Stanford Machine Learning - Week VIII

Eric N Johnson

September 1, 2016

Part I
Clustering

1 Unsupervised Learning: Introduction

Supervised learning is the machine learning task of inferring a function from labeled training data. Unsupervised learning is a type of machine learning algorithm used to draw inferences from data sets consisting of input data without labeled responses.

I.e.,

- Supervised: $\{(x^{(1)}, y^{(1)}), ..., (x^{(m)}, y^{(m)})\}$
- Unsupervised: $\{x^{(1)}, x^{(2)}, ..., x^{(m)}\}$

We don't give an unsupervised learning algorithm any structure. Instead, the algorithm looks for that structure.

Some examples:

- Market segmentation
- Social network analysis
- Organize computing clusters
- Astronomical data...

1.1 K-Means Clustering

Here we are given an unlabeled data set and want to produce a set of K clusters. $K-Means$ is an iterative two-step algorithm:

1. Cluster Assignment Step:

   The algorithm picks a cluster for each of the points.

2. Group Centering Step

   The algorithm calculates the new means to become the centroids of the observations in the new clusters.

$K-Means$ takes, as input, the following:

- K (the number of clusters to produce)
- A training set: $\{x^{(1)}, x^{(2)}, ..., x^{(m)}\}$

where $x^{(i)} \in \mathbb{R}^n$. We drop $x_0 = 1$ by convention.

1.2 K-Means Algorithm

Randomly initialize K cluster centroids $\mu_1, \mu_2, ..., \mu_K \in \mathbb{R}^n$
Repeat{
    for i = 1 to m
        $c^{(i)}$:= index from 1 to K of cluster centroid closest to $x^{(i)}$
            := $\min_k |x^{(i)} - \mu_k|_2^2$
    for k = 1 to K
        μ_k := mean of points assignmed to cluster k
}

1. If we have run a $K-means$ algorithm until it converges and have $c^{(1)} = 3, c^{(2)} = 3, c^{(3)} = 5, ...$, then the first and second training examples have been assigned to cluster 3. The third example has been assigned to cluster 5, and we can say that "of the possible values of $k \in \{1, 2, ..., K\}$, the value $k = 3$ minimizes $|x^{(2)} - \mu_k|_2^2$.
2. If we have, say, $x^{(1)} \to c^{(1)} = 3, x^{(4)} \to c^{(4)} = 3, x^{(11)} \to c^{(11)} = 3$, (and no others) then clearly

$$\mu_3 = \frac{1}{3}\left(x^{(1)} + x^{(4)} + x^{(11)}\right) \in \mathbb{R}^n$$

What if our clusters are not well defined? $K - Means$ does not particularly care... It will produce some kind of output anyway.

1.3 Optimization Objective

We will minimize the following cost function in producing the the K clusters. Denote

$$\mu_{c^{(i)}} = \text{Cluster centroid of cluster to which example } x^{(i)} \text{ has been assigned.}$$

For example, if $x^{(i)} \to 5$, it follows that $c^{(i)} = 5$ and $\mu_{c^{(i)}} = \mu_5$.

Optimization Objective

$$J(c^{(1)}, ..., c^{(m)}, \mu_1, ..., \mu_K) = \frac{1}{m}\sum_{i=1}^{m}\left|x^{(i)} - \mu_{c^{(i)}}\right|_2^2$$

$$\min J(c^{(1)}, ..., c^{(m)}, \mu_1, ..., \mu_K)$$

$$\text{w.r.t. } \{c^{(1)}, ..., c^{(m)}\} \text{ and } \{\mu_1, ..., \mu_K\}$$

Equivalently, we minimize the value of $c^{(i)}$ and the distance to the mean that $x^{(i)}$ belongs to. The second step of the algorithm minimizes the cost with respect to the set $\{\mu_1, \mu_2, ..., \mu_K\}$.

Note: The cost should decrease monotonically. If we see oscillations in the value of the cost function there is likely a bug in the code.

1.4 Random Initialization

How do we initialize $K - Means$? How can we avoid local minima? Perform a random initialization.

