Wanat_Assignment_5_final

July 28, 2019

0.1 Assignment 5: Principal Component Analysis

[1]:
```
# import base packages into the namespace for this program
from sklearn.datasets import fetch_openml
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
#from sklearn.cross_validation import cross_val_score
from sklearn.model_selection import cross_val_predict
from sklearn import metrics
from sklearn.pipeline import make_pipeline
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report
import time      #measure time of calculations



# seed value for random number generators to obtain reproducible results
RANDOM_SEED = 1
```

According to the assignment instructions, the following parameters were used: max_features = 'sqrt', bootstrap = TRUE, n_estimators = 10

0.2 Import data set and prepare for analysis

[2]:
```
# fetch dataset and save to object mnist
mnist = fetch_openml('mnist_784', version=1)
```

[3]:
```
# save data variables and target to X and y objects, respectively
X, y = mnist["data"], mnist["target"]
```

[4]:
```
print('The shape of X is: {}'.format(X.shape))
print('The shape of 7 is: {}'.format(y.shape))
```

```
The shape of X is: (70000, 784)
The shape of 7 is: (70000,)
```

[5]:
```
print('The datatype of X is: {}'.format(X.dtype))
print('The datatype of y is: {}'.format(y.dtype))
```

```
The datatype of X is: float64
The datatype of y is: object
```

[6]:
```
# split X and Y into train and test sets
# X_train and y_train will contain 60,0000
# X_test and y_test will contain 10,0000
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]
```

[7]:
```
# starter code provided to prepare data with correct data type
# Convert to 4-dim array by reshaping
# Divide by 255 to normalize data
X_train = X_train.astype(np.float32).reshape(-1,28*28)/255.0
X_test = X_test.astype(np.float32).reshape(-1,28*28)/255.0
y_train = y_train.astype(np.int32)
y_test = y_test.astype(np.int32)
```

[8]:
```
X_train[5,].size
```

[8]:
```
784
```

0.3 Part 1: Random Forest Classifier

Fit a random forest classifier using the full set of 784 explanatory variables and the model development set of 60,000 observations. Record the time it takes to fit the model and evaluate the model on the holdout data. Assess classification performance using the F1 score.

[9]:
```
#Begin timing the process
#Instantiate Random Forest Classifier
#Fit the model with the X_train and y_train data
#Use the X_test data to predict the y_test data
#End timing the process

t0 = time.time()
rfc = RandomForestClassifier(n_estimators=10, n_jobs=-1,␣
 ↪random_state=RANDOM_SEED, max_features='sqrt')
rfc.fit(X_train, y_train)
y_test_predict = rfc.predict(X_test)
t1 = time.time()
print("Time to fit and evaulate model: {:.3f} seconds".format((t1 - t0)))
rfc_time = t1 - t0
```

```
Time to fit and evaulate model: 1.502 seconds
```

[10]:
```
#Calculate the F1 score by comparing the predicted y_test against the true␣
 ↪y_test data
```

```
rfc_f1_score = metrics.f1_score(y_test, y_test_predict, average='micro')
print('\n----------------------------')
print('\nClassifier evaluation for Random Forest:')
print(rfc)
print('F1 score: {:.3f}'.format(rfc_f1_score))
```

```
----------------------------

Classifier evaluation for Random Forest:
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='sqrt', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=-1,
                       oob_score=False, random_state=1, verbose=0,
                       warm_start=False)
F1 score: 0.949
```

[11]:
```
#Print the classification report.
#This report indicates the precision, recall, f1-score and
#the number of variables measured for each target variable
print(classification_report(y_test, y_test_predict))
```

```
              precision    recall  f1-score   support

           0       0.94      0.99      0.96       980
           1       0.99      0.99      0.99      1135
           2       0.93      0.96      0.95      1032
           3       0.92      0.93      0.93      1010
           4       0.94      0.95      0.95       982
           5       0.93      0.93      0.93       892
           6       0.98      0.96      0.97       958
           7       0.97      0.94      0.95      1028
           8       0.94      0.92      0.93       974
           9       0.93      0.92      0.93      1009

    accuracy                           0.95     10000
   macro avg       0.95      0.95      0.95     10000
weighted avg       0.95      0.95      0.95     10000
```

0.4 Part 2: Principal Component Analysis

Execute principal component analysis (PCA) on the full set of 70,000 generating principal components that represent 95 percent of the variability in the explanatory variables. Record the time it takes to identify the principal components.

