Week 5 Assignment

By: Valon Tika

Summary and Problem Definition

As progression of data science continues, the need for predictive modeling with unstructured/unique data becomes a need in various markets within the industry. One aspect of this, computer vision analysis, can be used to leverage key insights and analytics for many tasks. One of the well-known datasets that have given data scientists the ability to expand on their skills for visual-based data is the MNIST (Modified National Institute of Standards and Technology) dataset. This dataset was released in 1999 that provides hand written images of numbers. This is considered as a benchmark in predictive modeling, specifically in classification modeling efforts such as random forests. The problem that I would like to try and solve is Kaggle's competition of creating a predictive model that can try and identify as many digits of the MNIST dataset using various model attempts while providing performance and output model metrics to determine the best model that can perform this task.

Research Design, Measurement and Statistical Method

The MNIST dataset consists of 70,000, 28 by 28-pixel images of digits hand-written between 0 to 9. Kaggle provided both a training and a testing dataset to use for the model efforts. The testing dataset that was provided had 28,000 records with 784 observations while the training dataset had 42,000 records with 785 observations. The goal will be to create CSV files of the output of each model attempt to submit to Kaggle along with running performance tests on each model. The use of principal component analysis and random forest classifiers will be used for this analysis.

Overview of Programming Work

A benchmarking method will be created first to set a guideline for the rest of the model efforts. For the benchmark, the first model that will run will be a random forest classifier on the training dataset using all 784 exploratory variables. After the model has produced a CSV file to submit to Kaggle, we can use that to help understand the performance of the other models. The second model will be a principal component analysis, which

will result with fewer than 784 exploratory observations that will represent at least 95% of variability of both the testing and training dataset together. The next step after running the PCA will be to run another random forest classifier on the identified exploratory variables.

The biggest design flaw with this is that fitting a PCA on both the train and test data is wrong due to the need for the test dataset to be hidden until prediction to actually test the model with a new set of data. The final set will be to run a PCA with just the training set without the testing dataset and rerun the random forest classifier to see what the TRAINING model tells us.

Review of Results – Recommendations

After running all three model attempts to review the performance and prediction performance, the results below show the end results:

Model Type (Submission to Kaggle)	Time to run model (seconds)
Benchmark Random Forest Classifier	6.222901821136475
PCA (training and testing data)	18.542973518371582
Random Forest (training and testing data)	19.519927740097046
Random Forest and PCA (training only)	26.143166303634644

The train time for the benchmark showed significantly slower compared to running the PCA and RF+PCA due to the amount of time that it didn't need to determine which principal components to put into the random forest's model. Running a PCA on the testing and training set resulted in keeping 154 principal components, which were fed in a second FR model. Again, the key fault was performing a model on both the test and training data. After the removal of the test data, the PCA plus random forest combination were ran again, which resulted in overall less time to perform (~11.91 total) than with the testing data included. The overall prediction accuracy of the model resulted that the benchmark random forest classifier performed the best, with an overall scored submitted through Kaggle of .938, compared to the other two submissions of the random forest plus PCA, which were ~.87. More about the prediction accuracy can be found on my Kaggle account of valontika16. The link of the submissions can be found here: https://www.kaggle.com/valontika16

Code Appendix:

```
#!/usr/bin/env python
# coding: utf-8

# In[18]:


import sklearn
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import PCA


# In[19]:


train = pd.read_csv('./train.csv')
test = pd.read_csv('./test.csv')


# In[20]:


train.head()


# In[21]:


train.shape


# In[22]:


X_train = train.iloc[:,1:].values
y_train = train['label'].values
X_test = test.values


# In[23]:


test.shape


# In[24]:


import time


# ### Step 1 : Fitting a random Forest

# In[25]:


rf1 = RandomForestClassifier(random_state=0)
# Calculate the time
```

```
start = time. time()
rf1.fit(X_train,y_train)
end = time. time()
print('The training time is ' + str(end - start))


# In[26]:


# Submission 1
pred = rf1.predict(X_test)
submission_1 = pd.DataFrame(columns=['ImageId','Label'])
submission_1['ImageId'] = range(len(X_test))
submission_1['Label'] = pred
submission_1.to_csv('Submission_1.csv',index=None)


# ### Step 2 : Fit PCA on train and test data

# In[27]:


np.random.seed(0)
matrix = np.vstack([X_train,X_test])

start = time. time()

pca = PCA(n_components=0.95,random_state=0)
pca.fit(matrix)

end = time.time()

print('The PCA Fitting time is ' + str(end - start))
print('Minimum components to fit PCA to at least 0.95 variability is '+
str(len(pca.explained_variance_ratio_.cumsum())))


# ### Step 3 : Fitting a random Forest using previous PCA

# In[28]:


start = time. time()

#Transorm X_train without fitting again
X_train_pca = pca.transform(X_train)
rf2 = RandomForestClassifier(random_state=0)
rf2.fit(X_train_pca,y_train)

end = time.time()
print('The training time is ' + str(end - start))


# In[32]:


# Submission 2
X_test_pca = pca.transform(X_test)
pred = rf2.predict(X_test_pca)
submission_2 = pd.DataFrame(columns=['ImageId','Label'])
submission_2['ImageId'] = range(len(X_test))
submission_2['Label'] = pred
submission_2.to_csv('Submission_2.csv',index=None)
```

```
# ### Step 4, AFTER SUBMISSION

# ### Step 5 , Fitting PCA only on the train data

# Fitting PCA on both the train and test data is wrong as test data have to hidden
till prediction, thus test data can't be used in PCA.fit

# In[30]:


np.random.seed(0)

start = time. time()

pca = PCA(n_components=0.95,random_state=0)
pca.fit(X_train)

#Transorm X_train without fitting again
X_train_pca = pca.transform(X_train)

rf3 = RandomForestClassifier(random_state=0)
rf3.fit(X_train_pca,y_train)

end = time.time()
print('The training time and PCA fitting is ' + str(end - start))
print('Minimum components to fit PCA to at least 0.95 variability is '+
str(len(pca.explained_variance_ratio_.cumsum())))


# In[31]:


# Submission 3
X_test_pca = pca.transform(X_test)
pred = rf3.predict(X_test_pca)
submission_3 = pd.DataFrame(columns=['ImageId','Label'])
submission_3['ImageId'] = range(len(X_test))
submission_3['Label'] = pred
submission_3.to_csv('Submission_3.csv',index=None)


# In[ ]:
```