Austin Beller

IST 707

December 9, 2019

Neural Nets and Digit Recognition

Intro

For this project we decided to work with Kaggle project digit recognition. In this database we attempted to classify and understand handwritten numbers using different algorithms and pixels to predict each digit. Some of this was covered in the classroom while other information was not. In this case we took the analysis one step further by seeing if we could create a neural network in R which could accurately predict each digit. The applications of such a project could help in developing better ways to identify other things such as a pencil or another object and could even be later developed into facial recognition software.

Question’s

1. What can we do to clean the data to make it better?
2. Can we develop a neural net in R for this dataset?
3. How else can this be applied to other industries?
4. What other learning algorithms work?

Background

MNIST ("Modified National Institute of Standards and Technology") is the standard introduction dataset of computer vision. The dataset was first released in 1999 and has served as a benchmark for people developing algorithms in the area of computer vision. This has been used to test new machine learning algorithms and methods that have been developed.

In our quest we found that some have decided to apply neural nets which is a type of machine learning that mimics the actions of the brain. The human brain was inspiration for this kind of network.

We chose in this case the test data from the Kaggle competition however for class the train dataset was reduced in order to save time. This dataset from class was used in order to be more efficient with the time given to do the assignment.

Cleaning the Data

In this case some of the data we had to clean up due to the fact that there were some missing variables and some columns with no or very little information. We checked for NA’s and also zeros. In order to properly prepare the data for analysis we removed the label to properly graph the data. Below is some code that was used to examine the data.

***sum(duplicated(testset)) # no duplicate rows***

***0***

***sum(duplicated(trainset)) # no duplicate rows***

***0***

***sum(sapply(testset, function(x) sum(is.na(x)))) # There are no missing values***

***0***

***sum(sapply(testset, function(x) sum(is.na(x))))# There are no missing values***

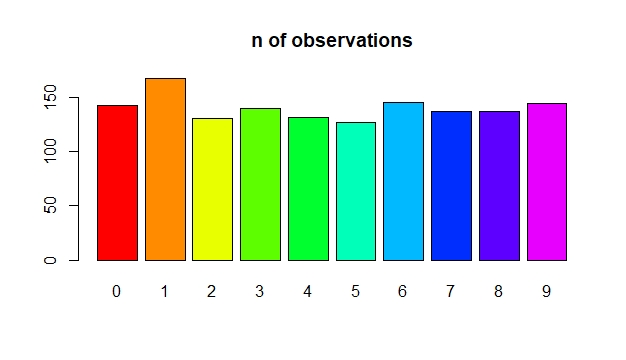
***0***

We also used a function in order to normalize the data found below.

***normalize <- function(x) {***

***return ((x - min(x)) / (max(x) - min(x))) }***

Below is a simple chart of the distribution of data.



Methods and Analysis

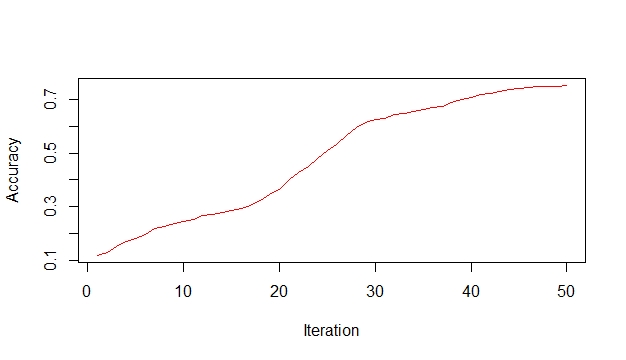
The method of our task involved some research into a neural network which was simple enough to work with the data and not out of our depth of explanation. The two methods we choose to compare with the neural network were k nearest neighbor and SVM. We chose a simple neural network with fully Connected layers, each with 128, 64, and 10 neurons, respectively. These layers are connected by two Activation Layers, which use the ReLU activation function. The ReLU activation function is the most common function used for neural networks. This was used using the mxnet library. We also wanted to see the timing of the neural net to see how efficient it would run so we created a timer.

Below is a figure of how our neural network will be set up.



Results

The results of our analysis showed that as the number of iterations increase the accuracy and time of the analysis also increased as shown in the chart below.



