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.
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. The dataset is the "Credit Approval Data Set" from the UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Credit+Approval) without the categorical variables and without data points that have missing values.

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.

```r
setwd('/home/alifahsanul/Documents/analytics_modelling')
data = read.table('./hw1/week_1_data-summer/data 2.2/credit_card_data-headers.txt',
                  header=TRUE)
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)
c_params = c(0.001, 0.01, 0.1, 1, 10)
model_list = c()
conf_matrix_list = c()
modelling_result = NULL
for (c in c_params){
  cat(sprintf('\n---------------------------\n'))
```

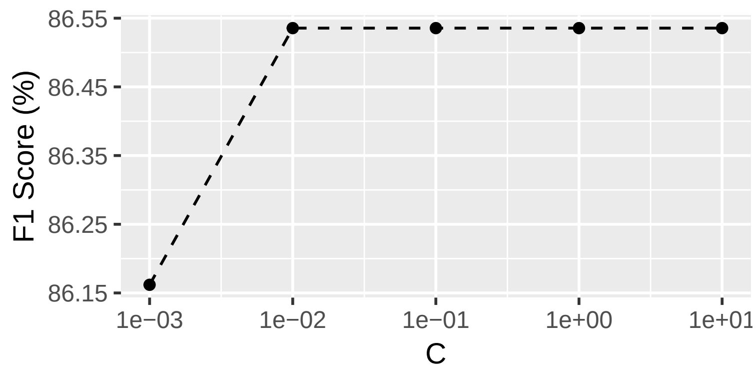
```
  cat(sprintf('C: %.6f\n', c))
  model = ksvm(X, y, type='C-svc', kernel='vanilladot', C=c, scaled=TRUE)
  y_pred = predict(model, X)
  conf_matrix = confusionMatrix(y_pred, y)
  accuracy = conf_matrix[['overall']][['Accuracy']] * 100
  f1_score = conf_matrix[['byClass']][['F1']] * 100
  print(conf_matrix[["table"]])
  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
```

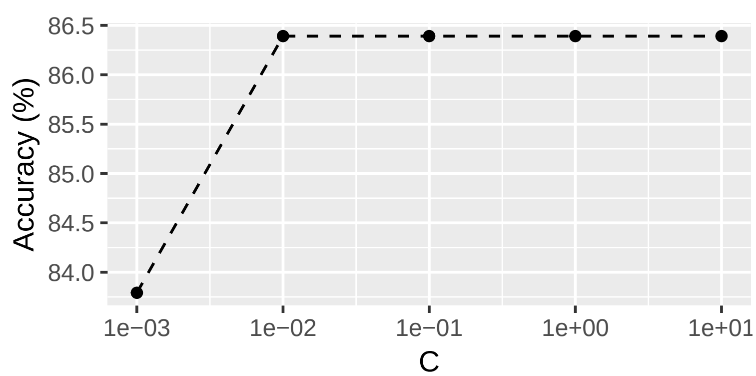
```
## Prediction   0   1
##          0 286  17
##          1  72 279
## Accuracy: 86.39 %    F1 Score: 86.54 %
```

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

```r
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 (%)')
```




```r
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 (%)')
```




```r
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')]
cat(sprintf('Best model with higest F1 Score of %.2f%% is model with C = %.6f\n',
            best_f1_score, best_c_params))
```

```
## Best model with higest F1 Score of 86.54% is model with C = 0.010000
```

```r
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 = break_string(equation, '+', 70)
cat(sprintf('Best model equation is:\n%s\n', equation))
```

```
## 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
```

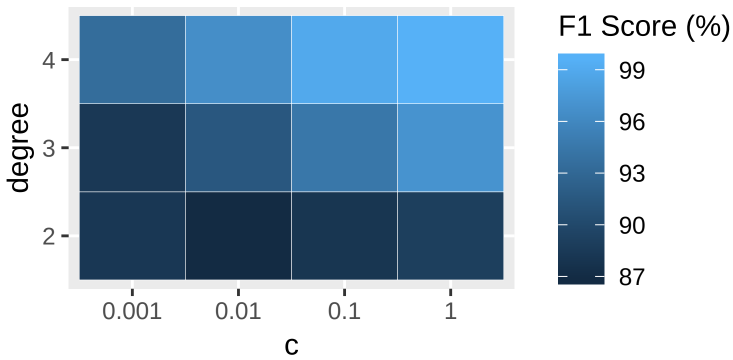
Question 2.2.2

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)
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 = 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 (%)')
```




```r
source('/home/alifahsanul/Documents/analytics_modelling/function_lib.R')
best_poly_model = best_model(poly_model_result, 'f1_score')
print('Best Polynomial Kernel Model:')
```

```
## [1] "Best Polynomial Kernel Model:"
```

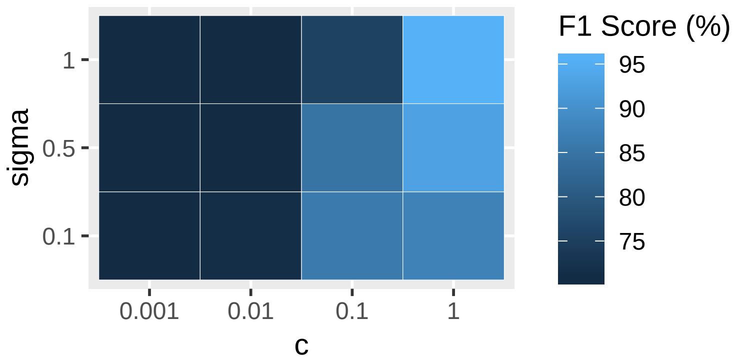
```r
print(best_poly_model, row.names = FALSE)
```

```
##  c degree accuracy f1_score     kernel
##  1      4 99.54128 99.58275 polynomial
```

RBF Kernel SVM

```r
sigma_params = c(0.1, 0.5, 1)
rbf_models = c()
conf_matrix = c()
rbf_model_result = NULL
for (sigma in sigma_params){
  for (c in c_params){
    my_svm_model = 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'
```

```r
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 (%)')
```




```r
source('/home/alifahsanul/Documents/analytics_modelling/function_lib.R')
best_rbf_model = best_model(rbf_model_result, 'f1_score')
print('Best RBF Kernel Model:')
```

```
## [1] "Best RBF Kernel Model:"
```

```r
print(best_rbf_model, row.names = FALSE)
```

```
##  c sigma accuracy f1_score kernel
##  1     1 95.10703 95.49296    rbf
```

The best model is

```r
library(gtools)
all_model_result = smartbind(modelling_result, poly_model_result, rbf_model_result)
best_svm_model = best_model(all_model_result, 'f1_score')
print('Best SVM model among 3 kernels (linear, polynomial, RBF):')
```

```
## [1] "Best SVM model among 3 kernels (linear, polynomial, RBF):"
```

```r
print(best_svm_model, row.names = FALSE)
```

```
##  c accuracy f1_score     kernel degree sigma
##  1 99.54128 99.58275 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