PCA

Introduction

Dimensionality reduction

- A second type of unsupervised learning problem called dimensionality reduction.
- In the more general cases, we have n-dimensional data and we'll want to reduce it to k-dimensions. what it means is that we would like to find a k-dimensional surface, direction on which most of the data seems to lie, and project all the data onto that surface, and by doing so, we need only k features to specify the position of a n-dimensional point on the surface.

Motivations

- Fewer dimensions to learn
  - This will allow us to make our learning algorithms run more quickly because of fewer dimensions
- Enable clustering
  - When we apply clustering algorithm like k-means algorithms into date set, the time cost of counting distance between data in n dimensions means a lot. Dimensionality could help us reduce the time cost significantly.
- Better visualization
  - We reduce data from 50 dimensions or whatever, down to two dimensions, or maybe down to three dimensions, so that we can plot it and understand the data better.

Summary of PCA

- For dimensionality reduction, by far the most popular and commonly used algorithm is something called principal components analysis, or PCA.
- So what PCA does formally is that it tries to find a lower dimensional surface, onto which to project the data so that the sum of projection errors is minimized.
- The distances between each point and its projected version are called projection errors.

PCA: the algorithm

Data preprocessing

- Before applying PCA, it is always important to perform mean normalization, and then depending on your data, maybe perform feature scaling as well. By doing this,the features should have zero mean and comparable ranges of values.
- Unlabeled training set: x^((1)), x^((2)), x^((3)),…, x^((m))

- Preprocessing: for mean normalization we first compute the mean of each feature and then we replace each feature, X, with X minus its mean, and so this makes each feature now have exactly zero mean.
- $\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}$ replace each $x_j^{(i)}$ with $x_j^{(i)} - \mu_j$
- Preprocessing: for feature scaling, if different features on different scales, we scale features to have comparable range of values.

PCA algorithm

- Let's say we want to reduce the data to n dimensions to k dimension
- What we're going to do is first computing something called the covariance matrix, and the covariance matrix is commonly denoted by this Greek alphabet sigma.
- $\Sigma = \frac{1}{m}\sum_{i=1}^{n}(x^{(i)})(x^{(i)})^T$
- What we need to do next is computing something called the eigenvectors of the matrix sigma. The way you do that is using this command, u s v equals s v d of sigma. SVD, by the way, stands for singular value decomposition.
- [U, S, V]= svd(Sigma)
- Actually there is another function called I, which can also be used to compute the same thing. and It turns out that the SVD function and the I function it will give you the same vectors, although SVD is a little more numerically stable.

SVD function

- Given an n×n square matrix Σ, for unit vector u and scalar value λ
- Σu = λu, u : eigenvector of Σ, λ : eigenvalue of Σ
- We can get the matrix u by ordering the eigenvectors in the decreasing order of eigenvalues.
- $$U = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \ldots & u^{(n)} \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{n\times n}$$
- And then we grab the first k columns of the u matrix as matrix u-truncate.
- $U_{truncate} = U(:, 1{:}k)$
- Finally this defines how we go from a feature vector x to this reduce dimension representation z.
- $z^{(i)} = (U_{truncate})^T x^{(i)}$

FastMap

Introduction

- What if the data cannot be represented by 2 dimensional or 3 dimensional points directly
- Example: DNA Strings
    - Distance measure: edit distance
        - the minimum number of insertions, deletions or substitutions that are needed to transform first string to the second one
    - Not points in 2d or 3d space, only distance information available

Data Types

- Multimedia data
- Medical data
- Time Series data, e.g. financial data
- String database, e.g. DNA, OCR

K-d space mapping

Problem definition
- **Given** *N* objects and distance function *D* between 2 objects
- **Find** *N* points in a k-dimensional space
- **Such that** the distances are maintained as well as possible

Benefits
- Accelerate search time for queries
- Help with visualization, clustering and data mining
- Plotting objects as points in k=2 or 3 dimensions can reveal general structure of the dataset such as major clusters, the general shape of the distribution, etc.

