Neural net without PCA

|  |
| --- |
| #Load the mnist data  library(readr)  library(caret)  library(nnet)  library(doParallel)  library(reshape)  #data load  Dat<-read.csv("c:/users/lux/Desktop/ML/P2/mnist.csv",header = FALSE)  n=nrow(Dat)  #Random selectio  trainIndex = sample(1:n, size = round(0.75\*n), replace=FALSE)  train\_orig = Dat[trainIndex ,]  test\_orig = Dat[-trainIndex ,]  train\_orig\_labels <- train\_orig[, 1]  train\_orig\_labels <- as.factor(train\_orig\_labels)  summary(train\_orig\_labels)  # split the training data into train and test to do local evaluation  set.seed(123)  rows <- sample(1:nrow(train\_orig), as.integer(0.7\*nrow(train\_orig)))  # Get train and test labels  train\_labels <- train\_orig[rows, 1]  test\_labels <- train\_orig[-rows, 1]  # convert the labels to factors  train\_labels <- as.factor(train\_labels)  # custom normalization function  normalize <- function(x) {  return(x / 255)  }  # create the train and test datasets and apply normalization  train\_norm <- as.data.frame(lapply(train\_orig[rows, -1], normalize))  test\_norm <- as.data.frame(lapply(train\_orig[-rows,-1], normalize))  summary(train\_norm$V100)  summary(train\_norm$V99)  # create the class indicator matrix  train\_labels\_matrix = class.ind(train\_labels)  head(train\_labels)  head(train\_labels\_matrix)  # train model without cross validation  set.seed(123)  startTime <- proc.time()  # Using K Cross validation  library(doParallel)  no\_cores <- detectCores()  cl <- makeCluster(no\_cores)  cl  registerDoParallel(cl)  # Set up training parameters  TrainingParameters <- trainControl(method = "cv", number = 3)  grid\_nn <- expand.grid(.size = c(1, 3, 5, 10),  .decay = 0)  grid\_nn  # use all of the given training data  train\_norm <- as.data.frame(lapply(train\_orig[, -1], normalize))  startTime <- proc.time()  set.seed(123)  nn2\_all <- train(train\_norm, train\_orig\_labels,  trControl= TrainingParameters,  method = "nnet",  tuneGrid = grid\_nn,  MaxNWts = 20000  )  proc.time() - startTime  nn2\_all  # normalize test data and predict on it  test\_norm <- as.data.frame(lapply(test\_orig, normalize))  NNPredictions <-predict(nn2\_all, test\_norm)  # output predictions for submission  predictions <- data.frame(ImageId=1:nrow(test\_orig),  Label=levels(train\_orig\_labels)[NNPredictions])  head(predictions)  write\_csv(predictions, "final\_caret\_nnet.csv")  # stop the cluster for parallel processing  stopCluster(cl)  plot(nn2\_all)  library(devtools)  source\_url('https://gist.githubusercontent.com/fawda123/7471137/raw/466c1474d0a505ff044412703516c34f1a4684a5/nnet\_plot\_update.r')  plot.nnet(nn2\_all)  test\_norm$V1<-as.factor(test\_norm$V1)  postResample(predictions$Label,test\_norm$V1)  postResample(predictions$Label,train\_norm$V1)  postResample(predictions$Label,test\_norm$V1) |

Neural net with PCA

PCA

|  |
| --- |
| dat<-read.csv("c:/users/lux/Desktop/ML/P2/mnist.csv",header = FALSE)  length(dat)#785 with first column  #Removing first column  #dat<-dat[,-1]  length(dat) #784  #Princomp function to perform pca on the data  mnist\_data=dat[,-(which((1:784)%%28<=2|(1:784)%%28>=26|1:784%/%28<=2|1:784%/%28>=26)+1)]  # only 530 variables include grey data .  prin<-princomp(dat[-1])  summary(prin)  #Report the number of principal components needed to account for 98% ofthe variance of the original data set.  PoV <- prin$sdev^2/sum(prin$sdev^2)  var<-data.frame(PoV[1:261])  #Question 3  #Contribution of all 261 components leading to 98%  write.csv(var,'pcaper.csv')  View(var)  #261 components contribute to 98%  sum(PoV[1:261])  cumsum(prin$sdev^2 / sum(prin$sdev^2)) |

