HW9 Recitation
Learning Paradigms

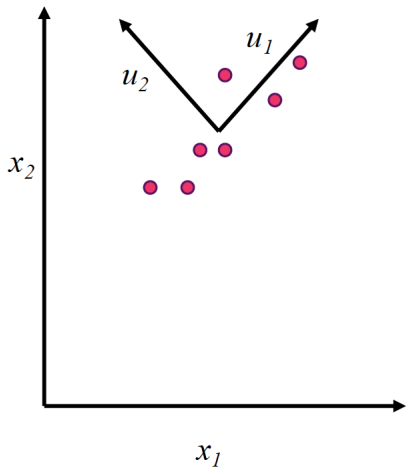
10-301/10-601: Introduction to Machine Learning

12/1/2025

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.



In low dimensions, finding the principal components can be done visually as seen above, but in higher dimensions we need to approach the problem mathematically. We find orthogonal unit vectors $\mathbf{u}_1 \dots \mathbf{u}_M$ such that the reconstruction error $\frac{1}{N}\sum_{i=1}^{N} ||\mathbf{x}^{(i)} - \hat{\mathbf{x}}^{(i)}||^2$ is minimized, where $\hat{\mathbf{x}}^{(i)} = \sum_{m=1}^{M} (\mathbf{u}_m^T \mathbf{x}^{(i)})\mathbf{u}_m$ are the reconstructed vectors.

If we have M new vectors and d original vectors, with $M = d$, we can reconstruct the original data with 0 error. If $M < d$, it is usually not possible to reconstruct the original data without losing any error. In other words, all the reconstruction error comes from the $M - d$ missing components. This error can be expressed in terms of the covariance matrix of the original data, and is minimized when the principal component vectors $\mathbf{u}_1 \dots \mathbf{u}_M$ 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 the reconstruction error. 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 **K-Means**.

In K-Means, we aim to minimize the objective function:

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

Below is the K-Means algorithm:

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.

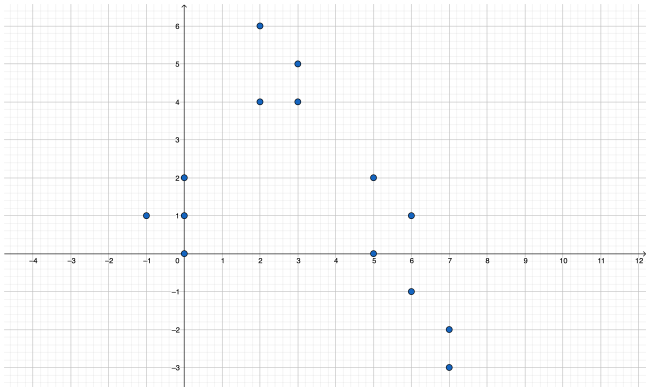
Initialize k cluster centers $\{\mathbf{c}^{(1)}, ..., \mathbf{c}^{(k)}\}$ where $\mathbf{c}^{(i)} \in \mathbb{R}^d$

Repeat until convergence:

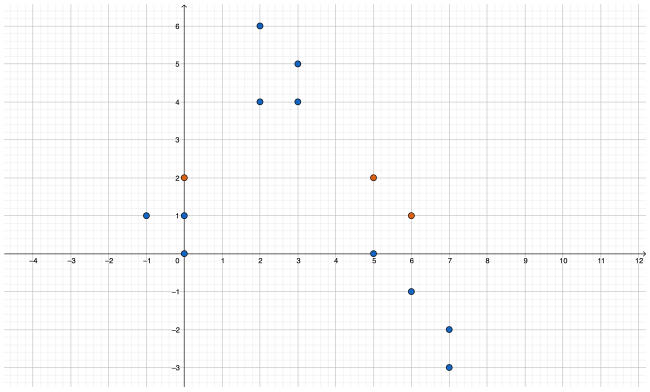
1. Assign each point $\mathbf{x}^{(i)}$ to a cluster $\mathcal{C}^{(j)}$ where $j = \arg\min_{1 \leq r \leq k} ||\mathbf{x}^{(i)} - \mathbf{c}^{(r)}||$
2. Recompute each $\mathbf{c}^{(i)}$ as the mean of points in $\mathcal{C}^{(i)}$

2.1 Walking through an example

Lets walk through an example of K-Means 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:

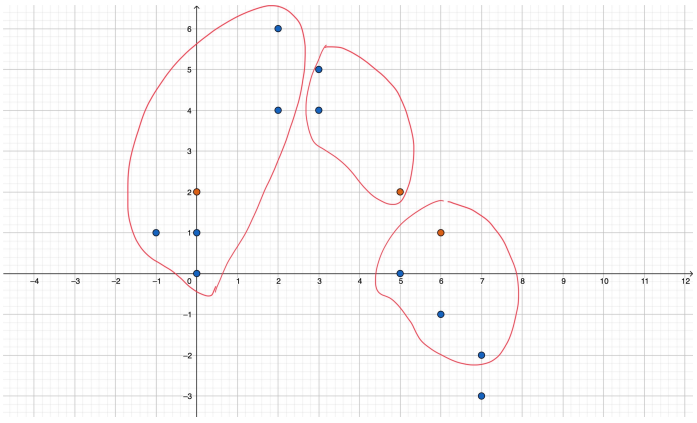


Perform one iteration of the K-Means algorithm:

1. What are the cluster assignments?

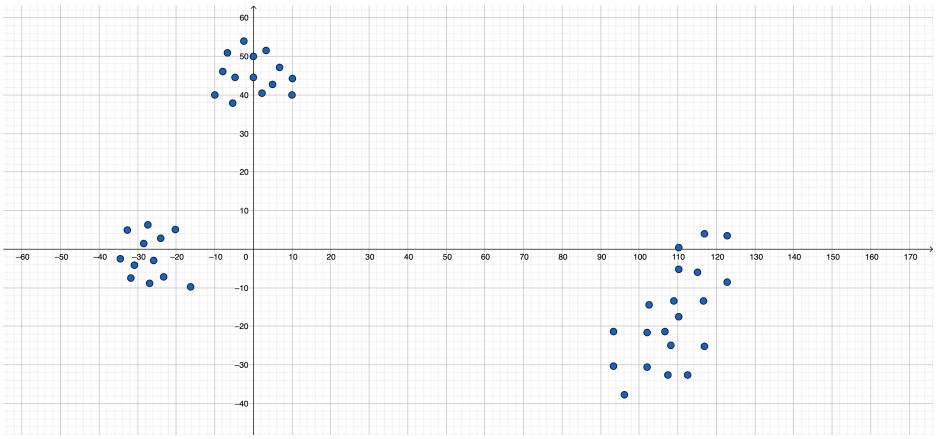
2. What are the recomputed cluster centers?

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. Give an example of a set of initialization points such that the K-Means algorithm would converge to a global minimum.

2. Give an example of a set of initialization points such that the K-Means algorithm would converge to a local minimum instead of the global minimum.

3 Ensemble Methods

The idea of ensemble methods is to build a model for prediction by combining the strengths of a group of simpler models. We'll cover two examples of ensemble methods: random forests and AdaBoost.

3.1 Random Forests

1. For each data point $\mathbf{x}^{(i)}$, define $t^{(-i)}$ to be the set of decision trees that $\mathbf{x}^{(i)}$ was not used to train. Use each tree in $t^{(-i)}$ to make a prediction for $\mathbf{x}^{(i)}$, and use these predictions to make an aggregated prediction $\overline{t^{(-i)}}(\mathbf{x}^{(i)})$ (i.e. for classification take the majority vote). Then, we can define the *out-of-bag* error as follows:

$$E_{OOB} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}\left(\overline{t^{(-i)}}(\mathbf{x}^{(i)}) \neq y^{(i)}\right)$$

   Why can we use E_{OOB} for hyperparameter optimization even though it was calculated using training points we used to learn the decision trees with?

2. **Random Forest Example:** Suppose we train a random forest with two decision trees on the following dataset, using the provided bootstrap samples. Assume that for ties, we predict $Y = 1$.

All	X_0	X_1	X_2	X_3	Y
1	1	0	0	0	1
2	0	0	1	0	1
3	0	0	0	1	1
4	0	0	0	0	0
5	0	1	0	1	1

Sample 1	X_0	X_1	X_2	X_3	Y
1	1	0	0	0	1
4	0	0	0	0	0
5	0	1	0	1	1

Sample 2	X_0	X_1	X_2	X_3	Y
3	0	0	0	1	1
4	0	0	0	0	0
5	0	1	0	1	1

(a) Suppose we train our first tree on Sample 1 and the split feature randomization chooses $\{X_1, X_2\}$ for the feature candidates at the root. What feature will we split on at the root?

(b) Suppose we then recurse on the left child (with feature value 0) of the root and split feature randomization chooses $\{X_0, X_2\}$ for the feature indices. What feature will we split on?

(c) Suppose we train our second tree on Sample 2 and the split feature randomization chooses $\{X_2, X_3\}$ for the feature candidates at the root. What feature will we split on at the root?

(d) What is the training error of the ensemble?

(e) What is the out of bag error of the ensemble?

3.2 AdaBoost

3.2.1 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.

2. AdaBoost also assigns weights $\mathcal{D}_t(i)$ for each data point. Explain in broad terms how the weights assigned to examples get updated in each iteration.

