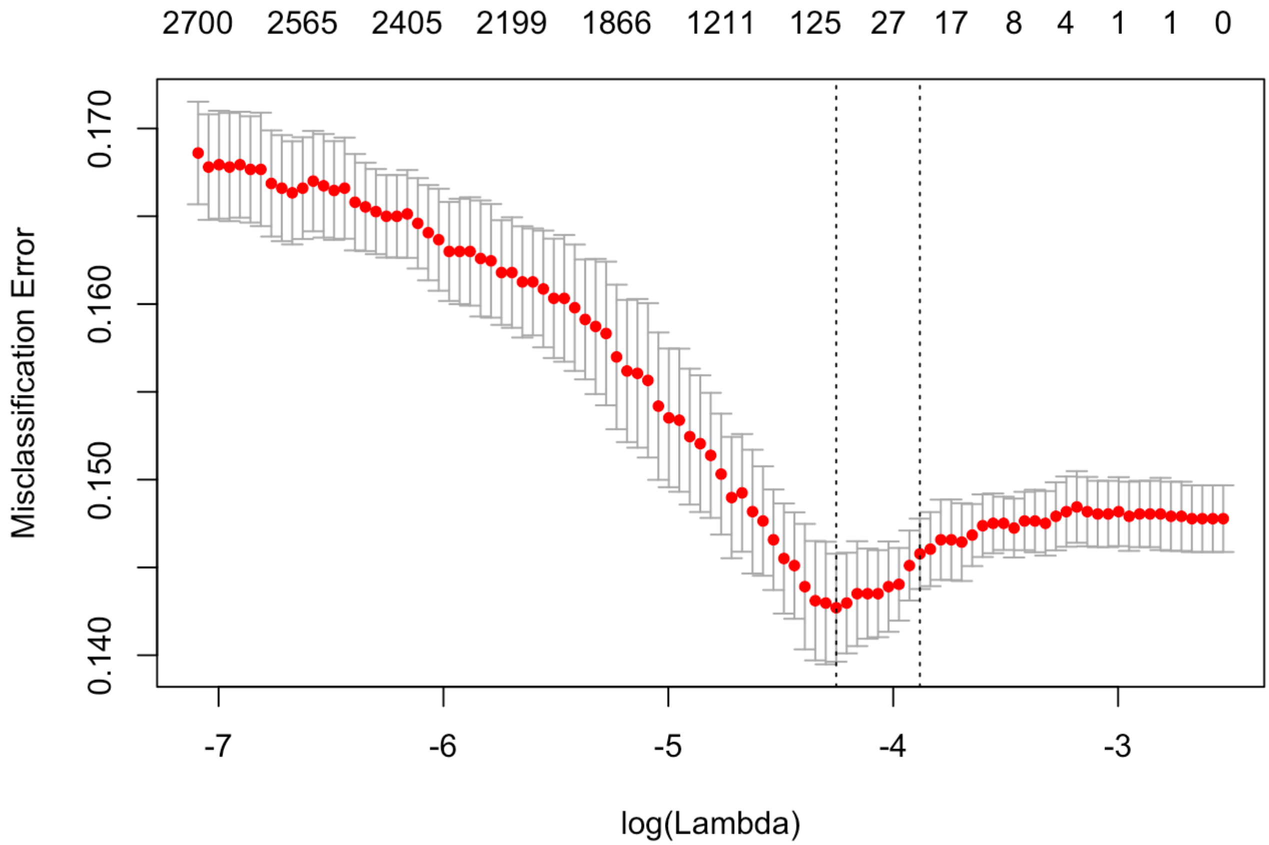

```
plot(glmnet_classifierL)
```



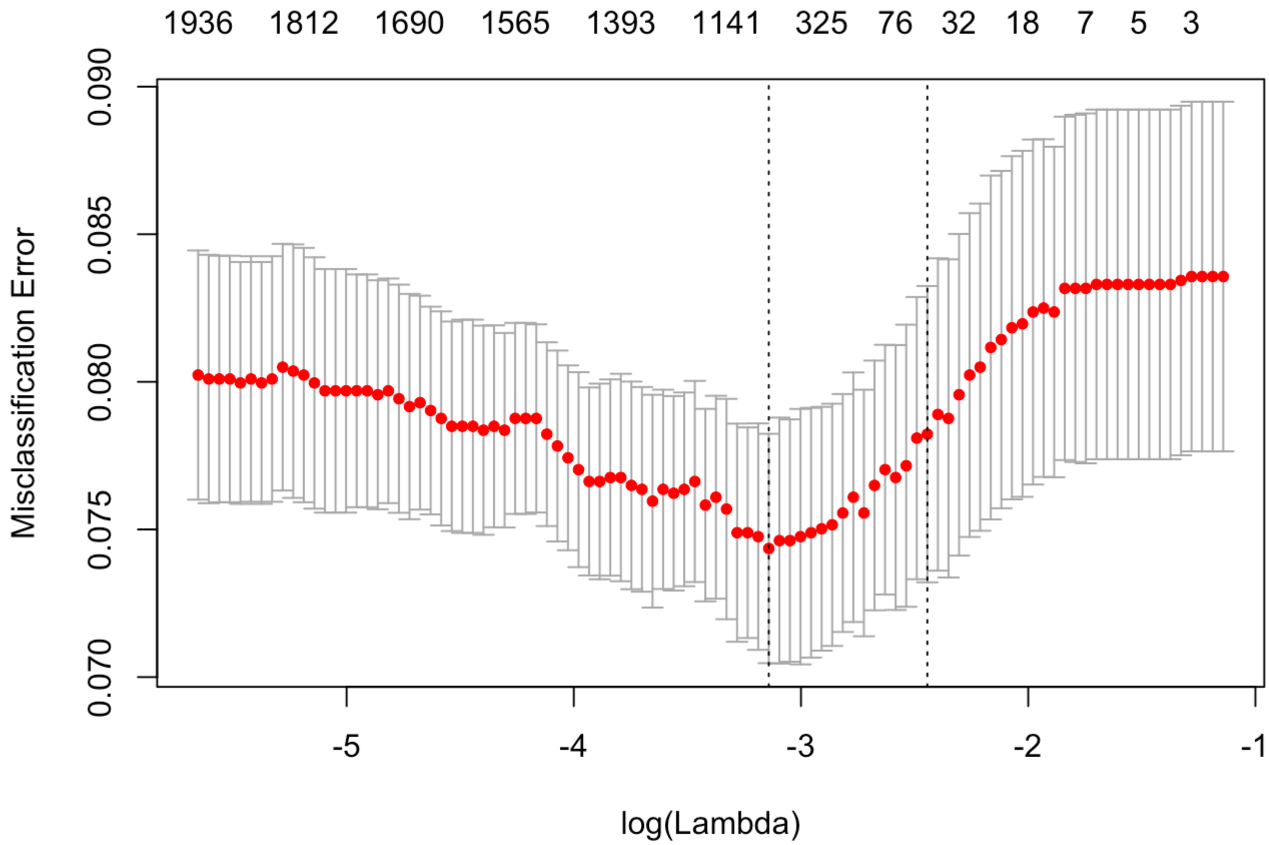

```
plot(glmnet_classifierM)
```



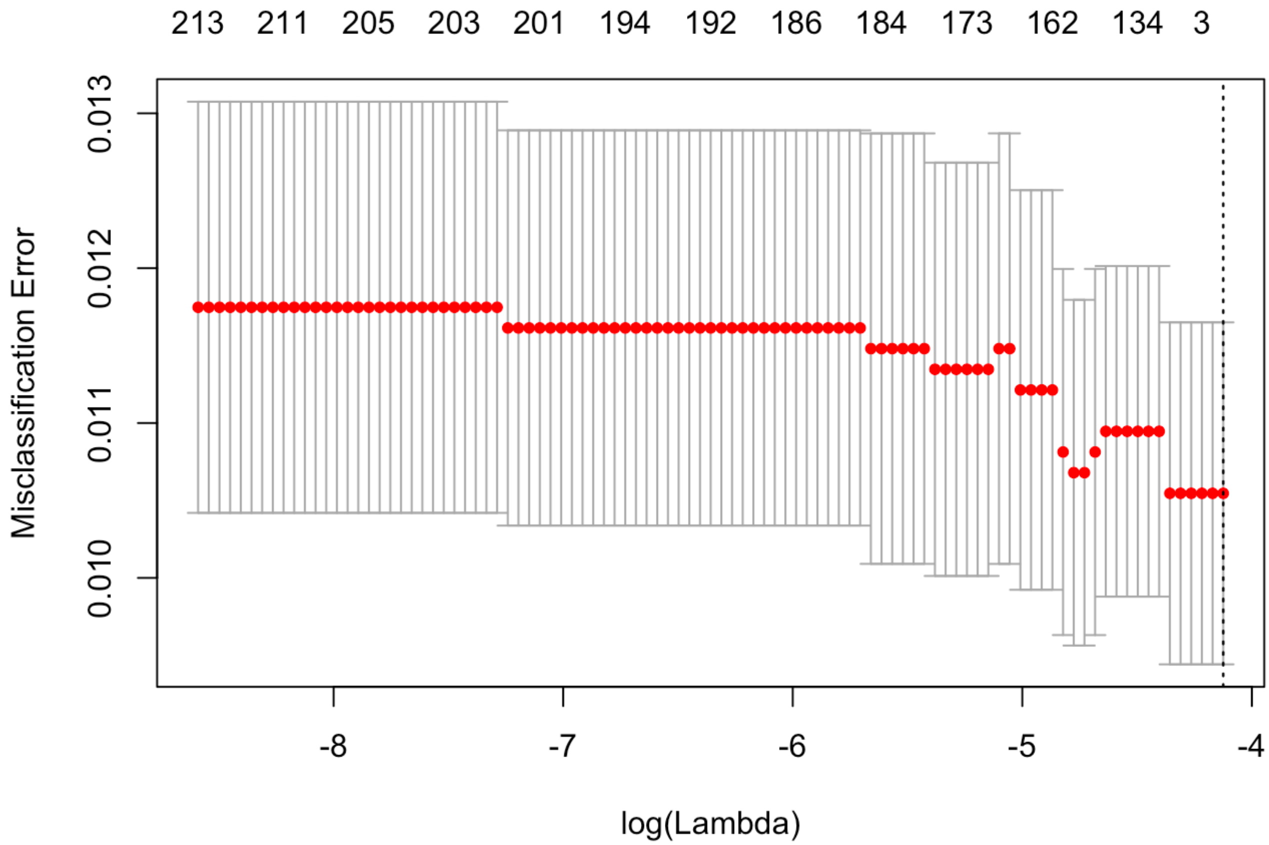

```
plot(glmnet_classifierN)
```



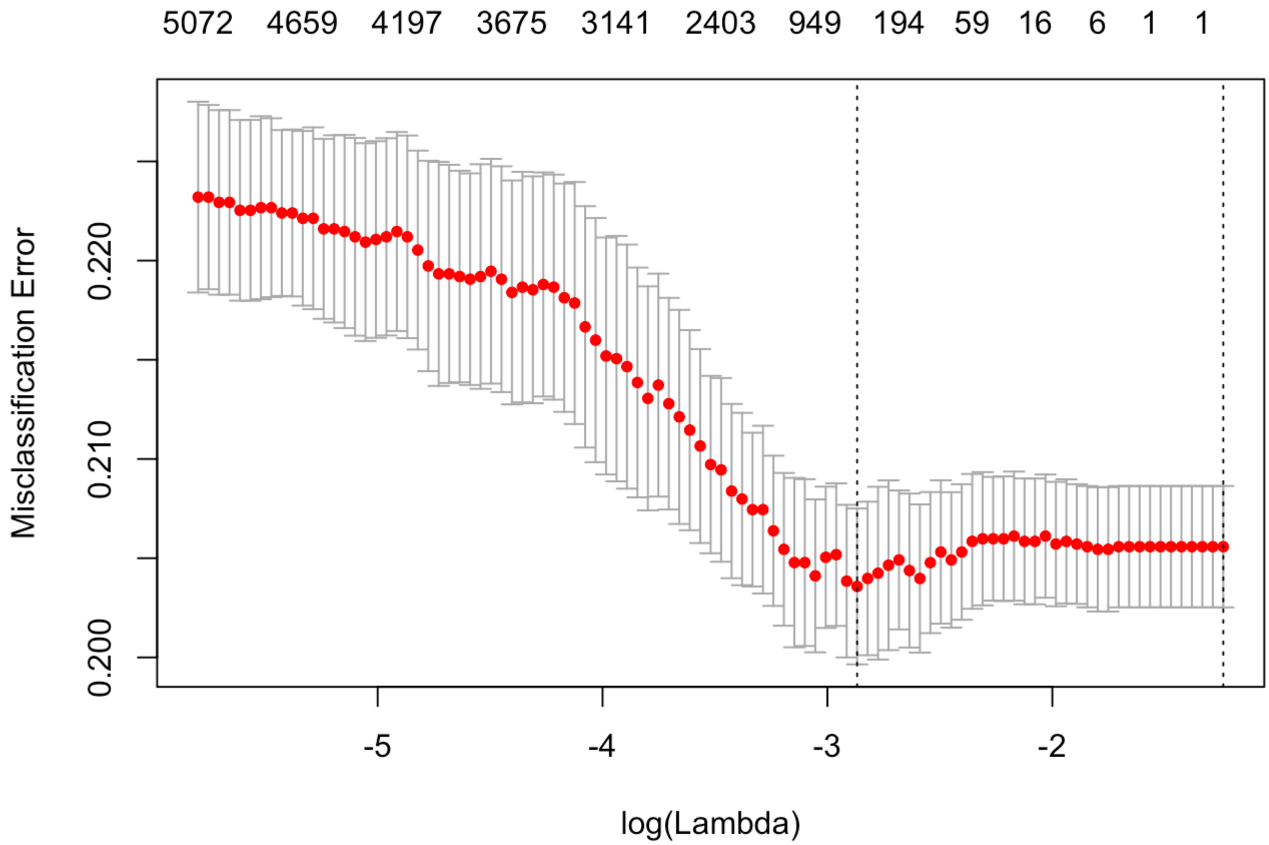

```
plot(glmnet_classifier0)
```



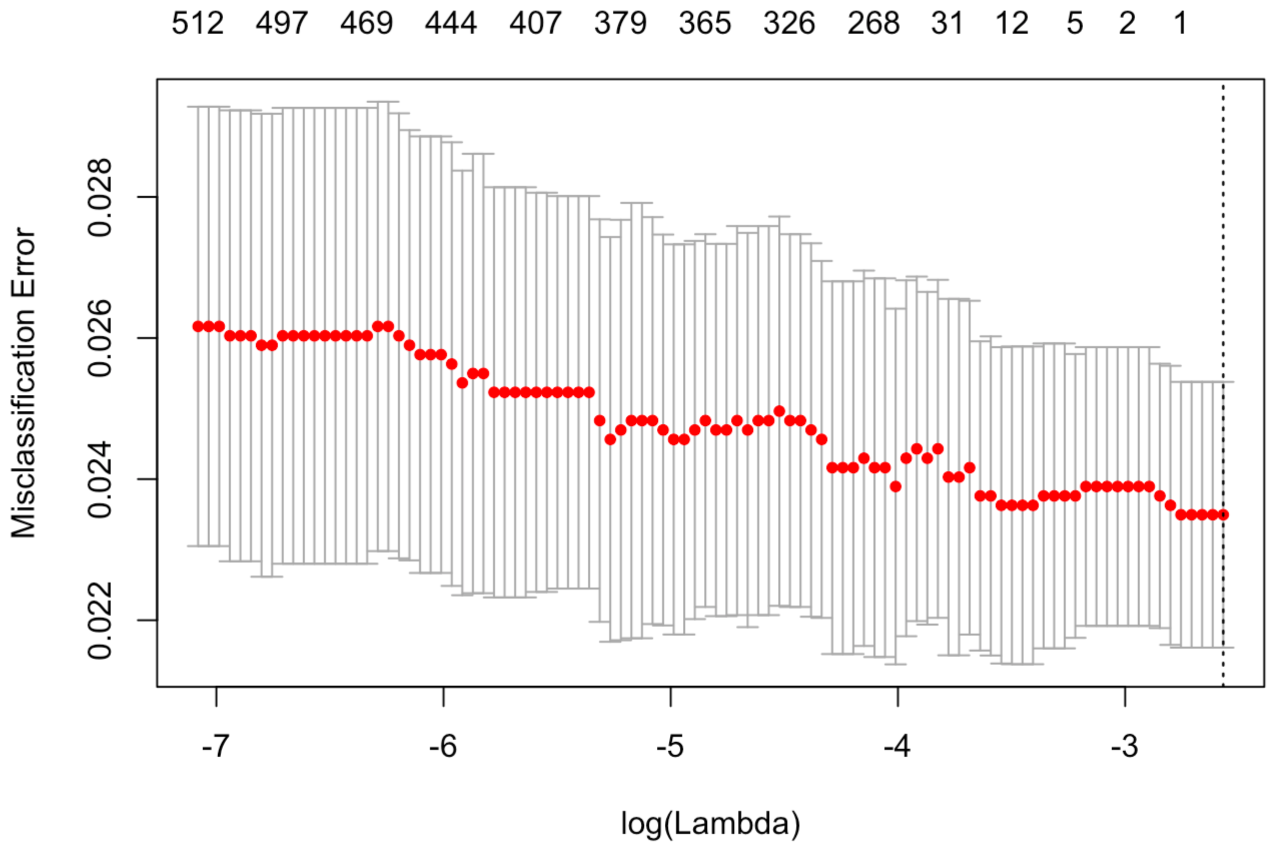

```
plot(glmnet_classifierP)
```



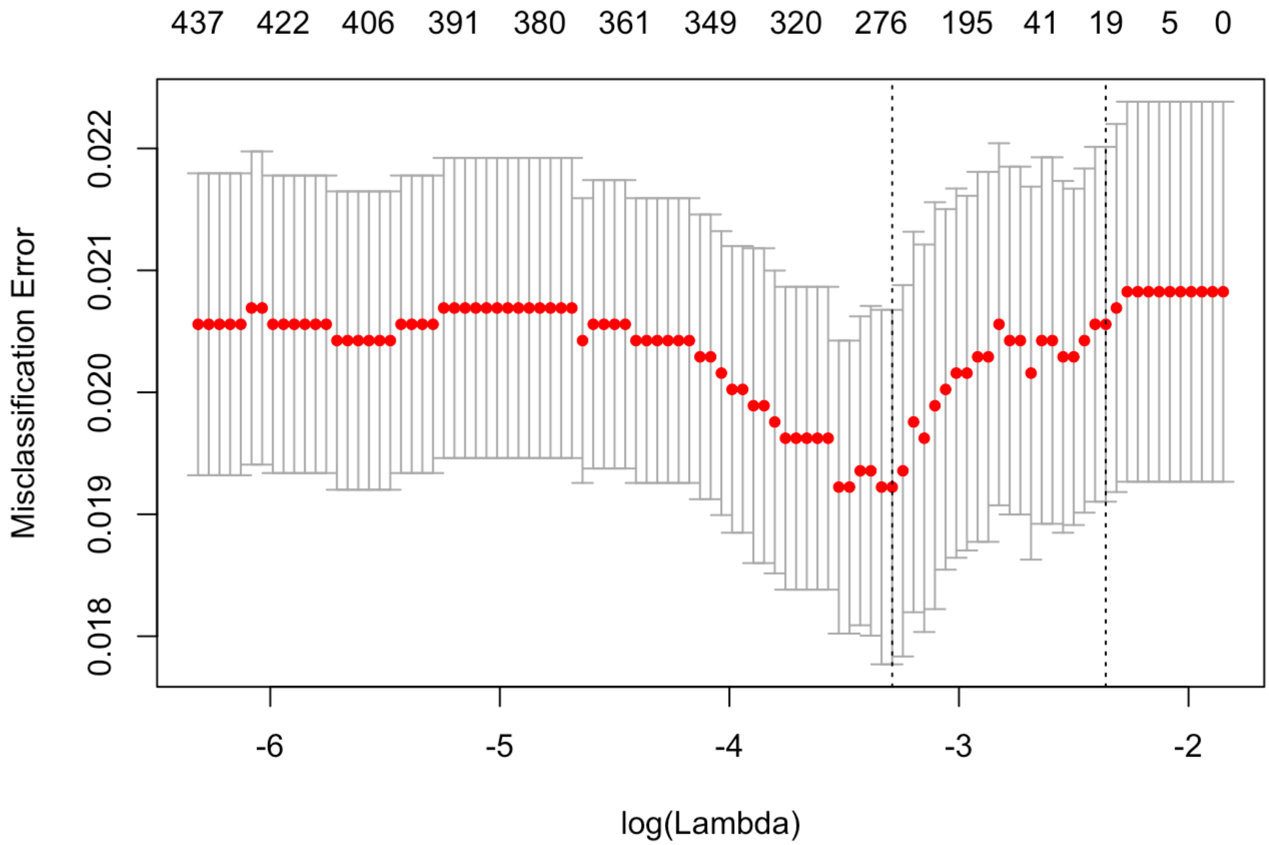

```
plot(glmnet_classifierQ)
```




```
plot(glmnet_classifierR)
```




```
plot(glmnet_classifierS)
```



The misclassification error plots are useful for visualizing the journey of the misclassification error across different values of lambda- this helps to put the lambda min value used for predictions into context, and helps provide a basis for fine tuning the model if needed.

```
#predictions on test DTM
