(https://colab.research.google.com/github/esamsalam/UTS_ML2019_ID13216914/blob/master/A2.ipynb)

Faces recognition example using eigenfaces and SVMs

RuiZheWang 13163617 Esam Salam 13216914

Plain Link to GitHub: *https://github.com/esamsalam/UTS_ML2019_ID13216914/blob/master/A2.ipynb (https://github.com/esamsalam/UTS_ML2019_ID13216914/blob/master/A2.ipynb)*

1. Introduction

We chose a data mining project utilizing an existing package for facial recognition. Facial recognition models are one of the most striking and successful areas in deep learning due to its wide variety of uses in the real world, producing rich results that can be useful in various applications. For our practical project, we use the eigenfaces with SVMs to study and analyze the problem of unconstrained facial recognition.

In the past few decades, there has been a significant rise of interest and research into facial recognition in videos and images due to its crucial importance and uses; and while these have been very productive, there has only been incremental increase in quality due to much of the work being restricted to constrained settings. This means that due to most primitive models being constrained to specific settings, they are unable to adjust to real world scenarios. For example, most primitive models are unable to distinguish between lighting, facial expressions, poses and so on. For our chosen eigenfaces model however, we focus on the 'real-life' aspect of data mining, by curating the model to adhere to the aforementioned aspects of lighting, facial expressions and poses. We believe this could foster the development process of the previous general techniques and produce more flexible facial recognition systems, ones that are able to be utilized in real time, using more advanced and scalable systems.

Facial recognition is especially important in modern society as it can be used for various tasks, mainly for security reasons. Some of these reasons could include providing access to restricted access areas such as airports, government restricted areas, hospitals, administrations e.t.c., it can also be used to provide access to personal products such as for unlocking phones or gates and doors at one's homes, it can furthermore be utilized to check attendance, for example on exams; facial recognition can be used to verify student's identity and so forth. For these multitude of reasons, facial recognition is a crucial part of modern society as it aids in tasks that are too mundane and monotonous and thus increasing efficiency in these tasks, such as for example, spotting high value criminal targets between crowds of people.

For our eigenfaces algorithm, we input pre-processed mugshots (well- aligned and properly cropped images) from the University of Massachusetts database called the Labelled Faces in the Wild (LFW) database, a globally recognized facial recognition benchmark. The database contains more than 13,000 compiled images categorized by each person’s name. Although there are 1680 people with two or more distinct images in the dataset, we have chosen to only include people with over 50 or more distinct images for our classification. Utilizing this input, the algorithm would be trained to output a prediction of the face from a given test set, hopefully classifying the correct person with the given test faces. This is an example of supervised learning as our model is given each person’s name and face and is expected to correspondingly classify each test image for output.

2. Exploration

Before discussing the main component analysis, we should talk about and focus on our problem. The challenge is distinguish and classify face in input picture on face recognition area. As for face recognition, it needs an existing faces database. If there is a new picture of people face, system need to report person's name. This is the different than the face detection because the last one need to determin if there is a face picture in the input picture. There is two way of realizing this. First one is to flatten new picture it into a vector and compute this picture between the Euclidean distance and all the other pictures in the database. But this approach has several objections.

We can imagine that we built a big face recognition system and have large faces databases. Do the comparison for every face will need to take a while. As for real-time face recognition, our dataset larger, the slower our algorithm. But the more face pictures will also produce a good result if we choose the neural network. There is a big issue with using neural network: pictures will be large. If there is an M*N picture, the process has to flat it out into an M*N*N vector into our input neural network. It will hurt size because it is a large picture. On the other hand, Euclidean distance is high-dimensional and will give us high misclassifications! So we will keep high-dimensional face pictures and boil them down to smaller dimensional and retaining the key parts of pictures.

Another way is to use a dimensionality reduction technique. We take the face pictures of higher-dimensional data and to represent it to lower-dimensional data. The example is the following picture.