[12]:
```
#Begin timing the process
#Instantiate PCA specifying the ratio of variance desired
#Fit the PCA with the X_train data
#Transform the X_train and X_test - apply the dimensionality reduction
#Do not fit_transform the X_test data or X_test will be reduced different than␣
↪X_train
#End timing the process

t0 = time.time()
pca = PCA(n_components=0.95)
X_train_reduced = pca.fit_transform(X_train)
X_test_reduced = pca.transform(X_test)
t1 = time.time()
print('Time to identify principal components and transform variables: {:.3f}'.
↪format(t1 - t0))
pca_time = t1 - t0
```

```
Time to identify principal components and transform variables: 2.116
```

[13]:
```
# The explained variance ratio of each principal component indicates
# the proportion of the datasets variance that lies along
# the axis of each principal component.

pca.explained_variance_ratio_
```

[13]:
```
array([0.09704707, 0.07095943, 0.06169203, 0.05389408, 0.04868803,
       0.04312216, 0.03271934, 0.0288389 , 0.02762022, 0.02356997,
       0.02109188, 0.02022984, 0.01715809, 0.01692112, 0.01578638,
       0.01482951, 0.01324557, 0.012769  , 0.01187259, 0.01152684,
       0.01066162, 0.0100671 , 0.0095357 , 0.00912541, 0.00883405,
       0.00839317, 0.00812577, 0.00786365, 0.0074473 , 0.00690857,
       0.00658091, 0.00648149, 0.00602614, 0.00586581, 0.0057002 ,
       0.00543629, 0.00505783, 0.00487857, 0.00481429, 0.00472263,
       0.00456746, 0.00444836, 0.00418501, 0.00398215, 0.00384973,
       0.00375102, 0.00362009, 0.00351591, 0.00340056, 0.00321873,
       0.00319016, 0.00312806, 0.00295982, 0.00288954, 0.00284131,
       0.00271435, 0.00269521, 0.00258473, 0.0025377 , 0.0024478 ,
       0.00240506, 0.00239261, 0.00230407, 0.00221532, 0.00213721,
       0.00207226, 0.00203042, 0.00196782, 0.00192852, 0.00188632,
       0.00186976, 0.00181083, 0.00177562, 0.00174898, 0.00165758,
       0.00163893, 0.00161462, 0.00155116, 0.00147613, 0.00143176,
       0.00142094, 0.00141153, 0.00140174, 0.00135737, 0.00133847,
       0.00132397, 0.00130157, 0.00125872, 0.00122828, 0.00121585,
       0.00117034, 0.00114873, 0.00113244, 0.00110885, 0.00109002,
       0.00106923, 0.00104195, 0.00104007, 0.00101256, 0.00100527,
       0.00098402, 0.00094969, 0.00094134, 0.00091616, 0.00090785,
       0.00089687, 0.00086539, 0.00085517, 0.00084562, 0.00082249,
       0.00079158, 0.00078594, 0.00078461, 0.00076883, 0.00076401,
```

```
0.00075309, 0.00073678, 0.00072713, 0.00071965, 0.00070681,
0.00069542, 0.00069216, 0.0006833 , 0.00067406, 0.00066688,
0.00064526, 0.00063559, 0.00063164, 0.00062293, 0.00060529,
0.00060359, 0.00059448, 0.00058831, 0.00058652, 0.00058134,
0.00057683, 0.00056537, 0.00055476, 0.00053517, 0.00052592,
0.00052509, 0.00051025, 0.00050297, 0.00050108, 0.00049871,
0.00049107, 0.00048553, 0.00048285, 0.00047401, 0.00046835,
0.0004666 , 0.00046332, 0.00045929, 0.00045038], dtype=float32)
```

