HW9 Recitation
Learning Paradigms

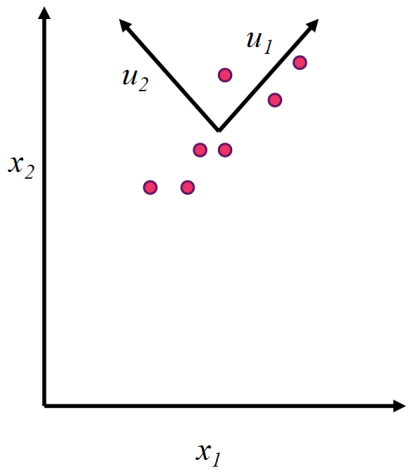
10-301/10-601: Introduction to Machine Learning

9/2/2022

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?

   There are 4 principal components because the original feature space has dimension 4. Thus, any new basis we construct can only have up to 4 independent components.

2. How much of the variance in the data is preserved by the first two principal components? (3.265 + 0.999) / (3.265 + 0.999 + 0.043 + 0.014) = 4.264 / 4.321 = 0.987 * 100 = 99% of the variance.

3. How much of the variance in the data is preserved by the first and third principal components? (3.265 + 0.043) / (3.265 + 0.999 + 0.043 + 0.014) = 3.308 / 4.321 = 0.766 * 100 = 76% of the variance.

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?

   The PCA'd dataset is:

$$\begin{bmatrix} 0.52 & -0.36 \\ -1.1 & -1.86 \\ 0.23 & 0.39 \\ -1.9 & 0.41 \\ -4.5 & -0.14 \\ -1.1 & 1.06 \end{bmatrix}$$

Projected back up to 4 dimensions, we get:

$$\begin{bmatrix} -0.36 & -0.5 & -0.06 & -0.1 \\ 0.16 & -0.77 & 1.33 & 1.51 \\ -0.03 & 0.16 & -0.28 & -0.32 \\ 1.1 & 1.21 & 0.64 & 0.83 \\ 2.36 & 2.09 & 2.02 & 2.5 \\ 0.83 & 1.28 & -0.01 & 0.07 \end{bmatrix}$$

Reconstruction error is 0.542.

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?

   The new dataset is:

$$\begin{bmatrix} 0.52 & -0.17 \\ -1.1 & -0.08 \\ 0.23 & -0.06 \\ -1.9 & -0.52 \\ -4.5 & -0.03 \\ -1.1 & 0.07 \end{bmatrix}$$

   Projected back up to 4 dimensions, we get:

$$\begin{bmatrix} -0.36 & -0.21 & -0.32 & -0.17 \\ 0.54 & 0.56 & 0.43 & 0.65 \\ -0.15 & -0.1 & -0.13 & -0.09 \\ 0.76 & 1.07 & 0.55 & 1.37 \\ 2.37 & 2.2 & 1.94 & 2.45 \\ 0.62 & 0.52 & 0.52 & 0.54 \end{bmatrix}$$

   Reconstruction error is 5.259.

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?

   Using the first and second principal components:
   Error $= (0.92 - 1.1)^2 + (1.31 - 1.21)^2 + (0.31 - 0.64)^2 + (1.19 - 0.83)^2 = 0.28$

   Using the first and third principal components:
   Error $= (0.92 - 0.76)^2 + (1.31 - 1.07)^2 + (0.31 - 0.55)^2 + (1.19 - 1.37)^2 = 0.17$

   This is because PCA minimizes the mean reconstruction error over all rows, so there may be rows/data points whose reconstruction errors are not minimized (i.e. another choice of projection might yield lower error for those points).

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 (as will be presented in class on Monday):

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.