predsA = predict(glmnet_classifierA, newx = dtm_test_tfidf, s = "lambda.min", type = "cl
ass")
predsB = predict(glmnet_classifierB, newx = dtm_test_tfidf, s = "lambda.min", type = "cl
ass")
predsC = predict(glmnet_classifierC, newx = dtm_test_tfidf, s = "lambda.min", type = "cl
ass")

predsD = predict(glmnet_classifierD, newx = dtm_test_tfidf, s = "lambda.min", type = "cl
ass")
predsE = predict(glmnet_classifierE, newx = dtm_test_tfidf, s = "lambda.min", type = "cl
ass")
predsF = predict(glmnet_classifierF, newx = dtm_test_tfidf, s = "lambda.min", type = "cl
ass")

predsG = predict(glmnet_classifierG, newx = dtm_test_tfidf, s = "lambda.min", type = "cl
ass")
predsH = predict(glmnet_classifierH, newx = dtm_test_tfidf, s = "lambda.min", type = "cl
ass")
predsI = predict(glmnet_classifierI, newx = dtm_test_tfidf, s = "lambda.min", type = "cl
ass")

predsJ = predict(glmnet_classifierJ, newx = dtm_test_tfidf, s = "lambda.min", type = "cl
ass")
predsK = predict(glmnet_classifierK, newx = dtm_test_tfidf, s = "lambda.min", type = "cl
ass")
predsL = predict(glmnet_classifierL, newx = dtm_test_tfidf, s = "lambda.min", type = "cl
ass")

predsM = predict(glmnet_classifierM, newx = dtm_test_tfidf, s = "lambda.min", type = "cl
ass")
predsN = predict(glmnet_classifierN, newx = dtm_test_tfidf, s = "lambda.min", type = "cl
ass")
predsO = predict(glmnet_classifierO, newx = dtm_test_tfidf, s = "lambda.min", type = "cl
ass")

predsP = predict(glmnet_classifierP, newx = dtm_test_tfidf, s = "lambda.min", type = "cl
ass")
predsQ = predict(glmnet_classifierQ, newx = dtm_test_tfidf, s = "lambda.min", type = "cl
ass")
predsR = predict(glmnet_classifierR, newx = dtm_test_tfidf, s = "lambda.min", type = "cl
ass")
predsS = predict(glmnet_classifierS, newx = dtm_test_tfidf, s = "lambda.min", type = "cl
ass")
```

