ML Final Project

Sunny Lee

4/8/2021

For our project, we will be reducing the dimensionality of the MNIST handwritten digit dataset and seeing its effects on a neural network and the k-nearest neighbors algorithm. We will be using the built in prcomp function to calculate the principal components and using Tensorflow.Keras to build and train a neural network. This library will also be used to import the dataset. The k-nearest neighbors algorithm will also be using the built in function. To start, we import the dataset and divide the entire matrix by 255, in order to get a matrix of floats between 0 and 1 rather than a matrix of integers between 1 and 255. This dataset also comes as a 3 dimensional matrix, and thus we need to reshape the data into a 2 dimensional matrix. We will do this by taking our 2 dimensional picture of a digit and turning it into a 1 dimensional vector.

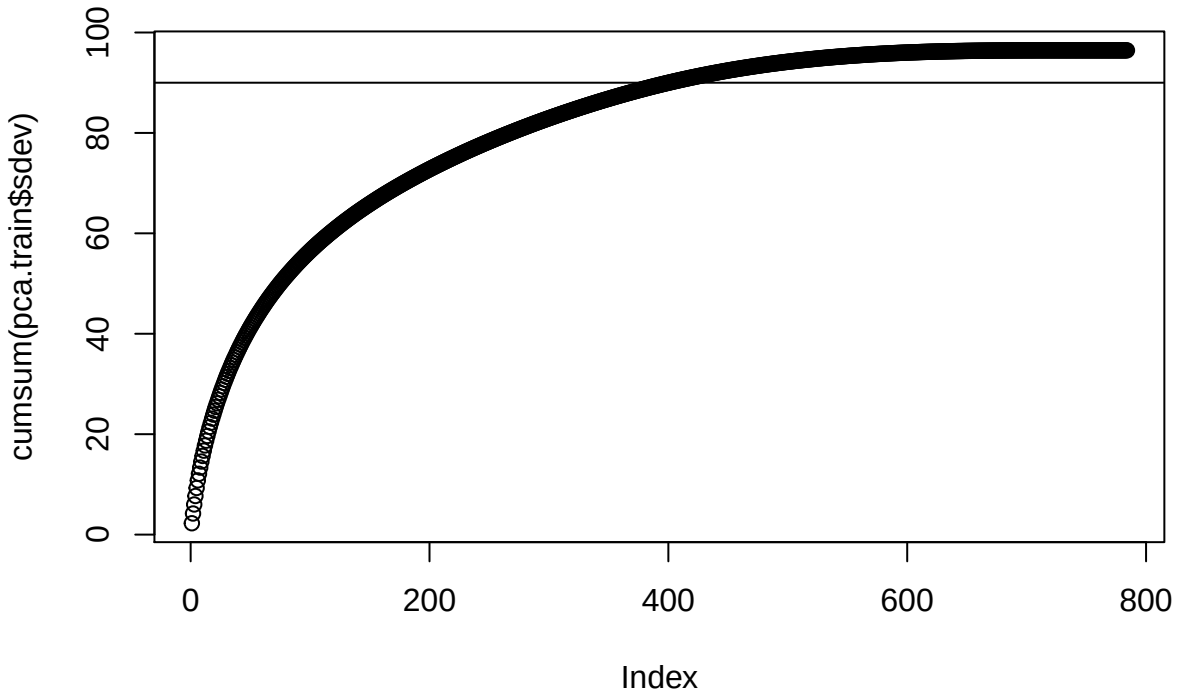
```
library(keras)
library(tensorflow)
library(dplyr)
library(class)

mnist <- dataset_mnist()
mnist$train$x <- mnist$train$x/255
mnist$test$x <- mnist$test$x/255
train <- matrix(mnist$train$x, 60000, 784)
test <- matrix(mnist$test$x, 10000, 784)
```

To calculate the principal components we take our train dataset and pass it through the prcomp function.

```
pca.train <- prcomp(train, scale = F)
pca.test <- prcomp(test, scale = F)
```

One way to see how much variation is captured within each principal component is to plot the cumulative sum of the singular values. By plotting the cumulative sum, we can see that the first few hundred principal components make up most of the variation.

```
train_proj <- train %*% pca.train$rotation[, 1:64]
test_proj <- test %*% pca.train$rotation[, 1:64]
plot(cumsum(pca.train$sdev))
abline(h = 90)
```



