Midterm Update

Grant Doan, Prayash Joshi, Reagan Orth, Ved Patel

May 3, 2023

1 Technical Description

1.1 Introduction

1.1.1 Problem Description

Movie recommendation systems always have biases, and a successful system will eliminate as many of these as possible. A very prominent example is popularity, because people trying to see "the best films of all time" tend to watch many of the same, unrelated movies. In the IMDb recommended selections for Mr. Smith Goes to Washington (a 1939 comedy/drama about a boy scout leader who becomes a moral idealist US senator), we can find the film Wages of Fear (a tense 1955 French suspense thriller about hired drivers transporting dangerous explosives across a hazardous road). Both are highly rated films, but they have very little do with each other, and there is little reason to believe that a user who likes one will like the other beyond the fact that both are old and filmed in black and white. A more modern example is that Lord of the Rings (a fantasy journey film), Joker (a dystopian antihero film) and Interstellar (a huge-production sci-fi space travel film) are recommended on the page for The Shawshank Redemption (a relatively slow-paced, atmosphere- and feeling-driven prison film). None of these three recommended films have a single prison scene, they don't share starring actors or a director, they don't share the slow pace, and as a whole are very different films that appeal to very different audiences.

Another frequent issue is grouping films with other films of similar ratings. When looking at a film of rating 8.0, the recommended films tend to be in the high 7's and 8's. When looking at films of rating 5.0, the recommended films tend to be high 4's and low 5's. However, clearly every user will rate films differently, and users will not base their ratings on the IMDb average user rating. Therefore, we believe that although rating-based recommendation has some limited merits, it is more harmful than helpful overall.

1.1.2 Motivation

The goal of this project is to create a movie recommendation system that removes common biases and issues with the popular systems today. Movie preferences are very complex, and it's often fairly difficult as a viewer to know what kinds of films you enjoy. Even after you know, some themes or genres are easier to find, some are harder. In all cases we're confronted with the problem that we do not know how much we will like a movie until we see it, which costs us between one and four hours for each one. Online lists can be very helpful, reading reviews and watching trailers might sway one's opinion or give a little indication of how good the film will be, but there is no way to ever know for sure.

In order to solve this problem, we intend to create a recommendation system that takes in commentary from users who have already seen the movies, and use that information to better understand if it might apply to the user in question. In this manner, we believe we will be able to predict how much the user will enjoy a movie to some degree of accuracy, which in turn will allow the user to watch more movies that they enjoy and fewer that they dislike.

1.1.3 Challenges and Solutions

Our primary focus is creating a recommendation system that does not contain these two common and harmful biases. The premise of our solution is that the things that are said about a movie can provide rich insight into what might appeal to viewers and what turns them away. We therefore intend to use IMDB's film summary and featured plot synopsis in conjunction with other critical film information in order to create an undirected, unweighted graph, with nodes representing movies and edges representing similarities. After this, we will use IMDB's user reviews and ratings in order to develop a rating prediction model using our graph with its basis on the individual user. With a large enough dataset, we should learn useful similarities and preferences, such that a user who liked certain movies will be recommended movies with similar casts or genres or plot elements, or other factors that we may not think to mention. Even more important for our model than recommending perfect films is not recommending dissimilar films.

1.2 Literature Review

General overviews of the recommendation system research space decompose filtering methods into 3 categories, each with their own frameworks and methodologies of execution: Content based filtering, collaborative filtering, and hybrid filtering. Each of these filtering approaches are addressed along with recent executions in the literature space.

