
# A Machine Learning model for facial recognition using Labeled Faces in the Wild (lfw_people) dataset

**By Sanjeev Hirudayaraj**

**RIT Email ID**: sh9114@rit.edu

**Course**: DSCI 633: Foundations of Data Science
**Instructor**: Dr. Nidhi Rastogi
**Date**: 15 December 2023

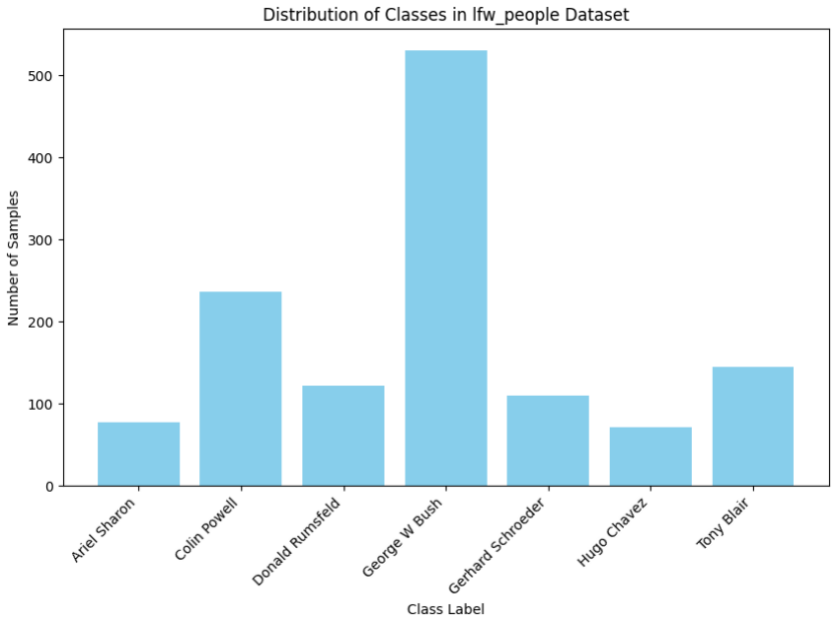
# I. Data Science Problem

The problem is to perform face identification—classifying faces into multiple classes via supervised learning.

# II. Data and Model Description

The LFW dataset is a collection of more than 13,000 images of faces collected from the web. Each face has been labeled with the name of the person pictured. The dataset has been widely used for face recognition research and benchmarking. I have used pre-processed funneled version called the lfw_people. This dataset contains 1288 samples, 1850 features and 7 classes.

a. Class Distribution

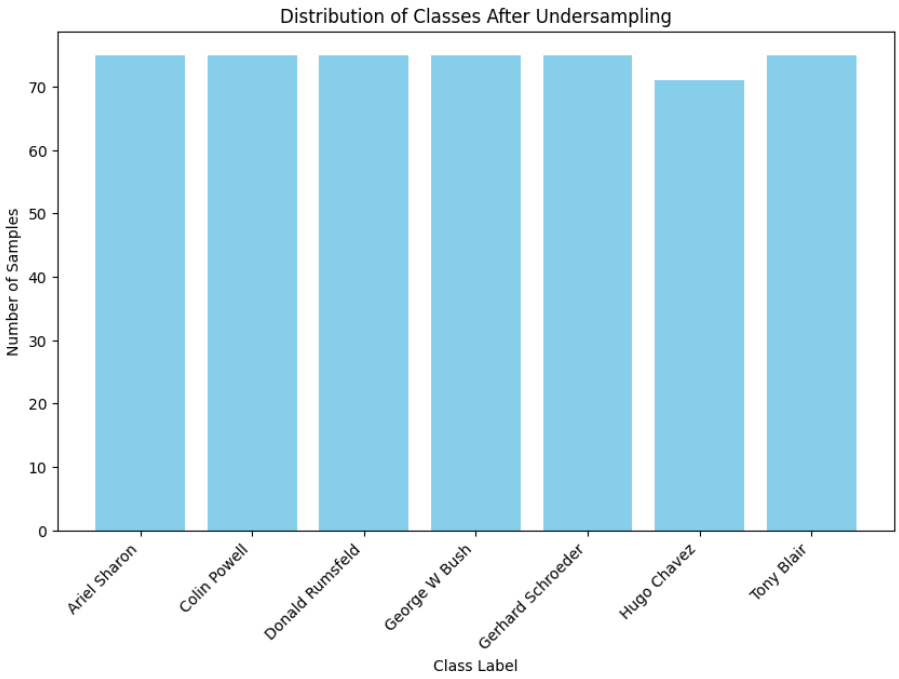



As we can see, the classes are unevenly distributed with George W Bush, and Colin Powell are over-represented. There are several methods including SMOTE to address this class imbalance. I have chosen the 'under sampling' method over SMOTE.

