Classification, Regularization, NLP

Jack Sampiere

```
library(caTools)
library(ROCR)
library(tm)
library(SnowballC)
library(glmnet)
library(glmnetUtils)
library(rpart)
library(rpart.plot)
library(randomForest)
```

QUESTION 1: Parole Violations

```
# read data
parole = read.csv('parole.csv')
```

Part 1: Loading the data

```
print(nrow(parole))
```

```
## [1] 675
```

```
print(nrow(parole[parole$violator == 1,]))
```

```
## [1] 78
```

a) There are 675 parolees in the dataset.

b) 78 of the parolees violated their parole.

c) The variables that are factors are race, state, and crime.

Part 2: Building the model

```
# convert factor variables
parole$race = as.factor(parole$race)
parole$state = as.factor(parole$state)
parole$crime = as.factor(parole$crime)

set.seed(144)
# perform split
spl = sample.split(parole$violator, SplitRatio=0.7)
# split the data
parole.train = subset(parole, spl == TRUE)
parole.test = subset(parole, spl == FALSE)
# estimate logistic regression with training data
parole.glm = glm(violator ~ ., data=parole.train, family='binomial')
# examine coefficients
summary(parole.glm)
```

```
##
## Call:
## glm(formula = violator ~ ., family = "binomial", data = parole.train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.7041  -0.4236  -0.2719  -0.1690   2.8375
##
## Coefficients:
##                     Estimate Std. Error z value Pr(>|z|)
## (Intercept)       -3.3544382  1.2374487  -2.711  0.00671 **
## male               0.3869904  0.4379613   0.884  0.37690
## race2              0.8867192  0.3950660   2.244  0.02480 *
## age               -0.0001756  0.0160852  -0.011  0.99129
## state2             0.4433007  0.4816619   0.920  0.35739
## state3             0.8349797  0.5562704   1.501  0.13335
## state4            -3.3967878  0.6115860  -5.554 2.79e-08 ***
## time.served       -0.1238867  0.1204230  -1.029  0.30359
## max.sentence       0.0802954  0.0553747   1.450  0.14705
## multiple.offenses  1.6119919  0.3853050   4.184 2.87e-05 ***
## crime2             0.6837143  0.5003550   1.366  0.17180
## crime3            -0.2781054  0.4328356  -0.643  0.52054
## crime4            -0.0117627  0.5713035  -0.021  0.98357
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 340.04  on 472  degrees of freedom
## Residual deviance: 251.48  on 460  degrees of freedom
## AIC: 277.48
##
## Number of Fisher Scoring iterations: 6
```

a) race2, state4, and multiple.offenses are all significant at the 0.05 level.

b) Based on the coefficient on multiple.offenses, we can say that the log of the odds that a parolee has of being a violator is 1.61 higher for those that have committed multiple offenses relative to those that have not.

c) Considering someone that is male (male = 1), white (race1), 50 years old (age = 50), from MD (state1), served 3 months (time.served = 3), had a maximum sentence of 12 months (max.sentence = 12), did not commit multiple sentences (multiple.offenses = 0), and committed a larceny (crime2):

```
# pack characteristics of the individual into a vector
individual = c(1,1,0,50,0,0,0,3,12,0,1,0,0)
# extract coefficients from model
coeffs = coef(parole.glm)
# take dot product
logit = individual%*%coeffs
print(paste0('Logit: ', logit))
```

```
## [1] "Logit: -1.70062980028895"
```

```
# exponentiate logit to get odds
odds = exp(logit)
print(paste0('Odds: ', odds))
```

```
## [1] "Odds: 0.182568506139459"
```

```
# find probability
prob = 1 / (1 + exp(-1*logit))
print(paste0('Probability of violation: ', prob))
```

```
## [1] "Probability of violation: 0.154383027445455"
```

Part 3: Out-of-sample prediction

```
# predict on the test set
parole.predict = predict(parole.glm, newdata=parole.test, type='response')
# display maximum predicted probability
print(paste0('Max predicted probability: ', max(parole.predict)))
```

```
## [1] "Max predicted probability: 0.907279069042028"
```

```
# generate confusion matrix
conf.mat = table(parole.test$violator, parole.predict > 0.5)
# pull values from confusion matrix
TN = conf.mat[1,1]
TP = conf.mat[2,2]
FN = conf.mat[2,1]
FP = conf.mat[1,2]
# compute sensitivity, specificity, accuracy, and baseline accuracy
sens = TP / (TP + FN)
print(paste0('Sensitivity: ', sens))
```

```
## [1] "Sensitivity: 0.521739130434783"
```

```
spec = TN / (TN + FP)
print(paste0('Specifity: ', spec))
```

```
## [1] "Specifity: 0.932960893854749"
```

```
acc = sum(diag(conf.mat)) / nrow(parole.test)
print(paste0('Accuracy: ', acc))
```

```
## [1] "Accuracy: 0.886138613861386"
```