1.2.1 Content Based Filtering

Content based filtering takes the approach of finding items that share characteristics motivated by the idea that if a user enjoys item A, and item A is similar to item B, then the user in focus will enjoy item B. As a user, it is easy to find similarities between movies through making the connections between genres, actors or actresses, directors, and other qualitative factors, but for machines, quantifying these things precisely and efficiently is the focus of nearly all of the current landscape of recommendation systems. A content based recommendation system approach supplied by Pujahari and Sisodia in 2022 propose matrix factorization-based feature refinement using textual data. Term Frequency Inverse Document Frequency (TF-IDF) extracts the essential words or phrases in teh text documents and displays it in a vector space, weighing highly reoccurring words as essential. In order to combat the sparsity issue that plagues vectorization of text data, the authors use a factorization technique that attempts to lower the dimension finding factors that correlations that are redundant and optimizing feature refinement. The item feature matrix F is subject to reduction based on projecting the feature matrix on a vector W where that vector is the combination that either includes or doesn't include a certain quality in the similarity measure. This is upheld by the constraint that $\mathrm{W}^{\mathrm{t}} * \mathrm{W} = \mathrm{I}$. Therefore, by finding optimal matrix of weights for our feature space, sparseness and redundancy between factors is minimized. Furthermore, finding optimal values for W allows them to select the most important features for consideration by the system, which is a trade off if there is a limitation on existing computing power. They then propose an iterative optimization process for W and another factor K which is the coefficient matrix used for projecting the original item's feature space to the selected feature space. An important issue tackled by their approach is data sparsity, which is a common limiting factor for evaluating text based features (reviews, plot synopsis, tags, etc.) as the feature vectors that are considered are large in dimension. Using smaller vectors that do not include textual information is an approach, but often suffers from being blanket generalizations or categorizations of the information if we use the information just supplied by what can be found on a search. For example, saying to action movies are similar because of their genre is a true statement, but user preferences tend to be more specialized than simply liking only action movies, therefore a system using only this consideration may not have the best performance on recommending something specifically to a user's taste.

1.2.2 Collaborative Filtering

Collaborative filtering removes the item from consideration and is motivated only by the idea that a user with similar taste gives a better idea of items a user may enjoy. This space takes advantage of potentially recommending items that are outside of what the user considers their "taste", but may still be in line with something they enjoy. An important part of filtering with this method is creating relationships between users that link them based on similarity. A survey by Goyani calls these "neighbours" but other terms such as trust have been used to describe the relationship. Ratings given by the user and other users form a user-item matrix and is a large catalog of items rated by the user. In The Adaptive Web, by Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen, a primary technique to accomplish the task of finding neighbours to a target users uses k-Nearest-Neighbors to classify a target profile with historical and current data of other users in an effort to classify based on similar preferences [3]q. Similarity between users to classify into a neighbor or not a neighbor has a variety of approaches. One such framework supplied by Bobadilla and colleagues use Mean square different between 2 users x and y, given similar movies they have rated together. Given these 2 user's meet a similarity threshold, the system begins predicting creating a weight for how similar a user is to another user and considers k neighbors for the predictive rating of a user onto an item. Average weighted sum and adjusted weighted aggregation are both used as predictions for ratings, and a normalization factor is applied to these aggregated sums based on the similarity of the user to the user that whose rating is being predicted. This way, the nearest neighbor's contribution to the rating in question is considered as more important to the prediction than a neighbor who is further away. The authors explore a variety of similarity measures such as Constrained Pearson Correlation, Pearson Correlation, Spearman rank Correlation, cosine similarity, and Mean Squared Differences to their neighbor classification systems and reject MSD as out performing more conventional measures of similarity. K hear can be any number of neighbors so long as efficiency of computing the values for similarities is inline with computation limitations. For enterprise systems like Netflix and Hulu users exist in the millions, therefore computation must be scaled adequately to provide for efficient prediction and recommendations. This will be further explained in limitations.

1.2.3 Hybrid Filtering

Hybrid filtering alleviates the filtering system from focusing significantly on the user and significantly on the item, combining both to yield recommendations of items both similar to the taste of the user and of the taste of similar users. In Goyani's survey, outputs between content based methodologies and collaborative filtering based methodologies are used as input for recommendation. Deng and colleague's work in hybrid recommendation combined used popular user-item rating matrices as well as static feature's pulled from the Movielen's database to create dynamic and static features that are then hybridized to calculate how a user may be interested in a certain movie. This is then used to generate similarity matrices and neighborsets of both movies and users, respectively, and combined again with the user's item rating matrix to generate a prediction for the item. These feature matrix calculations are sparse matrices that indicate a user's interest in a certain feature, so for a user u and feature k, a user can be interested(1) or not interested(0). The features comprise of characteristics about the movie itself being things like genre, movie tags, and other relevant information users associate with movie content that are general across many movies and genres. This worked in combination with kNN to predict values for movies yet to be rated by the user in focus.

