Review Rating Prediction using Recurrent Neural Networks
CSE 258: Web Mining and Recommender Systems

Utkarsh Jain[1], Marialena Sfyraki[2]

University of California, San Deigo

[1]utjain@ucsd.edu [2]msfyraki@ucsd.edu

Abstract

E-commerce websites, like Amazon, Ikea, etc, allow users to buy/sell products online and let them leave an online review and a star rating for the overall experience they had with the product. An online review usually consists of four important parts - the user, the item, a free-form text called a review, and a rating out of 5. The problem of predicting the rating a user will assign to an item has become an interesting problem in recommendation systems that use these predictions to recommend new products to users. This problem is called Rating Prediction, and in this project, we treat it as a regression task. In this regard, we build various models which use different features, for e.g. user-item interactions and reviews, to predict the rating a user will give to an item. We start by implementing the Similarity-based Rating Prediction model as a baseline. We then shift our focus to slightly more powerful Latent-Factor models. Next, we move to purely text-based methods, where we use models like Bag-of-Words and Term-Frequency Inverse Document Frequency (TD-IDF) to predict the rating. Finally, we turn to more powerful deep learning models such as Elman Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) Networks, and Gated Recurrent Units (GRU) to exploit the sequential nature of reviews to make rating predictions. We analyze the performance of each model and conclude with the observation that deep learning models outperform the rest.

1. Exploratory Analysis of the Data

In this assignment, we use the **epinions dataset** [2] [13]. The dataset consists of the following files:

	user	item	review	rating
0	karleigh	Fisher_Price_Loving_Family_Sweet_Sounds_Dollhouse	i researched and looked at all the fisher pric...	4.0
1	mfw1982	Nokia_E62_Smartphone	no it doesnt have a camera and yes the keys ar...	5.0
2	davydanger	pr-Dell_DJ_15GB_MP3_Player	i was very excited to buy this product given a...	1.0
3	kbmg	Blue_s_Clues_Bath_Time_Blue	my two kids ages 2 and 4 are big blues clues f...	2.0
4	pluckyduck	Spider_Man_Gloves	dad was walking through toys r us the other we...	3.0

Figure 1. First five rows of training data

1. **epinions.txt**: contains interactions between users and items. It contains the following columns: (1) *user*: unique user handle, (2) *item*: unique item handle, (3) *paid*: amount paid by the user to buy that item. Not used in this assignment, (4) *time*: timestamp when the review was made. Not used in this assignment, (5) *rating*: rating given by the user to the item. The values belong to the set $\{1,2,3,4,5\}$, and (6) *review*: text field containing the review left by the user.

2. **network_trust.txt**, **network_trustedby.txt**: contains pairs of users who follow and trust each other. We will not use these files for this assignment.

Our dataset contains 195,410 total reviews, with 118,183 unique users and 42,739 unique items. It is shuffled and split into three parts - training (80%), validation (10%), and testing (10%). Please refer to Fig 1 for the first five rows from the training dataset.

1.1. Rating Statistics

Please refer to Table 1 for the basic statistical properties of the ratings in the complete dataset. A boxplot of the ratings is presented in Fig 2. A majority of the ratings in the dataset were 5.0, which explains the shifted overall mean. The next most common rating was 4.0, followed by 1.0, then 3.0, and 2.0. Fig 3

Property	Value
Count	195,410
Mean	20.30
Standard Dev.	12.61
Mininum	1.0
25^{th} Percentile	2.0
50^{th} Percentile	4.0
75^{th} Percentile	5.0
Maximum	5.0

