HW9 Recitation
Learning Paradigms

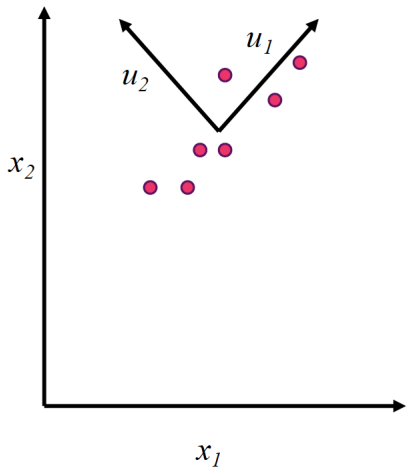
10-301/10-601: Introduction to Machine Learning

2022-04-22

1 Principal Component Analysis

Principal Component Analysis aims to project data into a lower dimension, while preserving as much as information as possible.

How do we do this? By finding an orthogonal basis (a new coordinate system) of the data, then pruning the "less important" dimensions such that the remaining dimensions minimize the squared error in reconstructing the original data.



Finding the vectors is quite easy visually as seen above, but how do we do this mathematically? We find the orthogonal vectors $\mathbf{u}_1 \dots \mathbf{u}_M$ such that it minimizes the sum of the errors of $||\mathbf{x}^{(n)} - \hat{\mathbf{x}}^{(n)}||^2$, where $\mathbf{x}^{(n)}$ is the given data point and $\hat{\mathbf{x}}^{(n)}$ is the reconstructed vector (obtained by projecting $\mathbf{x}^{(n)}$ to the lower dimension, then projecting back to the original dimension).

Algorithm 1: Principal Component Analysis

Given $M < d$
Given zero-mean data matrix $X \in \mathbb{R}^{d \times N}$, where $\mathbf{x}^{(i)}$ refers to the ith column of X
Find set of M vectors $[\mathbf{u}_1, \ldots, \mathbf{u}_M]$
that minimizes $E_M = \frac{1}{N} \sum_{n=1}^{N} \|\mathbf{x}^{(n)} - \hat{\mathbf{x}}^{(n)}\|^2$
where $\hat{\mathbf{x}}^{(n)} = \sum_{i=1}^{M} z_i^{(n)} \mathbf{u}_i$
where $z_i^{(n)} = (\mathbf{u}_i)^T \mathbf{x}^{(i)}$
and $\|\mathbf{u}_i\| = 1$ for all i

If we have M new vectors, and d original vectors, where $M < d$, it is usually not possible to reconstruct the original data without losing any error. In other words, if $M = d$, we can reconstruct the original data with 0 error. So, we know that all the error comes from the (M-d) missing components (vectors). This error can be expressed in terms of the covariance matrix of the original data, and the error is minimized when the principal component vectors $\mathbf{u}_1 \ldots \mathbf{u}_M$ are the top M eigenvectors of the covariance matrix (in terms of eigenvalues). The higher the eigenvalues for these eigenvectors are, the more information they store and the lower reconstruction error becomes.

For the following questions, use this Colab notebook.

Let's assume we've performed PCA on the following dataset:

Row	X1	X2	X3	X4
1	-0.21	-0.61	-0.35	0.08
2	0.15	-0.77	1.26	1.57
3	0.03	0.12	-0.39	-0.25
4	0.92	1.31	0.31	1.19
5	2.51	1.99	1.86	2.57
6	0.91	1.23	-0.01	0.04

And we've obtained the following principal components:

PC1	PC2	PC3	PC4
-0.53	0.23	0.48	-0.66
-0.49	0.7	-0.27	0.44
-0.43	-0.46	0.52	0.57
-0.54	-0.49	-0.65	-0.21

Which correspond to the following eigenvalues:

[3.265, 0.999, 0.043, 0.014]

1. Why are there only 4 principal components?

2. How much of the variance in the data is preserved by the first two principal components?

3. How much of the variance in the data is preserved by the first and third principal components?

4. Perform a dimensionality reduction on the points such that we project them onto the first two principal components. Then, inverse transform it back to four dimensions. What is the reconstruction error for this sample?

