# Lecture 20

Nov 12th 2008

# Clustering with Mixture of Gaussians

- Underlying model for data: assumes a generative process:
  - Each cluster follows a Gaussian distribution
  - First choose which cluster it belongs to based on the prior distribution of the clusters $P(c_i)$ for i=1,…, K
  - Then generate data based on the Gaussian distribution of the selected cluster, and so on …
- This is the same generative model that we have assumed in the Bayes classifier
  - The difference is that we can only observe x, and the cluster (class) label is missing
  - This is often referred to as the hidden variable problem in learning
  - Expectation Maximization (EM) is an general approach for solving such hidden variable inference problem

# Clustering using the EM algorithm

- Initialize the parameters
  - This can be done in two ways:
    - Randomly assign data points to clusters, and estimate the parameters based on these randomly formed clusters (more commonly used)
    - Randomly initialize the parameters
- Iterate between E-step and M-step:

  **E-step:** computes for each data point its probability of being assigned to each cluster

  **M-step:** estimate the parameters for each cluster
    - $\alpha_i$ - the cluster prior for cluster i
    - $\mu_i$ – the cluster i mean
    - $\Sigma_i$ – the covariance matrix for cluster i: in practice often restricted to simple forms such as diagonal matrices to reduce model complexity (less parameters to estimate, less chance of over-fitting)

This is highly similar to the iterative procedure of Kmeans, we know K-means optimizes the sum-squared-error criterion (what is this?), what does this procedure optimize?

# EM with GMM optimizes the likelihood

Let's see what is the likelihood function here:

$$L(D) = \log P(D) = \log P(x_1, x_2, \ldots, x_n) = \Sigma_i \log P(x_i)$$

- Note that the generative model assumes:

$$P(x_i) = \Sigma_k P(x_i | c_k) P(c_k)$$

- The likelihood of data is

$$L(D) = \Sigma_i \log \Sigma_k P(x_i | c_k) P(c_k)$$

- Similar to kmeans, the EM steps always improve this likelihood, and converge to a local optimum

# Comparing to Kmeans

- Assignment step
  - GMM: soft assignment
  - Kmeans: hard assignment
- Parameter estimation step
  - GMM: estimate mean and covariance matrix
  - Kmeans: estimate mean only, can be viewed as using identity matrix as the covariance matrix
- Which one converges faster
  - kmeans
- Which one is more flexible: GMM can capture clusters of more flexible shapes

# Selecting K

- For GMM, increasing K always leads to improved likelihood, thus we cannot choose k to optimize the data likelihood
- Heuristic approaches for choosing k based on likelihood
  - adding a complexity term to penalize model complexity
- One can also use cross-validated likelihood for choosing K
  - Hold out a validation data set
  - Estimate the parameters of the Gaussian mixtures using the "training set"
  - Compute the likelihood of the validation data using the obtained model
- Another rather general approach: Gap statistics (rough idea)
  - Generate multiple reference data sets that match the feature ranges of the given data and contain no clustering structure (e.g. uniformly distributed data)
  - For each K, cluster the reference data sets to give us an expectation of the behavior of the clustering algorithm when applied to such reference data
  - Compare the Sum-Squared-Error obtained on real data using the same K with what's obtained on the reference data
  - Large gap indicates a good K

# How to evaluate clustering results?

- By user interpretation
  - does a document cluster seem to correspond to a specific topic?
  - This can be difficult in many domains
- Internal measure: different objective functions used in clustering have also been used to gauge the quality of the clustering solutions
  - Sum-squared-error
  - Likelihood of the data
  - Each provides some information about how well the clustering structure is able to explain the (variation that we see in) data
- External measure: the quality of a clustering can also be measured by its ability to discover some known class structure
  - Use labeled data to perform evaluation and test how well the discovered cluster structure matches the class labels

# Unsupervised dimensionality reduction

- Consider data without class labels
  - Try to find a more compact data representation
  - Create new features defined as functions over all of the original features
- Why?
  - Visualization: need to display low-dimensional version of a data set for visual inspection
  - Preprocessing: learning algorithms (supervised and unsupervised) often work better with smaller numbers of features both in terms of runtime and accuracy