For the SVM and the KNN models we found that they had a accuracy of 100% and 87 % respectively. Below are the statistics for the KNN model and SVM model respectively.

***Overall Statistics***

***Accuracy : 0.8905***

***95% CI : (0.8875, 0.8935)***

***No Information Rate : 0.1115***

***P-Value [Acc > NIR] : < 2.2e-16***

***Kappa : 0.8783***

***Support Vector Machine object of class "ksvm"***

***SV type: C-svc (classification)***

***parameter : cost C = 100***

***Gaussian Radial Basis kernel function.***

***Hyperparameter : sigma = 1.86626346678359e-07***

***Number of Support Vectors : 920***

***Objective Function Value : -13.137 -30.5007 -31.7689 -17.4662 -52.6796 -44.2028 -25.0239 -29.0142 -23.6677 -34.8201 -28.6311 -17.2924 -39.9917 -29.075 -36.1773 -46.3627 -23.3621 -58.1525 -36.9726 -45.3426 -54.228 -44.4019 -55.8467 -38.1543 -29.3363 -95.2439 -37.4837 -37.3146 -80.3794 -43.3735 -42.3965 -36.5114 -44.8981 -36.2224 -115.6711 -58.882 -39.5524 -77.6365 -52.0915 -21.7284 -44.5387 -27.0276 -38.0182 -90.2029 -42.5436***

***Training error : 0***

Discussion

In this case SVM is seen as the most accurate method for this dataset we will see if we can create a neural net which matches this accuracy. Here we demonstrated that it is possible to create a neural net and that the neural net can perform as well as the SVM however it cannot perform as quickly. This is likely due to the fact that the SVM is still a lazy learner and does not hold the information. If enough iterations are performed, we could get a perfect score just as the SVM. This shows that although neural nets are extremely useful, they are not always the best solution to every problem.

Neural nets have been shown to have a great amount of potential in many different fields such as medicine. In this case our neural net could later be developed into a facial recognition program or could be expanded to read not just numbers but letters and recognize words. This is a good start for anyone learning to program. Some things did not go well and we did have some trouble in fully understanding all of the background and theories behind these machine learning algorithms but in the end it was worth taking the time to apricate the effort of others.

References:

https://api.rpubs.com/dardodel/digit\_recognition\_KNN\_NN\_SVM

https://dataaspirant.com/2017/01/09/knn-implementation-r-using-caret-package/

https://afit-r.github.io/svm

https://www.kaggle.com/srlmayor/easy-neural-network-in-r-for-0-994/code

https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7

Appendix

install.packages("e1071")

install.packages("rpart.plot")

install.packages("https://github.com/jeremiedb/mxnet\_winbin/raw/master/mxnet.zip", repos = NULL)

library(mxnet)

library(readr)

library(e1071)

library(caret)

library(rpart)

library(rpart.plot)

library(class)

require(kernlab)

#This first model we use KNN

#we do not need to preprocess as most of it was already done for us

#the data has already been split into a test and train version

#Here we develop the dicision tree

trainset <- read.csv("/Users/Me/Desktop/Data Science Books/Data science Documents for class/IST-707/Week 8/Kaggle-digit-train-sample-small-1400.csv")

testset <- read.csv("/Users/Me/Desktop/Data Science Books/Data science Documents for class/IST-707/Week 8/Kaggle-digit-train.csv")

trainset$label <- as.factor(trainset$label)

#There are collumns with zeros which are not needed either

#here I borrowed code from the website below to remove the zeros

train\_orig <- trainset

test\_orig <- testset

nzv.data <- nearZeroVar(trainset, saveMetrics = TRUE)

drop.cols <- rownames(nzv.data)[nzv.data$nzv == TRUE]

train <- trainset[,!names(trainset) %in% drop.cols]

test <- testset[,!names(testset) %in% drop.cols]

registerDoMC(cores = 3)

tc <- trainControl(method = "cv", number = 4, verboseIter = F, allowParallel = T)

modSVMR1 <- train(label ~. , data= training, method = "svmRadial", trControl = tc)

