Homework Week 2

2022-08-29

Question 3.1

Using the same data set (credit_card_data.txt or credit_card_data-headers.txt) as in Question 2.2, use the ksvm or kknn function to find a good classifier:

1. *using cross-validation (do this for the k-nearest-neighbors model; SVM is optional); and*
2. *splitting the data into training, validation, and test data sets (pick either KNN or SVM; the other is optional).*

KNN Cross Validation

First, lets grab the kknn package and bring in our data.

```
library(kknn)
```

```
loans <- read.csv("credit_card_data-headers.txt", sep = '\t')
```

Lets go ahead and check our structure to see what we are working with.

```
str(loans)
```

```
## 'data.frame':    654 obs. of  11 variables:
##  $ A1 : int  1 0 0 1 1 1 1 0 1 1 ...
##  $ A2 : num  30.8 58.7 24.5 27.8 20.2 ...
##  $ A3 : num  0 4.46 0.5 1.54 5.62 ...
##  $ A8 : num  1.25 3.04 1.5 3.75 1.71 ...
##  $ A9 : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ A10: int  0 0 1 0 1 1 1 1 1 1 ...
##  $ A11: int  1 6 0 5 0 0 0 0 0 0 ...
##  $ A12: int  1 1 1 0 1 0 0 1 1 0 ...
##  $ A14: int  202 43 280 100 120 360 164 80 180 52 ...
##  $ A15: int  0 560 824 3 0 0 31285 1349 314 1442 ...
##  $ R1 : int  1 1 1 1 1 1 1 1 1 1 ...
```

Now we need to separate our data. Since we are using cross validation, we will want 3 sets of data; training, validation, and test. HOWEVER, the cv.kknn package takes in only one argument, training data. Why is that? Well it handles the splitting out of training/validation data for us given kcv number of folds.

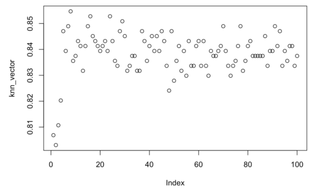
What about testing data? Well, the cv.kknn model does not actually create a model, rather, it returns the fitted (yhat) values of a given value of k. Remember, the whole point of cross validation is not to build a model for predicting, it is to determine the best hyper-parameters so that we can *then* go build a model with those hyper parameters. Thus, we will:

1. Split the data in two sets, training and test
    i. Let cv.kknn handle the training/validating
2. *after* finding the best hyper parameters, we will then use test data to report on the accuracy.

```
sample_size <- floor(.8 * nrow(loans))
set.seed(123)
train_ind <- sample(seq_len(nrow(loans)), size = sample_size)

k_train <-loans[train_ind,]
k_test <- loans[-train_ind,]
```

Now, lets loop over values of K to find the one that performs best.

```
k_list <- as.list(c(1:100))
k_accuracy <- c()
pred <- rep(0, nrow(k_train)) # create a vector of numbers for as many rows in the data set
knn_vector <- vector("numeric") #create an empty vector to store the accuracy of the model
set.seed(123)
for (K in seq_along(k_list)){ #loop over k to find the accuracy

  k_model <-  cv.kknn(R1~.,
                      k_train,
                      kcv = 10,
                      k = K,
                      scale = TRUE)



  pred <- as.data.frame(round(k_model[[1]],2))
  actual <-as.data.frame(k_model[[1]],2))
  k_accuracy <- sum(pred == actual) / nrow(pred) #calc how many predictions were correct

  knn_vector <- c(knn_vector, k_accuracy) #store the models accuracy for each value of k
}
plot(knn_vector)
```



Now find the most accurate value of K

```
c(which.max(knn_vector)),max(knn_vector))
```

```
## [1] 8.0000000 0.8546845
```

As mentioned above, cv.kknn doesn't actually return a model we can reuse to predict new points (it returns a list y (actual feature value) and yhat (predicted feature value) so that we can calculate the accuracy of each different values of k. Therefore, we have to now create a model object using the best value of K as determined above using the WHOLE training set.

```
best_k_model <-   kknn(k_train$R1~.,
                  k_train,
                  k_test,
                  k = 8,
                  scale = TRUE)
```

Now, we use the test set to measure the accuracy of our model!

```
best_pred <- round(fitted(best_k_model))
sum(best_pred == k_test$R1)/nrow(k_test)
```

```
## [1] 0.8244275
```

KNN Cross Validation - Leave one out

Lets also use train.kknn to do the leave one out method of cross validation. This is essentially the same thing, just using a value of kcv (number of folds) equal to n (number of rows in the training data.

