Regression/Classification Models

2023-12-07

Contents

```r
# import train and test data
train_data = read.csv("data/condensed_train_data.csv")
test_data = read.csv("data/condensed_test_data.csv")

# X is just an index, remove it
train_data <- subset(train_data, select = -X)
test_data <- subset(test_data, select = -X)


head(train_data)
```

```
##         id total_events total_nonproduction_events total_remove_cut_events
## 1 001519c8         2557                        120                     417
## 2 0042269b         4136                        175                     439
## 3 0059420b         1556                         99                     151
## 4 0075873a         2531                         72                     517
## 5 0093f095         1765                         34                     148
## 6 009e23ab         2353                        155                     222
##   total_paste_events total_replace_events avg_action_time max_action_time
## 1                  0                    7       116.24677            2259
## 2                  0                    7       101.83777            3005
## 3                  1                    1       121.84833             806
## 4                  0                    0       123.94390             701
## 5                  0                    0       109.71785             501
## 6                  0                    1        90.75563             803
##   min_action_time sd_action_time final_word_count avg_text_change_length
## 1               0       91.79737              256               1.370747
## 2               0       82.38377              404               1.422872
## 3               0      113.76823              206               1.462725
## 4               0       62.08201              252               1.199131
## 5               0       37.01833              242               1.134844
## 6               0       41.93495              308               1.463238
##   total_text_removed max_cursor_movement avg_cursor_movement sd_cursor_movement
## 1                524                 591           0.4092332           43.37815
## 2                970                1826           0.1187424           72.08293
## 3                168                 100           0.5125402           10.01403
## 4                517                 468           0.5541502           24.40666
## 5                148                 223           0.8134921           11.11349
## 6                228                1613           0.7444728          102.45500
##   score
## 1   3.5
```

```
## 2   6.0
## 3   2.0
## 4   4.0
## 5   4.5
## 6   4.0
```

```r
head(test_data)
```

```
##         id total_events total_nonproduction_events total_remove_cut_events
## 1 0022f953         2454                        254                     260
## 2 0081af50         2211                         76                     338
## 3 00e1f05a         7826                        228                    1446
## 4 0190ff4c         1922                         46                     118
## 5 01c359fc         2934                        155                     251
## 6 01d602a7         3573                         73                     358
##   total_paste_events total_replace_events avg_action_time max_action_time
## 1                  1                    1       112.22127            1758
## 2                  0                    3        81.40434            1102
## 3                  0                    7        93.34321           11017
## 4                  0                    0        98.37773             219
## 5                  0                    0       126.43626             582
## 6                  0                    0       145.40862             501
##   min_action_time sd_action_time final_word_count avg_text_change_length
## 1               0       55.43119              323               1.729014
## 2               0       40.65305              275               1.251922
## 3               0      198.89669              739               1.347048
## 4               0       28.53198              299               1.167534
## 5               0       46.29272              430               1.369802
## 6               0       50.09804              487               1.143017
##   total_text_removed max_cursor_movement avg_cursor_movement sd_cursor_movement
## 1                271                1336           0.6192417          85.350330
## 2                366                 477           0.6538462          28.833439
## 3               2573                2668           0.5252396          73.546520
## 4                118                 513           0.4232171          36.891782
## 5                251                1970           0.6137061          87.148047
## 6                358                  33           0.7793953           1.160606
##   score
## 1   3.5
## 2   2.0
## 3   4.5
## 4   4.0
## 5   3.5
## 6   4.5
```

```r
dim(train_data)
```

```
## [1] 1976   17
```

```r
dim(test_data)
```

```
## [1] 495  17
```

```r
train_x = train_data[, 2:16]
train_y = train_data[, 17]

test_x = test_data[, 2:16]
test_y = test_data[, 17]
```