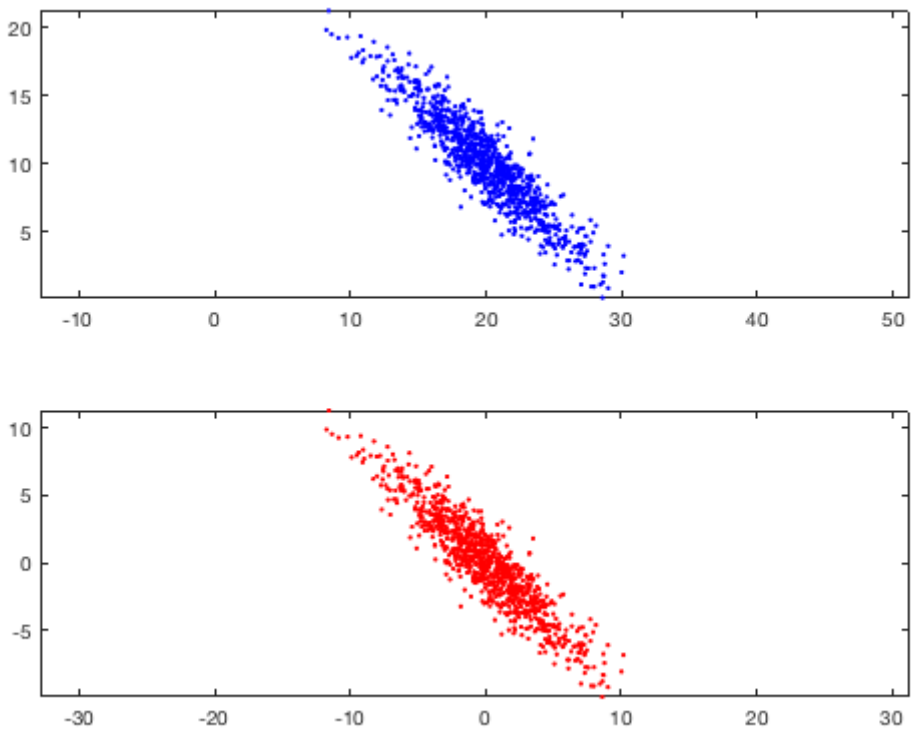

Figure 1: Dimentionality Reduction [1]

These plot pictures show the different bottom chart but the same data. The dimensionality reduction will reduce data from 2D to 1D pictures.

As for the component analysis, we choose the principal component analysis (PCA). The result is we want to

use the hyperplane could make all points onto it. In the above example, the potential axis is the x-axis or y-axis. However, that's not the best axis and we will use the axis where the data would be the most spread to cuts through our data diagonally.

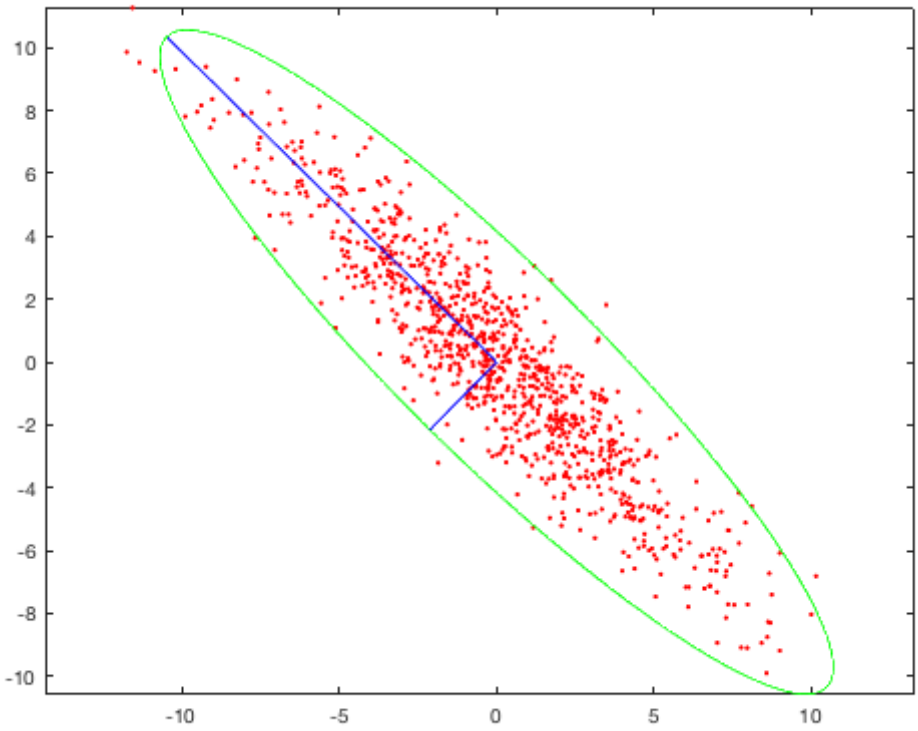

Figure 2: Orthogonal axis of PCA [1]

The blue axis is the correct axis and they would be maximally spread if we project our points onto this axis. And how do we figure out this axis? In fact, we can comput our data covariance matris and consider covariance matrix largest eigenvectors. These is our principal axes. Use this approach, we can conceptualize our M*N pictures as points in M*N*N dimensional space. It will improve and speed up our computations and make robust to variation and noise.