5. Perform a dimensionality reduction such that we project the points onto the first and third principal components. Then, inverse transform it back to four dimensions. What is the reconstruction error of this new dataset?

6. Consider the reconstruction error of the fourth row in particular. Is it lower using the first and second principal components or using the first and third? Why might this be the case?

2 K-Means

Clustering is an example of unsupervised machine learning algorithm because it serves to partition **unlabeled** data. There are many different types of clustering algorithms, but the one that is used most frequently and was introduced in class is **KMeans**.

In KMeans, we aim to minimize the objective function:

$$\sum_{i=1}^{n} \min_{j \in \{1,...,k\}} ||\mathbf{x}^{(i)} - \mathbf{c}_j||^2 \tag{1}$$

Recall the KMeans algorithm from class:

Let $\mathcal{D} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(n)}\}$ where $\mathbf{x}^{(i)} \in \mathbb{R}^d$ be the set of input examples that each have d features.

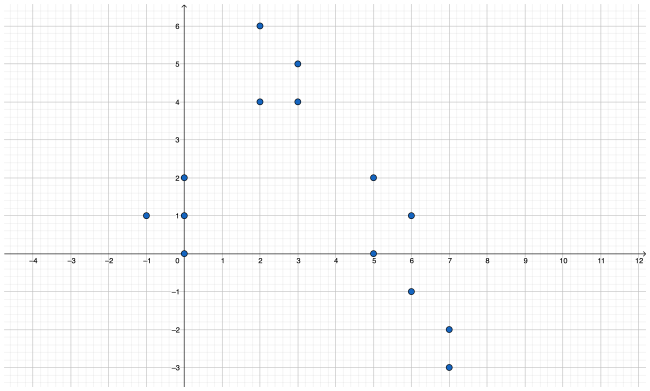
Initialize k cluster centers $\{\mathbf{c}^{(1)}, ..., \mathbf{c}^{(k)}\}$ where $\mathbf{c}^{(i)} \in \mathbb{R}^d$

Repeat until convergence:

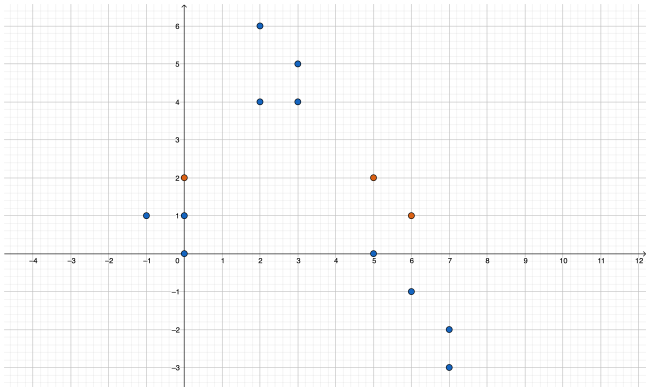
1. Assign each point $\mathbf{x}^{(i)}$ to a cluster $\mathcal{C}^{(j)}$ where $j = \arg\min_{1 \leq r \leq k} ||\mathbf{x}^{(i)} - \mathbf{c}^{(r)}||$
2. Recompute each $\mathbf{c}^{(i)}$ as the mean of points in $\mathcal{C}^{(i)}$

2.1 Walking through an example

Lets walk through an example of KMeans with $k = 3$ using the following dataset for the first iteration:



Let the cluster centers be initialized to $\mathbf{c}^{(1)} = (0, 2)$, $\mathbf{c}^{(2)} = (5, 2)$, $\mathbf{c}^{(3)} = (6, 1)$ as depicted below in the orange:

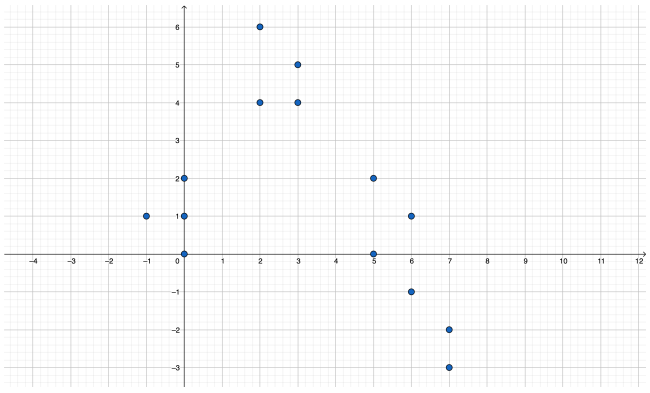


Perform one iteration of the KMeans algorithm:

1. What are the cluster assignments?

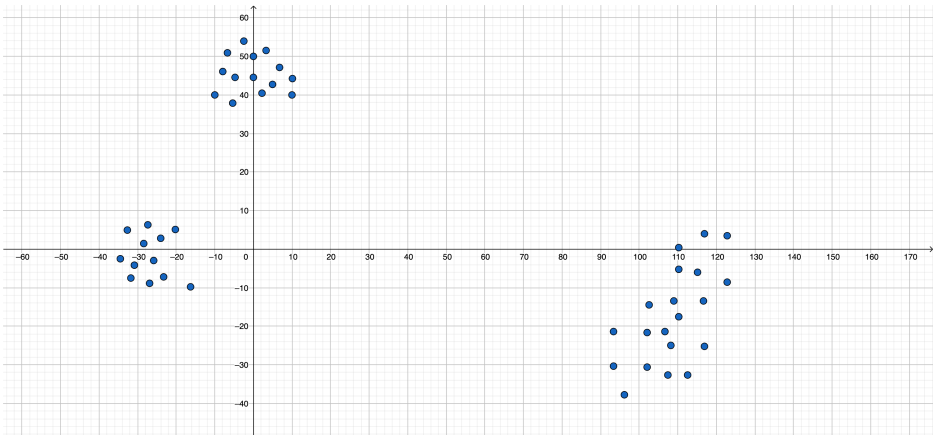
2. What are the recomputed cluster centers?

3. Draw the cluster assignments after the first iteration on the graph below.



2.2 The importance of initialization

Given the points in the graph below, and assume we will have $k = 3$ cluster centers.



1. Given an example of a set of initialization points such that the KMeans algorithm would converge to a global minimum.

2. Given an example of a set of initialization points such that the KMeans algorithm would converge to a local minimum instead of the global minimum.

3 Ensemble Methods

3.1 AdaBoost

3.1.1 AdaBoost Definitions

- T: The number of iterations used to train AdaBoost.
- N: The number of training samples.
- $S = \{(x^{(1)}, y^{(1)}), \cdots, (x^{(N)}, y^{(N)})\}$: The training samples with binary labels ($y^{(i)} \in \{-1, +1\}$).
- d: The VC-dimension of the weak learner hypothesis class.
- $\mathcal{D}_t(i)$: The weight assigned to training example i at time t. Note that $\sum_i \mathcal{D}_t(i) = 1$.
- h_t: The weak learner constructed at time t.
- ϵ_t: The error of h_t on $\mathcal{D}_t$.
- $Z_t = 2\sqrt{\epsilon_t(1-\epsilon_t)}$: The normalization factor for the distribution update at time t.
- $\alpha_t = \frac{1}{2}\ln((1-\epsilon_t)/\epsilon_t)$: The weight assigned to the learner h_t in the composite hypothesis.
- $f_t(x) = \left(\sum_{t'=1}^{t} \alpha_{t'} h_{t'}(x)\right) / \left(\sum_{t'=1}^{t} \alpha_{t'}\right)$: The majority vote of the weak learners, rescaled based on the total weights.
- $H_t(x) = \text{sign}(f_t(x))$: The voting classifier decision function.
- $\hat{\epsilon}_S(H)$: The training error of classifier H.
- $\epsilon(H)$: The generalization error of classifier H.

