2.1: One real-life classification example that I have run into is determining whether a person is going to pass pilot training. Some classifiers included GPA (whether they have the ability to finish the written portion of the training), pilot battery test scores (whether they have the aptitude flying-related tasks like target tracking), and physical fitness test scores (whether they will physically be able to withstand the rigors of flying).

2.2:

Equation using C=100:

Y= -0.08158492 + (-0.0010065348*A1) +(-0.0011728048*A2) + (-0.0016261967*A3) + (0.0030064203*A8) + (1.0049405641*A9) + (-0.0028259432*A10) + (0.002600295*A11) + (-0.0005349551*A12) + (-0.0012283578*A14) + (0.1063633995*A15)

Accuracy: 0.8639

Varying C from 1 to 10000 didn't cause much impact, and it didn't force all of my predicted values to 0 or 1. Making C super small (.0000001) made the accuracy much worse at 0.5474. C=1000000 made my computer work much harder and only put the accuracy at 0.6254.

2.3

I tested both Besseldot and RBFdot. Besseldot gave me an accuracy of 0.925. RBFdot gave me accuracy of 0.951. I left the C value at 100 for both of these models.

2.4

Using the kknn function, I created a new function that used a for loop and kknn with data[-i,] and data[i,] to omit the current row from the calculation. I scaled the data. I used k=N, with N being the input of my function so I could later run the function and vary N as a way to find the best k value. I then put the predicted value into a vector.

I ran my function to test the best k values using N= 0 to 50, and I found, using the max() and which.max() functions, that k=12 has the highest accuracy at 0.853211.

3.1.a

Using cross-validation and k nearest neighbors:

I used the train.kknn model, and I broke my data into 75% training and 25% testing. I gave the model multiple kernel options and it gave back that "biweight" was my best kernel, with my best k as 69. This gave me an accuracy of 0.847561.

---

3. 1.b

I used KVSM for this section. First I split the data into 60% training, 20% validation, and 20% testing. I used the training data to find 3 models using 3 different kernels (vanilladot, besseldot, and rbfdot). I then applied the 3 models to my validation data, and I got the highest accuracy with vanilladot (0.8931), with rbfdot next best (0.8321) and besseldot last (0.8168). Using the testing data and the best model (vanilladot), I found the accuracy to be 0.8779.

The equation for my best model would be:

Y=-.1274082+(-0.0026496445*A1) + (-0.0096544691 * A2) + (0.0007262427*A3) + (0.9997711230*A9) + (-0.0055269705*A10) + (0.0055952231*A11) + (0.0033354734*A12) + (-0.0092123644*A14) + (0.1260694356*A15)

CODE:

2-1: KVSM using all data to train the model (varied C manually)

```
require(kernlab)
myData <- read.table("c:/users/kkisner/Desktop/gradclass/creditCard.txt", header = TRUE)
set.seed(10)

#assign x and y just to keep my model function cleaner
x <- as.matrix(myData[,1:10])
y <- as.factor(myData[,11])



# call ksvm. Vanilladot is a simple linear kernel.
model <- ksvm(x, y, type="C-svc", kernel='vanilladot', C=100, scaled=TRUE)

# calculate a1…am
a <- colSums(model@xmatrix[[1]] * model@coef[[1]])
a
# calculate a0
a0 <- model@b
a0
# see what the model predicts
pred <- predict(model,myData[,1:10])
pred
# see what fraction of the model's predictions match the actual classification
sum((pred == (myData[,11])) / nrow(myData))
```

2-2: KVSM using non-linear kernels

```
require(kernlab)
myData <- read.table("c:/users/kkisner/Desktop/gradclass/creditCard.txt", header = TRUE)
set.seed(10)

#assign x and y just to keep my model function cleaner
x <- as.matrix(myData[,1:10])
y <- as.factor(myData[,11])



# call ksvm using Besseldot.  Also tried rbfdot
model <- ksvm(x,y, type="C-svc", kernel='besseldot', C=100, scaled=TRUE)

# calculate a1…am
a <- colSums(model@xmatrix[[1]] * model@coef[[1]])
a
# calculate a0
a0 <- model@b
a0
# see what the model predicts
pred <- predict(model,myData[,1:10])
pred
# see what fraction of the model's predictions match the actual classification
sum((pred == (myData[,11])) / nrow(myData))
```

