Lucas Derr
Dr. Burke
INFO 5612
20 December 2023

Fresh Finds: Final Project Report

1. Project Overview

1.1. Task

The initial motivation for this project was to build a recommendation system that could help me with a task I encounter regularly: purchasing music. In 2023, music consumption can be split into two categories: streaming platforms and physical mediums such as vinyl and CDs. While streaming platforms are heavily reliant on recommendation, physical mediums remain old-fashioned in their existence as solely physical product. This project will build a recommendation system that brings the technology of streaming platforms to the aging vinyl and CD industry.

1.2. Dataset

The dataset for this project is a compiled subset of the Amazon Product Reviews called Multimodal Album Reviews Dataset (MARD) containing only album reviews (most of which are for CD or Vinyl). The dataset has reviews that span from the years 1997 to 2014. There are 263,525 total reviews encompassed by 64,637 unique albums and 187,090 unique users. There are three tables in the dataset: ratings, album metadata, and audio data. The ratings table pairs user IDs with item IDs and review text. The album metadata has specific information about each album such as price, genre, and label. The audio data is about 800 features compiled from the AcousticBrainz API.

1.3. Goals

Remembering Dr. Burke state that many modern recommendation systems “use as much data as they can get their hands on”, the initial goals for this project were to emulate this approach: utilizing ratings, text data, content data, and audio data to build a comprehensive recommendation system. Due to unforeseen implementation complexity and lack of computational power, the goals were dialed back to only include the use of the review text, in addition to the interactions matrix between users and items.

2. Project Activies

2.1. Evaluation Metrics

In recommendation, evaluation is much more complicated than in other realms of machine learning. Simply optimizing an objective function such as mean squared error can fail to give the context to appropriately assess the behavior of a recommendation model. I decided to utilize four metric functions to give a more complete analysis and means of comparing the performance of different models: NDCG@30, Precision@30, Novelty, and Coverage.

NDCG stands for normalized discounted cumulative gain. It takes the position of the recommended items into account using a gain function. Precision measures the proportion of recommended items that are deemed relevant. Both Precision and NDCG are specified with @30, meaning they consider recommendation lists of size 30. The justification for the size 30 recommendation lists is that albums are a longer form of content. Users will spend more time sifting through recommendations because the decision to buy an album is more consequential than the decision to click on a song, for example.

The other two metrics are Novelty and Coverage. Novelty measures the average entropy of each recommendation list. This metric is meant to indicate the degree to which the recommender tends towards popular items. Finally, Coverage is a simple ratio of the number of unique items recommended in all recommendation lists divided by the total number of items in the dataset. It measures the model's ability to recommend all items in the dataset.

2.2. Baseline Model

My baseline model is a Popularity Recommender. It returns the same top k most popular items for each user. This model was chosen as the baseline because it gives insight into the extent more sophisticated models outperform a naive solution.

2.3. Collaborative Filtering Only Model

The collaborative filtering model will implement the LightFM base model class and only input the interaction data between the users and the items. Explicit embeddings will not be used for this model.

2.4. Cleaning Text Data

To generate item embeddings, a natural language processing pipeline was implemented to train a Word2Vec model. The first step in the pipeline is cleaning text. Cleaning is useful to standardize the format and reduce the amount of noise in the text data. This way, the Word2Vec model can extract more meaning from each review. My text cleaning function includes the following methods:

3. Convert text to lowercase
4. Remove punctuation
5. Remove unnecessary whitespace
6. Remove URLs
7. Remove non-characters
8. Remove stopwords
9. Lemmatization

Word2Vec works on individual words. Each of these methods plays a part in reducing each review down to only its most essential words.

9.1. Training Word2Vec

Once the text is cleaned, it is split into a double list of sentences and words and used to train the Word2Vec model. One of the difficult things about embedding models such as Word2Vec is that there is no ground truth to measure their accuracy. To confirm that the model was working, a plot was generated after converting the embeddings from 300 to 2 dimensions with SKlearn's t-SNE.

