Numeric Prediction, Logistic Regression, and Feature Selection

Blake Pappas

2023-12-17

Part I. Numeric Prediction in R

```
library(caret)
library(rpart)
library(rpart.plot)
library(glmnet)
library(dplyr)
```

Import "purchase.csv" into R:

```
purchase = read.csv("purchase.csv")
```

1. Use 5-fold cross-validation to evaluate the performance of the regression tree model on this dataset. Report the average (1) MAE; (2) MAPE; (3) RMSE.

Then, build a single regression tree model on the entire dataset, plot the tree, and answer the following two questions.

(a) How many decision nodes are there in the tree?

(b) Pick one decision rule from the tree and interpret it.

```
# Cross-Validation
cv = createFolds(y = purchase$Spending, k = 5)

tree_mae_cv = c()

tree_mape_cv = c()
```

```r
tree_rmse_cv = c()

for (test_row in cv) {

  purchase_train = purchase[-test_row, ]
  purchase_test = purchase[test_row, ]

  tree = rpart(Spending ~ ., data = purchase_train)

  pred_tree = predict(tree, purchase_test)

  # MAE
  tree_mae = mean(abs(pred_tree - purchase_test[, 23]))
  mae_cv = c(tree_mae_cv, tree_mae)

  # MAPE
  tree_mape = mean(abs((pred_tree - purchase_test[, 23]) / purchase_test[, 23]))
  mape_cv = c(tree_mape_cv, tree_mape)

  # RMSE
  tree_rmse = sqrt(mean((pred_tree - purchase_test[, 23])^2))

  rmse_cv = c(tree_rmse_cv, tree_rmse)
}

# Average MAE
print(mean(mae_cv))
```

```
## [1] 101.1279
```

```r
#Average MAPE
print(mean(mape_cv))
```

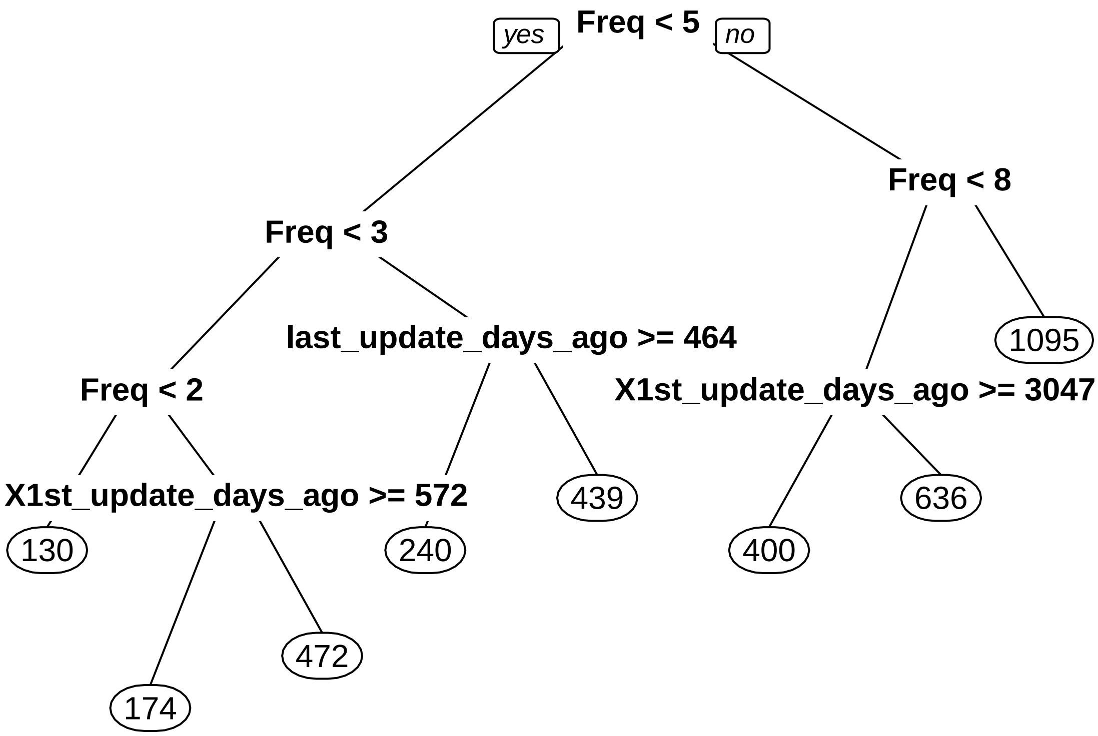
```
## [1] 1.404391
```

```r
# Average RMSE
print(mean(rmse_cv))
```

```
## [1] 173.8571
```

```r
# Build a Single Regression Tree Model
tree = rpart(Spending ~ ., data = purchase_train)

# Plot the Tree
prp(tree, varlen = 0)
```



Answer: There are ___ decision nodes in the tree.

