Homework 1

2022-08-23

Question 2.1

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.

My company has a complex engineering system for invoice creation. Currently, there are many breakdowns in the process of creating an invoice that ultimately leads to an incorrect balance on an invoice sent to the customer. Under current state, the customer gets the incorrect invoice, has to call into customer service, report the incorrect balance, and then a ticket gets created so that it will be resolved. While the ideal path would be to cut these issues off at their source (fix the cause of the failure), a stop gap solution is needed in the meantime to prevent losing customers.

Incorrect balance issue is one of the top reasons that customers churn is because of frustration with billing problems. If we could build a classifier, based on the invoice details, and whether or not a customer complained about an incorrect balance, we could proactively investigate invoices that are at a high risk for “failure” before a customer even receives the invoice.

Some of these invoice features would be:

1. Product type
2. Payment Schedule
3. Auto/Manual invoice?
4. Customer ID (have they had a history of invoice failures?)
5. Dollar Value

Question 2.2

1. Using the support vector machine function ksvm contained in the R package kernlab, find a good classifier for this data. Show the equation of your classifier, and how well it classifies the data points in the full data set.

   $y_i = -0.004117738_{A1}-0.086896089_{A2}+0.12971526_{A3}-0.083744032_{A8}+0.988381368_{A9}+0.031253888_{A10}-0.055666972_{A11}-0.037281856_{A12}+0.021940744_{A14}+0.018521785_{A15} -0.08054451$

   The model classifies the data points with an accuracy of 86.4% accuracy using a linear kernel.

2. You are welcome, but not required, to try other (nonlinear) kernels as well; we’re not covering them in this course, but they can sometimes be useful and might provide better predictions than vanilladot.

   See “Further Analysis” section below.

3. Using the k-nearest-neighbors classification function kknn contained in the R kknn package, suggest a good value of k, and show how well it classifies that data points in the full data set. Don’t forget to scale the data (scale=TRUE in kknn).

   The most accurate value of K I’ve determined is 12, and it classifies the data in the whole set with an accuracy of 85.3%. See below for code and commentary on assumptions used.

Code & Commentary

Data

Read in the data

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

Check the structure of the data.

```
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 ...
```

I notice here that several of the fields appear to be boolean/factors of 1 or 0, but the data type is still an integer. This may need to be converted in our models, but I have left it as unconverted so we can factor it on an as needed basis for each model. Next I check the summary to confirm which fields are factors and which may have other values.

```
summary(loans)
```

```
##        A1               A2              A3               A8
##  Min.   :0.0000   Min.   :13.75   Min.   : 0.000   Min.   : 0.000
##  1st Qu.:0.0000   1st Qu.:22.58   1st Qu.: 1.040   1st Qu.: 0.165
##  Median :1.0000   Median :28.46   Median : 2.855   Median : 1.000
##  Mean   :0.6896   Mean   :31.58   Mean   : 4.831   Mean   : 2.242
##  3rd Qu.:1.0000   3rd Qu.:38.25   3rd Qu.: 7.438   3rd Qu.: 2.615
##  Max.   :1.0000   Max.   :80.25   Max.   :28.000   Max.   :28.500
##        A9              A10              A11              A12
##  Min.   :0.0000   Min.   :0.0000   Min.   : 0.000   Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.: 0.000   1st Qu.:0.0000
##  Median :1.0000   Median :1.0000   Median : 0.000   Median :1.0000
##  Mean   :0.5352   Mean   :0.5612   Mean   : 2.498   Mean   :0.5382
##  3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.: 3.000   3rd Qu.:1.0000
##  Max.   :1.0000   Max.   :1.0000   Max.   :67.000   Max.   :1.0000
##       A14              A15               R1
##  Min.   :   0.00   Min.   :     0   Min.   :0.0000
##  1st Qu.:  70.75   1st Qu.:     0   1st Qu.:0.0000
##  Median : 160.00   Median :     5   Median :0.0000
##  Mean   : 180.08   Mean   :  1013   Mean   :0.4526
##  3rd Qu.: 271.00   3rd Qu.:   399   3rd Qu.:1.0000
##  Max.   :2000.00   Max.   :100000   Max.   :1.0000
```

KSVM

Linear Model

First we make a list that has several cost amounts we can loop over to see which returns the best results. I am starting with a very wide range of cost amounts.

```
costs <- c(10^(-5:5))
costs_list <- as.list(costs) #turn the costs into a list so we can loop over it
```

Initiate an empty list to store to accuracy result into

```
accuracy <- c()
```

Start looping over our cost functions and calling ksvm! We are going to start with the linear kernel, as that means we are running our classifier without transforming the data points to see if they are linearly separable.