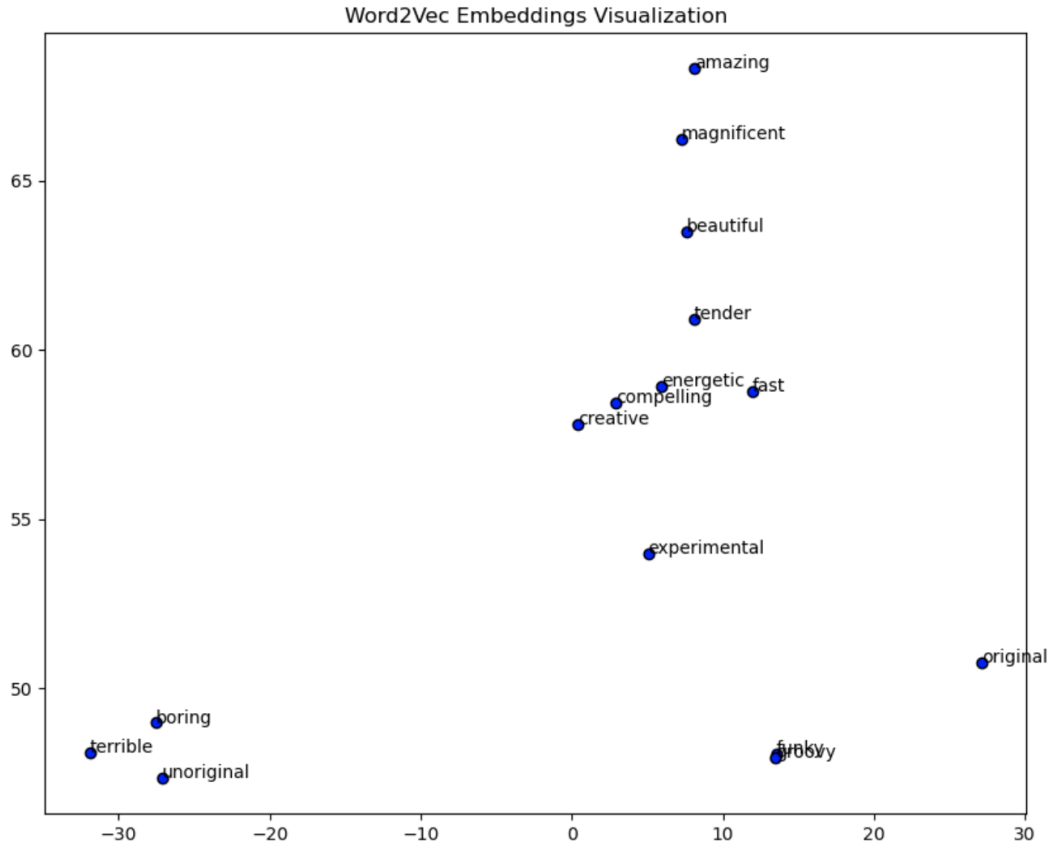


Figure 1: Word2Vec Visualization

In *Figure 1,* the visual inspection to determine the accuracy of the custom Word2Vec model appears successful. Negative words such as terrible, boring, and unoriginal are shown far away from positive words such as amazing and magnificent. Additionally, energetic and fast are close together, and creative, compelling, and experimental are close to one another.

9.2. Generating Embeddings

The issue with Word2Vec is that it can capture the meaning of words, but not groups of words. To capture the meaning of an entire review, Word2Vec first generates an embedding for each word in the review, then takes the column-wise mean. The result is an embedding of size 300 for each review in the dataset. To generate item embeddings, the mean review embedding is taken for each item.

9.3. LightFM Model With Item Embeddings

This model will build on top of the base LightFM collaborative filtering by including the item embeddings generated from the NLP pipeline.

9.4. PyFM Model with Review Embeddings

This model takes a slightly different approach, using review embeddings (not item embeddings) to predict rating scores as a regression task. Due to lack of time, a fully fleshed out implementation of this model was not used. Instead, a single model was trained and evaluated only with mean squared error, achieving a test score of 0.684591947419034.

10. Evaluation

10.1. Baseline Model Evaluation

Popularity Recommender Metrics	
Metric	**Score**
NDCG@30	0.09515
Precision@30	0.00309
Novelty	0.025720
Coverage	0.00046

Figure 2: Baseline Model Metrics

Figure 2 gives the metrics for the baseline Popularity Recommender. Notice the high novelty and low coverage, which indicate the recommender tends towards popular item and is only able to recommend 0.046% of the itemset.