Results indicate that some genres are better predicted than others. For example, Drama and Comedy are notably lower in classification accuracy relative to other genres. This is perhaps because of lack of specific overview words that describe these genres, but may also be because of an artifact of the data- depending on how the genres are assigned to different movies, or how overview words are added for movies of these genres. Indeed, in light of the results of the other models in this study, which also show that these two genres are poorly predicted in general.

```
#accuracy
sprintf("Action genre classification accuracy is: %.3f percent", (Accuracy(predsA,
test[,15])*100) )
```

```
## [1] "Action genre classification accuracy is: 82.018 percent"
```

```
sprintf("Adventure genre classification accuracy is: %.3f percent", (Accuracy(predsB, te
st[,16])*100))
```

```
## [1] "Adventure genre classification accuracy is: 89.788 percent"
```

```
sprintf("Animation genre classification accuracy is: %.3f percent", (Accuracy(predsC, te
st[,17])*100))
```

```
## [1] "Animation genre classification accuracy is: 94.834 percent"
```

```
sprintf("Comedy genre classification accuracy is: %.3f percent", (Accuracy(predsD,
test[,18])*100))
```

```
## [1] "Comedy genre classification accuracy is: 71.125 percent"
```

```
sprintf("Crime genre classification accuracy is: %.3f percent", (Accuracy(predsE,
test[,19])*100))
```

```
## [1] "Crime genre classification accuracy is: 89.227 percent"
```

```
sprintf("Documentary genre classification accuracy is: %.3f percent", (Accuracy(predsF,
test[,20])*100))
```

```
## [1] "Documentary genre classification accuracy is: 95.034 percent"
```

```
sprintf("Drama genre classification accuracy is: %.3f percent", (Accuracy(predsG,
test[,21])*100))
```

```
## [1] "Drama genre classification accuracy is: 68.082 percent"
```

```
sprintf("Family genre classification accuracy is: %.3f percent", (Accuracy(predsH,
test[,22])*100))
```

```
## [1] "Family genre classification accuracy is: 90.188 percent"
```

```
sprintf("Fantasy genre classification accuracy is: %.3f percent", (Accuracy(predsI, test[,23])*100))
```

```
## [1] "Fantasy genre classification accuracy is: 93.112 percent"
```

```
sprintf("History genre classification accuracy is: %.3f percent", (Accuracy(predsJ, test[,24])*100))
```

```
## [1] "History genre classification accuracy is: 97.076 percent"
```

```
sprintf("Horror genre classification accuracy is: %.3f percent", (Accuracy(predsK, test[,25])*100))
```

```
## [1] "Horror genre classification accuracy is: 87.545 percent"
```

```
sprintf("Music genre classification accuracy is: %.3f percent", (Accuracy(predsL, test[,26])*100))
```

```
## [1] "Music genre classification accuracy is: 96.035 percent"
```

```
sprintf("Mystery genre classification accuracy is: %.3f percent", (Accuracy(predsM, test[,27])*100))
```

```
## [1] "Mystery genre classification accuracy is: 93.352 percent"
```

```
sprintf("Romance genre classification accuracy is: %.3f percent", (Accuracy(predsN, test[,28])*100))
```

```
## [1] "Romance genre classification accuracy is: 85.583 percent"
```

```
sprintf("Science Fiction genre classification accuracy is: %.3f percent", (Accuracy(pred
sO, test[,29])*100))
```

```
## [1] "Science Fiction genre classification accuracy is: 92.351 percent"
```

```
sprintf("TV Movie genre classification accuracy is: %.3f percent", (Accuracy(predsP, tes
t[,30])*100))
```

```
## [1] "TV Movie genre classification accuracy is: 98.919 percent"
```

```
sprintf("Thriller genre classification accuracy is: %.3f percent", (Accuracy(predsQ, tes
t[,31])*100))
```

```
## [1] "Thriller genre classification accuracy is: 80.737 percent"
```

```
sprintf("War Fiction genre classification accuracy is: %.3f percent", (Accuracy(predsR,
test[,32])*100))
```

```
## [1] "War Fiction genre classification accuracy is: 97.557 percent"
```

```
sprintf("Western Fiction genre classification accuracy is: %.3f percent", (Accuracy(pred
sS, test[,33])*100))
```

```
## [1] "Western Fiction genre classification accuracy is: 97.797 percent"
```

```
confusionMatrix(predsA, test$Action)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1966  409
##          1   40   82
##
##                Accuracy : 0.8202
##                  95% CI : (0.8045, 0.8351)
##     No Information Rate : 0.8034
##     P-Value [Acc > NIR] : 0.01754
##
##                   Kappa : 0.2053
##  Mcnemar's Test P-Value : < 2e-16
##
##             Sensitivity : 0.9801
##             Specificity : 0.1670
##          Pos Pred Value : 0.8278
##          Neg Pred Value : 0.6721
##              Prevalence : 0.8034
##          Detection Rate : 0.7873
##    Detection Prevalence : 0.9511
##       Balanced Accuracy : 0.5735
##
##        'Positive' Class : 0
##
```

```
confusionMatrix(predsB, test$Adventure)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2242  254
##          1    1    0
##
##                Accuracy : 0.8979
##                  95% CI : (0.8853, 0.9095)
##     No Information Rate : 0.8983
##     P-Value [Acc > NIR] : 0.543
##
##                   Kappa : -8e-04
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9996
##             Specificity : 0.0000
##          Pos Pred Value : 0.8982
##          Neg Pred Value : 0.0000
##              Prevalence : 0.8983
##          Detection Rate : 0.8979
##    Detection Prevalence : 0.9996
##       Balanced Accuracy : 0.4998
##
##        'Positive' Class : 0
##
```

```
confusionMatrix(predsC, test$Animation)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2354  127
##          1    2   14
##
##                Accuracy : 0.9483
##                  95% CI : (0.9389, 0.9567)
##     No Information Rate : 0.9435
##     P-Value [Acc > NIR] : 0.1593
##
##                   Kappa : 0.1688
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.99915
##             Specificity : 0.09929
##          Pos Pred Value : 0.94881
##          Neg Pred Value : 0.87500
##              Prevalence : 0.94353
##          Detection Rate : 0.94273
##    Detection Prevalence : 0.99359
##       Balanced Accuracy : 0.54922
##
##        'Positive' Class : 0
##
```

```
confusionMatrix(predsD, test$Comedy)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1440  561
##          1  160  336
##
##                Accuracy : 0.7113
##                  95% CI : (0.693, 0.729)
##     No Information Rate : 0.6408
##     P-Value [Acc > NIR] : 4.982e-14
##
##                   Kappa : 0.3045
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9000
##             Specificity : 0.3746
##          Pos Pred Value : 0.7196
##          Neg Pred Value : 0.6774
##              Prevalence : 0.6408
##          Detection Rate : 0.5767
##    Detection Prevalence : 0.8014
##       Balanced Accuracy : 0.6373
##
##        'Positive' Class : 0
##
```

```
confusionMatrix(predsE, test$Crime)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2206  250
##          1   19   22
##
##                Accuracy : 0.8923
##                  95% CI : (0.8794, 0.9042)
##     No Information Rate : 0.8911
##     P-Value [Acc > NIR] : 0.4394
##
##                   Kappa : 0.1153
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.99146
##             Specificity : 0.08088
##          Pos Pred Value : 0.89821
##          Neg Pred Value : 0.53659
##              Prevalence : 0.89107
##          Detection Rate : 0.88346
##    Detection Prevalence : 0.98358
##       Balanced Accuracy : 0.53617
##
##        'Positive' Class : 0
##
```

```
confusionMatrix(predsF, test$Documentary)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2352  112
##          1   12   21
##
##                Accuracy : 0.9503
##                  95% CI : (0.9411, 0.9585)
##     No Information Rate : 0.9467
##     P-Value [Acc > NIR] : 0.2261
##
##                   Kappa : 0.2369
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9949
##             Specificity : 0.1579
##          Pos Pred Value : 0.9545
##          Neg Pred Value : 0.6364
##              Prevalence : 0.9467
##          Detection Rate : 0.9419
##    Detection Prevalence : 0.9868
##       Balanced Accuracy : 0.5764
##
##        'Positive' Class : 0
##
```

```
confusionMatrix(predsG, test$Drama)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1135  533
##          1  264  565
##
##                Accuracy : 0.6808
##                  95% CI : (0.6621, 0.6991)
##     No Information Rate : 0.5603
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3347
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8113
##             Specificity : 0.5146
##          Pos Pred Value : 0.6805
##          Neg Pred Value : 0.6815
##              Prevalence : 0.5603
##          Detection Rate : 0.4545
##    Detection Prevalence : 0.6680
##       Balanced Accuracy : 0.6629
##
##        'Positive' Class : 0
##
```

```
confusionMatrix(predsH, test$Family)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2232  232
##          1   13   20
##
##                Accuracy : 0.9019
##                  95% CI : (0.8895, 0.9133)
##     No Information Rate : 0.8991
##     P-Value [Acc > NIR] : 0.3356
##
##                   Kappa : 0.1198
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.99421
##             Specificity : 0.07937
##          Pos Pred Value : 0.90584
##          Neg Pred Value : 0.60606
##              Prevalence : 0.89908
##          Detection Rate : 0.89387
##    Detection Prevalence : 0.98678
##       Balanced Accuracy : 0.53679
##
##        'Positive' Class : 0
##
```

```
confusionMatrix(predsI, test$Fantasy)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    0    1
##          0 2321  169
##          1    3    4
## 
##                Accuracy : 0.9311          
##                  95% CI : (0.9205, 0.9407)
##     No Information Rate : 0.9307          
##     P-Value [Acc > NIR] : 0.4888          
## 
##                   Kappa : 0.0393          
##  Mcnemar's Test P-Value : <2e-16          
## 
##             Sensitivity : 0.99871         
##             Specificity : 0.02312         
##          Pos Pred Value : 0.93213         
##          Neg Pred Value : 0.57143         
##              Prevalence : 0.93072         
##          Detection Rate : 0.92952         
##    Detection Prevalence : 0.99720         
##       Balanced Accuracy : 0.51092         
## 
##        'Positive' Class : 0               
## 
```

```
confusionMatrix(predsJ, test$History)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2424   73
##          1    0    0
##
##                Accuracy : 0.9708
##                  95% CI : (0.9634, 0.977)
##     No Information Rate : 0.9708
##     P-Value [Acc > NIR] : 0.5311
##
##                   Kappa : 0
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.9708
##          Neg Pred Value :    NaN
##              Prevalence : 0.9708
##          Detection Rate : 0.9708
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : 0
##
```

```
confusionMatrix(predsK, test$Horror)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2056  263
##          1   48  130
##
##                Accuracy : 0.8755
##                  95% CI : (0.8619, 0.8882)
##     No Information Rate : 0.8426
##     P-Value [Acc > NIR] : 2.001e-06
##
##                   Kappa : 0.3961
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9772
##             Specificity : 0.3308
##          Pos Pred Value : 0.8866
##          Neg Pred Value : 0.7303
##              Prevalence : 0.8426
##          Detection Rate : 0.8234
##    Detection Prevalence : 0.9287
##       Balanced Accuracy : 0.6540
##
##        'Positive' Class : 0
##
```

```
confusionMatrix(predsL, test$Music)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    0    1
##          0 2385   89
##          1   10   13
##                                           
##                Accuracy : 0.9604          
##                  95% CI : (0.9519, 0.9677)
##     No Information Rate : 0.9592          
##     P-Value [Acc > NIR] : 0.4059          
##                                           
##                   Kappa : 0.1959          
##  Mcnemar's Test P-Value : 4.531e-15       
##                                           
##             Sensitivity : 0.9958          
##             Specificity : 0.1275          
##          Pos Pred Value : 0.9640          
##          Neg Pred Value : 0.5652          
##              Prevalence : 0.9592          
##          Detection Rate : 0.9551          
##    Detection Prevalence : 0.9908          
##       Balanced Accuracy : 0.5616          
##                                           
##        'Positive' Class : 0               
## 
```

```
confusionMatrix(predsM, test$Mystery)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2331  166
##          1    0    0
##
##                Accuracy : 0.9335
##                  95% CI : (0.923, 0.943)
##     No Information Rate : 0.9335
##     P-Value [Acc > NIR] : 0.5206
##
##                   Kappa : 0
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.9335
##          Neg Pred Value :    NaN
##              Prevalence : 0.9335
##          Detection Rate : 0.9335
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : 0
##
```

```
confusionMatrix(predsN, test$Romance)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2092  330
##          1   30   45
##
##                Accuracy : 0.8558
##                  95% CI : (0.8414, 0.8694)
##     No Information Rate : 0.8498
##     P-Value [Acc > NIR] : 0.209
##
##                   Kappa : 0.1578
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9859
##             Specificity : 0.1200
##          Pos Pred Value : 0.8637
##          Neg Pred Value : 0.6000
##              Prevalence : 0.8498
##          Detection Rate : 0.8378
##    Detection Prevalence : 0.9700
##       Balanced Accuracy : 0.5529
##
##        'Positive' Class : 0
##
```

```
confusionMatrix(predsO, test$Science.Fiction)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2268  186
##          1    5   38
##
##                Accuracy : 0.9235
##                  95% CI : (0.9124, 0.9336)
##     No Information Rate : 0.9103
##     P-Value [Acc > NIR] : 0.0102
##
##                   Kappa : 0.2634
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9978
##             Specificity : 0.1696
##          Pos Pred Value : 0.9242
##          Neg Pred Value : 0.8837
##              Prevalence : 0.9103
##          Detection Rate : 0.9083
##    Detection Prevalence : 0.9828
##       Balanced Accuracy : 0.5837
##
##        'Positive' Class : 0
##
```

```
confusionMatrix(predsP, test$TV.Movie)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2470   27
##          1    0    0
##
##                Accuracy : 0.9892
##                  95% CI : (0.9843, 0.9929)
##     No Information Rate : 0.9892
##     P-Value [Acc > NIR] : 0.5509
##
##                   Kappa : 0
##  Mcnemar's Test P-Value : 5.624e-07
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.9892
##          Neg Pred Value :    NaN
##              Prevalence : 0.9892
##          Detection Rate : 0.9892
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : 0
##
```

```
confusionMatrix(predsQ, test$Thriller)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1969  469
##          1   12   47
##
##                Accuracy : 0.8074
##                  95% CI : (0.7913, 0.8227)
##     No Information Rate : 0.7934
##     P-Value [Acc > NIR] : 0.0432
##
##                   Kappa : 0.1264
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.99394
##             Specificity : 0.09109
##          Pos Pred Value : 0.80763
##          Neg Pred Value : 0.79661
##              Prevalence : 0.79335
##          Detection Rate : 0.78855
##    Detection Prevalence : 0.97637
##       Balanced Accuracy : 0.54251
##
##        'Positive' Class : 0
##
```

```
confusionMatrix(predsR, test$War)
```

