CSE/STAT 416
Dimensionality Reduction & Recommender Systems Intro

Amal Nanavati
University of Washington
Aug 8, 2022

Adapted from Hunter Schafer’s slides
**Administrivia**

- **This Week:** Dimensionality Reduction, Recommender Systems
- **Next Week:** Course Wrap-Up & Guest Speakers
- **Deadlines:**
  - HW6 due TOMORROW, Tues 8/9 11:59PM
    - Submit Concept on Gradescope
    - Submit Programming on EdSTEM
  - HW7 (final HW) released Wed 8/10
    - Due Tues 8/16 11:59PM, **NO LATE DAYS**
  - LR 8 due Fri 8/12 11:59PM
  - **Extra Credit** Guest Panel Mon 8/15 during lecture.
  - Take-Home Final Exam: Wed 8/17 – Thurs 8/18
Addressing LR Questions
1. “Fairness through Unawareness” 
   1. To avoid unfair decisions, prevent the model from every looking at protected attribute (e.g., race, gender). 
   2. Doesn’t work in practice 
2. Statistical Parity 
   1. Idea: Equal performance across groups. 
      $$\Pr(\hat{Y} = + | A = \blacksquare) = \Pr(\hat{Y} = + | A = \bigcirc)$$ 
   2. Also phrased as matching demographic statistics (e.g., if 33% of population are Circles, 33% of those admitted should be Circles). 
3. Equal Opportunity 
   1. Idea: True positive rate should be equal across groups 
      $$\Pr(\hat{Y} = + | A = \blacksquare, Y = +) = \Pr(\hat{Y} = + | A = \bigcirc, Y = +)$$
Statistical Parity Example
Equality of Opportunity Example

Majority

Minority

Recruited

Qualified candidates

Unqualified candidates
Unsupervised Learning

- A type of machine learning that detects underlying patterns in \textit{unlabeled} data.

- Examples of unsupervised learning tasks:
  - Cluster similar articles together.
  - Cluster gene sequences.
  - Recommend items, searches, movies, etc.

**Coupled indoor navigation for people who are blind**

A Nanavati, XZ Tan, A Steinleid - Companion of the 2018 ACM/IEEE..., 2018 - dl.acm.org

This paper presents our design of an autonomous navigation system for a mobile robot that guides people who are blind and low vision in indoor settings. It begins by presenting user ...
In many real world contexts, there aren’t clearly defined labels so we won’t be able to do classification.

We will need to come up with methods that uncover structure from the (unlabeled) input data $X$.

Clustering is an automatic process of trying to find related groups within the given dataset.

Input: $x_1, x_2, \ldots, x_n$

Output: $z_1, z_2, \ldots, z_n$
Think

2 min

- Which word(s) have the largest IDF? Which word(s) have the smallest IDF?

<table>
<thead>
<tr>
<th>Review</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Sushi was great, the food was awesome, but the service was terrible”</td>
</tr>
<tr>
<td>“Terrible food; the sushi was rancid.”</td>
</tr>
</tbody>
</table>

Note that if we divide the Bag of Words embedding by the num words in the document, we get the TF!
k-means is trying to minimize the heterogeneity objective

$$\arg\min_{z,\mu} \sum_{j=1}^{k} \sum_{i=1}^{n} 1\{z_i = j\} \left\| \mu_j - x_i \right\|^2_2$$

Step 0: Initialize cluster centers

Repeat until convergence:

Step 1: Assign each example to its closest cluster centroid

Step 2: Update the centroids to be the mean of all the points assigned to that cluster

Coordinate Descent alternates how it updates parameters to find minima. On each of iteration of Step 1 and Step 2, heterogeneity decreases or stays the same.

=> Will converge in finite time
Finding Shapes

k-means

Mixture Models

Hierarchical Clustering
Dendrogram

x-axis shows the datapoints (arranged in a very particular order)
y-axis shows distance between pairs of clusters
Choose a distance $D^*$ to “cut” the dendrogram
- Use the largest clusters with distance $< D^*$
- Usually ignore the idea of the nested clusters after cutting
Large Dimensionality

Input data might have thousands or millions of dimensions!