Note: we are setting scaling = TRUE, since in our summary of the data we can see that some of the values vary as little from 0 to 1, and some are 0 to 100,000. Since we don't know what these values indicate, we cannot know for sure if the distance etween the variables is actually as large as the values make it seem, so we scale them between 1 and 0 to remove this variability.

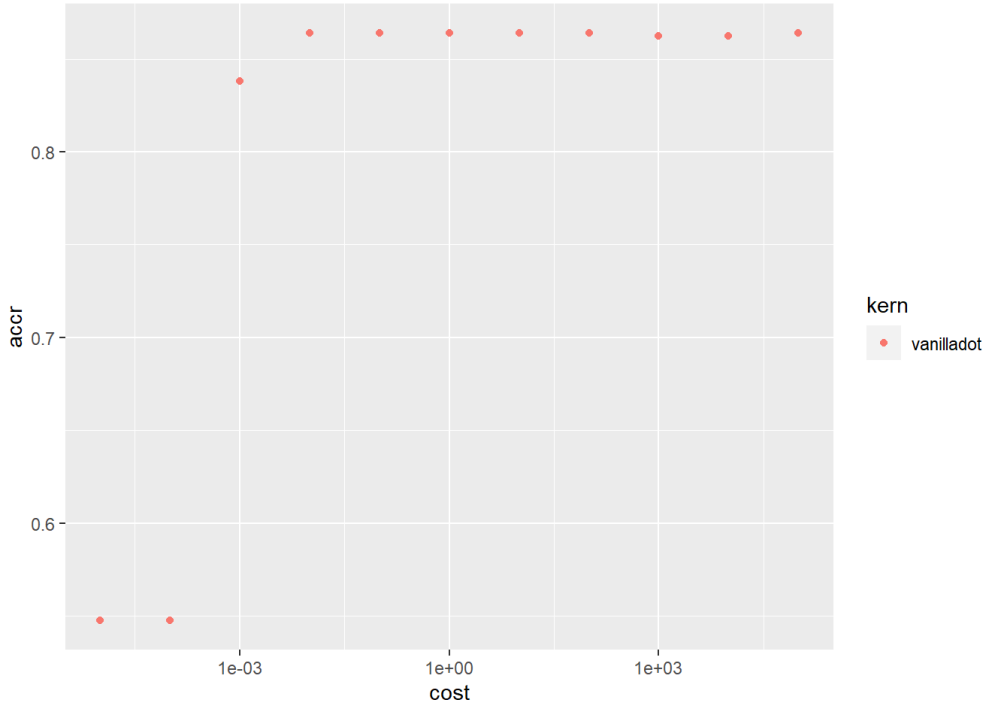
```
for(i in seq_along(costs_list)){ #we use sequence alon instead of sequence, so that if we want to just loop over one cost we won't have any errors.
  model <- ksvm(loans[,11]~., #predict 11th collumn with all other cols
                data = loans[,1:10], #use this data to predict
                type="C-svc", #we use this as it is cost classification
                kernel="vanilladot", #regular linear kernel
                C=costs_list[[i]], #the cost that we will vary
                scaled=TRUE) #see above.

  pred <- predict(model, loans[,1:10]) # prediict what the points would be using our model

 accuracy[i] <- sum(pred==loans[,11])/nrow(loans) #now calculate how accurate the pred was

}
```

Lets look at our costs and their accuracy's. I've a factor column for the type of kernel, so that we can evaluate other kernel types and plot them later down below.

```
svm_accuracy <- do.call(rbind, Map(data.frame, cost= costs_list, accr = accuracy, kern = factor("vanilladot")))
svm_accuracy
```

```
##     cost      accr       kern
## 1  1e-05 0.5474006 vanilladot
## 2  1e-04 0.5474006 vanilladot
## 3  1e-03 0.8379205 vanilladot
## 4  1e-02 0.8639144 vanilladot
## 5  1e-01 0.8639144 vanilladot
## 6  1e+00 0.8639144 vanilladot
## 7  1e+01 0.8639144 vanilladot
## 8  1e+02 0.8639144 vanilladot
## 9  1e+03 0.8623853 vanilladot
## 10 1e+04 0.8623853 vanilladot
## 11 1e+05 0.8639144 vanilladot
```

```
sp <- ggplot(svm_accuracy, aes(x= cost, y= accr, col=kern))+geom_point()
sp + scale_x_continuous(trans='log10')
```