NNET after PCA

|  |
| --- |
| library(devtools)  library(readr)  library(caret)  library(nnet)  library(doParallel)  library(reshape)  mnist<-read.csv("c:/users/lux/Desktop/ML/P2/mnist.csv",header = FALSE)  #ignoring the borders  mnist=mnist[,-(which((1:784)%%28<=2|(1:784)%%28>=26|1:784%/%28<=2|1:784%/%28>=26)+1)]  #Due to PCA performed and 96 % of variance explained by 261 components  Dat<-mnist[1:261]  n=nrow(Dat)  #Random selection  trainIndex = sample(1:n, size = round(0.75\*n), replace=FALSE)  train\_orig = Dat[trainIndex ,]  test\_orig = Dat[-trainIndex ,]  train\_orig\_labels <- train\_orig[, 1]  train\_orig\_labels <- as.factor(train\_orig\_labels)  summary(train\_orig\_labels)  # split the training data into train and test to do local evaluation  set.seed(123)  rows <- sample(1:nrow(train\_orig), as.integer(0.7\*nrow(train\_orig)))  # Get train and test labels  train\_labels <- train\_orig[rows, 1]  test\_labels <- train\_orig[-rows, 1]  # convert the labels to factors  train\_labels <- as.factor(train\_labels)  # custom normalization function  normalize <- function(x) {  return(x / 255)  }  # create the train and test datasets and apply normalization  train\_norm <- as.data.frame(lapply(train\_orig[rows, -1], normalize))  test\_norm <- as.data.frame(lapply(train\_orig[-rows,-1], normalize))  summary(train\_norm$V100)  summary(test\_norm$V99)  # create the class indicator matrix  train\_labels\_matrix = class.ind(train\_labels)  head(train\_labels)  head(train\_labels\_matrix)  library(doParallel)  no\_cores <- detectCores()  cl <- makeCluster(no\_cores)  cl  registerDoParallel(cl)  # Set up training parameters  TrainingParameters <- trainControl(method = "repeatedcv", number = 10)  grid\_nn <- expand.grid(.size = c(1, 3, 5, 10),  .decay = 0)  grid\_nn  # use all of the given training data  train\_norm <- as.data.frame(lapply(train\_orig[, -1], normalize))  startTime <- proc.time()  set.seed(123)  nn2 <- train(train\_norm, train\_orig\_labels,  trControl= TrainingParameters,  method = "nnet",  tuneGrid = grid\_nn,  MaxNWts = 20000  )  proc.time() - startTime  nn2  # normalize test data and predict on it  test\_norm <- as.data.frame(lapply(test\_orig, normalize))  NNPredictions <-predict(nn2, test\_norm)  # output predictions for submission  predictions <- data.frame(ImageId=1:nrow(test\_orig),  Label=levels(train\_orig\_labels)[NNPredictions])  head(predictions)  write\_csv(predictions, "caret\_nnetk.csv")  # stop the cluster for parallel processing  stopCluster(cl)  #accuracy  #nn2$bestTune  #summary(nn2)  plot(nn2)  #Load the mnist data  library(readr)  library(caret)  library(nnet)  library(doParallel)  #data load  Dat<-read.csv("c:/users/lux/Desktop/ML/P2/mnist.csv",header = FALSE)  n=nrow(Dat)  #Random selectio  trainIndex = sample(1:n, size = round(0.75\*n), replace=FALSE)  train\_orig = Dat[trainIndex ,]  test\_orig = Dat[-trainIndex ,]  train\_orig\_labels <- train\_orig[, 1]  train\_orig\_labels <- as.factor(train\_orig\_labels)  summary(train\_orig\_labels)  # split the training data into train and test to do local evaluation  set.seed(123)  rows <- sample(1:nrow(train\_orig), as.integer(0.7\*nrow(train\_orig)))  # Get train and test labels  train\_labels <- train\_orig[rows, 1]  test\_labels <- train\_orig[-rows, 1]  # convert the labels to factors  train\_labels <- as.factor(train\_labels)  # custom normalization function  normalize <- function(x) {  return(x / 255)  }  # create the train and test datasets and apply normalization  train\_norm <- as.data.frame(lapply(train\_orig[rows, -1], normalize))  test\_norm <- as.data.frame(lapply(train\_orig[-rows,-1], normalize))  summary(train\_norm$V100)  summary(train\_norm$V99)  # create the class indicator matrix  train\_labels\_matrix = class.ind(train\_labels)  head(train\_labels)  head(train\_labels\_matrix)  # train model  set.seed(123)  startTime <- proc.time()  nn = nnet(train\_norm, train\_labels\_matrix,  size = 1,  softmax = TRUE  )  proc.time() - startTime  nn  #Evaluating performance of the model  # get predictions  pred = predict(nn, test\_norm, type="class")  cbind(head(pred), head(test\_labels))  # evaluate the model  accuracy <- mean(pred == test\_labels)  print(paste('Accuracy:', accuracy))  # Using K Cross validation  library(doParallel)  no\_cores <- detectCores()  cl <- makeCluster(no\_cores)  cl  registerDoParallel(cl)  # Set up training parameters  TrainingParameters <- trainControl(method = "cv", number = 3)  grid\_nn <- expand.grid(.size = c(1, 3, 5, 10),  .decay = 0)  grid\_nn  # use all of the given training data  train\_norm <- as.data.frame(lapply(train\_orig[, -1], normalize))  startTime <- proc.time()  set.seed(123)  nn2 <- train(train\_norm, train\_orig\_labels,  trControl= TrainingParameters,  method = "nnet",  tuneGrid = grid\_nn,  MaxNWts = 20000  )  proc.time() - startTime  nn2  # normalize test data and predict on it  test\_norm <- as.data.frame(lapply(test\_orig, normalize))  NNPredictions <-predict(nn2, test\_norm)  # output predictions for submission  predictions <- data.frame(ImageId=1:nrow(test\_orig),  Label=levels(train\_orig\_labels)[NNPredictions])  head(predictions)  write\_csv(predictions, "caret\_nnet1.csv")  # stop the cluster for parallel processing  stopCluster(cl)  plot(nn2)  library(devtools)  source\_url('https://gist.githubusercontent.com/fawda123/7471137/raw/466c1474d0a505ff044412703516c34f1a4684a5/nnet\_plot\_update.r')  plot.nnet(nn2)  #RMSE  #postResample(NNPredictions,test\_norm$V1)  #postResample(NNPredictions,predictions$Label)  test\_norm$V1<-as.factor(test\_norm$V1)  postResample(predictions$Label,test\_norm$V1)  postResample(predictions$Label,train\_norm$V1)  postResample(predictions$Label,test\_norm$V1) |