GTx: ISYE6501x - Homework 1

Muh Alif Ahsanul Islam

5/18/2019

Question 2.1

Describe a situation or problem from your job, everyday life, current events, etc., for which a classification model would be appropriate. List some (up to 5) predictors that you might use.

Answer:

Classifying manufacturing product into acceptable and not-acceptable product.
After a product is produced it is necessary to do quality control process. Products that has defects, or do not meet requirements must be inspected. To automate the process machine learning models can be used.
Predictors:
1. Geometrical shape
2. Color
3. Mass

Question 2.2

The files credit_card_data.txt (without headers) and credit_card_data-headers.txt (with headers) contain a dataset with 654 data points, 6 continuous and 4 binary predictor variables. It has anonymized credit card applications with a binary response variable (last column) indicating if the application was positive or negative.

Question 2.2.1

Using the support vector machine function ksvm contained in the R package kernlab, find a good classifier for this data. Show the equation of your classifier, and how well it classifies the data points in the full data set.

```
y_col_name = 'R1'
cols = c(y_col_name)
data[cols] = lapply(data[cols], factor)
y_col_ind = grep(y_col_name, colnames(data))
X = data[,-y_col_ind]
y = data[,y_col_ind]
X = as.matrix(X)
y = (y)
c_params = c(0.001, 0.01, 0.1, 1, 10, 100, 1000)
model_list = c()
conf_matrix_list = c()
modelling_result = NULL
for (c in c_params){
  cat(sprintf('\n---------------------------\n'))
  cat(sprintf('C: %.6f\n', c))
  model = ksvm(X, y, type='C-svc', kernel='vanilladot', C=c, scaled=TRUE)
  y_pred = predict(model, X)
  conf_matrix = confusionMatrix(y_pred, y)
  accuracy = conf_matrix[['overall']][['Accuracy']] * 100
  f1_score = conf_matrix[['byClass']][['F1']] * 100
```

```
  print(conf_matrix[["table"]])
  cat(sprintf('Accuracy: %.2f %%\tF1 Score: %.2f %%\n', accuracy, f1_score))
  model_list = c(model_list, model)
  conf_matrix_list = c(conf_matrix_list, conf_matrix)
  modelling_result = rbind(modelling_result, data.frame(c, accuracy, f1_score))
}
```

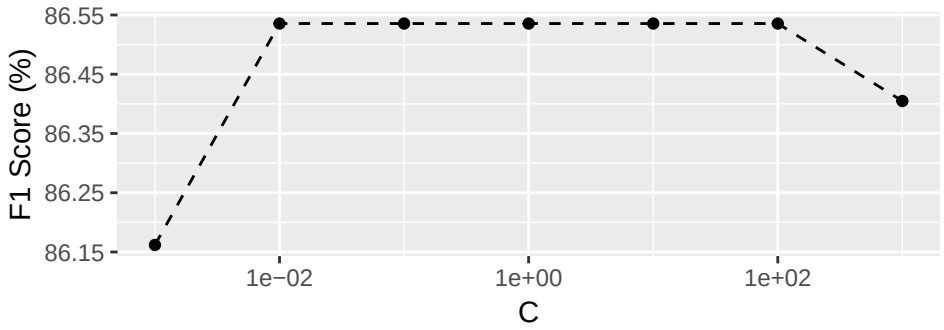
```
##
## ----------------------------
## C: 0.001000
##  Setting default kernel parameters
##           Reference
## Prediction   0   1
##          0 330  78
##          1  28 218
## Accuracy: 83.79 %    F1 Score: 86.16 %
##
## ----------------------------
## C: 0.010000
##  Setting default kernel parameters
##           Reference
## Prediction   0   1
##          0 286  17
##          1  72 279
## Accuracy: 86.39 %    F1 Score: 86.54 %
##
## ----------------------------
## C: 0.100000
##  Setting default kernel parameters
##           Reference
## Prediction   0   1
##          0 286  17
##          1  72 279
## Accuracy: 86.39 %    F1 Score: 86.54 %
##
## ----------------------------
## C: 1.000000
##  Setting default kernel parameters
##           Reference
## Prediction   0   1
##          0 286  17
##          1  72 279
## Accuracy: 86.39 %    F1 Score: 86.54 %
##
## ----------------------------
## C: 10.000000
##  Setting default kernel parameters
##           Reference
## Prediction   0   1
##          0 286  17
##          1  72 279
## Accuracy: 86.39 %    F1 Score: 86.54 %
##
## ----------------------------
```

```
## C: 100.000000
##  Setting default kernel parameters
##            Reference
## Prediction   0   1
##          0 286  17
##          1  72 279
## Accuracy: 86.39 %    F1 Score: 86.54 %
##
## ---------------------------
## C: 1000.000000
##  Setting default kernel parameters
##            Reference
## Prediction   0   1
##          0 286  18
##          1  72 278
## Accuracy: 86.24 %    F1 Score: 86.40 %
```

