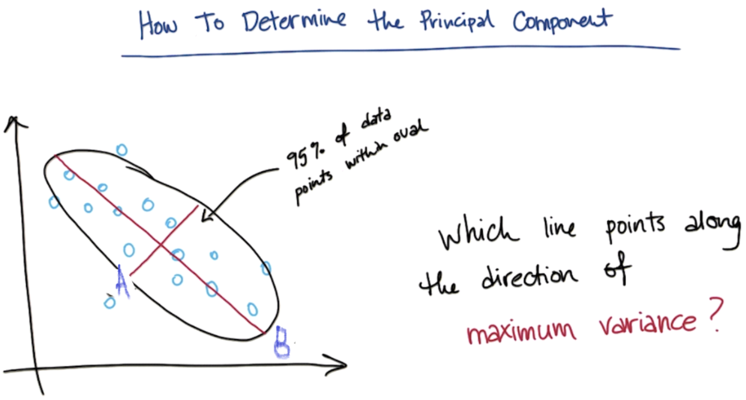
National University of Singapore
School of Computing
BT5151 Foundation in Data Analytics II
Tutorial 6:
PCA and GMM

Introduction to the Principal Component Analysis

- Systematized way to transform input features into Principal Component
- These Principal Components acts as new features
- Principal Components are directions in data that maximizes the variance and minimize the information loss, when you project or compress down to them
- More variance of data along Principal Component, higher the principal component is raked
- Maximum number of PC = Number of Input Features



When to use PCA

- To find latent features driving the patterns in Data
- Dimensional Reduction
  - To Visualize High Dimensional Data
  - To Reduce Noise
  - To make algorithms like Classification and Regression work better with reduced dimensionality

Application: Facial recognition using PCA + SVM.

1. Download the 'fetch_lfw_people' dataset from sklearn datasets using 'fetch_lfw_people(min_faces_per_person=70, resize=0.4)'. Introspect the parameters of dataset. Print the target_names parameters. Visualize a few images at random.

```
from sklearn.datasets import fetch_lfw_people
import numpy as np
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.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.decomposition import PCA
from sklearn.svm import SVC

# 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)
print(lfw_people.keys())
print(lfw_people.data.shape)
print(lfw_people.images.shape)
print(lfw_people.target.shape)
print(lfw_people.target_names)
plt.imshow(lfw_people.images[0])
```

2. Create your X variable (the features) and the y variable (the labels).

```
X = lfw_people.data
y = lfw_people.target
```

3. Create a train-test split in your data using the SKLearn Train-Test split library.

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

4. Compute a PCA on the face dataset with n_component=150. This will help in dimensionality reduction. Create new features after PCA for the train and test data.

```
n_components = 150
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)
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
```

5. Now using the new features fit the SVM classifier predict the targets. Try using GridSearchCV to tune your C and gamma parameters. Print the best_estimator_ of the GridSearchSV.

```
print("Fitting the classifier to the training set")
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_)
```

6. Create predictions on the test set using the best estimated fitted classifier and use the SKLearn Classification_report library to generate a classification report. Discuss your results.

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

7. Visualize your prediction by plotting few images and its corresponding actual target and predicted target.

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

n_samples, h, w = lfw_people.images.shape

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)

plt.show()
```

Colab Notebook Link:
https://colab.research.google.com/drive/1tiTkq620_QDDqZlISjgymsHULWuz_qUD

Introduction to the GMM

- A GMM attempts to model the data as a collection of Gaussian blobs.
- You can use it as unsupervised clustering algorithm which attempts to find distinct groups of data without reference to any labels.

Application: Segmentation of Image using GMM Clustering, i.e. giving each pixel a label

8. Generally, Image consists of 3 frames (Blue, Green, Red), with each pixel ranging from 0-255. Load a sample image from sklearn dataset with 'load_sample_image('china.jpg')'. Visualize the image using 'plt.imshow'. Print and Save your original image shape. Let the shape of image be (h, w, 3).

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_sample_image

X = load_sample_image('china.jpg')
plt.imshow(X)
h, w, _ = X.shape
print(X.shape)
```

9. Assign each pixel inside (h, w, 3) a label from 0-5 using sklearn 'GaussianMixture' clustering. To do that first flatten the image numpy array using '.reshape(-1, 3)'. Now fit the GaussianMixture with n_clusters=5, to assign labels to flattened array.

```
from sklearn.mixture import GaussianMixture
X = X.reshape(-1,3)
print("X shape = ", X.shape)
gmm = GaussianMixture(covariance_type='full', n_components=5)
gmm.fit(X)
labels = gmm.predict(X)
print("Clusters shape = ", labels.shape)
```

10. Reshape your predicted labels to (h, w) shape, i.e. size of original image. Now visualize your segmented image using 'plt.imshow(new_image, cmap='gray')'.

```
clusters = clusters.reshape(-1, w)
print("Clusters shape = ", clusters.shape)
plt.imshow(clusters, cmap='gray')
```

Colab Notebook link:
https://colab.research.google.com/drive/1BcsV6oCqzjxG6HxIENRiEOLzAM5Qvb5g