We will start with elastic net regression, to do both feature selection and address collinearity issues that may exist within the data.

```r
set.seed(432)
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-8
```

```r
# Grid of alpha values to try
# alpha_values <- c(0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1)
alpha_values <- seq(0, 1, by = 0.01)


# Create an empty matrix to store results
results <- matrix(NA, nrow = length(alpha_values), ncol = 2, dimnames = list(NULL, c("Alpha", "RMSE")))

# Perform grid search
for (i in seq_along(alpha_values)) {
  alpha <- alpha_values[i]
  lasso.fit <- cv.glmnet(data.matrix(train_x), train_y, alpha = alpha)

  # Find the index of the lambda that minimizes CV error
  min_lambda_index <- which.min(lasso.fit$cvm)

  # Optimal lambda and corresponding RMSE
  optimal_lambda <- lasso.fit$lambda[min_lambda_index]
  predictions <- predict(lasso.fit, newx = data.matrix(test_x), s = optimal_lambda)
  rmse <- sqrt(mean((predictions - test_y)^2))

  # Store results
  results[i, ] <- c(alpha, rmse)
}

# Find the row with the minimum RMSE
min_rmse_row <- which.min(results[, "RMSE"])

# Optimal alpha and lambda
optimal_alpha <- results[min_rmse_row, "Alpha"]
optimal_lambda <- lasso.fit$lambda[which.min(lasso.fit$cvm)]

# Print optimal values
cat("Optimal Alpha:", optimal_alpha, "\n")
```

```
## Optimal Alpha: 0.98
```

```r
cat("Optimal Lambda:", optimal_lambda, "\n")
```

```
## Optimal Lambda: 0.002004471
```

```r
cat("Optimal RMSE:", results[min_rmse_row, "RMSE"], "\n")
```

```
## Optimal RMSE: 0.7546587
```

As we can see, a the values alpha = 0.98, lambda = 0.002004471 return the optimal RMSE for the elastic net models, with an RMSE = 0.7546587

```r
# Assuming elastic_net_fit is your fitted Elastic Net model
elastic_net_fit <- cv.glmnet(data.matrix(train_data[, 2:16]), train_data[, 17], alpha = 0.98)

# Get coefficients for the optimal lambda
optimal_lambda <- 0.002004471

coefficients <- coef(elastic_net_fit, s = optimal_lambda)

# Print or inspect the coefficients
print(coefficients)
```

```
## 16 x 1 sparse Matrix of class "dgCMatrix"
##                                        s1
## (Intercept)                  2.285071e+00
## total_events                 3.845065e-04
## total_nonproduction_events -1.903365e-04
## total_remove_cut_events    -4.182445e-04
## total_paste_events         -1.161791e-03
## total_replace_events        1.840437e-02
## avg_action_time            -1.252771e-04
## max_action_time             1.594265e-06
## min_action_time            -4.849160e-02
## sd_action_time              .
## final_word_count            1.049640e-03
## avg_text_change_length     -9.385741e-02
## total_text_removed         -1.689733e-04
## max_cursor_movement         .
## avg_cursor_movement        -7.187129e-02
## sd_cursor_movement          5.070199e-03
```

```r
order(abs(coefficients), decreasing = TRUE)
```

```
## <sparse>[ <logic> ]: .M.sub.i.logical() maybe inefficient

##  [1]  1 12 15  9  6 16  5 11  4  2  3 13  7  8 10 14
```

```r
rownames(coefficients)[order(abs(coefficients), decreasing = TRUE)]
```

```
## <sparse>[ <logic> ]: .M.sub.i.logical() maybe inefficient
```

```
##  [1] "(Intercept)"                "avg_text_change_length"
##  [3] "avg_cursor_movement"        "min_action_time"
##  [5] "total_replace_events"       "sd_cursor_movement"
##  [7] "total_paste_events"         "final_word_count"
##  [9] "total_remove_cut_events"    "total_events"
## [11] "total_nonproduction_events" "total_text_removed"
## [13] "avg_action_time"            "max_action_time"
## [15] "sd_action_time"             "max_cursor_movement"
```

As we can see from the coefficient output, the elastic net model removed the variables "sd_action_time" and "max_cursor_movement" indicating that they may not be important for predicting the score of the row. Noting the largest coefficients by magnitude, we note that the variables "avg_text_change_length", "avg_cursor_movement", "min_action_time" seem important for predicting the score.