This function works a little different than the above, instead of giving us the yhat values and having us calculate the accuracy ourself, loop through various values of K to determine the best, train.kknn actually does the lifting for us - we tell it the maximum values of K we want it to loop through for us (or specific values of K we want it to loop(over) and it just return the best value of k it found (the most accurate). Then we will need to build a new model using that parameter if we want to test the model.

```
k_accuracy <- c()
#pred <- rep(0, nrow(k_train)) # create a vector of numbers for as many rows in the data set

#set.seed(3)

k_loo_model <-  train.kknn(R1~.,
                    k_train,
                    kmax = 100,
                    kernel = "optimal",
                    scale = TRUE) #see "further analysis"



#actual <- as.data.frame(k_train$R1)
#pred <- round(predict(k_loo_model, k_train))
#k_accuracy <- sum(pred == actual) / nrow(k_train)

k_loo_model
```

```
##
## Call:
## train.kknn(formula = R1 ~ ., data = k_train, kmax = 100, kernel = "optimal",     scale = TRUE)
##
## Type of response variable: continuous
## Minimal mean absolute error: 0.1873805
## Minimal mean squared error: 0.1035524
## Best kernel: optimal
## Best k: 48
```

```
#k_accuracy
```

The best value appears to be 48, so lets build a model using that value and run our test data through it.

```
best_k_model <-   kknn(R1~.,
                  k_train,
                  k_test,
                  k = k_loo_model$best.parameters[[2]], #returns the best k from above
                  scale = TRUE)
best_pred <- round(fitted(best_k_model))
sum(best_pred == k_test$R1)/nrow(k_test)
```

```
## [1] 0.8320611
```

As it appears, our model is 83% accurate!

Question 3.1b

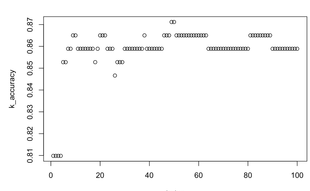
Now, instead of doing cross validation, lets do "normal validation".

As such, we will randomly split our data into three groups. See documentation therein for process:

```
set.seed(123)

train_size <- floor(.5 * nrow(loans)) #first, calculate how many rows we want. Here I am saying 50% of the rows f
or training
train_ind <- sample(seq_len(nrow(loans)), size = train_size) # now, get indecies for 50% of the data at random

test_val <- loans[-train_ind,] #remove the training data so we can split the remaining data into validation/test
data

val_size <- floor(.5 * nrow(test_val)) # we want 50% of the REMAINING data to be for validation, so calculate wha
t that number is

val_ind <- sample(seq_len(nrow(test_val)), size = val_size) # now get the indecies

#now store those indecies into variables
k_train <-loans[train_ind,]
k_val <- test_val[val_ind,]
k_test <- test_val[-val_ind,]
```

Now, lets loop over 100 values of K and plot the accuracy.

```
k_list <- as.list(c(1:100))
k_accuracy <- c()
pred <- rep(0, nrow(k_train))

for (K in k_list) {
    k_model <-   kknn(k_train$R1~.,
                      k_train,
                      k_val,
                      k = K,
                      scale = TRUE)
    pred <- round(fitted(k_model))
    k_accuracy[K] <- sum(pred == k_val$R1)/nrow(k_val)
}


plot(k_accuracy)
```




```
c(which.max(k_accuracy),max(k_accuracy))
```

```
## [1] 49.0000000  0.8711656
```

Now we see the best value of k is 49 with an accuracy of 87%. Now lets make a model with that info and see how accurate it is on our test data:

```
best_k_model <-   kknn(k_train$R1~.,
                  k_train,
                  k_test,
                  k = which.max(k_accuracy),
                  scale = TRUE)
best_pred <- round(fitted(best_k_model))
sum(best_pred == k_test$R1)/nrow(k_test)
```

```
## [1] 0.8414634
```

It appears the model is 84% accurate.

Further Thoughts

It would appear we had a higher accuracy with normal validation vs cross validation, but that is not an accurate statement. That only happened by chance, as if we remove the seed values from the above code that may shift. The change an accuracy is really only a result in being trained on different data and used different values of k. However, I will note, they were extremely similar, both were within 1% accuracy of each other and K only differed by 1. Had our data set been much larger, there may have been even less of a difference.

Homework Week 2, Question 4

2022-09-05

Question 4.1

Describe a situation or problem from your job, everyday life, current events, etc., for which a clustering model would be appropriate. List some (up to 5) predictors that you might use.

