HW9 Recitation
Learning Paradigms

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

2021-11-29

1 Ensemble Methods

1.1 Weighted Majority Algorithm

The first setting we discussed for ensemble algorithms was that of 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? n

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

   Since the algorithm is deterministic, we can exactly model its behavior on any partial sequence of input we have constructed so far. We can then keep picking the next input's label to be the opposite of what the classifier would output.

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

$n/2$ There must be at most $n/2$ of one of positive and negative labels.

4. 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?

Our classifier makes at least $2m$ mistakes.

The above example illustrates the weakness of building ensemble classifiers in the setting we specified. One possible solution to this weakness is introducing randomization into the classifier output. Another possible solution is to relax our conditions: we allow the learning of weak classifiers from data, and we allow ourselves to iterate over the data as many times as we need. This creates the setting of AdaBoost.

1.2 AdaBoost

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

1. In the binary classification setting, what condition do we want on the error ϵ_t of weak learner h_t? We want $\epsilon_t < 1/2$.

2. What happens to the weight $\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$ of classifier h_t if this condition is not met? It becomes negative (check the log term).

Given the above condition on all weak learners, the bound on training error decreases exponentially fast in the number of iterations T.

We always talk about AdaBoost with "weak" learners, why can't we ensemble together "strong" learners? Let's take another look at the bounds from the homework assignment.

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

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

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? The bounds loosen/increase.

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

   It becomes a more complex function, as it is a sum of more complex functions.

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

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:

$$\sum_{i=1}^{n} \min_{j \in \{1,...,k\}} ||x_i - c_j||^2 \tag{1}$$

Recall the KMeans algorithm (Lloyd's algorithm) from class:

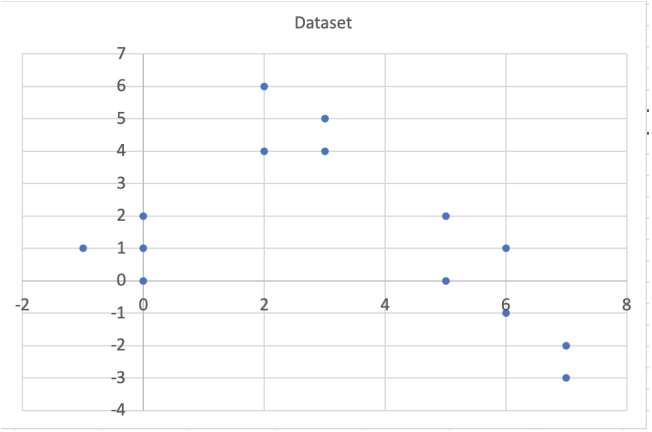
Let $\mathcal{D} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(n)}\}$ where $\mathbf{x}^{(i)} \in \mathbb{R}^d$ be the set of input examples that each have d features.

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

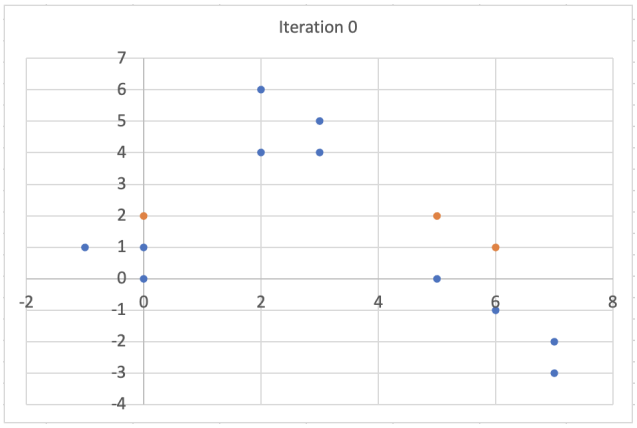
Repeat until convergence:

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 $\mathcal{C}^{(i)}$

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



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?

   $\mathcal{C}^{(1)} = \{(0,0), (-1,1), (0,1), (0,2), (2,4), (2,6)\}$

   $\mathcal{C}^{(2)} = \{(3,4), (3,5), (5,2)\}$

   $\mathcal{C}^{(3)} = \{(5,0), (6,1), (6,-1), (7,-2), (7,-3)\}$

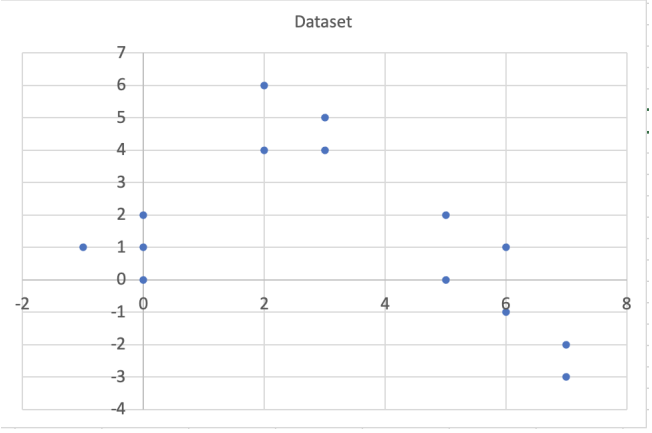
2. What are the recomputed cluster centers?