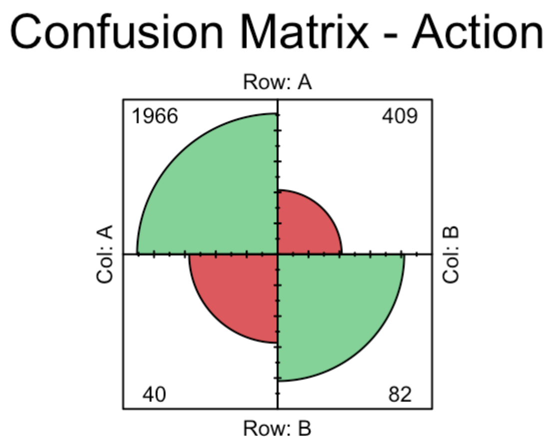
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2436   61
##          1    0    0
##
##                Accuracy : 0.9756
##                  95% CI : (0.9687, 0.9813)
##     No Information Rate : 0.9756
##     P-Value [Acc > NIR] : 0.534
##
##                   Kappa : 0
##  Mcnemar's Test P-Value : 1.564e-14
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.9756
##          Neg Pred Value :    NaN
##              Prevalence : 0.9756
##          Detection Rate : 0.9756
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : 0
##
```

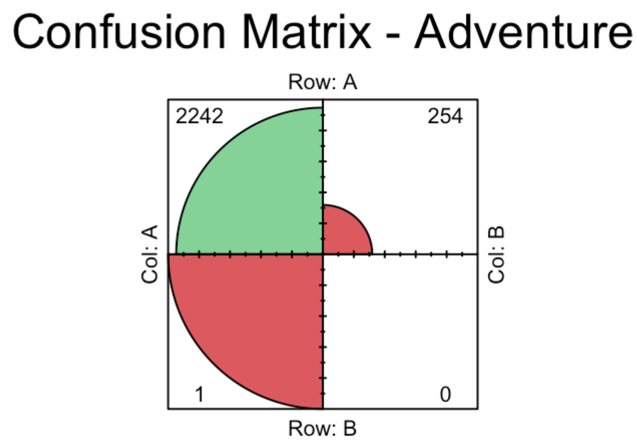
```
confusionMatrix(predsS, test$Western)
```

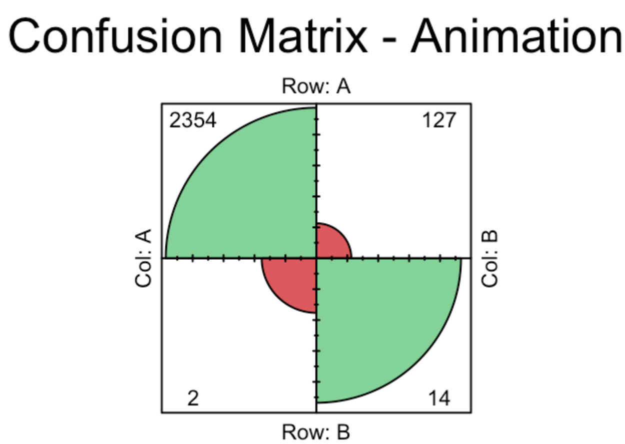
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2439   48
##          1    7    3
##
##                Accuracy : 0.978
##                  95% CI : (0.9714, 0.9834)
##     No Information Rate : 0.9796
##     P-Value [Acc > NIR] : 0.7422
##
##                   Kappa : 0.0923
##  Mcnemar's Test P-Value : 6.906e-08
##
##             Sensitivity : 0.99714
##             Specificity : 0.05882
##          Pos Pred Value : 0.98070
##          Neg Pred Value : 0.30000
##              Prevalence : 0.97958
##          Detection Rate : 0.97677
##    Detection Prevalence : 0.99600
##       Balanced Accuracy : 0.52798
##
##        'Positive' Class : 0
##
```