Formula

First, get the coefficients for a_1-a_m. **Note:** since this is using the last value of "model", and we ran a loop over that variable, the value in the model is the last value of the loop. As usch, this is the formula when the value of C is 10^5. Since the accuracy of this cost was highest, we will use it as our answer.

```
a <- colSums(model@xmatrix[[1]] * model@coef[[1]])
a
```

```
##           A1           A2           A3           A8           A9          A10
## -0.004117738 -0.086896089  0.129715260 -0.083744032  0.988381368  0.031253888
##          A11          A12          A14          A15
## -0.055666972 -0.037281856  0.021940744  0.018521785
```

Now, lets get the a0

```
a0 <- model@b
a0
```

```
## [1] -0.08054451
```

Now put it all together, and our formula is

$$y_i = -0.004117738_{A1}-0.086896089_{A2}+0.12971526_{A3}-0.083744032_{A8}+0.988381368_{A9}+0.031253888_{A10}-0.055666972_{A11}-0.037281856_{A12}+0.021940744_{A14}+0.018521785_{A15} -0.08054451$$

K Nearest Neighbor

Since we are now measuring different values of k (instead of cost like above), we need to create a list of k to iterate over.

```
k_list <- as.list(c(1:50))
```

Create a list to store the accuracy

```
k_accuracy <- c()
```

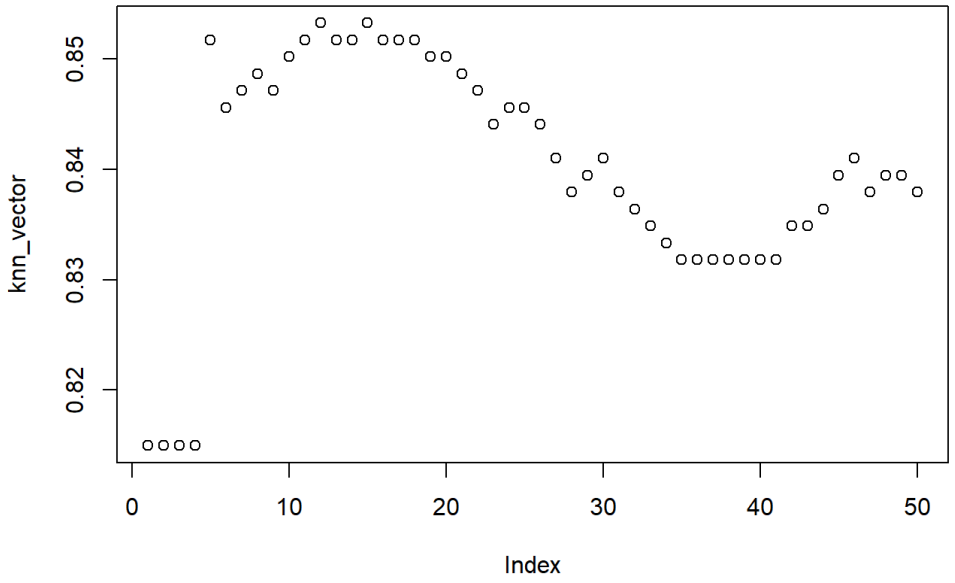
Now, start looping over a model for k, trying out different values of k and calculating their accuracy. But first, lets think about why we are iterating over each point and calculating whether it is correct using a given value of k.

When we look at our SVM, we trained one model, and then used that one model to evaluate all points, vs here, where we are effectively making one modell for every point, for every value of k we are testing. Why don't we make a model for every point in the SVM test?