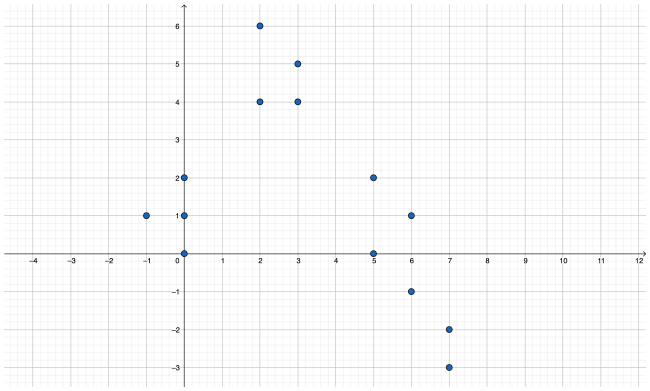
Initialize k cluster centers $\{\mathbf{c}^{(1)}, ..., \mathbf{c}^{(k)}\}$ where $\mathbf{c}^{(i)} \in \mathbb{R}^d$

Repeat until convergence:

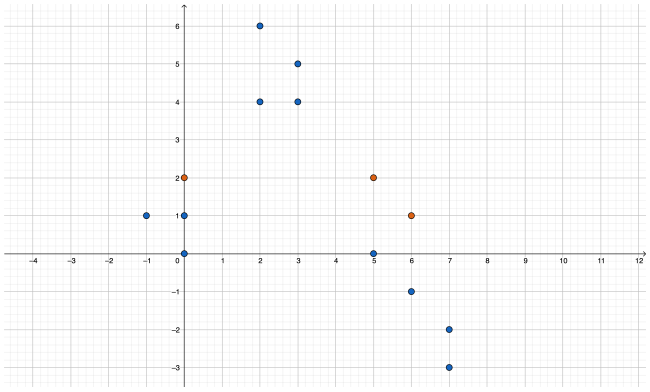
1. Assign each point $\mathbf{x}^{(i)}$ to a cluster $\mathcal{C}^{(j)}$ where $j = \arg\min_{1 \leq r \leq k} ||\mathbf{x}^{(i)} - \mathbf{c}^{(r)}||$
2. Recompute each $\mathbf{c}^{(i)}$ as the mean of points in $\mathcal{C}^{(i)}$

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

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

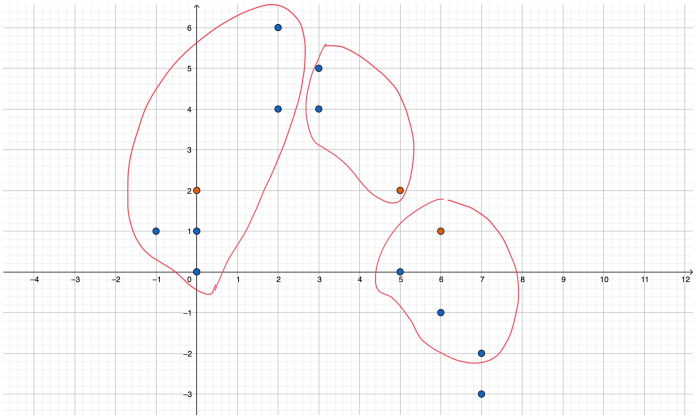
   $\mathcal{C}^{(3)} = \{(5,0), (6,1), (6,-1), (7,-2), (7,-3)\}$