1.3 Limitations of Current Approaches

Each of these filtering methods come with common challenges that are limited by a variety of factors. *Cold Start* : For collaborative and hybrid filtering systems, effectiveness only comes when there are an adequate amount of users in the system to find neighbors and, therefore, similarity. Moreover, actually being able to rate 2 users as similar requires even greater specificity between the users that are engaged with the system. Therefore, there is trade-off between the performance of the neighboring classification and the users that are in the system as recommenders can attempt to begin their classification with little users, but the users that are classified as neighbors to a particular user may not be similar at all. This comes as a result of not having nearly enough users to find similarities between, throwing off the accuracy of item recommendation. *Data Sparsity* : Following Cold start, data sparsity comes commonly with the matricization of items into feature matrices to use for similarity. Consider matrices composed of a user and their ratings. Collaborative filtering dictates that users can have connections drawn between them by rating similar items, but it is common that a user will have ratings for movies or items that do not exist for the focused user. As we scale, we have matrices of user's ratings for movies that are empty but are still taking up space on the system. This same issue is paralleled in content based filtering where similarity between tags is often compared using NLP methods. As a result, comparisons between corpuses of text involve creating matrices where words in one synopsis, for example, do not show up for another synopsis. We have this issue of the data that is being fed into our model being mostly empty and taking up space on what could otherwise be used for data with value. While this is not an issue on small systems, a final issue of scalability presents itself. *Scalability* In order for our systems to create valuable insight into recommending items to users, it requires we have lots of data for the model to learn from and recommend. As a result, the more users, movies, and other items we have added on to the model for consideration can become a computational burden due to the expense of processing. This is the tradeoff that is expected when attempting to tune high functioning models that can succeed across a wide variety of users. Paired with the issue of data being sparse, issues arise with the model being exposed to data that is of high quantity but varying quality.

1.4 Proposed Approach

1.4.1 Problem Definition

Our task can be condensed into two core mathematical problems: edge prediction for our graph [6], and rating prediction for individual users. For the graph, we will input our features and obtain the predicted edges, which we can then use for similarity measurements: the shortest path from any movie to any other movie is the similarity measurement. We will likely not need extensive graph traversal calculations, as we will generally be using films that are very similar to other films on our graph. For the user rating prediction, we will be using the graph and inputting the user's preprocessed reviews and film ratings and obtain an approximate rating prediction out of ten.

1.4.2 Data Pre-Processing/Feature Engineering

Preprocessing and feature selection are a critical part of any machine learning model, and even more so in such a complex problem as this. There are many important decisions to make in order to prioritize different selection optimizations, and it is therefore imperative to include information that we believe to be critically important to recommendations, and omit information that might cause the problems we intend to address.

Surrounding popularity bias, we will not be taking into account overall ratings or number of watches/reviews. Additionally, we will try to mitigate the effect of popularity by strategically dropping reviews in an inversely proportional manner to their popularity, such that uncommon movies will keep all of their reviews and extremely popular movies will only have a fraction of their body of reviews.

Our data was scraped from the IMDb site, and therefore had a number of formatting issues and type inconsistencies. To clean this dataset, we fixed all of these formatting issues, and subsequently assigned each director, actor, and genre to an index in order to make a binary matrix of 0s and 1s. We also added one feature for each decade, as individual years are far sparser and do not tend to give us much additional information. These features were chosen from our own experience searching for movies, and therefore are the factors that we believe should have the largest impact on what kind of movie each one is and which films it may be similar to. We initially intended to make use of the plot synopsis, but as we've seen promising results without, we have decided not to overly complicate the model with thousands of NLP features or a condensed representation thereof.