Table 1. Rating Statistics

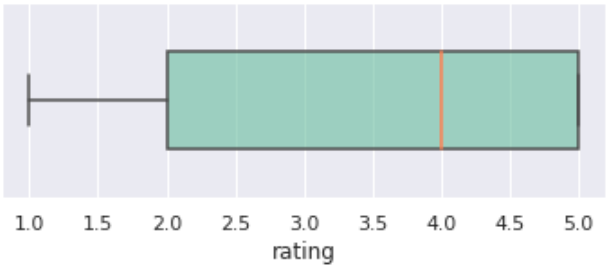

Figure 2. Boxplot of ratings

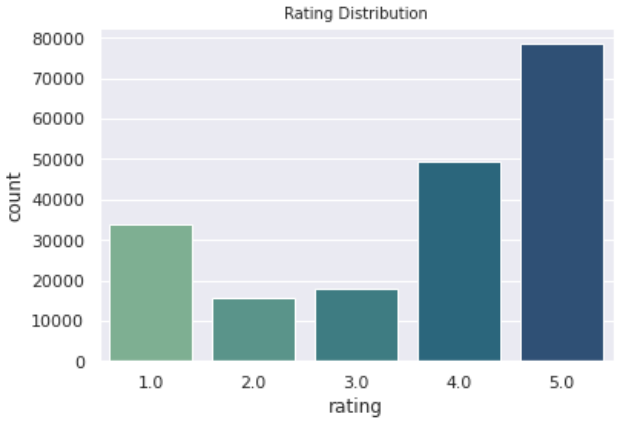

Figure 3. Histogram of ratings

presents a histogram of the distribution of ratings.

1.2. Users and Items Statistics

Now, we present our analysis of how dense the user-item interactions are, i.e the distribution of reviews left by each user, and the distribution of reviews left on each item.

The left table in Table 2 presents the statistics of the reviews left by each user. As we can see, on average a user leaves around 2 reviews. Similarly, the right table in Table 2 shows the statistics of reviews left on each item. Each item receives an average of 5 reviews.

Property	Value
Count	118,183
Mean	1.65
Standard Dev.	2.33
Minimum	1.0
25^{th} Percentile	1.0
50^{th} Percentile	1.0
75^{th} Percentile	1.0
Maximum	89.0

Property	Value
Count	42,739
Mean	4.57
Standard Dev.	9.20
Minimum	4.0
25^{th} Percentile	1.0
50^{th} Percentile	2.0
75^{th} Percentile	4.0
Maximum	278.0

Table 2. **Left**: Distribution of reviews left by each user. **Right**: Distribution of reviews left on each item.

This can partly be explained by the fact that our dataset contains many more unique users than items.

1.3. Reviews and Word Clouds

We now present a visualization of the different words used in reviews in each rating class. Since the dataset contains reviews on electronic items, we ignore words like ”bought”, ”purchased”, ”model”, etc while creating the word cloud to emphasize more expressive words like ”liked”, ”loved”, ”hate”, etc.

Please refer to Fig 4 and Fig 5 for word cloud for reviews on items rated 5.0 and 1.0 respectively.

Figure 4. Word Cloud for reviews on items with 5.0 rating

Based on Fig 4, we can see that words like ”great”, ”buy”, ”good”, ”love”, ”using”, etc are commonly used for items with top ratings. A similar analysis of Fig 5 reveals that words like ”good”, ”great”, ”love”, etc make their way into reviews for low-rated items too. However, we also see the word ”not” in the word cloud, which means that it was probably used with the positive words in the previous sentence (for e.g in expressions like ”not good”, ”not great”, ”did not love”, etc). We can also see words like ”worst”, ”problem”, ”horrible”, ”never”, etc are used for items with low

Figure 5. Word Cloud for reviews on items with 1.0 rating

ratings.

In addition to this, we read some of the reviews in the dataset and found that a good proportion of users wrote seemingly positive (negative) reviews, but left a low (high) rating. As explained later in section 5.5, these discrepancies between the tone of the review and the rating will limit our deep learning models from making more accurate predictions.

2. Predictive Task

In this paper, we use the **Epinions** dataset and try to predict the rating a user will give to an item. This problem is commonly known as Rating Prediction, and we treat it as a regression task. The main motivation behind choosing this particular problem statement is to see how purely text-based models outperform the similarity-based and latent-factor models we learned in the course.

Additionally, although the ratings in our model appear in a categorical form ($\{1, 2, 3, 4, 5\}$), we still treat Rating Predictions as a regression task because we aim to build models which are robust to the nature of the labels. The models we develop in this paper can be used as is on any different dataset with the same schema, even if the ratings take real-valued form or change their range.

Evaluation Metric

Since we are treating our problem as a regression task, we evaluate the models using Mean Squared Error (MSE). The MSE is defined as below:

$$MSE = \frac{\sum_1^N (y^i_{actual} - y^i_{predicted})^2}{N} \qquad (1)$$

where, (1) N: number of data points, (2) y^i_{actual}: the actual rating for the i^{th} data point, (3) $y^i_{predicted}$: the predicted rating for the i^{th} data point.