2.3 Using kknn

```
require(kknn)
myDataNH <- read.table("c:/users/kkisner/Desktop/gradclass/creditCardnoHeader.txt", header = FALSE)
```

```
set.seed(10)

#build a function that will test different values of k
check_k = function(N){

#build a vector to hold the predicted values
predknn <- rep(0, (nrow(myDataNH)))

#kknn model that loops through every observation
 for( i in 1:nrow(myDataNH)){
  knnmodel = kknn(V11 ~ V1+V2+V3+V4+V5+V6+V7+V8+V9+V10,
                  myDataNH[-i,],
                  myDataNH[i,],
                  k=N,
                  scale = TRUE)
  predknn[i] <- as.integer((fitted(knnmodel)+0.5))
 }

accuracy = sum(predknn == myDataNH[,11])/nrow(myDataNH)
return(accuracy)
}

#now run the function to loop through the k values
Accuracy = rep(0,50)
for(N in 1:50){
 Accuracy[N] = check_k(N)
}
```

```
#identify the max accuracy level and which observation has that level

max(Accuracy)

which.max(Accuracy)
```

3.1.a KKNN with cross-validation

```
rm(list = ls())


require(kknn)

myDataNH <- read.table("c:/users/kkisner/Desktop/gradclass/creditCardnoHeader.txt", header = FALSE)

set.seed(10)


# Count number of rows in file

rowCount = nrow(myDataNH)


# Randomly select 1/4 of the indexes from the number of rows

SampleIndex = sample(1:rowCount, size = round(rowCount/4), replace = FALSE)


#training data set selected by leaving out 1/4 of data

trainData = myDataNH[-SampleIndex,]

#test data set by including the other 3/4 of data

testData = myDataNH[SampleIndex,]


# Train the model – gave it lots of kernel options so it could tell me the best one

CVknnmodel = train.kknn(V11 ~. ,  data = trainData, kmax=100, kernel = c("rectangular", "triangular",
"biweight","triweight","cos", "inv", "gaussian", "optimal"), scale = TRUE)

CVknnmodel


#Test the model on test data set

predCV <- predict(CVknnmodel, testData)
```

```
pred_bin <- round(predCV)
CVaccuracy <- table(pred_bin, testData$V11)
CVaccuracy

#Show the prediction accuracy
sum(pred_bin == testData$V11)/length(testData$V11)
```

3.1.b KVSM using training, validation, and testing data sets

```
require(kernlab)
myData <- read.table("c:/users/kkisner/Desktop/gradclass/creditCard.txt", header = TRUE)

set.seed(10)

#Set the sample indices – 60% train, 20% validate, and 20% test
spec = c(trainI = .6, testI = .2, validateI = .2)
g = sample(cut(
   seq(nrow(myData)),
   nrow(myData) * cumsum(c(0,spec)),
   labels = names(spec)
   ))
res = split(myData,g)

#build the train, validate, and test sets using the indices found above
trainData <- res$trainI
validData <- res$validateI
testData <- res$testI

#build x and y for the training model
```

```
trainx <- as.matrix(trainData[,1:10])
trainy <- as.factor(trainData[,11])

#################  Model 1  ##############################
# Model1 built on training data. Vanilladot is a simple linear kernel.
model1 <- ksvm(trainx, trainy, type="C-svc" , kernel='vanilladot', C=100,scaled=TRUE)

# calculate a1…am
a <- colSums(model1@xmatrix[[1]] * model1@coef[[1]])

# calculate a0
a0 <- model1@b

# see what the model predicts using validation set
pred1 <- predict(model1,validData[,1:10])

# see what fraction of the model's predictions match the actual classification using validation set
sum((pred1 == (validData[,11])) / nrow(validData))


#################  Model 2  ##############################
# Model2 built on training data. Using Besseldot kernel.
model2 <- ksvm(trainx, trainy, type="C-svc", kernel='besseldot', C=100, scaled=TRUE)

# calculate a1…am
a <- colSums(model2@xmatrix[[1]] * model2@coef[[1]])

# calculate a0
a0 <- model2@b
```

```
# see what the model predicts using validation set
pred2 <- predict(model2,validData[,1:10])

# see what fraction of the model's predictions match the actual classification using validation set
sum((pred2 == (validData[,11])) / nrow(validData))



#################  Model 3   ##############################
# Model3 built on training data. Using rbfdot kernel.
model3 <- ksvm(trainx, trainy, type="C-svc", kernel='rbfdot', C=100, scaled=TRUE)

# calculate a1…am
a <- colSums(model3@xmatrix[[1]] * model3@coef[[1]])

# calculate a0
a0 <- model3@b

# see what the model predicts using validation set
pred3 <- predict(model3, validData[,1:10])

# see what fraction of the model's predictions match the actual classification using validation set
sum((pred3 == (validData[,11])) / nrow(validData))



#################  Test the best model   ##############################
#I ran this section of code separately after running the previous code and looking at the accuracy results
# see what the best model (model1) shows using the test set:
predtest <- predict(model1,testData[,1:10])
```

```
# see what fraction of the model's predictions match the actual classification using the test set
sum((predtest == (testData[,11])) / nrow(testData))



# get the coefficients for my best model (vanilladot)
# calculate a1…am
a <- colSums(model1@xmatrix[[1]] * model1@coef[[1]])
a
# calculate a0
a0 <- model1@b
a0
```