**Pre-Processing the Data**

We have no missing values , our data set is just full of Booleans and we do not have the need to normalize our data. So no data scaling needed

As each column is a pixel that is our feature now.

We should not perform dimensionality reduction as the rows are already flattened and if we do we lose a lot of information.

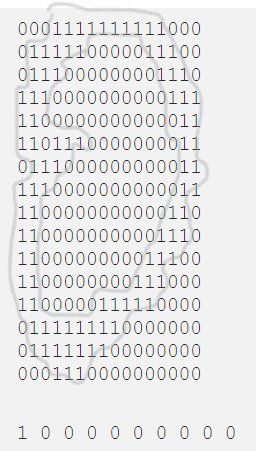
So, we only transformed our data a little bit.

Below are the steps we performed on the semeion data set before feeding the algorithm to the data:-

The semeion data set has 256 columns in each row. Each column represents a pixel which is converted to a Boolean.

These 256 columns are my features now and the last column says which digit these 256 pixels represent.

Using these pixels in a row we have to train a model to classify the digits respectively.



IN the above figure, if you notice we split one of the rows into 16 character lines and get an actual drawn zero(tried highlighting it using a grey color in MS-paint for a better view)

So this is how the all the digits ( the entire dataset )looks like.

This was how the data looked before removing the decimal values(.0000 ) from the digits 1 and 0

We even changed the last column that represents the digits in the form of one and zeroes to actual digits for the ease of model building

So, 1 0 0 0 0 0 0 0 0 0 is “0”

0 1 0 0 0 0 0 0 0 0 is “1”

0 0 1 0 0 0 0 0 0 0 is “2”

And so on until digit “9”

**Model Building**

**Talk about how you were asked to frame the test cases**

Finally, the interesting stage of the whole process.

We have tried the following classification techniques to classify :-

1. K -nearest neighbours

This method when implemented gave an accuracy greater than 90% most of the time.

Then performed K-fold cross validation and saw that the model is giving 80% accuracy.

Guess we just avoided over training by performing this train control method.

train.control <- trainControl(method = "cv", number = 10)

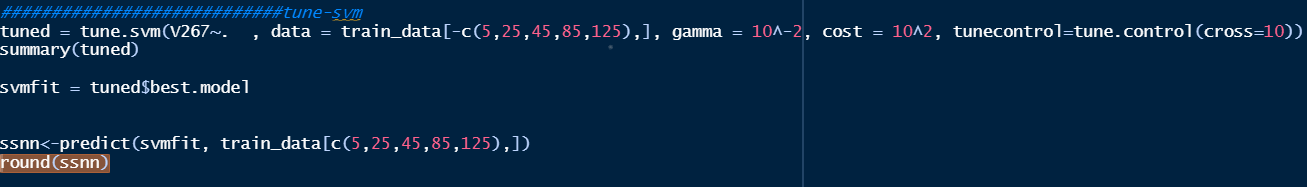
But again ,for most of the test cases without cross validation the accuracy was always greater than 95%

2. Support Vector machines

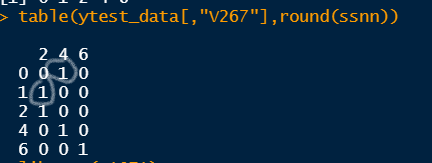
Even after running this method for “n” number of test cases it gave an

accuracy of no greater than 60%

So then tried to tuning function to perform the k-fold cross validation

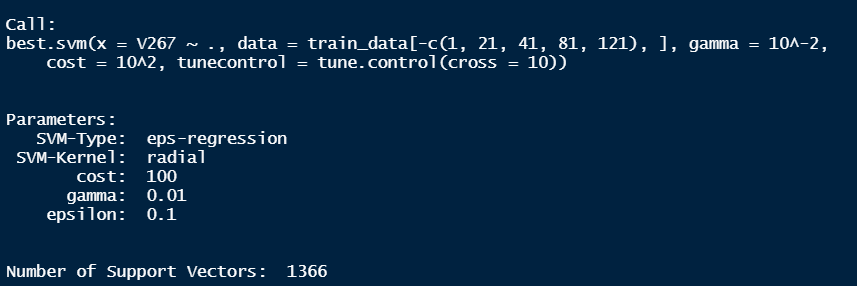


Even after using this new tuning function and though played around with cost function



As you can see even after performing the tuning out of the 5 digits 2 were wrongly predicted hence the accuracy is still 60%

I give up on the model now!