Let is use code a face recognition using scikit-learn and we use the LFW dataset.

1.

```
# ##############################################################################
# Download the data, if not already on disk and load it as numpy arrays

lfw_people = fetch_lfw_people(min_faces_per_person=70, resize=0.4)

# introspect the images arrays to find the shapes (for plotting)
n_samples, h, w = lfw_people.images.shape

# for machine learning we use the 2 data directly (as relative pixel
# positions info is ignored by this model)
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)


# ##############################################################################
# Split into a training set and a test set using a stratified k fold

# split into a training and testing set
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42)
```

The Scikit-learn provides two loaders which could automatically download, parse and store data files. Identify Challenges:Now we meet the process to input data into a single call for splitting (and optionally subsampling) data in a oneliner. On the other hand, as for the number of face about the dataset,we use a limited word to control the full number of the name.

In this dataset, we have 5749 classes, 13233 samples pictures total, 5828 dimensionality and the fractures between 0 to 255. These lead to the first challenge: recognize faces by people photos.The each famous people picture is centred on a single face. This process is called the Face Verification process: the process choose a pair of two pictures and binary classifier have to distinguish the two images if from the same person.

As for the result, we use the label to predict is the id of the person. Target label links the LFK datasets, and to distinguish samples and features classes. The second challenge is to plot the person pictures which find the shapes by introspecting the images arrays. So we entered the code by using shape classes to control the size of the people images.Next step is to split the matrices, arrays and classes into the test subsets and the random train by use the "ShuffleSplit" classes. To input data into splitting and optionally subsampling data by into a single call in a oneliner.

2.

```
# ##############################################################################
# Compute a PCA (eigenfaces) on the face dataset (treated as unlabeled
# dataset): unsupervised feature extraction / dimensionality reduction
n_components = 150

print("Extracting the top %d eigenfaces from %d faces"
      % (n_components, X_train.shape[0]))
t0 = time()
pca = PCA(n_components=n_components, svd_solver='randomized',
          whiten=True).fit(X_train)
print("done in %0.3fs" % (time() - t0))

eigenfaces = pca.components_.reshape((n_components, h, w))

print("Projecting the input data on the eigenfaces orthonormal basis")
t0 = time()
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
print("done in %0.3fs" % (time() - t0))


# ##############################################################################
# Train a SVM classification model

print("Fitting the classifier to the training set")
t0 = time()
param_grid = {'C': [1e3, 5e3, 1e4, 5e4, 1e5],
              'gamma': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.1], }
clf = GridSearchCV(SVC(kernel='rbf', class_weight='balanced'),
                   param_grid, cv=5, iid=False)
clf = clf.fit(X_train_pca, y_train)
print("done in %0.3fs" % (time() - t0))
print("Best estimator found by grid search:")
print(clf.best_estimator_)
```

As for the Face Recognition or Identification is to given and search an unknown person face picture, identify and distinguish the person name by referring to gallery of previously for seen pictures of distinguished persons.

The two advantages of SVM(support vector machines) is it effective in all cases where the number of the dimensions is more than the samples number and could use a subset of training data points in the support vectors. In face recognition area, eigenfaces mean the set of eigenvectors used in kinds of vision problem. Because each image is treated as one vector in the face recognition area, eigenfaces are derived from the covariance matrix of distribution over the high face images vector space.

3.

```
# ##############################################################################
# Quantitative evaluation of the model quality on the test set

print("Predicting people's names on the test set")
t0 = time()
y_pred = clf.predict(X_test_pca)
print("done in %0.3fs" % (time() - t0))

print(classification_report(y_test, y_pred, target_names=target_names))
print(confusion_matrix(y_test, y_pred, labels=range(n_classes)))
```

All the facea have already been stored to a normal size. This is a very important process for facial recognition and needs large collection of training data. The labal and print classed code is a way to control the target size and display their names and details. Finally, we can make a prediction and use a function to print out an entire report of quality for each class.