> My wife is a Realtor, and often times she must send out marketing campaigns to her lead database. This database includes information such as:
>
> 1. Year they last purchased
> 2. Annual Income
> 3. Size of current house
> 4. Number of family members
> 5. Location (Lattitude/ Longitude)
> 6. Age
>
> This information could all be used as predictors to segment her leads into different groups so she could tailor her messages to various groups. For example, her message to somone who just bought a new home this year, has a newborn baby, and lives in a 2b1b probably does not want to see the newest million dollar listing. They would be better off only getting a christmas card from my wife so her name sticks in their head for when they are ready to buy their new home.
>
> Conversely, she may find a cluster of leads who have been in their house for around 5 years, have $400k+ of annual income, and lives in a 5k sqft home. This person may be VERY interested in what their home is worth now and what homes in their price range are now on the market if they are looking for an upgrade.

Question 4.2

The iris data set iris.txt contains 150 data points, each with four predictor variables and one categorical response. The predictors are the width and length of the sepal and petal of flowers and the response is the type of flower. The data is available from the R library datasets and can be accessed with iris once the library is loaded. It is also available at the UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Iris). The response values are only given to see how well a specific method performed and should not be used to build the model.

Use the R function kmeans to cluster the points as well as possible. Report the best combination of predictors, your suggested value of k, and how well your best clustering predicts flower type.

> Per the below analysis, it appears the best value of K is k = 3, and the best predictors of species is *not* all of the measures, rather just petal length and width, which has a 96% accuracy.

Analysis

First, lets take a look at the data we have to work with.

```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

Looks like we have 4 values that can be used to determine the Species. I'd like to look into what combination provides the best result. Should we use all four? Just the Sepal info? Or just the Petal info?

As a note, kmeans is an unsupervised learning method - we are going to assume we are building this model out to determine how many potential classifications there could be, rather than determining what classification the flower *is* given a set of data points. As such, we will only use the species information in determining which model is most accurate rather than using that information in our training data.

To scale, or not to scale?

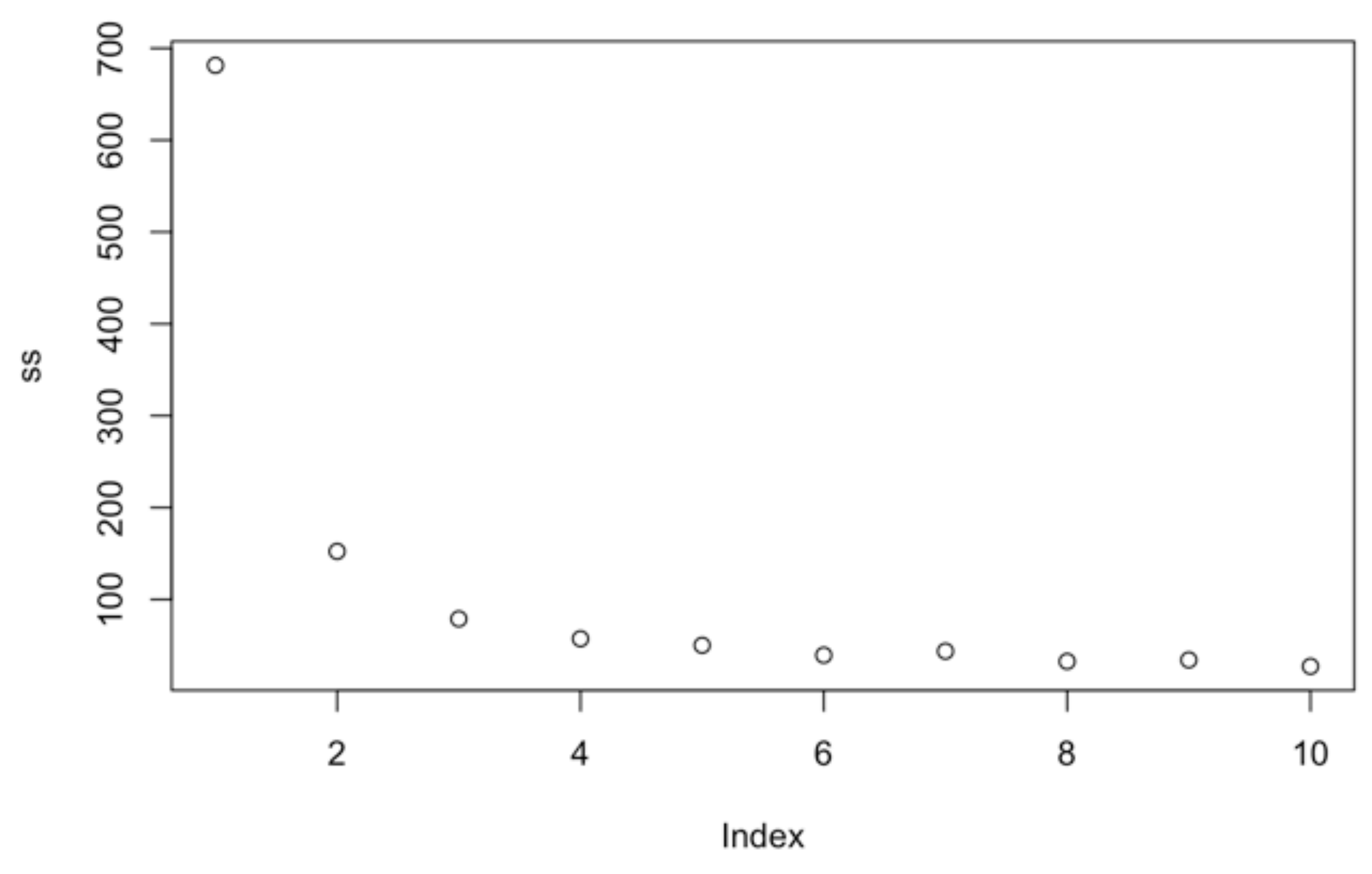
Typically, it is a good idea to scale data before using algorithms that are driven by distance, as the distance between predictors is not always 1:1. For example, if our data consisted of one measure in miles, and one measure in temperature, the delta of mile means something entirely different than a delta of 1 degree, so we scale the measures to make them equivalent.

However, with our iris data set, all measures are in the same measure of length, so scaling would not be appropriate, as we would lose the nuance of the distance measurements.

Determine how many classes to use

First, we want to see what the best value of k is to use. i.e. how many species should we separate iris flowers into? We will do this by using the "elbow" method, i.e. seeing what value of k gives the least sum of squares (distance between the points and the centroid) for each value of k, before our results begin to diminish.