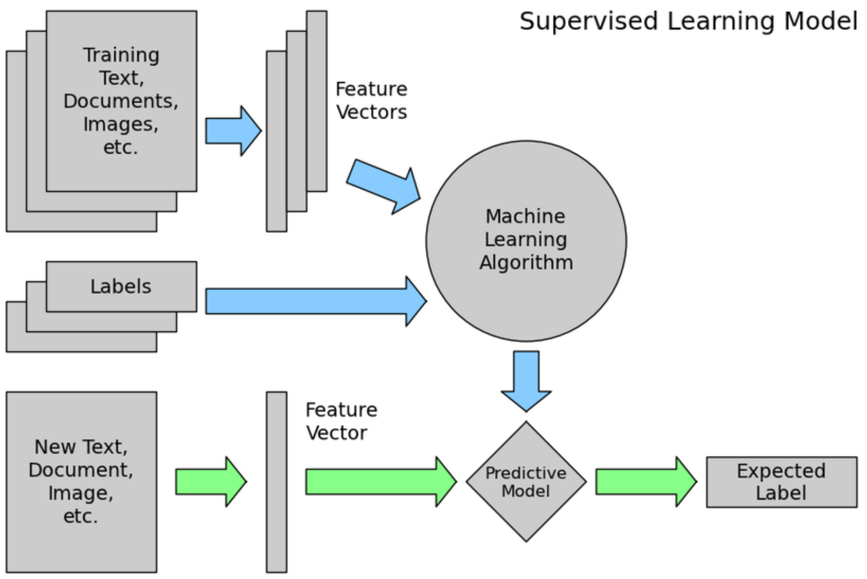
NLP preprocessing utilizes a pretrained classification model known as BERT. BERT is a text encoding transformer that is characterized by deep learning bidirectional text representations. The model is pretrained on general purpose texts such as wikipedia and has been pretrained to perform tasks such as sentiment analysis, emotion categorizations, speech to text translation, and many others [4].

For this project, we explore aspect based sentiment analysis on a pretrained bert model by classifying statements into positive, negative, or neutral categories, then use nltk to parse out figures of speech that are the subject under review. As a result of the use of a pretrained BERT model, we leave our text as a whole when putting into our BERT model, but lemmatize and tag parts of speech to find our relevant subjects.

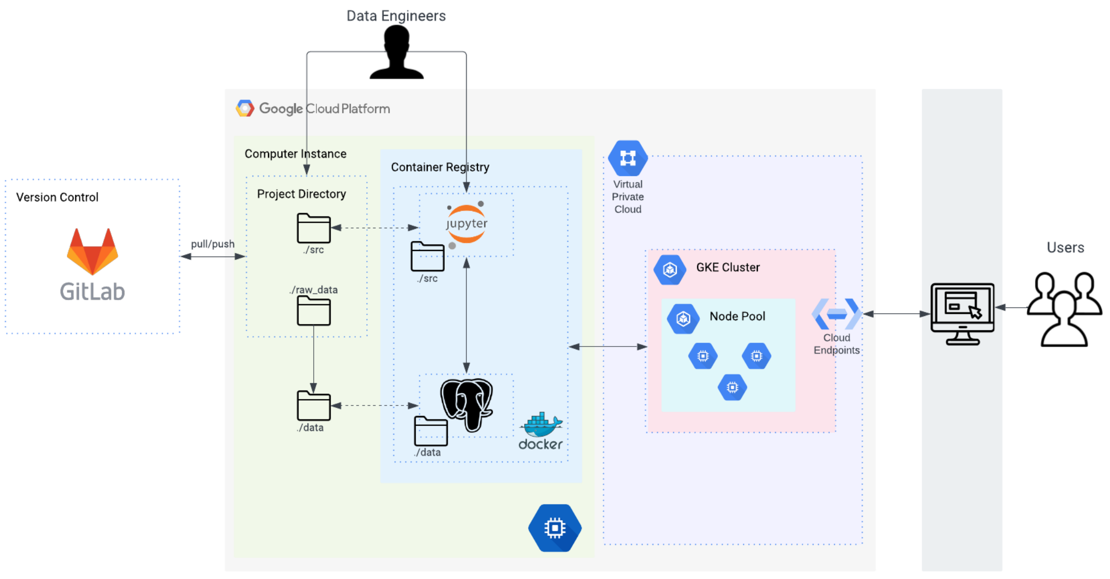
1.4.3 Workflow/pipeline/system architecture diagram

Our final deployment product is a movie recommender for online users. By processing a few reviews of movies they've watched, we will use a classification and recommendation model that can find something they could enjoy. Our model needs to go through a series of pipelines in order to take the user's input, launch the model, and display the result to the user. [1]