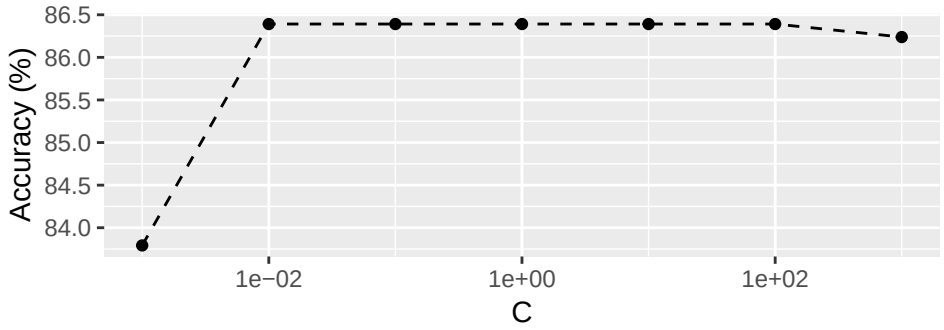
```
modelling_result$kernel = 'linear'
```

F1 Score is more appropriate metric for classification problem instead of accuracy because it considers both the precision and the recall of the test to compute the score. More on F1 score can be found here (https://en.wikipedia.org/wiki/F1_score).

```
ggplot(data=modelling_result, aes(x=c, y=f1_score)) +
  scale_x_continuous(trans='log10') +
  geom_line(linetype = "dashed") + geom_point() +
  labs(x='C', y='F1 Score (%)')
```




```
ggplot(data=modelling_result, aes(x=c, y=accuracy)) +
  scale_x_continuous(trans='log10') +
  geom_line(linetype = "dashed") + geom_point() +
  labs(x='C', y='Accuracy (%)')
```



```
best_model_ind = which.max(modelling_result[, c('f1_score')])
best_model = model_list[[best_model_ind]]
best_c_params = c_params[best_model_ind]
best_f1_score = modelling_result[best_model_ind, c('f1_score')]
a = colSums(best_model@xmatrix[[1]] * best_model@coef[[1]])
a0 = -best_model@b
model_col_name = names(a)
equation = ''
for (i in seq_along(model_col_name)){
  coef = a[i]
  name = model_col_name[i]
  equation = paste(equation, sprintf('%.2e * %s + ', coef, name), sep='')
}
equation = paste0(paste(equation, sprintf('%.2e = 0',a0)))
equation = f_break_string(equation, '+', 70)
```

ANSWER OF QUESTION 2.2.1

```
## Best model with higest F1 Score of 86.54% is model with C = 0.010000
##  Best model equation is:
## -1.50e-04 * A1  + -1.48e-03 * A2 + 1.41e-03 * A3 + 7.29e-03 * A8 +
##  9.92e-01 * A9 + -4.47e-03 * A10 + 7.15e-03 * A11 + -5.47e-04 * A12 + -1.69e-03 * A14 +
##  1.05e-01 * A15 +  8.20e-02 = 0
```