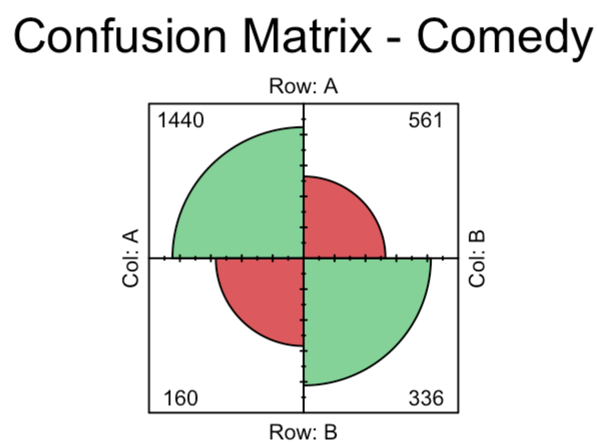
Confusion matrices were visualized to graphically see the proportions of false positives and false negatives. Overall, the confusion matrices “dot plots” show that the green bubbles (true positives and true negative) are relatively larger than the red bubbles (false positives and negatives), indicating that using the overview words to predict movie genres might be a feasible and fruitful approach.

```
#Note: I used directions from here to construct a four dot plot: https://stackoverflow.c
om/questions/23891140/r-how-to-visualize-confusion-matrix-using-the-caret-package

par(mfrow=c(2,2))
ctable <- as.table(matrix(c(1966, 409, 40, 82), nrow = 2, byrow = TRUE))
fourfoldplot(ctable, color = c("#CC6666", "#99CC99"),
             conf.level = 0, margin = 1, main = "Confusion Matrix - Action")
ctable2 <- as.table(matrix(c(2242, 254, 1, 0), nrow = 2, byrow = TRUE))
fourfoldplot(ctable2, color = c("#CC6666", "#99CC99"),
             conf.level = 0, margin = 1, main = "Confusion Matrix - Adventure")
ctable3 <- as.table(matrix(c(2354, 127, 2, 14), nrow = 2, byrow = TRUE))
fourfoldplot(ctable3, color = c("#CC6666", "#99CC99"),
             conf.level = 0, margin = 1, main = "Confusion Matrix - Animation")
ctable4 <- as.table(matrix(c(1440, 561, 160, 336), nrow = 2, byrow = TRUE))
fourfoldplot(ctable4, color = c("#CC6666", "#99CC99"),
             conf.level = 0, margin = 1, main = "Confusion Matrix - Comedy")
```