b. Under-Sampling

By randomly removing the over-represented class or classes, we can reduce the dominance of the over-represented class/es. This method is chosen over methods like SMOTE, where the under-represented samples are duplicated randomly by generating

synthetic samples. To avoid the problem of overfitting the model, under-sampling was chosen. Though there are still other viable methods like Weighted Loss where different weights are assigned to minority classes during training, to penalize misclassifying. However, Under-sampling the over-represented classes is a simpler method to work around the problem. Therefore, after the under-sampling the class distribution looked like this:



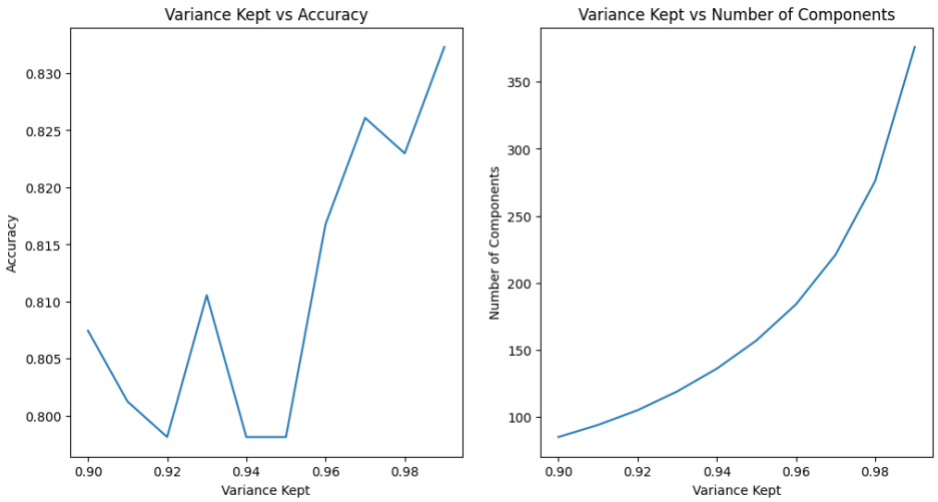

Principal Component Analysis (PCA)

Principal component analysis, or PCA, is a dimensionality reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.

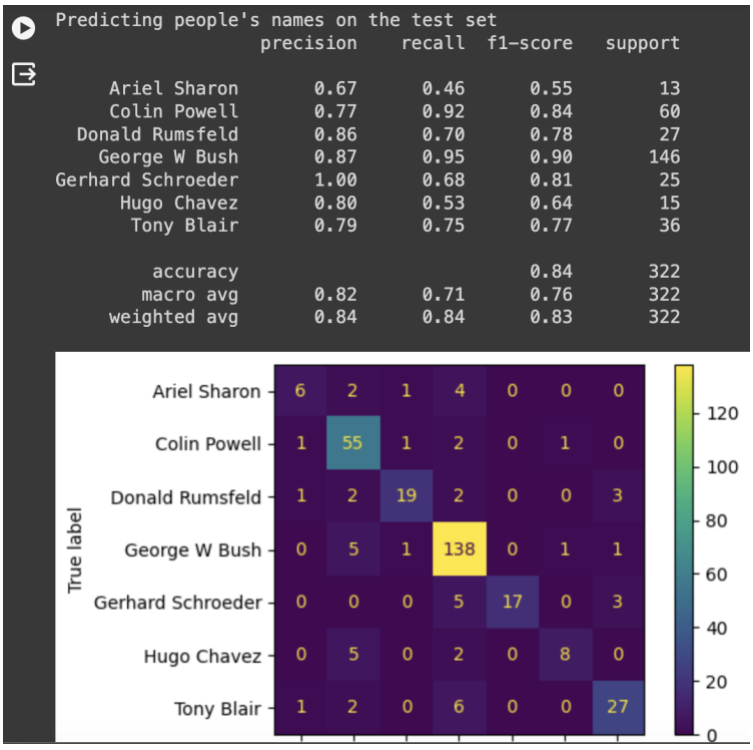
Reducing the number of variables of a data set naturally comes at the expense of accuracy, but the trick in dimensionality reduction is to trade a little accuracy for simplicity. Because smaller data sets are easier to explore and visualize and make analyzing data points much easier and faster for machine learning algorithms without extraneous variables to process.