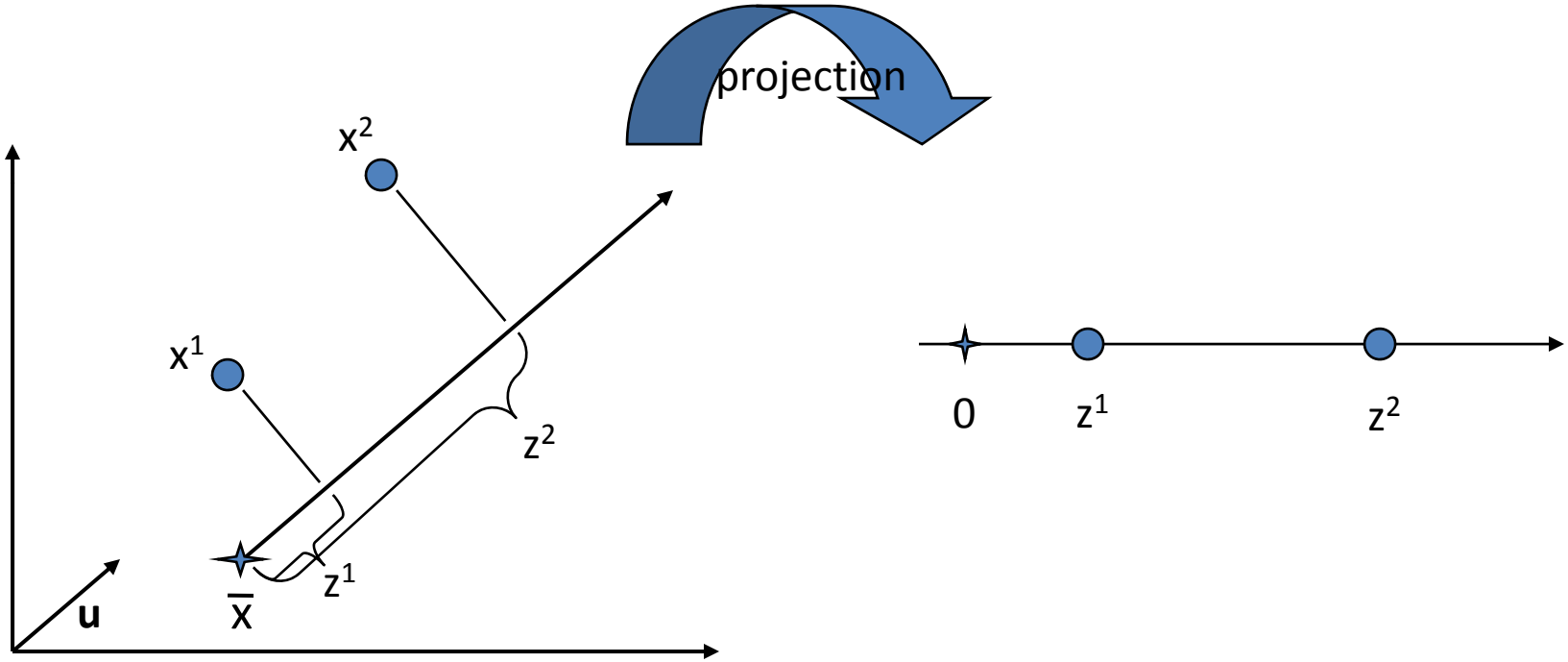
# Principal Component Analysis

- PCA is a classic and often very effective dimensionality reduction technique
- Linearly project n-dimensional data onto a k-dimensional space while preserving information:
  - e.g., project space of 10k words onto a 3d space
- Basic idea: find the linear projection that retains the most variance in data
  - I.e. the projection that loses the least amount of information