Answer:

2. Use 5-fold cross-validation to evaluate the performance of the linear regression model on this dataset. Report the average (1) MAE; (2) MAPE; (3) RMSE.

Then, build a single linear regression model on the entire dataset and examine the model.

Pick a coefficient and interpret it.

```
# Cross-Validation
cv = createFolds(y = purchase$Spending, k = 5)

lm_mae_cv = c()
```

```
lm_mape_cv = c()

lm_rmse_cv = c()

for (test_row in cv) {

  purchase_train = purchase[-test_row, ]
  purchase_test = purchase[test_row, ]

  lm_model = lm(Spending ~ ., data = purchase_train)

  pred_lm = predict(lm_model, purchase_test)

  # MAE
  lm_mae = mean(abs(pred_lm - purchase_test[, 23]))
  mae_cv = c(lm_mae_cv, lm_mae)

  # MAPE
  lm_mape = mean(abs((pred_lm - purchase_test[, 23]) / purchase_test[, 23]))
  mape_cv = c(lm_mape_cv, lm_mape)

  # RMSE
  lm_rmse = sqrt(mean((pred_lm - purchase_test[, 23])^2))

  rmse_cv = c(lm_rmse_cv, lm_rmse)
}

# Average MAE
print(mean(mae_cv))
```

```
## [1] 90.39615
```

```
#Average MAPE
print(mean(mape_cv))
```

```
## [1] 1.106005
```

```
# Average RMSE
print(mean(rmse_cv))
```

```
## [1] 148.9209
```

```
# Build a Single Linear Regression Model
lm_model = lm(Spending ~ ., data = purchase_train)

# Examine the Model
summary(lm_model)
```

```
##
## Call:
## lm(formula = Spending ~ ., data = purchase_train)
```

```
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -469.43  -90.36  -16.23   50.54 1201.50
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)          7.297e+01  6.059e+01   1.204    0.229
## US                  -1.558e+01  1.660e+01  -0.939    0.348
## source_a             5.206e+01  5.830e+01   0.893    0.372
## source_c            -1.777e+01  6.346e+01  -0.280    0.780
## source_b             2.156e+01  6.436e+01   0.335    0.738
## source_d            -7.009e+00  6.319e+01  -0.111    0.912
## source_e             5.760e+00  5.895e+01   0.098    0.922
## source_m            -1.995e+01  7.070e+01  -0.282    0.778
## source_o             1.407e+01  7.972e+01   0.176    0.860
## source_h            -1.361e+02  8.327e+01  -1.634    0.103
## source_r             8.840e+01  6.045e+01   1.462    0.144
## source_s             2.693e+00  6.660e+01   0.040    0.968
## source_t             8.370e-01  6.678e+01   0.013    0.990
## source_u             3.827e+01  5.835e+01   0.656    0.512
## source_p             7.945e+00  7.779e+01   0.102    0.919
## source_x            -4.509e+00  7.235e+01  -0.062    0.950
## source_w             7.802e+00  5.946e+01   0.131    0.896
## Freq                 9.492e+01  5.135e+00  18.486  < 2e-16 ***
## last_update_days_ago -9.784e-03 1.025e-02  -0.955    0.340
## X1st_update_days_ago -1.684e-02 1.207e-02  -1.395    0.163
## Web.order            2.476e+00  1.201e+01   0.206    0.837
## Gender.male          6.726e+00  1.202e+01   0.559    0.576
## Address_is_res      -8.065e+01  1.470e+01  -5.487 5.54e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 167.3 on 777 degrees of freedom
## Multiple R-squared:  0.4965, Adjusted R-squared:  0.4822
## F-statistic: 34.83 on 22 and 777 DF,  p-value: < 2.2e-16
```

A one-unit increase in X is associated with b-units increase in Prediction Y.