Data Reduction with PCA

With PCA, there is a trade-off between the variance in the data, and accuracy. We can decide on the ‘sweet spot’ which gives a high level of accuracy with right percentage of variance kept in the data. In the graph below on the right, the smooth curve changes direction at about no. of components = 260, at about 98% variance kept, forming an elbow. By doing this, we can arrive at same levels of accuracy with reduced dimensions (262 exactly, from 966).

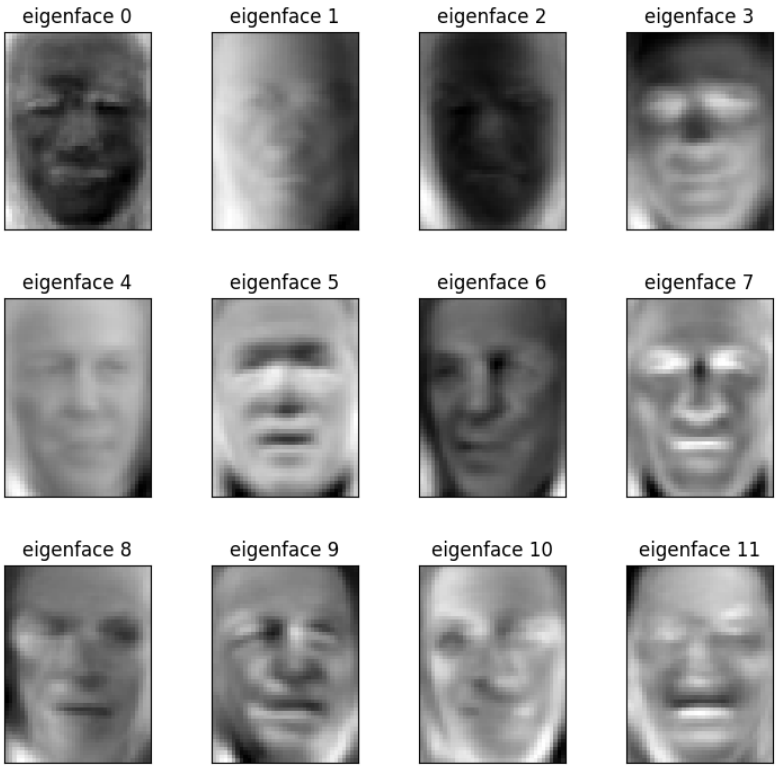



Quantitative Evaluation of the model



Eigenfaces

The basis of the eigenfaces method is the Principal Component Analysis (PCA). We start with a group of original face images and calculate the best vector system for image compression. The Principal Component Analysis is a method of projection to a subspace and is widely used in pattern recognition. An objective of PCA is the replacement of correlated vectors of large dimensions with the uncorrelated vectors of smaller dimensions. Another objective is to calculate a basis for the data set. Main advantages of the PCA are its low sensitivity to noise, the reduction of the requirements of the memory and the capacity, and the increase in the efficiency due to the operation in a space of smaller dimensions. The strategy of the Eigenfaces method consists of extracting the characteristic features on the face and representing the face in question as a linear combination of the so called 'eigenfaces' obtained from the feature extraction process. The principal components of the faces in the training set are calculated. Recognition is achieved using the projection of the face into the space formed by the eigenfaces. A comparison based on the Euclidian distance of the eigenvectors of the eigenfaces and the eigenface of the image under question is made. If this distance is small enough, the person is identified. On the other hand, if the distance is too large, the image is regarded as one that belongs to an individual for which the system must be trained. These are the eigenfaces in order of their importance:



# III. Analysis Strategy

The idea is to run four models (Logistic Regression, Random Forest, K-Nearest Neighbor & SVM) to get a baseline accuracy. Then reduce the dimensionality by doing a PCA, and running the same four models again, to see if there is difference in the accuracy metric. The choice of the above four models is purely based on convenience, mainly to find the effect of PCA on Accuracy. The results are given as a table below:

| | LOGISTIC REGRESSION | RANDOM FOREST | K-NN | SVM |
|---|---|---|---|---|
| PRE-PCA ACCURACY % | 82.61 | 65..84 | 57.76 | 76.71 |
| POST-PCA ACCURACY % (VARIANCE KEPT = 98%) | 84.16 | 53.11 | 56.83 | 76.4 |

At first glance, we see that 3 out of the 4 models have performed worse off after PCA than before, in terms of accuracy. But it is worth noting that, with reduced dimensions from 966 to 262 (almost 73%), we can achieve the equally high levels of accuracy with SVM, and even slightly better accuracy for Logistic Regression. This goes to prove that PCA can be a valuable pre-processing step when we deal with high number of features, and can help build a more efficient model.

