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)

Table of Contents

Finding Support Vector Machine Classifier in R

```
library("kernlab")
```

- Setting the library

Set directory by going to Session < Set Working Directory < Choose Directory < Select folder that contains the dataset

```
> CCdata <- read.table("credit_card_data.txt", header = FALSE) # read in data
> head(CCdata) # view first 6 rows
  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
```

- Reading in the credit card dataset without headers and viewing the first 6 rows.

```
CCmodel <- ksvm(V11~V1+V2+V3+V4+V5+V6+V7+V8+V9+V10, data = CCdata,
type = "C-svc", kernel = "vanilladot", C = 10, scaled = TRUE) # runs
the model
```

- **V11~V1+V2+V3+V4+V5+V6+V7+V8+V9+V10** defines the relationship between the target variable (V11) and the predictor variables (V1 – V10).
- **data = CCdata** is the dataset that holds the V1 – V11 variables.
- **type = "C-svc"** is the most common and default type for classification tasks.
- **kernel = "vanilladot"** takes the data to a higher dimension to be separated by a hyperplane.
- **C = 10** is the fitting of the training data
- **scaled = TRUE** standardizes the ranges of the V1 – V10 variables

```
a <- colSums(CCmodel@xmatrix[[1]]*CCmodel@coef[[1]]) # calculating
coefficients
```

- Viewed another way colSums(support matrix * provided coefficients for the support vectors)
- This code gives us the "a" values for the equation of our classifier.
- The following is a table of all of the coefficients that were generated:

Column	Coefficient
V1	-3.020625
V2	18.107857
V3	3.819122
V4	25.555087
V5	30.451923
V6	-12.600424
V7	14.307075
V8	-10.225388
V9	-32.753620
V10	35.483505

```
a0 <- -CCmodel@b # calculates y-intercept
```

- This gave us a y-intercept of 0.4453205.

This gives us the equation for our classifier which is: $f(x) = -3.020625\mathbf{X_1} + 18.107857\mathbf{X_2} + 3.819122\mathbf{X_3} + 25.555087\mathbf{X_4} + 30.451923\mathbf{X_5} - 12.600424\mathbf{X_6} + 14.307075\mathbf{X_7} - 10.225388\mathbf{X_8} - 32.753620\mathbf{X_9} + 35.483505\mathbf{X_{10}} + 0.4453205$

```
1-CCmodel@error
```

- 100% - error % = accuracy
- The accuracy of our SVM model with a fitting to the training data of C = 10 to the training data come out to 86.39%

I brute forced C values of 1, 100, and 1000 to find the optimal C value for our data.

```
CCmodel <- ksvm(V11~V1+V2+V3+V4+V5+V6+V7+V8+V9+V10, data = CCdata,
type = "C-svc", kernel = "vanilladot", C = 1, scaled = TRUE)
```

- Reset the model to have a C value of 1.

```
1-CCmodel@error
```

- Re-ran the test for model accuracy.
- The output came out the same accuracy of 86.39%

I did the same thing for C values of 100 and 1000 and received the same accuracy of 86.39%. This is because our training data is the same as our test data.

Model Training and Evaluation Using K-Nearest Neighbors in R

With Cross Validation

```
rm(list = ls())
```

- Clearing environment

```
library(caret)
```

- Loading libraries

```
set.seed(916)
```

- Setting the seed for the random number generator to ensure reproducibility. This is a good thing to do for training a model. However, it should be taken off when the model is deployed in production.

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

- Read in data

```
train_control <- trainControl(method = "cv", number = 10)
```

- the **trainControl()** function is used in caret to define the training parameters
- **method = "cv"** means that we want to use cross-validation.
- **number = 10** means we want to cross-validate across 10 folds. The data will be split into 10 subsets. We will validate on 1 subset/fold and train on the remaining 9, then rinse and repeat until all folds have been used to create the model

```
knn_model <- train(x = as.matrix(data[, 1:10]), y = as.factor(data[,
11]), method = "knn", trControl = train_control, preProcess =
c("scale", "center"), tuneLength = 10)
```