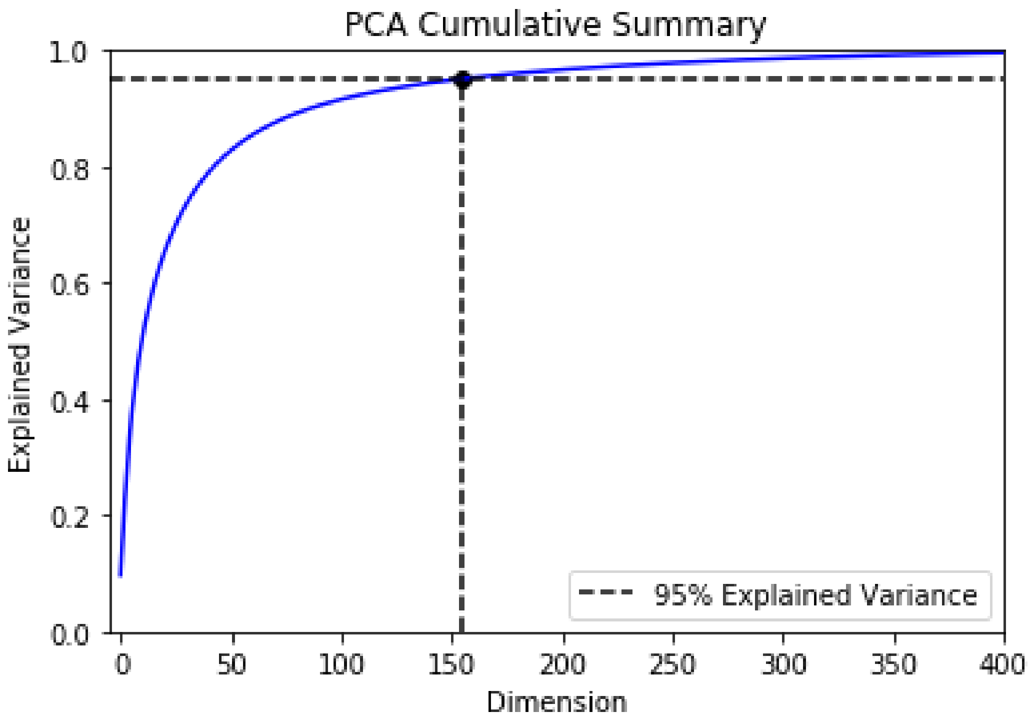
[14]:
```
print('The number of principal components that explain 95% of the variablity:␣
↪{}'.format(pca.n_components_))
```

```
The number of principal components that explain 95% of the variablity: 154
```

[15]:
```
import matplotlib.pyplot as plt
%matplotlib inline

#plot the explained variance as a function of the number of dimensions
pca_all = PCA()
pca_all.fit(X_train)

cumsum = np.cumsum(pca_all.explained_variance_ratio_)
plt.plot(cumsum, c='blue')
ax = plt.gca()
ax.set_xlim([-5,400])
ax.set_ylim([0,1.0])
plt.plot([-5, 400], [0.95, 0.95], color='k', linestyle='--', label = '95%␣
↪Explained Variance')
#plt.axhline(y = 95, color='k', linestyle='--', label = '95% Explained␣
↪Variance')
plt.plot([154, 154], [0, 0.95], color='k', linestyle='--')
plt.scatter(154, 0.95, color='k')
plt.xlabel("Dimension")
plt.ylabel("Explained Variance")
plt.title("PCA Cumulative Summary")
plt.legend(loc='best')
plt.savefig('pca_cumulative_summary.pdf')
plt.show()
```



0.5 Part 3: Random Forest Classifier with PCA components

Using the identified principal components from part 2, use the first 60,000 observations to build another random forest classifier. Record the time it takes to fit the model and to evaluate the model on the holdout data. Assess classification performance using the F1 score.

[16]:
```
#Instantiate the Random Forest Classifier
#Begin timing the process
#Fit the model with the reduced X_train and y_train data set
#With the X_test_reduced data, predict the y_test data
#End timing the process

rfc_pca = RandomForestClassifier(n_estimators=10, n_jobs=-1,␣
 ↪random_state=RANDOM_SEED, max_features='sqrt')
t0 = time.time()
rfc_pca.fit(X_train_reduced, y_train)
y_test_predict = rfc_pca.predict(X_test_reduced)
t1 = time.time()
print("Time to fit Random Forest Classifier with reduced data and evaulate␣
 ↪model: {:.3f} seconds".format((t1 - t0)))
rfc_pca_time = t1 - t0
```

```
Time to fit Random Forest Classifier with reduced data and evaulate model: 3.302
seconds
```