Goals
- Fast, should be better than O(N^2), or it will be unsuitable for large dataset
- Preserve distance information as well as possible
- Provide very fast algorithm to map new object to image in O(1) or O(n)

MDS: older attempt to solve the problem

- Multidimensional Scaling
- Used to discover the spatial structure of a set of data items from the similarity info
- Steps:
    - Assigns each item to a k-d point using some heuristic or even at random

    - Moves the point to minimize the discrepancy between the actual dissimilarities and the estimated k-d distances
    - Intuitively, it treats each pairwise distance as a “spring” between two points
    - The algorithm tries to re-arrange the positions of the k-d points to minimize the “stress” of the spring
    - $$stress = \sqrt{\frac{\sum_{i,j} \left(\hat{d}_{ij} - d_{ij}\right)^2}{\sum_{i,j} {d_{ij}}^2}}$$
    - dij is the dissimilarity measure, d’ij is the distance between their images Pi and Pj
- Drawback:
    - O(N^2), not suitable for large datasets
    - Complexity of answering a query would be as bad as sequential scanning, remains questionable in information retrieving

FasrMap: the algorithm

Introduction

- Time complexity: O(Nk)
- Key idea: to pretend that objects are indeed points in an unknown n-dimensional space, and to try to project these points on an k-dimensional space
- Challenge: to compute these projections from the distance matrix/function only -> the only input we have

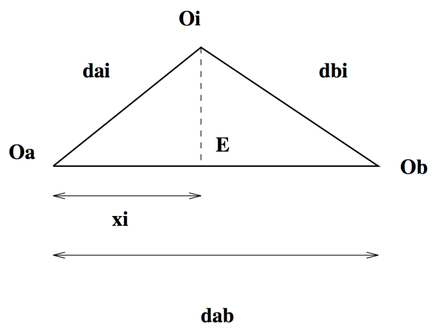
Choosing the pivot objects

- We would like to find a line on which the projections are as far apart from each other as possible.
- To perform Fastmap in a fast way, we choose the farthest pair of objects in the following way:

```
Algorithm 1 choose-distant-objects ( $\mathcal{O}$, $dist()$ )
begin
  1) Chose arbitrarily an object, and declare it to be the second pivot object $O_b$
  2) set $O_a$ = (the object that is farthest apart from $O_b$) (according to the distance function
     $dist()$)
  3) set $O_b$ = (the object that is farthest apart from $O_a$)
  4) report the objects $O_a$ and $O_b$ as the desired pair of objects.
end
```

- Figure 4: Heuristic to choose two distant objects.
- Of course we can compute every pair in the dataset and compare which pair yields best performance, but for the sake of computation, Fastmap makes a compromise on quality of the projection in order to be Fast.
- Normally it takes O(n) on time complexity, and to optimize the result we can iterate on the middle two steps for 4 or 5 times.

The problem for k = 1

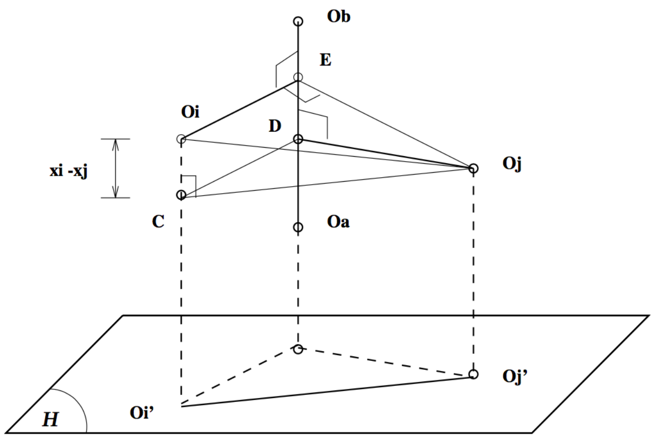


- Heart of Fastmap: to project the objects on a carefully selected “line” -> we choose two objects Oa and Ob as pivot objects and consider line OaOb
- Projections of objects on line OaOb, the coordinate of which denoted as Xi, are computed using the cosine law.

$$x_i = \frac{{d_{a,i}}^2 + {d_{a,b}}^2 - {d_{b,i}}^2}{2d_{a,b}}$$

-
- The computation of Xi only needs the distances between objects, which are given
- Objects mapped into points on a line, preserving some of the distance info

Extension for many dimensions



- Pretending that the objects are points in n-dim space
- Consider a (n-1)-d hyperplane H that is perpendicular to line OaOb
- Project objects on this hyperplane H
- Need to determine the distance function D’ between two of the projections on the hyperplane H, Oi’Oj’
- Using Pythagorean theorem

$$(\mathcal{D}'({O_i}', {O_j}'))^2 = (\mathcal{D}(O_i, O_j))^2 - (x_i - x_j)^2 \quad i, j = 1, \ldots, N$$

-

- Ability to compute D’ allows us to projects objects on a second line
- After k iterations, objects have been projected onto k orthogonal axes, each of the objects has a k-d coordinate in k-dimensional space