# Conceptual Algorithm

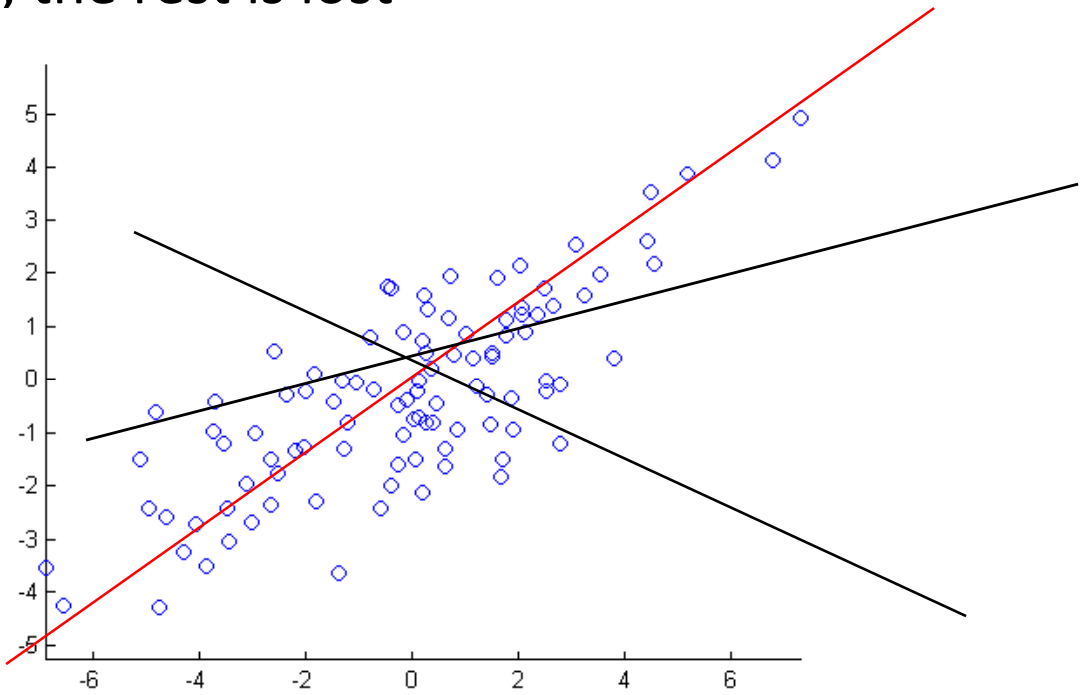
- Find a line such that when the data is projected onto that line, it has the maximum variance

# First, what is a linear projection



# Conceptual Algorithm

- Find a line such that when data is projected to that line, it has the maximum variance
- the variance of the projected data is considered as retained by the projection, the rest is lost



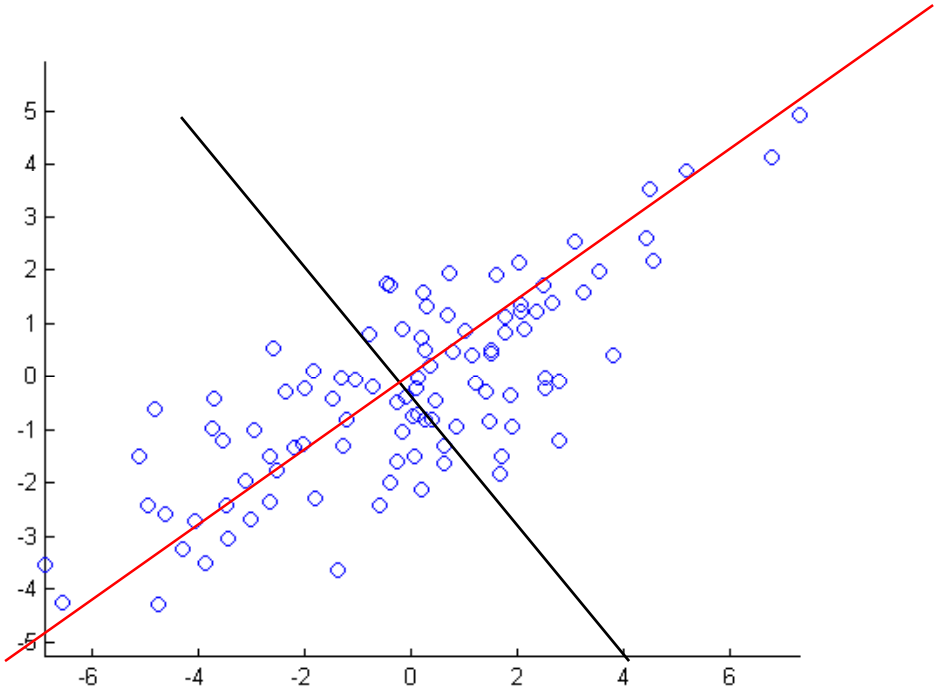
# Conceptual Algorithm

- Once you have found the first projection line, we continue to search for the next projection line by:

  finding a new line, orthogonal to the first, that has maximum projected variance:

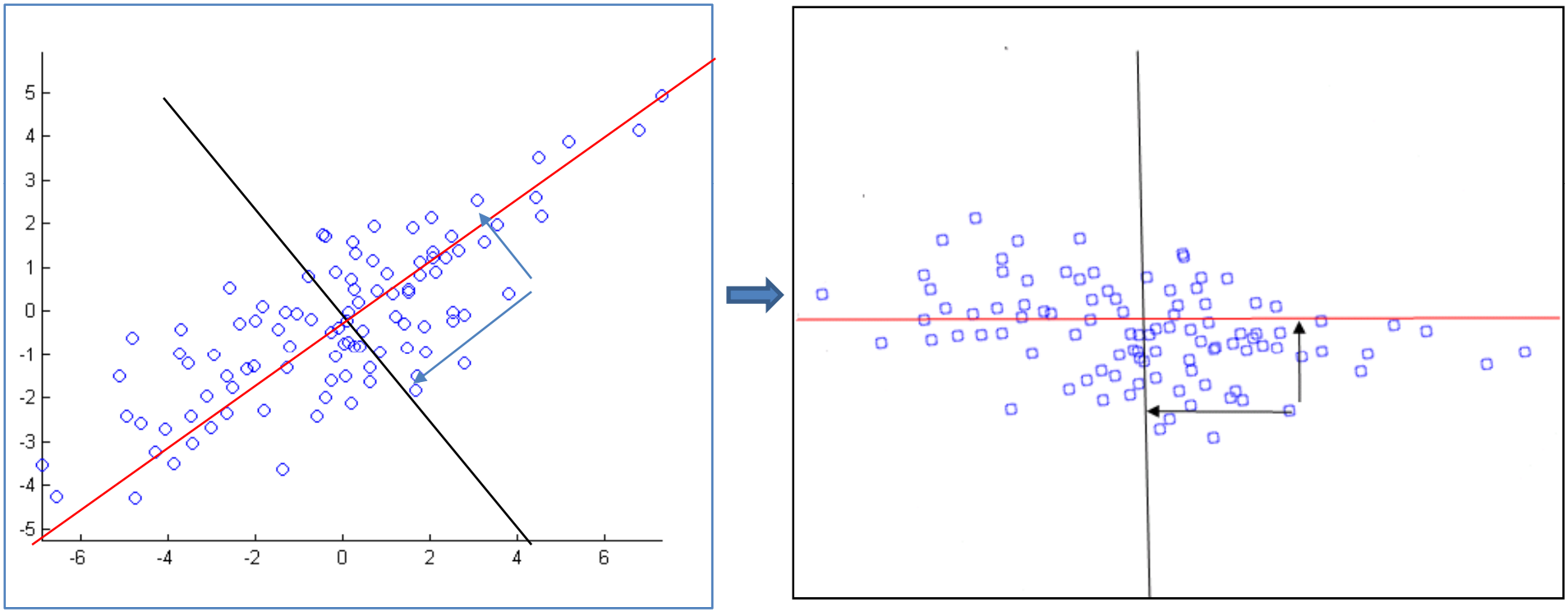
In this case, we have to stop after two iterations, because the original data is 2-d.

But you can imagine this procedure being continued for higher dimensional data.



# Repeat Until k Lines

- The projected position of a point on these lines gives the coordinates in the m-dimensional reduced space




How can we compute this set of lines?