10.2. Collaborative Filtering Only Evaluation

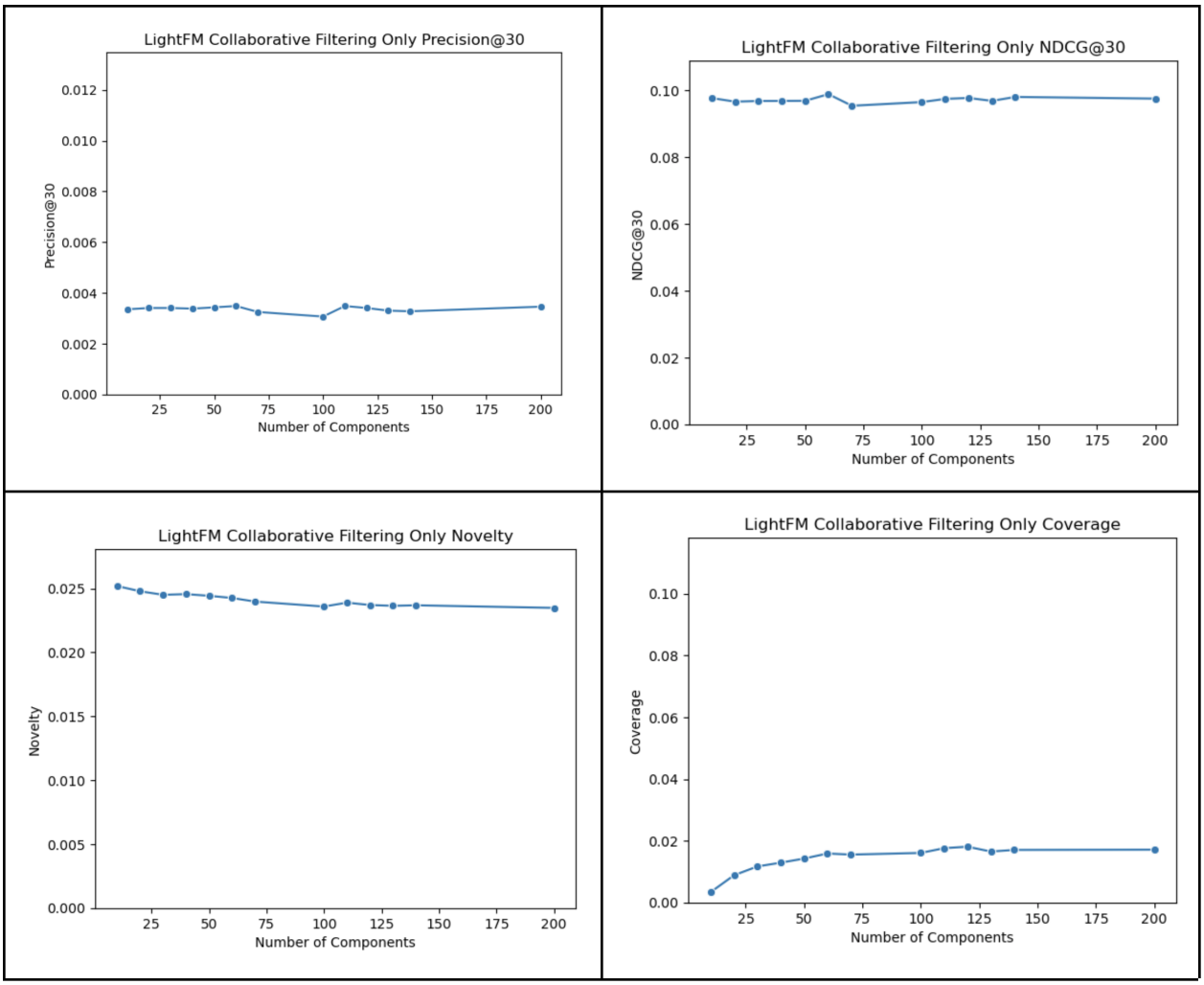
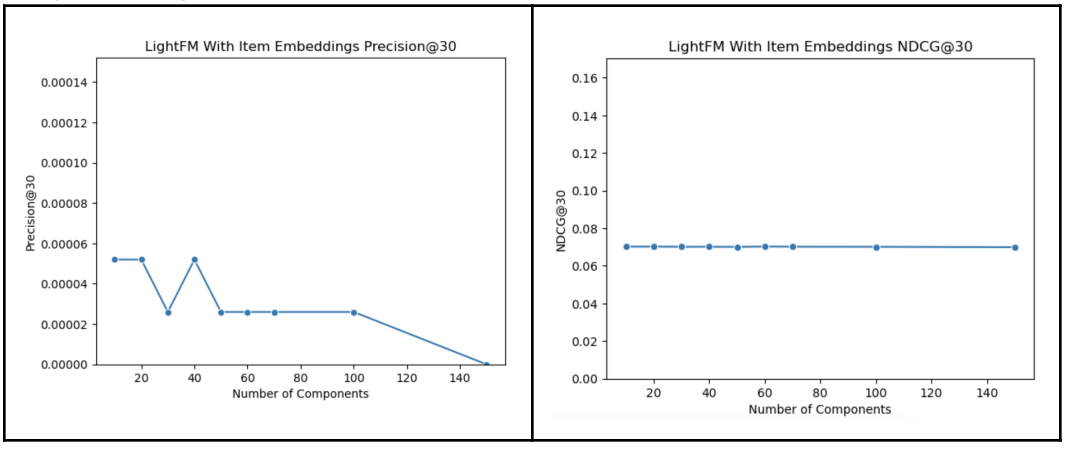


Figure 3: Collaborative Filtering Only LightFM Evaluation

Figure 3 gives four graphs depicting the change in Precision@30, NDCG@30, Novelty, and Coverage as the number of components is increased on the x-axis for the LightFM Model which only considers the ratings matrix.