Part II. Logistic Regression and Feature Selection

```r
library(FSelectorRcpp)
library(pROC)
```

1. Import the data. Convert the "spam" variable to a factor.

```r
spambase = read.csv("spambase.csv")

spambase$spam = factor(spambase$spam)
```

2. Use 5-fold cross-validation to evaluate the performance of the logistic regression model on this dataset. Report the average (1) accuracy; (2) precision, recall, and F-measure of class "spam"; (3) AUC of class "spam".

```r
cv = createFolds(y = spambase$spam, k = 5)

accuracy = c()

for (test_row in cv) {

  spambase_train = spambase[-test_row, ]
  spambase_test = spambase[test_row, ]

  logit_model = glm(spam ~ ., data = spambase_train,
                    family = binomial(link = "logit"))

  # Log Odds
  pred = predict(logit_model, spambase_test)

  # Predicted Probability
  pred_prob = predict(logit_model, spambase_test,
                      type = "response")

  # Binary Predictions
  pred_binary = ifelse(pred_prob > 0.5, "yes", "no")

  cm = confusionMatrix(factor(pred_binary), factor(ifelse(spambase_test$spam == "1", "yes", "no")),
                  mode = "prec_recall", positive = "yes")

  accuracy = c(accuracy, cm$overall[1])
}

# Average Accuracy
print(mean(accuracy))
```

```
## [1] 0.9271864
```

```r
# Precision
print(cm$byClass[5])
```

```
## Precision
## 0.9088398
```

```r
# Recall
print(cm$byClass[6])
```

```
##    Recall
## 0.9063361
```

```r
# F-Measure
print(cm$byClass[7])
```

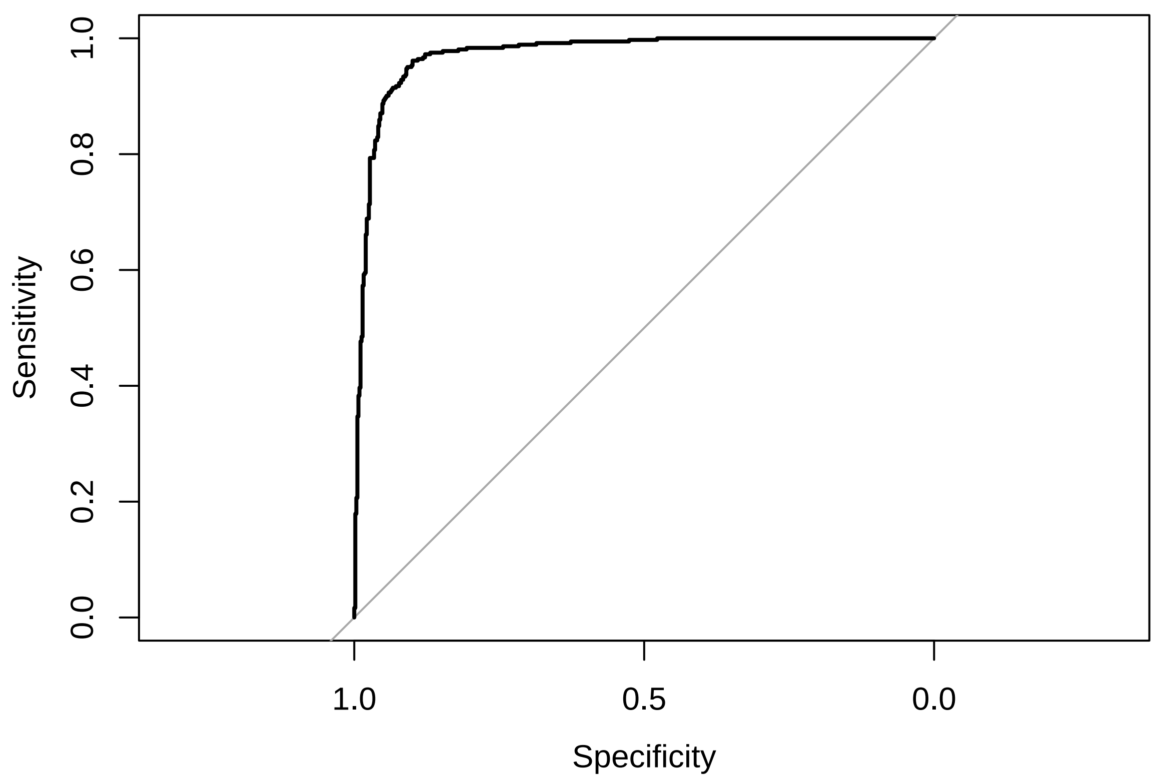
```
##        F1
## 0.9075862
```

```r
# AUC
roc_logit = roc(response = spambase_test$spam,
                predictor = pred_prob)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```r
plot(roc_logit)
```



```
auc(roc_logit)
```

```
## Area under the curve: 0.9724
```