We will now try k-nearest neighbors

```r
set.seed(432)

# "1-nearest neighbor" regression using kknn package
library(kknn)
```

```
## Warning: package 'kknn' was built under R version 4.3.2
```

```r
knn.fit = kknn(y ~ ., train = data.frame(x = train_data[, 2:16], y = train_data[, 17]),
               test = data.frame(x = test_data[, 2:16]),
               k = 1, kernel = "rectangular")
test.pred = knn.fit$fitted.values



# Calculate the root mean squared error (RMSE)
rmse <- sqrt(mean((test.pred - test_y)^2))

# Print or use the RMSE
cat("Root Mean Squared Error (RMSE):", rmse, "\n")
```

```
## Root Mean Squared Error (RMSE): 1.028876
```

```r
library(kknn)

set.seed(432)

# Specify a range of k values to test
k_values <- seq(1, 200, by = 1)  # Adjust the range as needed

# Initialize variables to store results
rmse_values <- numeric(length(k_values))

# Loop over different k values
for (i in seq_along(k_values)) {
  # Fit KNN model
  knn_fit <- kknn(y ~ ., train = data.frame(x = train_x, y = train_y),
```

```r
                 test = data.frame(x = test_x),
                 k = k_values[i])

  # Make predictions
  test_pred <- knn_fit$fitted.values

  # Calculate RMSE
  residuals <- test_pred - test_y
  mse <- mean(residuals^2)
  rmse_values[i] <- sqrt(mse)
}

k_values
```

