snewt
ISYE6501x HW#1
Due: 24 May 2018

Question 2.1

The Situation: I'm not a teacher anymore (praise be), but when I was in grad school, I had to teach many sections of chemistry lab. Students' pass rate could have been modeled by a good classifier, where the result was pass/fail binary.

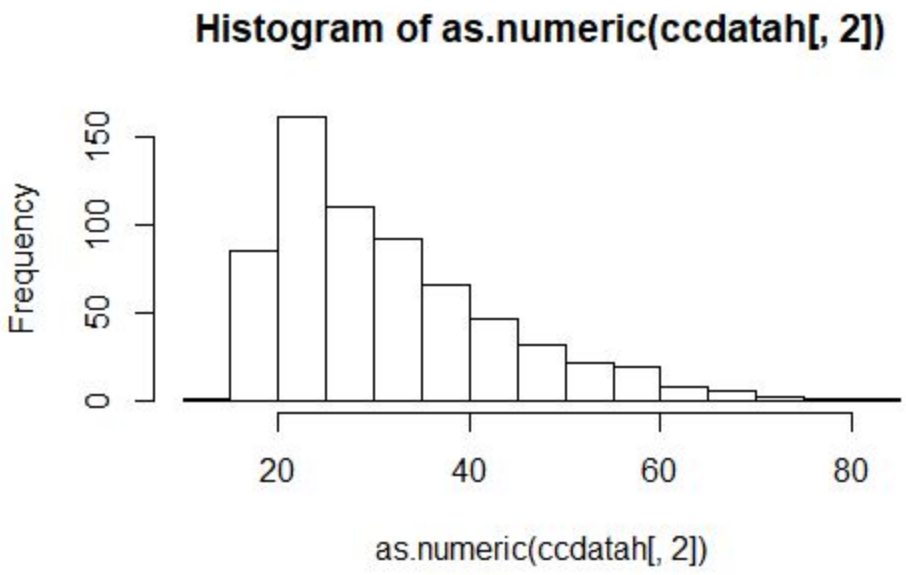
Some appropriate classifiers:

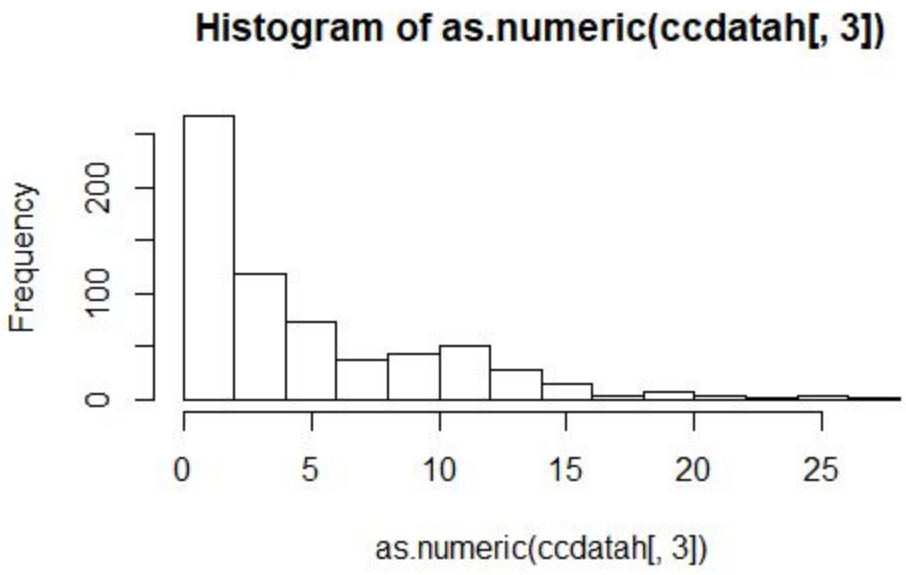
- Did the student attend my office hours when possible? (y/n)
- Did the student make appointments if they were unable to meet during office hours? (y/n)
- Did they complete the quizzes? (%)
- How did they perform on the lab reports? (%)
- I'd like to say sex didn't matter but what if I was punitively harder on male students? (M/F)
- How'd they engage in lectures? (maybe a 1-5 with a good rubric?)
- Did they attend their lectures? (y/n or %)
- How often did they read material before it was lectured on? (%)