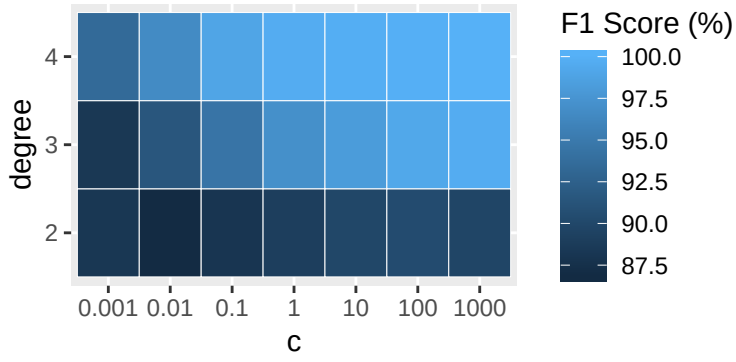
Question 2.2.2

You are welcome, but not required, to try other (nonlinear) kernels as well; we're not covering them in this course, but they can sometimes be useful and might provide better predictions than vanilladot.

Polynomial Kernel SVM

```
c_params = c(0.001, 0.01, 0.1, 1, 10, 100, 1000)
degree_params = c(2, 3, 4)
poly_models = c()
conf_matrix = c()
poly_model_result = NULL
for (degree in degree_params){
  for (c in c_params){
    my_svm_model = f_my_ksvm(X, y, 'polydot', c_params, arg_list=list('degree'=degree),
                             verbose=FALSE)
    poly_models = c(poly_models, my_svm_model$model)
    conf_matrix = c(conf_matrix, my_svm_model$conf_matrix)
    accuracy = my_svm_model$accuracy
    f1_score = my_svm_model$f1_score
    poly_model_result = rbind(poly_model_result, data.frame(c, degree,
                                                             accuracy,
                                                             f1_score))
  }
}
poly_model_result$kernel = 'polynomial'
```

```
cols = c('c', 'degree')
poly_model_result[cols] = lapply(poly_model_result[cols], factor)
ggplot(poly_model_result, aes(x=c,
                              y=degree,
                              fill = f1_score)) +
  geom_tile(color='white') +
  scale_fill_gradient(name = 'F1 Score (%)')
```



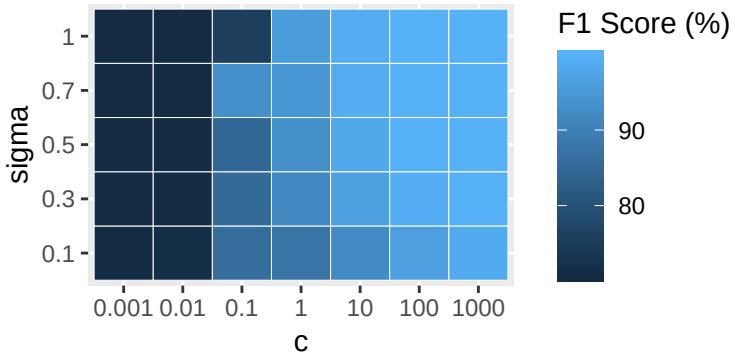

```
## Best Polynomial Kernel Model:
```

```
##     c degree accuracy f1_score     kernel
##  1000      4      100      100 polynomial
```

