Assignment_7

February 16, 2022

1 MSDS 422 Assignment 7: Digit Recognizer

Kay Quiballo | 02/16/2022

1.1 Management/Research Question

The classic MNIST Digit Recognizer problem is a competition on Kaggle.com, and you will compete in this competition. For this assignment, you will develop a classifier that may be used to predict which of the 10 digits is being written.

1.2 Requirements

1. Fit a random forest classifier using the full set of explanatory variables and the model training set (csv).
2. Record the time it takes to fit the model and then evaluate the model on the csvdata by submitting to Kaggle.com. Provide your Kaggle.com score and user ID.
3. Execute principal components analysis (PCA) on the combined training and test set data together, generating principal components that represent 95 percent of the variability in the explanatory variables. The number of principal components in the solution should be substantially fewer than the explanatory variables.
4. Record the time it takes to identify the principal components.
5. Using the identified principal components from step (2), use thecsvto build another random forest classifier.
6. Record the time it takes to fit the model and to evaluate the model on the csvdata by submitting to Kaggle.com. Provide your Kaggle.com score and user ID.
7. Use k-means clustering to group MNIST observations into 1 of 10 categories and then assign labels. (Follow the example here if needed: kmeans mnist.pdf Download kmeans mnist.pdf).kmeans mnist-2.pdf Download kmeans mnist-2.pdf
8. Submit the RF Classifier, the PCA RF, and k-means estimations to Kaggle.com, and provide screen snapshots of your scores as well as your Kaggle.com user name.
9. The experiment we have proposed has a major design flaw. Identify the flaw. Fix it. Rerun the experiment in a way that is consistent with a training-and-test regimen, and submit this to Kaggle.com. Report total elapsed time measures for the training set analysis. It is sufficient to run a single time-elapsed test for this assignment. In practice, we might consider the possibility of repeated executions of the relevant portions of the programs, much as the

Benchmark Example programs do. Some code that might help you with reporting elapsed total time follows.

start=datetime.now() rf2.fit(trainimages,labels) end=datetime.now() print(end-start)

[1]:
```
import pandas as pd
import numpy as np
import time

from sklearn.metrics import classification_report, confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
```

[2]:
```
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
```

[3]:
```
train.head()
```

[3]:
```
   label  pixel0  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  \
0      1       0       0       0       0       0       0       0       0
1      0       0       0       0       0       0       0       0       0
2      1       0       0       0       0       0       0       0       0
3      4       0       0       0       0       0       0       0       0
4      0       0       0       0       0       0       0       0       0

   pixel8  …  pixel774  pixel775  pixel776  pixel777  pixel778  pixel779  \
0       0  …         0         0         0         0         0         0
1       0  …         0         0         0         0         0         0
2       0  …         0         0         0         0         0         0
3       0  …         0         0         0         0         0         0
4       0  …         0         0         0         0         0         0

   pixel780  pixel781  pixel782  pixel783
0         0         0         0         0
1         0         0         0         0
2         0         0         0         0
3         0         0         0         0
4         0         0         0         0

[5 rows x 785 columns]
```

[4]:
```
test.head()
```

[4]:
```
   pixel0  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  pixel8  \
0       0       0       0       0       0       0       0       0       0
1       0       0       0       0       0       0       0       0       0
2       0       0       0       0       0       0       0       0       0
3       0       0       0       0       0       0       0       0       0
```

```
4       0       0       0       0       0       0       0       0       0

   pixel9  …  pixel774  pixel775  pixel776  pixel777  pixel778  pixel779  \
0       0  …         0         0         0         0         0         0
1       0  …         0         0         0         0         0         0
2       0  …         0         0         0         0         0         0
3       0  …         0         0         0         0         0         0
4       0  …         0         0         0         0         0         0

   pixel780  pixel781  pixel782  pixel783
0         0         0         0         0
1         0         0         0         0
2         0         0         0         0
3         0         0         0         0
4         0         0         0         0

[5 rows x 784 columns]
```

[5]:
```
X_train = train.drop('label', axis=1)
y_train = train['label']
```

1. Fit a random forest classifier using the full set of explanatory variables and the model training set (csv).

[6]:
```
start = time.process_time()

#create model
model1 = RandomForestClassifier()
#sm = SMOTE(sampling_strategy='minority', random_state=42)
#Xsm_train, ysm_train = sm.fit_resample(x_train, y_train)
model1 = model1.fit(X_train, y_train)
#prediction = model1.predict(X_train)
#print(classification_report(y_train, prediction))

run_time = time.process_time() - start

#eval
model = 'Random Forest Classifier'
kaggle_score = 0.96503
print(model, ",", run_time, "seconds, Score:", kaggle_score)
```

```
Random Forest Classifier , 31.406936 seconds, Score: 0.96503
```

[7]:
```
#output file
output = pd.DataFrame({'ImageId': np.linspace(1, 28000, 28000).astype(int),␣
↪'Label': model1.predict(test)})
output.to_csv('output_rfc.csv', index=False)
```

2. Record the time it takes to fit the model and then evaluate the model on the csvdata by submitting to Kaggle.com. Provide your Kaggle.com score and user ID. It took ~29 seconds to execute. The score for this model is 0.96503.