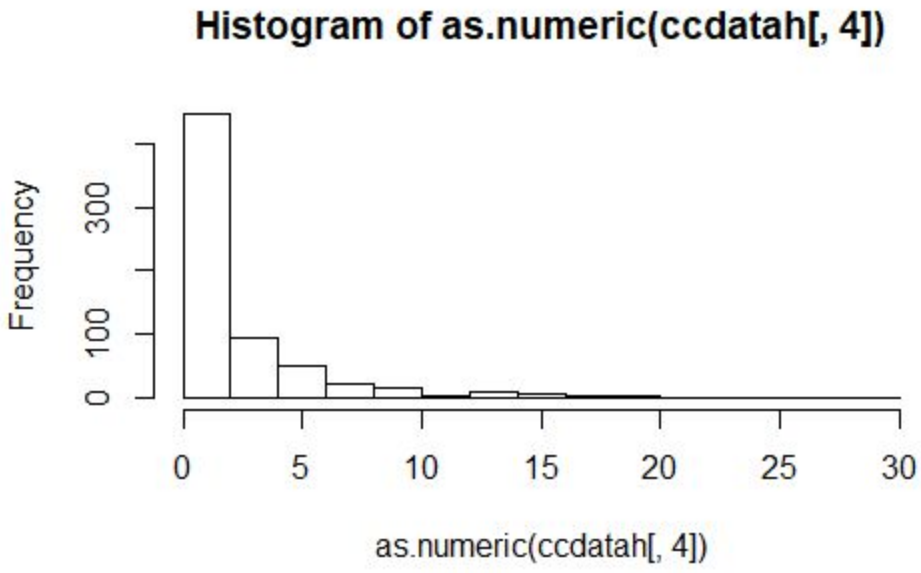
Question 2.2.1

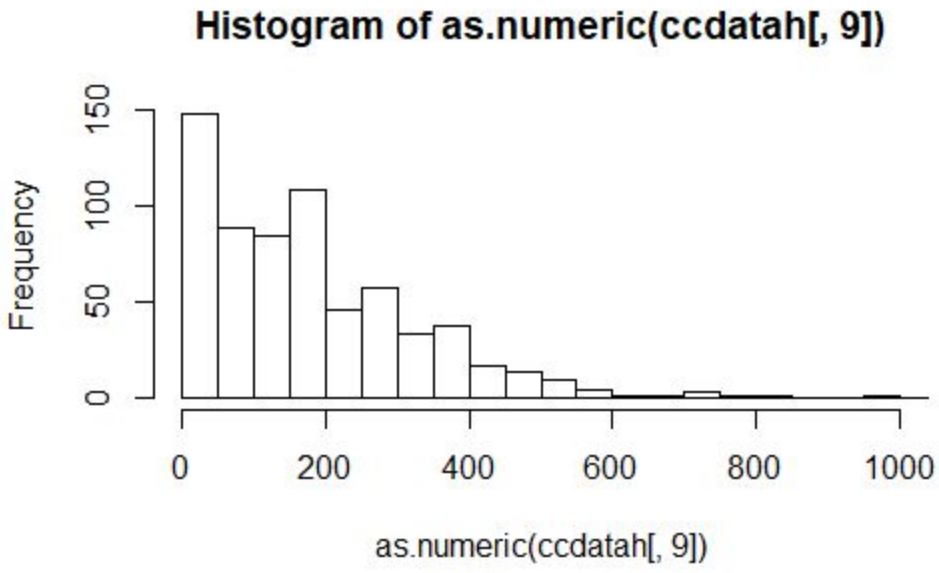
Although the histograms weren't part of the assignment nor were they particularly relevant, I still like to get a feel for my data to have some idea of whether my model makes any sense at all. Peer graders, feel free to skip past these to the next text block.

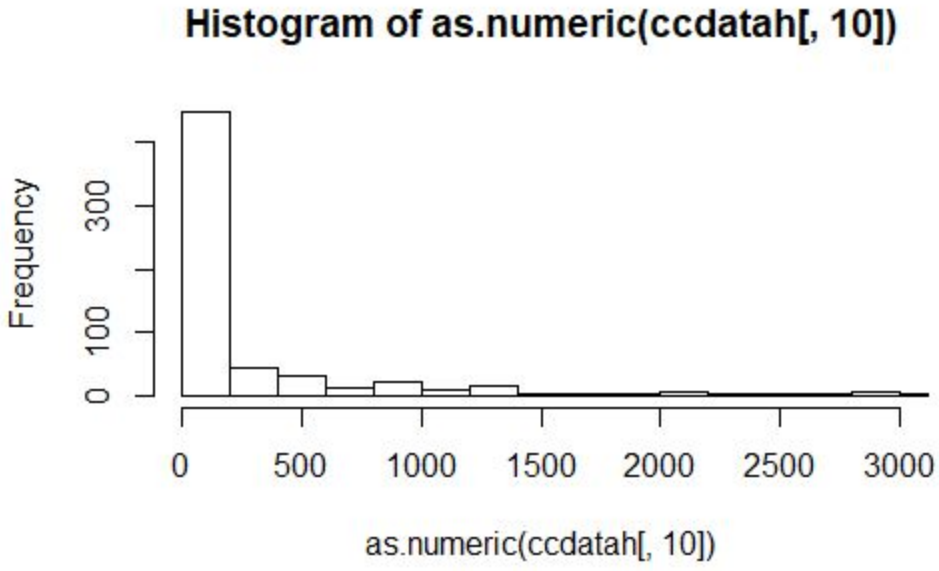
Histograms of relevant items:




Histogram of as.numeric(ccdatah[, 3])
Frequency
0
100
200
0
5
10
15
20
25
as.numeric(ccdatah[, 3])


Histogram of as.numeric(ccdatah[, 4])
Frequency
0
100
300
0
5
10
15
20
25
30
as.numeric(ccdatah[, 4])






```
Model1 <-
ksvm(ccdatah[,1:10],ccdatah[,11],type="C-svc",kernel="vanilladot",C=100,scaled=TRUE)

Model1 <-
ksvm(ccdatah[,1:10],ccdatah[,11],type="C-svc",kernel="vanilladot",C=100,scaled=TRUE)
Error: unexpected input in "Model1 <- ksvm(ccdatah[,1:10],ccdatah[,11],type=""
```

Problem was slanty quotes

```
> model1 <-
ksvm(ccdatah[,1:10],ccdatah[,11],type="C-svc",kernel="vanilladot",C=100,scaled=TRUE)
Error in vanilladot(length = 4, lambda = 0.5) :
  unused arguments (length = 4, lambda = 0.5)
```

Need to convert data to matrix format!

First: Try it with C or λ = 100.

```
> model
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

a<- colSums(model@xmatrix[[1]]*model@coef[[1]])
> a
         A1          A2          A3          A8          A9         A10
-0.0010065348 -0.0011729048 -0.0016261967  0.0030064203  1.0049405641 -0.0028259432
        A11         A12         A14         A15
 0.0002600295 -0.0005349551 -0.0012283758  0.1063633995

> a0 <- -model@b
> a0
[1] 0.08158492

> pred<- predict(model,data[,1:10])

> sum(pred == data_set[,11])/ nrow(data_set)
[1] 0.8639144
```

So this gives us a prediction accuracy of 86.39% - pretty good, and definitely a baseline.

Then: let's try it with a larger C, C=1000:

```
> model1e3
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 1000

Linear (vanilla) kernel function.

Number of Support Vectors : 275

Objective Function Value : -178024.8
Training error : 0.137615

> a <- colSums(model1e3@xmatrix[[1]] * model1e3@coef[[1]])
> a
          A1           A2           A3           A8           A9          A10
-0.0002149185  0.0007097786  0.0011645166  0.0005673024  0.9987192088 -0.0005037912
         A11          A12          A14          A15
 0.0007155434 -0.0009130079  0.0007969700  0.0010062350

> a0 <- -model1e3@b
> a0
[1] 0.07017871

> pred <- predict(model1e3, data_set[,1:10])
> sum(pred == data_set[,11]) / nrow(data_set)
[1] 0.8623853
```

So here we've got a model with a prediction accuracy that's still pretty good, but not as good as C=100.

Finally: let's try it with a tiny C, C=0.001:

```
> model1em3
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 0.001

Linear (vanilla) kernel function.

Number of Support Vectors : 556

Objective Function Value : -0.438
Training error : 0.16208

> a <- colSums(model1em3@xmatrix[[1]] * model1em3@coef[[1]])
> a
         A1          A2          A3          A8          A9         A10
-0.002159778  0.032338170  0.046612449  0.111223162  0.375305335 -0.202026081
        A11         A12         A14         A15
 0.169560847 -0.004923501 -0.025210266  0.081189766

> a0 <- -model1em3@b
> a0
[1] -0.2226155

> pred <- predict(model1em3, data_set[,1:10])
> sum(pred == data_set[,11]) / nrow(data_set)
[1] 0.8379205
```

```
> model10
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

> a <- colSums(model10@xmatrix[[1]] * model10@coef[[1]])
> a
         A1          A2          A3          A8          A9         A10
-0.0010065348 -0.0011729048 -0.0016261967  0.0030064203  1.0049405641 -0.0028259432
        A11         A12         A14         A15
 0.0002600295 -0.0005349551 -0.0012283758  0.1063633995

> a0 <- -model10@b
> a0
[1] 0.08158492


> pred <- predict(model10,data_set[,1:10])
> sum(pred == data_set[,11]) / nrow(data())
numeric(0)
> sum(pred == data_set[,11]) / nrow(data_set)
[1] 0.8639144
```

Since we got the same match for classification/prediction from 10, 100, (86.39144%) we're sticking with one of those two models - they're better than both 1000 (86.23853), 0.001 (83.79205).