RBF Kernel SVM

```
sigma_params = c(0.1, 0.3, 0.5, 0.7, 1)
rbf_models = c()
conf_matrix = c()
rbf_model_result = NULL
for (sigma in sigma_params){
  for (c in c_params){
    my_svm_model = f_my_ksvm(X, y, 'rbfdot', c_params, arg_list=list('sigma'=sigma),
                             verbose=FALSE)
    rbf_models = c(rbf_models, my_svm_model$model)
    conf_matrix = c(conf_matrix, my_svm_model$conf_matrix)
    accuracy = my_svm_model$accuracy
    f1_score = my_svm_model$f1_score
    rbf_model_result = rbind(rbf_model_result, data.frame(c, sigma,
                                                          accuracy,
                                                          f1_score))
  }
}
rbf_model_result$kernel = 'rbf'
```

```
cols = c('c', 'sigma')
rbf_model_result[cols] = lapply(rbf_model_result[cols], factor)
ggplot(rbf_model_result, aes(x=c,
                             y=sigma,
                             fill = f1_score)) +
  geom_tile(color='white') +
  scale_fill_gradient(name = 'F1 Score (%)')
```




```
## Best RBF Kernel Model:
```

```
##     c sigma accuracy f1_score kernel
##  1000   0.5 99.69419 99.72145    rbf
```

ANSWER OF QUESTION 2.2.2

```
## Best SVM model among 3 kernels (linear, polynomial, RBF):
```

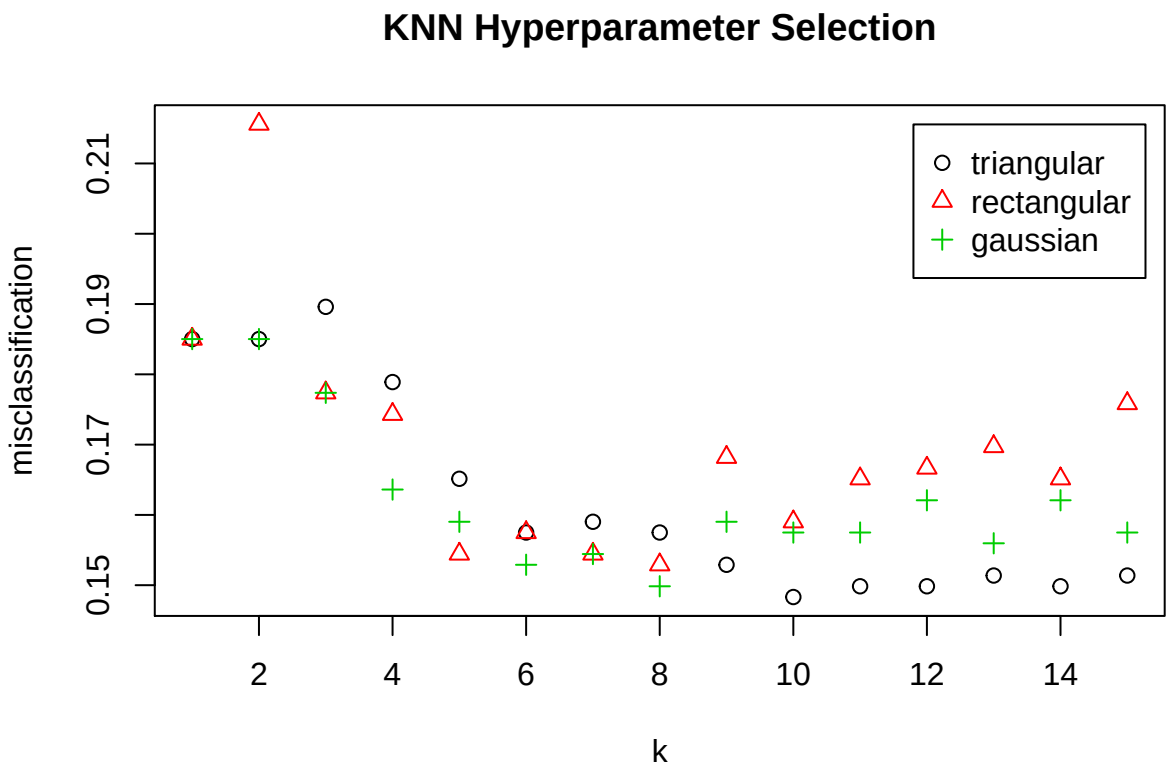
```
##     c accuracy f1_score     kernel degree sigma
##  1000      100      100 polynomial      4  <NA>
```

Because the best model is selected based on test data score which has same data with training data, it will overfit the data.