Our project's first phase is classification. The classification model is trained on a fixed dataset of IMDB movie reviews. The feature vectors and labels are fed into our GNN classification model. We include a workflow pipeline that takes the user input as new text, processes its feature vectors, and runs it through a predictive model to generate the expected labels. [5]


Supervised Learning Model
Training Text, Documents, Images, etc.
Feature Vectors
Machine Learning Algorithm
Labels
New Text, Document, Image, etc.
Feature Vector
Predictive Model
Expected Label

Below is a sample of a possible system architecture on Google Cloud Platform using its Compute Engines:
[2]



1.5 Experimental Evaluation

1.5.1 Dataset Exploration and Patterns

The Movie dataset is a dataset of IMDb's top 1000 movies. We have most of the critical information for each movie, such as name, year, director, starring actors, plot synopsis, MPAA rating, length, and genres. The dataset is inherently limited to popular movies, but we believe 1000 movies will still allow us to make meaningful recommendations. There are not many patterns to this dataset, as the range of movies that people love is so broad and encompasses most facets of the film industry. However, it should be noted that certain echo chambers exist within this dataset, such as the inclusion of many of the most famous foreign-language movies and few others, or directors that modern audiences love such as Christopher Nolan or Quentin Tarantino, or popular franchises such as The Avengers or The Lord of the Rings. The dataset includes a mix of popular films, classics, and hidden gems, and does not include the 1000 most highly rated movies, but rather uses a closed-source selection algorithm. We do know that it is largely based on ratings that have been built up over the past twenty-five years, and therefore certain films naturally have more attention and certain ones have less.

An additional dataset used in our sentiment analysis comes from user reviews of these same top 1000 movies. We scrape information such as the user that gave the review, the rating of the movie, and the proportion of users that found that movie review helpful. The review that is scraped is the most relevant piece of information as the content of review as this review is used to the strengths and weaknesses of a movie as well as what a user enjoys about movies.

1.6 Predictive Model

Our predictive model comes in three parts. The first is with sentiment analysis, where we take in user-written reviews and predict sentiment on individual sentences and the aspects that are given those reviews. We do this in an effort to investigate what about a movie corresponds to its ratings. For example, common strengths from a movie can be its aesthetics, characters, story, acting, etc. and we aim to tag each of these movies with common strengths. We then map these strengths for the movies we have reviews for. Furthermore, these tags can be characterize a reviewer as well. Seeing these sentiments appear commonly throughout a user's reviews gives a general idea of the particular film elements a user enjoys.

The second part is the forming of the graph, where the model predicts links between movies. This is done using a Graph Neural Network with two SAGEConv layers, taking in an n x m matrix of True/False features, where n is the number of movies and m is the number of features, and outputting an n x n matrix of similarities. We used a similarity function in order to train the network, which compares features between movies and outputs similarity score. In our trained model, we can clearly see that movies with similar features tend to be rated very similarly to each other. For example, the most similar film to The Godfather is The Godfather Part II, the most similar to The Dark Knight is the two other Batman movies in that series, and the Lord of the Rings movies all feature the other two as the most similar.

The third part comes in the form of using our graph to predict movies to recommend. If we simply wanted single-movie similarity, we could simply use the most similar movies on the graph. However, as we want to combine preferences, we need to take multiple movies into account. Currently, we are using a manual, distance-based searching algorithm on the graph to find movies that are close to the films a user likes and far from the films a user does not like. We have had reasonably good results so far, but are considering other approaches to better take advantage of all of the information we have.