- We should have $K < m$.
- Randomly pick K training examples from these m examples.
- Set $\mu_1, ..., \mu_K$ equal to these K examples.

Doing this is considered best practice for $K-Means$ optimization. It is good to attempt multiple random initializations.

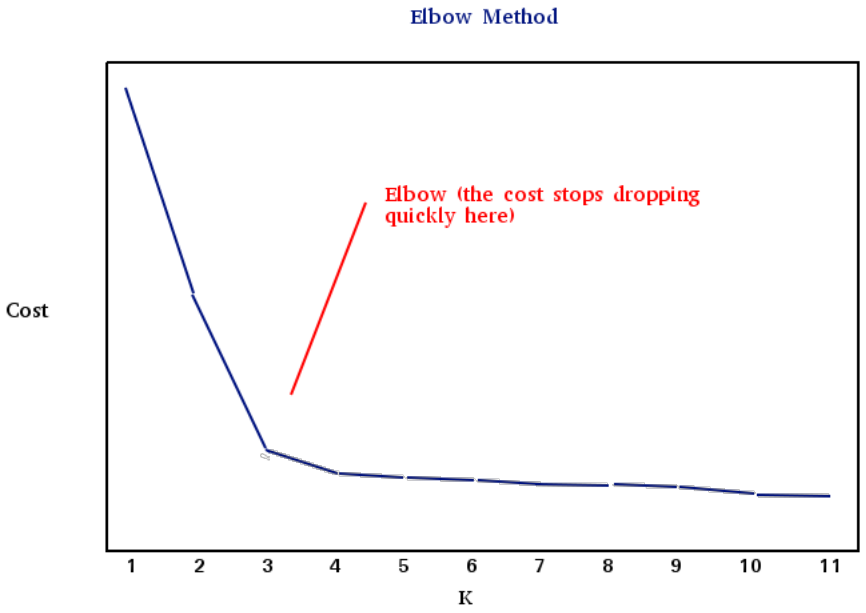
for i = 1 to 100 {
    Randomly initialize *K-Means*
    Run *K-Means* to obtain $c^{(1)}, ..., c^{(m)}$ and $\mu_1, ...\mu_K$
    Compute the cost function (distortion)
        $J(c^{(1)}, ..., c^{(m)}, \mu_1, ..., \mu_K)$
}
Pick the clustering that gives us the lowest cost.

Even if you have a large number of clusters, this method will work.

1.5 Choosing the Number of Clusters

Most commonly the number of clusters is chosen by hand. But there are other methods.

- *Elbow Method*: Run *K-Means* with $K = 1, 2, ...$ clusters. Plot the cost function against K. If the cost goes down quickly after some value of K, pick that.



However, this method doesn't always work because sometimes the cost will drop too smoothly and will not provide a clear value of K.

- *Downstream Method*: Consider what is being done with the *K-Means* algorithm. What value of K will work best there? For instance, if you're running *K-Means* to evaluate what shirt sizes to produce, there is a reasonable limit to K since you won't produce 25 different sized shirts.

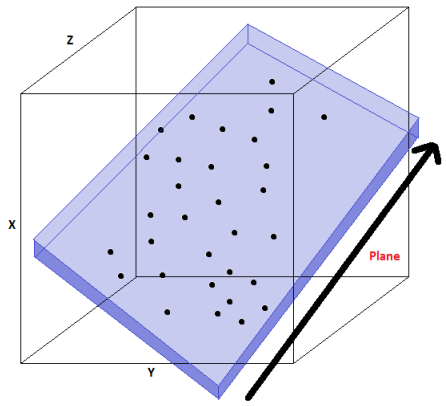
Part II
Dimensionality Reduction

If some data is redundant or linearly-dependent, we may be able to drop one feature. For example, if a data column is in inches and another column is in centimeters. Other highly correlated features may not be this obvious but it is always worth looking at. More sophisticated methods exist that help us reduce feature dimension.

2 Motivation

2.1 Motivation I: Data Compression

Here is an example image of 3-dimensional data projected onto a plane



which involves projecting $x \in \mathbb{R}^3$ onto $z \in \mathbb{R}^2$.