3. Perform feature selection using the information gain metric. Build the best logistic regression with the selected features.

Note: Use a for loop to try different numbers of features. The "best" model is defined as the one with highest AUC for class "spam".

Report the features and AUC of the "best" model.

```
best_auc = 0
best_model = vector()

for (i in 1:57) {
  IG = information_gain(spam ~ ., data = spambase_train)

  topK = cut_attrs(IG, k = i)

  train = spambase_train %>% select(topK, spam)
  test = spambase_test %>% select(topK, spam)

  logit_model = glm(spam ~ ., data = train, family = binomial(link = "logit"))
  pred_prob = predict(logit_model, test, type = "response")
  auc = auc(ifelse(test$spam == 1, 1, 0), pred_prob)
  if (auc > best_auc){
    best_auc = auc
    best_model = topK
  }
}
```

```
## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases
```

```
best_model
```

```
##  [1] "char_freq_..3"              "char_freq_..4"
##  [3] "capital_run_length_longest" "word_freq_remove"
##  [5] "word_freq_your"             "capital_run_length_average"
##  [7] "word_freq_free"             "word_freq_money"
##  [9] "capital_run_length_total"   "word_freq_000"
## [11] "word_freq_hp"               "word_freq_george"
## [13] "word_freq_you"              "word_freq_our"
## [15] "word_freq_hpl"              "word_freq_receive"
## [17] "word_freq_all"              "word_freq_business"
## [19] "word_freq_address"          "word_freq_credit"
## [21] "word_freq_internet"         "word_freq_mail"
## [23] "word_freq_order"            "word_freq_over"
## [25] "word_freq_email"            "word_freq_will"
## [27] "char_freq_..5"              "word_freq_addresses"
## [29] "char_freq_..1"              "word_freq_re"
## [31] "word_freq_1999"             "word_freq_lab"
## [33] "word_freq_labs"             "word_freq_make"
## [35] "word_freq_85"               "word_freq_edu"
## [37] "word_freq_650"              "word_freq_meeting"
## [39] "word_freq_telnet"           "word_freq_people"
## [41] "word_freq_857"              "word_freq_original"
## [43] "word_freq_415"              "word_freq_pm"
## [45] "word_freq_report"           "word_freq_data"
## [47] "word_freq_project"          "word_freq_cs"
## [49] "word_freq_technology"       "word_freq_conference"
## [51] "char_freq_..2"              "word_freq_font"
## [53] "word_freq_3d"               "char_freq_."
## [55] "word_freq_direct"           "word_freq_table"
## [57] "word_freq_parts"
```

```
best_auc
```

```
## Area under the curve: 0.9724
```

4. Perform forward feature selection. Build the best logistic regression with the selected features.

The "best" model is defined as the one with highest AUC for class "spam", evaluated using 5-fold cross-validation.

Report the features and AUC of the "best" model.

Forward Feature Selection

```
best_auc = 0
selected_features = c()
while (TRUE) {
  feature_to_add = -1
  # The elements of setdiff(x, y) are those elements in x, but not in y
  for (i in setdiff(1:57, selected_features)) {
    train = spambase_train %>% select(selected_features, i, spam)
    test = spambase_test %>% select(selected_features, i, spam)
    spambase_best = spambase %>% select(selected_features, i, spam)

    logit_model = glm(spam ~ ., data = train,
                      family = binomial(link = "logit"))

    pred_prob = predict(logit_model, test,
                        type = "response")

    auc = auc(test$spam, pred_prob)

    if (auc > best_auc) {
      best_auc = auc
      feature_to_add = i
    }
  }

  if (feature_to_add != -1) {
    selected_features = c(selected_features, feature_to_add)
  }
  else break
}

print(selected_features)
```

```
##  [1] 52  7 53 25 27 46  5 42 44  9 20  8 16 24 41 12 33 39 26 48 10 35 29
```

```
print(best_auc)
```

```
## Area under the curve: 0.9786
```