4.

```
# ##############################################################################
# Qualitative evaluation of the predictions using matplotlib

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=.01, right=.99, top=.90, hspace=.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(())


# plot the result of the prediction on a portion of the test set

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)

# plot the gallery of the most significative eigenfaces

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

plt.show()
```

This code classed maske a list to display the result of faces recognition.

3. Methodology

While the code is pretty self explanatory, a brief explanation is provided here, as well as supporting text blocks per code block where further information is required. The algorithm strongly references the scikit package for eigenface recognition, and hence follows a similar logical structure. The algorithm takes in an input of 1560 images of 12 individual peoples faces, determined by the min_faces_per_person parameter, which is set to 50, placing a minimum threshold of 50 images per person for consideration from the whole LFW set. These images are then sized down as PCA would greatly suffer from performance with larger images and hence each image is sized down to 40%. The algorithm proceeds to split the 1560 images into training and testing set with a 25% ratio towards the testing set. The algorithm then extracts the top 75 eigenfaces from the training split data and

performs pca on it, reducing its dimentionality. The algorithm then proceeds to train an SVM classifier, using a parameter grid to tune the values of *C* and *gamma* parameters, which are responsible for controlling the function margin and the degree of influence from a single training example respectively. A gridsearch is then applied to select the greatest fitting result which is then finally used to predict the names from the target set. The algorithm is relatively simple and takes on average less than a minute to execute.

###Setup

In [3]:

```
from time import time
import logging
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV

from sklearn.datasets import fetch_lfw_people

from sklearn.decomposition import PCA
from sklearn.svm import SVC

from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

# ###############################################################################
#
# Display progress logs on stdout. Basic setups and import.
#

print(__doc__)
logging.basicConfig(level=logging.INFO, format='%(asctime)s %(message)s')
```

```
Automatically created module for IPython interactive environment
```

Initialization

A min face threshold of 50 is applied and each image in the array is resized to 40% of its original size, to save time during PCA. Testing and training sets are also declared and initialized with a 0.75:0.25 training to testing set split.

In [0]:

```
# ##############################################################################
#
# Downloads the LFW data, if not already on disk and loads it as numpy arrays.
# The min faces threshold is updated to 50 and each image is resized to a
# smaller size to improve efficiency of running, since PCA doesn't change alot
# with higher res photos. We then set up all the information for printing in
# specific variables. After that, we set up all the test information variables,
# not to be confused with the test set, as we do that later.
#

lfw_people = fetch_lfw_people(min_faces_per_person = 50, resize = 0.4)
num_samples, h, w = lfw_people.images.shape
lfw_data = lfw_people.data
num_features = lfw_data.shape[1]

lfw_target = lfw_people.target
target_names = lfw_people.target_names
num_classes = target_names.shape[0]

# ##############################################################################
#
# Printing all the model metadata onto the console
#

print("Total dataset size:")
print("num_samples: %d" % num_samples)
print("num_features: %d" % num_features)
print("num_classes: %d" % num_classes)


# ##############################################################################
#
# Instantiate new variables for testing and training sets. X_train and Y_train
# are the training sets and X_test and y_test are the testing sets. The sets
# are split to a 0.75:0.25 training to testing set ratio, picked randomly.
#

X_train, X_test, y_train, y_test = train_test_split(
    lfw_data, lfw_target, test_size=0.25, random_state=42)
```

```
Downloading LFW metadata: https://ndownloader.figshare.com/files/5976012 (https://nd
ownloader.figshare.com/files/5976012)
2019-09-25 02:45:02,494 Downloading LFW metadata: https://ndownloader.figshare.com/f
iles/5976012 (https://ndownloader.figshare.com/files/5976012)
Downloading LFW metadata: https://ndownloader.figshare.com/files/5976009 (https://nd
ownloader.figshare.com/files/5976009)
2019-09-25 02:45:02,784 Downloading LFW metadata: https://ndownloader.figshare.com/f
iles/5976009 (https://ndownloader.figshare.com/files/5976009)
Downloading LFW metadata: https://ndownloader.figshare.com/files/5976006 (https://nd
ownloader.figshare.com/files/5976006)
2019-09-25 02:45:03,012 Downloading LFW metadata: https://ndownloader.figshare.com/f
iles/5976006 (https://ndownloader.figshare.com/files/5976006)
Downloading LFW data (~200MB): https://ndownloader.figshare.com/files/5976015 (http
s://ndownloader.figshare.com/files/5976015)
2019-09-25 02:45:03,277 Downloading LFW data (~200MB): https://ndownloader.figshare.
com/files/5976015 (https://ndownloader.figshare.com/files/5976015)
```

```
Total dataset size:
num_samples: 1560
num_features: 1850
num_classes: 12
```

Principal Component Analysis

Computes a PCA on the LFW dataset by extracting the data into a 75 dimentional space. Additionaly, the data is whitened, which is a linear transformation done to improve the results of the prediction model.