So

$$Model10 = -0.0010065348A1 - 0.0011729048A2 - 0.0016261967A3 + 0.0030064203A8 + 1.0049405641A9$$

Using add equation didn't fit, whoops.

```
Model10 = - 0.0010065348A1 - 0.0011729048A2 - 0.0016261967A3 + 0.0030064203A8 +
1.0049405641A9 - 0.0028259432A10 + 0.0002600295A11 - 0.0005349551A12 -
0.0012283758A14 + 0.1063633995A15 - 0.08158492
```

Which predicts the outcome of the credit approval 86.39144% of time in the full data set.

Question 2.2.2

Since it's not required to try other, non-linear kernels, I'm putting that off until/unless I get through the rest of this problem set.

Question 2.2.3

Based on the table of k values, distances, and prediction accuracies gotten from the formula:

```
Modelx <- kknn(R1~., train, test, k=#, distance = #, kernel = 'optimal', scale=TRUE)
Predx <- sum(modelx$fitted.values == test[,11]) / length(test[,11])
```

So it's worth noting that I wasn't actually able to figure this particular splitting by myself. I found somewhere on the internet (and now can't find it again to cite my sources) that someone had published a solution to this problem where they stuck with k=11. I'm using their method, but still performing my own calculations (so I don't feel like this is plagiarism, but if you do, please let me know and I won't do it this way again). I couldn't ever make it actually work in R with the subtracting out only the i'th data point, so I'd really appreciate some peer advice (or TA advice, or professor advice) if someone has time much earlier in the day than 9 pm to explain it to me.

First we needed to get train, test:

```
data <- data_set
set.seed(9)
rowindices <- sample(1:nrow(data), round(.8*nrow(data)), replace=FALSE)
train <- data[rowindices,]; test <- data[-rowindices,]
```

Which told us that both closer (smaller) distances, and fewer k-nearest neighbors would teach us the most about a given point.

```
Distance = 1
kval<-c(3,5,7,9,11,13)
dist<-c(1:3)
preds<-c()
for (i in kval){
  modelkxdx <-kknn(R1~.,train,test,k=kval[i],distance=1,kernel='optimal',scale=TRUE)
  predkxdx <-sum(modelkxdx$fitted.values == test[,11])/length(test[,11])
  preds<-c(preds,predkxdx)
  i<-i+1
}
```

```
> predtable <- cbind(kval,preds)
> predtable
     kval     preds
[1,]    3 0.6106870
[2,]    5 0.5038168
[3,]    7 0.4274809
[4,]    9 0.3664122
[5,]   11 0.2977099
[6,]   13 0.2671756
```

```
Distance = 2 added to that:

total_preds
      kval     preds
 [1,]    3 0.6106870 1
 [2,]    5 0.5038168 1
 [3,]    7 0.4274809 1
 [4,]    9 0.3664122 1
 [5,]   11 0.2977099 1
 [6,]   13 0.2671756 1
 [7,]    3 0.6106870 2
 [8,]    5 0.5038168 2
 [9,]    7 0.4274809 2
[10,]    9 0.3664122 2
[11,]   11 0.2977099 2
[12,]   13 0.2671756 2
```

So we see that a k-value of 3 is going to take the three closest items, but is also going to give the best predictions. It's still not a prediction I'd go with, though: that's 61.1% accuracy? I'll stick with the SVM, thanks. Definitely possible that I did this one wrong, though, because it took about a million tries to get this one going.

