Models to Predict Qualified Credit_card_data Applicants

Find a good classifier to predict the qualified applicant

credit_card_data.txt contains 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.

```
# Clear environment
rm(list = ls())
# Load the kernlab library (contains the ksvm function)
library(kernlab)
data <- read.table("credit_card_data.txt", stringsAsFactors = FALSE, header = FALSE)
head(data)
```

```
##   V1    V2    V3   V4 V5 V6 V7 V8  V9 V10 V11
## 1  1 30.83 0.000 1.25  1  0  1  1 202   0   1
## 2  0 58.67 4.460 3.04  1  0  6  1  43 560   1
## 3  0 24.50 0.500 1.50  1  1  0  1 280 824   1
## 4  1 27.83 1.540 3.75  1  0  5  0 100   3   1
## 5  1 20.17 5.625 1.71  1  1  0  1 120   0   1
## 6  1 32.08 4.000 2.50  1  1  0  0 360   0   1
```

```
set.seed(1)
```

Fit the model using scaled=TRUE.

```
model_scaled <- ksvm(as.matrix(data[,1:10]),as.factor(data[,11]),
           type = "C-svc", # Use C-classification method
              kernel = "vanilladot", # Use simple linear kernel
              C = 100,
           scaled=TRUE) # have ksvm scale the data for you
```

```
##  Setting default kernel parameters
```

```
# alternation

model_scaled <- ksvm(V11~.,data=data,
              type = "C-svc", # Use C-classification method
              kernel = "vanilladot", # simple linear kernel
              C = 100,
           scaled=TRUE) # have ksvm scale the data
```

```
##  Setting default kernel parameters
```

```r
#attributes(model_scaled)
model_scaled
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 100
##
## Linear (vanilla) kernel function.
##
## Number of Support Vectors : 189
##
## Objective Function Value : -17887.92
## Training error : 0.136086
```

Calculating the a coefficients

```r
a_scaled <- colSums(model_scaled@xmatrix[[1]] * model_scaled@coef[[1]])
a0_scaled<- -model_scaled@b

a_scaled
```

```
##            V1            V2            V3            V4            V5
## -0.0010065348 -0.0011729048 -0.0016261967  0.0030064203  1.0049405641
##            V6            V7            V8            V9           V10
## -0.0028259432  0.0002600295 -0.0005349551 -0.0012283758  0.1063633995
```

```r
a0_scaled
```

```
## [1] 0.08158492
```

Calculating the predicted values

```r
predicted_scaled<-rep(0,nrow(data))

#For each data point, perform the transformation, calculate a*scaled(data point)+a0,
#and predict value of data point based on the resulting value

for (i in 1:nrow(data)){

  #If the data point is above the classifier, predicted value = 1

  if (sum(a_scaled*(data[i,1:10]-model_scaled@scaling$x.scale$`scaled:center`)/model_scaled@scaling$x.sc
    predicted_scaled[i] <- 1
  }

  #If the data point is below the classifier, predicted value = 0

  if (sum(a_scaled*(data[i,1:10]-model_scaled@scaling$x.scale$`scaled:center`)/model_scaled@scaling$x.sc
    predicted_scaled[i] <- 0
  }
}
predicted_scaled
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1
## [223] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1
## [260] 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
## [297] 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
## [334] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [371] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [408] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [445] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [482] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [519] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1
## [556] 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
## [593] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
## [630] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```r
pred_scaled <- predict(model_scaled,data[,1:10])
pred_scaled
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1
## [223] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1
## [260] 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
## [297] 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
## [334] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [371] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [408] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [445] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [482] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [519] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1
## [556] 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
## [593] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
## [630] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Calculating the model's accuracy

```r
sum(pred_scaled == data$V11) / nrow(data)
```

```
## [1] 0.8639144
```

```r
sum(predicted_scaled == data$V11) / nrow(data)
```

```
## [1] 0.8639144
```

Fit the model using scaled=FALSE

```
model_unscaled <- ksvm(V11~.,data=data,
              type = "C-svc", # Use C-classification method
              kernel = "vanilladot", # Use simple linear kernel
              C = 100,
           scaled=FALSE)
```

```
##  Setting default kernel parameters
```

```
a_unscaled <- colSums(model_unscaled@xmatrix[[1]] * model_unscaled@coef[[1]])

a0_unscaled <- -model_unscaled@b
predicted_unscaled<-rep(0,nrow(data))


for (i in 1:nrow(data)){

  #If the data point is above the classifier, predicted value = 1

  if (sum(a_unscaled*data[i,1:10]) + a0_unscaled >= 0){
    predicted_unscaled[i] <- 1
  }

  #If the data point is below the classifier, predicted value = 0

  if (sum(a_unscaled*data[i,1:10]) + a0_unscaled < 0){
    predicted_unscaled[i] <- 0
  }
}
pred_unscaled <- predict(model_unscaled,data[,1:10])
sum(pred_unscaled == data$V11) / nrow(data)
```