# IV. Analysis Code

```
# Importing libraries
from time import time

import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import loguniform

from sklearn.datasets import fetch_lfw_people
from sklearn.decomposition import PCA
from sklearn.metrics import ConfusionMatrixDisplay, classification_report
from sklearn.model_selection import RandomizedSearchCV, train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC

# Extract from the dataset only pictures of people that have at least 70 different samples of the individual.
# Resize is the ratio used to resize each image
lfw_people = fetch_lfw_people(min_faces_per_person=70, resize=0.4)

# lfw_people.images is a 3D array representing the images in the dataset.
# The shape of this array is (number_of_samples, height, width).
# The code extracts these dimensions into the variables n_samples, h (height), and w (width).
n_samples, h, w = lfw_people.images.shape

# lfw_people.data is a 2D array where each row corresponds to a flattened version of an image.
# n_features is the number of features (or pixels) in each flattened image.
# The shape of lfw_people.data is (number_of_samples, number_of_features).
X = lfw_people.data
n_features = X.shape[1]

# The label to predict is the id of the person
y = lfw_people.target
target_names = lfw_people.target_names
n_classes = target_names.shape[0]

print("Total dataset size:")
print("n_samples: %d" % n_samples)
print("n_features: %d" % n_features)
print("n_classes: %d" % n_classes)

# Distribution of classes
```

```
class_counts = np.bincount(y)

# Plot the distribution of classes
plt.figure(figsize=(10, 6))
bars = plt.bar(range(len(class_counts)), class_counts, tick_label=target_names, color='skyblue')

# Rotate x-axis labels by 45 degrees
plt.xticks(rotation=45, ha='right')

plt.xlabel('Class Label')
plt.ylabel('Number of Samples')
plt.title('Distribution of Classes in lfw_people Dataset')
plt.show()

#Under-sampling
from imblearn.under_sampling import RandomUnderSampler
import numpy as np
import matplotlib.pyplot as plt

# Specify the desired range of samples
min_samples = 70
max_samples = 75

# Calculate the desired number of samples for each class within the specified range
desired_samples_per_class = {}
for class_idx in range(len(target_names)):
    class_count = np.sum(y == class_idx)
    if class_count > max_samples:
        desired_samples_per_class[class_idx] = max_samples
    elif class_count < min_samples:
        desired_samples_per_class[class_idx] = min_samples
    else:
        desired_samples_per_class[class_idx] = class_count

# Display the distribution before undersampling
class_counts_before = np.bincount(y)
plt.figure(figsize=(10, 6))
bars_before = plt.bar(range(len(class_counts_before)), class_counts_before,
tick_label=target_names, color='skyblue')
plt.xticks(rotation=45, ha='right')
plt.xlabel('Class Label')
plt.ylabel('Number of Samples')
plt.title('Distribution of Classes Before Undersampling')
plt.show()
```

```
# Undersample the specified classes
rus = RandomUnderSampler(sampling_strategy=desired_samples_per_class, random_state=42)
X_resampled, y_resampled = rus.fit_resample(X, y)

# Display the distribution after undersampling
class_counts_after = np.bincount(y_resampled)
plt.figure(figsize=(10, 6))
bars_after = plt.bar(range(len(class_counts_after)), class_counts_after,
tick_label=target_names, color='skyblue')
plt.xticks(rotation=45, ha='right')
plt.xlabel('Class Label')
plt.ylabel('Number of Samples')
plt.title('Distribution of Classes After Undersampling')
plt.show()


# Splitting the dataset into train and test
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42
)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

#LOGISTIC REGRESSION
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Assuming 'X' is your feature matrix and 'y' is your target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

# Create and train the model
model = LogisticRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

```
#RANDOM FOREST
from sklearn.ensemble import RandomForestClassifier

# Create and train the model
model = RandomForestClassifier()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")

#KNN
from sklearn.neighbors import KNeighborsClassifier

# Choose the number of neighbors (you may need to tune this parameter)
k_value = 5

# Create and train the KNN model
model = KNeighborsClassifier(n_neighbors=k_value)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")


#SVM
from sklearn.svm import SVC

# Create and train the model
model = SVC()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
```

```
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")

#PCA
from sklearn.decomposition import PCA
pca = PCA(n_components=262)
pca.fit(X_train)
print(pca.n_components_)