Question 3.1.a

Using cross validation, knn model:

```
> train_mod<-train.kknn(R1~., data=data, kmax =25, kernel="optimal")
> View(train_mod)
> summary(train_mod)

Call:
train.kknn(formula = R1 ~ ., data = data, kmax = 25, kernel = "optimal")

Type of response variable: continuous
minimal mean absolute error: 0.1850153
Minimal mean squared error: 0.1086863
Best kernel: optimal
Best k: 25
```

Because the best k here was the largest number, I decided to then try a higher k - is this

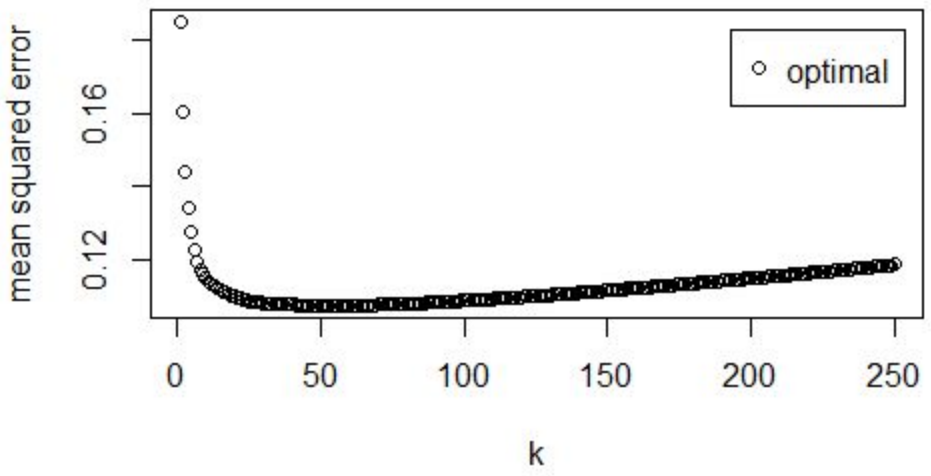
a decreasing trend? Do I need to go higher? Only way to find out is to run it again with a higher kmax.

Tried another order of magnitude, 250.

```
> train_mod<-train.kknn(R1~., data=data, kmax =250, kernel="optimal")
> summary(train_mod)

Call:
train.kknn(formula = R1 ~ ., data = data, kmax = 250, kernel = "optimal")

Type of response variable: continuous
minimal mean absolute error: 0.1850153
Minimal mean squared error: 0.1073792
Best kernel: optimal
Best k: 58
> plot(train_mod)
```



Here both the graph and the actual output of the train.kknn model told us that our best k value (least mean squared error) was k = 58. Translation: if you use the 58 nearest neighbors and take the most common nearest neighbor result from those 58, that's the best model for choosing value of R1 for this dataset. That minimum mean squared error is actually around 11%, so that's better than our best model so far - maybe we should stick with choosing our 58 nearest neighbors.

Question 3.1.b

Data split into training, validation, test data sets

```
> set.seed(9)
> rowindices <- sample(1:nrow(data), round(.6*nrow(data)), replace=FALSE)
> train<-data[rowindices,]
> data2<-data[-rowindices,]
> rowindices2 <- sample(1:nrow(data2), round(.5*nrow(data2)), replace=FALSE)
> test<-data2[rowindices2,]
> validate<-data2[-rowindices2,]
```

```
> model3_1 <-
ksvm(as.matrix(train[,1:10]),as.factor(train[,11]),type="C-svc",kernel="vanilladot",C=100,s
caled=TRUE)
 Setting default kernel parameters
> a<-colSums(model3_1@xmatrix[[1]]*model3_1@coef[[1]])
> a
         A1          A2          A3          A8          A9          A10
 2.224198e-05 -1.207661e-05 -2.644561e-05  8.354297e-05  9.931139e-01
-2.773420e-06
        A11          A12          A14          A15
 6.829961e-05  2.937568e-05 -1.061858e-04  3.132040e-04
> a0<- -model3_1@b
> a0
[1] 0.1275971
> pred<-predict(model3_1,train[,1:10])
> trainpred<-sum(pred == train[,11])/nrow(train)
> trainpred
[1] 0.8520408
```

So, actually, validation isn't something that makes sense, here, because we're not picking between multiple models. We can test twice, though, on test data and validation data, so we will do that:

Validate set:

```
> predv<-predict(model3_1,validate[,1:10])
```

```
> valpred<-sum(predv == validate[,11])/nrow(validate)
> valpred
[1] 0.9007634
```

Wow, so the validation set performed much better than the training set, about 91% of the time.

Test set:

```
> predte<-predict(model3_1,test[,1:10])
> testpred<-sum(predte == test[,11])/nrow(test)
> testpred
[1] 0.8549618
```

The test set, however, performed about as well as the training set, about 85% of the time.

What does that really tell us? It tells us that the true performance of the model is probably somewhere between 85% and 91% accuracy. This was unexpected, as I thought I understood from the lessons that validation & testing will show that data is performing less well than the training set, according to the model, except for a small amount of rounding error.