   $\mathbf{c}^{(1)} = (0.5, 2.33)$

   $\mathbf{c}^{(2)} = (3.67, 3.67)$

   $\mathbf{c}^{(3)} = (6.2, -1)$

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



Perform another iteration of the KMeans algorithm:

1. What are the cluster assignments?

   $\mathcal{C}^{(1)} = \{(0,0), (-1,1), (0,1), (0,2)\}$

$\mathcal{C}^{(2)} = \{(2,4),(2,6),(3,4),(3,5),(5,2)\}$

$\mathcal{C}^{(3)} = \{(5,0),(6,1),(6,-1),(7,-2),(7,-3)\}$

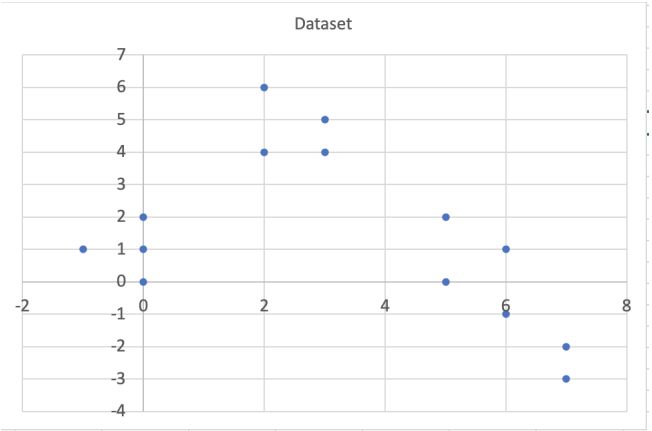
2. What are the recomputed cluster centers?

   $\mathbf{c}^{(1)} = (-0.25, 1)$

   $\mathbf{c}^{(2)} = (3, 4.2)$

   $\mathbf{c}^{(3)} = (6.2, -1)$

3. Draw the cluster assignments after the second iteration on the graph below?



Say you have performed the KMeans algorithm until convergence:

1. What are the final cluster centers?

   $\mathbf{c}^{(1)} = (-0.25, 1)$

   $\mathbf{c}^{(2)} = (2.5, 4.75)$

   $\mathbf{c}^{(3)} = (6, -0.67)$

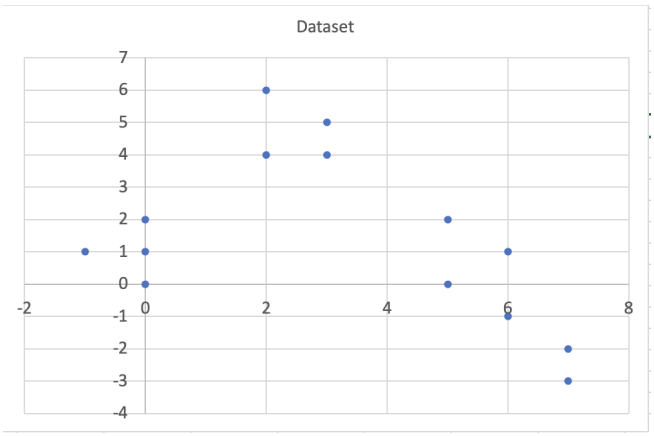
2. What are final cluster assignments?

   $\mathcal{C}^{(1)} = \{(0,0),(-1,1),(0,1),(0,2)\}$

   $\mathcal{C}^{(2)} = \{(2,4),(2,6),(3,4),(3,5)\}$

   $\mathcal{C}^{(3)} = \{(5,0),(5,2),(6,1),(6,-1),(7,-2),(7,-3)\}$

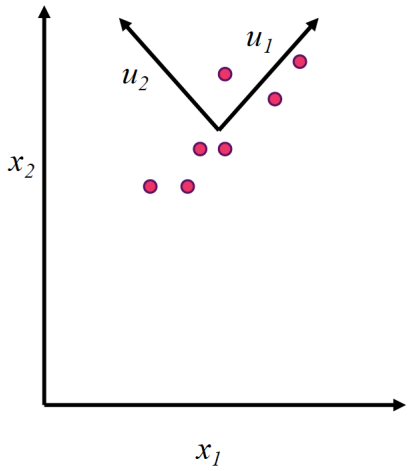
3. Draw the final cluster assignments on the graph below?