[17]:
```
#Calculate the F1 score by comparing the predicted y_test against the true␣
 ↪y_test data
rfc_pca_f1_score = metrics.f1_score(y_test, y_test_predict, average='micro')
print('\n----------------------------')
print('\nClassifier evaluation for Random Forest:')
print(rfc)
print('F1 score: {:.3f}'.format(rfc_pca_f1_score))
```

```
----------------------------

Classifier evaluation for Random Forest:
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='sqrt', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=-1,
                       oob_score=False, random_state=1, verbose=0,
                       warm_start=False)
F1 score: 0.894
```

[18]:
```
#Print the classification report.
#This report indicates the precision, recall, f1-score and
#the number of variables measured for each target variable
print(classification_report(y_test, y_test_predict))
```

```
              precision    recall  f1-score   support

           0       0.89      0.96      0.93       980
           1       0.97      0.98      0.97      1135
           2       0.86      0.90      0.88      1032
           3       0.83      0.89      0.86      1010
           4       0.86      0.91      0.88       982
           5       0.87      0.80      0.83       892
           6       0.93      0.92      0.92       958
           7       0.93      0.89      0.91      1028
           8       0.90      0.81      0.85       974
           9       0.90      0.85      0.88      1009

    accuracy                           0.89     10000
   macro avg       0.89      0.89      0.89     10000
weighted avg       0.90      0.89      0.89     10000
```

0.6 Part 4: Compare Performance of the Two Modeling Approaches

[19]:
```
#add the time for PCA and Random Forest Classifier with reduced data set
rfc2_time = pca_time + rfc_pca_time

#create a dictionary of times and F1 scores
summary_data = {
    'Time (seconds)' : [round(rfc_time, 3), round(rfc2_time, 3)],
    'F1 Score' : [round(rfc_f1_score, 3), round(rfc_pca_f1_score, 3)]
}
```

[20]:
```
summary_data
```

[20]:
```
{'Time (seconds)': [1.502, 5.418], 'F1 Score': [0.949, 0.894]}
```

[21]:
```
#covert the dictionary into a dataframe and add labels
summary_df = pd.DataFrame(summary_data, columns = ['Time (seconds)', 'F1␣
↪Score'])
summary_df.rename(index = {0: "RandomForest",
                     1:"RandomForestReduced"},
                                  inplace = True)
summary_df
```

[21]:

	Time (seconds)	F1 Score
RandomForest	1.502	0.949
RandomForestReduced	5.418	0.894

Even though PCA reduced the complexity of the MNIST data set by reducing the number of components from 784 to 154, the total time needed to fit a Random Forest Classifier and evaluate the data increased. The reduced complexity of the data set is making it harder for the Random Forest Classifier to generate trees. The Random Forest Classifier generates a split by looking for a feature and threshold that results in the purest subsets. The criteria to evaluate this split is the Gini index, which is a criterion to minimize the probability of misclassification. The training data with the reduced components contains less information for the Random Forest Classifier to optimally operate, therefore it takes more time.

The F1 score for the RandomForestReduced model is less than the RandomForest model, but this is expected as we fit the PCA analysis to explain only 95% of the variance.

0.7 Part 5: Re-design the Experiment and Repeat Analysis

Instead of using Random Forest Classifier with PCA, lets use a different classification algorithm to see if there is an improvement with PCA. A Logistic Classification model will be used to evaluate PCA.

[22]:
```
from sklearn.linear_model import LogisticRegression

#Instantiate the Logistic Regression Classifier
#Begin timing the process
#Fit the model with the X_train and y_train data set
#With the X_test data, predict the y_test data
#End timing the process
```

```
lr = LogisticRegression(multi_class="multinomial", solver="lbfgs",␣
 ↪random_state=RANDOM_SEED, max_iter=100)
t0 = time.time()
lr.fit(X_train, y_train)
y_test_predict = lr.predict(X_test)
t1 = time.time()
print("Time to fit Logistic Regression and evaulate model: {:.3f} seconds".
 ↪format((t1 - t0)))
lr_time = t1 - t0
```

```
Time to fit Logistic Regression and evaulate model: 8.749 seconds

/Users/jmwanat/anaconda3/lib/python3.6/site-
packages/sklearn/linear_model/logistic.py:947: ConvergenceWarning: lbfgs failed
to converge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
```