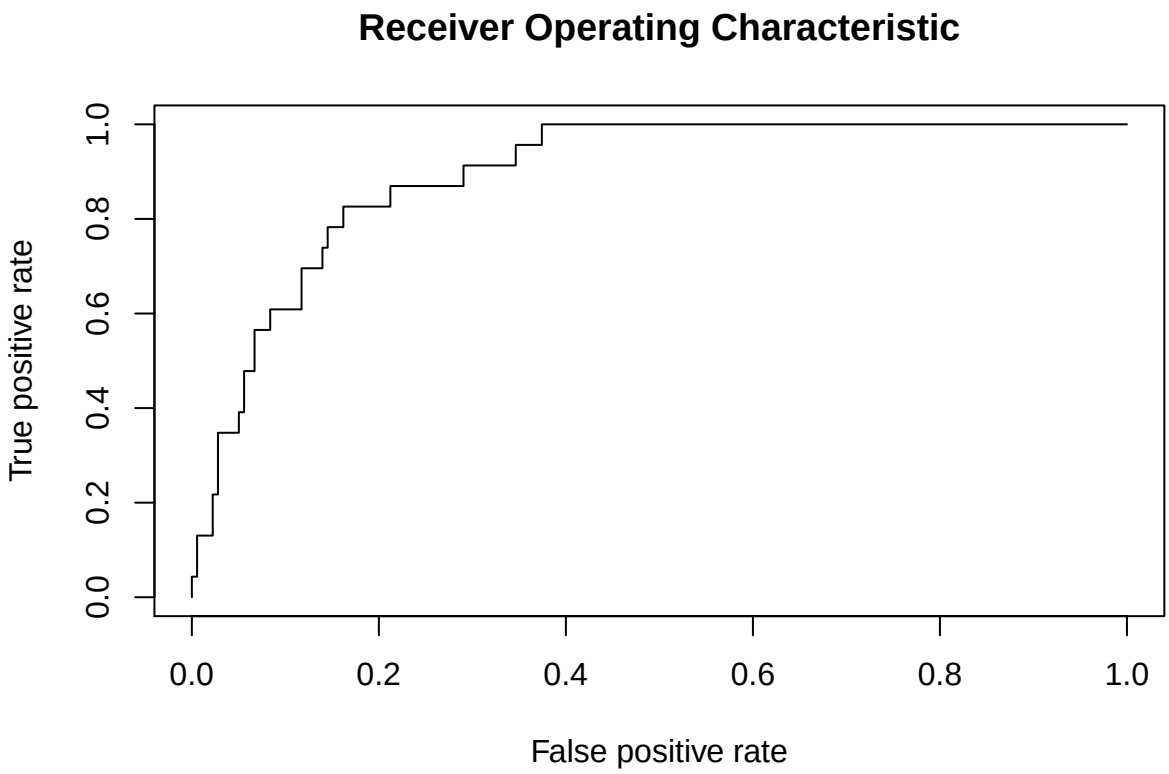
```
#table(parole.test$violator)
baseline.acc = 179 / (179 + 23)
print(paste0('Baseline accuracy: ', baseline.acc))
```

```
## [1] "Baseline accuracy: 0.886138613861386"
```

a-e) Above

Part 4: ROC curve / AUC

```
# plot ROC curve for model from part 2
roc.pred = prediction(parole.predict, parole.test$violator)
roc.perf = performance(roc.pred, 'tpr', 'fpr')
plot(roc.perf, main='Receiver Operating Characteristic')
```




```
# compute AUC
auc = as.numeric(performance(roc.pred, 'auc')@y.values)
print(paste0('AUC: ', auc))
```

```
## [1] "AUC: 0.894583434539713"
```

a-b) Above

c) The AUC of 0.8946 in part b means that, given 2 parolees–one random violator and one random non-violator–our model would correctly tell us which is which with a probability of 0.8946.

