machine learning_week8

Created: 18/11/26 08:12 **Updated:** 18/11/27 10:37

Clustering

- introduction
  - recall supervised learning: data set have **label y**👉find model to fit **hypothesis**
  - **unsupervised learning:**
    - **no label**👉find **structure** in data set
    - cluster applications:
      - Market segmentation
      - Social network analysis
      - Organizing computer clusters
      - Astronomical data analysis
- **K-Means**
  - notations:
    - $c^{(i)}$ = index of cluster (1,2,...,K) to which example x(i) is currently assigned
    - μ_k = cluster centroid k (μk∈ℝn)
    - $\mu_{c^{(i)}}$ = cluster centroid of cluster to which example x(i) has been assigned
  - automatically grouping data into coherent subsets
  - algorithm:

K-means algorithm

finally get K classes

Randomly initialize K cluster centroids $\mu_1, \mu_2, \ldots, \mu_K \in \mathbb{R}^n$

Repeat {

for i = 1 to m — Cluster Assignment

$c^{(i)}$:= index (from 1 to K) of cluster centroid closest to $x^{(i)}$

for k = 1 to K — Move Centroids

μ_k := average (mean) of points assigned to cluster k

}

    - cluster assignment

$$c^{(i)} = argmin_k \, ||x^{(i)} - \mu_k||^2$$

That is, each $c^{(i)}$ contains the index of the centroid that has minimal distance to $x^{(i)}$.

    - move centroids

$$\mu_k = \frac{1}{n}[x^{(k_1)} + x^{(k_2)} + \cdots + x^{(k_n)}] \in \mathbb{R}^n$$

Where each of $x^{(k_1)}, x^{(k_2)}, \ldots, x^{(k_n)}$ are the training examples assigned to group $m\mu_k$.

    - cluster centroid with 0 assigments
      - re-initialize (if u certainly need k cluster)
      - eliminate it
  - more: data is **non-separated?**
    - K-means can still segment your data into K subsets, which can still be useful in some cases (like market segmentation).

- **trick**
  - **Optimization Objective**
    - **distortion** of training examples:

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

    - aim:

In the **cluster assignment step**, our goal is to:

Minimize J(...) with $c^{(1)},\dots,c^{(m)}$ (holding $\mu_1,\dots,\mu_K$ fixed)

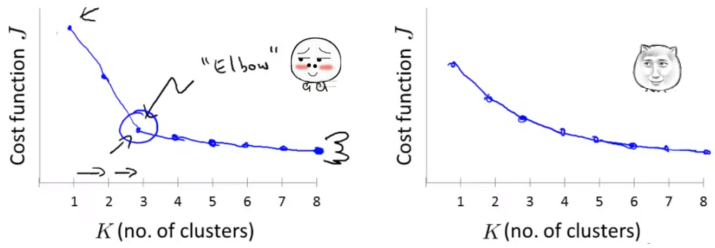
In the **move centroid** step, our goal is to:

Minimize J(...) with $\mu_1,\dots,\mu_K$

With k-means, it is **not possible for the cost function to sometimes increase**. It should always descend.

  - **Random Initialization**
    - Randomly pick K unique training examples from training set.
    - set μ equal to those K examples
    - caution!
      - as we make random selection, K-Means may finally end up different clusters. Further, it may get stuck in **local optima.**
      - **solution**
        - run K-Means many times and choose the min J one.
  - **Choosing K**
    - manual
    - Elbow method

Elbow method:



      - compute j with different k, and find **elbow**(show in the left chart)
      - don't have high expectation LOL
    - choose K that proves to be most useful for some **goal** you're trying to achieve from using these clusters.
- **drawback**
  - summary
    - k-means assumes the variance of the distribution of each attribute (variable) is spherical;
    - all variables have the same variance;
    - each cluster has roughly equal number of observations
    - can use SSE(cost function) to mearsure
  - by knowing drawback, we could know how to fix them.
  - details: How to understand the drawbacks of K-means

Dimensionality Reduction_PCA

- **motivation**
  - **data compression**
    - save disk place
    - clear redundant features
    - speed up learning algorithm
  - **data visualization**
    - we may loose the exactly meaning of features but by reducing dimensions to 2 or 3 D, we can plot them, which let us get intuition of data set.
- PCA(Principal Component Analysis)

- Reduce from n-dimension to k-dimension:
  - Find k vectors $u^{(1)}, u^{(2)}, \ldots, u^{(k)}$ onto which to project the data
  - **minimize** the projection error.
- **projection error:** the average of all the **distances** of every feature to the projection line. **(shortest orthogonal distances)**
- **algorithm**
  - (remember: features scaling and mean normalization.)