The important question is why this model isn’t performing well?

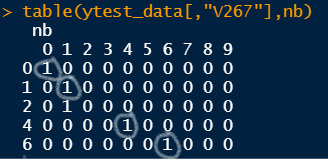
3.Naive Bayes

This model gives an accuracy of 80% most of the time

Library used : library(e1071)

Satisfied with this model’s predictions but So far kNN has given the best

Predictions.



The circled are the right predictions

Here 2 got predicted wrong in this test case

Similarly for other test cases 0 also got predicted as 8 but 6 seems to be predicted right most of the test cases.

4. **Neural Nets**

**Use this to show figures for nnet**

**https://datascienceplus.com/neuralnet-train-and-test-neural-networks-using-r/**

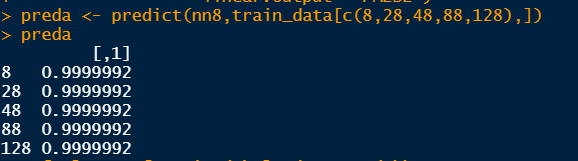
Coming to the model we wanted to implement in the first place

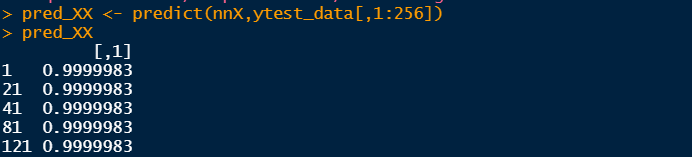
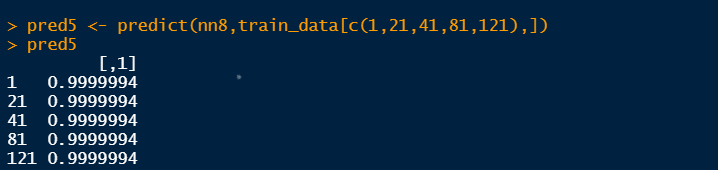
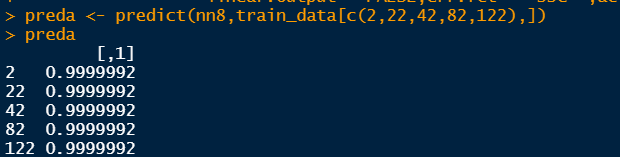
Well because neural nets were created to process visual data and learn to recognize objects

What better model to use for recognizing digits!

Libraries : library(neuralnet)

library(nnet)

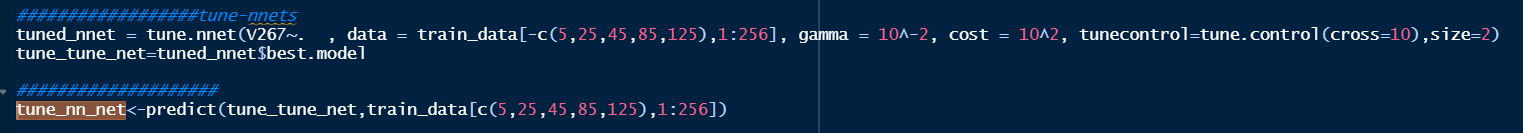


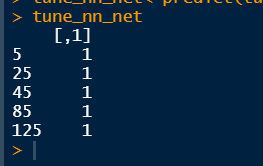
So this neural net model is giving the probability of the prediction being the actual digit

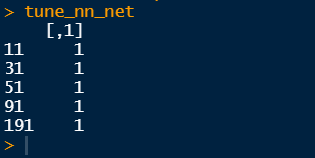
You can see these probabilities are really high no less than 99%

Just for our satisfaction we do tuning on the neural net (k=fold cross validation)



Not so surprisingly, after tuning the nnet the probabilities become 1 for all of the test cases





These are the results for all of the test cases we could see

So We can say that neural nets beat kNN model that was giving the best results so far!