Cross-Validation

```
cv = createFolds(y = spambase_best$spam, k = 5)

accuracy = c()

for (test_row in cv) {

  spambase_best_train = train
  spambase_best_test = test

  logit_model = glm(spam ~ ., data = spambase_best_train,
                    family = binomial(link = "logit"))

  # Log Odds
  pred = predict(logit_model, spambase_best_test)

  # Predicted Probability
  pred_prob = predict(logit_model, spambase_best_test,
                      type = "response")

  # Binary Predictions
  pred_binary = ifelse(pred_prob > 0.5, "yes", "no")

  cm = confusionMatrix(factor(pred_binary), factor(ifelse(spambase_best_test$spam == "1", "yes", "no"))
                       mode = "prec_recall", positive = "yes")

  accuracy = c(accuracy, cm$overall[1])
}

# Confusion Matrix
print(cm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  534  39
##        yes  23 324
##
##                Accuracy : 0.9326
##                  95% CI : (0.9144, 0.9479)
##     No Information Rate : 0.6054
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.8579
```

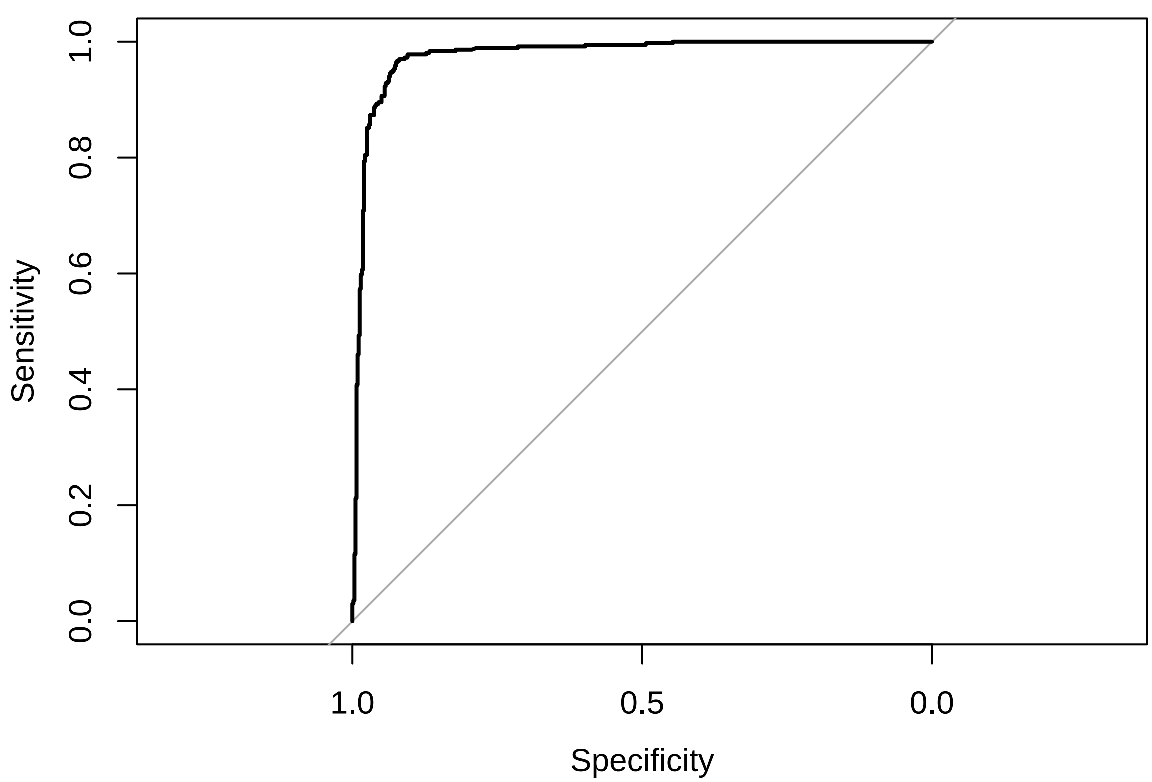
```
##
##  Mcnemar's Test P-Value : 0.05678
##
##               Precision : 0.9337
##                  Recall : 0.8926
##                      F1 : 0.9127
##              Prevalence : 0.3946
##          Detection Rate : 0.3522
##    Detection Prevalence : 0.3772
##       Balanced Accuracy : 0.9256
##
##        'Positive' Class : yes
##
```

```r
# AUC
roc_logit = roc(response = spambase_best_test$spam,
                predictor = pred_prob)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```r
plot(roc_logit)
```



```r
auc(roc_logit)
```

```
## Area under the curve: 0.977
```