The course lecture does not go into any detail about how to calculate these projections. However, the expressions are simple.

- A scalar projection of a vector b onto a vector a is given by

$$\begin{aligned} \text{comp}_a b &= \frac{a \cdot b}{|a|} \\ &= \frac{a^T b}{|a|_2} \end{aligned}$$

- A vector projection of a vector b onto a vector a is given by

$$\begin{aligned} \text{proj}_a b &= \frac{a \cdot b}{|a|} \frac{a}{|a|} \\ &= \frac{a \cdot b}{|a|^2} a \\ &= \frac{a^T b a}{|a|_2^2} \end{aligned}$$

  Note that $a \cdot b = a^T b = b^T a = 0$ iff $a \perp b$, and that $a \cdot b = |a|_2 |b|_2 \cos(\phi)$, where ϕ is the angle between a and b.

We can easily project our training set onto a lower-dimensional basis set and accomplish a dimension reduction.

2.2 Motivation II: Visualization

Dimensionality reduction will also help us to visualize data. If we have a 50-dimensional feature set, we can project all of the information onto a some smaller $z \in \mathbb{R}^2$ or $z \in \mathbb{R}^3$.

3 Principal Component Analysis

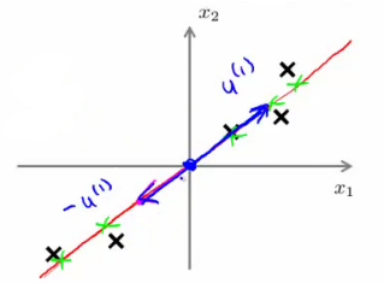
PCA tries to find a lower-dimensional surface onto which it can project data so that the sum of squares for error (or projection error) is minimized.

PCA is used to:

- Compress data.
  - Reduce memory and disk space needed to store data
  - Speed up learning algorithms by using smaller sets
- For visualizing high-dimensional data by reducing it into something that can be plotted.

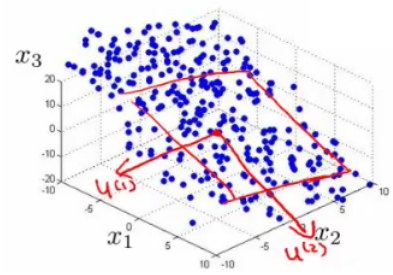
3.1 Principal Component Analysis Problem Formation

- To reduce from 2-dimension to 1-dimension, we find a direction (vector $u^{(1)} \in \mathbb{R}^1$) onto which to project the data so as to minimize the projection error.



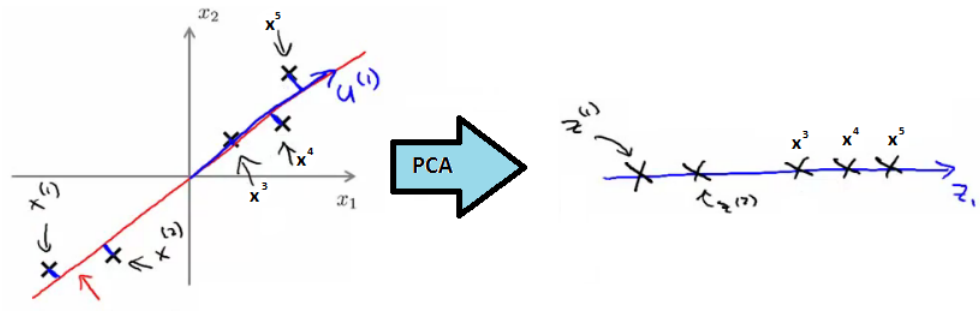
  Here $x^{(i)} \in \mathbb{R}^2 \rightarrow z^{(i)} \in \mathbb{R}$.

- To reduce from n-dimension to k-dimension, we find k vectors $u^{(1)}, u^{(2)}, ..., u^{(k)}$ onto which to project the data so as to minimize the projection error.



  Here $x^{(i)} \in \mathbb{R}^3 \rightarrow z^{(i)} \in \mathbb{R}^2$.

Aside from some cosmetic similarities, PCA is *not* linear regression. The quantity being minimized (projection error) is orthogonal to $u^{(1)}$.



