Predicting numbers from MNIST database with Neural network and PCA

DS:630 – Project 2

Jinlian HowPanHieMeric & Pranay Katta

12/19/2017

Contents

Executive Summary 2
Methodology 3
Data exploration and preparation 3
Neural network without PCA 3
Setting up the Cross Validation 3
Identifying the parameters for the Neural Network 3
Finalizing the model 3
Neural network with PCA 4
Applying PCA to the dataset 4
Setting up the Cross Validation 4
Identifying the parameters for the Neural Network 4
Finalizing the model 5
Results and Conclusion 5
Appendix 1: Code 6

Submitted by: Jinlian HowPanHieMeric & Pranay Katta

Executive Summary

This report outlines the methodology and result of predicting handwritten numbers through a Neural Network model in R. The goal was to identify a neural network configuration in R with the neuralnet package on the MNIST dataset to predict handwritten numbers from 0 to 9 with all features and reduced dimensionality (less features) and compare accuracy and time to reach a desired output. The Neural Networks were configured differently for both models using various k-folds, hidden layer values and altering the stepmax function to reach results. The original dataset consisted 59,999 observations, with 785 features.

Two different models were run, one on the entire dataset and one on a dataset with reduced dimensionality. In order to successfully create the latter Principal Component Analysis (PCA) was used and a new dataset with 154 features was created to represent 95% of the cumulative variance of the original data. Both datasets consisted of one column representing the number handwritten and detailed by the remaining features. The original dataset had 784 representing the pixels of a 28x28 matrix and with a value from 0 -255 to represent the gradient filling for said pixel. As for the reduced dataset, the same concept existed in terms of what the features represented, but across 154 features only.

The Neural Network models were configured separately with different values of hidden layer count and size and different stepmax to achieve a desired output in a timely manner. In addition to this, in order to reduce overfitting issues and achieve higher level of accuracy in predication the datasets were divided into folds, 6 with PCA and 8 without PCA. However, unlike the traditional method of cross fold validation, due to the nature of the largesse of the dataset the model was run against one fold at a time, rather than all folds but one, to be able to generate an output as less observations were being modeled. Each model looked at the accuracy for each fold by comparing predictions to original value and then the average of all accuracies for a dataset.

Unfortunately, despite efforts to reduce computing time and generate results from both Neural Networks and calculate accuracy of each folds individually and models, both models were not able to complete in time. Models have been running for more than 48 hours in certain situations, and still no results were generated. However, from our tests we were able to get above 60% accuracy for the model with PCA and 97% accuracy for the model with PCA.

Submitted by: Jinlian HowPanHieMeric & Pranay Katta

Methodology

Data exploration and preparation

The dataset provided had 785 variables with 59,999 observations. Prior to modeling the variable types were verified so as to make sure all variables were coded correctly to carry out the necessary actions to proceed with the Neural Network. From this step, it was identified that the first column, representing the numbers written for that row, was not factorized and would not successfully convert to a data matrix to complete the set-up of the neural network. As a result, the column values were factorized to 10 levels ranging from 0 to 9. This was then binded to the original dataset and the first column dropped resulting in 794 variables. The dataset was now ready for the model.

Neural network without PCA

Setting up the Cross Validation

As an unconventional method of cross validation was carried out by training the data against one fold at a time, rather than all but one designated one the value of K was determined by encompassing enough observations for a better model, but not too high so the model would take too long to converge and generate an output. For the dataset with no cross validation this was set at K = 7 so that each fold contained 8570 to 8573 observations, thus the model was trained 7 times against each fold individually.

Identifying the parameters for the Neural Network

Two parameters of the Neural Network were considered when tuning and performing a trial and error to find the “best” parameters for best prediction. These two parameters were hidden layers and step max. The former indicates the number of hidden layers between the input and output and desired nodes per layer, while the latter is the maximum steps for the training of the neural network. In order to complete trial and errors successfully, this was run only against the first 1000 observations of the original dataset so an output could be generated quicker. However, it was understood that a neural network best performs when there is more data available. Nonetheless, multiple parameters were inputted and with various hidden layers of size to facilitate convergence in a timely fashion. It was observed that too many layers would increase time for the model to complete, and thus the final model was chosen with 3 layers of 700, 350 and 70. As for step max, the higher the value the better, as greater accuracy can be achieved, however, due to the large amount of observations and features this was fixed at 8000, as values greater seemed to slow the model greatly and below accuracy was undesirable. This generated accuracy of above 60% in the trial model, although not the best, it was the highest, as all other test generated accuracy below 30%.