1.6.1 Midterm results

1.6.1.1 Sentiment Analysis

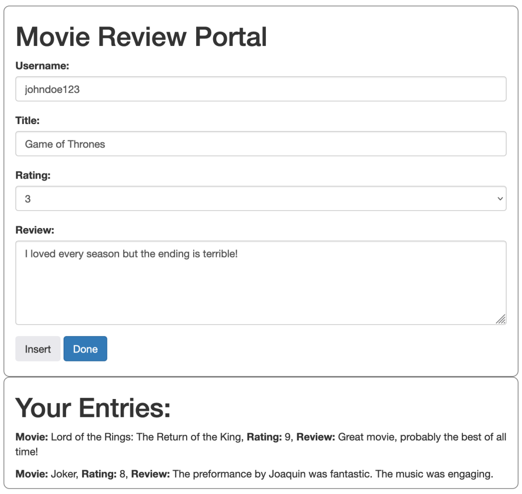
In the Graph Neural Network section, we have been able to build the similarities graph. As this is an unsupervised problem (we do not have labels in advance), we cannot compare to any objective standard, but upon inspection we have been fairly happy with the results. We are still looking into improvements, most prominently gathering more film data so we have more similarities to make use of. We have also been able to input movies and obtain an output recommendation. Fine tuning our pretrained BERT model to our movie reviews was a simple task in adding additional information to continue training. Our BERT model currently does a good job of evaluating overall sentiment from movie reviews in totality and individual statements with an accuracy of about 90 percent and our loss converging on a value of about .20. However, extracting the topics that are the subject of such reviews has been a much more difficult task as parsing out exactly which nouns contribute or don't contribute to the label has been difficult. There is certainly more to be added on to tune our BERT model in neutrality detection and aspect retrieval, but an elementary classification of positive and negative has been obtained and works with confidence. An additional lead that was pursued in our project was the modeling of common topics in an effort to find film elements the user's enjoyed. We are pivoting to aspect based sentiment analysis as a result of our failures in using TF-IDF and BERTopic to find commonly occuring words and clusters of review text, respectively. We believe that the aspect based sentiment analysis proposed as part of the PyABSA can be fine tuned to include movie reviews well and will lead to better synthesization of common elements in movie reviews [7]. Furthermore, neutrality in statements is an important element that need be parsed out in these movie reviews as some can just be stating facts about a film or its elements as opposed to a user giving an opinion on something.

1.6.1.2 Website

Based on our plans, we aim to take in a given set of reviews which the user can insert on the website and provide a recommendation based on their reviews. First iteration of the webapp took in one review at a time which was stored in a Firebase real time database. We are storing user reviews from our website for additional metrics to feed into our recommender in the long run. In this way, we can hyper-parameterize review recommendations that utilizes collaborative and content based filtering based on the user's previous history of movie reviews.

The first iteration of the website was a static website with a form that used an API to connect to a remote online database. The second iteration of the website allowed for users to input multiple movies at a time. In addition, we transitioned from a static website to a dynamic website which had the ability to route json formatted input and output from our front-end to our back-end recommendation model.

Below are the preliminary website and database review structure respectively:



Movie Review Portal

Username: johndoe123

Title: Game of Thrones

Rating: 3

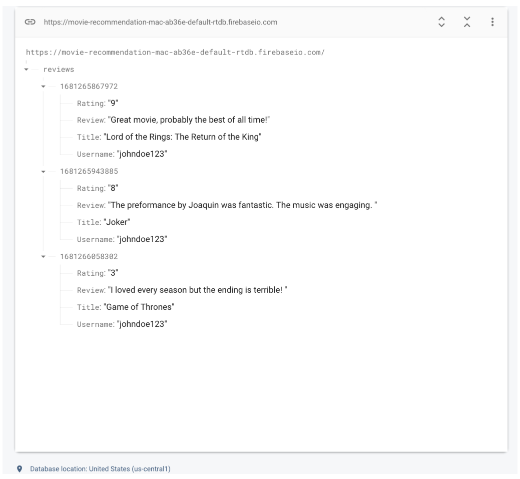
Review: I loved every season but the ending is terrible!

Your Entries:

Movie: Lord of the Rings: The Return of the King, **Rating:** 9, **Review:** Great movie, probably the best of all time!

Movie: Joker, **Rating:** 8, **Review:** The preformance by Joaquin was fantastic. The music was engaging.

(a) Dynamic Webapp



```
https://movie-recommendation-mac-ab36e-default-rtdb.firebaseio.com

https://movie-recommendation-mac-ab36e-default-rtdb.firebaseio.com/
reviews
  1681265867972
    Rating: "9"
    Review: "Great movie, probably the best of all time!"
    Title: "Lord of the Rings: The Return of the King"
    Username: "johndoe123"
  1681265943885
    Rating: "8"
    Review: "The preformance by Joaquin was fantastic. The music was engaging. "
    Title: "Joker"
    Username: "johndoe123"
  1681266058302
    Rating: "3"
    Review: "I loved every season but the ending is terrible! "
    Title: "Game of Thrones"
    Username: "johndoe123"

Database location: United States (us-central1)
```

(b) RealTime Database

Figure 1: Frontend/Backend Integration

1.7 Future Work

As our models become more computationally intensive, we cannot depend on the server side of the application to process the recommendations on our current architecture. Thus, that is why if we need to expand our resources, we will utilize a GPU hosted on Google Cloud using Cloud Run and migrate our database to a bucked on the Cloud hosted services.