3. Execute principal components analysis (PCA) on the combined training and test set data together, generating principal components that represent 95 percent of the variability in the explanatory variables. The number of principal components in the solution should be substantially fewer than the explanatory variables.

[8]:
```
from sklearn.decomposition import PCA

X_combined = pd.concat([X_train, test])

start = time.process_time()
pca = PCA(n_components=10)
principalComponents = pca.fit_transform(X_combined)
print(time.process_time() - start)

principalDf_train = pd.DataFrame(data = principalComponents).head(42000)
principalDf_test = pd.DataFrame(data = principalComponents).tail(28000)

#confirming principal components represent 95 percent of the variability in the␣
↪explanatory variables
ex_variance=np.var(principalComponents,axis=0)
ex_variance_ratio = ex_variance/np.sum(ex_variance)
print(ex_variance_ratio)
0.19923771+0.14627718+0.12571351+0.1104602+0.099943+0.08801081+0.06701678+0.
↪0590723+0.05638858+0.04787952
```

```
10.451918
[0.19923771 0.14627718 0.12571351 0.1104602  0.0999434  0.08801081
 0.06701677 0.0590723  0.05638861 0.04787951]
```

[8]:
```
0.9999995899999999
```

4. Record the time it takes to identify the principal components. It took ~4 seconds to execute.

5. Using the identified principal components from step (2), use the csv to build another random forest classifier.

[9]:
```
start = time.process_time()

#create model
model2 = RandomForestClassifier()
#sm = SMOTE(sampling_strategy='minority', random_state=42)
#Xsm_train, ysm_train = sm.fit_resample(x_train, y_train)
```

```
model2 = model2.fit(principalDf_train, y_train)
#prediction = model2.predict(X_train)
#print(classification_report(y_train, prediction))

run_time = time.process_time() - start

#eval
model = 'Random Forest Classifier (PCA)'
kaggle_score = 0.90885
print(model, ",", run_time, "seconds, Score:", kaggle_score)
```

```
Random Forest Classifier (PCA) , 15.546256999999997 seconds, Score: 0.90885
```

[10]:

```
#output file
output = pd.DataFrame({'ImageId': np.linspace(1, 28000, 28000).astype(int),␣
 ↪'Label': model2.predict(principalDf_test)})
output.to_csv('output_pca.csv', index=False)
```

6. Record the time it takes to fit the model and to evaluate the model on the csvdata by submitting to Kaggle.com. Provide your Kaggle.com score and user ID. It took ~15 seconds to execute. The score for this model is 0.90885.

7. Use k-means clustering to group MNIST observations into 1 of 10 categories and then assign labels.

[11]:

```
#X_combined = pd.concat([X_train, test])
#pca = PCA(n_components=10)
#principalComponents = pca.fit_transform(X_combined)
#principalDf_train = pd.DataFrame(data = principalComponents).head(42000)
#principalDf_test = pd.DataFrame(data = principalComponents).tail(28000)


from sklearn.preprocessing import StandardScaler
X_std = StandardScaler().fit_transform(X_combined.values)

pca = PCA(n_components=10)
principalComponents = pca.fit_transform(X_std)
principalDf = pd.DataFrame(data = principalComponents)
#principalDf_train = pd.DataFrame(data = principalComponents).head(42000)
#principalDf_test = pd.DataFrame(data = principalComponents).tail(28000)

from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=10)
X_clustered = kmeans.fit_predict(principalDf)
X_clustered_train = X_clustered[:42000]
X_clustered_test = X_clustered[42000:]
```

```
len(X_clustered_train)
len(X_clustered_test)
len(y_train)
```

[11]: 42000

[12]:
```
start = time.process_time()

model3 = RandomForestClassifier()
model3 = model3.fit(X_clustered_train.reshape(-1, 1), y_train)

run_time = time.process_time() - start

#eval
model = 'KMeans'
kaggle_score = 0
print(model, ",", run_time, "seconds, Score:", kaggle_score)
```

```
KMeans , 1.1798689999999965 seconds, Score: 0
```

[13]:
```
#output file
output = pd.DataFrame({'ImageId': np.linspace(1, 28000, 28000).astype(int),␣
↪'Label': model3.predict(X_clustered_test.reshape(-1, 1))})
output.to_csv('output_kmeans.csv', index=False)
```

8. Submit the RF Classifier, the PCA RF, and k-means estimations to Kaggle.com, and provide screen snapshots of your scores as well as your Kaggle.com user name. It took ~1 second to execute. The score for this model is 0.53025.