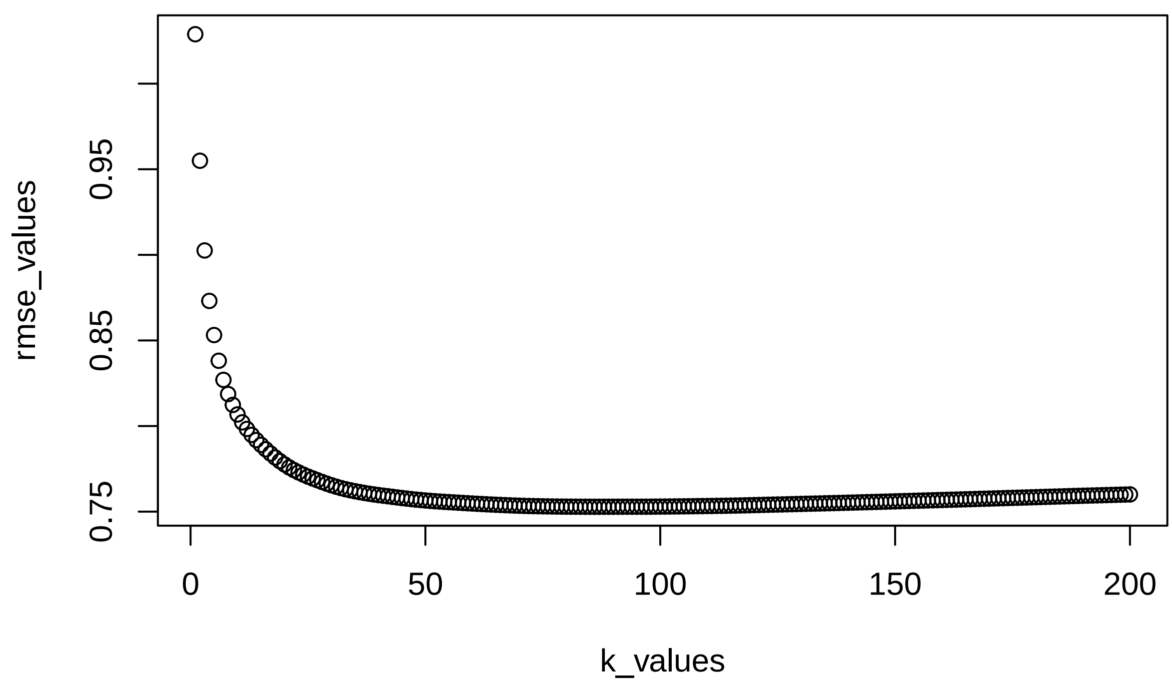
```
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
##  [19]  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
##  [37]  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
##  [55]  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
##  [73]  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
##  [91]  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108
## [109] 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126
## [127] 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144
## [145] 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162
## [163] 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180
## [181] 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198
## [199] 199 200
```

```r
rmse_values
```

```
##   [1] 1.0288760 0.9549758 0.9025537 0.8730921 0.8531476 0.8381311 0.8269766
##   [8] 0.8186804 0.8123432 0.8067794 0.8021411 0.7983028 0.7948448 0.7917984
##  [15] 0.7890232 0.7864230 0.7839734 0.7816491 0.7795199 0.7775846 0.7758163
##  [22] 0.7742726 0.7729047 0.7716265 0.7704527 0.7693762 0.7683433 0.7673268
##  [29] 0.7663343 0.7654100 0.7645595 0.7637800 0.7630919 0.7624803 0.7619254
##  [36] 0.7614086 0.7609268 0.7604792 0.7600553 0.7596652 0.7593228 0.7589931
##  [43] 0.7586347 0.7582742 0.7579403 0.7576231 0.7573123 0.7570240 0.7567658
##  [50] 0.7565206 0.7562866 0.7560785 0.7558970 0.7557252 0.7555520 0.7553841
##  [57] 0.7552202 0.7550559 0.7548916 0.7547329 0.7545797 0.7544280 0.7542872
##  [64] 0.7541491 0.7540156 0.7538965 0.7537818 0.7536648 0.7535507 0.7534454
##  [71] 0.7533521 0.7532740 0.7532145 0.7531655 0.7531129 0.7530626 0.7530167
##  [78] 0.7529718 0.7529318 0.7529054 0.7528932 0.7528783 0.7528619 0.7528502
##  [85] 0.7528404 0.7528356 0.7528342 0.7528376 0.7528442 0.7528505 0.7528566
##  [92] 0.7528587 0.7528585 0.7528593 0.7528622 0.7528681 0.7528788 0.7528984
##  [99] 0.7529231 0.7529468 0.7529727 0.7530016 0.7530310 0.7530605 0.7530916
## [106] 0.7531267 0.7531635 0.7532006 0.7532397 0.7532797 0.7533159 0.7533514
## [113] 0.7533930 0.7534406 0.7534918 0.7535461 0.7536054 0.7536688 0.7537352
## [120] 0.7538043 0.7538755 0.7539474 0.7540183 0.7540864 0.7541522 0.7542194
## [127] 0.7542882 0.7543571 0.7544262 0.7544970 0.7545676 0.7546368 0.7547074
## [134] 0.7547794 0.7548516 0.7549235 0.7549943 0.7550651 0.7551364 0.7552086
## [141] 0.7552832 0.7553595 0.7554383 0.7555192 0.7555994 0.7556800 0.7557604
## [148] 0.7558400 0.7559205 0.7560000 0.7560793 0.7561574 0.7562348 0.7563153
## [155] 0.7563975 0.7564785 0.7565577 0.7566371 0.7567169 0.7567968 0.7568764
## [162] 0.7569567 0.7570384 0.7571213 0.7572047 0.7572879 0.7573706 0.7574532
```

```
## [169] 0.7575348 0.7576155 0.7576966 0.7577779 0.7578601 0.7579432 0.7580263
## [176] 0.7581099 0.7581950 0.7582806 0.7583654 0.7584506 0.7585359 0.7586192
## [183] 0.7587022 0.7587858 0.7588686 0.7589518 0.7590352 0.7591166 0.7591958
## [190] 0.7592743 0.7593522 0.7594291 0.7595062 0.7595840 0.7596619 0.7597405
## [197] 0.7598195 0.7598987 0.7599778 0.7600565
```

```r
# want to plot rmse vs k_values
plot(k_values, rmse_values)
```




```r
# Find the k that gives the lowest RMSE
optimal_k <- k_values[which.min(rmse_values)]

# Print results
cat("Optimal K:", optimal_k, "\n")
```

```
## Optimal K: 87
```

```r
cat("Minimum RMSE:", min(rmse_values), "\n")
```

```
## Minimum RMSE: 0.7528342
```

We see that the optimal k = 87 returns the lowest RMSE for the k-nearest neighbors regression with RMSE = 0.7528342.