2. What are the recomputed cluster centers? $\mathbf{c}^{(1)} = (0.5, 2.33)$

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

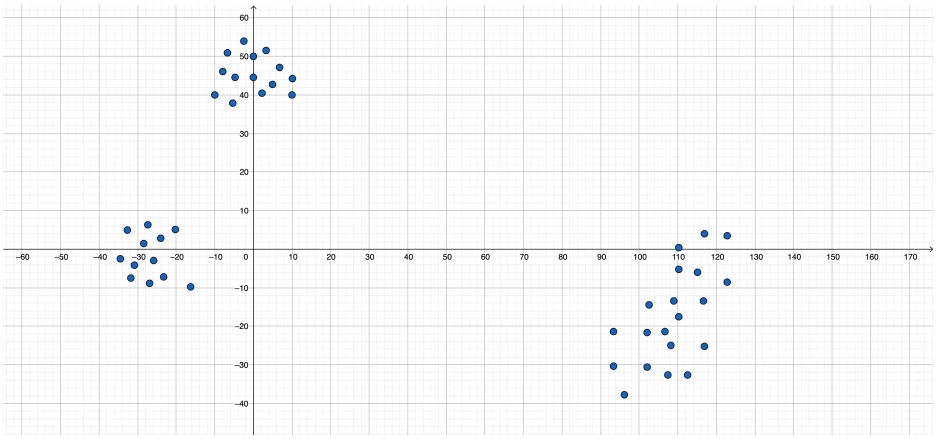
   $\mathbf{c}^{(3)} = (6.2, -1)$

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



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 K-Means algorithm would converge to a global minimum.

   Any three points where each belongs to a different cluster

2. Given 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.

   For example, one to the upper left corner, the other two at the bottom right corner

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: AdaBoost and the weighted majority algorithm.

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 ($y^{(i)} \in \{-1, +1\}$).
- $\omega_t(i)$: The weight assigned to training example i at time t. Note that $\sum_i \omega_t(i) = 1$.
- h_t: The weak learner constructed at time t (a function $X \to \{-1, +1\}$).
- ϵ_t: The weighted (by ω_t) error of h_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? It becomes negative (check the log term). This "inverts" the output of this weak learner for every input, turning it into a classifier with $\epsilon_t < 0.5$.

   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 $\omega_t(i)$ for each data point. Explain in broad terms how the weights assigned to examples get updated in each iteration.

Generally, points that get incorrectly classified get up-weighted and points that get correctly classified get down-weighted. The amount by which they're weighted depends on the importance of the weak learner - better (lower error) learners lead to stronger up-weights and down-weights. The weights are also normalized to have sum 1 by Z_t.

Update rule: $\omega_t(i) = \frac{\omega_{t-1}(i)e^{-\alpha_t y^{(i)} h_t(x^{(i)})}}{Z_t}$

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.

   $\text{margin}_t(x, y) \leq 0$.

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.

   $y^{(i)} f_t(x^{(i)})$

   INTUITION: f_t tells us the difference between the probabilities on the positive and negative labels, which is the margin in the case that the real label is positive. In the case that the real label is negative, f_t is the opposite of the margin. In both cases, this gives us the margin as $f_t(x^{(i)})$ multiplied by the true label $y^{(i)}$.

   To find $\text{margin}_t(x^{(i)}, y^{(i)})$, we must find expressions for $p_{correct}$ and $p_{incorrect}$. We will do this by first finding expressions for p_+ and p_-, the probabilities assigned to the positive and negative classes during our decision.

   First, note that each weak learner has an associated weight α_t. These weights are not normalized: the weight sum $\alpha = \sum_{t'=1}^{t} \alpha_{t'}$ is not necessarily 1.

   When we classify a point $x^{(i)}$, we use the function $f_t(x^{(i)})$. This function computes the weighted summation over the outputs of all individual weak learners and normalizes

this by the total sum of weights. Because each weak learner outputs ± 1, we have $f_t(x^{(i)}) \in [-1, 1]$.

Now, let's rewrite $f_t(x^{(i)}) = \sum_{t'=1}^{t} \frac{\alpha'_t}{\alpha} h_{t'}(x^{(i)})$ (pushing in that denominator summation α we defined earlier). We can think of $\frac{\alpha'_t}{\alpha}$ as the probability assigned to the weak learner $h_{t'}$ from time t'.

Let's look at a single element of this summation $\frac{\alpha'_t}{\alpha} h_{t'}(x^{(i)})$. If $h_{t'}$ predicts positive on $x^{(i)}$, then $h_{t'}(x^{(i)}) = 1$ and this element is $\frac{\alpha'_t}{\alpha}$. Similarly, if $h_{t'}$ predicts negative, this is $-\frac{\alpha'_t}{\alpha}$.