3.2 Principal Component Analysis Algorithm

Before we perform PCA, we need to execute some data **preprocessing** to scale features and perform mean normalization.

PCA Preprocessing:

Training set: $x^{(1)}, x^{(2)}, ..., x^{(m)}$

Preprocessing (feature scaling / mean normalization):

$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}$

Replace each $x_j^{(i)}$ with $x_j - \mu_j$

If different features are on different scales, scale the features so they all have comparable ranges.

$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{s_j}$

where s_j is standard deviation or some similar quantity

PCA Algorithm:

To reduce data from n to k dimensions:

Compute covariannce matrix

$\Sigma = \frac{1}{m}\sum_{i=1}^{n}(x^{(i)})(x^{(i)})^T \in \mathbb{R}^{n\times n}$

(Sigma = (1/m)*X'*X;)

Compute the eigenvectors of this matrix $\Sigma \in \mathbb{R}^{n\times n}$ using Singular Value Decomposition

([U,S,V] = svd(Sigma);)

Take first k columns of the upper-triangular matrix $U_{\text{reduced}} \in \mathbb{R}^{n\times k}$ from the SVD

(Ureduce = U(:,1:k);)

Set z

$z = \left[\begin{bmatrix} u^{(1)} & u^{(2)} & \cdots & u^{(k)} \end{bmatrix}^T x\right] \in \mathbb{R}^{n\times k} x$

such that $z_j = (u^{(j)})^T x$

(z = Ureduce'*x;)

4 Applying PCA

Since we can 'compress data' using PCA, we should be able to reconstitute the data back into its original form. In other words, how do we compute $z \in \mathbb{R} \rightarrow x \in R^2$?

4.1 Reconstruction from Compressed Representation

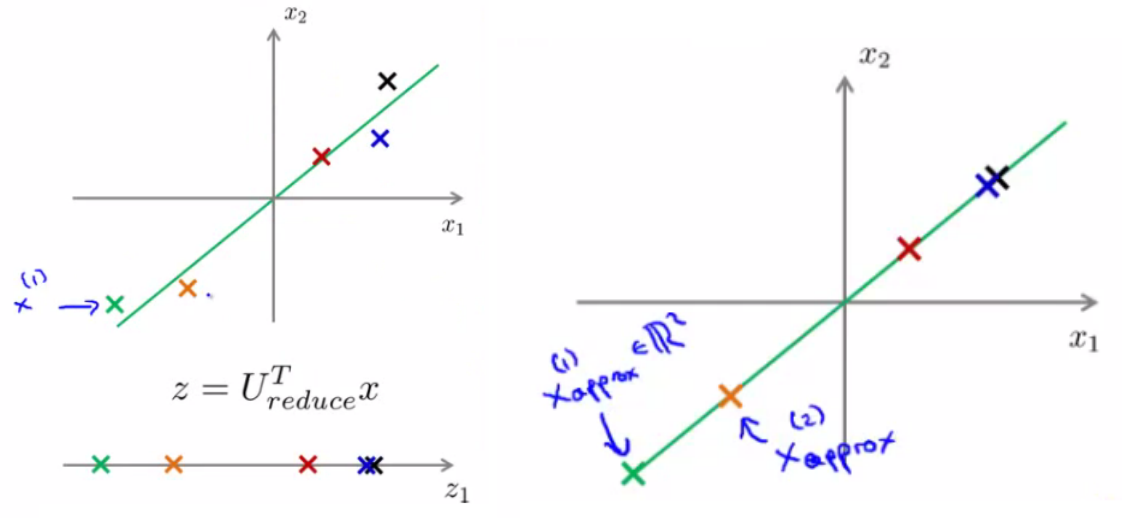
We have

$$z = U_{\text{reduce}}^T \, x$$

and may get *an approximation to* x by computing

$$x_{\text{approx}} = U_{\text{reduce}} \, z$$

which is called a *reconstruction* of x.



4.2 Choosing the Number of Principal Components k

PCA tries to minimize the average squared projection error:

$$\text{Proj}_{\text{error}} = \frac{1}{m} \sum_{i=1}^{m} \left| x^{(i)} - x_{\text{approx}}^{(i)} \right|_2^2$$

where the total variation in the data is given by:

$$\text{Var} = \frac{1}{m} \sum_{i=1}^{m} \left| x^{(i)} \right|_2^2$$

We typically choose k to be the smallest value so that

$$\frac{\frac{1}{m} \sum_{i=1}^{m} \left| x^{(i)} - x_{\text{approx}}^{(i)} \right|_2^2}{\frac{1}{m} \sum_{i=1}^{m} \left| x^{(i)} \right|_2^2} \leq 0.01$$

In other words, we try to keep this ratio below 1%. Thus 99 % of the variance is retained. We can often reduce the dimensionality of the data but still retain 90% of the variance.

Algorithm for Choosing k:

Try PCA with $k = 1, k = 2, k = 3, ...$

Compute

U_{reduce}

$z^{(1)}, z^{(2)}, ..., z^{(m)}$

$x^{(1)}_{\text{approx}}, x^{(2)}_{\text{approx}}, ..., x^{(m)}_{\text{approx}}$

Check the ratio:

$$\frac{\frac{1}{m}\sum_{i=1}^{m}\left|x^{(i)} - x^{(i)}_{\text{approx}}\right|_2^2}{\frac{1}{m}\sum_{i=1}^{m}\left|x^{(i)}\right|_2^2} \leq 0.01 \quad \text{(threshold)}$$

This seems inefficient, but we can use the Singular Value Decomposition to make this process simpler. When we compute [U,S,V] = svd(Sigma), the matrix S is a diagonal matrix:

$$S = \begin{bmatrix} s_{11} & 0 & 0 & \cdots & 0 \\ 0 & s_{22} & 0 & \cdots & 0 \\ 0 & 0 & \ddots & 0 & 0 \\ \vdots & 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & 0 & s_{nn} \end{bmatrix}$$

and for a given value of k, this 'ratio' term we were calculating can be approximated by simply finding this quantity:

$$1 - \frac{\sum_{i=1}^{k} s_{ii}}{\sum_{i=1}^{n} s_{ii}} \leq 0.01 \quad \text{(threshold)}$$

which we can use to find a good value of k without doing more work than finding the SVD. Note also that we only have to produce the SVD one time.

The threshold value can be anything from 1% to 15 %.

4.3 Advice for Applying PCA

We can use PCA to speed up learning algorithms. Say we have

$$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(m)}, y^{(m)})$$

that are very high-dimensional (as in $x^{(i)} \in \mathbb{R}^{10,000}$). But with PCA, we can reduce this dimension. Here's how:

1. Extract input vectors to produce an unlabeled dataset

$$x^{(1)}, x^{(2)}, ..., x^{(m)}$$

2. Use PCA to reduce this dimension (to maybe $z \in \mathbb{R}^{1000}$)

$$z^{(1)}, z^{(2)}, ..., z^{(m)}$$

3. This produces a lower-dimensional training set

$$(z^{(1)}, y^{(1)}), (z^{(2)}, y^{(2)}), ..., (z^{(m)}, y^{(m)})$$

Note: The mapping $x^{(i)} \to z^{(i)}$ should be defined by running PCA on ONLY the training set. The matrices U_{reduce} and those from the Singular Value Decomposition created with PCA should be computed only with the training data.

This mapping can be applied as well to the examples $x^{(i)}_{cv}$ and $x^{(i)}_{\text{test}}$.

4.3.1 Don't do this

- Sometimes people use PCA to prevent overfitting. This works because they use $z^{(i)}$ instead of $x^{(i)}$ to reduce the number of features to $k < n$. With fewer features a model is less likely to overfit. Why? Why not use regularization instead? PCA is always going to throw away information from data.
- There is a tendency to use PCA where it should not be used. Consider: Can this ML system work without doing PCA? If that doesn't work then perhaps doing PCA will be appropriate - but this should not be done by default.
- Remember that PCA should be used to:

  1. Lower the amount of space data will take
  2. To speed up ML algorithms
  3. To visualize high-dimensional data

  PCA should not be used to do anything else.