[23]:
```
#Calculate the F1 score by comparing the predicted y_test against the true␣
 ↪y_test data
lr_f1_score = metrics.f1_score(y_test, y_test_predict, average='micro')
print('\n----------------------------')
print('\nClassifier evaluation for Logistic Regression:')
print(rfc)
print('F1 score: {:.3f}'.format(lr_f1_score))
```

```
----------------------------

Classifier evaluation for Logistic Regression:
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='sqrt', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=-1,
                       oob_score=False, random_state=1, verbose=0,
                       warm_start=False)
F1 score: 0.926
```

[24]:
```
#Print the classification report.
#This report indicates the precision, recall, f1-score and
#the number of variables measured for each target variable
print(classification_report(y_test, y_test_predict))
```

```
              precision    recall  f1-score   support
```

```
              0       0.95      0.98      0.96       980
              1       0.96      0.98      0.97      1135
              2       0.93      0.90      0.91      1032
              3       0.90      0.91      0.90      1010
              4       0.94      0.93      0.93       982
              5       0.90      0.87      0.89       892
              6       0.94      0.95      0.94       958
              7       0.94      0.92      0.93      1028
              8       0.88      0.88      0.88       974
              9       0.92      0.92      0.92      1009

       accuracy                           0.93     10000
      macro avg       0.92      0.92      0.92     10000
   weighted avg       0.93      0.93      0.93     10000
```

[25]:
```
#Instantiate the Logistic Regression Classifier
#Begin timing the process
#Fit the model with the X_train_reduced (from PCA) and y_train data set
#With the X_test_reduced (from PCA) data, predict the y_test data
#End timing the process


lr_pca = LogisticRegression(multi_class="multinomial", solver="lbfgs",␣
 ↪random_state=RANDOM_SEED, max_iter=100)
t0 = time.time()
lr_pca.fit(X_train_reduced, y_train)
y_test_predict = lr_pca.predict(X_test_reduced)
t1 = time.time()
print("Time to fit and evaulate model: {:.3f} seconds".format((t1 - t0)))
lr_pca_time = t1 - t0
```

```
Time to fit and evaulate model: 2.968 seconds

/Users/jmwanat/anaconda3/lib/python3.6/site-
packages/sklearn/linear_model/logistic.py:947: ConvergenceWarning: lbfgs failed
to converge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
```

[26]:
```
#Calculate the F1 score by comparing the predicted y_test against the true␣
 ↪y_test data
lr_pca_f1_score = metrics.f1_score(y_test, y_test_predict, average='micro')
print('\n----------------------------')
print('\nClassifier evaluation for Logistic Regression:')
print(rfc)
print('F1 score: {:.3f}'.format(lr_pca_f1_score))
```

```
----------------------------

Classifier evaluation for Logistic Regression:
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='sqrt', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=-1,
                       oob_score=False, random_state=1, verbose=0,
                       warm_start=False)
F1 score: 0.923
```

[27]:
```
#Print the classification report.
#This report indicates the precision, recall, f1-score and
#the number of variables measured for each target variable
print(classification_report(y_test, y_predict))
```

```
              precision    recall  f1-score   support

           0       0.95      0.98      0.96       980
           1       0.97      0.98      0.97      1135
           2       0.92      0.90      0.91      1032
           3       0.90      0.91      0.90      1010
           4       0.93      0.93      0.93       982
           5       0.90      0.87      0.88       892
           6       0.94      0.95      0.94       958
           7       0.93      0.92      0.93      1028
           8       0.88      0.89      0.89       974
           9       0.90      0.90      0.90      1009

    accuracy                           0.92     10000
   macro avg       0.92      0.92      0.92     10000
weighted avg       0.92      0.92      0.92     10000
```

0.8 Summary of Time and F1 Score

[28]:
```
#Add Logistic Regression Classification time and F1 score to summary dataframe
summary_df.loc['LogisticRegression'] = [round(lr_time, 3), round(lr_f1_score,␣
 ↪3)]
summary_df.loc['LogisticRegressionReduced'] = [round(lr_pca_time, 3),␣
 ↪round(lr_pca_f1_score, 3)]
```