- **Images**: 200x200 image is 120,000 features!
- **Text**: # features = # n-grams 😊
- **Course Success**: dozen(s) of features
- **User Ratings**: 100s of ratings (one per rate-able item)
Issues with Too Many Dimensions

- **Visualization**: Hard to visualize more than 3D.
- **Overfitting**: Greater risk of overfitting with more features/dimensions.
- **Scalability**: Some ML approaches (e.g., k-nn, k-means) perform poorly in high-dimensional spaces (curse of dimensionality).
- **Redundancy**: High-dimensional data often occupies a lower-dimensional subspace.
  - Most pixels in MNIST (digit recognition) are white – are they necessary?
    - Image Compression

![Original (400-dim) vs Compressed (40-dim)](image-url)
Dimensionality Reduction is the task of representing the data with a fewer number of dimensions, while keeping meaningful relations between data.
Example: Embedding Pictures

Example: Embed high dimensional data in low dimensions to visualize the data
- Goal: Similar images should be near each other.
Example: Embedding Words
Example: Embedding Words
One very popular dimensionality reduction algorithm is called **Principal Component Analysis (PCA)**.

Idea: Use a linear projection from $d$-dimensional data to $k$-dimensional data

- E.g. 1000 dimension word vectors to 3 dimensions

Choose the projection that minimizes **reconstruction error**

- Idea: The information lost if you were to ”undo” the projection
Principal Component Analysis (PCA)

Regions with no data. Data exists close to a lower-dimensional subspace.
Linear Projection

**Project** data into 1 dimension along a line
Reconstruct original data only knowing the projection
Compute the 2D coordinates of the following point. Then compute its reconstruction error.

- $x_i = [0, -7, 3, 2, 5]$  
- $u_1 = [-0.5, 0, 0.5, -0.5, 0.5]$  
- $u_2 = [0.5, 0, 0.5, -0.5, -0.5]$  
- $z_i = ??$  
- $\hat{x}_i = ??$  
- $\|\hat{x}_i - x_i\|_2^2 = ??$
Compute the 2D coordinates of the following point. Then compute its reconstruction error.

- \( x_i = [0, -7, 3, 2, 5] \)
- \( u_1 = [-0.5, 0.5, -0.5, 0.5] \)
- \( u_2 = [0.5, 0, 0.5, -0.5, -0.5] \)
- \( z_i = ?? \)
- \( \hat{x}_i = ?? \)
- \( \|\hat{x}_i - x_i\|^2 = ?? \)
Think of PCA as giving each datapoint a new "address."

- Earlier, you could find the datapoint by going to the location \((x, y, z)\).
- Now, if you are just moving in the projection plane, you can (approximately) find the datapoint by going to the location \((u_1, u_2)\).
How do we find the best projection vector(s)?

Pick the vector(s) along which the datapoints have the most variation!
There is a quantity in linear algebra that does exactly that!

The **eigenvectors** of a $d$-dimensional dataset* are a collection of $d$ perpendicular vectors that point in the directions of greatest variation amongst the points in the dataset.

- Eigenvectors rotate the axes of the $d$ dimensional space.

* (caveat) the eigenvectors are actually associated with the **covariance matrix** of the dataset
Each eigenvector has a corresponding \textit{eigenvalue}, indicating how much the dataset varies in that direction.

Greater eigenvalue $\Rightarrow$ greater variance.

\textbf{PCA}: Take the $k$ eigenvectors with greatest eigenvalues.
Input Data: An \( n \times d \) data matrix \( X \)
- Each row is an example

1. **Center Data:** Subtract mean from each row
   \[ X_c \leftarrow X - \bar{X}[1:d] \]

2. **Compute spread/orientation:** Compute covariance matrix \( \Sigma \)
   \[ \Sigma[t, s] = \frac{1}{n} \sum_{i=1}^{n} x_{c,i}[t]x_{c,i}[s] \]

3. **Find basis for orientation:** Compute eigenvectors of \( \Sigma \)
   - Select \( k \) eigenvectors \( u_1, \ldots, u_k \) with largest eigenvalues

4. **Project Data:** Project data onto principal components
   \[ z_i[1] = u_1^T x_{c,i} = u_1[1]x_{c,i}[1] + \cdots + u_1[d]x_{c,i}[d] \]
   \[ \vdots \]
   \[ z_i[k] = u_k^T x_{c,i} = u_k[1]x_{c,i}[1] + \cdots + u_k[d]x_{c,i}[d] \]
Using principal components and the projected data, you can reconstruct the data in the original domain.

\[
\hat{x}_i[1:d] = \bar{X}[1:d] + \sum_{j=1}^{k} z_i[j] u_j
\]
Example: Eigenfaces

Apply PCA to face data

Input Data

Principal Components
Reconstructing Faces

Depending on context, it may make sense to look at either original data or projected data.

In this case, let’s see how the original data looks after using more and more principal components for reconstruction.

- Each image shows additional 8 principal components
Other times, it does make sense to look at the data in the projected space! (Usually if $k \leq 3$)
Example: Genes

Dataset of genes of Europeans (3192 people; 500,568 loci) and their country of origin, ran PCA on the data and plotted 2 principal components.
Consider the genetic data projected onto its top two PCs. How would you interpret PC1 and PC2?
Consider the genetic data projected onto its top two PCs. How would you interpret PC1 and PC2?
General Steps to Take as an ML Practitioner

Given a new dataset:

- Split into train and test sets.
- Understand the dataset:
  - Understand the feature/label types and values
  - Visualize the data: scatterplot, boxplot, PCA, clustering
- Use that intuition to decide:
  - What features to use, and what transformations to apply to them (pre-processing).
  - What model(s) to train.
- Train the models, evaluate them using a validation set or cross-validation.
- Deploy the best model.
You have $n$ users and $m$ items in your system
- Typically, $n \gg m$. E.g., Youtube: 2.6B users, 800M videos
- Based on the content, we have a way of measuring user preference.
- This data is put together into a user-item interaction matrix.

**Task**: Given a user $u_i$ or item $v_j$, predict one or more items to recommend.
Solution 0: Popularity
Solution 0: Popularity

**Simplest Approach:** Recommend whatever is popular

- Rank by global popularity (i.e., Squid Game)
Solution 0
(Popularity)
Pros / Cons

Pros:
- Easy to implement

Cons:
- No Personalization
- Feedback Loops
- Top-K recommendations might be redundant
  - e.g., when a new Harry Potter movie is released, the others may also jump into top-k popularity.
Concerned parents: if all your friends jumped into the fire would you follow them?

Machine learning algorithm:
Solution 1: Nearest User (User-User)

User-User Recommendation:

- Given a user $u_i$, compute their $k$ nearest neighbors.
- Recommend the items that are most popular amongst the nearest neighbors.
Think

What do you see as pros / cons of the nearest user approach to recommendations?

Tell me about your friends(who your neighbors are) and I will tell you who you are.
What do you see as pros / cons of the nearest user approach to recommendations?

Tell me about your friends *(who your neighbors are)* and *I will tell you who you are.*
Solution 1 (User-User) Pros / Cons

Pros:
- Personalized to the user.

Cons:
- Nearest Neighbors might be too similar
  - This approach only works if the nearest neighbors have interacted with items that the user hasn’t.
- Feedback Loop (Echo Chambers)
- Scalability
  - Must store and search through entire user-item matrix
- Cold-Start Problem
  - What do you do about new users, with no data?
Solution 2: “People Who Bought This Also Bought...”

*Item-Item*
Solution 2: “People Who Bought This Also Bought...” (Item-Item)

Item-Item Recommendation:

- Create a **co-occurrence matrix** $C \in \mathbb{R}^{m \times m}$ ($m$ is the number of items). $C_{ij} =$ # of users who bought both item $i$ and $j$.

- For item $i$, predict the top-k items that are bought together.

![Matrix Diagram]

<table>
<thead>
<tr>
<th></th>
<th>Sunglasses</th>
<th>Baby Bottle</th>
<th>Diapers</th>
<th>Swim Trunks</th>
<th>Baby Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sunglasses</td>
<td>500</td>
<td>15</td>
<td>9</td>
<td>130</td>
<td>20</td>
</tr>
<tr>
<td>Baby Bottle</td>
<td>15</td>
<td>45</td>
<td>6</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Diapers</td>
<td>9</td>
<td>6</td>
<td>30</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>Swim Trunks</td>
<td>130</td>
<td>10</td>
<td>9</td>
<td>200</td>
<td>8</td>
</tr>
<tr>
<td>Baby Formula</td>
<td>20</td>
<td>10</td>
<td>6</td>
<td>8</td>
<td>50</td>
</tr>
</tbody>
</table>
Normalizing Co-Occurrence Matrices

**Problem:** popular items drown out the rest!

**Solution:** Normalizing using Jaccard Similarity.

\[
S_{ij} = \frac{\text{# purchased } i \text{ and } j}{\text{# purchased } i \text{ or } j} = \frac{C_{ij}}{C_{ii} + C_{jj} - C_{ij}}
\]

![Jaccard Similarity Matrix Example](image-url)
What if I know the user $u$ has bought a baby bottle and formula? 

**Idea:** Take the average similarity between items they have bought

$$Score(u, diapers) = \frac{S_{diapers,baby bottle} + S_{diapers,baby formula}}{2}$$

Could also weight them differently based on recency of purchase!

