Technische Universität München
Fakultät für Informatik
Prof. N. Navab, Ph.D.

WS 2016
Exercise Sheet 2
October 25, 2016

Practical Course: Machine Learning in Medical Imaging

PCA and K-Means

For your submission, each exercise should be accompanied by a single Matlab/Python file that runs your implementation to produce the results – such as tables or plots. You can load initial data sets from `mat` files in your submission. Please submit the results to Shadi Albarqouni, shadi.albarqouni@tum.de. Make sure you include the names of all group members.

1 Principle Component Analysis (PCA)

1.1 Implement PCA

In this exercise, you should implement the PCA algorithm in two different ways, i.e. using Singular Value Decomposition (SVD) and the eigendecomposition of the Covariance matrix as explained in our lecture.

1.1.1 Singular Value Decomposition (SVD)

Given a data matrix $X \in \mathbb{R}^{N\times d}$, where N is the number of samples (observations) and d is the feature dimension, the singular value decomposition (SVD) can be computed as follows:

$$X = U\Sigma V^T, \tag{1}$$

where $U \in \mathbb{R}^{N\times N}$ is the left-singular vectors, the diagonal elements of $\Sigma \in \mathbb{R}^{N\times d}$ are the singular values, and $V \in \mathbb{R}^{d\times d}$ is the right-singular vector. The eigenvectors are the same as the right-singular vector, where the eigenvalues are the diagonal elements of $\Sigma^T\Sigma$.

1.1.2 Eigen-decomposition of Covariance Matrix

Given a covariance matrix $C \in \mathbb{R}^{d\times d}$, which can be computed from the data matrix, i.e. $C = X^T X$, the eigenvectors and eigenvalues can be computed as follows:

$$CV = \Lambda V, \tag{2}$$

where $V \in \mathbb{R}^{d\times d}$ is the eigenvectors matrix and the diagonal elements of $\Lambda \in \mathbb{R}^{d\times d}$ represent the eigenvalues.

It should be noted that data matrix X has zero mean. The projected data can be obtained by $X_{proj} = XV$.

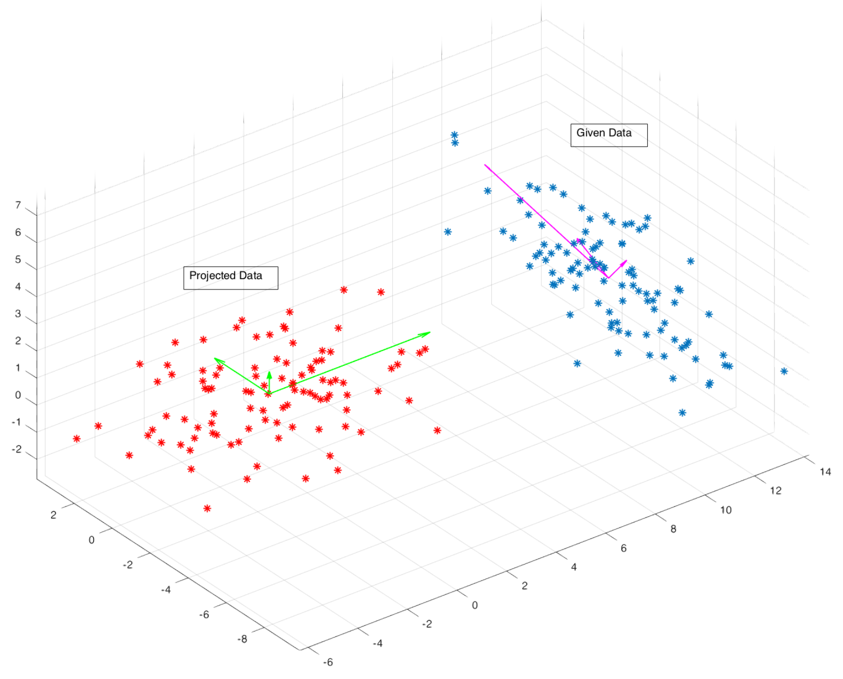

Figure 1: Use the following figure to compare your algorithm

In the exercise folder, there is a class called `myPCA.m, myPCA.py`, where you need to write your functions `usingSVD(arg)` and `usingCOV(arg)`. The input arguments are your data matrix (`dataMatrix`) and the desired variance percentage (`desiredVariancePercentage`), while the output should be in a structure format (i.e. `objSVD`) and returns the following: `eigvecs, eigvals, meanDataMatrix, demeanedDataMatrix, projectedData`.

Sanity Check. In the exercise folder, there is a small toy dataset `toydata.mat` with a data matrix `D` on which you can try your PCA implementation. The data is a point cloud with 100 samples, sampled from a normal distribution which was non-uniformly scaled, rotated and translated in space.

Use your PCA implementation to find the principal components of the data and the variance along each PCA direction. Plot the original data cloud like in Figure 1, with the principal components coordinate-system centered at the cloud mean, and stretched with their respective eigenvalue in order to show the amount of variance along each principal axis. Project and plot the de-meaned data using the principal components (see lecture slide) and observe that the point cloud is now at the Cartesian origin, with principal components re-oriented along the regular $x - y - z$ axes.