However, the first priority is getting a our user rating mappings and recommendation models to our desired performance specs.

Our next steps are to increase the size of our dataset, improve the aggregated user rating mappings, improve sentiment analysis, and combine all of our separate parts into one cohesive program.

1.8 Conclusion

As mentioned in our initial paper, reddit is currently the most popular place to get movie recommendations. Because it involves people making recommendations, this is highly biased. We want to deploy a smart automated machine learning model that can classify a user's review of a given movie and make a real-time recommendation. Rather than simply looking at ratings or genres, our goal is to create a bias-free system based on the user's preferences.

We want to base our recommendations on what is said about a specific movie because it provides the most insightful information. We will create a graph with a GNN using our web-scraped IMDb data, including summaries and plot synopses, and then use the graph and user reviews to create our recommendation model. Content-based filtering, collaborative filtering, and hybrid filtering have all been investigated. We used the first stages of our TF/IDF NLP preprocessing scripts after gathering a collection of movies to web-scrape.

Sentiment analysis upon use of a pretrained BERT model supplied by HuggingFace has yielded great success upon the additional tuning including the movie reviews we scrape from IMDb. Despite its binary classification, it is still able to classify individual statements with high levels of accuracy of .90 and a converging loss of .20. We hope to transition build atop this model with multi-class aspect sentiment analysis to find specific film elements to connect to movies and users as to create a more personalized character for the recommendation system to consider. We hope to build upon the filtering systems as mentioned in the section 1.2 by utilizing the topics that have positive associations in user reviews.

This model will eventually help us recommend movies to users. The next step is to expand our GNN, improve rating mappings, improve sentiment analysis, and experiment with various other techniques until we are satisfied. One of our priorities was maintaining a realistic industry level workflow. This meant communicating, developing and resting regularly. Through trial and error, we were about to come up with a workflow that used all the latest tools in the industry as it deemed necessary. As a team, we've followed our schedule as intended and our current set of tasks will help us get closer to our goal of creating a recommendation model.

2 Project Management

2.1 Milestone Description

We have updated the milestones and listed Tasks for each milestone below:

2.1.1 Milestone 1: Classification

- Task 1: Gather and clean dataset
- Task 2: Initial neural network architecture
- Task 3: LSTM implementation
- Task 4: Dynamic Web-app initial implementation