Question 2.2.3

Using the k-nearest-neighbors classification function kknn contained in the R kknn package, suggest a good value of k, and show how well it classifies that data points in the full data set

```
library(kknn)
train_kknn = train.kknn(formula=R1~., data=data, kmax=15, scale=TRUE,
                        kernel = c('triangular', 'rectangular', 'gaussian'),
                        distance=2)
plot(train_kknn, main='KNN Hyperparameter Selection')
```




```
best_kernel = train_kknn[["best.parameters"]][["kernel"]]
best_k = train_kknn[["best.parameters"]][["k"]]
best_missclasify = train_kknn[["MISCLASS"]][best_k, best_kernel]
best_accuracy = 100 * (1 - best_missclasify)
```

ANSWER OF QUESTION 2.2.3

train.knn is leave-one-out crossvalidation and used to train the model. This will exclude the current validation point from training points and it will not count itself as neighbor.

```
## From  figure above we can see triangular kernel with k=10 produce lowest
## misclassification of 0.1483  or accuracy of 85.17%
```

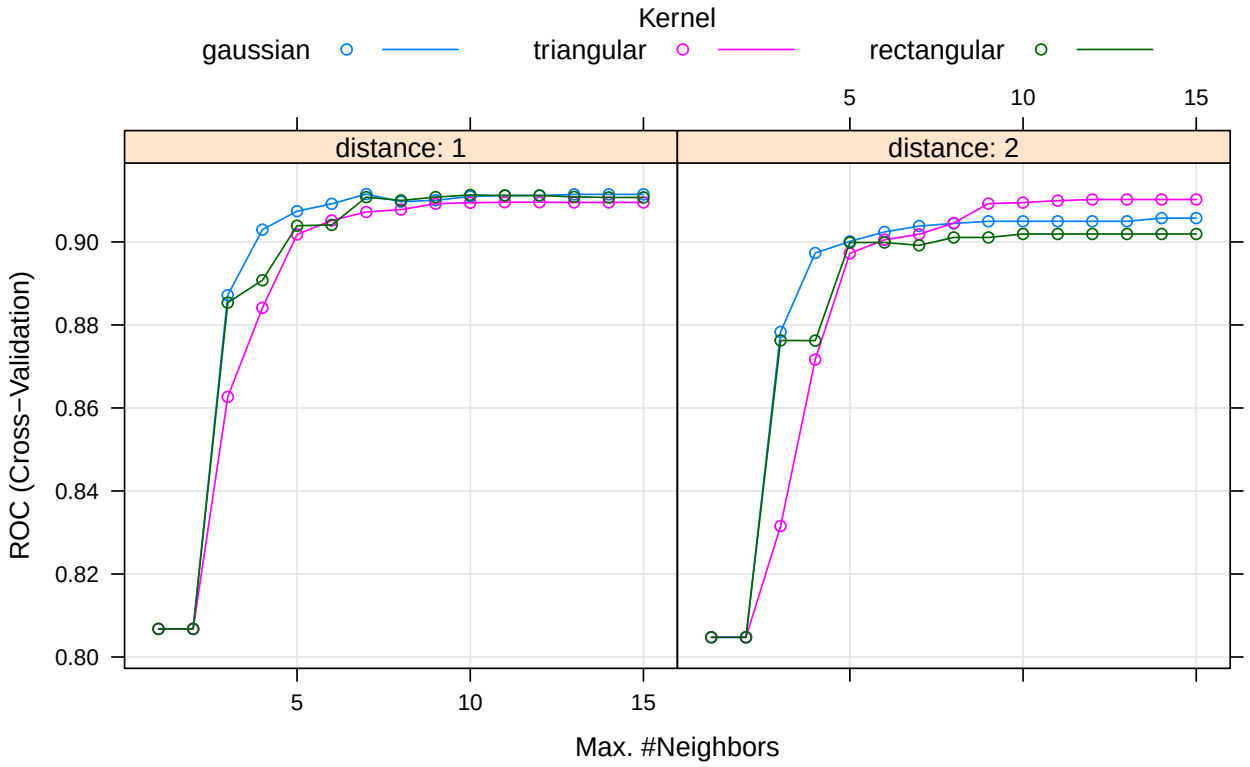
Question 3.1

Using the same data set (credit_card_data.txt or credit_card_data-headers.txt) as in Question 2.2, use the ksvm or kknn function to find a good classifier:
(a) using cross-validation (do this for the k-nearest-neighbors model; SVM is optional); and
(b) splitting the data into training, validation, and test data sets (pick either KNN or SVM; the other is optional).

Question 3.1.a

KNN model cross validation

```
kknn_data = data.frame(data)
levels(kknn_data$R1) <- c("zero", "one")
k_cv = 10