AdaBoost

Algorithm 1 AdaBoost Algorithm

1: Given: $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$ where $\mathbf{x}_i \in \mathbb{R}^M, y_i \in \{-1, +1\}$
2: Initialize $D_1(i) = \frac{1}{N}$
3: **for** $t = 1, \ldots, T$ **do**
4: Train weak learner using distribution D_t.
5: Get weak hypothesis $h_t : \mathbb{R}^M \to \{-1, +1\}$ with error

$$\epsilon_t = P_{i \sim D_t}[h_t(\mathbf{x}_i) \neq y_i]$$

6: Choose $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$. for high error, we get _____ α_t
for low error, we get _____ α_t
7: **for** $i = 1, \ldots, N$ **do**
8: Update:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(\mathbf{x}_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(\mathbf{x}_i) \neq y_i \end{cases}$$

if correct, _____
if incorrect, _____
↳ for high error, _____
↳ for low error, _____

$$= \frac{D_t(i)\exp(-\alpha_t y_i h_t(\mathbf{x}_i))}{Z_t}$$

where normalization const. Z_t chosen s.t. D_{t+1} is a distribution.

9: Output the final hypothesis: $H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right)$.

Algorithm adapted from (Freund & Schapire, 1999)

3.2.2 Weak Learners

We always talk using 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 $\mathcal{H}$.

Let d be the VC-dimension of the hypothesis class. Consider the following bounds with respect to d:

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

$$\text{Bound 2 (Margin Analysis)}: \quad \epsilon(H_T) \leq \hat{P}_S\left[\text{margin}_T \leq \theta\right] + O\left(\frac{1}{\theta}\sqrt{d}\sqrt{\frac{\log^2 N}{N}}\right)$$

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

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

3.3 Implementation

You are tasked with performing a three-step AdaBoost process on a dataset of eight points, starting with equal weights $D_1(i) = \frac{1}{8}$ The linear decision boundaries for the weak classifiers at each step have already been provided to you, so your goal is to implement the calculations based on the lecture notes from class.

1. **Iterations**

   - Use the given decision boundary to calculate the error rate ϵ_t and weak classifier weight α_t.

$$\epsilon_t = \sum_{i=1}^{N} D_t(i) \cdot \mathbb{1}[h_t(x_i) \neq y_i], \alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

   - Update the weights $D_t(i)$.

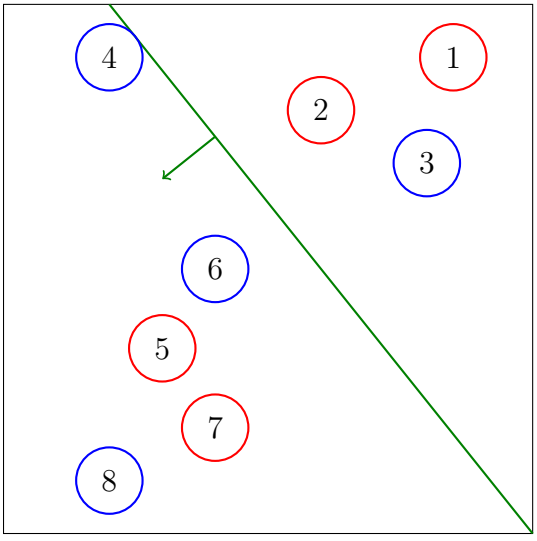
$$D_{t+1}(i) = \frac{D_t(i) \cdot e^{-\alpha_t y_i h_t(x_i)}}{Z_t}, \quad Z_t = \sum_{i=1}^{N} D_t(i) \cdot e^{-\alpha_t y_i h_t(x_i)}$$

2. **Classification**

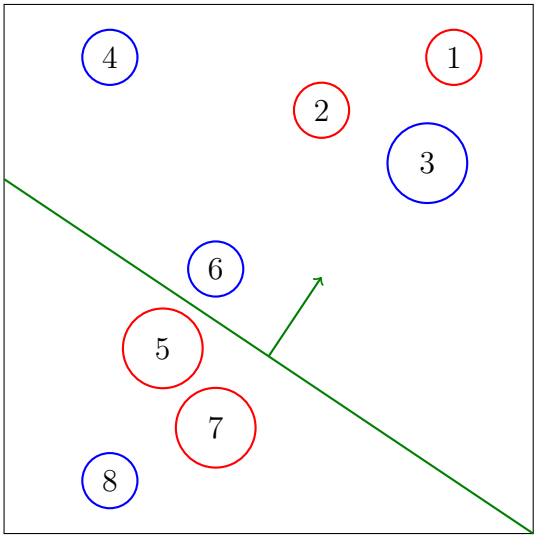
   - Shade the area that you will classify as positive based on your previous calculation and formula:

$$H_{\text{final}} = \text{sign}\left(\alpha_2 h_2(x) + \alpha_3 h_3(x) + \alpha_4 h_4(x)\right)$$