SVMRadial\_predict1 <- as.numeric(predict(modSVMR1,newdata = validating))-1

# Duplicated rows

sum(duplicated(testset)) # no duplicate rows

sum(duplicated(trainset)) # no duplicate rows

# Checking for NAs

sum(sapply(testset, function(x) sum(is.na(x)))) # There are no missing values

sum(sapply(testset, function(x) sum(is.na(x))))# There are no missing values

#Normalization

normalize <- function(x) {

return ((x - min(x)) / (max(x) - min(x))) }

train\_norm <- as.data.frame(lapply(train[2:275], normalize))

#Now we can try to use KNN

trainknn <- trainControl(method = "cv", number = 10, repeats = 3)

set.seed(3333)

knn\_fit <- train(label ~., data = train, method = "knn",

trControl=trainknn, preProcess = c("center", "scale"), tuneLength = 10)

#here we see the most accurate models

knn\_fit

#so how accurate is our model

test\_pred <- predict(knn\_fit, newdata = test)

test\_pred

#We use a confusion matrix to see and make sure that label is a factor

test$label <- as.factor(test$label)

confusionMatrix(test\_pred, test$label )

#it predicts with 89% accuracy!

#in this case we did not need to normalize the data but the formula

#is useful to other applications

#This part of the HWMK was made with the help of this website

#https://api.rpubs.com/dardodel/digit\_recognition\_KNN\_NN\_SVM

#and https://dataaspirant.com/2017/01/09/knn-implementation-r-using-caret-package/

#Now in this section we will begin with SVM

#This was used with the help of the website below

#https://afit-r.github.io/svm

model <- ksvm(label ~ ., data = train, type = "C-svc", kernel = "rbfdot",

C = 100, gamma = 0.001, scaled = FALSE)

model

#In this case our model had no training error

#So 100% accuracy!

#So to extend my project here I am looking to see if I can create a nural

#network which was first described by the author of this kaggle

#https://www.kaggle.com/srlmayor/easy-neural-network-in-r-for-0-994/code

#so let us explore the data with a bar graph

#here we drop the label

dim(trainset[,-1])

#we see how many observations

dim(testset)

#here is our barplot

barplot(table(trainset[,1]), col=rainbow(11, 1), main="n of observations")

#So we can see the amount of observations

#Now we have to prepare our nural net by creating some x and y planes

trainset.x <- trainset[,-1] #remove 'label' column

trainset.y <- trainset[,1] #label column

trainset.x <- t(trainset.x/255)

testset.x <- t(testset/255)

#so after we download the mxnet data we can now layer

neural.data <- mx.symbol.Variable("data") # each layer is passed to the next

neural.fc1 <- mx.symbol.FullyConnected(neural.data, name="firstlayer", num\_hidden=128)

neural.act1 <- mx.symbol.Activation(neural.fc1, name="firstactivation", act\_type="relu")

neural.fc2 <- mx.symbol.FullyConnected(neural.act1, name="secondlayer", num\_hidden=64)

neural.act2 <- mx.symbol.Activation(neural.fc2, name="secondactivation", act\_type="relu")

neural.fc3 <- mx.symbol.FullyConnected(neural.act2, name="thirdlayer", num\_hidden=10)

neural.softmax <- mx.symbol.SoftmaxOutput(neural.fc3, name="neuralsoft")

graph.viz(neural.softmax)

#S

times <- mx.metric.logger$new() #to keep track of the results each iterration

starts <- proc.time() #mark the start time

mx.set.seed(0)

neural1 <- mx.model.FeedForward.create(neural.softmax, #the network configuration made above

X = trainset.x, #the predictors

y = trainset.y, #the labels

#ctx = mx.cpu(),

num.round = 50,

array.batch.size = 100,

array.layout="colmajor",

learning.rate = 0.001,

momentum = 0.95,

eval.metric = mx.metric.accuracy,

initializer = mx.init.uniform(0.07),

epoch.end.callback = mx.callback.log.train.metric(1,times)

)

print(paste("The Training was:", round((proc.time() - starts)[3],2),"sec"))

#this was how long it took after one

#lets see it in a graph

plot(times$train, type="l", col="red", xlab="Iteration", ylab="Accuracy")

#We can see the amount of itterations increases the accuracy of selction

#In this case we made a sucessful neural network we can play around

#with the parameters to make it more acurate.