Face recognition using Principal Component Analysis (PCA)

Anthony Ibarra

Computer Engineer, UIC College of Engineering

University of Illinois at Chicago

28aibarra@gmail.com

April 2022

I. Introduction

Regarding how one can implement PCA (Principal Component Analysis). In this project PCA was used for face recognition from the dataset from the sklearn python module.

II. Method description

Using the sklearn module in python and the matplotlib to be able to plot the images. Through the use of sklearn and based on the scikit learn website which the module has access to the dataset of faces for facial recognition PCA algorithm for implementation of machine learning mlpclassifier as well as the classification report to easily see the changes throughout each run.

III. Experimental results

At the start of the experiment, I used the easiest dataset where there is only one person thus ending up in a 100% accuracy and then increasing the dataset in terms of size and people. As the data and pictures started to increase in number the accuracy no longer remained as 100 percent and varied throughout the iterations although it should be noted that it did not go below 80%.

Throughout the experiment the whole iteration and CA process has taken longer as the dataset started to increase. Eventually taken up to three if not more times longer than the early test and still obtaining the same accuracy as compared to those earlier runs.

IV. Conclusion

In conclusion with the use of PCA or principal component analysis, one should keep in mind that with larger and larger datasets then the time consumptions will also increase at a faster rate as well. Seeing as the complexity for PCA is $O(p^2n+p^3)$ thus increasing in an a very rapid pace. In the end when using or implementing PCA one would need to take into account how large their dataset would be ideal for their situation.

V. Reference

[1] F. a. V. G. a. G. A. a. M. V. Pedregosa, "Scikit-learn: Machine Learning in {P}ython," *Journal of Machine Learning Research,* vol. 12, pp. 2825--2830, 2011.

[2] L. a. W. Y. a. W. W. a. C. W. a. S. S. a. Z. Q. Liu, "Benchmarking Deep Learning Frameworks: Design Considerations,

Metrics and Beyond," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, 2018, pp. 1258-1269.

[3] Y. W. a. L. L. a. C. P. a. W. C. a. S. S. a. W. W. a. Q. Zhang, "A Comparative Measurement Study of Deep Learning as a Service Framework," *IEEE Transactions on Services Computing,* pp. 1-1, 2019.

[4] Y. a. a. C. W. a. S. S. a. L. L. Wu, "Experimental Characterizations and Analysis of Deep Learning Frameworks," in *2018 IEEE 38th International Conference on Big Data*, 2018.

VI. Appendix: Code List

```python
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_lfw_people
from sklearn.metrics import classification_report
from sklearn.decomposition import PCA
from sklearn.neural_network import MLPClassifier



# Load data
L_dataset = fetch_lfw_people(min_faces_per_person=500)

_, h, w = L_dataset.images.shape
X =  L_dataset.data
y = L_dataset.target
target_names = L_dataset.target_names

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)


n_components = 100
pca = PCA(n_components=n_components, whiten=True).fit(X_train)

X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)


print("Fitting the classifier to the training set")
clf = MLPClassifier(hidden_layer_sizes=(1024,), batch_size=256, verbose=True,
early_stopping=True).fit(X_train_pca, y_train)

y_pred = clf.predict(X_test_pca)
print(classification_report(y_test, y_pred, target_names=target_names))




def plot_gallery(images, titles, h, w, rows=3, cols=4):
    plt.figure()
    for i in range(rows * cols):
        plt.subplot(rows, cols, i + 1)
        plt.imshow(images[i].reshape((h, w)), cmap=plt.cm.gray)
```

```python
        plt.title(titles[i])
        plt.xticks(())
        plt.yticks(())
def titles(y_pred, y_test, target_names):
    for i in range(y_pred.shape[0]):
        pred_name = target_names[y_pred[i]].split(' ')[-1]
        true_name = target_names[y_test[i]].split(' ')[-1]
        yield 'predicted: {0}\ntrue: {1}'.format(pred_name, true_name)
print("print plot gallery")
prediction_titles = list(titles(y_pred, y_test, target_names))
plot_gallery(X_test, prediction_titles, h, w)
plt.show()
```