We can then rewrite our summation f_t as follows:

$$f_t(x^{(i)}) = \sum_{t':h_{t'}(x^{(i)})=1} \frac{\alpha_{t'}}{\alpha} - \sum_{t':h_{t'}(x^{(i)})=-1} \frac{\alpha_{t'}}{\alpha}$$

where the first summation contains all the weights of weak learners that predicted positive and the second summation contains all the weights of weak learners that predicted negative. Note that $\sum_{t'} \frac{\alpha_{t'}}{\alpha} = \frac{1}{\alpha} \sum_{t'} \alpha_{t'} = \frac{\alpha}{\alpha} = 1$, s the total weight here does sum to 1. Thus, we can now define $p_+ = \sum_{t':h_{t'}(x^{(i)})=1} \frac{\alpha_{t'}}{\alpha}$ and $p_- = \sum_{t':h_{t'}(x^{(i)})=-1} \frac{\alpha_{t'}}{\alpha}$, and $f_t(x^{(i)}) = p_+ - p_-$.

Note that if the true label $y^{(i)} = +1$, our margin is $p_{correct} - p_{incorrect} = p_+ - p_- = f_t(x^{(i)})$. If the true label is $y^{(i)} = -1$, our margin is $p_{correct} - p_{incorrect} = p_- - p_+ = -f_t(x^{(i)})$. Thus, our margin is always given by $y^{(i)} f_t(x^{(i)})$.

3.1.4 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:

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

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

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

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? 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 wrong predictions.
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?

   Our classifier makes at least $2m$ mistakes. Using question 3, $\frac{n}{2} \leq m \iff n \leq 2m$ where n equals the number of mistake we make.

   We can generalize this proof idea to arbitrary sets of classifiers. Under the deterministic version of this algorithm, there always exists an adversarial sequence of n inputs that yields n mistakes, but we can always ensure that the best

4 Recommender Systems

We generally divide recommender systems into two types. What are those types? collaborative filtering and content-based filtering

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?

No. Because we want our closest neighbor of our target user to be chosen only based on their shared experiences with our target user. We also want these neighbors to have a set of many experiences which the target user does not have, so that we can recommend unseen items. Thus, we do not want to penalize by increasing distance if the other users have experienced many things the target user has not.

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.

   We cannot use SVD to find the decomposition of the user/rating matrix, because the R matrix has missing values. SVD cannot be performed on a dataset with missing values.

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.

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

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

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

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?

   $\nabla_{\vec{u_i}} J_{ij}(\mathbf{U},\mathbf{V}) = (r_{ij} - \vec{u_i}^T\vec{v_j})\vec{v_j}$

   $\nabla_{\vec{v_j}} J_{ij}(\mathbf{U},\mathbf{V}) = (r_{ij} - \vec{u_i}^T\vec{v_j})\vec{u_i}$

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?

Least squares (from linear regression), hence the name.

Write the block coordinate descent pseudocode for ALS.

$$\mathbf{U} \leftarrow \arg\min_{\mathbf{U}} J(\mathbf{U}, \mathbf{V})$$

$$\mathbf{V} \leftarrow \arg\min_{\mathbf{V}} J(\mathbf{U}, \mathbf{V})$$

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? actors/actresses, year of release, genre, length

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? Train a supervised learning algo on the features and labels and make predictions on unseen movies

What is one advantage of content-based filtering over collaborative filtering? We don't need other users in the system at all; can start making predictions without user/item interactions. We also don't need to take and store user data, which improves data privacy. This can also be much more explainable than specifically MF for collaborative filtering, since we can choose our features instead of having the optimally computed but hard to understand lower-dimensional space.

What is one advantage of collaborative filtering over content-based filtering? It can be hard or computationally expensive to find and compute content information or train such recommender algorithms.