9. The experiment we have proposed has a major design flaw. Identify the flaw. Fix it. Rerun the experiment in a way that is consistent with a training-and-test regimen, and submit this to Kaggle.com. Report total elapsed time measures for the training set analysis. It is sufficient to run a single time-elapsed test for this assignment. In practice, we might consider the possibility of repeated executions of the relevant portions of the programs, much as the Benchmark Example programs do. Some code that might help you with reporting elapsed total time follows.

[14]:
```
#start=datetime.now()
#rf2.fit(trainimages,labels)
#end=datetime.now()
#print(end-start)
```

To implement consistent training-and-testing regimen we need to split the training data into training/testing subset and us cross validation to check validity of models.

[15]:
```python
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_train, y_train,␣
 ↪test_size=0.20, shuffle=False)

start = time.process_time()
model1 = RandomForestClassifier()
model1 = model1.fit(X_train, y_train)
run_time = time.process_time() - start

prediction = model1.predict(X_test)
print(classification_report(y_test, prediction))

#eval
model = 'Random Forest + CV'
kaggle_score = 0
print(model, ",", run_time, "seconds, Score:", kaggle_score)
```

```
              precision    recall  f1-score   support

           0       0.97      0.98      0.98       852
           1       0.98      0.99      0.98       950
           2       0.96      0.96      0.96       801
           3       0.97      0.95      0.96       882
           4       0.97      0.96      0.97       811
           5       0.97      0.95      0.96       747
           6       0.97      0.99      0.98       831
           7       0.98      0.98      0.98       884
           8       0.95      0.96      0.95       802
           9       0.95      0.94      0.95       840

    accuracy                           0.97      8400
   macro avg       0.97      0.97      0.97      8400
weighted avg       0.97      0.97      0.97      8400

Random Forest + CV , 22.077422999999996 seconds, Score: 0
```

[16]:
```python
#output file
output = pd.DataFrame({'ImageId': np.linspace(1, 28000, 28000).astype(int),␣
 ↪'Label': model1.predict(test)})
output.to_csv('output_rfc_cv.csv', index=False)
```

It took ~10 second to execute. The score for this model is 0.95414.

2 EDA Continued

2.1 Requirements

1. Conduct your analysis using a cross-validation design.
2. Conduct / refine EDA.
3. Conduct Design of Experiments to evaluate the performance of various neural networks by changing the layers and nodes. Tested neural network structures should be explored within a benchmark experiment, a 2x2 completely crossed design. An example of a completely crossed designed with {2, 5} layers and {10,20} nodes follows. Layers Nodes Time Training Accuracy Testing Accuracy 2 10 63.61 0.935 0.927 2 20 115.25 0.967 0.952 5 10 74.28 0.944 0.933 5 20 75.1 0.964 0.952
4. Due to the time required to fit each neural network, we will observe only one trial for each cell in the design.
5. You will build your models on csv and submit your forecasts for test.csv to Kaggle.com, providing your name and user ID for each experimental trial.
6. Evaluate goodness of fit metrics on the training and validation sets.
7. Provide a multi-class confusion matrix.
8. Discuss how your models performed. In summary, this assignment asks you to fit a number of neural networks, comparing processing time and performance across experimental treatments. Processing time will be recorded for the fitting on the train.csv. Kaggle.com accuracy scores will be reported for all benchmarks.