10.3. Hybrid LightFM Evaluation



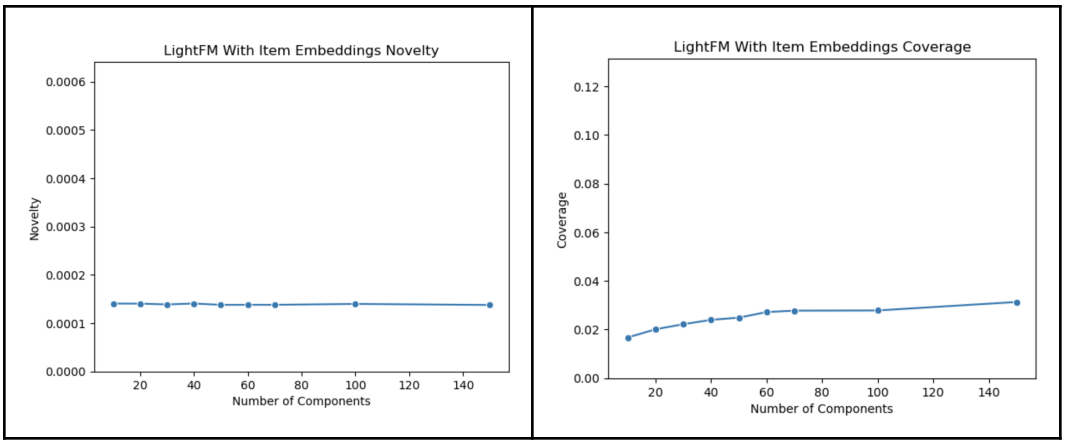


Figure 4: LightFM Collaborative Filtering with Item Embeddings Metrics

Similar to *Figure 3, Figure 4* demonstrates the established metrics with the LightFM model including the item embeddings generated from Word2Vec.

10.4. Discussion

Unfortunately, the metrics for the LightFM Collaborative Filtering-Only, LightFM with Item Embeddings, and PyFM do not yield favorable results. As is seen from the high Novelty values in *Figure 3*, the Collaborative Filtering-Only model tended to recommend the most popular items. The LightFM with Item Embeddings Model had much worse Precision and NDCG but improved Novelty and Coverage. The large embedding size most likely introduced a lot of noise to the dataset, preventing it from performing well on the test data.

Most of the issues with these results are most likely a reflection of poor implementation practices rather than issues inherent to the dataset or algorithms. It is difficult to pinpoint exactly what and where things went wrong. Still, the following is a speculative list of things that would be done given more time on the project.

1. Implement K-Core and/or an anti-train/test set
    - Preprocessing steps such as these could have reduced noise and improved the results of the model

2. Making Changes to Embeddings
    - The embedding length of 300 could have done more harm than good by adding more noise than signal
    - Using a simpler embedding model such as CountVectorizer or TF-IDF could have also improved results

3. Trying Different Algorithms
    - The LightFM Model (with many combinations of parameters) did not perform well on this dataset. Spending more time exploring different types of models could have improved performance. Finishing the PyFM model would be a great start.

4. Incorporating Different Types of Data
    - Text Data is inherently messy due to its lack of structure. Using some of the other content data first might have been more effective.

5. Adding a Reranking Algorithm
    - Example: Forcing the item recommendations to have a specific genre distribution

Figure 5: List of Things to Try

It Is unclear if any (or all) these changes would have improved the results. These are just based on my novice intuition.

11. Conclusion

11.1. Takeaways

This project was a lot of fun to implement. Despite the results, I was able to get a lot of things working. I also learned a great deal about recommender systems and gained practical experience in machine learning, especially the NLP portion of the project, something I had to learn. The main takeaway is that machine learning is difficult. Every step of the process can be implemented correctly, but the model can still spit out undesirable results. The real challenge is not the initial implementation, but improvements made afterwards. I would love to come back to a project like this after a few years of work experience and see how my approach would change.

11.2. Limitations

Several barriers prevented efficient progress in this progress. First, working solo meant I could not get as much done, and I lacked the perspective and collaboration of a partner. The next limitation is computation speed. Doing anything- preprocessing, embeddings, model training, model evaluation- took a significant amount of time on my 6-year-old laptop. Another significant time waster was switching between libraries too much. There is a lack of standardization in the recommendation library world. Every library uses its own data structures and implementation functions. Lots of things had to be built from scratch just to get the code running the way I wanted. After getting a library working, switching to a new library required writing brand new code to fit within that new library's implementation.

11.3. Appendix: GitHub Repository Link