Part 5: Operationalizing the model

a) The parole board could make decisions about who to grant parole with my model by selecting all the individuals with a probability of violation below some threshold. This is a feasible option since they could set the threshold very low to reduce the number of false negatives since these errors are likely more problematic. They could also select some number of individuals with the lowest probability of violation.

```
# determine the 150 individuals least likely to violate parole
parole.predict.sorted = sort(parole.predict)[1:150]
# get indices of these individuals
indices.grant = rownames(data.frame(parole.predict.sorted))
# subset parole to contain only the 150 to whom we'll grant parole
parole.grant = parole[indices.grant,]
# determine how many of these individuals actually violated parole
print(sum(parole.grant$violator))
```

```
## [1] 4
```

```
# track number of violations
violations = 0
# randomly sample 150 individuals from the test set 100 times, record violations
for (i in 1:100)
{
  rand.indices = sample(c(1:nrow(parole.test)), 150, replace = FALSE)
  parole.random.grant = parole.test[rand.indices,]
  violations = violations + sum(parole.random.grant$violator)
}
avg.violations = violations / 100
print(avg.violations)
```

```
## [1] 16.99
```

b) Using the model to choose 150 individuals to whom to grant parole, 4 of these individuals violate parole.

c) Randomly sampling from the test set, roughly 17 of the individuals granted parole end up violating.

d) Comparing the answers from (b) and (c), we conclude that the model is indeed useful. This is because we get much better results by using the model as opposed to randomly deciding to whom to grant parole.

QUESTION 2: Electronic Discovery

```
# read data
energy = read.csv('energy_bids.csv')
```

Part 1: Processing the data

```
# convert emails to a corpus object
corpus = Corpus(VectorSource(energy$email))
# convert to lowercase, remove punctuation and stop words, stem
corpus = tm_map(corpus, tolower)
corpus = tm_map(corpus, removePunctuation)
corpus = tm_map(corpus, removeWords, stopwords('english'))
corpus = tm_map(corpus, stemDocument)
# convert corpus object to a document-term matrix
dtm = DocumentTermMatrix(corpus)
# remove any words that do not appear in at least 3% of emails
dtm.sparse = removeSparseTerms(dtm, 0.97)
# convert to data frame
emails.df = as.data.frame(as.matrix(dtm.sparse))
# make all words valid
names(emails.df) = paste("w_", names(emails.df),  sep ='')
# add dependent variable
emails.df$responsive = energy$responsive

# count number words in final data frame
print(paste0('Number of words: ', ncol(emails.df))) # one of these is the dependent variable
```

```
## [1] "Number of words: 789"
```

```
# find most frequent words (i.e. columns with highest sums)
print('Most frequent words:')
```

```
## [1] "Most frequent words:"
```

```
print(sort(colSums(emails.df[,1:length(emails.df)]),decreasing=TRUE)[1:5])
```

```
##   w_will  w_power w_market  w_enron  w_email
##     1580     1199     1170     1047     1001
```

```
# determine fraction of emails that are responsive
print(paste0('Fraction of emails that are responsive: ', sum(emails.df$responsive) / nrow(emails.df)))
```

```
## [1] "Fraction of emails that are responsive: 0.162573099415205"
```