Once you obtain a comparable result (note that the direction of the axes may flip, depending on your implementation), you can proceed with the following exercises.

For submission, please create a small script `scriptToyData.m, scriptToyData.py`, included in the folder. It should load the data, compute the PCA and create the plot as described above.

1.2 Heart Disease

The dataset `filtHeartDataSet.mat` is a filtered version and subset of `Heart` dataset [1] which contains a binary outcome `labels` for 299 patients who presented with chest pain. An outcome value of `Yes` indicates the presence of heart disease based on an angiographic test, while `No` means no heart disease. The data matrix `dataMatrix` contains 13 features (measurements) including Age, Sex, Chol (a cholesterol measurement), and other heart and lung function measurements.

Tasks

1. Use your class `myPCA.m, myPCA.py` to find the eigenvectors and the corresponding eigenvalues that preserve at least 98% of the variance.
2. Project your data to the first three principle components (eigenvectors) and visualize the point cloud based on their corresponding labels, i.e. red colors for heart disease, blue colors for normal patients.
3. (**Bonus**) Can you tell which features are really important? [2].

2 K-Means Clustering

2.1 Implement K-Means

In this exercise, you should implement the K-Means algorithm as explained in the lecture.

2.1.1 K-Means algorithm

Given a data matrix X and the number of clusters K, the clusters, i.e. $\nabla = \{C_1, C_2, ..., C_K\}$, can be obtained by:

1. **Initialize:** Pick K random samples from the dataset X as the cluster centroids $\mu_k = \{\mu_1, \mu_2, ..., \mu_K\}$.
2. **Assign Points to the clusters**: Partition the point cloud X into K clusters $\nabla = \{C_1, C_2, ..., C_K\}$ based on the Euclidean distance between the points and centroids (searching for the closest centroid).
3. **Centroid update**: Based on the points assigned to each cluster, a new centroid is computed μ_k.
4. **Repeat**: Do step 2 and 3 until convergence.
5. **Convergence**: if the cluster centroids barley change, or we have compact and/or isolated clusters. Mathematically, when the cost (distortion) function $\mathcal{L}(\nabla) = \sum_{k=1}^{K} \sum_{i \in C_k} \|x_i - \mu_k\|^2 \leq Tol$.

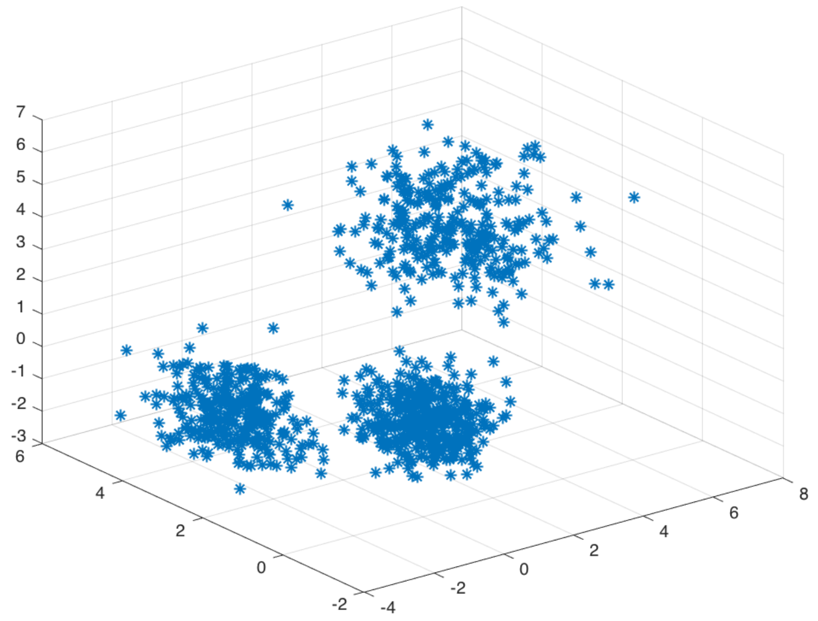

Figure 2: Click once to see the animation.

In the Exercise folder, there is a class called `myKmeans.m`, `myKmeans.py`, where you need to write your helping functions `assignPoints(arg)`, `updateCentroids(arg)`, and `computeCost(arg)`. Then, write `runKmeans` function where its input arguments are your data matrix (`dataMatrix`), the number of clusters `numberOfCluster`, and tolerance `Tol`. The output should be in a structure format (i.e. `objKmeans`) and returns the following: `assignedPoints, currentCentroids, cost`.

Sine the initialization in K-means is considered a limitation, run the K-means from many random initialization, and then take the minimum cost. Modify `runKmeans` to accept one more input argument, i.e the number of runs `numberOfRuns`. Further, write another function `runKmeansVis` to visualize every single update of the centroids and the assigned points similar to Fig. 2.

Sanity Check. In the exercise folder, there is a small toy dataset `toydata2.mat` with a data matrix `D` on which you can try your K-means implementation. The data is a point cloud with 1000 samples, sampled from three normal distribution with different variances, then translated and rotated in the space.

References

[1] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

[2] Pabitra Mitra, CA Murthy, and Sankar K Pal. Unsupervised feature selection using feature similarity. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(3):301–312, 2002.