Dataset
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-2
0
2
4
6
8

3 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 projection to the data that minimizes the squared error in reconstructing original data. In other words, find a new coordinate system.



Finding the vectors is quite easy visually as seen above, but how do we do this mathematically? We find the orthogonal vectors $u_1 \dots u_M$ such that it minimizes the sum of the errors of $||x^n - \hat{x}^n||^2$, where x^n is the given data point and $\hat{x}^n$ is the new vector (new dimension).

PCA: given M<d. Find $\langle \mathbf{u}_1 \dots \mathbf{u}_M \rangle$

that minimizes $E_M \equiv \sum_{n=1}^{N} ||\mathbf{x}^n - \hat{\mathbf{x}}^n||^2$

where $\hat{\mathbf{x}}^n = \bar{\mathbf{x}} + \sum_{i=1}^{M} z_i^n \mathbf{u}_i$

Mean

$\bar{\mathbf{x}} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}^n$

If we have M new vectors, and d original vectors, where $M < d$, it is 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 each vector $u_1 \dots u_M$ are eigenvectors of the covariance matrix. The higher the eigenvalues are, the more information it stores (less error).

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

Row	X1	X2	X3	X4
1	0.48	0.29	0.35	0.30
2	0.90	0.75	0.45	0.16
3	0.21	0.04	0.83	0.89
4	0.45	0.45	0.59	0.5
5	0.98	0.78	0.42	0.99
6	0.84	0.71	0.02	0.99

And we've obtained the following principal components:

PC1	PC2	PC3	PC4
-0.71	0.29	-0.06	0.63
0.70	0.30	-0.09	0.64
-0.07	0.91	-0.02	-0.42
0.03	0.06	0.99	0.1

Which correspond to the following eigenvalues:

[0.004, 0.052, 0.194, 0.374]

1. Why are there only 4 principal components?

   There are 4 principal components because there are 4 features, and each feature captures variance in a different direction. The maximum variance we could capture would be all the variance in all 4 features. Assuming none of these features are perfectly correlated, that leaves us with 4 possible directions of variance, or 4 principal components.

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

   (0.374 + 0.194) / (0.374 + 0.194 + 0.052 + 0.004) = 0.568 / 0.624 = 0.9102 * 100 = 91% of the variance.

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

(0.374 + 0.052) / (0.374 + 0.194 + 0.052 + 0.004) = 0.426 / 0.624 = 0.6827 * 100 = 68% of the variance.

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

The PCA'd dataset is:

$$\begin{bmatrix} -0.01 & 0.58 \\ 0.03 & 1.03 \\ 0.01 & 0.96 \\ 0.17 & 0.92 \\ 0.01 & 1.10 \\ 0.08 & 0.57 \end{bmatrix}$$

Projected back up to 4 dimensions, we get:

$$\begin{bmatrix} 0.18 & 0.17 & 0.53 & 0.04 \\ 0.28 & 0.33 & 0.93 & 0.07 \\ 0.27 & 0.30 & 0.87 & 0.06 \\ 0.14 & 0.40 & 0.83 & 0.06 \\ 0.31 & 0.34 & 1.00 & 0.07 \\ 0.11 & 0.23 & 0.52 & 0.04 \end{bmatrix}$$

Reconstruction error is 6.2579.

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

The new dataset is:

$$\begin{bmatrix} -0.01 & 0.11 \\ 0.03 & -0.16 \\ 0.01 & 0.86 \\ 0.17 & 0.32 \\ 0.01 & 0.82 \\ 0.08 & 0.85 \end{bmatrix}$$

Projected back up to 4 dimensions, we get:

$$\begin{bmatrix} 0.00 & -0.02 & -0.00 & 0.10 \\ -0.01 & 0.04 & 0.00 & -0.16 \\ -0.06 & -0.07 & -0.01 & 0.85 \\ -0.14 & 0.09 & -0.01 & 0.32 \\ -0.06 & -0.08 & -0.01 & 0.82 \\ -0.11 & -0.03 & -0.01 & 0.85 \end{bmatrix}$$

Reconstruction error is 8.6919.