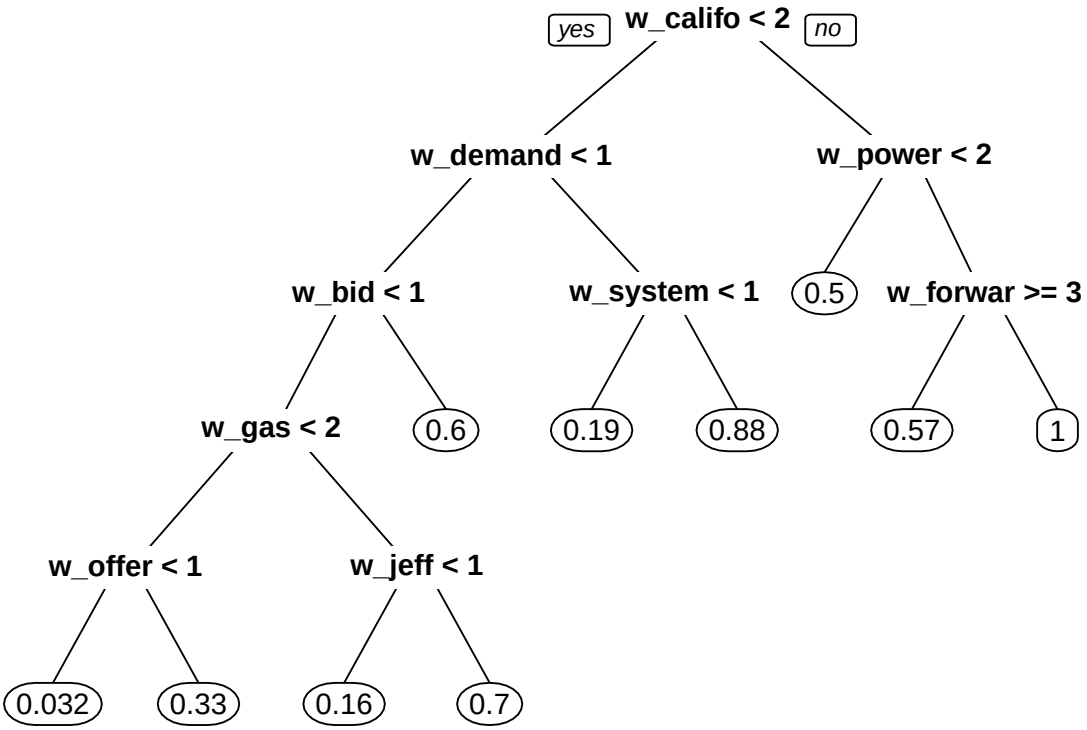
a) There are 788 words in the final data frame.

b) The most frequent words in the final data frame are "will" (1580 occurrences), "power" (1199 occurrences), "market" (1170 occurrences), "enron" (1047 occurrences), and "email" (1001 occurrences).

c) Approximately 16.26% of the emails are responsive.

Part 2: A CART model

```
set.seed(144)

# perform split
spl = sample.split(emails.df$responsive, SplitRatio=0.7)
# split the data
emails.train = emails.df[spl,]
emails.test = emails.df[!spl,]
# fit decision tree
emails.rpart = rpart(responsive ~ ., data=emails.train)
prp(emails.rpart)
```



```
# make out-of-sample predictions
emails.rpart.predict = predict(emails.rpart, newdata=emails.test)
# generate confusion matrix
conf.mat = table(emails.test$responsive, emails.rpart.predict >= 0.5)
# compute accuracy from confusion matrix
acc = sum(diag(conf.mat)) / nrow(emails.test)
print(paste0('Accuracy: ', acc))
```

```
## [1] "Accuracy: 0.856031128404669"
```

```
# compute AUC
pred = prediction(emails.rpart.predict, emails.test$responsive)
auc = as.numeric(performance(pred, 'auc')@y.values)
print(paste0('AUC: ', auc))
```

```
## [1] "AUC: 0.814784053156146"
```

a) The words that appear in my CART are californ, demand, power, bid, system, forward, gas, offer, and jeff.

b) The word that is the first name of an individual is jeff. This represents Jeffrey Skilling, the CEO of Enron at the time.

c) The out-of-sample accuracy is roughly 85.60%.

d) The out-of-sample AUC is roughly 0.8148.