[17]:
```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Lambda, Flatten, Conv2D, MaxPool2D,␣
↪MaxPooling2D
#from keras.optimizers import Adam,RMSprop
from sklearn.model_selection import train_test_split
from keras import backend as K
from keras.preprocessing.image import ImageDataGenerator
from tensorflow import keras
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn import neural_network
from sklearn import  metrics
```

Neural Networks

[18]:
```
Y = train['label'][:10000] # use more number of rows for more training
X = train.drop(['label'], axis = 1)[:10000] # use more number of rows for more␣
↪training
x_train, x_val, y_train, y_val = train_test_split(X, Y, test_size=0.20,␣
↪random_state=42)
```

```
mlp = neural_network.MLPClassifier()
mlp.fit(x_train, y_train)
accuracy_score(y_val, mlp.predict(x_val))

#time
#training accuracy, goodness of fit, confusion matrix
#testing accuracy, goodness of fit, confusion matrix
#kaggle score
```

[18]: 0.9115

[19]:
```
#Code inspired by Dhairya Kothari
i = 0
df = pd.DataFrame(columns =␣
↪['alpha','max_iter','train_acc','test_acc','train_time'])
for a in [0.00001,0.0001,0.001,0.01, 0.1, 1, 10]:
    for mi in [10,100,200,500,1000,2000]:
        st = time.time()
        mlp = neural_network.MLPClassifier(alpha=a, max_iter=mi)
        mlp.fit(x_train, y_train)
        end = time.time() - st

        acc_tr = accuracy_score(y_train, mlp.predict(x_train)) # Train Accuracy
        acc = accuracy_score(y_val, mlp.predict(x_val)) # Test Accuracy
        df.loc[i] = [a,mi,acc_tr,acc,end]
        i=i+1
df
```

```
/Users/kagenquiballo/opt/anaconda3/lib/python3.8/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and
the optimization hasn't converged yet.
  warnings.warn(
/Users/kagenquiballo/opt/anaconda3/lib/python3.8/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and
the optimization hasn't converged yet.
  warnings.warn(
/Users/kagenquiballo/opt/anaconda3/lib/python3.8/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and
the optimization hasn't converged yet.
  warnings.warn(
/Users/kagenquiballo/opt/anaconda3/lib/python3.8/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and
the optimization hasn't converged yet.
```

```
  warnings.warn(
/Users/kagenquiballo/opt/anaconda3/lib/python3.8/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and
the optimization hasn't converged yet.
  warnings.warn(
/Users/kagenquiballo/opt/anaconda3/lib/python3.8/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and
the optimization hasn't converged yet.
  warnings.warn(
/Users/kagenquiballo/opt/anaconda3/lib/python3.8/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and
the optimization hasn't converged yet.
  warnings.warn(
/Users/kagenquiballo/opt/anaconda3/lib/python3.8/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

[19]:

	alpha	max_iter	train_acc	test_acc	train_time
0	0.00001	10.0	0.985750	0.9020	3.610248
1	0.00001	100.0	0.994125	0.9225	7.491536
2	0.00001	200.0	0.986125	0.9060	9.062621
3	0.00001	500.0	0.993375	0.9255	7.808027
4	0.00001	1000.0	0.987375	0.9170	12.447070
5	0.00001	2000.0	0.988125	0.9145	10.421733
6	0.00010	10.0	0.987000	0.9015	3.471869
7	0.00010	100.0	0.996750	0.9130	12.642729
8	0.00010	200.0	0.992000	0.9120	7.791306
9	0.00010	500.0	0.986000	0.9110	7.693944
10	0.00010	1000.0	0.991750	0.9115	7.677173
11	0.00010	2000.0	0.992125	0.9175	9.278460
12	0.00100	10.0	0.989375	0.9085	3.007733
13	0.00100	100.0	0.995750	0.9270	7.722763
14	0.00100	200.0	0.997125	0.9180	10.089150
15	0.00100	500.0	0.993250	0.9275	8.761061
16	0.00100	1000.0	0.992750	0.9185	11.146565
17	0.00100	2000.0	0.993125	0.9195	11.509155
18	0.01000	10.0	0.989125	0.9025	2.655989
19	0.01000	100.0	0.991750	0.9165	8.365441
20	0.01000	200.0	0.994750	0.9280	8.798387
21	0.01000	500.0	1.000000	0.9090	9.610706
22	0.01000	1000.0	0.988750	0.9100	8.931914
23	0.01000	2000.0	0.993375	0.9120	10.037698