[29]:
```
summary_df
```

[29]:
```
                           Time (seconds)  F1 Score
RandomForest                        1.502     0.949
```

```
RandomForestReduced                 5.418     0.894
LogisticRegression                  8.749     0.926
LogisticRegressionReduced           2.968     0.923
```

Using the Logistic Regression Classifier with reduced complexity training data obtained from PCA reduced the calculation time over 50% and resulted in almost the same F1 score (0.923 vice 0.926). The Logistic Regression Classifier has a lower F1 score compared to Random Forest Classification that used the full set of 784 explanatory variables. The decision to use one classification algorithm verses another will depend on whether time or accuracy is more important. The decision to use PCA will depend on the type of algorithm used and not all algorithms benefit from a dataset with reduced dimensionality.

0.9 Plot of MNIST digit: Comparison of 784 dimensions and 154 dimensions

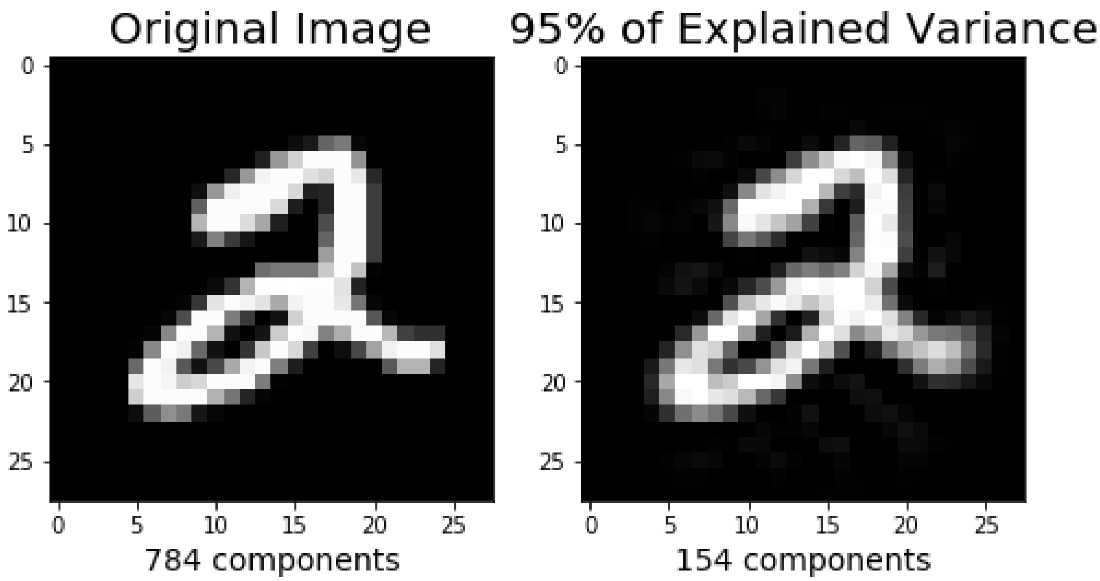
[30]:
```
#The code below was copied and modified from this web site:
#https://github.com/mGalarnyk/Python_Tutorials/blob/master/Sklearn/PCA/
 ↪PCA_Image_Reconstruction_and_such.ipynb?
 ↪source=post_page---------------------------

#Decompress the reduced data set from 154 dimensions back to 784
approximation = pca.inverse_transform(X_train_reduced)

#plot and compare the same digit using 784 component vs 154 components
plt.figure(figsize=(8,4));

# Original Image
plt.subplot(1, 2, 1);
plt.imshow(mnist.data[5].reshape(28,28),
              cmap = plt.cm.gray, interpolation='nearest',
              clim=(0, 255));
plt.xlabel('784 components', fontsize = 14)
plt.title('Original Image', fontsize = 20);

# 154 principal components
plt.subplot(1, 2, 2);
plt.imshow(approximation[5].reshape(28,28)*255,
              cmap = plt.cm.gray, interpolation='nearest',
              clim=(0, 255));
plt.xlabel('154 components', fontsize = 14)
plt.title('95% of Explained Variance', fontsize = 20);
plt.savefig('MNIST_digit.pdf')
plt.show()
```


Original Image
95% of Explained Variance
784 components
154 components

[ ]: