Project T Final: Collaborative Filtering

CS 189/289A, Fall 2020

Maxwell Chen and Abinav Routhu

This notebook serves to introduce and explore the topic of Collaborative Filtering through mathematical formulation, along with application to the Netflix dataset.

Collaborative Filtering is a process or algorithm to filter information or patterns through the collaboration of multiple users, agents, or data sources.

We shall approach this through three paradigms:

1. K-Nearest-Neighbors (KNN)
2. Cosine Similarity
3. SVD

In [ ]:
```
# Load Packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

To begin, we will load the MovieLens Dataset. MovieLens was a research project launched by GroupLens Research at the University of Minnesota, and was one of the earliest modern projects that investigated personalized recommendations via recommender systems. We will be using their Dataset for a similar purpose: recommending a user which movie to watch based on their own interests or preferences.

Question 1: Loading the Dataset

1.1: Import `zipfile` and `from urllib.request import urlretrieve` . Use these libraries to load the MovieLens Dataset stored at http://files.grouplens.org/datasets/movielens/ml-100k.zip (http://files.grouplens.org/datasets/movielens/ml-100k.zip) -- this is the "small" Dataset containing 100,000 ratings. If you are up for it, you can also load the expanded MovieLens Dataset stored at http://files.grouplens.org/datasets/movielens/ml-latest.zip (http://files.grouplens.org/datasets/movielens/ml-latest.zip) -- this contains 27,000,000 ratings. For the purposes of this assignment, loading either Dataset will work.

Reference Material:

- DataCamp Tutorial on zipfile module (https://www.datacamp.com/community/tutorials/zip-file)
- GeeksForGeeks Tutorial on zipfile module (https://www.geeksforgeeks.org/working-zip-files-python/)
- urllib.request Documentation (https://docs.python.org/3/library/urllib.request.html)

In [ ]:
```
from urllib.request import urlretrieve
import zipfile

urlretrieve("http://files.grouplens.org/datasets/movielens/ml-100k.zip", "movielens.zip")
zip_ref = zipfile.ZipFile('movielens.zip', "r")
zip_ref.extractall()
```

1.2: We now have a raw .csv file containing our Dataset. As with many other problems involving machine learning or data mining, we must manipulate our raw data to a form that we can use.

First, investigate the structure of the zipped dataset we just downloaded. Open up each of the unzipped files on DataHub or your local machine, and describe the contents of each file:

Answer:

1.3: Use your knowledge of data cleaning and processing from the first week of 16ML to load the different .csv files into multiple Pandas DataFrames. Use the provided columns stored in `user_features`, `ratings_features`, and `movie_features`. Use appropriate naming conventions for these DataFrames, such as "movies", for example. Then combine the Dataframes into a single DataFrame, using `user_id` as a primary key.

Hint #1: When using `pd.read_csv`, you MUST use the flag `encoding='latin-1'` to properly read from the files.

Hint #2: Use `sep="|"` when reading in the csv file

In [ ]:
```
# Users
user_features = ["user_id", "age", "sex", "occupation", "zip_code"]
users = pd.read_csv(...)

# Ratings
ratings_features = ["user_id", "movie_id", "rating", "unix_timestamp"]
ratings = pd.read_csv(...)

# Movies
movie_features = ['movie_id', 'title', 'release_date', "video_release_date",
"imdb_url", "genre_unknown", "Action",
                  "Adventure", "Animation", "Children", "Comedy", "Crime", "Do
cumentary", "Drama", "Fantasy",
                  "Film-Noir", "Horror", "Musical", "Mystery", "Romance", "Sci
-Fi", "Thriller", "War", "Western"]
movies = pd.read_csv(...)

all_data = ratings.merge(movies, on='movie_id').merge(users, on='user_id')
all_data.head()
```

Question 2: Understanding and Visualizing the Dataset

2.1: Distribution of Movie Genres

Utilize the knowledge of Pandas and NumPy you picked up in Week 1 to make an appropriate plot of the distribution of movies across different genres.

Hint: Try a bar plot

In [ ]:
```

```

2.2: It is important to identify biases in our dataset that can skew our results or impact how generalizable our recommendation system is to novel users and novel movies. What might be some issues we run into by using this dataset?

Answer:

2.2: Distribution of User Ratings

As a sanity check, pick three different genres and plot the distribution of user ratings for each -- that is, plot the number of each rating from 1 to 5 received by movies belonging to each of the three genres. For example, plot the user rating for all Children movies, all Fantasy movies, and all Film-Noir movies. Note any similarities or differences, and whether these were expected.

[Hint: Try multiple histograms or bar plots]

```
In [ ]:
```

Answer:

Question 3: Memory-Based Approaches and Clustering

The first approach we will investigate is the "Cluster-Based" Approach. We first define the following matrix known as the "user-item interaction matrix", where each row represents a user's ratings and each column represent's the ratings for a movie. In other words, A_{ij} represents user i's rating of movie j.

3.1: How many unique users and unique movies are there in our dataset? Assign your answers to `num_users` and `num_movies`, respectively.

Hint: What data structure from CS 61B could be helpful here?

In [ ]:
```
...

num_users = ...
num_movies = ...
```

3.2: How do these values differ from the dimensions of our raw data matrix? What does that tell us about the nature of our data?

Answer:

3.3: Construct the User-Item Interaction Matrix described above. Call it `interaction_matrix`. Print out its dimensions and the first few rows. Confirm that the dimensions match with the number of unique users and movies you found in **3.1**.

Hint: Look into the Pandas function `df.pivot` (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.pivot.html (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.pivot.html))

In [ ]:
```
interaction_matrix = ...
```

The simplest way to make a recommendation using this matrix is by grouping together users who are most similar to one another. In this way, we create groups, or clusters, that represent users who give similar ratings. This boils down to the algorithm known as K-Nearest Neighbors (KNN), which is used to classify data into K clusters of greatest similarity. This is spiritually similar to the K-Means Clustering you saw in lecture earlier this week -- K-Means Clustering is an unsupervised learning technique that assigns the data into K clusters, while KNN looks at the K data points most similar to a certain training point in order to assign it a class or label.

For the purposes of this assignment, we will look at a pre-imlemented KNN Algorithm (https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.NearestNeighbors.html#sklearn.neighbors.NearestNeighbo
from scikit-learn (https://scikit-learn.org/), the machine learning library we have been working with over the last few weeks of 16ML. The KNN Algorithm we use here is a bit simpler than what KNN truely should be, because instead of assigning labels based on the K nearest neighbors, we will use this to simply identify the K nearest neighbors.

[Note: Later on in EECS 16B, you will revisit K-Means Clustering. This will be used in lab to classify voice commands to control your car.]

3.4: Read the linked documentation to understand how to use the KNN Algorithm from Scikit-Learn. What are the relevant functions and return values we can use?

Answer:

3.5: Run KNN on `interaction_matrix` from the earlier part of this question. For now, use k = 10.

In [ ]:
```
from sklearn.neighbors import NearestNeighbors
...
```

3.5: Let's look at the first user in our `interaction_matrix`. What other users is our user most similar to? What are the IDs of the movies these users liked?

In [ ]:
```

```

Question 4: Cosine Similarity

One accessible way to perform collaborative filtering is through a method known as "cosine similarity". It boils down to the following:

- Treat each user row or movie column as a vector
- Determine the cosine of the angle between every pair of vectors

This can be expressed with the following expression: For vectors i and j of length n, their similarity is calculated as:

$$cos(\theta) = \frac{v_i v_j}{|v_i||v_j|} = \sum_{k=1}^{n} \frac{v_{i,k} v_{j,k}}{\sqrt{\sum_{k=1}^{n} v_{i,k}^2} \sqrt{\sum_{k=1}^{n} v_{j,k}^2}}$$

4.1: Compute a matrix representing the cosine similarity between users, i.e. treating each row of `interaction_matrix` as a vector. Assign this to `user_similarity`.

[Note: You may want to consider adding a small "fudge" factor such as 1e-10 to prevent any issues with ratings of 0.]

In [ ]:
```
...
user_similarity = ...
display(user_similarity)
```

4.2: What does each value in the matrix represent? If there is a certain value along the diagonal, why is that the case? How could we now use this matrix to recommend movies to a user? What drawbacks are there with using this approach?

Answer:

4.3: Now compute a matrix representing the cosine similarity between movies, i.e. treating each column of `interaction_matrix` as a vector. Assign this to `item_similarity` .

[Note: You may want to consider adding a small "fudge" factor such as 1e-10 to prevent any issues with ratings of 0.]

In [ ]:
```
item_similarity = ...
display(item_similarity)
```

4.4: What does each value in the matrix represent? If there is a certain value along the diagonal, why is that the case? How could we now use this matrix to recommend movies to a user? What drawbacks are there with using this approach?

Answer:

Question 5: Model-Based Approach: Matrix Factorization and Embeddings

There are a few issues with the cluster-based approach in Question 3. The biggest issue is that this casts the data in terms of the users we have data for. In other words, we are taking a 'User-User Approach' that is highly reliant on similar data or ratings between users in order to form reliable or accurate clusters. In reality, our interaction matrix is very sparse, meaning that many of the entries or values in it are 0. This means that it may be hard to identify similarity between users, and any slight changes will alter our recommendations.

Another approach we can use is known as an 'item-item approach' that instead leverages matrix factorization to break apart matrices into other matrices or vectors that have special meanings or interpretations.

5.1: What are the two Matrix Factorizations you have seen in EECS 16A and EECS 16B? What are the tradeoffs or different use cases for one factorization over the other?

Answer:

Here is a quick refresher on the SVD:

For a matrix $A \in \mathbb{R}^{mxn}$, the "Full SVD" is the following matrix product:

$$A = U\Sigma V^T$$

where

$$U \in \mathbb{R}$$
$$\Sigma \in \mathbb{R}$$
$$V^T \in \mathbb{R}$$

Alternatively, there is a "Compact SVD" that involves truncating these matrices to remove zero-value singular vectors corresponding to the Nullspace of A:

$$A = U_c\Sigma_c V_c^T$$

where

$$U_c \in \mathbb{R}$$
$$\Sigma_c \in \mathbb{R}$$
$$V_c^T \in \mathbb{R}$$

As popularized during the 2006 Netflix Prize, there is a famed "SVD" Matrix Factorization Algorithm used for creating a recommendation system. NOTE: THIS IS A DIFFERENT SVD! Even though they are both called "SVD", and are spiritually related in the sense that they are both related to matrix decomposition/factorization, they are in fact, different algorithms/procedures entirely. They are related in the sense that after the following procedure, the resulting matrix A should converge to the SVD.

Our goal will be to factorize `interaction_matrix` and approximate it as the product $A = UV^T$. For a matrix $A \in \mathbb{R}^{mxn}$, it will be equal to the product $U \in \mathbb{R}^{mxd}$ and $V^T \in \mathbb{R}^{dxn}$.

In the literature, U is referred to as the User Embeddings, while V is referred to as the Movie Embeddings. Each row of U, represented by U_i, denotes the "essence" of user i, while each row of V i.e. each column of V^T, represented by V_j, denotes the "essence" of movie j.

5.2: How can we approximate the matrix A using a sum instead of a matrix product?

Answer:

5.3: Why can't we simply ues the row and column vectors in `interaction_matrix` to compute this?

Answer:

5.4: Calculate A^*, the approximation for A, using gradient descent with randomly-initialized embedded vectors for U and V.

In [ ]:
```
U = np.random.normal(size=[interaction_matrix.shape[0], 5])
V = np.random.normal(size=[interaction_matrix.shape[1], 5])
```

In [ ]:
```
gamma = 0.1