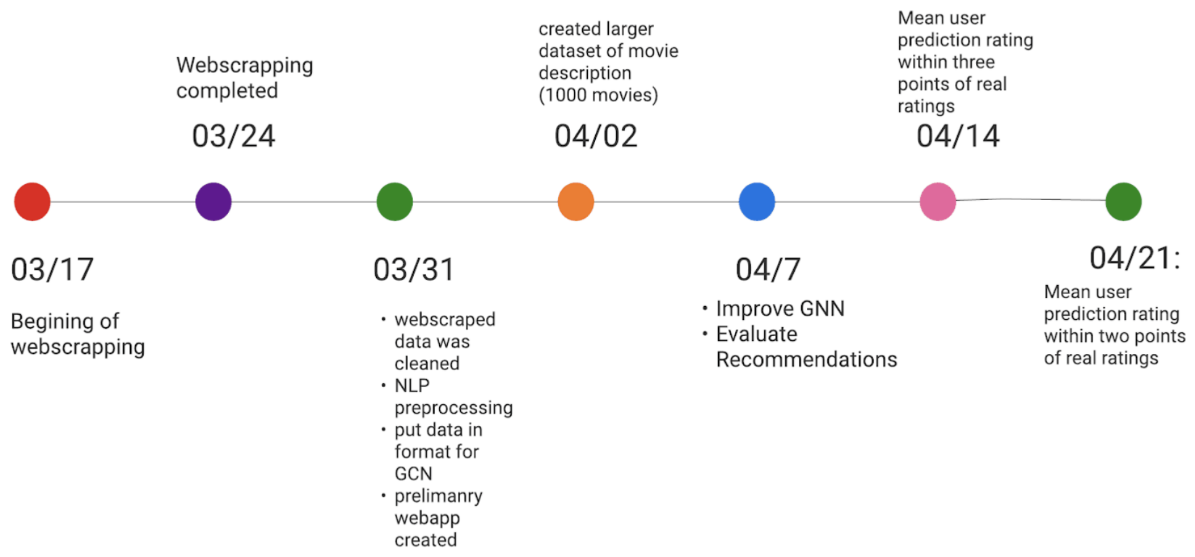
2.1.2 Milestone 2: Recommendation

- Task 4: GNN implementation
- task 5: Mean user prediction rating within three points of real ratings
- Task 6: Mean user prediction rating within two points of real ratings
- Task 7: Project complete

2.2 Project Timeline

Below are dates and the corresponding tasks completed on that date. We've also included a graphic showing out project timeline thus far:

- 03/17: Beginning of web scraping
- 03/24: Web scraping Completed
- 03/31: Cleaned Web scraped data
- 03/31: NLP preprocessing
- 03/31: Put data in format for GCN
- 03/31: Preliminary Web app created
- 04/02: Created Larger dataset of movie descriptions
- 04/07: Improved GNN
- 04/07: Evaluated Recommendations
- 04/14: Mean user prediction within three points of real rating
- 04/14: Mean user prediction within two points of real rating



Webscrapping completed
03/24
created larger dataset of movie description (1000 movies)
04/02
Mean user prediction rating within three points of real ratings
04/14
03/17
Begining of webscrapping
03/31
• webscraped data was cleaned
• NLP preprocessing
• put data in format for GCN
• prelimanry webapp created
04/7
• Improve GNN
• Evaluate Recommendations
04/21:
Mean user prediction rating within two points of real ratings

2.3 Contributions

- Grant Doan - Data cleaning, Sentiment Analysis, User Preference Mapping
- Prayash Joshi- Docker, Website, Backend Database
- Reagan Orth - Data cleaning, GNN, User Preference Mapping
- Ved Patel - Dataset Gathering, Frontend Graphics

2.4 Libraries and Tools Used

Below are the tools we've used in this project. We classified them by development, version control and deployment.

2.4.1 Development

- Python
- PyTorch
- PyG
- pandas
- matplotlib
- numpy
- scikit-learn
- VS Code
- selenium

2.4.2 Version Control

- Gitlab
- Docker

2.4.3 Deployment

- Render

2.5 Risks of Issues

Some of the main risks include R-rated movies that can currently be recommended to children. However, to combat this, we can consider the movie's rating before recommending it. This would ensure that our recommendations reach the correct audience. Another area a user may need clarification on is the availability of the movie. For example, a recommendation not being available for free or even in their location. An easy solution would be to check IMDb availability before informing the user. Additionally, some movies share a name which could be confusing unless we assign a specific id to each movie. Finally, as is the case with many ML models, we cannot guarantee accuracy.

References

[1] Satvik Garg, Pradyumn Pundir, Geetanjali Rathee, P.K. Gupta, Somya Garg, and Saransh Ahlawat. On continuous integration / continuous delivery for automated deployment of machine learning models using mlops. 2021.

[2] Google. Mlops: Continuous delivery and automation pipelines in machine learning nbsp;—nbsp; cloud architecture center nbsp;—nbsp; google cloud, Feb 2023.

[3] Woon-hae Jeong, Se-jun Kim, Doo-soon Park, and Jin Kwak. Performance improvement of a movie recommendation system based on personal propensity and secure collaborative filtering. *Journal of Information Processing Systems*, 9(1):157–172, 2013.

[4] M. V. Koroteev. Bert: A review of applications in natural language processing and understanding, 2021.

[5] Dan Ofer. Machine learning for protein function. 03 2016.

[6] Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. Graph neural networks in recommender systems: A survey, 2020.

[7] Heng Yang and Ke Li. A modularized framework for reproducible aspect-based sentiment analysis. *CoRR*, abs/2208.01368, 2022.