In [0]:

```
# ###############################################################################
#
# Compute a PCA (eigenfaces) on the face dataset for dimensionality reduction
# and print a visual representation of the 75 eigenfaces.
#

num_components = 75
print("Extracting the top %d eigenfaces from %d faces."
      % (num_components, X_train.shape[0]))
t0 = time()

pca = PCA(n_components = num_components, svd_solver = 'randomized',
          whiten = True).fit(X_train)
print("done in %0.3fs" % (time() - t0))

eigenfaces = pca.components_.reshape((num_components, h, w))

print("\nProjecting the input data on the eigenfaces orthonormal basis",
      "(dimentional \nreduction on the X_train and X_test sets.)")
t0 = time()
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
print("done in %0.3fs" % (time() - t0))

print("\nVisual Representation of the chosen eigenfaces:")
fig = plt.figure(figsize=(10, 6))
for i in range(75):
    ax = fig.add_subplot(5, 15, i + 1, xticks=[], yticks=[])
    ax.imshow(pca.components_[i].reshape(lfw_people.images[0].shape),
              cmap=plt.cm.bone)
```

```
Extracting the top 75 eigenfaces from 1170 faces.
done in 0.236s

Projecting the input data on the eigenfaces orthonormal basis (dimentional
reduction on the X_train and X_test sets.)
done in 0.021s

Visual Representation of the chosen eigenfaces:
```

SVM Classification

The SVM classifier is trained in this code block and a parameter grid is set up in order to tweak the *C* and *gamma* parameters of the SVM classifier. A gridsearch is used to pick out the best result (i.e. the highest grid score.).

In [0]:

```
# ##############################################################################
#
# Train a SVM classification model. Training previous dimentionally reduced
# data from pca with the target variable y_train. Param_grid sets up all the
# various parameter settings to consider for the SVM classifier. It then
# utilizes gridsearch for parameter optimization, and selects the most fitting.
#

print("Fitting the SVM classifier to the training set.")
t0 = time()
param_grid = {'C': [1e3, 5e3, 1e4, 5e4, 1e5],
              'gamma': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.1], }
clf = GridSearchCV(SVC(kernel='rbf', class_weight='balanced'),
                   param_grid, cv=5, iid=False)

clf = clf.fit(X_train_pca, y_train)
print("done in %0.3fs" % (time() - t0))
print("\nBest estimator found by grid search:")
print(clf.best_estimator_)
```

```
Fitting the SVM classifier to the training set.
done in 36.138s

Best estimator found by grid search:
SVC(C=1000.0, cache_size=200, class_weight='balanced', coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.01, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

Report and Evaluation

In [0]:

```
# ###############################################################################
#
# Generate and print report and evaluation of the model on the pca reduced
# test set.
#

print("Predicting people's names on the test set.")
t0 = time()
y_pred = clf.predict(X_test_pca)
print("done in %0.3fs" % (time() - t0))

print(classification_report(y_test, y_pred, target_names = target_names))
print("Confusion Matrix: \n",
      confusion_matrix(y_test, y_pred, labels = range(num_classes)))


# ###############################################################################
#
# Further report evaluation using matplotlib for data visualisation. Plots the
# a portion of the test set for qualitative review.
#

def plot_gallery(images, titles, h, w, n_row=4, n_col=4):
    """Helper function to plot a gallery of portraits with relevant
    information"""
    plt.figure(figsize=(1.8 * n_col, 2.4 * n_row))
    plt.subplots_adjust(bottom=0, left=.01, right=.99, top=.90, hspace=.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):
    """Helper function to grab all the proper names to match with the images"""
    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()
```

```
Predicting people's names on the test set.
done in 0.068s
                   precision    recall  f1-score   support