for u in range(len(U)):
    for v in range(len(V)):
        for _ in range(100):
            ...
```

In [ ]:
```
A_star = np.dot(U, V.T)
```

Question 6: Extensions

Through this assignment, we hvae explored the mathematical underpinnings for two methods of collaborative filtering that are used to build recommendation systems. If you wish to further investigate this application of Machine Learning, here are some external packages that are used in production settings, as well as links to recent developments:

Surprise

Surprise is a Package for SciPy. It is a dedicated SciPy package for building and analyzing Recommender Systems. In particular, it has native access to various datasets such as MovieLens, and also has a wide array of prediction algorithms.

- SurpriseLib Website (http://surpriselib.com/)
- GitHub Link (https://github.com/NicolasHug/Surprise)

Deep Learning and Deep Neural Networks

Later on in 16ML, we will introduce the idea of Deep Learning, which relies heavily on using Neural Networks with many, many layers to perform complex computations. These are considered state-of-the-art models that are at the forefront of recent advances in many different fields of Machine Learning, and are a valid option for developing a recommendation system.

Regularization

When performing Model-Based Collaborative Filtering, we can run into the issue of underregularization. This is a concept that was discussed earlier in 16ML during Week 4. In essence, regularization allows us to "smooth" out our data by "lifting" up small, near-zero values in our data that can cause numerical instability and/or unexpected magnification of small values. Regularization can help with Collaborative Filtering by dealing with movies that were not rated by many users, or users who did not rate many movies. In this way, it can help reduce error in predicting or recommending new movies to users, or conversely, identifying new users who might like a given movie.

References

- Baptiste Rocca. Introduction to recommender systems. https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada,2019 (https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada,2019).
- Simon Funk. Netflix update:Try this at home. https://sifter.org/simon/journal/20061211.html (https://sifter.org/simon/journal/20061211.html), 2006.
- Yehuda Koren. The bellkor solution to the netflix grand prize. Published on Netflix PrizeForums, 2009.
- Build a Recommendation Engine With Collaborative Filtering. https://realpython.com/build-recommendation-engine-collaborative-filtering/ (https://realpython.com/build-recommendation-engine-collaborative-filtering/)
- Prince Grover. Various Implementations of Collaborative Filtering https://towardsdatascience.com/various-implementations-of-collaborative-filtering-100385c6dfe0 (https://towardsdatascience.com/various-implementations-of-collaborative-filtering-100385c6dfe0)
- Intro to Recommender Systems: Collaborative Filtering https://www.ethanrosenthal.com/2015/11/02/intro-to-collaborative-filtering/ (https://www.ethanrosenthal.com/2015/11/02/intro-to-collaborative-filtering/)
- Alesha Tony. All You Need to Know About Collaborative Filtering https://www.digitalvidya.com/blog/collaborative-filtering/ (https://www.digitalvidya.com/blog/collaborative-filtering/)
- MovieLens Dataset https://grouplens.org/datasets/movielens/ (https://grouplens.org/datasets/movielens/)