Part 3: A random forest model

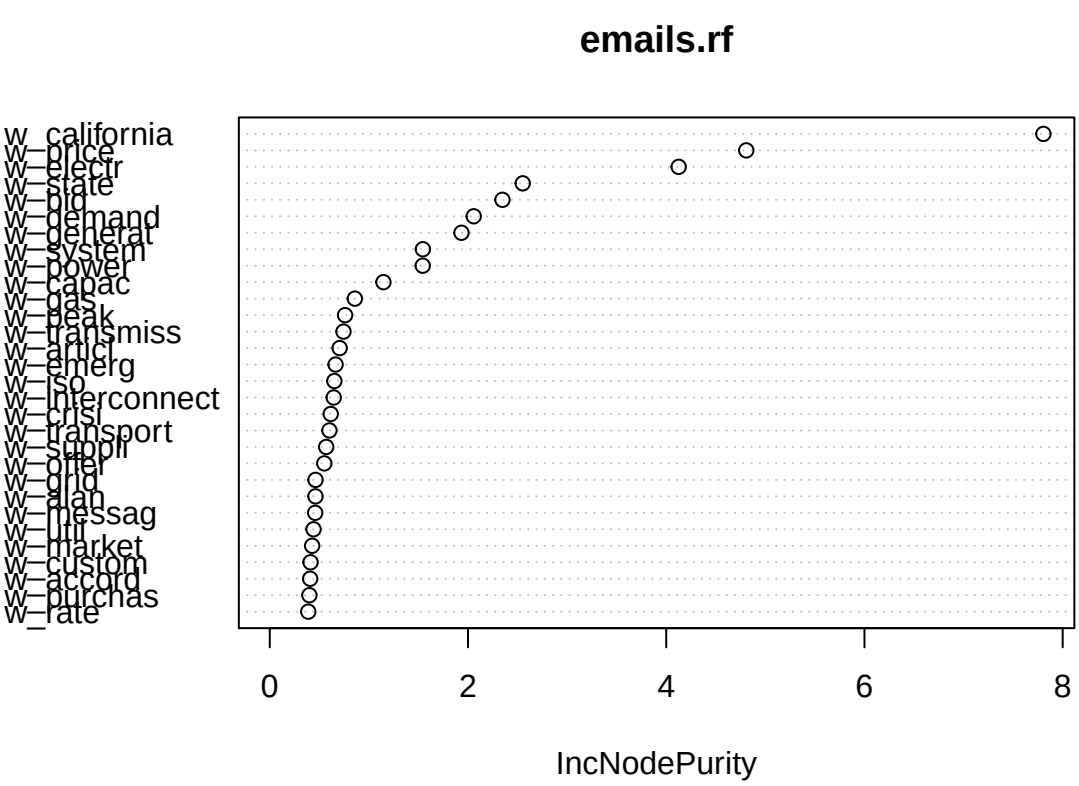
```
set.seed(100)

# fit random forest
emails.rf = randomForest(responsive ~ ., data=emails.train)
# make predictions on test set
emails.rf.predict = predict(emails.rf, newdata=emails.test)
# generate confusion matrix
conf.mat = table(emails.test$responsive, emails.rf.predict >= 0.5)
# compute accuracy from confusion matrix
acc = sum(diag(conf.mat)) / nrow(emails.test)
print(paste0('Accuracy: ', acc))
```

```
## [1] "Accuracy: 0.875486381322957"
```

```
# compute AUC
pred = prediction(emails.rf.predict, emails.test$responsive)
auc = as.numeric(performance(pred, 'auc')@y.values)
print(paste0('AUC: ', auc))
```

```
## [1] "AUC: 0.919822812846069"
```

```
varImpPlot(emails.rf)
```



a) The top 5 most important words in the random forest are californ, price, electr, state, and bid.

b) The out-of-sample accuracy is roughly 0.8755.

c) The out-of-sample AUC is roughly 0.9198.

```
# convert predictions to data frame
rf.pred.df = data.frame(emails.rf.predict)
# round probabilities to get predictions
rf.pred.df$pred = ifelse(emails.rf.predict < 0.5, 0, 1)
# keep only emails that are predicted to be responsive
rf.pred.yes1 = rf.pred.df[rf.pred.df$pred == 1,]
# extract original indices
rf.pred.yes1.ind = rownames(rf.pred.yes1)
# extract those emails from base data
rf.true.vals1 = emails.df[rf.pred.yes1.ind,]
# calculate and display percentage of predicted responsive that are actually responsive
print(sum(rf.true.vals1$responsive) / nrow(rf.pred.yes1))
```

```
## [1] 0.6388889
```

d) With a threshold of 0.5, roughly 63.89% of the documents predicted to be responsive are actually responsive.

```
# extract emails that are used in the predictions
rf.true.vals2 = emails.df[rownames(rf.pred.df),]
# keep only those that are responsive
rf.true.yes2 = rf.true.vals2[rf.true.vals2$responsive == 1,]
# get indices of responsive emails
```

```
rf.true.yes2.ind = rownames(rf.true.yes2)
# keep only those emails in the predictions
rf.check.yes = rf.pred.df[rf.true.yes2.ind,]
# check how many were predicted to be responsive and divide by amount known to be responsive
print(sum(rf.check.yes$pred) / nrow(rf.true.yes2))
```

```
## [1] 0.547619
```

e) With a threshold of 0.5, roughly 54.76% of the documents known to be responsive are predicted to be responsive.