     Ariel Sharon       0.71      0.75      0.73        16
     Colin Powell       0.82      0.91      0.86        66
  Donald Rumsfeld       0.71      0.74      0.73        27
    George W Bush       0.88      0.92      0.90       140
Gerhard Schroeder       0.72      0.75      0.73        24
```

Hugo Chavez	1.00	0.65	0.79	17
Jacques Chirac	0.75	0.60	0.67	10
Jean Chretien	0.85	0.79	0.81	14
John Ashcroft	0.88	0.88	0.88	16
Junichiro Koizumi	1.00	0.73	0.84	11
Serena Williams	0.86	0.67	0.75	9
Tony Blair	0.89	0.85	0.87	40
accuracy			0.84	390

4. Evaluation

Report and Results

Our model produced the following result, as printed in the results section:

	precision	recall	f1-score	support
Ariel Sharon	0.92	0.69	0.79	16
Colin Powell	0.74	0.94	0.83	66
Donald Rumsfeld	0.83	0.74	0.78	27
George W Bush	0.83	0.94	0.88	140
Gerhard Schroeder	0.90	0.75	0.82	24
Hugo Chavez	1.00	0.47	0.64	17
Jacques Chirac	1.00	0.40	0.57	10
Jean Chretien	0.91	0.71	0.80	14
John Ashcroft	1.00	0.81	0.90	16
Junichiro Koizumi	1.00	0.82	0.90	11
Serena Williams	0.86	0.67	0.75	9
Tony Blair	0.92	0.90	0.91	40
accuracy			0.84	390
macro avg	0.91	0.74	0.80	390
weighted avg	0.86	0.84	0.84	390

As we can see, our model produced a weighted average f1-score of 84% and an equivalent recall of 84%. Our model specifically and significantly suffered with Jacques Chirac's predicton due to his face sharing a strong resemblence with Gerhard Schroeder as shown in the confusion matrix.

Testing and Efficiency Analysis

While most of the testing on *C* and *gamma* parameters for the SVM learner was implented in the grid search during training, as it is more dependant on each individual instance of learning, the number of principal components parameter for PCA was done seperately. This was tested for time efficiency and the value of 75 was picked as it produced the best results consistently, and as shown by the final plotted graph, increasing the number of components had a near linear effect on time, while arguably reducing model usefulness.

In [0]:

```
from sklearn import decomposition, svm, metrics
from sklearn.pipeline import Pipeline

n_comp = [75, 150, 175, 250, 600]
times = []

for i in range(len(n_comp)):
  clf = Pipeline([('pca', decomposition.PCA(n_components = n_comp[i],
                                            whiten=True)),
                  ('svc', svm.LinearSVC(C=1.0, max_iter=1e4))])
  t0 = time()

  clf.fit(X_train, y_train)
  print("The following are the metrics when num_components are: %s\n"
        %n_comp[i])
  y_pred = clf.predict(X_test)
  t1 = time() - t0
  print("Time Taken: %0.3fs" % (t1))
  times.append(t1)
  print(metrics.classification_report(y_test, y_pred))
  plt.show()

plt.plot(n_comp, times)
plt.ylabel('Time Taken (sec)')
plt.xlabel('num_components')
plt.show()
```

```
The following are the metrics when num_components are: 75

Time Taken: 2.326s
              precision    recall  f1-score   support

           0       0.77      0.62      0.69        16
           1       0.88      0.88      0.88        66
           2       0.60      0.56      0.58        27
           3       0.83      0.85      0.84       140
           4       0.54      0.54      0.54        24
           5       0.73      0.65      0.69        17
           6       0.60      0.60      0.60        10
           7       0.59      0.71      0.65        14
           8       0.87      0.81      0.84        16
           9       0.75      0.82      0.78        11
          10       0.56      1.00      0.72         9
          11       0.79      0.65      0.71        40

    accuracy                           0.77       390
```

While our models overall accuracy can be increased by implementing other learning methods (such as boosted learning), the SVM classifier's results are satisfactory as we are classifying 12 different faces. As seen in the succeeding code block, with a lower number of faces, such as 5, the algorithm performs the algorithm performs almost equivalently, but however, it is not representative of a real world scenario, and hence 12 faces were chosen as a good ratio between semi-real-time results (under a minute) and while increasing the number of classes from an absurdly low amount of 5.

In [8]:

```
from sklearn import decomposition, svm, metrics
from sklearn.pipeline import Pipeline
# ###############################################################################
#
# Downloads the LFW data, if not already on disk and loads it as numpy arrays.
# The min faces threshold is updated to 50 and each image is resized to a
# smaller size to improve efficiency of running, since PCA doesn't change alot
# with higher res photos. We then set up all the information for printing in
# specific variables. After that, we set up all the test information variables,
# not to be confused with the test set, as we do that later.
#

lfw_people = fetch_lfw_people(min_faces_per_person = 100, resize = 0.4)
num_samples, h, w = lfw_people.images.shape
lfw_data = lfw_people.data
num_features = lfw_data.shape[1]

lfw_target = lfw_people.target
target_names = lfw_people.target_names
num_classes = target_names.shape[0]

# ###############################################################################
#
# Printing all the model metadata onto the console
#

print("Total dataset size:")
print("num_samples: %d" % num_samples)
print("num_features: %d" % num_features)
print("num_classes: %d" % num_classes)

# ###############################################################################
#
# Instantiate new variables for testing and training sets. X_train and Y_train
# are the training sets and X_test and y_test are the testing sets. The sets
# are split to a 0.75:0.25 training to testing set ratio, picked randomly.
#

X_train, X_test, y_train, y_test = train_test_split(
    lfw_data, lfw_target, test_size=0.25, random_state=42)

clf = Pipeline([('pca', decomposition.PCA(n_components = 75, whiten=True)),
                ('svc', svm.LinearSVC(C=1.0, max_iter=1e4))])
t0 = time()

clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print("Time Taken: %0.3fs" % (time() - t0))
print(metrics.classification_report(y_test, y_pred))
plt.show()
```

```
Total dataset size:
num_samples: 1140
num_features: 1850
num_classes: 5
Time Taken: 1.207s
              precision    recall  f1-score   support
```

0	0.89	0.84	0.86	64
1	0.71	0.69	0.70	32
2	0.87	0.91	0.89	127
3	0.79	0.76	0.77	29
4	0.84	0.79	0.81	33
accuracy			0.84	285
macro avg	0.82	0.80	0.81	285
weighted avg	0.84	0.84	0.84	285

5. Conclusion

Now we discussed a famous approach to face recognition area which called eigenfaces. In the deeper, essence of the face eigenfaces called PCA and reduce the pictures dimensionality into small data. We have smaller face pictures representation.

The common face verification addresses usually handle the mainly large intra-class variations, for example, like the expression, illumination and pose. But in LFW databases, we can find main limiting which is even all the negative face pairs are quite easy to find and distinguish. It largely misread the restriction of the unconstrained face verification task. However, the LFW database collection is based on the random imposter assumption, so the main problem for this face verification is still easily with large examples of inter-class variance.

6. Ethical

There are a multitude of ethical issues involved with any facial recognition technology primarily surrounding privacy, most prominently; security and safety. Privacy is often a major concern for rising technologies, as often, data mining models will exhibit privacy concerns for ethical use or misuse of the involved data. For facial recognition specifically however, there is a greater need for attention towards implications, as they can negatively affect users of facial recognition due to the technology inherently involving in storing and identifcation of peoples faces. This raises a security concern as the data stored needs to be monitored and encrypted, as people may not want thier faces to be exhibited to various data breaches or hacks that would make the data public or manifested for misuse. Safety is also another concern that is raised with emerging social networking features attributing facial recognition, as any malicious person can upload a photo of a person, relying on the social networks algorithm to identify the said person. This can present a major safety concern for vurneable people and must be addressed by either a opt-in or opt-out model of approach.

Overall however, facial recognition brings about more good than evil, and hence an utilitarian approach is often taken, as it can equally help in identifying repeat offenders in shopping stores, wanted criminals roaming out in the public or in airports, authorized personal in restricted areas and can also help in identifying authorized owners of smartphones to unlock thier phones and etc. For these reasons of vastly increased efficiency in mundane and monotonous tasks, a utilitarian approach towards the ethical issues of facial recognition are taken due to its aid in the mentioned areas.

7. Video Pitch

See appendix

8. Appendix

Video Link:

Bibliography

[1] *M. Deshpande, "Face Recognition with Eigenfaces", Python Machine Learning, 2019. [Online]. Available: https://pythonmachinelearning.pro/face-recognition-with-eigenfaces/ (https://pythonmachinelearning.pro/face-recognition-with-eigenfaces/). [Accessed: 25- Sep- 2019].*