```
k_list <- c(1:10)
ss <- c()
for (K in seq_along(k_list)){
  k_means.model.1 <- kmeans(iris[,1:4],K)
  ss[K] <- k_means.model.1$tot.withinss
}
plot(ss)
```



As we can see above, k = 3 seems to give us the most bang for our buck. Lets compare that to how many species there *actually* is in the data set.

```
length(unique(iris$Species))
```

```
## [1] 3
```

As it appears, there is only 3 different species of iris flowers! As such, we will use k= 3 for our models below.

Using Sepal and Petal Data

First, I'd like to see how accurate a model would be if we used all of the available data to predict what species a flower is given a new set of data. First we need to build the model.

```
set.seed(123)
k_means.model.1 <- kmeans(iris[,1:4],3)
k_means.model.1
```

```
## K-means clustering with 3 clusters of sizes 50, 62, 38
##
## Cluster means:
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1     5.006000    3.428000     1.462000    0.246000
## 2     5.901613    2.748387     4.393548    1.433871
## 3     6.850000    3.073684     5.742105    2.071053
##
## Clustering vector:
##   [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
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [75] 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 3 3 3 3 2 3 3 3 3
## [112] 3 3 2 2 3 3 3 3 2 3 2 3 2 3 3 2 2 3 3 3 3 3 2 3 3 3 3 2 3 3 3 2 3 3 3 2 3
## [149] 3 2
##
## Within cluster sum of squares by cluster:
## [1] 15.15100 39.82097 23.87947
##  (between_SS / total_SS =  88.4 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

Now, let's compare our predictions against actual species classifications

```
table(k_means.model.1$cluster, iris$Species)
```

```
##
##     setosa versicolor virginica
##   1     50          0         0
##   2      0         48        14
##   3      0          2        36
```

It appears we have 125 data points (50 + 48 + 36) classified correctly, and 38 (36 + 2) classified incorrectly, giving us a 89% accuracy.

Sepal Length/Width

Now we will do the same for just sepal length/width.

```
k_means.model.2 <- kmeans(iris[,1:2],3)
table(k_means.model.2$cluster,iris$Species)
```

```
##
##     setosa versicolor virginica
##   1     50          0         0
##   2      0         38        15
##   3      0         12        35
```

Using Sepal length/width, we get 123 correct classifications, giving us an accuracy 82%

Petal Length/Width

```
k_means.model.3 <- kmeans(iris[,3:4],3)
table(k_means.model.3$cluster,iris$Species)
```

```
##
##     setosa versicolor virginica
##   1     50          0         0
##   2      0         48         6
##   3      0          2        44
```

Using just the petal length and width, we get 142 correct classifications and 8 incorrect, for an accuracy of 94.5%. Therefore, this is the best indicator of classification.