Milestone: The Love Story of Netflix and IMDB

Ishita Bhandari
Information School
University of Washington
Seattle, WA 98195
ishita10@uw.edu

Ajinkya Sheth
Information School
University of Washington
Seattle, WA 98195
ajinkya@uw.edu

Octavian-Vlad Murad
Paul G. Allen School of Computer Science & Engineering
University of Washington
Seattle, WA 98195
ovmurad@uw.edu

Abstract

In an age where there is simply too much information for humans to process during their lifetimes, recommender systems have become an ubiquitous tool designed to prevent information overload, increase customer satisfaction, and boost businesses' income and efficiency. One of the tasks traditionally associated with recommender systems is that of offering movie suggestions tailored to the viewer's preferences. This task rose to fame in 2006 when Netflix posted a $1.000.000 reward for whoever writes the best movie recommendation algorithm. In 2009, the winners of the challenge proposed a purely Collaborative Filtering solution. However, since multiple sources of information about movies are available and since the Machine Learning field has come a long way since 2009, we design a hybrid recommender system based on deep neural networks which leverages data from a complementary source of movie information. More specifically, we propose a deep learning algorithm which combines a Content-Based Filtering part learned from movie data available on IMDB with a Collaborative-Filtering part that is learned from the Netflix Challenge data. By combining two complementary sources of data and two complementary recommendation paradigms, as well as by capitalizing on the awesome power of neural networks, we are able to outperform the winning algorithm of Netflix Challenge by 2.7% in term of RMSE.

1 Introduction and Previous Work

Recommender Systems gather information about the users' preferences for a set of items with the purpose of recommending new ones. Modern RS generally combine various sources of information(e.g. user behavior, tags, intrinsic characteristics of the items, ratings collected from the users) in order to provide accurate, well-rounded, and novel recommendations. Collaborative Filtering methods are a particularly successful and popular class of RS. This type of methods use the preferences of like-minded users to make suggestions. Most often, CF methods rely on one of two principles. They could use a similarity measure between users and/or items in order to suggest similar items to the ones that the user rates highly or to suggest items rated highly by similar users. Alternatively, a latent vector space can be learned in which the dimensions correspond to abstract concepts that describe the users and the items and in which similar users and items are naturally grouped together. However, pure CF methods suffer from the infamous cold-start problem[1]. The cold-start problem refers to the inability of a CF algorithm to make reliable recommendations when it doesn't have

Preprint. Under review.

enough available ratings for an item or when it encounters a new user. A common solution[1] for this problem is creating a hybrid RS that has an additional Content-Based Filtering component. CBF methods leverage item features in order to make recommendations using the principle that if a user tends to rate items with certain features highly, then he is likely to enjoy other items with similar features. Since they rely solely on features to make recommendations, CBF methods are immune to the popularity of an item, thus partially alleviating the cold-start problem of CF methods when item ratings are lacking. Furthermore, it can help the equivalent problem when we encounter a new user by recommending items that fit the users' predefined preferences which we can collect as part of a sign up process or infer through the users' demographics.

Learning embedding vectors in a deep learning framework has become a very popular machine learning strategy, mainly due to their very successful usage in the Natural Language Processing field[2]. However, in recent years, embedding vectors have also been used for prediction tasks using categorical features[3], as well as for CF systems[4]. Intuitively, an embedding is a map from some categorical entities to an Euclidean space. In the context of deep learning, this map represents a matrix which associates a real valued row vector to each category and which is learned through optimizing a neural network, in a supervised setting, with respect to an error function. Concretely, in the case of CF for movies, the entities are users and movies which we aim to embed into a latent space that would capture meaningful concepts such as genres, quality of acting, tone, etc. Although the algorithm is agnostic to these concepts, we encourage it to learn useful representations by using the latent vectors to predict how a user is going to rate a movie through a NN. Then, using the difference between the prediction and the actual rating, we create an error signal which we back-propagate through the NN and then use to update the embedding map. Once the embedding map and the NN are trained, they can take as input movie-user pairs and output the rating prediction.

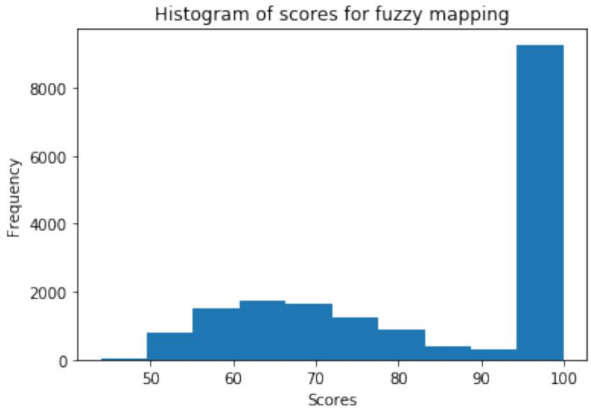
Auto-encoders are another machine learning algorithm enjoying recent success and popularity[5]. An auto-encoder is a neural network comprising of two parts: the encoder, E, which compresses the input x from a large dimensional space to a much smaller one, and the decoding part, D, which takes the compressed representation and maps it back to the original large space. The objective of an auto-encoder is to reduce the reconstruction error $\|D(E(x)) - x\|$ over all datapoints. By doing so, we ensure that the compressed representation $E(x)$ learns to be as expressive as possible. Auto-encoders have been used for CBF[4] by learning a compressed representation of item features which, along with some similarity metric, was then utilized to cluster alike items. Our algorithm relies on auto-encoders in two ways. Firstly, the IMDB data contains movie overviews which we first transform to tf-idf features and then compress using an auto-encoder. Secondly, the CBF component of our hybrid algorithm uses an auto-encoder inspired bottleneck architecture to compress the movie representation into a expressive lower dimensional space.

Providing movie suggestions is one of the tasks traditionally associated with RS. The task gained popularity in 2006 when Netflix posted a $1.000.000 reward for whoever is able to create the best movie RS. The Netflix Challenge data contains approximately 110 million ratings given by 17,770 users to 480,189 movies dating from 1890 to 2005. The organizers evaluated the performance of the submitted solutions by computing the RMSE achieved on a secret held-out set. The competition would enter its final call stage(only one more month left to submit solutions) when a submission would improve the RMSE of Cinematch, Netflix's original RS system, by 10% or more. The Cinematch algorithm achieves a RMSE of 0.9525 on the held-out set, so the final solutions would have to perform better than a 0.8572 RMSE. On September 19th 2009, the BellKor's Pragmatic Chaos team was announced the winner of the competition having achieved an RMSE of 0.8567. Their solution[6] is an ad hoc CF algorithm which reflects the team's deep understanding of the task, but lacks in generalizablity and simplicity.

However, we do not have access to the actual held-out dataset used by Netflix to evaluate the submissions for the competition. Instead, we randomly select 5% of the Netflix Data that we will not use for training as our test-bench for computing the RMSE that our models achieve. Despite having less data, our hybrid recommendation system obtains an RMSE of 0.8335 which is a 2.7% improvement on the winners' RMSE. Furthermore, we achieve this RMSE with minimal hyperparameter tuning and computational power and conjecture that using our general framework with more attention to detail and larger models could further improve the results.

The contributions of our paper are the following:

Figure 1: Distribution of fuzzy matching scores



- We present a simple, yet powerful hybrid RS for movie recommendations that requires minimal feature engineering.
- As a byproduct of our design, we obtain useful compressed representations of the movies and the users.
- We achieve a 2.7% improvement of RMSE when compared to the winners of the Netflix challenge.
- We successfully combine two datasets containing complementary data, even though the matches between the two datasets is far perfect. Furthermore, we present a method for combining categorical and continuous features, a task which is notoriously difficult.

2 Data

For our project, we plan to utilize two datasets: The Netflix Prize Dataset[7] and The Movies Dataset[8]. The Netflix data contains approximately 110 million ratings given by 17,770 users to 480,189 movies dating from 1890 to 2005. The Movies Dataset comprises of movie descriptors such as crew, plot keywords, budget, release date, etc., for 45,000 movies dated up to 2017. In order to leverage the content based information, we need to map each Netflix movie to its corresponding entry in the Movies Dataset.

Matching the databases is problematic due to the fact that the titles of the same movie can differ between the two databases and that some movies in the Netflix Database might not be present in the Movies Database. To partially handle these problems, we use the Python package *fuzzywuzzy* to compute the Edit Distance between movie titles. Informally, the Edit Distance between two words is the minimum number of single-character edits required to change one string into the other. We join the two databases by matching on the entry with the highest score from the other database. The distribution of the scores, which are on a scale from 0 to 100, is given in Figure 1. Because this is not enough to ensure that the matches are correct and because in some cases such a match might not even exist, we will further account for the uncertainty in the matches through the way we build our model, as described in Section 3.

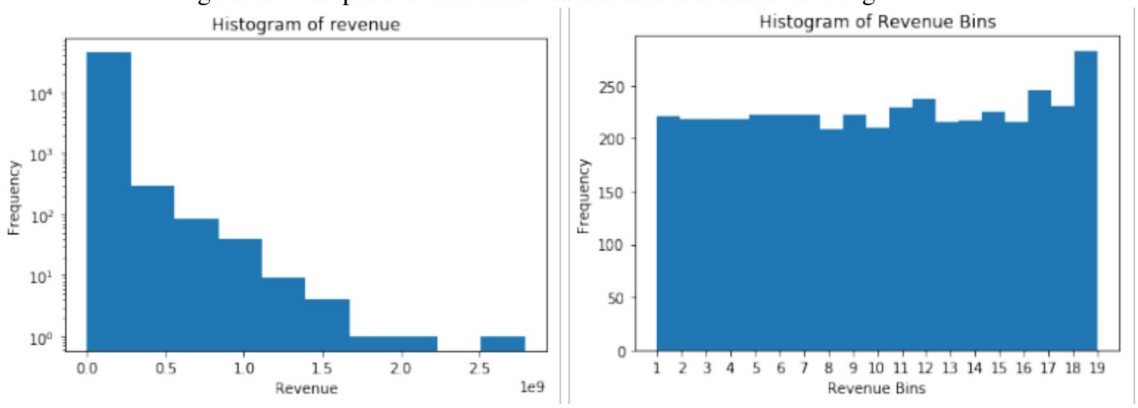
We have extracted movie CB features from the Movies Dataset, as well as some simple statistics from the Netflix Dataset. These features are shown in Table 1.

All the numerical features have been standardized and thus are floats. All the categorical features are converted to integer ids or list of ids if the features are sets(such as genres or keywords). Although we converted all numerical values to floats, we selected some of them to also be converted to discrete variables through histogram binning. For each such variable we used 20 bins and computed the bin sizes in a way that makes the bin frequency as even as possible. We choose to do this for two reasons. First of all, some numerical values such as revenue and income follow power laws and are thus unsuitable for standardization which works well only for normally distributed features. We offer an example in Figure 2. Second of all, certain features such as the release year or the popularity of a

Feature Name	Type	Range
release_year_bin	int	[0, 19]
budget_bin	int	[0, 19]
revenue_bin	int	[0, 19]
runtime_bin	int	[0, 19]
popularity_bin	int	[0, 19]
language	int	[0, 62]
director	int	[0, 6887]
writer	set of 4 ints	[0, 3870]
producer	int	[0, 4569]
actors	int	[0, 4411]
genres	set of int	[0, 20]
keywords	set of int	[0, 13076]
release_year	float	normalized float
budget	float	normalized float
revenue	float	normalized float
runtime	float	normalized float
vote_average_movies	float	normalized float
vote_count_movies	float	normalized float
ratings_average_movies	float	normalized float
ratings_std_movies	float	normalized float
vote_average_netflix	float	normalized float
vote_count_netflix	float	normalized float
vote_std_netflix	float	normalized float
match_score	float	normalized float

Table 1: Feature extracted from the Movies Databases

Figure 2: Example of the revenue feature before and after binning.



movie should be considered on a coarser scale than their original values. For example, by binning the release year we ignore small differences in this feature which are probably irrelevant for our prediction task. Instead, we consider longer intervals of time which makes sense in this case since a user might have stronger preferences when it comes to movies from different decades.

In addition to the features presented in Table 1, we also extracted from the Movies Database the overview of each movie. This feature is a paragraph of text representing a short description of the movie. We transform each overview into tf-idf features which we will further process as described in Section 3.

Our features are largely divided into four main categories:

1. **Human Features:** actors, directors, and other people of importance involved in the movie. Most users prefer certain actors and directors more than others, while some directors and writers tend to produce better movies. Thus, we hypothesize that adding such information to

our model will add to the quality of our predictions. We restrict our attention to the most important 4 actors of the movie in order keep their total number small while ensuring that they are relevant to the user's preferences.

2. **Descriptive Features:** genres, keywords, overview. Most users have a specific preference towards certain genres. Furthermore, keywords and plot descriptions could play an important role in determining the kind of scripts that a user tends to rate higher or lower.

3. **Global Evaluation Features:** mean ratings, standard deviation, popularity index. Mean ratings of movies from both the Netflix data and the Movies database gives a baseline of how well a movie was received by the audience on average. This in it of itself is an important feature when trying to predict how a user is going to rate a movie, but it can also be used in making suggestions to some quirky users how might actually enjoy watching horrible movies.

4. **Commercial and Temporal Features:** runtime length, revenue, budget, release date. Features such as the budget and revenue describe the commercial aspect of movies, while the release date and runtimelength capture temporal characteristics. A movie with a greater budget may be marketed better and a movie with a specific release date may cater to certain audiences more. Regardless, these features impact the overall perception of the movies and influence how they are received by the users.

While we have extracted most of these raw features, a major challenge we foresee is transforming this array of categorical and continuous variables into the feature vectors m_i and u_j we described in the Algorithm Section. The main problems are transforming categorical features into values or vectors in $\mathcal{R}^n$, and ensuring that some features don't have more impact on the outcome than others(for example, if we derive TF-IDF features from the movie description, then these would be large vectors in $\mathcal{R}^n$, while the budget is going to be represented by a single value in $\mathcal{R}$).

As a starting point in deriving movie feature vectors, we plan to standardize numerical variables such as budget, runtime, revenue, popularity, etc, in order to bring them on the same scale. For certain categorical features such as actors or directors we could do the following: if actor A has acted in movies M1, M2 and M3, then assign the average rating of movies M1, M2 and M3 to that actor. These kind of features would again be standardized. An alternative solution would be to use embedding layers for all categorical features which we would train in a similar manner to the method described in the Algorithm Section for learning movie and user embeddings. For the movie descriptions, we will use TF-IDF features. All these features will be gathered into a vector m_i in $\mathcal{R}^n$ which will describe the content of a movie.

3 Algorithm

The algorithm we propose is a deep learning based hybrid RS comprising of a CBF and a CF part. We successfully combine the awesome power of recent advancements in deep learning such as embedding layers and auto-encoders into a RS that leverages and combines content data extracted from the Movies Database and collaborative data gathered from the Netflix Database. This system would be used to predict the rating that a Netflix user will assign to an unseen movie. Additionally, the compact intermediate representations used by the deep learning model could potentially be utilized as a Euclidean space where movies and users with similar properties are clustered together.

As described in the Data Section, one of the biggest challenges we faced was merging the Movies Database with the Netflix Database, given that movie titles between the two databases might not match or that a movie in the Netflix Database might not be present in the Movies Database. Furthermore, the content data from the Movies Database comprises of both categorical and numerical features, making it challenging to model the two types of data in a common space. Thus, we begin by presenting two models which we used to integrate the content data from the Movies Database in a matter that tackles the aforementioned problems. Finally, once we obtain the content data through the first two models, we plug it into the hybrid recommender system which we will describe at the end of the section. Here we discuss the algorithms at a general level, while the implementation details are given in Implementation section.

Figure 3: Auto-encoder used to reduce the dimensionality of tfidf features

To fix some notation, we will use: i to index movies, j to index users, U the number of users, M the number of movies, r_{ij} to denote the real rating given to movie i by user j, $\hat{r_{ij}}$ to denote the predicted rating given to movie i by user j, and $x|y$ to denote the concatenation of the vectors x and y.

A substantial amount of information about the content of the movie is present in the movie's overview which is a short text describing what the movie is about. In order to integrate this information, we first transform the overview into tf-idf vectors which end up being vectors of size 815. Using vectors of this size would be too computationally expensive and it would drown out the signal coming from all the other movie features which have much lower dimensionality. Hence, we train auto-encoder to reduce the dimensionality of the tf-idf features to 32. More specifically, if we denote o_i to be the tf-idf features of the overview of movie i, our objective is to train an encoder-decoder pair (E, D) which minimizes the reconstruction error $\sum_{i=i}^{M} \|D(E(o_i)) - o_i\|$, where $E : \mathcal{R}^{815} \rightarrow \mathcal{R}^{32}$ and $D : \mathcal{R}^{32} \rightarrow \mathcal{R}^{815}$ are deep neural networks. Once the training is completed, we take $E(o_i) \in \mathcal{R}^{32}$ as the representation of the overview and will denote it by $E_o(i)$. The auto-encoder architecture is presented in Figure 3.

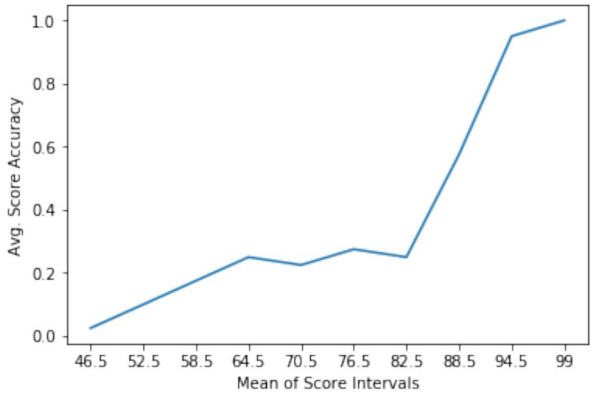
The next step is the integration of the categorical features we gathered or derived from the Movies Database as described in the Data Section. We achieve this by learning embedding maps for each of the categorical features. Namely, for each categorical feature f, which can take on discrete values $\{1, \ldots, F\}$, we aim to train an embedding map $G_f : \{1, \ldots, F\} \rightarrow R^{N_f}$, where N_f is the size of the embedding for feature f. More concretely, G_f is an $F \times N_f$ matrix where each row $k \in \{1, \ldots, F\}$ corresponds to the embedding of value k taken by feature f into its respective Euclidean space. Our categorical features, the number of discrete values they can take and the embedding sizes we picked are displayed in Table 2. We also add one additional feature, movie_vec, which learns one embedding for each movie in the database. The purpose of this feature is to account for other movie characteristics that we do not model directly, thus making our representation more flexible. We chose the embedding sizes in such a way that they reflect our intuition of how much each feature has a relative influence on the preferences of an user. That is, the larger the embedding size, the more importance we believe a feature has on the user's preference. For each movie i, the actors, genres, and keywords features are lists of integer values(size 4 for the actors, variable size for all the others), while all the other features are integer values. For the former features taking on value $[k_1, \ldots, k_p]$ for movie i, we compute the embedding as $E(f(i)) = \sum_{t=1}^{p} G_{fk_t}$ where G_{fk_t} is the k_t'th row of the embedding matrix of feature f. For the latter features taking on value k for movie i, we compute the embedding as $E(f(i)) = G_{fk}$. Finally, we concatenate these embeddings to obtain what we will refer to as the *categorical movie embedding* for movie i, $E_c(i) = E(f_1(i))|E(f_2(i))| \ldots |E(f_q(i))$, where q is the number of the categorical features, and $E_c(i) \in \mathcal{R}^{96}$.

As shown in [3], we can encourage the embedding matrices to learn useful mappings by further plugging the embeddings into a neural network which we then train to optimize some desired objective. Since we want to predict how a Netflix user j is going to rate a Netflix movie i, we devise a CF network that will collaboratively learn the embedding mappings for each of the categorical features by also learning an embedding for each user j and by fusing the user and movie embeddings through a neural network whose task is to predict the rating r_{ij}. More specifically, we have a user embedding matrix $G_u : \{1, \ldots, U\} \rightarrow R^{N_u}$, where $U = 480189$ and $N_u = 192$. We call the embedding of each user u using this matrix the *user embedding* and denote it by $E_u(j) = G_{uj}$, where G_{uj} is the j'th row of G_u. The categorical movie embedding and the user embedding are concatenated into one vector $E_{ij} = E_c(i)|E_u(j)$. We fuse the concatenated vector by training a neural network, which we will call the *categorical collaborative filtering network* and denote by $\mathcal{N}_{CCF}$, whose objective to make predictions $\hat{r_{ij}}$ as close as possible to the real r_{ij}. More specifically, we aim to jointly learn, by gradient descent, the categorical feature embedding matrices G_f for all categorical features, the user

Categorical Feature	Num Discrete Values(F)	Embedding Size(N_f)
release_year_bin	20	2
budget_bin	20	2
revenue_bin	20	2
runtime_bin	20	2
popularity_bin	20	2
language	63	2
director	6888	12
writer	3871	8
producer	4570	4
actors	24412	16
genres	21	4
keywords	13077	10
movie_vec	17770	30

Table 2: Categorical features, the number of discrete values they can take, and the embedding sizes

Figure 4: A graph of our estimation of the probability that two titles match given that their matching score is in an interval. The mean of the bins is given on the x axis



embedding matrix G_u, and the model $\mathcal{N}_{CCF}$ which minimize the error

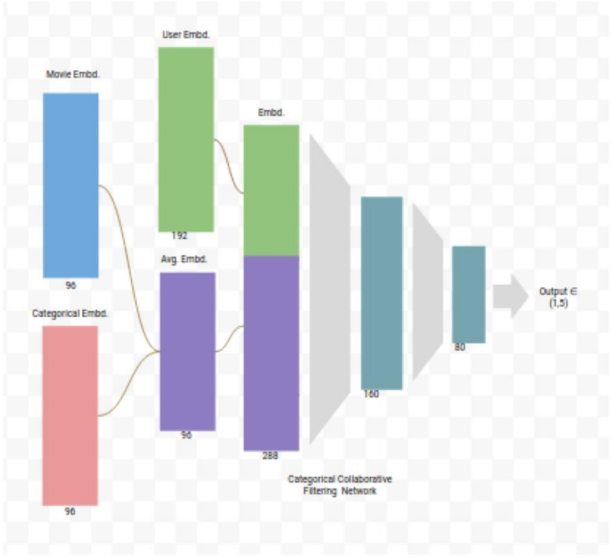
$$\sum_{(i,j,r_{ij})} \|\hat{r_{ij}} - r_{ij}\|_2^2, \qquad where \quad \mathcal{N}_{CCF}(E_{ij}) = \hat{r_{ij}}$$

where (i, j, r_{ij}) ranges over all (movie, user, rating) training pairs from the Netflix Dataset.

However, as we discussed in the Data Section, we have a great deal of uncertainty regarding how well Netflix movies were matched with movies from the Movies Database. Since the network described above takes as input a Netflix movie i, but uses the features $f(i)$ computed in the Movie Database, we have to account for how well the movie was matched across the two databases. When we performed the matching, we computed for each movie i a score $s_i \in [0, 100]$ which quantifies the quality of matching the movie titles between the two databases. We heuristically compute the probability of a match being correct given a score within a certain range by randomly selecting 20 movies that had a score in that range and deciding whether the match is correct or not. We plot the histogram we obtained through this process in Figure 4 and use the function $w(s_i) = \exp(-5.0 * (1.0 - (s_i - 44.0)/56.0)))$ to approximate the probability as a function of the score.

We will use $w(s_i)$ as the weight we place on the *categorical movie embedding* $E_c(i)$. We create a second movie embedding, which we will call the *auxiliary movie embedding* and denote by $E_a(i) \in \mathcal{R}^{96}$, with the purpose of learning a separate embedding for the cases in which the matches are unreliable. Namely, we will replace E_{ij} defined above, with $E_{ij} = E_{avg}(i)|E_u(j)$, where $E_{avg} = E_c(i) * w(s_i) + E_a(i)(1 - w(s_i))$. This means that $w(s_i)$ acts as a gate which decides when we train the categorical feature embeddings or not, depending on the quality of the match for i. We train this adjusted network using the error function above, and for each movie we will store

Figure 5: The network used for computing the average categorical movie embeddings, $E_{avg}(i)$, displayed in purple



the *average categorical movie embedding*, $E_{avg}(i)$, as the representation of the i-th movie's content based categorical features. The architecture we just described is presented in Figure 5.

It is now time to bring it all together and introduce our hybrid recommender system. We begin with the CBF part of our model. The input to this component will be the movie representations we learned above, namely $E_o(i)$ and $E_{avg}(i)$, plus the numerical features which we extracted from the Movies and Netflix Databases described in the Data section, which we will denote by $E_n(i) \in \mathcal{R}^{12}$. The concatenation of these vectors are plugged into a bottleneck neural network which we will call the *content based network* and will denote by $\mathcal{N}_{CB} : \mathcal{R}^{140} \to \mathcal{R}^{96}$. The output of this network is the *content based movie embedding* which we denote by $E_m(i) = \mathcal{N}_{CB}(E_o(i)|E_{avg}(i)|E_n(i))$. While we could have trained this network in an auto-encoder fashion, independently from the CF component, we decided to use what would correspond to the encoder part and use the code, $E_m(i)$, as the input to the CF part of our model, thus being able to simultaneously and co-dependently train the two components. To perform collaborative filtering we use exactly the same architecture we described above, but retrain G_u, the user embedding, and rename the fussing network to the *collaborative filtering network* which we denote by $\mathcal{N}_{CF}$. More specifically, we aim to jointly learn, by gradient descent, the models $\mathcal{N}_{CB}$ and $\mathcal{N}_{CF}$ which minimize the error

$$\sum_{(i,j,r_{ij})} \|\hat{r_{ij}} - r_{ij}\|_2^2, \;\; where \;\; \mathcal{N}_{CF}(E_m(i)|E_u(j)) = \hat{r_{ij}} \;\; and \;\; E_m(i) = \mathcal{N}_{CB}(E_o(i)|E_{avg}(i)|E_n(i))$$

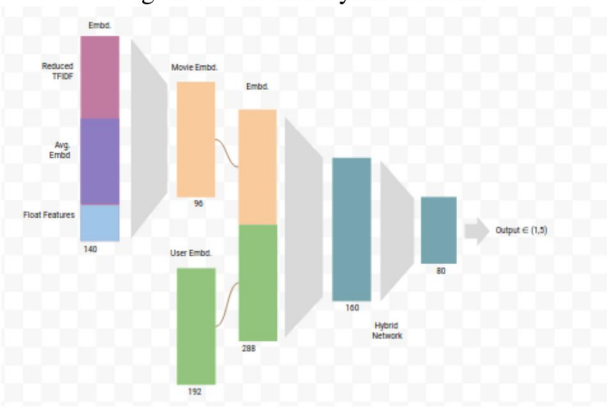
where (i, j, r_{ij}) ranges over all (movie, user, rating) training pairs from the Netflix Dataset.

This concludes our discussion of the model. For implementation details and results, consult Section 4.

4 Implementation Details and Results

We implemented all the models in TensorFlow and trained the models using only an i7 CPU. The tfidf autoencoder is a deep neural network architecture with fully connected layers of sizes $[815, 360, 36, 360, 815]$ and ReLU activation functions. We found L1 activation regularization of size 0.1 and L2 weight regularization of size 0.0005 to aid the learning process. We used Adagrad with an initial learning rate of 0.05 and trained until convergence. The embedding layer sizes have been described in Section 3. The categorical collaborative filtering network, $\mathcal{N}_{CCF}$, as well as the collaborative filtering network in the final hybrid system, $\mathcal{N}_{CF}$ are deep neural networks with fully connected layers of sizes $[160, 80]$ and ReLU activation functions. The final fully connected layer produces the predicted rating through a scaled and shifted sigmoid function which produces an output

Figure 6: The final hybrid network



between 0.5 and 5.5 which is then clipped to the interval $[1, 5]$. Since the Netflix ratings can have values $\{1, 2, 3, 4, 5\}$, it would make sense to have our output produce continuous values between 1 and 5. However, since a sigmoid function approaches its limits asymptotically at infinity, we chose to select 0.5 and 5.5 as our bounds in order to have the values 1 and 5 be achievable. Finally, the content based network, $\mathcal{N}_{CB}$ is a deep neural network with fully connected layers of sizes $[128, 96]$ and tanh activation functions. The reason why we used tanh activations is because the user embeddings have values on both sides of 0, so it makes sense for the content based movie embedding to do the same. We train both models using Adagrad with an initial step size of 0.05 and short stop the training when test error starts going up. We experimented with dropout as a form of regularization, but did not find it useful. Other forms of regularization were too computationally expensive for training on a small CPU.

Before describing our results we note that we kept our models small in order to have them trained using the limited computational power and time at our disposal. Furthermore, we did minimal hyper-parameter tuning which goes to show the robustness of our method. However, we conjecture that more powerful models with more careful hyperparameter tuning are likely to achive much better results.

In addition to the Netflix challenge winning algorithm, we also compare our results with a purely collaborative network that we implemented. This network follows a simple structure, similar to the one used in our hybrid network, but is instead learning a movie embedding which is agnostic to the CB data that we extracted from the Movies Database. In order to prove the power of the CB data we extracted from the Movies Database, we make this model much larger and use movie embeddings of size 150(compared to size 96 in our hybrid network) and user embeddings of size 300(compared to 188 in our hybrid network). The RMSE attained by our two models both beat the RMSE of the Netflix challenge winning algorithm, by some margin, but the hybrid network performs much better than the larger purely collaborative model. The RMSE is computed on the test bed described at the end of the introduction section.

Model	RMSE	Percent Improvement to Netflix Winner
Netflix Winner	0.8567	0%
Purely CF Net	0.8409	1.84%
Hybrid Net	0.8335	2.7%

Table 3: Categorical features, the number of discrete values they can take, and the embedding sizes

5 Conclusion and Future Work

In conclusion, our work presented a simple, general, and powerful hybrid RS for movie recommendations which uses expressive yet compact intermediate representations of the movies and the users. We show how to combine and leverage two movie datasets containing complementary data, even in the presence of imperfect matches between the two datasets. Furthermore, we present a method for

combining categorical and continuous features, a task which is notoriously difficult. We do this by learning useful embeddings for the categorical features by directly optimizing the task at hand. As noted before, our algorithm can be further improved by better hyperparameter tuning, larger models, better regularization, and a more structured approach to selecting the embedding or bin sizes.

6 Contributions

- Ishita Bhandari: Performed data cleaning, merged the two datasets, and created tf-idf vectors, minor contributions to report and plots, poster.
- Ajinkya Sheth: Data preprocessing and feature engineering(standardizing, binning, etc.), minor contributions to report and plots, poster.
- Octavian-Vlad Murad: Writing up the report, creating, coding, and testing the algorithm, poster.

References

[1] Bobadilla, J. , Ortega, F. , Hernando, A. & Gutiérrez, A. (2013) Recommender systems survey. *Knowledge-Based Systems* **46**: 109–132
[2] Mikolov, T. , Sutskever, I. , Chen, K. , Corrado, G. & Dean, J. (2013) Distributed Representations of Words and Phrases and their Compositionality.
[3] Guo, C. & Berkhahn, F. (2016) Entity Embeddings of Categorical Variables.
[4] Zhang, S. , Yao, L. , Sun, A. , & Tay, Y. (2018) Deep Learning based Recommender System: A Survey and New Perspectives.
[5] Kingma, D. & Welling, M. (2013) Auto-Encoding Variational Bayes.
[6] Koren, Y. (2009) The BellKor Solution to the Netflix Grand Prize
[7] The Netflix Prize Data. Retrieved from: https://www.kaggle.com/netflix-inc/netflix-prize-data
[8] Banik, R. The Movies Dataset. Retrieved from: https://www.kaggle.com/rounakbanik/the-movies-datasetratings.csv