3.1.2 AdaBoost Weighting

AdaBoost relies on building an ensemble of weak learners, assigning them weights based on their errors during training.

1. Assume we are in the binary classification setting. What happens to the weight $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$ of classifier h_t if its error $\epsilon_t > 0.5$? Why is this useful?

   Note that if we can find weak learners h_t with $\epsilon_t < 0.5$ for all t, training error will decrease exponentially fast in the total number of iterations T.

3.1.3 The Margin

In the following question, we will examine the generalization error of AdaBoost using a concept known as the *classification margin*.

For a binary classification task, assume that we use a probabilistic classifier that provides a probability distribution over the possible labels (i.e. $p(y|x)$ for $y \in \{+1, -1\}$). The classifier output is the label with highest probability. We define the *classification margin* for an input as the signed difference between the probability assigned to the correct label and the incorrect label $p_{correct} - p_{incorrect}$, which takes on values in the range $[-1, 1]$.

1. Let $\text{margin}_t(x, y)$ represent the margin for our AdaBoost classifier at iteration t on the sample (x, y). Write a single inequality in terms of $\text{margin}_t(x, y)$ that is true if and only if the classifier makes a mistake on the input (x, y) (i.e., provide a bound on the margin in the case the classifier is incorrect). Assume the classifier makes a mistake on ties.

2. For a given input and label $(x^{(i)}, y^{(i)})$, write $\text{margin}_t(x^{(i)}, y^{(i)})$ in terms of $x^{(i)}, y^{(i)}$, and f_t.

3.1.4 Weak Learners

We always talk about AdaBoost with "weak" learners, why can't we ensemble together "stronger" learners? Let's take a look at bounds on the test error of AdaBoost, fixing the number of samples N and number of training iterations T, but allowing variation in the hypothesis class of learners $\mathcal{H}$.

$$\text{Bound 1 (PAC Learning)} : \epsilon\left(H_T\right) \leq \hat{\epsilon}_S\left(H_T\right) + O\left(\sqrt{T \log T}\sqrt{d}\sqrt{\frac{\log N}{N}}\right)$$

$$\text{Bound 2 (Margin Analysis)} : \epsilon\left(H_T\right) \leq \hat{\Pr}_{(x^{(i)},y^{(i)})\sim S}\left[\text{margin}_T(x^{(i)}, y^{(i)}) \leq \theta\right] + O\left(\frac{1}{\theta}\sqrt{d}\sqrt{\frac{\log^2 N}{N}}\right)$$

Specifically, consider d, the VC dimension of the weak learners used.

1. What happens to our bounds on true error if we increase the VC dimension of the weak learner hypothesis space?

2. Intuitively, what happens to the complexity of the overall classifier if we use more complex weak learners?

3. What concept does this connection between classifier complexity and error relate to?

3.2 The Weighted Majority Algorithm

Consider the Weighted Majority Algorithm. In this setting, we are given a fixed set of available weak classifiers, and our goal is to create a single strong classifier that incorporates information from all of the weak classifiers, using a single pass over the input.

In lecture, we discussed upper bounds on the mistakes made by the Weighted Majority Algorithm. Today in recitation, we'll discuss lower bounds on the mistakes made by any deterministic prediction algorithm operating in this setting, where we have a fixed set of classifiers given to us.

Consider the case where we have exactly two available weak classifiers: one that always outputs $+1$ and another that always outputs -1.

1. What is the highest possible number of mistakes our weighted majority classifier may make over an input of length n?

2. How do we pick an input label sequence that achieves this accuracy?

3. In this setting, the best weak classifier makes at most how many mistakes?

4. Assume the number of mistakes equals to n, what is the resulting lower bound on the number of mistakes our weighted majority classifier makes, in terms of the number of mistakes m made by the best weak classifier?

4 Recommender Systems

We generally divide recommender systems into two types. What are those types?

