Part1: Labeled Faces in the Wild face recognition dataset

Attribute Information:

This dataset is a collection of JPEG pictures of famous people collected over the internet.

1.1. Load

Below are some basic information about the dataframe.

```
Total dataset size:
number_of_samples: 766
number_of_features: 1850
number_of_classes: 2
```

1.2. Split train/test data

We take 75% of the dataset for training and 25% for testing.

1.3. Standardizing Features

Multi-layer Perceptron is sensitive to **feature scaling**, so it is highly recommended to scale our data. For example, scale each attribute on the input vector X to [0, 1] or [-1, +1], or standardize it to have mean 0 and variance 1. Note that we must apply the same scaling to the test set for meaningful results. I use **StandardScaler** for standardization.

PCA is effected by scale so you need to scale the features in your data **before** applying PCA.

1.4. PCA

I use PCA for dimensionality reduction and computing the **eigenfaces** of our dataset. **Eigenfaces** is a method that is useful for face recognition and detection by determining the variance of faces in a collection of face images and use those variances to encode and decode a face in a machine learning way without the full information reducing computation and space complexity.

Below are some eigenfaces (reduced components of faces) after PCA.

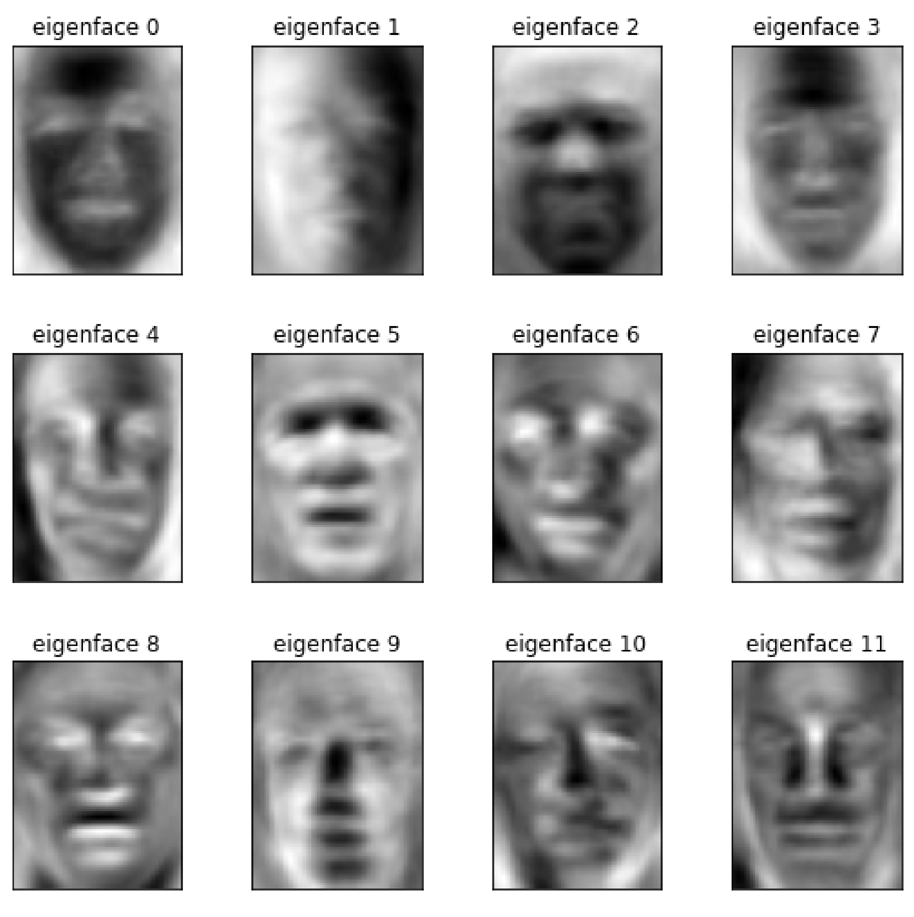


Figure 1: number of components = 150

I've tried other numbers for number of components but didn't achieve better results.

1.5. Multi-layer Perceptron Classification

I used the Class **MLPClassifier** which implements a multi-layer perceptron (MLP) algorithm that trains using Backpropagation.

MLP trains on two arrays: array X of size ($n\_samples$, $n\_features$), which holds the training samples represented as floating point feature vectors; and array y of size ($n\_samples$,), which holds the target values (class labels) for the training samples.

1.5.1. Parameter Tuning

The default *solver='adam'* works pretty well on relatively large datasets (with thousands of training samples or more) in terms of both training time and validation score. For small datasets, however, *'lbfgs'* can converge faster and perform better. <u>Since our dataframe is relatively small, I chose *solver='lbfgs'*</u>

Finding a reasonable regularization parameter α is best done using **GridSearchCV**, usually in the range **10.0**- np.arange(1, 7)**.

Using the information above, this is how I did my GridSearchCV:

```
clf_param_grid = {'solver': ['lbfgs'],
                  'hidden_layer_sizes':[[(i, j) for i in range(8,15) for j in
                                         range(2,6) if i>=j]],
                  'alpha': 10.0 ** -np.arange(1, 7),
                  'random_state':list(range(1,10))}
```

The result was:

```
1.0
{'alpha': 0.1, 'hidden_layer_sizes': (10, 2), 'random_state': 8, 'solver': 'lbfgs'}
```

As we can see this gave us 100% accuracy on our train dataset.