We choose MSE as an evaluation metric because it eliminates negative values from the differences and ensures that the error is always positive. Secondly, since we are using neural networks for predicting ratings, we want our loss function to penalize our model more for making larger errors as compared to smaller ones, and the MSE loss function achieves that. Thirdly, a differentiable function like MSE serves as a better objective in neural networks as compared to non-differentiable loss functions like Mean Absolute Error (MAE).

Baselines

We choose Similarity-based Rating Prediction model as a relevant baseline for this task. The performance of other models is then compared to this baseline.

For assessing the validity of our models' predictions, we check their performance on the validation set. This will especially prove important while training our deep learning models in order to avoid overfitting.

Feature Extraction

For all reviews in the training dataset, we first convert them to lowercase, remove punctuations, and use "basic-english" tokenizer from TorchText to tokenize them. The same preprocessing is applied to the validation and testing sets as well.

We use different features for different models, and they are explained below:

1. *Similarity-based Rating Prediction*
   The similarity-based prediction model uses only the *rating* feature and the corresponding *user* and *item* handles. As a collaborative filtering approach, the similarity-based prediction can be applied on a context that does not contain much information associated with the user/item, but rather relies on correlations between users/items.

2. *Latent-Factor Model*
   In the case of the latent-factor model, the model factorizes user ratings of products into user and item feature vectors and ignores other user and item

features. The hidden latent dimensions corresponding to user/item characteristics are identified by the model and are used to build lower-rank approximation user and item matrices.

3. *Bag-of-Word*
   To limit the size of the feature representation, we use the top-*K* frequent words (ignoring stopwords) in our preprocessed training dataset to use as features. To generate a feature vector of a review, we count the number of times each word in our vocabulary appears in the review. This way, every review is expressed as a feature $F \in R^K$

4. *TF-IDF*
   We choose the top-*K* frequent words (ignoring stopwords) across the training dataset as features. To generate a feature vector of a review, we use the $tfidf$ value for each of the vocabulary words in that review. This way, every review is expressed as a feature $F \in R^K$

5. *RNN, LSTM, and GRU*
   To keep a reasonable vocabulary size, we only consider words that appear more than 5 times in the preprocessed training dataset. By the nature of how these models work, we do not ignore stop words to avoid losing sentence structure and semantics. A special "<UNK>" token is added to the vocabulary to represent Out-of-Vocabulary words. Finally, each word in our vocabulary is mapped to a D-dimensional embedding space which is learned during training. This embedding matrix is represented as $E \in R^{V \times D}$, where V is the vocabulary size and D is the embedding dimension. To represent a review with T words and represented by the word sequence $(w_1, w_2, ...w_T)$ as a feature, the embedding for each word w_i is concatenated to generate a feature vector $F \in R^{T \times D}$.

3. Model

In this section, we explain the models used, the motivation behind using them, their strengths, and their weaknesses.

3.1. Similarity-Based Model

Our first approach to the predictive task was the traditional collaborative filtering algorithm (CF), which is essentially the process of using the interaction between similar groups to predict the preference of the object in question. In a typical CF algorithm there are two approaches: (1) the user-based, and (2) item-based approach. In the case of our task, we predict the rating a certain user would give to a certain item considering the similarity of the other users' evaluation of the said item (user-based) and the similarity of the other items that the said user has evaluated (item-based).

The motivation for using the similarity-based approach is based on the fact that the behavior of the target user can be predicted from the past behavior of a group of users (customers) similar to the target user. Similarly, the same idea applies to the target item with a group of similar items.

Among the similarity calculation methods, the most commonly used similarity measures are the Jaccard similarity, Cosine similarity and Pearson similarity, which are defined as follows for item-based similarity:

Jaccard Similarity

$$J(U_i, U_j) = \frac{|U_i \cap U_j|}{|U_i \cup U_j|}, \tag{2}$$

where U_i and U_j represent the set of users who rated the items i and j respectively.

Cosine Similarity

$$Cosine(i, j) = \frac{U_i \cdot U_j}{\|U_i\| \cdot \|U_j\|}, \tag{3}$$

where U_i and U_j represent the set of users who rated the items i and j respectively.

Pearson Similarity

$$PCC(i, j) = \frac{\sum_{u \in U_i \cap U_j} \left(R_{u,i} - \overline{R}_i\right)\left(R_{u,j} - \overline{R}_j\right)}{\sqrt{\sum_{u \in U_i \cap U_j} \left(R_{u,i} - \overline{R}_i\right)^2 \sum_{u \in U_i \cap U_j} \left(R_{u,j} - \overline{R}_j\right)^2}}, \tag{4}$$

where U_i and U_j represent the set of users who rated the items i and j respectively, $R_{u,i}$ and $R_{u,j}$ denote the rating that user u gave to items i and j respectively and $\overline{R}_i$ and $\overline{R}_j$ denote the mean ratings on i and j.

Analogous similarity measures can be used in the case of user-based similarity.

Several similarity measures have been proposed. Apart from the aforementioned similarity measures, we also implement the following similarity measures [8] for comparison:

Adjusted Cosine Similarity

The adjusted cosine similarity measure calculates the correlation value between two items i and j and it is defined as follows:

$$ACosine(i,j) = \frac{\sum_{u \in U_i \cap U_j} \left(R_{u,i} - \overline{R}_u\right)\left(R_{u,j} - \overline{R}_u\right)}{\sqrt{\sum_{u \in U_i \cap U_j} \left(R_{u,i} - \overline{R}_u\right)^2 \sum_{u \in U_i \cap U_j} \left(R_{u,j} - \overline{R}_u\right)^2}}, \quad (5)$$

where U_i and U_j represent the set of users who rated the items i and j respectively, $R_{u,i}$ and $R_{u,j}$ denote the rating that user u gave to items i and j respectively and $\overline{R}_u$ denotes the mean rating by user u.

Mean Squared Difference Dimilarity

The mean squared distance between two points in the Euclidean space can be used as a similarity measure. The MSD similarity between two items i and j is defined as follows:

$$MSD(i,j) = \frac{\sum_{u \in U_i \cap U_j} \left(R_{u,i} - R_{u,j}\right)^2}{|U_i \cap U_j|}, \quad (6)$$

where U_i and U_j represent the set of users who rated the items i and j respectively and $R_{u,i}$ and $R_{u,j}$ denote the rating that user u gave to items i and j respectively.

Improved Triangle Similarity

The improved triangle similarity consists of a product of two terms, the triangle similarity and the user rating preferences. The triangle similarity considers both the length of vectors and the angle between them, so it is more reasonable than the Cosine measure, which only considers the angle. The user rating preferences similarity considers the similarity of averages and variances of the users' ratings. Thus, improved triangle similarity (ITR) not only focuses on common ratings, but also takes into account the non-common rating of users.The ITR similarity of two items i and j is defined as follows:

$$ITR(i,j) = Triangle(i,j) \cdot URP(i,j), \quad (7)$$

where

$$Triangle(i,j) = 1 - \frac{\sqrt{\sum_{u \in U_i \cap U_j} \left(R_{u,i} - R_{u,j}\right)^2}}{\sqrt{\sum_{u \in U_i \cap U_j} R_{u,i}^2} + \sqrt{\sum_{u \in U_i \cap U_j} R_{u,j}^2}} \quad (8)$$

and

$$URP(i,j) = 1 - \frac{1}{1 + exp\left(-|\overline{R_i} - \overline{R_j}| \cdot |\sigma_i - \sigma_j|\right)}, \quad (9)$$

where

$$\sigma_i = \sqrt{\frac{\sum_{u \in U_i} \left(R_{u,i} - \overline{R}_i\right)^2}{|U_i|}}. \quad (10)$$

U_i and U_j represent the set of users who rated the items i and j respectively and $R_{u,i}$ and $R_{u,j}$ denote the rating that user u gave to items i and j respectively and $\overline{R}_i$ and $\overline{R}_j$ denote the mean ratings on i and j respectively.

Ratings Prediction Function

The aforementioned similarity measures can be used in an analogous manner in the case of user-based similarity.

We use similarities between users and similarities between items for rating prediction. The rating prediction function used in the case of item-based similarity is as follows:

$$r(u,i) = \overline{R}_i + \frac{\sum_{j \in I_u \setminus \{i\}} \left(R_{u,j} - \overline{R}_j\right) \cdot sim(i,j)}{\sum_{j \in I_u \setminus \{i\}} sim(i,j)} \quad (11)$$

However, as seen in section 1.2, the data is too sparse to find sufficiently many co-rated items, thus leading to inaccurate predictions. Due to the sparsity of the data, even though a considerable number of users or items are similar to the target, their information is not effectively applied to the desired prediction.

To alleviate this issue, as described in [14], we make a reasonable assumption that even similar items that do

not meet the constraint condition of prediction formulas may still make a valuable contribution to the prediction process. Our second implementation process in the case of item-similarity is as follows: we divide the similar items into two subsets, by judging whether the given user u has rated them. Items that meet this condition directly participate in the prediction process as in the traditional formula. For each item in the second subset, we search for the users who have rated it and are the most similar to the given user u. Hence, the user similarity is further added as a specified weight vector to adjust the error caused by this approximate substitution. The new prediction formula is then defined as:

$$r(u,i) = \overline{R}_i + \frac{\sum_{j \in I_u \setminus \{i\}} sim(i,j) \cdot \left(R_{u,j} - \overline{R}_j\right)}{\sum_{j \in I_u \setminus \{i\}} sim(i,j) + \sum_{j \in I_u^c \setminus \{i\}, U_j} sim(u,v) \cdot sim(i,j)} + \frac{\sum_{j \in I_u^c \setminus \{i\}, U_j} sim(u,v) \cdot sim(i,j) \cdot \left(R_{v,j} - \overline{R}_j\right)}{\sum_{j \in I_u \setminus \{i\}} sim(i,j) + \sum_{j \in I_u^c \setminus \{i\}, U_j} sim(u,v) \cdot sim(i,j)} \tag{12}$$

The similarity-based methods are generally intuitive and relatively simple to implement as they provide a concise and intuitive justification for the computed predictions. However, the similarity-based prediction model depends highly on whether or not we chose a "good" similarity measure. Furthermore, the frequently observed data sparsity can cause the cold-start problem, which describes the difficulty of making predictions when the users or the products do not have enough historical data. Additionally, the similarities have to be recomputed every time a new user or item is added. Finally, as the number of users/items increases and the amount of data expands, the model suffers from serious scalability issues.

3.2. Latent Factor Model

The Latent Factor Model (LFM) is one of the most successful methods for collaborative filtering (CF). Matrix factorization techniques are a class of widely successful Latent Factor models that attempt to find weighted low-rank approximations to the user-item matrix, where weights are used to hold out missing entries.

Bias-Only Model

CF models try to capture the interactions between users and items that produce the different rating values. However, much of the observed rating values are due to effects associated with either users or items, independently of their interaction. The motivation for this model lies in the fact that some users are more likely to give lower ratings than others, while some items are more likely to receive higher ratings

The Bias-Only model is defined as follows:

$$r(u,i) = \mu + \beta_u + \beta_i, \tag{13}$$

where μ is the overall average rating and β_u and β_i are the effects of user u and item i respectively.

The parameter estimates μ , $\{\beta_u\}_{u=1}^n$, $\{\beta_i\}_{i=1}^m$, are found by solving the minimization problem:

$$\arg\min_{\mu,\beta} \sum_{u,i} (\mu + \beta_u + \beta_i - R_{u,i})^2 + \lambda \left[\sum_u \beta_u^2 + \sum_i \beta_i^2 \right], \tag{14}$$

where λ is the regularization constant.

Although the Bias-Only model works better than the similarity-based approach, it still assumes that users and items are independent of each other.

Matrix Factorization Model

The matrix factorization model attempts to identify latent factors by factoring the user-item rating matrix into two low-dimensional matrices containing all d-dimensional user feature vectors and item feature vectors. This is achieved by performing singular value decomposition using the first d greatest eigenvalues and their corresponding eigenvectors to reduce the original matrix $\boldsymbol{R}$ to a matrix $\hat{\boldsymbol{R}}$ of rank d that provides the best low-rank linear approximation to the original matrix. The latent space tries to explain ratings by characterizing both items and users on factors automatically inferred from user feedback. Recently, matrix factorization models have gained popularity, thanks to their attractive accuracy and scalability.

The Matrix Factorization model is defined as follows:

$$r(u,i) = \mu + \beta_u + \beta_i + \gamma_u \cdot \gamma_i, \tag{15}$$

where μ is the overall average rating, β_u and β_i are the effects of user u and item i respectively and γ_u and γ_i are the feature vectors associated with user u and item i respectively.

The minimization problem in the case of matrix factorization models is the following:

$$\arg\min_{\mu,\beta,\gamma} \sum_{u,i} (\mu + \beta_u + \beta_i + \gamma_u \gamma_i - R_{u,i})^2 + \lambda \left[\sum_u \beta_u^2 + \sum_i \beta_i^2 + \sum_u \|\gamma_u\|_2^2 + \sum_i \|\gamma_i\|_2^2 \right], \tag{16}$$

where λ is the regularization constant.

One of the main advantages of the latent factor model is that there is no dependence on domain knowledge as embeddings are automatically learned from the model. Also, the system needs only the rating matrix to train a matrix factorization model and it does not require any contextual features.

However, the latent factor model still suffers from the cold start problem. In addition to that, the model solely relies on ratings, making it hard to explicitly and accurately model users' preferences and explain the underlying rationale behind them. Finally, since the latent factors of users and items are learned via a global optimization strategy, the model cannot achieve an optimal rating prediction locally. It cannot accurately capture the importance of each latent factor for a user towards an item and hence the performance could be severely compromised for individual users or items.

3.3. Bag-Of-Words (BoW)

As mentioned in section 1.3, we see in Fig 4 and 5 how the overall tone of words changes for top-rated and low-rated items. These words carry important information about the ratings, but unfortunately, there was no way to use this information in our previous models. However, a BoW model allows us to capture this apparent shift in the tone of the reviews and lets us represent them as a fixed-length vector which can be used to make more accurate predictions.

A BoW model is a method of extracting features from the text for use in machine learning algorithms. BoW approach enables us to represent any text in the form of the frequency of words appearing in the text. Traditionally, we need two things to accomplish this:

1. Vocabulary: A vocabulary of known words you want to consider. Usually, stopwords are removed from consideration because they carry no meaning.
2. Frequency: The frequency of known words appearing in a text.

Now, given any text, it can be converted to a fixed-size vector that expresses the said text in terms of the frequency of the known words. In our model, we have considered the top-1,000 frequent words to keep the feature size big enough to not lose any important information. The vector representation for all the reviews in the training dataset, along with their corresponding ratings, forms the training samples that are used for training a linear regression model.

Despite its simplicity, a BoW model is quite powerful and performs better than the previous models. However, it has certain disadvantages too. First, with this approach to text representation, one loses information concerning word order and semantics, which, as we will see later, plays a major role in the success of advanced natural language processing techniques. For e.g., this model represents "not bad, buy!" and "bad, not buy!" in the exact same way irrespective of the huge difference in the meaning they convey. Second, these models can quickly grow in size as you consider more words to include in the feature space, which results in increased training times and calls for more computational power. Finally, by considering the most frequent words as features, we miss out on rarer words that carry more "information content". For instance, due to the nature of the dataset, our vocabulary contains neutral, but frequent, words like "product", "phone", "dvd", etc, and ignores more expressive, but rarer, words like "terrible", "terrific", etc.

3.4. TF-IDF

As seen earlier, BoW ignores a lot of rare words which could potentially be carrying more information, and also assigned a higher weight to more frequent, but neutral, words. To address these drawbacks, we shift to the Term Frequency Inverse Document Frequency (TF-IDF) model.

There are two halves to this model:

1. Term frequency, $tf(t, d)$ = number of times term t appears in document d.
2. Inverse document frequency, $idf(t, D)$ = it is defined as the log of the ratio of total number of documents to number of documents in which term t

appears.

$$idf(t, D) = log\frac{N}{|d \in D : t \in d|} \quad (17)$$

Then, the TD-IDF of a term t in document d is computed as $tdidf(t, d, D) = tf(t, d) * idf(t, D)$. This way, as compared to BoW, frequently occurring words, due to low idf values, are assigned lower weights in the final document vector representation. However, rarer words, which sometimes convey more information, due to higher idf values, get higher weights. By using this approach, the document is represented by words that are most "characteristic" of the document.