Then all we need to do is find the item with the highest average score!
What do you see as pros / cons of the item-item approach to recommendations?
What do you see as pros / cons of the item-item approach to recommendations?
Solution 2
(Item-Item)
Pros / Cons

Pros:
- Personalizes to item (incorporating purchase history also personalizes to the user)

Cons:
- Can still suffer from feedback loops
  - (As can all recommender systems – but in some cases it’s worse than others)
- Scalability (must store entire item-item matrix)
- Cold-Start Problem
  - What do you do about new items, with no data?

Customers Who Bought This Item Also Bought

- Predictive Analytics For Dummies
  - Anassie Barri
  - Paperback
  - $17.72 +$Prime

- Predictive Analytics: The Power to Predict What..."nEric Sugar
  - Paperback
  - $29.99 +$Prime

- Quantifying the User Experience: Practical...
  - Jeff Souza
  - Paperback
  - $40.93 +$Prime

- Marketing Analytics: Strategic Models and...
  - Stephan Sorger
  - Paperback
  - $69.92 +$Prime

- Data Driven Marketing For Dummies
  - David Sammelroth
  - Paperback
  - $20.43 +$Prime
Solution 3: Feature-Based
What if we know what factors lead users to like an item?

Idea: Create a feature vector for each item. Learn a regression model!

<table>
<thead>
<tr>
<th>Genre</th>
<th>Year</th>
<th>Director</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>1994</td>
<td>Quentin Tarantino</td>
</tr>
<tr>
<td>Sci-Fi</td>
<td>1977</td>
<td>George Lucas</td>
</tr>
</tbody>
</table>

Define weights on these features for all users (global)

\[ w_G \in \mathbb{R}^d \]

Fit linear model
What if we know what factors lead users to like an item?

Idea: Create a feature vector for each item. Learn a regression model!

<table>
<thead>
<tr>
<th>Genre</th>
<th>Year</th>
<th>Director</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>1994</td>
<td>Quentin Tarantino</td>
<td>...</td>
</tr>
<tr>
<td>Sci-Fi</td>
<td>1977</td>
<td>George Lucas</td>
<td>...</td>
</tr>
</tbody>
</table>

Define weights on these features for all users (global)

\[ w_G \in \mathbb{R}^d \]

Fit linear model

\[ \hat{r}_{uv} = w_G^T h(v) = \sum_i w_{G,i} h_i(v) \]

\[ \hat{w}_G = \text{argmin}_w \frac{1}{\# \text{ratings}} \sum_{u,v,r_{uv} \neq ?} (w_G^T h(v) - r_{uv})^2 + \lambda ||w_G|| \]
Personalization: Option A

Add user-specific features to the feature vector!

<table>
<thead>
<tr>
<th>Genre</th>
<th>Year</th>
<th>Director</th>
<th>...</th>
<th>Gender</th>
<th>Age</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>1994</td>
<td>Quentin Tarantino</td>
<td>...</td>
<td>F</td>
<td>25</td>
<td>...</td>
</tr>
<tr>
<td>Sci-Fi</td>
<td>1977</td>
<td>George Lucas</td>
<td>...</td>
<td>M</td>
<td>42</td>
<td>...</td>
</tr>
</tbody>
</table>
Include a user-specified deviation from the global model.

\[ \hat{r}_{uv} = (\hat{w}_G + \hat{w}_u)^T h(v) \]

Start a new user at \( \hat{w}_u = 0 \), update over time.
- OLS on the residuals of the global model
- Bayesian Update (start with a probability distribution over user-specific deviations, update as you get more data)
Will feature-based recommender systems suffer from the cold start problem? Why or why not?

What about other pros/cons of feature-based?
Will feature-based recommender systems suffer from the cold start problem? Why or why not?

What about other pros/cons of feature-based?

**Collaborative information**
(The user-item interactions matrix)

**Content information**
Can be users or/and items features

**Model**
Takes user or/and items features and returns predicted interactions
Solution 3
(Feature-Based) Pros / Cons

Pros:
- No cold-start issue!
  - Even if a new user/item has no purchase history, you know features about them.
- Personalizes to the user and item.
- Scalable (only need to store weights per feature)
- Can add arbitrary features (e.g., time of day)

Cons:
- Hand-crafting features is very tedious and unscalable 😞
Recap

Dimensionality Reduction & PCA:
- Why and when it’s important
- High level intuition for PCA
- Linear Projections & Reconstruction
- Eigenvectors / Eigenvalues

Recommender Systems:
- Sol 0: Popularity
- Sol 1: Nearest User (User-User)
- Sol 2: “People who bought this also bought” (item-item)
- Sol 3: Feature-Base

Next Time (Rec System Continued):
- Sol 4: Matrix Factorization
- Sol 5: Hybrid Model
- Addressing common issues with Recommender Systems
- Evaluating Recommender Systems