# Basic PCA algorithm

- Start from m by n data matrix **X**
- Compute the mean/center to be the new origin:

  $x_c$= mean(X)
- Compute Covariance matrix Σ

$$\Sigma = \frac{1}{n}\sum_i (\mathrm{x}^i - x_c)(\mathrm{x}^i - x_c)^T$$

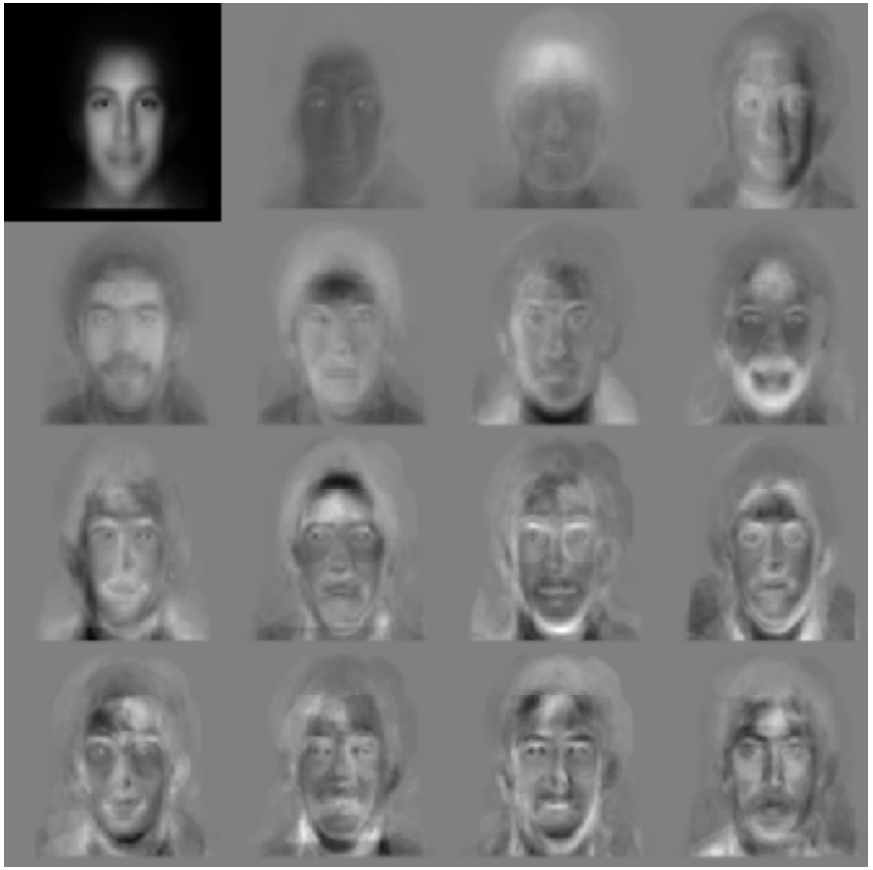
- Compute eigen-vectors and eigen-values of Σ
- Select the set of basis [$u_1$, …, $u_k$] for projection to be the k eigen-vectors with the largest eigen-values
- The new k-d coordinates are: $(x-x_c)^T[u_1, \ldots, u_k]$

# Example: Face Recognition

- An typical image of size 256 x 128 is described by n = 256x128 = 32768 dimensions – each dimension described by a grayscale value
- Each face image lies somewhere in this high-dimensional space
- Images of faces are generally similar in overall configuration, thus
  - They should be randomly distributed in this space
  - We should be able to describe them in a much lower dimensional space

# PCA for Face Images: Eigen-faces

- Database of 128 carefully-aligned faces.
- Here are the mean and the first 15 eigenvectors.
- Each eigenvector (32768 –d vector) can be shown as an image – each element is a pixel on the image
- These images are face-like, thus called eigen-faces



# Face Recognition in Eigenface space
(Turk and Pentland 1991)

- Nearest Neighbor classifier in the eigenface space
- Training set always contains 16 face images of 16 people, all taken under the same set of conditions of lighting, head orientation and image size
- Accuracy:
  - variation in lighting: 96%
  - variation in orientation: 85%
  - variation in image size: 64%

# Face Image Retrieval

- Left-top image is the query image
- Return 15 nearest neighbor in the eigenface space
- Able to find the same person despite
  - different expressions
  - variations such as glasses