```
24   0.10000     10.0  0.982000  0.9000   3.051688
25   0.10000    100.0  0.994375  0.9175   8.651698
26   0.10000    200.0  0.996250  0.9055   8.631246
27   0.10000    500.0  0.995750  0.9185   9.375137
28   0.10000   1000.0  0.981375  0.9045   8.325731
29   0.10000   2000.0  0.992000  0.9135   8.095484
30   1.00000     10.0  0.981250  0.8975   2.486773
31   1.00000    100.0  0.988375  0.9335  20.053615
32   1.00000    200.0  0.994625  0.9140  10.778344
33   1.00000    500.0  0.995000  0.9285   9.244834
34   1.00000   1000.0  0.991250  0.9060   9.153392
35   1.00000   2000.0  0.989000  0.9145   7.688175
36  10.00000     10.0  0.976750  0.8960   2.797680
37  10.00000    100.0  0.996875  0.9460  26.003817
38  10.00000    200.0  0.996125  0.9445  29.535222
39  10.00000    500.0  1.000000  0.9605  37.070370
40  10.00000   1000.0  0.999750  0.9555  37.885697
41  10.00000   2000.0  0.996250  0.9460  40.010744
```

The highest test accuracy is 0.9605 which took about 37 seconds and had parameters alpha=10 and max_iter=500.

[21]:
```
mlp = neural_network.MLPClassifier(alpha=10, max_iter=500)
mlp.fit(x_train, y_train)
accuracy_score(y_val, mlp.predict(x_val))
```

[21]:
```
0.955
```

[24]:
```
i = 0
df = pd.DataFrame(columns =␣
 ↪['hidden_layer_sizes','train_acc','test_acc','train_time'])
for hidden_layer_sizes in [(2,),(5,),(10,),(20,),(50,),(100,)]:
        st = time.time()
        mlp = neural_network.MLPClassifier(alpha=10, max_iter=500,␣
 ↪hidden_layer_sizes=hidden_layer_sizes)
        mlp.fit(x_train, y_train)
        end = time.time() - st

        acc_tr = accuracy_score(y_train, mlp.predict(x_train)) # Train Accuracy
        acc = accuracy_score(y_val, mlp.predict(x_val)) # Test Accuracy
        df.loc[i] = [hidden_layer_sizes,acc_tr,acc,end]
        i=i+1
df
```

[24]:
```
  hidden_layer_sizes  train_acc  test_acc  train_time
0               (2,)   0.669000    0.6140   32.658367
1               (5,)   0.912250    0.8475   32.694383
```

```
2                (10,)   0.960750    0.8780   29.958114
3                (20,)   0.995125    0.9300   34.440675
4                (50,)   0.999875    0.9520   27.132379
5               (100,)   0.997125    0.9555   39.140181
```

[26]:
```
i = 0
df = pd.DataFrame(columns =
 ↪['hidden_layer_sizes','train_acc','test_acc','train_time'])
for hidden_layer_sizes in [(100,),(500,),(1000,)]:
        st = time.time()
        mlp = neural_network.MLPClassifier(alpha=10, max_iter=500,
 ↪hidden_layer_sizes=hidden_layer_sizes)
        mlp.fit(x_train, y_train)
        end = time.time() - st

        acc_tr = accuracy_score(y_train, mlp.predict(x_train)) # Train Accuracy
        acc = accuracy_score(y_val, mlp.predict(x_val)) # Test Accuracy
        df.loc[i] = [hidden_layer_sizes,acc_tr,acc,end]
        i=i+1
df
```

[26]:

	hidden_layer_sizes	train_acc	test_acc	train_time
0	(100,)	0.982125	0.9295	30.166693
1	(500,)	0.994625	0.9495	130.197035
2	(1000,)	0.982250	0.9435	202.346827

The best testing accuracy is 0.9495 from 500 layers.

[27]:
```
mlp = neural_network.MLPClassifier(alpha=10, max_iter=500,
 ↪hidden_layer_sizes=(500,))
mlp.fit(x_train, y_train)
print(confusion_matrix(y_val, mlp.predict(x_val)))
```

```
[[184   1   0   0   0   0   1   0   0   0]
 [  0 206   1   0   0   0   0   2   0   1]
 [  2   2 212   0   1   0   0   1   2   0]
 [  2   0   4 177   0   3   0   0   1   3]
 [  0   0   0   0 185   0   1   0   0   2]
 [  0   2   0   5   1 182   2   0   1   1]
 [  3   0   0   1   1   0 185   0   0   0]
 [  0   1   7   2   2   0   0 214   0   7]
 [  0   2   1   2   0   0   2   1 186   3]
 [  1   1   0   2   2   1   1   1   0 183]]
```

From the confusion matrix we can see that the main diagonal has the majority of counts indicating that the model is correctly predicting the label for the majority of records.

[28]:
```python
#output file
output = pd.DataFrame({'ImageId': np.linspace(1, 28000, 28000).astype(int),␣
↪'Label': mlp.predict(test)})
output.to_csv('output_mlp.csv', index=False)
```

Our kaggle score is: 0.95160 This indicate that the neural network model has a high accuracy in prediction rate for the testing data.

[ ]:
```python

```