- **train()** is a function for training machine learning models.
- **x = as.matrix(data[, 1:10])** is the independent variable for the model, selecting the first 10 columns of the data in matrix form
- **y = as.factor(data[, 11])** is the dependent variable, selecting the 11^{th} column as the target variable and converting into a factor
- **method = "knn"** specifying that we want to train a k-Nearest Neighbors model.
- **trControl = train_control** defines the training parameters set in the previous code
- **preProcess = c("scale", "center"):** scales and centers the data which is important since this model heavily relies on distances between points.
- **tuneLength = 10** the function will try 10 different values of k

```
print(knn_model$results)
```

```
> print(knn_model$results)
    k  Accuracy     Kappa AccuracySD    KappaSD
1   5 0.8426107 0.6830588 0.03704017 0.07379338
2   7 0.8410490 0.6800920 0.03873323 0.07883006
3   9 0.8410256 0.6806526 0.03217553 0.06466735
4  11 0.8378788 0.6738681 0.03028255 0.06119202
5  13 0.8287879 0.6554494 0.03578361 0.07228118
6  15 0.8365268 0.6701936 0.04215056 0.08604562
7  17 0.8396037 0.6766237 0.03767107 0.07571858
8  19 0.8395571 0.6767842 0.03417568 0.06840024
9  21 0.8395338 0.6765335 0.03040332 0.06081492
10 23 0.8333566 0.6635270 0.02736266 0.05568209
```

- A good classifier is **k = 5** with the highest accuracy of 84.26%.

With Train/Validation/Test Splits

```
rm(list = ls())
library(kernlab)
set.seed(4)
data <- read.table("credit_card_data.txt", header = FALSE)
head(data)
```

```
> 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
```

```
randomized_data <- data[sample(nrow(data)),]
```

- Creating a new data set where all of the rows are randomized. This will help to randomize the training, validation, and test data.

```
head(randomized_data)
```

```
> head(randomized_data)
    V1    V2     V3    V4 V5 V6 V7 V8  V9 V10 V11
504  0 24.58  0.670 1.750  1  1  0  1 400   0   0
587  1 29.58  4.750 2.000  0  0  1  0 460  68   0
71   1 32.33  7.500 1.585  1  1  0  0 420   0   0
371  1 26.67 14.585 0.000  0  1  0  0 178   0   0
307  1 36.75  0.125 1.500  0  1  0  0 232 113   1
312  1 33.67  1.250 1.165  0  1  0  1 120   0   0
```

- The first 6 rows are now 504,587,71,371,307 and 312.

```
data_train <- randomized_data[1:392,]

data_valid <- randomized_data[393:523,]

data_test <- randomized_data[524:654,]
```

- If we did not randomize the data in the earlier step then these data sets would have had significant bias and would not have been representative of the whole data set.

```
model_SVM <- ksvm(as.matrix(data_train[,1:10]),
as.factor(data_train[,11]), type = "C-svc", kernel = "vanilladot",
C=10, scaled = TRUE)
```

- **as.matrix(data_train[,1:10]):** selects the columns that the data will be trained on and converts to matrix format
- **as.factor(data_train[,11]):** selects the target variable that we want to predict and converts to a factor
- **type = "C-svc":** the most common and default type for classification tasks.
- **kernel = "vanilladot":** takes the data to a higher dimension to be separated by a hyperplane.
- **C=10:** the fitting of the training data
- **scaled = TRUE:** standardizes the data

```
SVM_valid_pred <- predict(model_SVM,data_valid[,1:10])
```

- This takes the same columns we used to test the model except on the validation data set this time uses the target variable to see if the predictions were correct or incorrect.

```
SVM_valid_Accuracy <- sum(SVM_valid_pred == data_valid[,11]) /
length(SVM_valid_pred)
```

- This sums up the number of correct predictions and divides it by the total predictions to give us an accuracy %.

```
print(SVM_valid_Accuracy)

[1] 0.8854962
```

- We received an 88.55% accuracy with the vanilladot kernel. Now I am going to same thing but on different kernels. Here are the results:

Kernel	Accuracy
vanilladot	88.55%
rbfdot	83.21%
polydot	88.55%
tanhdot	78.63%
laplacedot	87.79%
besseldot	78.63%
anovadot	84.73%

- I selected the vanilla dot kernel and then ran the model against the test data.

```
SVM_test_pred <- predict(model_SVM,data_test[,1:10])

SVM_test_Accuracy <- sum(SVM_test_pred == data_test[,11]) /
length(SVM_test_pred)

print(SVM_test_Accuracy)

[1] 0.8778626
```

- It has a difference of 0.76% which suggests that the model does a good job generalizing. If the number were 5-10% then that would suggest that model is too optimized for the validation data and doesn't generalize well to test data.

