Intelligent Systems 1 - Summer Semester 2015

Exercise Sheet 10, Principle Component Analysis
submission date: 2015-07-16 (Beginning of next lecture)

2015-07-09

1 Exercise 1 (50 points)

Consider the following three-dimensional datapoints:

(1.3,1.6,2.8), (4.3,-.4,5.8), (-0.6,3.7,0.7), (-0.4,.32,5.8),(3.3,-0.4,4.3), (-0.4,3.1,0.9)

Perform principal component analysis by:

- Calculating the mean, $\mathbf{c}$, of the data.
- Calculating the covariance matrix $\Sigma = \frac{1}{6}\sum_{n=1}^{6} \mathbf{x}(\mathbf{x})^T - \mathbf{c}\mathbf{c}^T$.
- Finding the eigenvalues and eigenvectors $\mathbf{e}_i$ of the covariance matrix.
- You should find that only two eigenvalues are large, and therefore that the data can be well represented using two components only. Let $\mathbf{e}_1$ and $\mathbf{e}_2$ be the two eigenvectors with largest eigenvalues. Calculate the two-dimensional representation of each datapoint $(\mathbf{e}_1^T(\mathbf{x}^n - \mathbf{c}), \mathbf{e}_2^T(\mathbf{x}^n - \mathbf{c})), n = 1, \ldots, 6$
- Calculate the reconstruction of each datapoint $\mathbf{c} + (\mathbf{e}_1^T(\mathbf{x}^n - \mathbf{c}))\mathbf{e}_1 + (\mathbf{e}_2^T(\mathbf{x}^n - \mathbf{c}))\mathbf{e}_2, n = 1, \ldots, 6$

2 Exercise 2(50 points)

In this exercise you will use PCA to perform a classification task. The data you will be using is a collection of face images and your task is to implement a classifer that can assign the correct name to each photo. You will be using the Scikit Learn library to find the right parameterization of the model.

```
>>> %matplotlib inline
...
... from __future__ import print_function
...
... from time import time
... import logging
... import matplotlib.pyplot as plt
... import numpy as np
... from sklearn.cross_validation import train_test_split
... from sklearn.datasets import fetch_lfw_people
... from sklearn.grid_search import GridSearchCV
... from sklearn.metrics import classification_report
... from sklearn.metrics import confusion_matrix
... from sklearn.svm import SVC
... from numpy import linalg
... import numpy as np
```

2.1 Loading the data

Since you will be using one of Scikit Learn datasets, the library can download data for you. Assuming you have internet access, running the following code will load the data into variables you can use later.

```
>>> logging.basicConfig(level=logging.INFO, format='%(asctime)s %(message)s')
...
...
... ################################################################################
... # 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 image arrays to find the shapes (for plotting)
... n_samples, h, w = lfw_people.images.shape
...
... # for machine learning we use the 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)
```

```
Total dataset size:
n_samples: 1288
n_features: 1850
n_classes: 7
```

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

The next step is to split the data into trainning and test data. This will be useful for evaluation of the model you train later.

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

2.3 Compute PCA

The first part of the assignment is implementing the PCA algorithm. Implement this algorithm in the `PCAtrain` function below. The function should return four values: Principle vectors and singular values, mean and standard deviation of input features. PCA should consume the normalized features, i.e. mean 0 and standard deviation 1 for every input feature. You can do this by subtracting the mean and dividing by standard deviation across every feature column. Make sure you will use the same statistic for both training and test set.

```
>>> def PCAtrain(Xtrain):
...   #### IMPLEMENT HERE
...   return pc_vectors, singular_values, mean, std
```

Having implemented PCA you can now use it to find principle components. In the next cell implement a code that plots the cumulative distribution of singular values and find how many principle components you should use to cover 95% of the variation in input. You should call the `PCAtrain` function implemented earlier.

```
>>> ### IMPLEMENT HERE
... print("Number of components covering 0.95 of variance: %d" % 0)
```

2.4 Classification

In this step you should implement a classifier that can classify input data. You should use the transformed data from the previous step to complete this part. You should use `GridSearchCV` to find the right parameters of the model you are using. Consider using one of SVM or any other model that can perform better. The model should be a scikit learn classifier and set to a variable named `clf`. You may need to read the documentation of Scikit-learn to find out how the grid search algorithm works.

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

2.5 Results

If everything has been implemented correctly you should get classification report, confusion matrix and example outputs as well as eigen faces below. We could implement a model that performs above 80% on test set, can you do better?

```
>>> 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)))
```

```
>>> 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)
```