HW #1

Code ▾

Question 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.

In my day job I work in the insurance industry where classification models are used frequently. Most commonly classification models are used to predict how risky a prospective insurance policy is. For example, say that you are creating a model that is classifying customers who want an auto insurance policy. The model classifies them into one of five predefined groups: very risky, risky, average, safe, very safe. Some of the predictors used for this model could be traffic citations history, car make/model, safety rating of car, claim frequency and others.

Question 2.1

SVM Classifier for Credit Card Data

Hide

```
# Loading kernlab package
require(kernlab)
```

```
Loading required package: kernlab
```

Hide

```
# Reading in data from txt file URL
data <- read.table("https://d37djvu3ytnwxt.cloudfront.net/assets/cour
seware/v1/39b78ff5c5c28981f009b54831d81649/asset-v1:GTx+ISYE6501x+2T2
017+type@asset+block/credit_card_data-headers.txt", header = TRUE)
# Transforming dataframe to matrix
data <- as.matrix(data)
```

Hide

```
# C = .01
ksvm(data[ , 1:10], data[,11], type="C-svc", kernel= "vanilladot", C=
.01, scaled=TRUE)
```

```
 Setting default kernel parameters
Support Vector Machine object of class "ksvm"
```

```
SV type: C-svc  (classification) 
 parameter : cost C = 0.01 

Linear (vanilla) kernel function. 

Number of Support Vectors : 288 

Objective Function Value : -2.2926 
Training error : 0.136086 
```

Hide

```
# C = .1
ksvm(data[ , 1:10], data[,11], type="C-svc", kernel= "vanilladot", C=
.1, scaled=TRUE)
```

```
 Setting default kernel parameters  
Support Vector Machine object of class "ksvm" 

SV type: C-svc  (classification) 
 parameter : cost C = 0.1 

Linear (vanilla) kernel function. 

Number of Support Vectors : 197 

Objective Function Value : -18.3976 
Training error : 0.136086 
```

Hide

```
# C = 1
ksvm(data[ , 1:10], data[,11], type="C-svc", kernel= "vanilladot", C=
1, scaled=TRUE)
```

```
 Setting default kernel parameters  
Support Vector Machine object of class "ksvm" 

SV type: C-svc  (classification) 
 parameter : cost C = 1 

Linear (vanilla) kernel function. 

Number of Support Vectors : 190 
```

```
Objective Function Value : -179.385
Training error : 0.136086
```

Hide

```
# C = 10
ksvm(data[ , 1:10], data[,11], type="C-svc", kernel= "vanilladot", C=
10, scaled=TRUE)
```

```
 Setting default kernel parameters
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 10

Linear (vanilla) kernel function.

Number of Support Vectors : 190

Objective Function Value : -1789.252
Training error : 0.136086
```

Hide

```
# C = 100
ksvm(data[ , 1:10], data[,11], type="C-svc", kernel= "vanilladot", C=
100, scaled=TRUE)
```

```
 Setting default kernel parameters
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
```

Hide

```
# C = 500
```

```
ksvm(data[ , 1:10], data[,11], type="C-svc", kernel= "vanilladot", C=
500, scaled=TRUE)
```

```
 Setting default kernel parameters
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 500

Linear (vanilla) kernel function.

Number of Support Vectors : 193

Objective Function Value : -89338.75
Training error : 0.136086
```

It appears that the training accuracy doesn't change much with a change in C.

Without knowing more about the data or the bank it's difficult to pick a C value. No matter which value you choose, the model won't "improve" much- if the yardstick for improvement is model accuracy. Instead it would be best to dive further into the data to get a better understanding of how the C value would affect the model going forward as well as have a talk with the bank about what their goal for the model is.

Digging further into the data would be doing some exploratory data analysis. Looking into distribution of variables and the error of various models. If the data is highly skewed or the response classes imbalanced it could throw off the model. Likewise if the bank doesn't care about the occasional bad apple being approved as long as 3x more applicants are approved the C value should be very small. On the other end of the spectrum if the bank's finances are rocky and they don't have an excess of capital they may want to be more stringent with their approved loans and only give out loans to people they're sure will pay the loans back. In this case the C value should be set to something high.