1. KNN is much less computationally expensive. We can tell just by the speed they run, but also if we calculated their "Big O" speed we could see it there too. Therefore, it is easy to run a separate model for each point, leaving out the point we are testing, and can get an accuracy pretty quick.
2. Even if we had all the time in the world to run a separate SVM for each point to exclude it from the model, there is not much of a reason to do so. Think about the name, "Support Vector Machine". The model is really only dependent on the support vectors that touch the margin line. Any point outside of that doesn't impact the model, so there isn't a huge loss in not excluding a point from the model, unless that point happened to be the support vector. Even then, it wouldn't impact the SVM model by much as there's a good amount of data in our set, i.e. the next support vector would be close enough by and our model would not change significant;y.

```
pred_i <- rep(0, nrow(loans)) # 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
for (K in seq_along(k_list)){ #loop over k to find the accuracy
    for (i in 1:nrow(loans)){ #see above - we are finding the nearest neighbor of every value and we don't want the test val
ue to be included in the neighbors we look at
      k_model <- kknn(loans[-i, 11]~.,
                      loans[-i,1:10],
                      loans[i,1:10],
                      k = K,
                      scale = TRUE) #see "further analysis"
      pred_i[i] <- round(fitted(k_model)) #store the prediction into the vector generated earlier. See further discussion on
rounding in "Further Analysis"
      k_accuracy <- sum(pred_i == loans[,11]) / nrow(loans) #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)
```



And now we can see which value of K (12) is the most accurate (85.3%)

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

```
## 12 0.853211
```

Further Analysis

KSVM

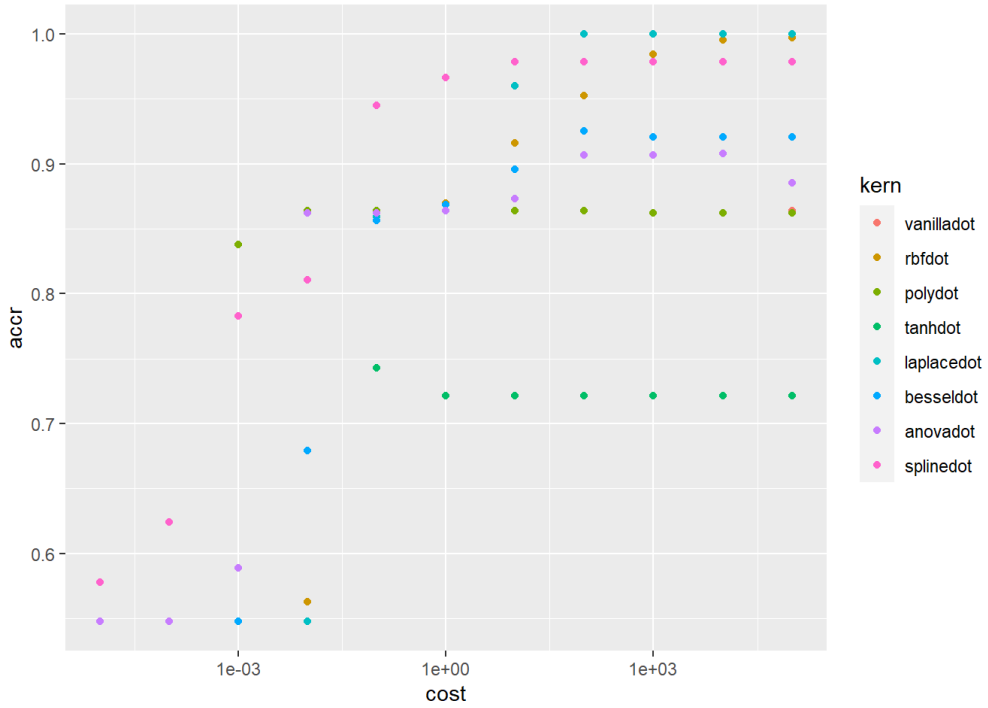
Lets take a look at some other kernel types and see if any of them can better classify the data. We are in no hurry, so let's try them all!

```
kern_types <- c("rbfdot","polydot","tanhdot","laplacedot","besseldot","anovadot","splinedot")
levels(svm_accuracy$kern) <- c(levels(svm_accuracy$kern),kern_types)
```

This loop is the same as the first ksvm loop we did above, so I have not annotated this one. However, the only difference is we are now going to append a row to svm_accuracy for every cost for every kernel, so we can plot them out.

```
for (kern_type in kern_types){
  for(i in seq_along(costs_list)){
    model <- ksvm(loans[,11]~.,
                  data = loans[,1:10],
                  type="C-svc",
                  kernel=kern_type,
                  C=costs_list[[i]],
                  scaled=TRUE)
    pred <- predict(model, loans[,1:10])

    other_kerns <- sum(pred==loans[,11])/nrow(loans)
    new_frame <- data.frame(cost = costs_list[[i]],
                            accr = other_kerns,
                            kern = kern_type)
    svm_accuracy <- rbind(svm_accuracy, new_frame)
  }
}
```

```
ggplot(svm_accuracy, aes(x= cost, y= accr, col=kern))+geom_point() + scale_x_continuous(trans='log10')
```



Conclusion

laplacedot kernel had the highest accuracy, with 100% accuracy. However, because we used all the available data as training data, we have to be careful assuming this is the best model. Using only train data can lead to our model being overfit - which is likely given the “100%” accuracy.