Finalizing the model

The model was to run on each fold and calculate the accuracy for each fold. In order to best achieve this in an automated manner a for loop was created as shown below. This loop would capture the model for each fold, as well as its accuracy. In addition, the system time function was applied to generate the amount of time it took to complete the entire model. Finally, the

average of the accuracies was calculated to show the percentage accuracy of the model for defined parameters.

```
set.seed(2487)
system.time(
  for (i in 1:k){
      testingFold <- folds[[i]]
      cvTest <- train[testingFold,]
      cvTestLabel <- model_outputs[testingFold,]
      nn_model[i] <- neuralnet(f, data = cvTest, hidden = c(700, 350, 70), stepmax = 8000, linear.output = FALSE)
      nn_predicted <- compute(nn_model, cvTest[, nn_model$model.list$variables])
      nn_predicted <- nn_predicted$net.result
      original_values <- max.col(cvTestLabel)
      nn_predicted_2 <- max.col(nn_predicted)
      accuracy<- mean(nn_predicted_2 == original_values)
    }
)
mean(accuracy)
```

Neural network with PCA

Applying PCA to the dataset

PCA allows to reduce dimensionality of a dataset, while trying to maintain as much of the original dataset's variance as desired. PCA was carried out on the original dataset features only using the prcomp package in R. Following this it rendered 784 principal components (PC) and showed that 95% of the variance from the original data could be captured using only 154 features. Thus a new dataset was created using the new values, also generated by prcomp, for the first 154 PC's and binded to the data matrix which represented the numbers being represented by each row. Thus the final dataset had 164 variables with 59,999 observations

Setting up the Cross Validation

As an unconventional method of cross validation was carried out by training the data against one fold at a time, rather than all but one designated one the value of K was determined by encompassing enough observations for a better model, but not too high so the model would take too long to converge and generate an output. For the dataset with no cross validation this was set at K = 6 so that each fold contained between 9,999 and 10,001 observations, thus the model was trained 8 times against each fold individually.

Identifying the parameters for the Neural Network

Similarly to the first model, two parameters of the Neural Network were considered when tuning and performing a trial and error to find the "best" parameters for best prediction. These two parameters were hidden layers and step max. The former indicates the number of hidden layers between the input and output and desired nodes per layer, while the latter is the maximum steps for the training of the neural network. In order to complete trial and errors successfully, this was run only against the first fold with 10,000 observations. Multiple parameters were inputted and with various hidden layers of size to facilitate convergence in a timely fashion. The final chosen parameters were that the hidden layers would be 150 and 70, with a step max of 5000. This generated accuracy of above 90% in the trial model, which is why

it was chosen. It was noted though that identifying "best" parameters were reached faster for this dataset.

Finalizing the model

Identically to the first model was to run on each fold and calculate the accuracy for each fold. In order to best achieve this in an automated manner a for loop was created as shown below. This

```
system.time(
    for (i in 1:k){
      testingFold <- folds[[i]]
      cvTestLabel_pca <- model_outputs[testingFold,]
      cvTest_pca <- new_modeldata[testingFold,]
      nn_model_pca <- neuralnet(f_pca, data = cvTest_pca, hidden = c(150, 70), stepmax = 5000, linear.output = FALSE)
      nn_predicted_pca <- compute(nn_model_pca, cvTest_pca[, nn_model_pca$model.list$variables])
      nn_predicted_pca <- nn_predicted_pca$net.result
      original_values_pca <- max.col(cvTestLabel_pca)
      nn_predicted_2_pca <- max.col(nn_predicted_pca)
      accuracy_pca <- mean(nn_predicted_2_pca == original_values_pca)
    }
)
```

loop would capture the model for each fold, as well as its accuracy. In addition, the system time function was applied to generate the amount of time it took to complete the entire model. Finally, the average of the accuracies were calculated to show the percentage accuracy of the model for defined parameters.