4 Recommender Systems

There are two types of recommender systems covered in the lecture:

Content Based Filtering recommends items to users based on side information about an item. For example, the side information about a song could be:

- Styles (rock, pop, etc.)
- Years (80s, 90s, 00s, 10s, recent, etc.)
- Singers (Taylor Swift, Michael Jackson, etc.)

Based on your own preference, new items will be recommended based on the similarity in terms of the side information.

Let's assume below is a record of years and genres of the songs you have listened:

	pop	funk	country
90s	25	33	12
00s	18	26	9
10s	10	19	10

By the similarity based on the count, What are the most likely years and genres of the songs recommended to you?

funks in 90s.

Collaborative Filtering recommends items to users based on other similar users' preference, 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

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. Get the target user's items
2. Collect all other users who have used the same items
3. Find their preferred items which the target user hasn't used

4. Recommend these items to the target user

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:

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

Without any knowledge about other users in a fixed data set (meaning that we only know the U vector of our target user), is the closest neighbor (in terms of the Euclidean distance) in this data set always in the collaborative filtering neighborhood?

No. Because a closest neighbor of our target user can possibly miss items that our target user has viewed. This is one of the major differences between nearest neighbor method and neighborhood method.

Matrix Factorization (MF)

Collaborative filtering by MF is an efficient and effective approach.

When we perform matrix factorization, we follow the same **common recipe**:

1. Define a model
2. Define an objective function
3. Optimize with your favorite black box optimizer (e.g. SGD, Block Coordinate Descent aka. Alternating Least Squares)

Unconstrained Matrix Factorization (UMF)

Based on the common recipe, we first define a model where $\hat{r_{ij}}$ is our prediction:

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

Now, recap from the lecture whiteboard that depending on the rating matrix $\mathbf{R}$ we can have two different optimization problems:

- Fully Observed $\mathbf{R}$ (unrealistic)

$$\hat{U}, \hat{V} = \arg\min_{\mathbf{U},\mathbf{V}} J(\mathbf{U}, \mathbf{V}) \quad \text{where} \quad J(\mathbf{U}, \mathbf{V}) \triangleq \frac{1}{2}\|\mathbf{R} - \mathbf{U}\mathbf{V}^T\|_2^2$$

- Partially Observed $\mathbf{R}$ (realistic)

$$r_{ij} \triangleq \mathbf{R}_{ij} \quad \leftarrow \text{rating of item } j \text{ by user } i$$

$$\vec{u_i} \triangleq \mathbf{U}_{ij} \quad \leftarrow \text{user factor (learned feature vector)}$$
$$\vec{v_j} \triangleq \mathbf{V}_{ij} \quad \leftarrow \text{item factor (learned feature vector)}$$

Let $\mathbb{Z} = \{(i,j) : r_{ij} \text{ is observed}\}$, then our objective function J would be:

$$J(\mathbf{U},\mathbf{V}) = \sum_{(i,j)\in\mathbb{Z}} J_{ij}(\mathbf{U},\mathbf{V}) = \frac{1}{2}\sum_{(i,j)\in\mathbb{Z}} (r_{ij} - \vec{u_i}^T\vec{v_j})^2$$

Finally, we perform SGD where we step over examples of actual ratings we have for (user, item) pairs:

- sample (i,j) from $\mathbb{Z}$
- step opposite gradient of $J_{ij}(\mathbf{U},\mathbf{V})$

Alternating Least Square (ALS) for UMF

Because both u_i and v_j are unknowns, our objective function is not convex. However, if we fix one of the unknowns, the optimization problem becomes quadratic and can be solved optimally. Thus, ALS techniques rotate between fixing the u_i's and fixing the v_j's. This algorithm is called **Block Coordinate Descent**:

- while not converged:
  - for in :
    - *
  - for in :
    - *

- for i in $\{1,\ldots,N_i\}$:
  - $\mathbf{u}_{i'} \leftarrow \arg\min_{\mathbf{u}_{i'}} J(\mathbf{U},\mathbf{V})$
- for j in $\{1,\ldots N_j\}$:
  - $\mathbf{v}_{j'} \leftarrow \arg\min_{\mathbf{v}_{j'}} J(\mathbf{U},\mathbf{V})$

Compare SGD with ALS. When is ALS more favorable?

While in general SGD is easier and faster than ALS, ALS is favorable in at least two cases:

1. Systems that utilize parallelization
2. Systems centered on implicit data

Referenced from *Matrix Factorization Techniques for Recommender Systems. Koren, Bell, and Volinsky (2009)*