Principal Component Analysis (PCA) algorithm summary

→ After mean normalization (ensure every feature has zero mean) and optionally feature scaling:

$$\text{Sigma} = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)})(x^{(i)})^T$$

$$X = \begin{bmatrix} — (x^{(1)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix}$$

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

```
[U,S,V] = svd(Sigma);
Ureduce = U(:,1:k);
z = Ureduce'*x;
```

- concretely:

1. Compute "covariance matrix"

1. $\Sigma = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)})(x^{(i)})^T$
2. **Compute "eigenvectors" of covariance matrix Σ**
   - by singular value decomposition (SVD)

3. Take the first k columns of the U matrix and compute z

We'll assign the first k columns of U to a variable called 'Ureduce'. This will be an n×k matrix. We compute z with:

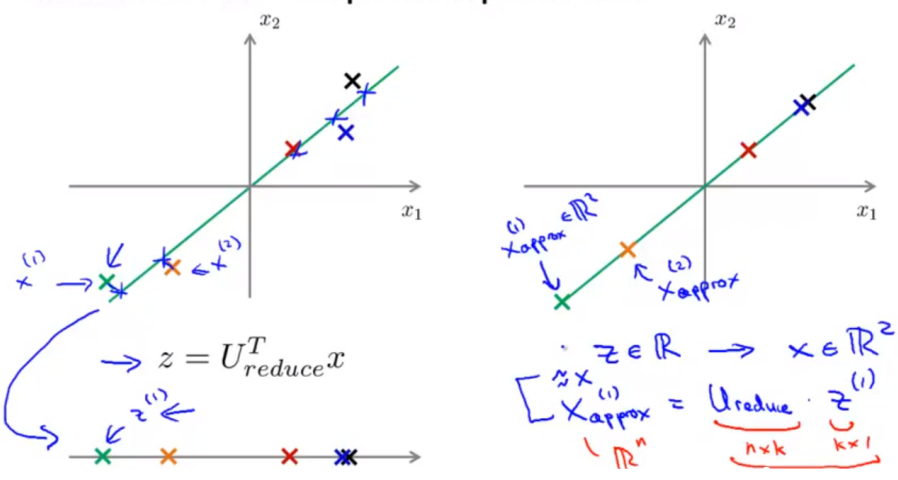
$z^{(i)} = Ureduce^T \cdot x^{(i)}$

3. $UreduceZ^T$ will have dimensions k×n while x(i) will have dimensions n×1. The product $Ureduce^T \cdot x^{(i)}$ will have dimensions k×1.

```
Sigma = (1/m) * X' * X; % compute the covariance matrix
[U,S,V] = svd(Sigma);   % compute our projected directions
Ureduce = U(:,1:k);     % take the first k directions
Z = X * Ureduce;        % compute the projected data points
```

- Recustruction
  - uncompress data,and go back to our original number of features

Reconstruction from compressed representation



  - 2-D example here:

- notice that after uncompressed, the X_{approx} may a bit different from original one.
- by linear algebra, we know that
  - z=U*X
  - so $X_{approx}=U^{-1}*z$
  - what's more, the U here is a Unitary Matrix .
    - One of the special properties of a Unitary Matrix is: $U^{*}=U^{-1}$,where the "*" means "conjugate transpose".
  - so we get the equation of uncompress.

Choosing the Number of Principal Components

- evaluate performance (**retained variance)**

One way to choose k is by using the following formula:

- Given the average squared projection error: $\frac{1}{m}\sum_{i=1}^{m}||x^{(i)}-x_{approx}^{(i)}||^2$
- Also given the total variation in the data: $\frac{1}{m}\sum_{i=1}^{m}||x^{(i)}||^2$
- Choose k to be the smallest value such that: $\frac{\frac{1}{m}\sum_{i=1}^{m}||x^{(i)}-x_{approx}^{(i)}||^2}{\frac{1}{m}\sum_{i=1}^{m}||x^{(i)}||^2}\leq 0.01$

  - (in this pic, we can say that 99% of the variance is retained)
- what's more, evaluation can be rewrite :

$$\frac{\sum_{i=1}^{k}S_{ii}}{\sum_{i=1}^{n}S_{ii}}\geq 0.99$$

  - S_{ij} is the diagonal elements in U
  - proof of this not included
- usage:
  1. choose k:
     1. first set the percentage of retained variance we want.
        1. Try PCA with k=1,2,...
        2. Compute U_{reduce}, z, x
        3. Check the formula given above that 99% of the variance is retained. If not, go to step one and increase k.
     2. then
     3. clever way: just use `[U,S,V] = svd(Sigma)` to get U and check $\frac{\sum_{i=1}^{k}S_{ii}}{\sum_{i=1}^{n}S_{ii}}$
  2. evaluate compression.
     - after compression, we check how much variance retain by:

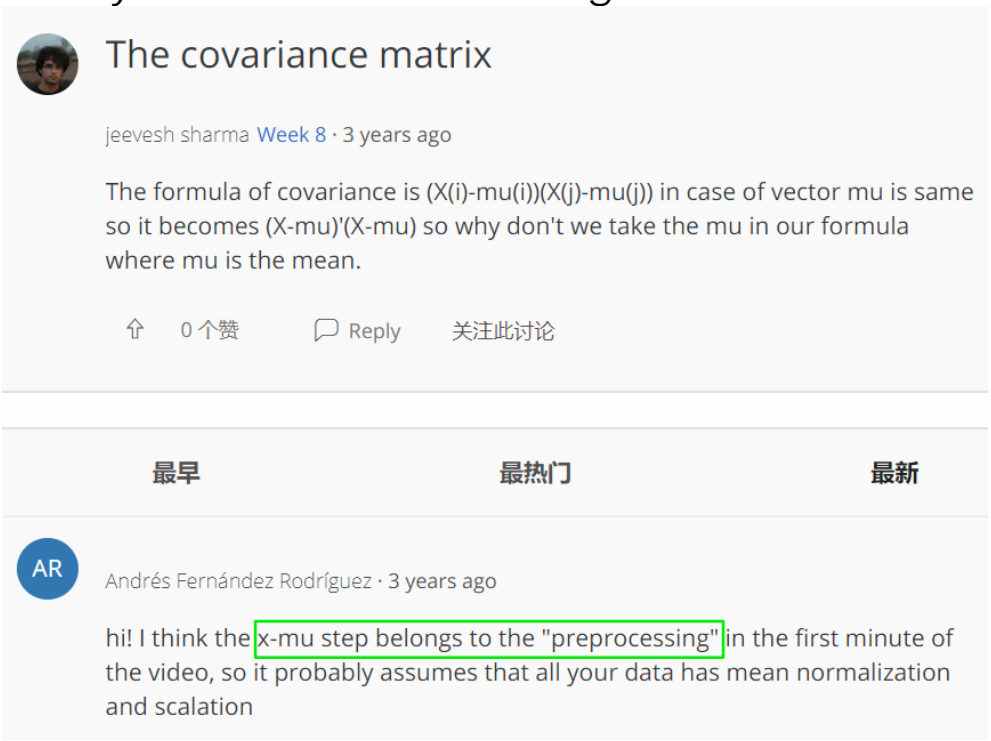
$$\frac{\sum_{i=1}^{k}S_{ii}}{\sum_{i=1}^{n}S_{ii}}$$

Advice for Applying PCA

- speed up supervised algorithm
  - reduce dimensions of input
    - e.g. Image recognition (a image's features can be 100X100)
  - essence: define mapping (caution: should only do on training set)
- misuse: prevent over-fitting
  - because PCA doesn't concern about label y. And when implementing PCA we may throw some valuable information related to label y.

Q&A:

1. why compute co-variance matrix by $\Sigma = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)})(x^{(i)})^T$ instead of $\Sigma = \mathrm{E}\left[(\mathbf{X} - \mathrm{E}[\mathbf{X}])\,(\mathbf{X} - \mathrm{E}[\mathbf{X}])^{\mathrm{T}}\right]$
   - because we already done in feature scaling.



   - thanks forum
2. why simply choose 1:k from matrix U?

Q1) How do we know which are the most significant features to retain? Why can we choose just the first K of them? They aren't naturally ordered by significance are they? Wouldn't it make more sense to select the K most influential dimensions?

Prof Ng doesn't get into the details of how Singular Value Decomposition (SVD) works, but it turns out that the output of SVD is exactly what we need for the purposes of PCA. It does "spectral decomposition" of the input matrix and gives it back expressed in a form in which it is obvious which are the important dimensions. The output of SVD is:

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

Where Sigma is the covariance matrix. The output values U and V are unitary matrices and the columns of U are the eigenvectors of the transformation. S is a diagonal matrix, containing the corresponding eigenvalues in decreasing order. In order words, the SVD has done the work to figure out which dimensions are the most significant and gives us the results in that order. Prof