UNIVERSITY OF CALIFORNIA
Los Angeles

Semantic Cinema:
Exploring NLP Applications to Recommender Systems

A thesis submitted in partial satisfaction
of the requirements for the degree
Master of Applied Statistics

by

Omar Moore

2023

ABSTRACT OF THE THESIS

Semantic Cinema:
Exploring NLP Applications to Recommender Systems

by

Omar Moore
Master of Applied Statistics
University of California, Los Angeles, 2023
Professor Frederic R. Schoenberg, Chair

This paper examines numerous machine learning methods in their application for predicting successful item recommendations to users within a network. By evaluating these methods, various industry-standard recommenders are compared in order to construct a baseline model for research purposes. Next we attempt to similarly generate recommendations using a natural language processing, or NLP, approach by leveraging available textual data. This is intended to explore if user preferences can be modeled using the semantic information about a given item. Various approaches are considered to capture this information and the final recommender is ultimately built with Google's Universal Sentence Encoder, an NLP model used to encode full sentences into high-dimension vectors that capture context [CYK18]. In the case of our research, the items being recommended are movies. Each model is trained using a large dataset of film reviews provided by the University of Minnesota. In addition to user reviews, detailed information for each movie ranging from the popularity and genre to descriptive fields like plot overviews are also used as features. With this, we are able to assess a variety of methods for providing recommendations. Ensemble methods are used to build a baseline model with a RMSE of 0.843. Different NLP algorithms are compared and a recommender is successfully built on the Universal Sentence Encoder model having similar performance to our baseline, an RMSE of 0.872

The thesis of Omar Moore is approved.

Tao Gao

Mahtash Esfandiari

Dave Zes

Frederic R. Schoenberg, Committee Chair

University of California, Los Angeles

2023

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

Introduction

The influence of AI-driven recommendation systems on consumption and discovery has become increasingly evident. From suggesting the next binge-worthy TV show to curating personalized book recommendations, these systems are ultimately shaping our choices and preferences. With so much of our technology becoming personalized, it is undeniable these systems play a pivotal role in mediating our access to information.

Put broadly, recommendation engines are systems designed to predict and suggest items that are likely to be of interest to individual users based on their preferences, historical behavior, and similarities to other users or items [LMY12]. Different types of recommenders leverage different strategies. Initially, content-based filtering emerged as a popular approach, relying on the individual users' interactions and preferences to deliver tailored recommendations. By considering the characteristics of items, this technique offered personalized suggestions based on product similarities. Alternatively, collaborative filtering began to cast a wider net, collecting and leveraging insights from the interactions of a multitude of users. Capturing the shared connections among users and items allows these models to intuit implicit user preferences. Other methods also exist to meet more unique needs like knowledge-based systems. These incorporate domain expertise to offer recommendations grounded in a deep understanding of users' needs and preferences [Bur00].

Ultimately what many firms do is employ a hybrid system to recommendations which combine techniques to overcome the limitations of individual approaches. By leveraging the strengths of different models, one can deliver enhanced personalization and overcome ramp up problems occurring when there is limited or no data available. In this paper, we build and evaluate multiple models for recommendation to test various methods. This is done

to identify a baseline model. Recognizing that textual data harbors a wealth of semantic meaning and contextual cues, NLP algorithms are then tested for use in a recommendation system. We deploy Google's Universal Sentence Encoder (USE), a deep-learning NLP model, to numerically translate multi-sentence plot overviews associated with each movie. With high-dimensional embeddings that capture the semantic content of these films, we pursue modeling ways in which contextual information about user preferences can be used to make accurate predictions.

CHAPTER 2

Data

2.1 Source

The data used for this research, known as the MovieLens dataset, comes from the University of Minnesota [HK15] and contain over 20 million ratings covering 27,278 movies. These preferences were collected through the MovieLens web site — a recommender system that asks its users to give movie ratings in order to receive personalized movie recommendations. These ratings were provided by 138,493 users over a twenty-year period. The dataset was generated on October 17, 2016, and is publicly available for download from the GroupLens website.

The primary tables used were one of user ratings and another of descriptive features of the movies rated. These were both merged into a final dataframe for the analysis where each row consisted of a userId, movieId, rating, and characteristics/metadata for the corresponding movie. A snapshot of the data is seen in Table 1 below.

TABLE 1: First six rows of main dataframe.

userId	movieId	title	rating	runtime	release_year	genres	popularity	vote_count	timestamp	overview
131160	21	Get Shorty	3.0	105.0	1995	677	12	305	789652009	Chili Palmer is a ...
85252	7	Sabrina	5.0	113.0	1954	755	7	284	792658421	Linus and David L...
14962	33794	Batman Begins	5.0	174.0	2005	243	28	7511	796546150	Driven by tragedy ...
99581	34624	Babe	5.0	89.0	1995	624	14	756	818249637	Babe is a little pig ...
347003	24	Powder	3.0	111.0	1995	921	12	143	822873600	Harassed by classm...

2.2 Feature Engineering and Selection

All columns were originally imported as character strings so each variable was changed to the appropriate datatype, integers or floats, as needed. The first step was general cleaning, the film titles in our metadata included the release years in parentheses. We split this column into two so that we can convert the films release year into a numeric variable as a feature for modeling. For compatibility and efficiency, we apply label encoding to the "Genre" field to retain this categorical information as numeric labels. A plot of budget's distribution revealed that a majority of these values were zero, suggesting this column may be inaccurate for modeling. Consequently, this variable was dropped from our dataset. A correlation matrix was used to further assist feature selection. Lastly, we sort all rows in an ascending fashion by the timestamp of the review in order to facilitate necessary sequential training. This has been known to significantly improve performance when more than a decade ago, Amazon researchers observed better movie recommendations from neural networks when they sorted the input data chronologically and used it to predict future movie preferences over a short period [Har19]. Temporal dependencies will inherently exist so we want our model to appropriately base predictions for the current time step on previous data only, functionally "predicting forward".

2.3 Exploratory Data Analysis (EDA)

First we can intuit general relationships through a correlation matrix. A heatmap, found in Figure 2.1, reveals a variety of relationships with most being relatively week. We can note expectedly low correlations from both our user and movie identifier variables as these will have little influence on how an item is rated. Comparing other variables to our response we can see a positive correlation with runtime of 0.09 suggesting that longer movies may receive slightly higher ratings. The negative correlation between release year and ratings, suggests that older films are more likely to see lower ratings. Positive correlations with rating can be seen from budget (0.05), popularity (0.08), and vote count (0.14) indicate that more popular

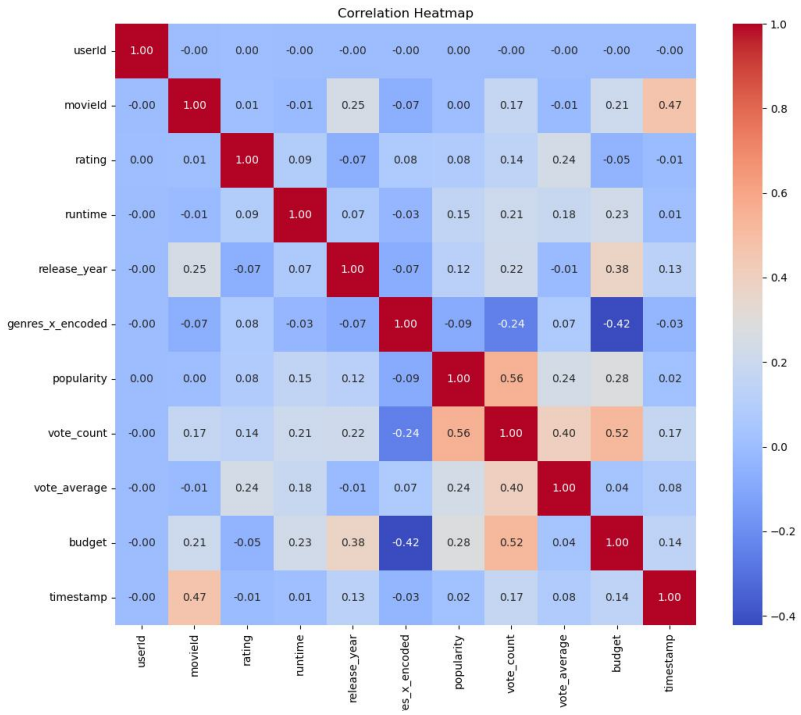

Figure 2.1: A heatmap correlation matrix of the MovieLens variables.

movies with higher budgets might receive higher ratings. The negative correlations between our encoded genres with popularity and vote count also suggest that some genres are less popular or receive less votes. Popularity and Vote Count share a positive relationship (0.56) as these two are directly related.

We next explore the distributions of our variables. Ratings shown in Figure 2.2 are visibly left skewed with comparatively more positive ratings (above 3) than negative ones (below 3). We attempt to square this variable but observe little change when applying this transformation and ultimately keep it as is. Looking at the volume of ratings over time confirms that our dataset covers two decades as expected with a consistently high volume of reviews each year, Figure 2.3. Looking into genres in Figure 2.4, we see high counts across all categories confirming this will be a good feature to include in our model after seeing its correlation to other variables of interest.

One predictor of key interest will be the film overviews/plot synopses that will be used

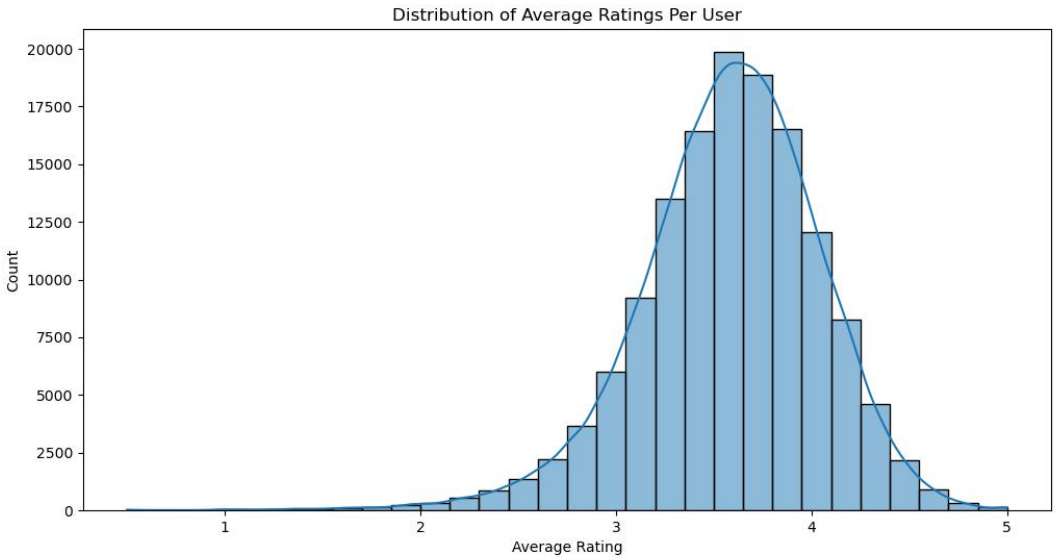

Figure 2.2: Distribution of Average Ratings per User

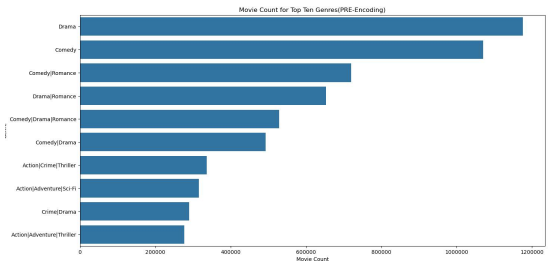

Figure 2.3: Movie Count by Genre

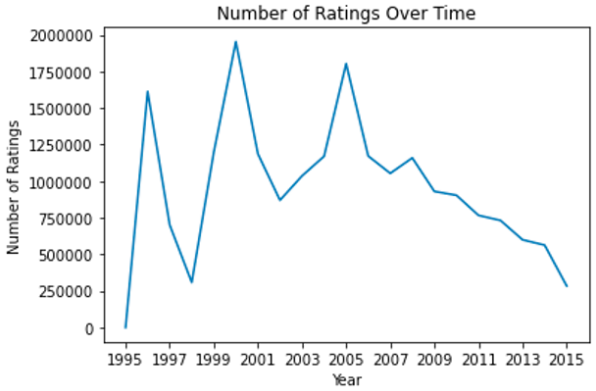

Figure 2.4: Rating Volume Over Time

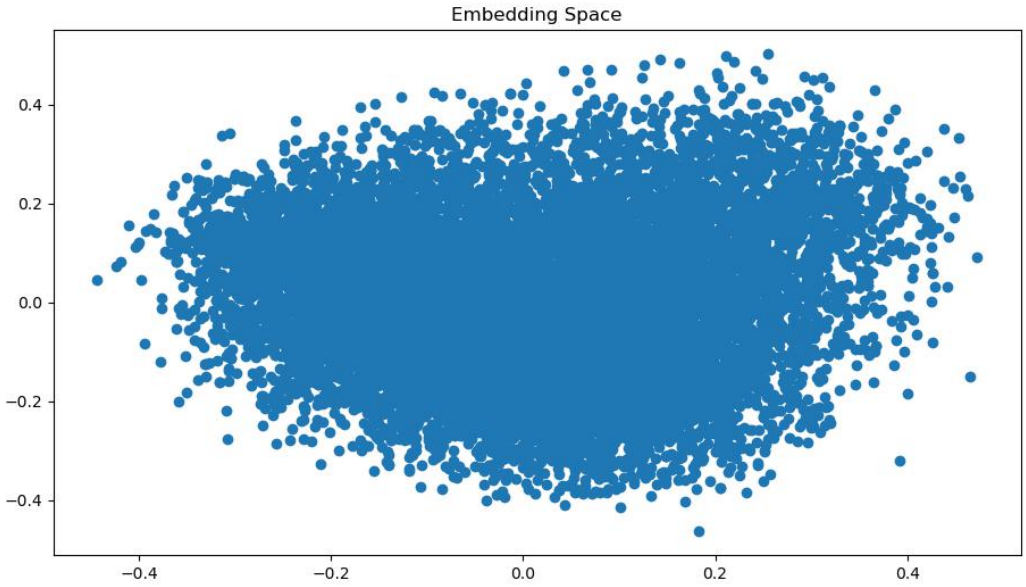

Figure 2.5: Film Synopses in a 2D Space

to capture the semantic information of an item. We used vectorized sentence embeddings to capture these paragraphs in numerical representations. Distance serves as a crucial measurement for comparison. To visualise this concept, the embedding space reflects semantic similarity in the distance between observations. Closer points in the embedding space represent movie overviews that are more similar in meaning or context. A plot of our movie overviews as embeddings in a 2D space is rendered using Principal Component Analysis, PCA, a dimmensionality reduction method where we sort eignevectors with the largest eigenvalues to form a matrix that moves our sample to a new subspace that retains as much as possible of the variation in our original data. This can be seen in Figure 2.5. In this space, the observations for “Batman” and “Superman” would be positioned near each other on the plot. Both of these however, would be far away from “Pride and Prejudice” due to high dissimilarities.

CHAPTER 3

Methods

3.1 Background

Despite becoming indispensable in modern marketplaces, the use of recommender systems dates back to the nineties. One of the pioneering works in this field was the GroupLens project in 1992, which was among the first to leverage matrix factorization with collaborative filtering, still widely used today, to better understand user-item interactions for recommending content. These algorithms in many cases can serve foundational roles to the success of companies. For instance, Netflix's recommendation engine drives 80 percent of content consumption and was recently valued at 1 billion USD for being responsible for the majority of streaming hours [GH16]. Similarly, Spotify's "Made for You" recommendations contribute to one-third of new artist discoveries on their platform. The market size for recommendation engines was valued at 3 billion in 2021 and is projected to grow considerably to 54 billion USD by 2030[Res].

Recall from Chapter 1 the primary type of recommenders. Content-based models analyze item properties while collaborative ones use similarity measures between users to make personalized suggestions. Collaborative recommendation is the most widely used and mature of the technologies [Bur02]. Both methods are frequently combined due to a recurring challenge in the field known as the "Cold Start Problem" which arises when there is limited or no data available to generate a prediction with. These approaches will always demonstrate ramp-up problems since both techniques need a database of ratings and will encounter new items or users. Various ensemble approaches can be used to address this [Chi21]. By combining the strengths of both techniques, a hybrid system can capture complementary information,

overcome data sparsity, and learn from missed data, leading to a more comprehensive understanding of user preferences.

More recently, NLP has risen in popularity due to advancements in deep learning and the availability of large labeled datasets for training. Firms are using NLP models for addressing many practical downstream tasks, however, LLM application to new recommendation paradigms remain relatively unexplored [JLX23]. With endless implications, NLP stands to potentially be a powerful tool at addressing the challenge of product recommendations. Research in this area commonly deals with sentiment classification for latent feature selection in established recommender systems [HK14]. This paper will differ by using a single feature, a multi-sentence plot descriptions, as the basis for prediction to understand user patterns. Using high-dimensional vector representations, called embeddings, to capture the rich semantic and syntactic information present in text data can allow the model to consider not only explicit user-item interactions but also implicit preferences.

3.2 Models

Prior to modeling, subsets were extracted from the merged table using an 80/20 train-test split in a fixed random state for reproducible results. Once in this form, we begin to build our recommenders. To keep a consistent framework for comparison each of the models built was structured to predict the expected rating (output) a user would assign a given movie (inputs).

3.2.1 Model 1: Content-Based Filtering

The first model we construct is intended to serve as our content based recommendation engine, focusing on item characteristics to make suggestions. Against other methods, regression was selected for this model due to being well suited for predicting numerical scores and better ability to handle the outliers from specific user preferences or noisy data. Our content-filtered recommender was ultimately built on top of the XGBRegressor algorithm as this had the best performance. This is a variant of the gradient boosting algorithm for

regression tasks. Similar to other regression models, our goal is to learn a predictive model that can estimate a target variable based on input features but XGBRegressor works by building an ensemble of decision trees iteratively, where each subsequent tree corrects the mistakes of the previous trees by minimizing the residuals. Fine tuning parameters for this model were also preferable since you are able to optimize a specific loss function to minimize the rate of error. Given the goal of accurate rating predictions, MSE, mean squared error, was selected thus making our objective to minimize the average squared difference between the predicted values and the actual target values. MSE is defined as

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3.1}$$

Here, n is the number of samples or data points in the dataset, y is the predicted value, and $\bar{y}$ is the true value. In this case MSE, provides a measure of the average squared deviation of the predicted values from the true values. The squared term allows us to penalize larger differences between predicted and true values more than smaller differences. Gradient descent is then used to iteratively update the parameters in the direction that reduces MSE. Given an initial parameter vector x, the algorithm iteratively updates the parameter values by taking steps in the direction opposite to the gradient of the function at each iteration. The gradient of f(x) is a vector that contains the partial derivatives of f(x) with respect to each component of x. We then adjust the loss function so that our model is trained to minimize the mean squared error (MSE) loss during regression while applying slight adjustments to set the learning rate to 0.9 and the number of decision trees to 80. The update rule for gradient-descent is:

$$x \leftarrow x - \eta \nabla f(x) \tag{3.2}$$

where x represents the updated parameter vector, n (eta) is the learning rate, a positive scalar that determines the step size, and f(x) is the gradient vector. By following the negative gradient, the algorithm seeks to find the steepest descent path towards the minimum of the loss function. This way, the model aims to find the parameter values that produce the

smallest average squared difference between the predicted and true values.

In order to prioritize accurate predictions, Root Mean Squared Error (RMSE) is used as a basis for evaluating the model. This applies the same approach of capturing the differences in the predicted and real values as previously described but uses a square root to scale the error metric so that it is in the same units as the dependent variable, making it easier to interpret. By penalizing large errors more, this method emphasizes outliers in the data more heavily.

3.2.2 Model 2: Collaborative-Based Filtering

We next move on to a collaborative recommendation engine, considering user similarities to determine rating. Utility matrices are an essential method for collaborative-filtered recommenders [HK08]. We are primarily interested in two entities within our data, users and items (movies in our case). Users have preferences for certain items, and these preferences must be teased out of the data. The data itself is represented as a utility matrix, Table 2, giving for each user-item pair, a value that represents what is known about the preference of that user for that item. For our data, integers are scaled from 1–5 to represent the rating that the user gave for that item. Generally, we can assume that the matrix is sparse, meaning that most entries are "unknown" or each user has only used a small fraction of the items. Data sparsity is confirmed in the plot in Figure 3.2. An unknown rating implies that we have no explicit information about the user's preference for the item. This poses a challenge for the next task which is to determine how to measure similarity of users from their rows or columns in the utility matrix. The utility matrix can be viewed as telling us about users or about items, or both. For measuring similarity we considered jaccard, cosine, and euclidean distance but chose cosine since this method doesn't rely on magnitude and our data was high-dimensional and non-binary. Cosine similarity is defined as:

$$cos\theta = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \cdot \|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \cdot \sqrt{\sum_{i=1}^{n} B_i^2}} \tag{3.3}$$

Once this is determined, we compute a condensed distance matrix to store the calculated

TABLE 2: First six rows of utility matrix.

userId	4	8	15	23	26	27	29	32	...	166925
movieId										
319	3.0	0.0	0.0	4.0	0.0	4.0	0.0	0.0	...	0.0
17	0.0	0.0	0.0	3.0	2.0	0.0	0.0	2.0	...	0.0
692	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	...	2.0
45	1.0	0.0	0.0	0.0	3.0	0.0	0.0	0.0	...	0.0
22	0.0	0.0	0.0	0.0	0.0	0.0	4.0	0.0	...	0.0

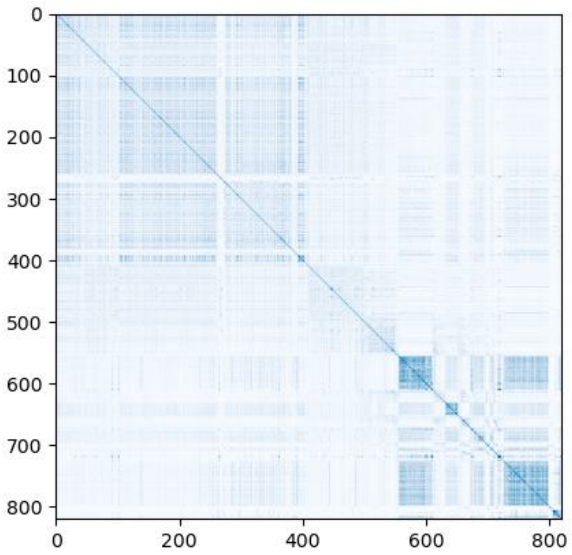

Figure 3.1: Matrix plotted to show Data Sparsity

pairwise cosine distances between movies. This represents the similarity between movies based on user preferences. We lastly obtain a similarity matrix by subtracting the condensed distance matrix from 1. This is done because cosine similarity and distance are inversely related so this produces similarity scores from 0 to 1. With these matrices we are able to identify the ratings of a given user and the similarity score to generate predicted ratings. The prediction equation we use is slightly tailored from the one posited in Neal Lathia's paper which finds the rating for a user-item pair by approximating the dot product between the user's feature vector and the item's feature vector [LHC07]. Instead we have a condensed pairwise matrix, so we take the dot product of user ratings and movie similarity, and then divide it by the sum of movie similarity scores for only those movies that the user has rated, excluding any zero ratings. This was better for overcoming previously mentioned sparsity as well for more robust predictions. Relying solely on the dot product between user and item feature vectors can be heavily influenced by individual ratings whereas our approach considers the aggregated similarity scores from multiple movies. Our predictions (P) can be defined as:

$$P = \frac{U \cdot S}{\sum_i S_i}, \quad \text{where } S_i \in S \text{ and } U_i \neq 0 \tag{3.4}$$

Where U is a vector of the user ratings, S is our movie similarity score to weight the ratings, and our denominator of (Si) is a sum of the movie similarity scores for only those movies the user has seen to act as a normalizing factor. With this function we are able to predict the blanks in our utility matrix for unrated movies.

3.2.3 Model 3: Hybrid System

As previously mentioned, most use cases today rely on a combination of content and collaborative recommendations. The third model we construct is intended to serve as a baseline to resemble many of these traditional models already in production. Naturally, we leverage both of the previously described models to generate our hybrid predictions. We merge the two predictions in a weighted fashion looping through fixed intervals of different weights

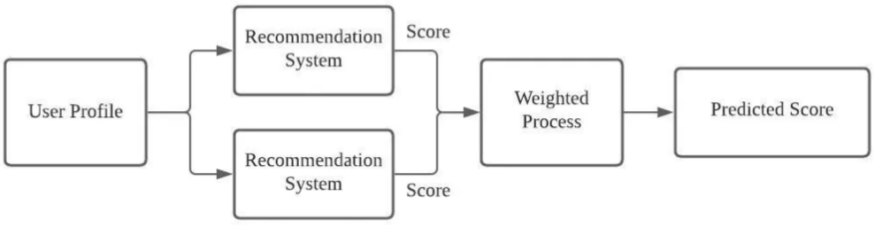

Figure 3.2: Diagram of Hybrid Recommender Strategy

to, similar to our XGBRegressor model, optimize through minimizing our loss function. A diagram representing this model is above in Figure 3.2.

3.2.4 Model 4: Semantic Recommender

Next, we pursue a new natural language processing approach, focusing on the semantic context and meanings of each film to predict a rating. When testing this model we considered various NLP methods for interpreting text. The initial challenge was identifying the best method of translating characters to numeric arrays while retaining their meanings in the body of text. As with many NLP tasks, we knew that vectorizing text would generally be a good place to start. The first run was a model using Term Frequency - Inverse Document Frequency (TF-IDF) packages. This method is able to generate numerical representations to reflect the importance of a term in a document within a a larger body of documents with a fairly simple approach. TF is a measure of the frequency of a term within a document relative to the total number of terms in the document.

$$TF(t,d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d} \tag{3.5}$$

IDF instead assesses the term across all bodies of text used to determine its significance by assigning more weight to unique or rare words and less weight to common ones. Similarly, the Inverse Document Frequency (IDF) of a term t in a collection of documents D is calculated as:

$$IDF(t, D) = \log_e \left(\frac{\text{Total number of documents in the collection } D}{\text{Number of documents containing the term } t} \right) \tag{3.6}$$

Here, a logarithmic transformation is applied to provide a scaling effect and enhance interpretability. The Term Frequency-Inverse Document Frequency (TF-IDF) combines a term's local importance (TF) with its global importance (IDF) by generating a score from their product:

$$TF - IDF(t, d, D) = TF(t, d) \cdot IDF(t, D) \tag{3.7}$$

This method aims to capture a term's importance based on both its frequency within a document and its rarity across the entire document collection. after testing a simple item-to-item recommender we quickly realized limitations to this method. The most notable of these issues being a disregard for word order. This approach treats each given word independently, regardless of the words before and after it or the relationship to those words. It also lacks any understanding of the context a word is used in, so two documents using the same words in different contexts will end up with similar TF-IDF scores. Examples of these less-than-ideal recommendations can be found in Figure 3.2 and 3.3. We see that after embedding "James Bond", the nearest recommendations are all 007 related films. When trying to use "Beethoven" however we receive a mix of family friendly pictures and music documentaries as the algorithm is unsure of what context to consider. These barriers ultimately make the model very limited in actually understanding the semantic meaning of a film synopsis. A TF-IDF model can still be valuable for identifying and prioritizing key terms but are highly restricted in capturing any semantic meaning from full sentences or even a single term in the context of its surrounding terms.

As more comprehensive models were researched the current popularity and fervor surrounding generative artificial intelligence naturally brought me to Large Language Models, or LLMs. These differed significantly by using deep neural networks to learn contextualized word embeddings. LLMs encode words in the context of their surrounding words, capturing more nuanced and contextual representations of text. Various transformer models were

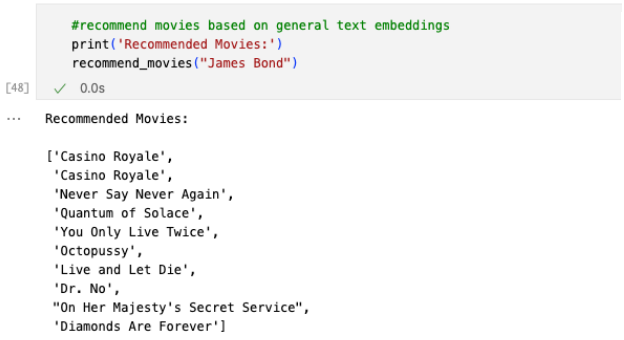

Figure 3.3: Recs for "James Bond"

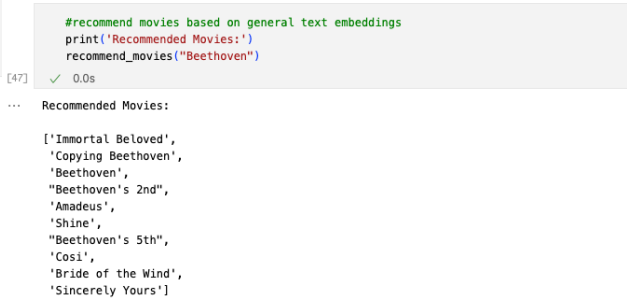

Figure 3.4: Recs for "Beethoven"

considered. BERT (Bidirectional Encoder Representations from Transformers) and GPT3 (Generalized Pretrained Transformer) were the first to be evaluated. Both models were able to generate the sentence encodings to make predictions but BERT was preferred here given its ability to interpret words from both directions (bidirectionally) against GPT's unidirectional encoding. Both are able to learn contextual representations through considerable training with the use of transformer architecture. Transformers are a fairly new type of neural network consisting of an encoder and decoder. These were first introduced in 2017 through the paper "Attention Is All You Need" [VSP17].

The key to capturing the contextual meaning of a sentence lies in the self-attention mechanism, which allows the model to weigh the importance of words within a sentence based on varying relationships and dependencies. The self-attention mechanism requires three matrices, Q (query), K (key), and V (value), such that $Q = XW^Q$, $K = XW^K$, and $V = XW^V$ with the weight matrices W^* learned during training. A self-attention matrix Z with a new embedding for each word is then defined by:

$$Z = softmax\left(\frac{QK^T}{\sqrt{d}}\right)V,$$

Where d is the embedding dimension of each word. The matrix $softmax\left(\frac{QK^t}{\sqrt{d}}\right)$ contains the correlations between each pair of words in the sentence. The self-attention scores are computed by taking the dot product between the query and key vectors, scaled by a factor of the square root of the dimension of the key vectors. The attention scores determine the importance or relevance of each word to other words in the sequence. Higher scores indicate

stronger relationships or dependencies. Matrix V essentially adapts the correlation matrix to different tasks.

Following self-attention, the transformer includes a feed-forward neural network (FFN) consisting of two linear transformations with an activation function to transform the weighted sums of inputs to an output value. In testing this model, we use Rectified Linear Unit or ReLU for computational efficiency. Defined as $ReLU(x) = \max(0, x)$, this function is used for learning nonlinear relationships and countering gradient vanishing problems often associated with back-propagation in deep neural networks.

The transformer architecture uses multi-head-attention to capture relationships between words by focusing on different parts of the input simultaneously, running through self-attention mechanisms in parallel. FFNs then further process the representations. These components are stacked in layers, with each layer taking the output of the previous layer as input.

As a result, the transformer learns optimal weight matrices and parameters to capture contextual information and relationships within the input sequence, enabling it to excel in various NLP tasks. By employing these methods, LLMs can quickly become computationally expensive. Using LLMs proved to be an arduous and time consuming process when performed on a local machine. The datasets used in our research were then moved into a cloud environment to be accessed by an AWS Sagemaker notebook with more computing resources. Even using BERT in this environment to generate embeddinggs for our full dataset still proved to be a heavy task, taking multiple hours to execute. To overcome this we changed our embedding model from BERT to Google's Universal Sentence Encoder, or USE. This model is designed to encode entire sentences into fixed-size vectors, making it more straightforward for various downstream tasks as well as generally smaller and lighter in memory compared to BERT. USE is similarly bidirectional. The transformer based variant employs a self attention method, so the encoder will use a combination of recurrent and convolutional neural networks to process input text. The recurrent neural network (RNN) is in place to deal with sequential data and capture the relationships between items in a sequence. This processes the words and subwords one at a time to incorporate information

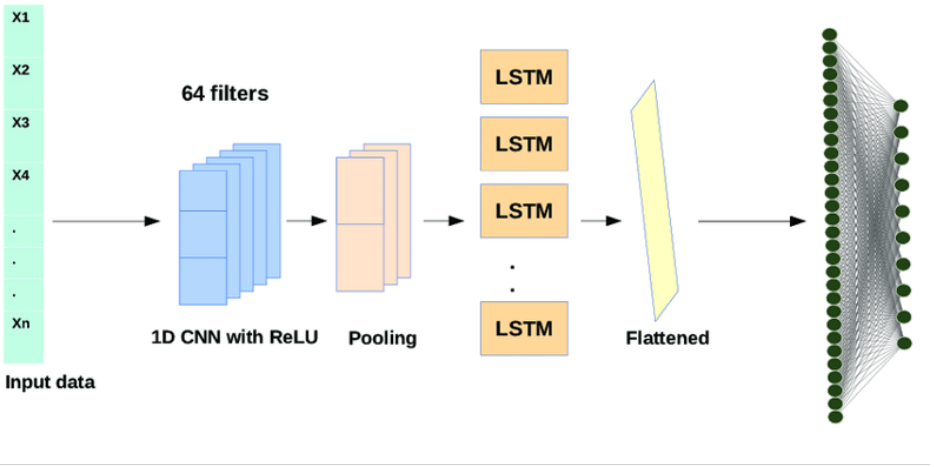

Figure 3.5: Tranformer Learning: Feed Forward Neural Network

from the previous words for context when tokenizing a term. Convolutional neural networks (CNN) are needed for extracting local patterns. The different patterns identified by the RNN and CNN are combined to form meaningful sentence embeddings. A diagram of this model is shown above in Figure 3.5.

With our encoder model determined, we begin by generating an embedding for every unique film in our dataset and store them in an embedding dictionary to be referenced for our predictions so that we can avoid potential memory issues. The goal now is to similarly generate an embedding for each user to be compared with a given movie. Multiple approaches are used to capture users in a similar format. We attempted a genre-targeted approach where we subset all movies a user had seen with overlapping/similar genres to the input movie to calculate an aggregate embedding but this approach encountered even higher sparsity. Using the full viewing history of a user to generate average embeddings yielded the best results. We weighted these averages by the ratings received by each movie. With both our movies and users now represented as vectors of matching dimensions, we use cosine similarity to determine the likeness of both and predict a rating that is weighted according to the similarity score.

3.3 Cross Validation

To control for the temporal dependencies present in our data, time-series cross-validation was employed to evaluate the performance of predictive models on time-ordered data. Essentially, this allows us to asses the models' performance over time across changing conditions. This was first structured as a sliding window where training "window" and test "horizon" were of fixed intervals and moved across the data. Ultimately, the models were cross validated using a rolling origin style to better replicate the real world, where only the origin was fixed and the training and test sets added new data as they moved forward in time.

CHAPTER 4

Findings

A working recommender was successfully built using each of the methods described in the previous chapter. Using XGBoost we were able to generate content-based recommendations using features of a given movie. While this model had the highest error rate, it was still less than 1, on a scale of 1 to 5, proving reasonably effective. Accuracy improved with the collaborative model, likely due to being able to capture implicit preferences in user behavior patterns through user/item similarity. Both models were tested on "Cold Start" cases where predictions were generated for newly occurring items and users. These predictions always had higher than average error. To reduce this, we modified both of our models to default to system averages/mean votes when encountering these cases which improved performance slightly. The code used in testing is published on the central repository for this thesis, URL can be found in the references [OM23].

The next step was to combine these two models. If we successfully leverage both methods to compensate for each other's shortcomings, our hybrid model should prove to have the highest accuracy. We identify the optimal weights to apply to each prediction in a straightforward manner, by iterating through different weight combinations and recording their error rates at each fold. A plot of the changing RMSEs is shown below in Figure 4.1. This reveals the ideal weight distribution is 30/70 for content/collaborative based predictions, expectedly being weighted more heavily toward the better model.

Once the weights for a hybrid model are finalized to serve as a baseline, we can move on to the Semantic/NLP recommender. Proximate movies were selected by computing the cosine similarities directly as opposed to using KNeighborsRegressor which proved more time consuming to generate the distance calculations. Ultimately we are able to produce a

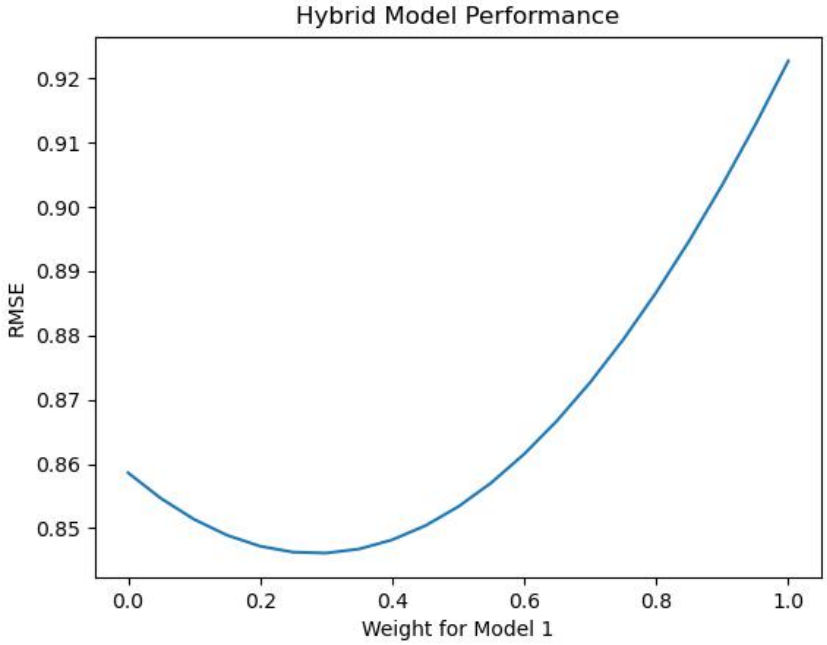

Figure 4.1: Hybrid Performance Across Different Weights

TABLE 3: Model Performance Compared.

	Model	Description	RMSE	RMSE_CV
1	Content-Filtered	XGBRegressor on movie features	0.935	0.976
2	Collaborative-Filtered	User-Item cosine similarities	0.857	0.894
3	Hybrid System	Weighted combination of models 1 and 2	0.843	0.861
4	Semantic Recommender	Translates plot overviews as embeddings	0.872	0.880

model of similar performance, having slightly lower accuracy than our baseline. This proves that a model can learn user preferences from the contextual meanings of those items/movies consumed. A comparison of predictive performance across each of the models can be seen in in the below table.

Finally we attempt to gauge the generalizability of our baseline and semantic models through time-series cross validation. The mean error rates across each fold are also included in the results table. In CV, we initialize training our model on a subset of the earliest data and move forward across folds in time adding new data to our train set each iteration.

Our initial training size is 800,000 and moves forward by 10,000 rows at each fold. Over time, little to no change was seen in volatility, however both models enjoyed relatively stable accuracy, generally having their RMSE fall between 0.5 and 2. Constant performance is good, as it implies the model generalizes well to unseen data.

CHAPTER 5

Discussion

5.1 Computational Limitations

The computational barriers encountered in this research were primarily rooted in the scale of the dataset and the resource-intensive nature of the machine learning models. Initially conducted on a local machine, the models faced significant challenges due to the sheer size of the data, comprising 16 million rows. The local environment struggled to handle the computational demands, resulting in crashes and timeouts during model execution. To overcome these limitations, the research transitioned to AWS SageMaker, leveraging the cloud platform's scalability. However, even in this cloud environment, challenges persisted. The kernel would frequently die due to memory constraints, prompting the exploration of various notebook instances.

In the quest for improved computational efficiency, comparisons were made between memory-optimized and compute-optimized instances. However, both types proved insufficient to handle the computational demands of the research. Ultimately, the models were executed on an advanced computing notebook instance, a resource-intensive option that provided the necessary computational power. Despite this transition, tests remained slow, prompting further investigation into the root cause. It was identified that a significant portion of the processing time was attributed to the first model using XGBoost, particularly the gradient descent process learning each tree sequentially. To address this bottleneck, a strategic move was made to an advanced compute-optimized notebook with GPU support, enhancing the speed and efficiency of the model training process. Although these adjustments were effective, it's noteworthy that such tests, especially those involving GPU usage,

are inherently time-consuming and come with increased computational expenses.

5.2 Final Thoughts

As research in NLP grows, applications to product recommendations will only increase. This paper primarily employs the Universal Sentence Encoder (USE), however, future studies could certainly benefit in using more advanced techniques and transformer-based models. These would allow more strategic autonomy through the fine tuning of hyper parameters not used in USE. predicting with larger models could provide a deeper contextual understanding of movie overviews, and record more accurate semantic representations. Moreover, additional textual fields being included in the modeling would only improve performance. Credits, Cast, keywords and other fields can similarly be embedded to understand preferences.

In selecting a baseline mode, it would have been interesting to compare different types of hybrid models. Depending on the availability of data we collaborative filtering could be combined with various knowledge systems using pre-acquired information about the user to overcome any ramp up issues. Enriching the user modeling aspect of collaborative filtering by including additional user-specific features could involve factors like demographics, viewing history, or implicit feedback.

Data preparation could have been approached differently before beginning to fit any models. Matrix normalization would have been a strategy to capture user behavior, greater tendencies to rate high/low. Essentially, for each of the n users subtract their average rating for items from their rating for i. Average the difference for those users who have rated I, and then add this average to the average rating that U gives for all items. This would adjust the estimate in the case that U tends to give very high or very low ratings, or a large fraction of the similar users who rated i (of which there may be only a few) are users who tend to rate very high or very low. Future research could involve experimenting with different normalization strategies and assessing their impact on recommendation accuracy. Normalization methods may contribute to more precise estimates, especially in cases where similar users who rated a particular item are skewed toward extreme ratings. Comparing

performance across subsets of different sparsity would also be interesting. In this paper, we treat blanks as a 0 value. However, this choice is questionable, since it has the effect of treating the lack of a rating as more similar to disliking the movie than liking it. Modifying our utility matrix to account for this would also potentially raise accuracy.

REFERENCES

[Bur00] Robin Burke. "Knowledge-based recommender systems." *Encyclopedia of library and information systems*, **69**(Supplement 32):175–186, 2000.

[Bur02] Robin Burke. "Hybrid recommender systems: Survey and experiments." *User modeling and user-adapted interaction*, **12**:331–370, 2002.

[Chi21] Jeffrey Chiang. "Types of Hybrid Recommendation System." *Analytics Vidhya*, 2021.

[CYK18] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. "Universal sentence encoder." *arXiv preprint arXiv:1803.11175*, 2018.

[GH16] Carlos A. Gomez-Uribe and Neil Hunt. "The Netflix Recommender System: Algorithms, Business Value, and Innovation." *ACM Trans. Manage. Inf. Syst.*, **6**(4), dec 2016.

[Har19] Larry Hardesty. "the-history-of-amazons-recommendation-algorithm." *Amazon Science*, 2019.

[HK08] Jiawei Han and Micheline Kamber. "Data Mining: Concepts and Techniques." 2008.

[HK14] Savita Harer and Sandeep Kadam. "Sentiment classification and feature based summarization of movie reviews in mobile environment." *International Journal of Computer Applications*, **100**(1):30–35, 2014.

[HK15] F. Maxwell Harper and Joseph A. Konstan. "The MovieLens Datasets: History and Context." *ACM Trans. Interact. Intell. Syst.*, **5**(4):19, 2015.

[JLX23] Jianchao Ji, Zelong Li, Shuyuan Xu, Wenyue Hua, Yingqiang Ge, Juntao Tan, and Yongfeng Zhang. "Genrec: Large language model for generative recommendation." *arXiv e-prints*, pp. arXiv–2307, 2023.

[LHC07] Neal Lathia, Stephen Hailes, and Licia Capra. "Private Distributed Collaborative Filtering Using Estimated Concordance Measures." p. 1–8, 2007.

[LMY12] Linyuan Lü, Matúš Medo, Chi Ho Yeung, Yi-Cheng Zhang, Zi-Ke Zhang, and Tao Zhou. "Recommender systems." *Physics Reports*, **519**(1):1–49, 2012. Recommender Systems.

[OM23] OM. "MAS Thesis Code Repository - Omar Moore." Technical report, 2023.

[Res] Straits Research. "Recommendation Engines Market." *Straits Research*.

[VSP17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need." *Advances in neural information processing systems*, **30**, 2017.