KNN

There are two things I’d like to discuss regarding KNN.

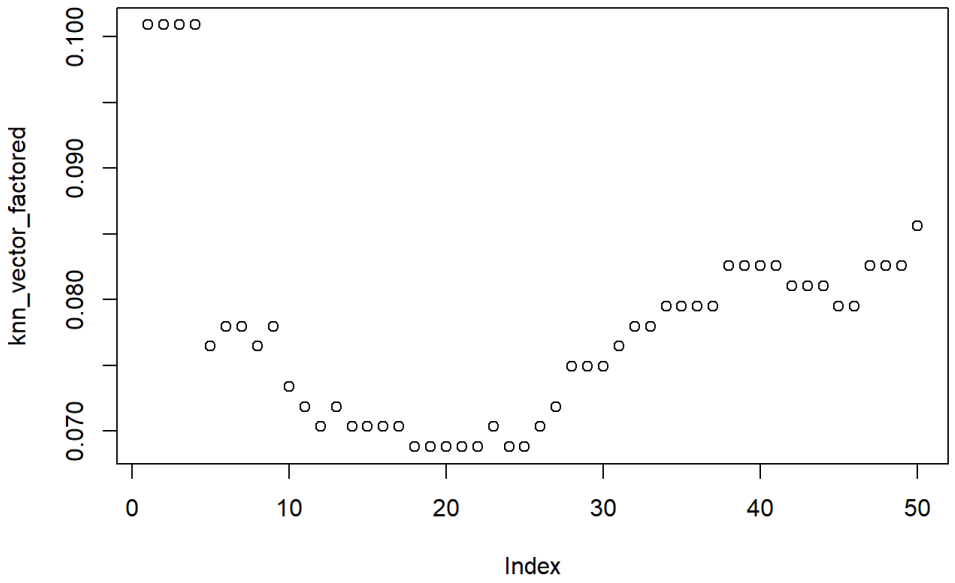
1. Should we factor our result to be a binary answer or 0 or 1, so that when KNN runs it takes a majority vote of the nearest neighbors, or should we leave it as continuous and round the final the average of the nearest neighbors to 1 or 0 to determine the final classification?
2. Should we scale our inputs?

1. To Factor, or not to Factor?

To determine if we should facotr or not before modeling, it would be wise to see how it would affect the model.

```
k_accuracy <- c()
pred_i <- rep(0, nrow(loans))
knn_vector_factored <- vector("numeric")
for (K in seq_along(k_list)){
    for (i in 1:nrow(loans)){
      k_model <- kknn(as.factor(loans[-i, 11])~., #factor the feature
                      loans[-i,1:10],
                      loans[i,1:10],
                      k = K,
                      scale = TRUE)
      pred_i[i] <- (fitted(k_model)) #removing the rounding
      k_accuracy <- sum(pred_i == loans[,11]) / nrow(loans)
    }
  knn_vector_factored <- c(knn_vector_factored, k_accuracy)
}
```

```
plot(knn_vector_factored)
```