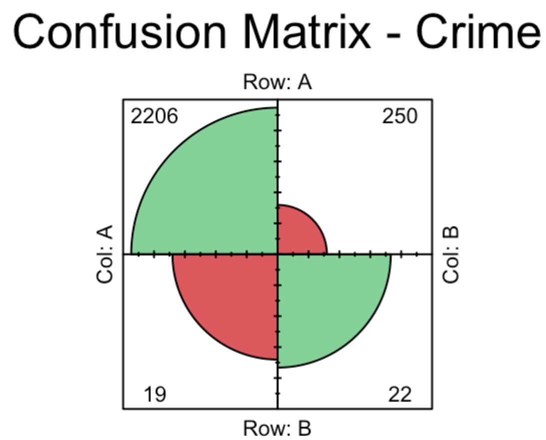


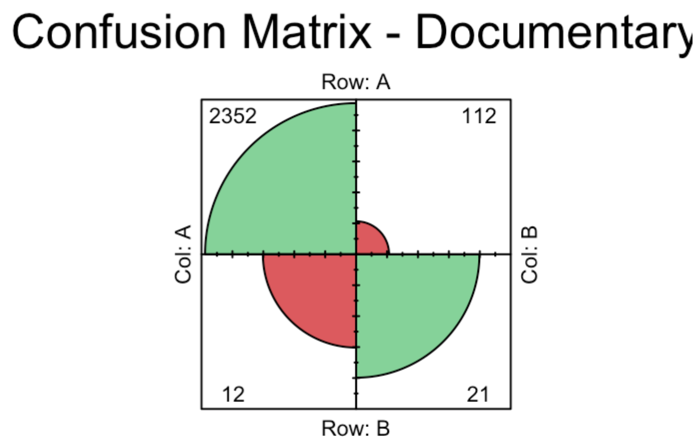


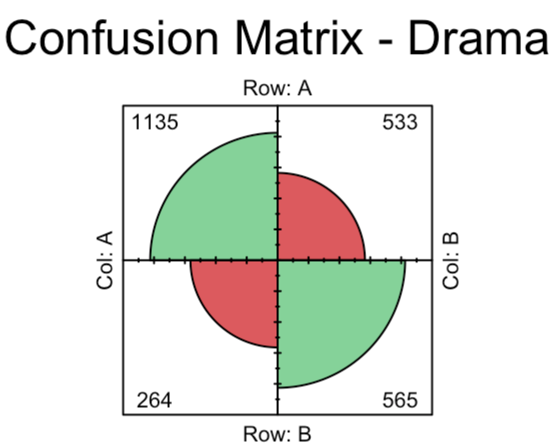


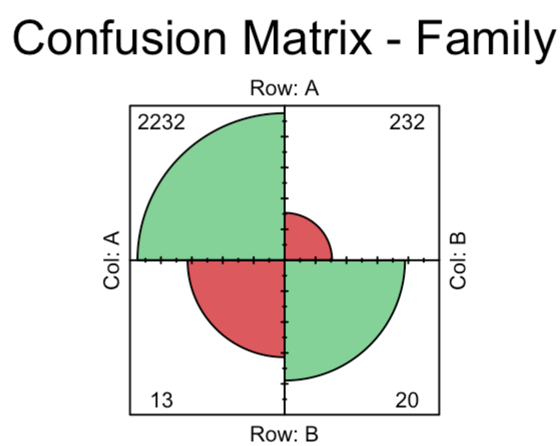



```
ctable5 <- as.table(matrix(c(2206, 250, 19, 22), nrow = 2, byrow = TRUE))
fourfoldplot(ctable5, color = c("#CC6666", "#99CC99"),
             conf.level = 0, margin = 1, main = "Confusion Matrix - Crime")
ctable6 <- as.table(matrix(c(2352, 112, 12, 21), nrow = 2, byrow = TRUE))
fourfoldplot(ctable6, color = c("#CC6666", "#99CC99"),
             conf.level = 0, margin = 1, main = "Confusion Matrix - Documentary")
ctable7 <- as.table(matrix(c(1135, 533, 264, 565), nrow = 2, byrow = TRUE))
fourfoldplot(ctable7, color = c("#CC6666", "#99CC99"),
             conf.level = 0, margin = 1, main = "Confusion Matrix - Drama")
ctable8 <- as.table(matrix(c(2232, 232, 13, 20), nrow = 2, byrow = TRUE))
fourfoldplot(ctable8, color = c("#CC6666", "#99CC99"),
             conf.level = 0, margin = 1, main = "Confusion Matrix - Family")
```

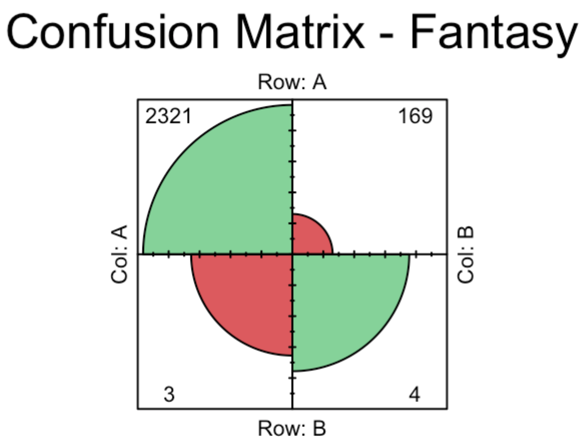


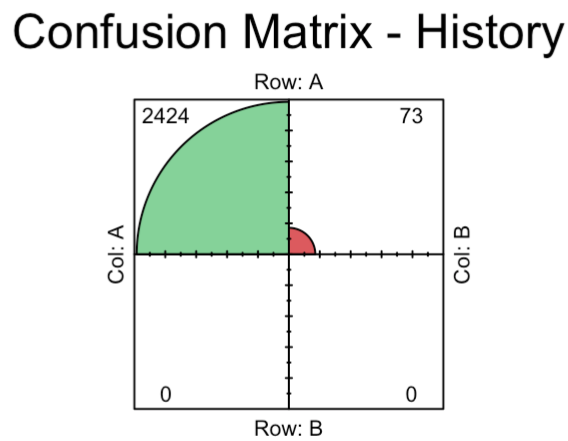


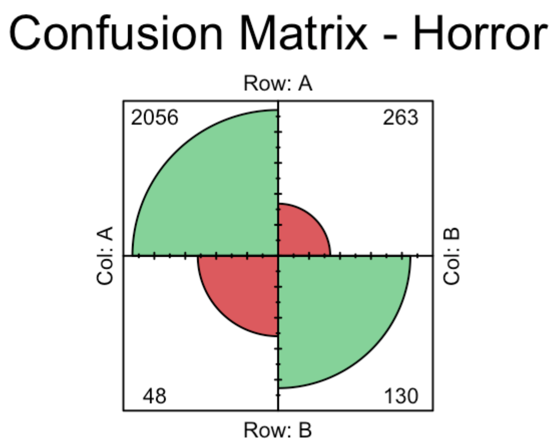


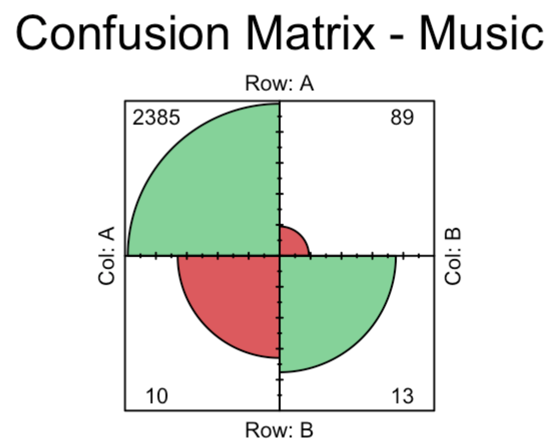



```
ctable9 <- as.table(matrix(c(2321, 169, 3, 4), nrow = 2, byrow = TRUE))
fourfoldplot(ctable9, color = c("#CC6666", "#99CC99"),
             conf.level = 0, margin = 1, main = "Confusion Matrix - Fantasy")
ctable10 <- as.table(matrix(c(2424, 73, 0, 0), nrow = 2, byrow = TRUE))
fourfoldplot(ctable10, color = c("#CC6666", "#99CC99"),
             conf.level = 0, margin = 1, main = "Confusion Matrix - History")
ctable11 <- as.table(matrix(c(2056, 263, 48, 130), nrow = 2, byrow = TRUE))
fourfoldplot(ctable11, color = c("#CC6666", "#99CC99"),
             conf.level = 0, margin = 1, main = "Confusion Matrix - Horror")
ctable12 <- as.table(matrix(c(2385, 89, 10, 13), nrow = 2, byrow = TRUE))
fourfoldplot(ctable12, color = c("#CC6666", "#99CC99"),
             conf.level = 0, margin = 1, main = "Confusion Matrix - Music")
```