```
## [1] 0.7217125
```

```
sum(predicted_unscaled == data$V11) / nrow(data)
```

```
## [1] 0.7217125
```

kknn library

```
library(kknn)
```

```
## Warning: package 'kknn' was built under R version 4.0.2
```

```
data <- read.table("credit_card_data.txt", stringsAsFactors = FALSE, header = FALSE)
check_accuracy = function(X){
  predicted <- rep(0,(nrow(data))) # predictions: start with a vector of all zeros
```

```
  # for each row, estimate its response based on the other rows

  for (i in 1:nrow(data)){

    # data[-i] means we remove row i of the data when finding nearest neighbors...
    #...otherwise, it'll be its own nearest neighbor!

    model=kknn(V11~V1+V2+V3+V4+V5+V6+V7+V8+V9+V10,data[-i,],data[i,],k=X, scale = TRUE) # use scaled da

    # record whether the prediction is at least 0.5 (round to one) or less than 0.5 (round to zero)

    predicted[i] <- as.integer(fitted(model)+0.5) # round off to 0 or 1
  }

  # calculate fraction of correct predictions

  accuracy = sum(predicted == data[,11]) / nrow(data)
  return(accuracy)
}

# call the function for values of k from 1 to 20 (you could try higher values of k too)

acc <- rep(0,20) # set up a vector of 20 zeros to start
for (X in 1:20){
  acc[X] = check_accuracy(X) # test knn with X neighbors
}

# report accuracies
acc
```

```
##  [1] 0.8149847 0.8149847 0.8149847 0.8149847 0.8516820 0.8455657 0.8470948
##  [8] 0.8486239 0.8470948 0.8501529 0.8516820 0.8532110 0.8516820 0.8516820
## [15] 0.8532110 0.8516820 0.8516820 0.8516820 0.8501529 0.8501529
```

cross-validation for kknn

```
rm(list = ls())

library(kknn)

data <- read.table("credit_card_data.txt", stringsAsFactors = FALSE, header = FALSE)

set.seed(1)

# set maximum value of k (number of neighbors) to test

kmax <- 30

# use train.kknn for leave-one-out cross-validation up to k=kmax
```

```r
model <- train.kknn(V11~.,data,kmax=kmax,scale=TRUE)

# create array of prediction qualities

accuracy <- rep(0,kmax)

# calculate prediction qualities

for (k in 1:kmax) {
    predicted <- as.integer(fitted(model)[[k]][1:nrow(data)] + 0.5) # round off to 0 or 1
    accuracy[k] <- sum(predicted == data$V11)
}

accuracy
```

```
##  [1] 533 533 533 533 557 553 554 555 554 557 557 558 557 557 558 558 558 557 556
## [20] 556 555 554 552 553 553 552 550 548 549 550
```

cv.kknn from kknn package

```r
set.seed(1)

kmax <- 30
accuracy_cv <- rep(0,kmax)
for (k in 1:kmax) {

    # run cross-validation for each value of k (number of neighbors)
    model <- cv.kknn(V11~.,data,
                                        kcv=10, # 10-fold cross-validation
                                        k=k, # number of neighbors
                                        scale=TRUE) # scale data

    predicted <- as.integer(model[[1]][,2] + 0.5) # round off to 0 or 1
    accuracy_cv[k] <- sum(predicted == data$V11)
}

accuracy_cv
```

```
##  [1] 522 527 526 530 556 560 550 551 554 555 561 559 554 557 562 553 564 548 547
## [20] 539 551 556 551 548 547 549 542 551 554 548
```

caret package

```r
#install.packages("caret",dependencies = TRUE)
#install.packages("quantreg")
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.0.2

## Loading required package: lattice

## Loading required package: ggplot2

##
## Attaching package: 'ggplot2'

## The following object is masked from 'package:kernlab':
##
##     alpha

##
## Attaching package: 'caret'

## The following object is masked from 'package:kknn':
##
##     contr.dummy
```

```r
set.seed(1)

# set number of values of k (number of neighbors) to test, the default here is to try odd numbers, to a

kmax <- 15

knn_fit <- train(as.factor(V11)~V1+V2+V3+V4+V5+V6+V7+V8+V9+V10,
    data,
    method = "knn", # choose knn model
    trControl=trainControl(
                method="repeatedcv", # k-fold cross validation
                number=10, # number of folds (k in cross validation)
                repeats=5), # number of times to repeat k-fold cross validation
    preProcess = c("center", "scale"), # standardize the data
    tuneLength = kmax) # max number of neighbors (k in nearest neighbor)

knn_fit
```

```
## k-Nearest Neighbors
##
## 654 samples
##  10 predictor
##   2 classes: '0', '1'
##
## Pre-processing: centered (10), scaled (10)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 589, 589, 588, 589, 588, 589, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    5  0.8487010  0.6949665
##    7  0.8462302  0.6908286
```