trctrl = trainControl(method='cv', number=k_cv,
                      summaryFunction = twoClassSummary, classProbs = TRUE)
tuneGrid = expand.grid(kmax=1:15, distance=1:2,
                       kernel=c('gaussian', 'triangular', 'rectangular'))
kknn_fit <- train(R1~., data=kknn_data, method='kknn', trControl=trctrl,
                  preProcess = c('center', 'scale'), tuneGrid=tuneGrid, metric='ROC')
plot(kknn_fit)
```



ROC is another good metric to evaluate prediction model, it shows trade off between true positive rate and false positive rate. (https://en.wikipedia.org/wiki/Receiver_operating_characteristic)

```
best_model = kknn_fit[["bestTune"]]
y_hat = predict(kknn_fit, kknn_data)
library(MLmetrics)
levels(kknn_data$R1) <- c("0", "1")
```

```
levels(y_hat) <- c("0", "1")
accuracy = 100 * sum(y_hat == kknn_data$R1) / nrow(data)
f1_score = F1_Score(data$R1, y_hat)
```

ANSWER OF QUESTION 3.1.a

```
## Best model among of KNN model using 10-fold cross validation approach:
```

```
##  kmax distance   kernel
##     7        1 gaussian
```

Question 3.1.b

Training, validation, test split with SVM Model

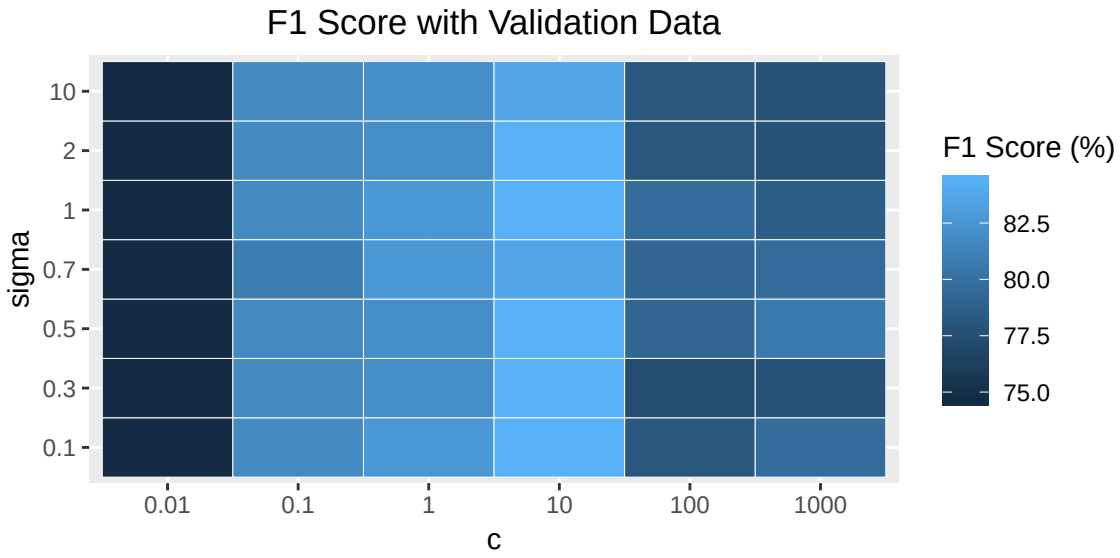
```
data_splitter = f_train_validation_test_split(data, 0.7, 0.2, 0.1)
data_train = data_splitter$train
data_validation = data_splitter$validation
data_test = data_splitter$test
X_train = as.matrix(data_train[, -y_col_ind])
y_train = data_train[, y_col_ind]
X_validation = as.matrix(data_validation[, -y_col_ind])
y_validation = data_validation[, y_col_ind]
X_test = as.matrix(data_test[, -y_col_ind])
y_test = data_test[, y_col_ind]
```