For our neural network, we will be using a 3-layer neural network. The input layers will be determined depending on the columns of the training and test datasets. The hidden layer will have 128 nodes and the activation function for those nodes will be the ReLU function. We will be using the sparse categorical crossentropy as our loss function and the adam optimizer. We will train our model for 5 epochs splitting our training data into a 80-20 split for our cross validation.

```
create_nn <- function(train, test)
{
  model <- keras_model_sequential() %>%
  layer_flatten(input_shape = c(ncol(train), 1)) %>%
  layer_dense(units = 128, activation = "relu") %>%
  layer_dense(10, activation = "softmax")

  model %>%
  compile(
    loss = "sparse_categorical_crossentropy",
    optimizer = "adam",
    metrics = "accuracy"
  )

  model %>%
  fit(
    x = train, y = mnist$train$y,
    epochs = 5,
    validation_split = .2,
    verbose = 2
  )
```

```
  predictions <- predict(model, test)
  model %>%
    evaluate(test, mnist$test$y, verbose = 0)
}
```

Now that we can build and train a neural network, we can train on our rank reduced dataset and our original dataset and compare how long they take to train.

```
#train and test the neural network with the rank reduced from 784 to 64
start <- proc.time()
create_nn(train_proj, test_proj)
```

```
##       loss   accuracy
## 0.07954907 0.97719997
```

```
proc.time() - start
```

```
##    user  system elapsed
##   12.83    1.36    7.99
```

```
#train and test the neural network on the original dataset
start <- proc.time()
create_nn(train, test)
```

```
##       loss   accuracy
## 0.08443659 0.97509998
```

```
proc.time() - start
```

```
##    user  system elapsed
##   23.45    1.76    9.09
```

From the two times above, even though we only completed 5 epochs, we see a significant increase in time going from our rank reduced dataset to our original dataset while losing a very small test accuracy. We can also see the effects of dimensionality reduction on the k-nearest neighbors algorithm. Since training the k-nearest neighbors algorithm is quite a bit slower than training the above neural network, we will be using a subset of the data for both the train and test sets.

```
size = 1000
```

Here we see the k-nearest neighbors algorithm on a subset of the original dataset:

```
start <- proc.time()
knn.pred <- knn(train[1:size, ], test[1:size, ], mnist$train$y[1:size], k = 1)
table(knn.pred, mnist$test$y[1:size])
```

```
##
## knn.pred   0   1   2   3   4   5   6   7   8   9
##        0  80   0   3   0   0   1   4   0   3   0
##        1   0 126   4   2   2   3   0   4   3   0
##        2   0   0  92   2   0   0   0   0   6   0
##        3   0   0   2  76   0   4   0   0   6   1
##        4   1   0   1   0  79   4   0   2   1   3
##        5   0   0   1  16   0  63   2   1   4   0
##        6   3   0   2   2   2   3  81   0   3   0
##        7   0   0   9   3   2   1   0  89   1   6
##        8   0   0   2   4   0   2   0   0  58   0
##        9   1   0   0   2  25   6   0   3   4  84
```

```
mean(knn.pred == mnist$test$y[1:size])
```

```
## [1] 0.828
```

```
proc.time() - start
```

```
##    user  system elapsed
##    3.95    0.00    3.95
```

And again on the same subset except with the dimension of the subset reduced:

```
start <- proc.time()
knn.pred.red <- knn(train_proj[1:size, ], test_proj[1:size, ], mnist$train$y[1:size], k = 1)
table(knn.pred.red, mnist$test$y[1:size])
```

```
##
## knn.pred.red   0   1   2   3   4   5   6   7   8   9
##            0  80   0   2   0   0   1   4   0   2   0
##            1   0 126   1   0   1   2   0   5   4   0
##            2   0   0  93   3   0   0   0   0   4   0
##            3   0   0   2  81   0   5   0   0   2   0
##            4   1   0   0   1  79   3   1   2   0   3
##            5   0   0   2  12   1  68   1   1   6   0
##            6   3   0   2   3   2   3  81   0   2   0
##            7   0   0   9   1   1   3   0  87   0   6
##            8   0   0   4   4   0   1   0   0  65   2
##            9   1   0   1   2  26   1   0   4   4  83
```

```
mean(knn.pred == mnist$test$y[1:size])
```

```
## [1] 0.828
```

```
proc.time() - start
```

```
##    user  system elapsed
##    0.06    0.00    0.06
```

We see both in the neural network and in the k-nearest neighbors that reducing the dimension of the training dataset yields very similar results while significantly reducing the time it takes to train the models.