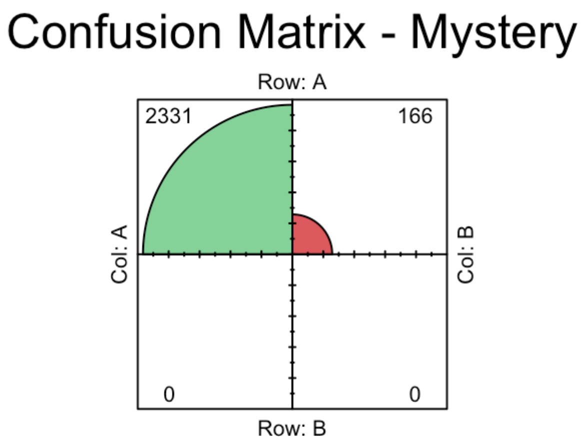


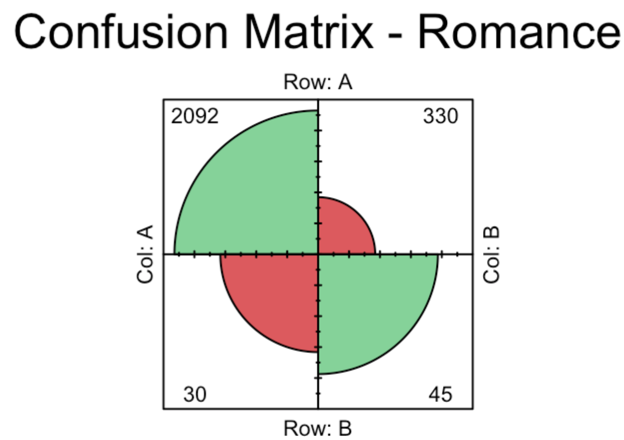


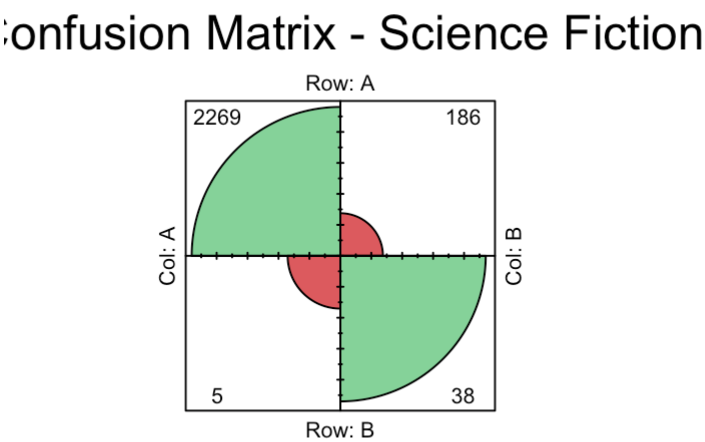


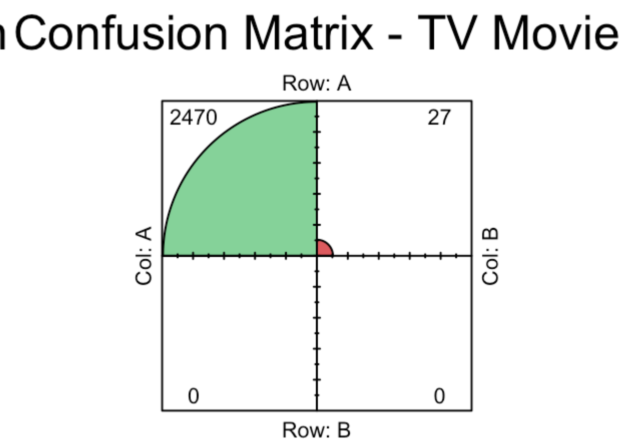



```
ctable13 <- as.table(matrix(c(2331, 166, 0, 0), nrow = 2, byrow = TRUE))
fourfoldplot(ctable13, color = c("#CC6666", "#99CC99"),
             conf.level = 0, margin = 1, main = "Confusion Matrix - Mystery")
ctable14 <- as.table(matrix(c(2092, 330, 30, 45), nrow = 2, byrow = TRUE))
fourfoldplot(ctable14, color = c("#CC6666", "#99CC99"),
             conf.level = 0, margin = 1, main = "Confusion Matrix - Romance")
ctable15 <- as.table(matrix(c(2269, 186, 5, 38), nrow = 2, byrow = TRUE))
fourfoldplot(ctable15, color = c("#CC6666", "#99CC99"),
             conf.level = 0, margin = 1, main = "Confusion Matrix - Science Fiction")
ctable16 <- as.table(matrix(c(2470, 27, 0, 0), nrow = 2, byrow = TRUE))
fourfoldplot(ctable16, color = c("#CC6666", "#99CC99"),
             conf.level = 0, margin = 1, main = "Confusion Matrix - TV Movie")
```



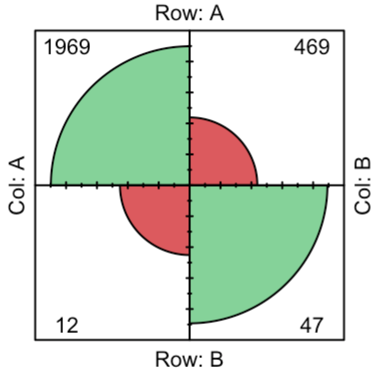




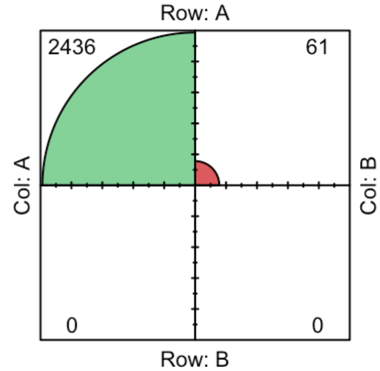



```
ctable17 <- as.table(matrix(c(1969, 469, 12, 47), nrow = 2, byrow = TRUE))
fourfoldplot(ctable17, color = c("#CC6666", "#99CC99"),
             conf.level = 0, margin = 1, main = "Confusion Matrix - Thriller")
ctable18 <- as.table(matrix(c(2436, 61, 0, 0), nrow = 2, byrow = TRUE))
fourfoldplot(ctable18, color = c("#CC6666", "#99CC99"),
             conf.level = 0, margin = 1, main = "Confusion Matrix - War")
ctable19 <- as.table(matrix(c(2439, 48, 7, 3), nrow = 2, byrow = TRUE))
fourfoldplot(ctable19, color = c("#CC6666", "#99CC99"),
             conf.level = 0, margin = 1, main = "Confusion Matrix - Western")
```

Confusion Matrix - Thriller



Confusion Matrix - War



Confusion Matrix - Western