RBF Kernel SVM

```
c_params = c(0.01, 0.1, 1, 10, 100, 1000)
sigma_params = c(0.1, 0.3, 0.5, 0.7, 1, 2, 10)
rbf_models = c()
conf_matrix = c()
rbf_model_result = NULL
verbose = FALSE
for (c in c_params){
  for (sigma in sigma_params){
    model = ksvm(X_train, y_train, type='C-svc', kernel='rbfdot', C=c,
                 sigma=sigma, scaled=TRUE)
    y_validation_pred = predict(model, X_validation)
    conf_matrix = confusionMatrix(y_validation_pred, y_validation)
    accuracy = conf_matrix[['overall']][['Accuracy']] * 100
    f1_score = conf_matrix[['byClass']][['F1']] * 100
    rbf_models = c(rbf_models, model)
    conf_matrix_list = c(conf_matrix_list, conf_matrix)
    rbf_model_result = rbind(rbf_model_result, data.frame(c, sigma, accuracy, f1_score))
    if (verbose){
      cat(sprintf('\n---------------------------\n'))
      cat(sprintf('C: %.6f\n', c))
      print(conf_matrix[["table"]])
      cat(sprintf('Accuracy: %.2f %%\tF1 Score: %.2f %%\n', accuracy, f1_score))
    }
  }
}
```

```
rbf_model_result$kernel = 'rbf'
best_rbf_model = f_best_model(rbf_model_result, 'f1_score')
best_model = rbf_models[[as.integer(row.names(best_rbf_model))]]
y_test_hat = predict(best_model, X_test)
test_accuracy = 100 * sum(y_test_hat == y_test) / length(y_test)
library(MLmetrics)
test_f1_score = 100 * F1_Score(y_test, y_test_hat)
```

```
cols = c('c', 'sigma')
rbf_model_result[cols] = lapply(rbf_model_result[cols], factor)
ggplot(rbf_model_result, aes(x=c,
                             y=sigma,
                             fill = f1_score)) +
  geom_tile(color='white') +
  scale_fill_gradient(name = 'F1 Score (%)') +
  ggtitle('F1 Score with Validation Data') +
  theme(plot.title = element_text(hjust = 0.5))
```




```
## SVM RBF Kernel Model with Highest F1 Score in Validation Data:
```

```
##    c sigma accuracy f1_score kernel
##  10   0.1 82.44275 84.35374    rbf
```

ANSWER OF QUESTION 3.1.b

The model can be considered good because the difference in accuracy and F1 score for validation and testing data is not big. It means the model can capture the real pattern in data and not noise and randomness in training or validation data.

```
## Best Parameters for SVM RBF:
```

```
## C: 10.0000   Sigma: 0.10
```

```
## For testing data:
```

```
## Accuracy: 83.33%
## F1 Score: 84.93%
```