We will now try random forest.

```r
set.seed(432)

# fit random forests with a selected tuning
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.3.2
```

```
## randomForest 4.7-1.1
```

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

```r
rf.fit = randomForest(train_x, train_y)

summary(rf.fit)
```

```
##                 Length Class  Mode
## call               3   -none- call
## type               1   -none- character
## predicted       1976   -none- numeric
## mse              500   -none- numeric
## rsq              500   -none- numeric
## oob.times       1976   -none- numeric
## importance        15   -none- numeric
## importanceSD       0   -none- NULL
## localImportance    0   -none- NULL
## proximity          0   -none- NULL
## ntree              1   -none- numeric
## mtry               1   -none- numeric
## forest            11   -none- list
## coefs              0   -none- NULL
## y               1976   -none- numeric
## test               0   -none- NULL
## inbag              0   -none- NULL
```

```r
# ntree = 500
# sampsize = ?
# mtry = 5
# nodesize ?

# Assuming 'test_x' contains your test features and 'test_y' contains your test target variable

# Make predictions on the test data
rf_predictions <- predict(rf.fit, newdata = test_x)

# Calculate the root mean squared error (RMSE)
rmse <- sqrt(mean((rf_predictions - test_y)^2))

# Print or use the RMSE
cat("Root Mean Squared Error (RMSE):", rmse, "\n")
```

```
## Root Mean Squared Error (RMSE): 0.6985063
```

Random forest has the best RMSE so far, with an RMSE = 0.6985063.

```r
library(randomForest)

# Set seed for reproducibility
set.seed(432)

# Define candidate values for parameters
ntree_values <- c(400, 500, 600, 700, 800)
sampsize_values <- c(90, 100, 110, 120, 130)
mtry_values <- c(12, 13, 14, 15)
nodesize_values <- c(1, 2, 3, 4, 5)

# Initialize variables to store optimal values
best_rmse <- Inf
optimal_ntree <- NULL
optimal_sampsize <- NULL
optimal_mtry <- NULL
optimal_nodesize <- NULL
best_model <- NULL

# Iterate over parameter combinations
for (ntree in ntree_values) {
  for (sampsize in sampsize_values) {
    for (mtry in mtry_values) {
      for (nodesize in nodesize_values) {

        # Train the model
        rf_model <- randomForest(train_y ~ ., data = train_x,
                                 ntree = ntree, mtry = mtry,
                                 sampsize = sampsize, nodesize = nodesize)

        # Make predictions on the test data
        rf_predictions <- predict(rf_model, newdata = test_x)

        # Calculate RMSE
        rmse <- sqrt(mean((rf_predictions - test_y)^2))

        # Check if current combination improves RMSE
        if (rmse < best_rmse) {
          best_rmse <- rmse
          optimal_ntree <- ntree
          optimal_sampsize <- sampsize
          optimal_mtry <- mtry
          optimal_nodesize <- nodesize
          best_model <- rf_model
        }
      }
    }
  }
}

# Print or use the optimal values
cat("Optimal ntree:", optimal_ntree, "\n")
```

```
## Optimal ntree: 700
```

```r
cat("Optimal sampsize:", optimal_sampsize, "\n")
```

```
## Optimal sampsize: 110
```

```r
cat("Optimal mtry:", optimal_mtry, "\n")
```

```
## Optimal mtry: 13
```

```r
cat("Optimal nodesize:", optimal_nodesize, "\n")
```

```
## Optimal nodesize: 1
```

```r
cat("Optimal Root Mean Squared Error (RMSE):", best_rmse, "\n")
```

```
## Optimal Root Mean Squared Error (RMSE): 0.6829625
```