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

#Data reduction
n_components_list = [0.90, 0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98, 0.99]
accuracy_list = []
n_components_kept_list = []
for n_components in n_components_list:
    pca = PCA(n_components=n_components)
    pca.fit(X_train)
    X_train_pca = pca.transform(X_train)
    X_test_pca = pca.transform(X_test)
    svm_pca = SVC(kernel='linear')
    svm_pca.fit(X_train_pca, y_train)
    y_pred_pca = svm_pca.predict(X_test_pca)
    acc_pca = accuracy_score(y_test, y_pred_pca)
    accuracy_list.append(acc_pca)
    n_components_kept_list.append(pca.n_components_)
    print(f'Variance kept: {n_components}, Accuracy: {acc_pca:.4f}, Components kept:
{pca.n_components_}')

#plotting the graphs
plt.figure(figsize=(12,6))
plt.subplot(1,2,1)
plt.plot(n_components_list, accuracy_list)
plt.xlabel('Variance Kept')
plt.ylabel('Accuracy')
plt.title('Variance Kept vs Accuracy')
plt.subplot(1,2,2)
plt.plot(n_components_list,n_components_kept_list)
plt.xlabel('Variance Kept')
plt.ylabel('Number of Components')
plt.title('Variance Kept vs Number of Components')
plt.show()
```

```
#Eigenfaces
n_components = 262

print(
    "Extracting the top %d eigenfaces from %d faces" % (n_components, X_train.shape[0])
)
pca = PCA(n_components=n_components, svd_solver="randomized", whiten=True).fit(X_train)

eigenfaces = pca.components_.reshape((n_components, h, w))
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)


#Train a SVM Classifier model
print("Fitting the classifier to the training set")
param_grid = {
    "C": loguniform(1e3, 1e5),
    "gamma": loguniform(1e-4, 1e-1),
}
clf = RandomizedSearchCV(
    SVC(kernel="rbf", class_weight="balanced"), param_grid, n_iter=10
)
clf = clf.fit(X_train_pca, y_train)
print("Best estimator found by grid search:")
print(clf.best_estimator_)

#Quantitative evaluation of the model
print("Predicting people's names on the test set")
y_pred = clf.predict(X_test_pca)

print(classification_report(y_test, y_pred, target_names=target_names))
ConfusionMatrixDisplay.from_estimator(
    clf, X_test_pca, y_test, display_labels=target_names, xticks_rotation="vertical"
)
plt.tight_layout()
plt.show()

def plot_gallery(images, titles, h, w, n_row=3, n_col=4):
    """Helper function to plot a gallery of portraits"""
    plt.figure(figsize=(1.8 * n_col, 2.4 * n_row))
    plt.subplots_adjust(bottom=0, left=0.01, right=0.99, top=0.90, hspace=0.35)
    for i in range(n_row * n_col):
        plt.subplot(n_row, n_col, i + 1)
        plt.imshow(images[i].reshape((h, w)), cmap=plt.cm.gray)
```

```
        plt.title(titles[i], size=12)
        plt.xticks(())
        plt.yticks(())

def title(y_pred, y_test, target_names, i):
    pred_name = target_names[y_pred[i]].rsplit(" ", 1)[-1]
    true_name = target_names[y_test[i]].rsplit(" ", 1)[-1]
    return "predicted: %s\ntrue:      %s" % (pred_name, true_name)


prediction_titles = [
    title(y_pred, y_test, target_names, i) for i in range(y_pred.shape[0])
]

plot_gallery(X_test, prediction_titles, h, w)

eigenface_titles = ["eigenface %d" % i for i in range(eigenfaces.shape[0])]
plot_gallery(eigenfaces, eigenface_titles, h, w)

plt.show()

#LOGIT POST PCA (VAR = 98%)
# Create and train the model
model = LogisticRegression()
model.fit(X_train_pca, y_train)

# Make predictions
y_pred = model.predict(X_test_pca)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")

#RANDOM FOREST POST PCA
# Create and train the model
model = RandomForestClassifier()
model.fit(X_train_pca, y_train)

# Make predictions
y_pred = model.predict(X_test_pca)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

```
#KNN POST PCA
# Create and train the model
# Choose the number of neighbors (you may need to tune this parameter)
k_value = 5

# Create and train the KNN model
model = KNeighborsClassifier(n_neighbors=k_value)
model.fit(X_train_pca, y_train)

# Make predictions
y_pred = model.predict(X_test_pca)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")

#SVM POST PCA
# Create and train the model
model = SVC()
model.fit(X_train_pca, y_train)

# Make predictions
y_pred = model.predict(X_test_pca)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```