1.5.2. Training the Model

Using the best parameters found from GridSearchCV, I built the model.

```
MLPClassifier(alpha=0.1, hidden_layer_sizes=(10, 2), random_state=8,
              solver='lbfgs')
```

1.5.3. Result of the Prediction

After fitting (training), the model can predict labels for new samples. So I tested the model on the test dataset. The results are:

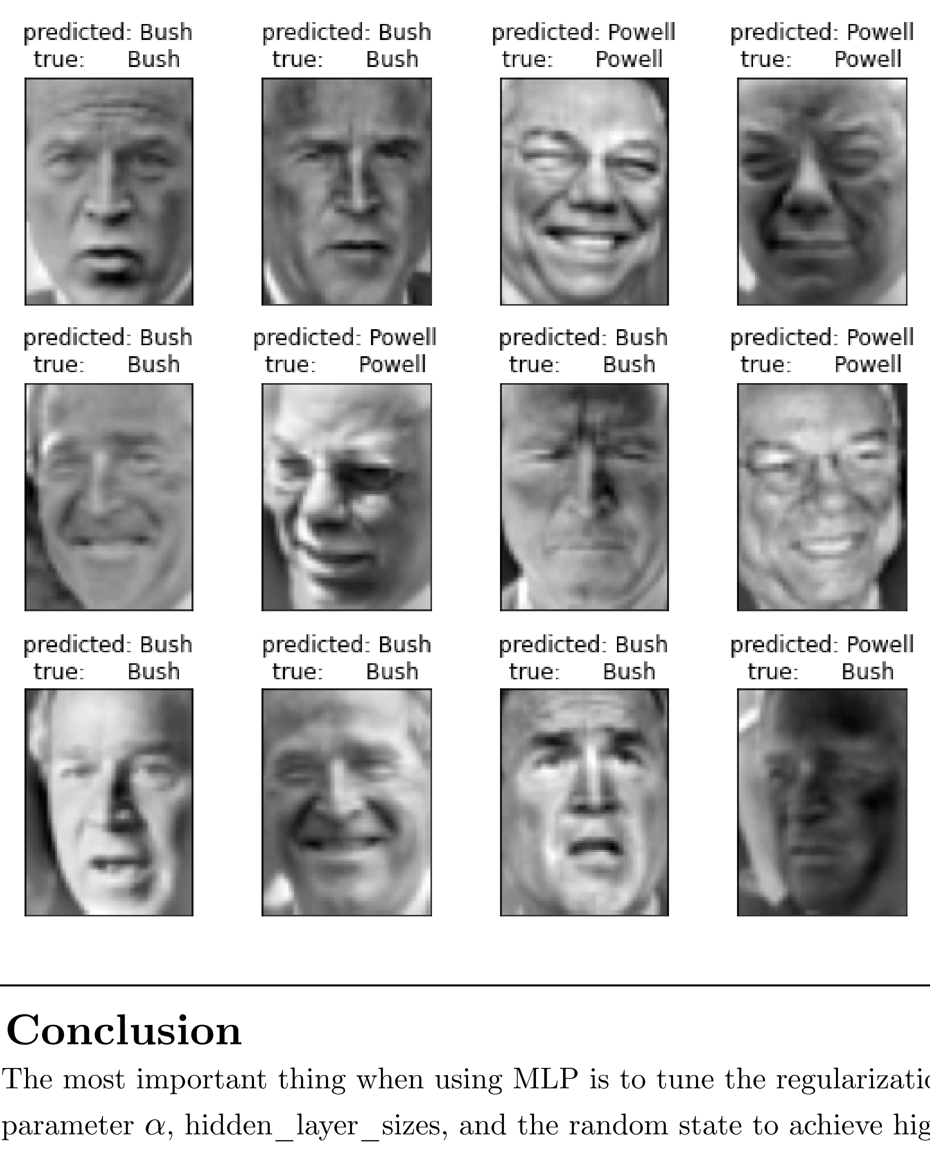
```
clf confusion_matrix :
 [[ 54   5]
 [  7 126]]
------------------------------
Accuracy: 93.75 %
Train accuracy: 100.0 %
Test accuracy: 93.75 %
------------------------------
classification_report:
              precision    recall  f1-score   support

           0       0.89      0.92      0.90        59
           1       0.96      0.95      0.95       133

    accuracy                           0.94       192
   macro avg       0.92      0.93      0.93       192
weighted avg       0.94      0.94      0.94       192
```

I've tried variations of different number of layers and nodes in each layer using the GridSearchCV, but couldn't improve the accuracy.

Here is the visualization of the prediction by plotting with Faces and train-test Prediction pairs.



1.6. Conclusion

- The most important thing when using MLP is to tune the regularization parameter α, hidden_layer_sizes, and the random state to achieve higher accuracy.
- After trying multiple grid searches, my model got **100% accuracy on train dataset** and **94% accuracy on test dataset**. Which I find to be a good accuracy.
- Although a 100% accuracy on the train dataset is likely an **overfit**, but since our model **performs well on the test dataset** we don't suffer from overfitting. This is closely related to the characteristics of our dataset.
- PCA in image classification can help us to reduce the dimension of our features and produces eigenfaces which are reduced components of faces.
- From the confusion matrix we have:
  - true negatives: 54,
  - false negatives: 7,
  - true positives: 126, and
  - false positives: 5.

  We can see that the false negatives and false positives are quite small and acceptable.
- From the classification report we have:
  - f1-score for the first class: 0.90
  - f1-score for the Second class: 0.95

  Recall that f1-score is the weighted average of *Precision* and *Recall*. Therefore, this score takes both false positives and false negatives into account. Thus our model performs better on recognizing the faces from the second class than the first class. Our f1-score is very close to 1 which again shows our model is performing very well.
- It's worth to mention that we could improve the accuracy of MPL even more with cross validation method, if we consider a wider grid search and do the search on more parameters.