Performance Evaluation

```r
# Confusion Matrix
print(cm)
```

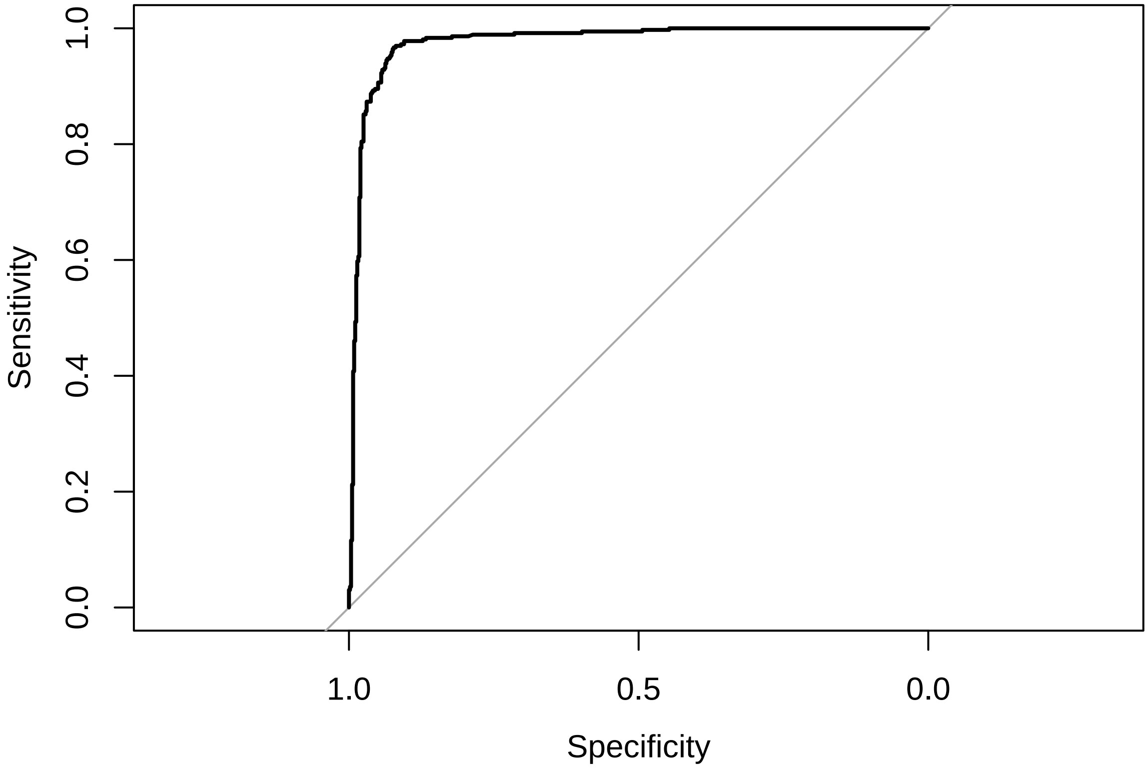
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  534  39
##        yes  23 324
##
##                Accuracy : 0.9326
##                  95% CI : (0.9144, 0.9479)
##     No Information Rate : 0.6054
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.8579
##
##  Mcnemar's Test P-Value : 0.05678
##
##               Precision : 0.9337
##                  Recall : 0.8926
##                      F1 : 0.9127
##              Prevalence : 0.3946
##          Detection Rate : 0.3522
##    Detection Prevalence : 0.3772
##       Balanced Accuracy : 0.9256
##
##        'Positive' Class : yes
##
```

```r
# AUC
roc_logit = roc(response = spambase_best_test$spam,
                predictor = pred_prob)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```r
plot(roc_logit)
```




```r
auc(roc_logit)
```

```
## Area under the curve: 0.977
```