Data mining and applications

Practical work

N°3

Principle Component Analysis (PCA) is a dimension reduction technique that can find the combinations of variables that explain the most variance. In this post we will demonstrate dimensionality reduction concepts including **facial image compression and reconstruction** using PCA.

Part 1: Warming step with Iris dataset

Let's start by examining a simple dataset, the Iris data available by default in scikit-learn. The data consists of measurements of three different species of irises. There are three species of iris in the dataset:

- Iris Virginica

- Iris Setosa

- Iris Versicolor.

1. Load the iris dataset and check shape of data and list of features (X matrix)
2. By using sickit-learn, import and instantiate PCA with 2 components, then fit PCA to the iris dataset and transform it into 2 principal components.
3. Plot the projected principal components and try to understand the data.
4. **pca.components_** has the meaning of each principal component, essentially how it was derived. Checking shape tells us it has 2 rows, one for each principal component and 4 columns, proportion of each of the 4 features #for each row.

```
print pca.components_
print pca.components_.shape
```

Try to decipher the meaning of the principal components.

```
print "Meaning of the 2 components:"
for component in pca.components_:
      print " + ".join("%.2f x %s" % (value, name)
            for value, name in zip(component, iris.feature_names))
```

5. How many components do we need if we only need a variance of 92% ? And how many for a variance of more than 97% ?

Part 2: Facial images

Now onto a bigger challenge, let's try and compress a facial image dataset using PCA. Going to use the **Olivetti face image dataset**, again available in scikit-learn. We'd like to reduce the original dataset using PCA, essentially compressing the images and see how the compressed images turn out by visualizing them.

Before using PCA, let us try and understand as well as display the original images. Note the Olivetti faces data is available in scikit-learn but not locally. It needs to be downloaded.

```
from sklearn.datasets import fetch_olivetti_faces

oliv=fetch_olivetti_faces()

print oliv.keys()

print oliv.data.shape #tells us there are 400 images that are 64 x 64
(4096) pixels each
```

```
fig = plt.figure(figsize=(6,6))

fig.subplots_adjust(left=0, right=1, bottom=0, top=1, hspace=0.05,
wspace=0.05)

# plot the faces, each image is 64 by 64 pixels

for i in range(64):

      ax = fig.add_subplot(8, 8, i+1, xticks=[], yticks=[])

      ax.imshow(oliv.images[i], cmap=plt.cm.bone, interpolation='nearest')

plt.show()
```

1. How much of the variance is retained if we compressed these down to a 8x8 (64) pixel images ?
2. Try to visualize the top 10 principal components. This is NOT a reconstruction of the original data, just visualizing the principal components as images. The principal components are vectors of the length = to the number of # features 4096. We'll need to reshape it to a 64 x 64 matrix.

Let's now try to reconstruct the images using the new reduced dataset. In other words, we transformed the 64x64 pixel images into 8x8 images. Now to visualize how these images look we need to inverse transform the 8x8 images back to 64x64 dimension.

Note that we're not reverting back to the original data, we're simply going back to the #actual dimension of the original images so we can visualize them.

```
X_inv_proj = pca_oliv.inverse_transform(X_proj)
#reshaping as 400 images of 64x64 dimension
X_proj_img = np.reshape(X_inv_proj,(400,64,64))
```

3. Setup a figure 8 inches by 8 inches.
4. Plot the faces so that each image is 64 by 64 dimension but 8x8 pixels. How do you find the result ? Not bad at all in my opinion ☺