Using the GermanCredit data set germancredit.txt from http://archive.ics.uci.edu/ml/machine-learning-databases/statlog/german / (description at http://archive.ics.uci.edu/ml/datasets/Statlog+%28German+Credit+Data%29),

Predicting Credit Risk Using Logistic Regression in R

```
rm(list = ls())
```

- Clearing the environment

```
germancredit <- read.table("germancredit.txt", sep = " ")
```

- Reading in the german credit data set

```
head(germancredit)
```

```
   V1 V2  V3  V4   V5  V6  V7 V8  V9  V10 V11
1 A11  6 A34 A43 1169 A65 A75  4 A93 A101   4
2 A12 48 A32 A43 5951 A61 A73  2 A92 A101   2
3 A14 12 A34 A46 2096 A61 A74  2 A93 A101   3
4 A11 42 A32 A42 7882 A61 A74  2 A93 A103   4
5 A11 24 A33 A40 4870 A61 A73  3 A93 A101   4
6 A14 36 A32 A46 9055 A65 A73  2 A93 A101   4
   V12 V13  V14  V15 V16  V17 V18  V19  V20
1 A121  67 A143 A152   2 A173   1 A192 A201
2 A121  22 A143 A152   1 A173   1 A191 A201
3 A121  49 A143 A152   1 A172   2 A191 A201
4 A122  45 A143 A153   1 A173   2 A191 A201
5 A124  53 A143 A153   2 A173   2 A191 A201
6 A124  35 A143 A153   1 A172   2 A192 A201
  V21
1   1
2   2
3   1
4   1
5   2
6   1
```

- We have a lot of categorical predictors as they go all the way up to A201. Also notice that the V21 column is a binary variable. We should set that to 0 and 1 instead of 1 and 2.

```
germancredit$V21[germancredit$V21 == 1] <- 0
```

```
germancredit$V21[germancredit$V21 == 2] <- 1
```

- Setting the binary values in V21 to 0 and 1.

```
head(germancredit)
```

```
   V1 V2  V3  V4   V5  V6  V7 V8  V9  V10 V11
1 A11  6 A34 A43 1169 A65 A75  4 A93 A101   4
2 A12 48 A32 A43 5951 A61 A73  2 A92 A101   2
3 A14 12 A34 A46 2096 A61 A74  2 A93 A101   3
4 A11 42 A32 A42 7882 A61 A74  2 A93 A103   4
5 A11 24 A33 A40 4870 A61 A73  3 A93 A101   4
6 A14 36 A32 A46 9055 A65 A73  2 A93 A101   4
   V12 V13  V14  V15 V16  V17 V18  V19  V20
1 A121  67 A143 A152   2 A173   1 A192 A201
2 A121  22 A143 A152   1 A173   1 A191 A201
3 A121  49 A143 A152   1 A172   2 A191 A201
4 A122  45 A143 A153   1 A173   2 A191 A201
5 A124  53 A143 A153   2 A173   2 A191 A201
6 A124  35 A143 A153   1 A172   2 A192 A201
  V21
1   0
2   1
3   0
4   0
5   1
6   0
```

- Perfect, the binary values changed to 0 and 1.

```
set.seed(916)
```

- Setting seed for reproducability

```
sample_gc <- sample(1:nrow(germancredit), size = 800)
```

- Randomly selecting 800 data points for our training set

```
germancredit_train <- germancredit[sample_gc, ]
```

- This will be our training set of 800 random samples

```
germancredit_test <- germancredit[-sample_gc, ]
```

- The remaining samples will be used for pur test set

```
germancredit_model <- glm(V21~., family = binomial(link =
"logit"), data = germancredit_train)
```

- Creating our logistic regression model.
- glm(): we are fitting our logistic regression model using the generalized linear framework (GLM)
- V21~.: this is our dependent variable that we are setting against all the other variables.
- family = binomial(link = “logit”): the family is the type of generalized linear model which is the logistic regression that takes on binary outcomes (true/false, yes/no, 0/1).

```
summary(germancredit_model)
```

```
Coefficients:
               Estimate Std. Error z value
(Intercept) -6.608e-01  1.332e+00  -0.496
V1A12       -6.371e-02  2.485e-01  -0.256
V1A13       -8.470e-01  3.924e-01  -2.158
V1A14       -1.810e+00  2.682e-01  -6.747
V2           2.938e-02  1.088e-02   2.700
V3A31        2.549e-01  6.009e-01   0.424
V3A32       -6.275e-01  4.744e-01  -1.323
V3A33       -7.951e-01  5.405e-01  -1.471
V3A34       -1.544e+00  4.811e-01  -3.208
V4A41       -1.687e+00  4.324e-01  -3.901
```

```
             Pr(>|z|)
(Intercept)  0.61976
V1A12        0.79765
V1A13        0.03090 *
V1A14       1.51e-11 ***
V2           0.00693 **
V3A31        0.67148
V3A32        0.18593
V3A33        0.14130
V3A34        0.00134 **
V4A41       9.57e-05 ***
V4A410       0.11862
V4A42        0.03009 *
V4A43        0.00675 **
V4A44        0.57880
```

```
Signif. codes:
0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken t
o be 1)

    Null deviance: 973.97  on 799  degrees of fre
edom
Residual deviance: 695.02  on 751  degrees of fre
edom
AIC: 793.02

Number of Fisher Scoring iterations: 14
```

- All looks good, we see some of our variables hold significance.

```
yhat <- predict(germancredit_model, germancredit_test, type =
“response”)
```

- This line of code is for making predictions using the test data set and our logistic regression model we trained.
- predict() is used to make predictions using our germancredit_model on the test data germancredit_test.
- type = "response" to make our output more intuitive. An output of 0.73 means a 73% probability of the event occurring. The default type is log-odds and log-odds of 1 means that there is a 73% probability of the event occurring which isn’t very intuitive.