Part 4: A logistic regression model

```
# fit logistic regression
emails.glm = glm(responsive ~ ., data=emails.train, family='binomial')
# predict on training set
emails.glm.predict.train = predict(emails.glm, newdata=emails.train, type='response')
# predict on test set
emails.glm.predict.test = predict(emails.glm, newdata=emails.test, type='response')
# generate confusion matrix for training data
conf.mat.train = table(emails.train$responsive, emails.glm.predict.train >= 0.5)
# generate confusion matrix for test data
conf.mat.test = table(emails.test$responsive, emails.glm.predict.test >= 0.5)
# compute in-sample accuracy
acc.train = sum(diag(conf.mat.train)) / nrow(emails.train)
# compute out-of-sample accuracy
acc.test = sum(diag(conf.mat.test)) / nrow(emails.test)
# compute AUC
pred = prediction(emails.glm.predict.test, emails.test$responsive)
auc = as.numeric(performance(pred, 'auc')@y.values)
# display
print(paste0('In-sample accuracy: ', acc.train))
```

```
## [1] "In-sample accuracy: 0.968227424749164"
```

```
print(paste0('Out-of-sample accuracy: ', acc.test))
```

```
## [1] "Out-of-sample accuracy: 0.544747081712062"
```

```
print(paste0('AUC: ', auc))
```

```
## [1] "AUC: 0.545902547065338"
```

a) The accuracy of the model on the training set is roughly 96.82%.

b) The accuracy of the model on the test set is roughly 54.47%.

c) The AUC of the model on the test set is roughly 0.5459.

d) The answers to (b) and (c) are low because ordinary logistic regression is not well-suited to the current case in which there is a high number of independent variables (788 words) relative to the number of observations (855 emails). The reason why (b) is much lower than (a) is because there is a substantial amount of overfitting occurring in this case.

Part 5: A regularized logistic regression model

```
set.seed(55)
library(coefplot)
# estimate lasso-regularized logistic regression with 5-fold cross-validation
emails.glmnet.cv = cv.glmnet(responsive ~., data=emails.train, family="binomial", nfolds=5)
```

```r
# display optimal lambda
print(emails.glmnet.cv$lambda.min)
```

```
## [1] 0.01981017
```

```r
# determine words included in regularized logistic regression model
print(extract.coef(emails.glmnet.cv)$Coefficient)
```

```
##  [1] "(Intercept)"          "w_establish"          "w_fuel"
##  [4] "w_gas"                "w_natur"              "w_perform"
##  [7] "w_price"              "w_regard"             "w_take"
## [10] "w_david"              "w_june"               "w_messag"
## [13] "w_robert"             "w_author"             "w_demand"
## [16] "w_emerg"              "w_ferc"               "w_investig"
## [19] "w_pipelin"            "w_havent"             "w_enough"
## [22] "w_interest"           "w_mike"               "w_pass"
## [25] "w_solut"              "w_transmiss"          "w_bit"
## [28] "w_articl"             "w_capac"              "w_generat"
## [31] "w_immedi"             "w_implement"          "w_interconnect"
## [34] "w_load"               "w_low"                "w_mark"
## [37] "w_plan"               "w_point"              "w_problem"
## [40] "w_product"            "w_similar"            "w_transport"
## [43] "w_wednesday"          "w_altern"             "w_revis"
## [46] "w_iso"                "w_storag"             "w_andor"
## [49] "w_therefor"           "w_bob"                "w_pay"
## [52] "w_book"               "w_status"             "w_decemb"
## [55] "w_languag"            "w_peak"               "w_steffesnaenronenron"
## [58] "w_payment"            "w_dasovichnaenron"
```

```r
# predict on test set
emails.glmnet.cv.predict = predict(emails.glmnet.cv, newdata=emails.test, type='response', s='lambda.mi
# generate confusion matrix for test data
conf.mat = table(emails.test$responsive, emails.glmnet.cv.predict >= 0.5)
# compute out-of-sample accuracy
acc = sum(diag(conf.mat)) / nrow(emails.test)
# compute AUC
pred = prediction(emails.glmnet.cv.predict, emails.test$responsive)
auc = as.numeric(performance(pred, 'auc')@y.values)
# display
print(paste0('Out-of-sample accuracy: ', acc))
```

```
## [1] "Out-of-sample accuracy: 0.90272373540856"
```

```r
print(paste0('AUC: ', auc))
```

```
## [1] "AUC: 0.899114064230344"
```

a) The optimal cross-validated value of lambda is roughly 0.0198.

b) The words included in the regularized logistic regression model are (see above).

c) The accuracy of the cross-validated model on the test set is roughly 90.27%.

d) The AUC of the cross-validated model on the test set is roughly 0.8991.