Conclusion

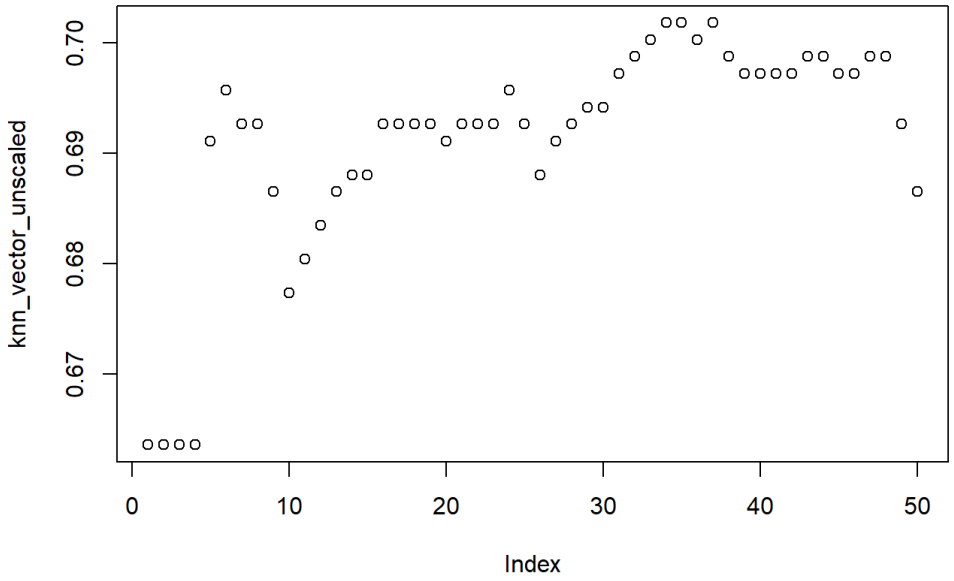
We can see in the graph that when we factor the feature we are trying to predict, we have very high accuracy (100%) when K is very low. However, this high of accuracy calls the question, is our model overfitted? If every point can be determined by looking at it's one nearest neighbor, I don't feel that it would handle outliers very well. As such, I determined it is best to not factor the predicted feature. Instead, it is best to take an average of the nearest neighbors (by rounding our guess to 1 or 0)

2. To Scale, or not to Scale?

Now, lets run the model again and determine how scaling impacts our data.

```
k_accuracy <- c()
pred_i <- rep(0, nrow(loans))
knn_vector_unscaled <- vector("numeric")
for (K in seq_along(k_list)){
    for (i in 1:nrow(loans)){
      k_model <- kknn((loans[-i, 11])~.,
                      loans[-i,1:10],
                      loans[i,1:10],
                      k = K,
                      scale = FALSE)
      pred_i[i] <- round(fitted(k_model))
      k_accuracy <- sum(pred_i == loans[,11]) / nrow(loans)
    }
  knn_vector_unscaled <- c(knn_vector_unscaled, k_accuracy)
}
```

```
plot(knn_vector_unscaled)
```




```
cat(which.max(knn_vector_unscaled), max(knn_vector_unscaled))
```

```
## 34 0.7018349
```

Conclusion

We can see that leaving our data un-scaled causes a drop in accuracy. Let's look back at the data summary. The distance between some of our parameters (like A15) are going to have a far greater distance between them than the parameters that have a min/max of 0/1. KNN calculates the distance to the closest data points using euclidean distance, so leaving these large parameters unscaled unfairly weights them. Because we don't have context of what the parameters are, we cannot make an informed decision on whether these factors should be weighted or not. Because the accuracy is higher when scaled, I have determined it is best to scale the inputs.

```
summary(loans)
```

```
##        A1               A2              A3               A8
##  Min.   :0.0000   Min.   :13.75   Min.   : 0.000   Min.   : 0.000
##  1st Qu.:0.0000   1st Qu.:22.58   1st Qu.: 1.040   1st Qu.: 0.165
##  Median :1.0000   Median :28.46   Median : 2.855   Median : 1.000
##  Mean   :0.6896   Mean   :31.58   Mean   : 4.831   Mean   : 2.242
##  3rd Qu.:1.0000   3rd Qu.:38.25   3rd Qu.: 7.438   3rd Qu.: 2.615
##  Max.   :1.0000   Max.   :80.25   Max.   :28.000   Max.   :28.500
##        A9              A10              A11              A12
##  Min.   :0.0000   Min.   :0.0000   Min.   : 0.000   Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.: 0.000   1st Qu.:0.0000
##  Median :1.0000   Median :1.0000   Median : 0.000   Median :1.0000
##  Mean   :0.5352   Mean   :0.5612   Mean   : 2.498   Mean   :0.5382
##  3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.: 3.000   3rd Qu.:1.0000
##  Max.   :1.0000   Max.   :1.0000   Max.   :67.000   Max.   :1.0000
##       A14               A15               R1
##  Min.   :   0.00   Min.   :     0   Min.   :0.0000
##  1st Qu.:  70.75   1st Qu.:     0   1st Qu.:0.0000
##  Median : 160.00   Median :     5   Median :0.0000
##  Mean   : 180.08   Mean   :  1013   Mean   :0.4526
##  3rd Qu.: 271.00   3rd Qu.:   399   3rd Qu.:1.0000
##  Max.   :2000.00   Max.   :100000   Max.   :1.0000
```