Finding Threshold Value

```
library(pROC)
```

- This library is Prediction Analysis for ROC Curves. ROC curves tell us how well a model performs across different threshold values.

```
roc(germancredit_test$V21, round(yhat))
```

- **roc()** is used to created the Receiver Operating Characteristic (ROC) curve.
- **germancredit_test$V21** is the target variable and contains the true, actual, outcome values.
- **round(yhat)** contains our predicted probabilities. 0.5 is rounded to 1 and less than 0.5 is rounded to 0.

```
Call:
roc.default(response = germancredit_test$V21, predictor = round(yhat))

Data: round(yhat) in 138 controls (germancredit_test$V21 0) < 62 cases (germa
ncredit_test$V21 1).
Area under the curve: 0.6542
```

- The area under the curve of 0.6542 tells us that our predictions are 65.42% correct. A value of 1 would mean that it is perfect and a value of 0.5 tells us that it is no better than randomly guessing.

```
threshold <- 0.8
```

- This will determine the cutoff point for determining if a value gets thrown into the 1 or positive class.

```
yhat_threshold <- as.integer(yhat > threshold)
```

- This code applies the threshold to the predicted probability.
- yhat > threshold checks if the predicted probability (yhat) is greater than the threshold and return a 1 or 0

```
conf_matrix <- as.matrix(table(yhat_threshold,
germancredit_test$V21))
```

- This is the code for our confusion matrix which is a table that will display values that were labeled correctly and values that were labeled incorrectly. The outputs will give us true positive, true negatives, false positives, and false negatives.
- as.matrix(table(yhat_threshold, germancredit_test$V21)) creates a table of our predicted probabilities and our actual values.

```
conf_matrix
```

```
yhat_threshold   0   1
             0 138  55
             1   0   7
```

- True Positives: 138
- True Negatives: 7
- False Negatives: 55
- False Positives: 0

Testing out different threshold values:

```
> threshold <- 0.8
> yhat_threshold <- as.integer(yhat > threshold)
> conf_matrix <- as.matrix(table(yhat_threshold, germancredit_test$V21))
> conf_matrix

yhat_threshold   0   1
             0 138  55
             1   0   7
```

```
> threshold <- 0.7
> yhat_threshold <- as.integer(yhat > threshold)
> conf_matrix <- as.matrix(table(yhat_threshold, germancredit_test$V21))
> conf_matrix

yhat_threshold   0   1
             0 133  50
             1   5  12
```

```
> threshold <- 0.6
> yhat_threshold <- as.integer(yhat > threshold)
> conf_matrix <- as.matrix(table(yhat_threshold, germancredit_test$V21))
> conf_matrix

yhat_threshold   0   1
             0 128  42
             1  10  20
```

```
> threshold <- 0.5
> yhat_threshold <- as.integer(yhat > threshold)
> conf_matrix <- as.matrix(table(yhat_threshold, germancredit_test$V21))
> conf_matrix

yhat_threshold   0   1
             0 116  33
             1  22  29
```

```
> threshold <- 0.4
> yhat_threshold <- as.integer(yhat > threshold)
> conf_matrix <- as.matrix(table(yhat_threshold, germancredit_test$V21))
> conf_matrix

yhat_threshold   0   1
             0 109  27
             1  29  35
```

```
> threshold <- 0.3
> yhat_threshold <- as.integer(yhat > threshold)
> conf_matrix <- as.matrix(table(yhat_threshold, germancredit_test$V21))
> conf_matrix

yhat_threshold  0  1
             0 93 22
             1 45 40
```

- I tried out a bunch of different threshold values. The best threshold for prediction was 0.6 at 74%. However, the question states that "that incorrectly identifying a bad customer as good [false positive], is 5 times worse than incorrectly classifying a good customer as bad.[false negative]"
    - The bottom left value in our tables indicate the false positive. Notice how in a threshold of 0.3 the false positive is 45 and it gradually decreases until we get to a false positive of 0 in the threshold of 0.8. This makes the best threshold to use 0.8.