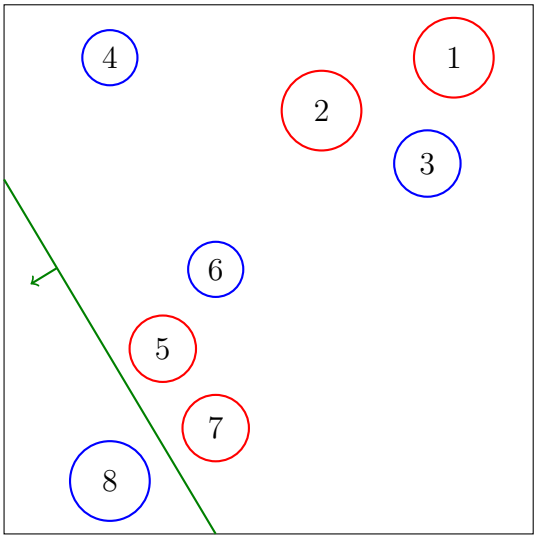
3.3.1 First Iteration



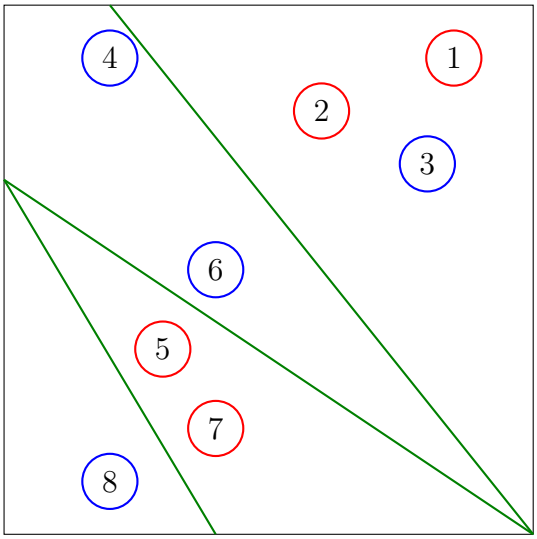
3.3.2 Second Iteration

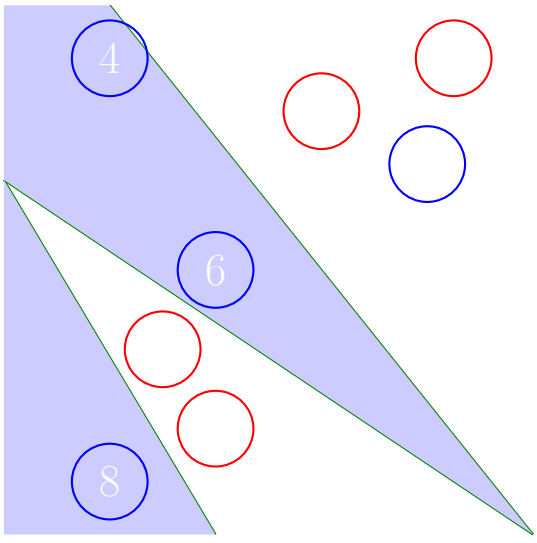


3.3.3 Third Iteration (Only Calculate ϵ, α)



3.3.4 Final Prediction (Shade)





i	1	2	3	4	5	6	7	8
h_1	-1	-1	-1	1	1	1	1	1
h_2	1	1	1	1	-1	1	-1	-1
h_3	-1	-1	-1	-1	-1	-1	-1	1
H	-1	-1	-1	1	-1	1	-1	1

Table 1: Prediction Summary

4 Recommender Systems

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 types of collaborative filtering methods in the lecture:

- Neighborhood Methods
- Latent Factor Methods (e.g., Matrix Factorization)

4.1.1 Neighborhood Methods

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. 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:

$$U_{items} = \{u_1, u_2, \cdots, u_k\}, u_i = \begin{cases} 1 & \text{if the user has viewed item } i \\ 0 & \text{if the user has not viewed item } i \end{cases}$$

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 into user matrix $U \in \mathbb{R}^{n\times d}$ and item matrix $V \in \mathbb{R}^{m\times d}$? If yes, write out the formula for the decomposition; if no, explain why not.

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**.

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

2. 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

1. Suppose we are trying to recommend movies to a user. We are given a feature vector for each movie with content information such as year of release and genre, and for movies the user has watched, we are given labels for whether or not they liked the movie. What learning paradigm is suited for our recommendation task?

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

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