Results and Conclusion

Unfortunately, we were not able to generate individual fold and model results due to time. While we were able to define a model that worked and reached convergence, during our testing phase in determining parameters, when running the full model on the folds computing time outweighed expectations and final conclusive results were not reached. Nonetheless, we hold the results from our testing phase where we achieved 67% accuracy with the original dataset and 97% accuracy for the model using the PCA dataset.

We were also able to note how much faster the Neural Network with reduced dimensionality ran a lot faster. When testing on 10,000 observations it was a matter of minutes in getting results, while with the original dataset, even with only 1000 observations it took on average 45 mins to run. Thus, when faced with a large dataset of numerous observations, but also primarily multiple features PCA is a recommendation to achieve an output faster and reduce computing time and still being able to generate higher accuracy if not more as the model can converge and compute faster. If all features are to be preserved other neural network methods, such as convolutional ones in deep learning are commended to achieve results quicker and more accurately.
r model.

Appendix 1: Code

```
library(neuralnet)
library(caret)
options(max.print=1000000)

# load data
mnist<- read.csv('/Users/JinLian/Google Drive/Saint Peters/Fall 17/630 - Machine Learning/HMWK/HMWK 5/mnist.csv')
str(mnist)
mnist[,'X5']<-factor(mnist[,'X5'])
str(mnist)

#create data matrix
model_outputs <- as.data.frame(model.matrix(~X5+0, mnist))
View(model_outputs)

#Bind data for model data
model_data<- mnist[,-1]
train <- as.data.frame(cbind(model_data, model_outputs))

#Create the folds
k=6
num <- mnist[,1]
folds <- createFolds(num, k = 6)

#Create the formula
#Update the formula
n<- names(train) # column names
dependent <- paste(n[785:794], collapse = "+")
independent <- paste(n[!n %in% n[785:794]], collapse = "+")
f <- as.formula(paste(dependent, "~", independent))
f

#see if model works now
set.seed(2487)
system.time(
  for (i in 1:k){
    testingFold <- folds[[i]]
    cvTest <- train[testingFold,]
    cvTestLabel <- model_outputs[testingFold,]
    nn_model <- neuralnet(f, data = cvTest, hidden = c(700, 350, 70), stepmax = 8000, linear.output = FALSE)
    nn_predicted <- compute(nn_model, cvTest[, nn_model$model.list$variables])
```

```
    nn_predicted <- nn_predicted$net.result
    original_values <- max.col(cvTestLabel)
    nn_predicted_2 <- max.col(nn_predicted)
    accuracy<- mean(nn_predicted_2 == original_values)
   }
)
mean(accuracy)

# For PCA
#Create the folds
k_pca=6
num <- mnist[,1]
folds <- createFolds(num, k = k_pca)

# For PCA
#get features only
mnist_features <-mnist[,-1]
prcomp_model <- prcomp(mnist_features)
summary(prcomp_model)

# Calculate PCA Vals for 95% variance at 154 PCs
new_data<- as.data.frame(prcomp_model$x[, 1:154])
new_modeldata <- as.data.frame(cbind(model_outputs, new_data))

# new formula
n_pca<- names(new_modeldata) # column names
dependent_pca <- paste(n_pca[1:10], collapse = "+")
independent_pca <- paste(n_pca[!n_pca %in% n_pca[1:10]], collapse = "+")
f_pca <- as.formula(paste(dependent_pca, "~", independent_pca))
f_pca

#Model
system.time(
  for (i in 1:k_pca){
   testingFold <- folds[[i]]
   cvTestLabel_pca <- model_outputs[testingFold,]
   cvTest_pca <- new_modeldata[testingFold,]
   nn_model_pca <- neuralnet(f_pca, data = cvTest_pca, hidden = c(150, 70), stepmax = 5000,
linear.output = FALSE)
   nn_predicted_pca <- compute(nn_model_pca, cvTest_pca[,
nn_model_pca$model.list$variables])
   nn_predicted_pca <- nn_predicted_pca$net.result
   original_values_pca <- max.col(cvTestLabel_pca)
   nn_predicted_2_pca <- max.col(nn_predicted_pca)
```

```
    accuracy_pca <- mean(nn_predicted_2_pca == original_values_pca)
  }
)

mean(accuracy_pca)
```