```
##    9  0.8437778  0.6858642
##   11  0.8425848  0.6834138
##   13  0.8401321  0.6781877
##   15  0.8379783  0.6738320
##   17  0.8383240  0.6741098
##   19  0.8398578  0.6772901
##   21  0.8410417  0.6794564
##   23  0.8379783  0.6725165
##   25  0.8407522  0.6775478
##   27  0.8398432  0.6755254
##   29  0.8391904  0.6736047
##   31  0.8394984  0.6739680
##   33  0.8397968  0.6743131
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 5.
```

Training, Validation, and Test data sets

```
rm(list = ls())
library(kernlab)
library(kknn)
data <- read.table("credit_card_data.txt", stringsAsFactors = FALSE, header = FALSE)
set.seed(1)
mask_train = sample(nrow(data), size = floor(nrow(data) * 0.6))
cred_train = data[mask_train,] # training data set
remaining = data[-mask_train, ]  # all rows except training
mask_val = sample(nrow(remaining), size = floor(nrow(remaining)/2))
cred_val = remaining[mask_val,]  # validation data set
cred_test = remaining[-mask_val, ] # test data set

# pick the best of 9 SVM models and 20 KNN models

acc <- rep(0,29)  # 1-9 are SVM, 10-29 are KNN

#SVM models
# values of C to test

amounts <- c(0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000)

for (i in 1:9) {

    # fit model using training set

     model_scaled <- ksvm(as.matrix(cred_train[,1:10]),
              as.factor(cred_train[,11]),
           type = "C-svc", # Use C-classification method
              kernel = "vanilladot", # Use simple linear kernel
              C = amounts[i],
           scaled=TRUE) # have ksvm scale the data for you
```

```r
    #  compare models using validation set

            pred <- predict(model_scaled,cred_val[,1:10])
            acc[i] = sum(pred == cred_val$V11) / nrow(cred_val)
}
```

```
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
```

```r
acc[1:9]
```

```
## [1] 0.5725191 0.5725191 0.6946565 0.8778626 0.8778626 0.8778626 0.8778626
## [8] 0.8778626 0.8778626
```

```r
cat("Best SVM model is number ",which.max(acc[1:9]),"\n")
```

```
## Best SVM model is number  4
```

```r
cat("Best C value is ",amounts[which.max(acc[1:9])],"\n")
```

```
## Best C value is  0.01
```

```r
cat("Best validation set correctness is ",max(acc[1:9]),"\n")
```

```
## Best validation set correctness is  0.8778626
```

```r
     model_scaled <- ksvm(as.matrix(cred_train[,1:10]),
              as.factor(cred_train[,11]),
           type = "C-svc", # Use C-classification method
              kernel = "vanilladot", # Use simple linear kernel
              C = amounts[which.max(acc[1:9])],
           scaled=TRUE)
```

```
##  Setting default kernel parameters
```

```r
cat("Performance on test data = ",sum(predict(model_scaled,cred_test[,1:10]) == cred_test$V11) / nrow(c
```

```
## Performance on test data =  0.8625954
```

Train KNN models

```r
for (k in 1:20) {

               # fit k-nearest-neighbor model using training set, validate on test set

     knn_model <- kknn(V11~.,cred_train,cred_val,k=k,scale=TRUE)

    #  compare models using validation set

     pred <- as.integer(fitted(knn_model)+0.5) # round off to 0 or 1

     acc[k+9] = sum(pred == cred_val$V11) / nrow(cred_val)
}

acc[10:29]
```

```
##  [1] 0.7557252 0.7557252 0.7557252 0.7557252 0.8167939 0.8244275 0.8167939
##  [8] 0.8167939 0.8167939 0.8396947 0.8396947 0.8396947 0.8396947 0.8396947
## [15] 0.8396947 0.8396947 0.8396947 0.8396947 0.8396947 0.8396947
```

```r
# find best-performing KNN model on validation data

cat("Best KNN model is k=",which.max(acc[10:29]),"\n")
```

```
## Best KNN model is k= 10
```

```r
cat("Best validation set correctness is ",max(acc[10:29]),"\n")
```

```
## Best validation set correctness is  0.8396947
```

```r
# run best model on test data

     knn_model <- kknn(V11~.,cred_train,cred_test,
               k=which.max(acc[10:29]),
               scale=TRUE)

     pred <- as.integer(fitted(knn_model)+0.5) # round off to 0 or 1

cat("Performance on test data = ",sum(pred == cred_test$V11) / nrow(cred_test),"\n")
```

```
## Performance on test data =  0.8778626
```

```r
# Evaluate overall best model on test data
#

if (which.max(acc) <= 9)  {        # if a ksvm method is best

         # evaluate the ksvm method on the test set to find estimated quality

         cat("Use ksvm with C = ",amounts[which.max(acc[1:9])],"\n")
         cat("Test performace = ",sum(predict(model_scaled,cred_test[,1:10]) == cred_test$V11) / nrow(c
```

```r
} else {    # the best is a knn method

        # evaluate the knn method on the test set to find estimated quality

        cat("Use knn with k = ",which.max(acc[10:29]),"\n")
        cat("Test performance = ",sum(pred == cred_val$V11) / nrow(cred_val),"\n")

}
```

```
## Use ksvm with C =  0.01
## Test performace =  0.8625954
```