As we can see from our gridsearch, the best model has ntree=700, sampsize=110, mtry=13, nodesize=1, with RMSE = 0.6829625.

```r
best_model$importance
```

```
##                           IncNodePurity
## total_events                 17.4698577
## total_nonproduction_events    3.7838847
## total_remove_cut_events       3.9744701
## total_paste_events            0.4763130
## total_replace_events          1.8217903
## avg_action_time               4.4580144
## max_action_time               4.0362660
## min_action_time               0.3842305
## sd_action_time                3.6217054
## final_word_count             51.0266429
## avg_text_change_length        4.1440921
## total_text_removed            3.9363097
## max_cursor_movement           5.3090077
## avg_cursor_movement           4.5206313
## sd_cursor_movement            4.7586477
```

```r
order(best_model$importance, decreasing = TRUE)
```

```
##  [1] 10  1 13 15 14  6 11  7  3 12  2  9  5  4  8
```

```r
rownames(best_model$importance)[order(best_model$importance, decreasing = TRUE)]
```

```
##  [1] "final_word_count"           "total_events"
##  [3] "max_cursor_movement"        "sd_cursor_movement"
##  [5] "avg_cursor_movement"        "avg_action_time"
##  [7] "avg_text_change_length"     "max_action_time"
##  [9] "total_remove_cut_events"    "total_text_removed"
## [11] "total_nonproduction_events" "sd_action_time"
## [13] "total_replace_events"       "total_paste_events"
## [15] "min_action_time"
```

From the variable importance we see that the variables "final_word_count", "total_events", and "max_cursor_movement" are important in predicting the outcome in the random forest model.

We will now try Support Vector Machine classification.

```r
set.seed(432)

library(e1071)

# Assuming 'train_data' contains your training data with an ordinal target variable
# Replace 'target' with the actual column name of your target variable

# Fit an SVM regression model
svm_model <- svm(as.factor(train_y) ~ ., data = train_x, kernel = "linear", type = "C-classification", 

# Make predictions on new data
predictions <- predict(svm_model, newdata = test_x)


# Evaluate the model's performance as needed
# Calculate the confusion matrix
conf_matrix <- table(Actual = test_y, Predicted = predictions)

# Calculate the RMSE
rmse <- sqrt(mean((as.numeric(predictions) - test_y)^2))

# Print the confusion matrix
print("Confusion Matrix:")
```

```
## [1] "Confusion Matrix:"
```

```r
print(conf_matrix)
```

```
##       Predicted
## Actual 0.5  1 1.5  2 2.5  3 3.5  4 4.5  5 5.5  6
##    0.5   0  0   0  0   0  1   0  0   0  0   0  0
##    1     0  0   0  0   0  6   0  0   0  0   0  0
##    1.5   0  0   0  0   0  9   5  2   0  0   0  0
##    2     0  0   0  0   0 15   6  0   0  0   0  0
##    2.5   0  0   0  0   0 33  10  3   0  0   0  0
##    3     0  0   0  0   1 31  19 10   4  0   0  0
##    3.5   0  0   0  0   0 29  41 24   8  0   0  0
##    4     0  0   0  0   0 14  22 37  17  0   1  0
##    4.5   0  0   0  0   0  2  10 29  41  0   0  0
```

```
##    5     0  0    0  0    0  1    2  6  19  0    2  0
##    5.5   0  0    0  0    0  0    1  4  21  0    3  0
##    6     0  0    0  0    0  0    0  2   3  0    1  0
```

```r
accuracy = sum(predictions == test_y) / length(test_y)
cat("Accuracy:", accuracy, "\n")
```

```
## Accuracy: 0.3090909
```

```r
# Print the RMSE
cat("Root Mean Squared Error (RMSE):", rmse, "\n")
```

```
## Root Mean Squared Error (RMSE): 3.898977
```

```r
set.seed(432)

library(e1071)

# Assuming 'train_data' and 'test_data' contain your training and testing data

# Define the values of cost to try
cost_values <- c(1, 2,3,4,5, 10, 50, 100)

# Initialize a vector to store RMSE values
rmse_values <- numeric(length(cost_values))

# Fit SVM models with different cost values
for (i in seq_along(cost_values)) {
  cost <- cost_values[i]

  # Fit the SVM model
  svm_model <- svm(as.factor(train_y) ~ ., data = train_x, kernel = "linear", cost = cost)

  # Make predictions on the test data
  predictions <- predict(svm_model, newdata = test_x)

  # Calculate RMSE
  rmse <- sqrt(mean((as.numeric(predictions) - test_y)^2))

  # Store the RMSE value
  rmse_values[i] <- rmse

  # Print or save other information as needed
  cat("Cost:", cost, " - RMSE:", rmse, "\n")

  # Assuming 'test_data' contains your testing data with 'target_class' as the true class

  # Calculate accuracy
  accuracy <- sum(predictions == test_y) / length(test_y)

  # Print the accuracy
  cat("Accuracy:", accuracy, "\n")
}
```

```
## Cost: 1  - RMSE: 3.898977
## Accuracy: 0.3090909
## Cost: 2  - RMSE: 3.925313
## Accuracy: 0.3070707
## Cost: 3  - RMSE: 3.928914
## Accuracy: 0.3070707
## Cost: 4  - RMSE: 3.925313
## Accuracy: 0.3070707
## Cost: 5  - RMSE: 3.929942
## Accuracy: 0.3090909
## Cost: 10  - RMSE: 3.948405
## Accuracy: 0.3131313
## Cost: 50  - RMSE: 3.935079
## Accuracy: 0.3111111
## Cost: 100  - RMSE: 3.93097
## Accuracy: 0.3111111
```

```r
# Find the optimal cost value
optimal_cost <- cost_values[which.min(rmse_values)]
cat("Optimal Cost:", optimal_cost, "\n")
```

```
## Optimal Cost: 1
```

