**8Experiment 5: Dimensionality Reduction through PCA**

**Objective** :To apply dimensionality reduction tasks (PCA) using Python.

**Time Required** : 3 hrs

**Programming Language** : Python

**Software Required** : Anaconda

**Introduction**

Data preprocessing is crucial in any data mining process as they directly impact success rate of the project. This reduces complexity of the data under analysis as data in real world is unclean. Data is said to be unclean if it is missing attribute, attribute values, contain noise or outliers and duplicate or wrong data. Presence of any of these will degrade quality of the results. Furthermore, data sparsity increases as the dimensionality increases which makes operations like clustering, outlier detection less meaningful as they greatly depend on density and distance between points. Purpose of dimensionality reduction is to:

* Avoid curse of dimensionality
* Reduces time required by algorithms
* Greatly reduces memory consumption
* Ease of visualization of data
* Eliminate irrelevant features

Principal Component Analysis (PCA) is a method used to reduce number of variables in your data by extracting important one from a large pool. It reduces the dimension of your data with the aim of retaining as much information as possible. In other words, this method combines highly correlated variables together to form a smaller number of an artificial set of variables which is called “principal components” that account for most variance in the data.

**TASK:**

Apply PCA on the Fisher’s Iris data set. The data contains 3 classes of 50 instances each, where each class refers to a type of iris plant. There are 4 different attributes describing the data. You will use principal component analysis to transform the data to a lower dimensional space.

***Steps to follow:***

1. Download the Iris data set from the following webpage: <http://archive.ics.uci.edu/ml/datasets/Iris>
2. Load all relevant packages and dataset.
3. Split feature vectors and labels.
4. Normalize the dataset which is done by subtracting the mean of each feature vector from the dataset so that the dataset should be centered on the origin.
5. Compute the covariance matrix which is basically a measure of the extent to which corresponding elements from two sets of ordered data move in the same direction.

* To compute the covariance matrix, use the np.cov() builtin method

1. Calculate the eigenvalues and eigenvectors.

**Remember:** The Eigenvectors of the Covariance matrix we get are Orthogonal to each other and each vector represents a principal axis. A higher Eigenvalue corresponds to a higher variability. Hence the principal axis with the higher Eigenvalue will be an axis capturing higher variability in the data. Orthogonal means the vectors are mutually perpendicular to each other.

* You can use the builtin method np.linalg.eigh(). It will return two objects, a 1-D array containing the eigenvalues, and a 2-D square array or matrix (depending on the input type) of the corresponding eigenvectors (in columns).

1. Sort the eigen values in descending order.

**Remember:** We order the eigenvalues from largest to smallest so that it gives us the components in order of significance. Each column in the Eigen vector-matrix corresponds to a principal component, so arranging them in descending order of their Eigenvalue will automatically arrange the principal component in descending order of their variability. Hence, the first column in our rearranged Eigen vector-matrix here will be a principal component that captures the highest variability.

* You can use the builtin method np.argsort()

1. Choose components and form a feature vector.

**Remember:** If we have a dataset with n variables, then we have the corresponding n eigenvalues and eigenvectors. To reduce the dimensions, we choose the first p eigenvalues and ignore the rest. Some information is lost in the process, but if the eigenvalues are small, we do not lose much.

In this task, select the first two principal components. n\_components = 2 means your final data should be reduced to just 2 dimensions.

1. Transform the data by having a dot product between the Transpose of the Feature Vector and the Transpose of the mean-centered data. By transposing the outcome of the dot product, the result we get is the data reduced to lower dimensions (2-D) from higher dimensions (4-D).

* You can use the following command for this purpose:

X\_reduced=np z 3.dot(eigenvector\_subset.transpose(), X\_meaned.transpose()).transpose()

1. Project the data onto its first two principal components and plot the results using the seaborn and matplotlib libraries. (**Hint:** Create Data Frame of reduced dataset and concatenate it with Labels (target variable) to create a complete Dataset).