Since we don't know any of that, I'll keep it at the default setting (1)

Hide

```
# creating our model instance variable
model <- ksvm(data[ , 1:10], data[,11], type="C-svc", kernel= "vanill
adot", C=1, scaled=TRUE)
```

```
 Setting default kernel parameters
```

Hide

```
# Our variables and fitted coefficients
a <- colSums(model@xmatrix[[1]] * model@coef[[1]])
print("a =")
```

```
[1] "a ="
```

Hide

```
print(a)
```

```
           A1            A2            A3            A8            A9 
A10 
-0.0011026642 -0.0008980539 -0.0016074557  0.0029041700  1.0047363456 
-0.0029852110 
          A11           A12           A14           A15 
-0.0002035179 -0.0005504803 -0.0012519187  0.1064404601 
```

Hide

```
# Calculating a0
a0 <- -model@b
print("a0 =")
```

```
[1] "a0 ="
```

Hide

```
print(a0)
```

```
[1] 0.08148382
```

Hide

```
# Running the above code should print out the SVM formula
```

Hide

```
# predicitions using the svm model on the dataset
pred <- predict(model, data[,1:10])
print("pred =")
```

```
[1] "pred ="
```

Hide

```
print(pred)
```

```
  [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 1 1 1 1 1 1 1
 [45] 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 1 1
1 1 1 1 1 1 1 1 1 1 1 1
 [89] 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 1 1 1 1 1 1 1
[133] 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 1 1 1 1 1 1 1 1 1 1
[177] 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 1 1 1 1 1 1 1
[221] 0 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 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0
[265] 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
0 0 0 0 0 0 1 0 1 0 0 0
[309] 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 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
[353] 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 0 0 0 0 0 0 0
[397] 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 0 0 0 0 0 0 0
[441] 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 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1
[485] 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 1 1 1 1 1 1 1
[529] 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 1 1 1 1 1
1 1 1 1 0 1 1 1 1 1 1 0
[573] 0 1 0 0 0 0 0 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 0 0 0 0
[617] 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 0 0 0 0 0
0 0 0 0 0 0
```

Hide

```
# Summing the matches between what our model predicted and the correc
t label
# Our simple accuracy (correct predictions/total predictions)
print("Prediction % =")
```

```
[1] "Prediction % ="
```

Hide

```
print(sum(pred == data[,11]) / nrow(data))
```

```
[1] 0.8639144
```

Question 2.2

KNN Model

Hide

```
# kknn package
require(kknn)
```

```
Loading required package: kknn
```

Hide

```
# kknn requires data as a dataframe and not matrix
# Reloading data
df <- read.table("https://d37djvu3ytnwxt.cloudfront.net/assets/course
ware/v1/39b78ff5c5c28981f009b54831d81649/asset-v1:GTx+ISYE6501x+2T201
7+type@asset+block/credit_card_data-headers.txt", header = TRUE)
```

Hide

```
# Training a k-nearest neighbours on the whole dataset (no train/test
/validation split)
# Scaled data with a max k value of 100 testing a variety of kernel o
ptions
# I believe train.kknn takes care of removing the data point as a "ne
ighbour" itself
knn_model <- train.kknn(R1 ~ ., data = df, scaled=TRUE, kmax = 100,
           kernel = c("optimal", "triangular", "rectangular",
           "epanechnikov","cos","inv", "gaussian","triweight","biweig
ht"))
summary(knn_model)
```

```


Call:
train.kknn(formula = R1 ~ ., data = df, kmax = 100, kernel = c("optim
al",     "triangular", "rectangular", "epanechnikov", "cos", "inv",
"gaussian", "triweight", "biweight"), scaled = TRUE)
```

```
Type of response variable: continuous
minimal mean absolute error: 0.1850153
Minimal mean squared error: 0.1061155
Best kernel: inv
Best k: 22
```

Predicting the acceptance or rejection of loan candidates with our model Rounding the predictions to match our binary labels (1 | 0)

Hide

```
pred_knn <- round(predict(knn_model, df[,1:10]))
```

Now to sum the matches between what our model predicted and the correct label This gives us our simple accuracy (correct predictions/total predictions)

Hide

```
print("Prediction % =")
```

```
[1] "Prediction % ="
```

Hide

```
print(sum(pred_knn == df[,11]) / nrow(df))
```

```
[1] 1
```

This returns 100% accuracy

Not too surprising given that we used the entire dataset as training as well as looked for the absolute best fit for a model by using many different kernels and a high max k value

In reality this model is likely ridiculously overfit and would not be very useful