4.1 Collaborative Filtering

Collaborative filtering recommends items to users based on other similar users' preferences, meaning that it depends on the ratings to an item from other users. We have covered two collaborative filtering methods in the lecture:

- Neighborhood Methods
- Matrix Factorization

4.1.1 Neighborhood Methods

Different from the k nearest neighbor method, the neighborhood methods in collaborative filtering extract a neighborhood given the user data (the items you have experienced) and recommend the items preferred by this neighborhood to the user. Specifically, the step-by-step approach is:

1. Observe the items the target user has experienced
2. Find the other user or users who have experienced the most of those items
3. Recommend the set of items not experienced by the target user that have been experienced by the largest number of these other users

Let's assume for each user, we can construct a following vector:

$$U_{items} = \{u_1, u_2, \cdots, u_k\}$$

where u_i term in the vector shows:

$$\begin{cases} 1 \text{ if the user has viewed this item} \\ 0 \text{ if the user has not viewed this item} \end{cases}$$

Is the closest neighbor by Manhattan or Euclidean distance of U_{items} vectors always in the neighborhood we use for recommendations?

4.1.2 Matrix Factorization

1. When doing PCA, given a dataset X, we are able to perform SVD to find the eigenvectors and eigenvalues of the covariance matrix $\frac{1}{N}X^TX$. If dealing with a user/item matrix $R \in \mathbb{R}^{n\times m}$, can we also use SVD to find the matrix decomposition R? If yes, write out the formula for the decomposition; if no, explain why not.

2. Suppose we have n users and m items, with a partially observed ratings matrix $R \in \mathbb{R}^{n\times m}$ For Matrix Factorization, our goal is to find user matrix $U \in \mathbb{R}^{n\times d}$ and item matrix $V \in \mathbb{R}^{m\times d}$ such that $\hat{R} = UV^T$ accurately reconstructs our observed ratings and predicts our unobserved ratings. Our reconstruction of the rating of item j by user i is given by

$$\hat{r_{ij}} = \vec{u_i}^T\vec{v_j}$$

, where $\vec{u_i} \in \mathbb{R}^d$ is our learned user vector and $\vec{v_j} \in \mathbb{R}^d$ is our learned item vector.

Let $\mathbb{I} = \{(i,j) : r_{ij} \text{ is observed}\}$. Describe the objective function $J(U,V)$, using a squared error loss.

3. We can learn our decomposition by using an optimization method like SGD. Describe below the steps applying SGD.

4. What are the gradients $\nabla_{\vec{u}_i} J_{ij}(\mathbf{U},\mathbf{V})$ and $\nabla_{\vec{v}_j} J_{ij}(\mathbf{U},\mathbf{V})$ for our gradient update?

4.1.3 Alternating Least Squares for Matrix Factorization

Because both U and V are unknowns, our objective function is non-convex and hard to optimize. However, if we fix one of the unknowns, the optimization problem becomes quadratic and can be directly solved. ALS rotates between fixing the U to optimize V and fixing V to optimize U. This algorithm is called **Block Coordinate Descent**.

If we fix one of the unknowns, what known problem (with a closed-form solution) does this reduce to?

Write the block coordinate descent pseudocode for ALS.

Now, let's look at the interpretation of our user and item vectors. Note that both types of vector inhabit the shared coordinate space $\mathbb{R}^d$, and that we compute similarity with a dot product. This allows us to interpret both user vectors and item vectors as representations in a shared lower-dimensional space.

4.2 Content-Based Filtering

Content-based filtering recommends items to users based on information about the content of an item. For example, suppose we are trying to recommend movies to users. What are some examples of content information we could use?

Suppose we are trying to recommend movies to a user. We are given a feature vector for each movie with content information, and for movies the user has watched, we are given labels for whether or not they liked the movie. What are some ways we could provide recommendations?

What is one advantage of content-based filtering over collaborative filtering?

What is one advantage of collaborative filtering over content-based filtering?