As we can see from the output, the optimal cost = 1, with an RMSE = 3.898977. What is interesting to note is that even though cost=10 has a higher RMSE, it also has better classification accuracy.

SVM Regression

```r
set.seed(432)

library(e1071)

# Assuming 'train_data' and 'test_data' contain your training and testing data

# Define the values of cost to try
cost_values <- c(1, 30, 50, 70, 75, 80, 100)

# Initialize a vector to store RMSE values
rmse_values <- numeric(length(cost_values))

# Fit SVM models with different cost values
for (i in seq_along(cost_values)) {
  cost <- cost_values[i]

  # Fit the SVM model
  svm_model <- svm(train_y ~ ., data = train_x, kernel = "linear", cost = cost)

  # Make predictions on the test data
  predictions <- predict(svm_model, newdata = test_x)

  # Calculate RMSE
  rmse <- sqrt(mean((predictions - test_y)^2))
```

```r
  # Store the RMSE value
  rmse_values[i] <- rmse

  # Print or save other information as needed
  cat("Cost:", cost, " - RMSE:", rmse, "\n")

  # Assuming 'test_data' contains your testing data with 'target_class' as the true clas
}
```

```
## Cost: 1  - RMSE: 0.7603172
## Cost: 30  - RMSE: 0.7601203
## Cost: 50  - RMSE: 0.760139
## Cost: 70  - RMSE: 0.7601615
## Cost: 75  - RMSE: 0.7580726
## Cost: 80  - RMSE: 0.7597441
## Cost: 100  - RMSE: 0.7607451
```

```r
# Find the optimal cost value
optimal_cost <- cost_values[which.min(rmse_values)]
cat("Optimal Cost:", optimal_cost, "\n")
```

```
## Optimal Cost: 75
```

```r
optimal_rmse <- min(rmse_values)
cat("Optimal RMSE:", optimal_rmse, "\n")
```

```
## Optimal RMSE: 0.7580726
```

As we can see, SVM regression returns a much better RMSE = 0.7580726 with cost = 75 than SVM for classification.

```r
models = c("elastic net", "k nearest neighbors", "random forests", "svm classification", "svm regressio
rmse_values = c(0.7546587, 0.7528342, 0.6829625, 3.898977, 0.7580726)

rmse_table = data.frame(Model = models, RMSE = rmse_values)
rmse_table
```

```
##                 Model      RMSE
## 1         elastic net 0.7546587
## 2 k nearest neighbors 0.7528342
## 3      random forests 0.6829625
## 4  svm classification 3.8989770
## 5      svm regression 0.7580726
```

```r
# sorted order
rmse_table[order(rmse_table$RMSE), ]
```

```
##                 Model      RMSE
## 3      random forests 0.6829625
## 2 k nearest neighbors 0.7528342
## 1         elastic net 0.7546587
## 5      svm regression 0.7580726
## 4  svm classification 3.8989770
```