Despite being slightly more complex than BoW model, TF-IDF method performs almost comparably to Elman RNN (results discussed later). However, TF-IDF faces similar disadvantages as BoW. By using this technique to represent a document, we lose the sentence structure and semantics. Furthermore, TF-IDF method is equally memory inefficient as BoW and suffers from sparse representations and the curse of dimensionality.

3.5. Elman Recurrent Neural Network

A Recurrent Neural Network (RNN) is a class of artificial neural networks that are adapted to work on sequential data. They differ from simple feed-forward networks by their unique ability to use prior information to model current input and output. Feed-forward networks assume independence among inputs and thus only rely on the current input to generate an output. However, RNNs are built in a way that they remember past information and use that knowledge, along with the current input, to generate a more "contextually-aware" output. Since a piece of text is a sequence of words, RNNs become a perfect choice of model to work with when dealing with textual data. We aim to train RNNs and use them to predict ratings given a review.

There are certain upsides to moving to an RNN model. First, it does a much better job of capturing the semantic structure of a review as compared to the simple BoW model. Second, since each word is represented by a low-dimensional embedding, we don't have to worry about our representation blowing up with the vocabulary size meanwhile remaining sparse. This allows us to consider rare words which we could not use in the BoW model. Finally, this approach frees us from dealing with sparse user-item interactions and solely relies on the review to make a rating prediction.

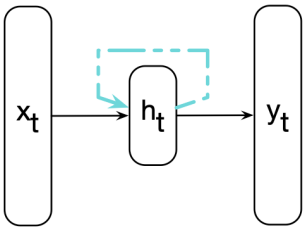

Figure 6. Elman Recurrent Neural Network [3]

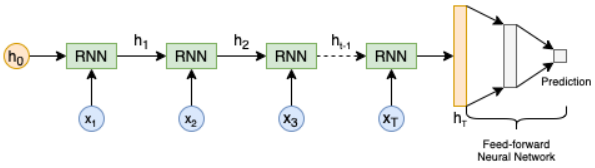

Figure 7. The architecture of our RNN model.

In this section, we consider the most basic class of RNNs called Elmann Networks [4] (or, Simple RNNs). Fig 6 illustrates the working behind an Elman RNN. The input vector at time t, x_t, is multiplied to a weight matrix and passed through a non-linear function to generate h_t, which is also called the hidden state at time t. This hidden state is then used to compute the corresponding output, y_t. As you can see in the figure, the recurrent link (represented with a dashed line) combines the hidden state from the preceding point in time, i.e h_{t-1} and input x_t to generate h_t. Using h_{t-1} this way serves as a memory, or context, that encodes previous computations and guides future decisions. Given x_t and h_{t-1}, h_t is computed as follows:

$$h_t = tanh(Uh_{t-1} + Wx_t) \quad (18)$$

where, $U \in R^{d_h \times d_h}$, $W \in R^{d_h, \times d_{in}}$, $h_t, h_{t-1} \in R^{d_h}$, $x_t \in R^{d_{in}}$, and $h_0 = \{0\}^{d_h}$ (d_{in} and d_h are the input and hidden sizes respectively).

3.5.1 Elman RNN

Now, we explain how we use Elman RNN to develop our Ratings Predictions model. The overall structure of our model is shown in Fig 7.

Text Representation

Assume a review with T words and represented by the word sequence $(w_1, w_2, ...w_T)$. We first map each of these T words to their corresponding embedding. The embedding of a word w is computed as $x_w = E(w)$. $E(w)$ represents the embedding of word w in an embedding matrix $E \in R^{V \times D}$, where V is the vocabulary size and D is the embedding dimension. After this embedding is done, we now have a word embedding matrix $X \in R^{T \times D}$. We then apply our Elman RNN model to get the final hidden state, i.e h_T.

At every time step t, the Elman RNN inputs the word embedding x_t of the word w_t and the previous hidden state h_{t-1} to compute h_t. We continue this until we have processed the very final word w_T that generates the final hidden state h_T. For our experiments, $d_{in} = d_h = D$.

Rating Prediction

We pass h_T through a feed-forward neural network as follows:

$$prediction = FFN(h_T) \tag{19}$$

where, $FFN(h_T) = max(0, h_T W_1 + b_1)W_2 + b_2, W_1 \in R^{D \times 0.5*D}$, $W_2 \in R^{0.5*D \times 1}$.

In practice, due to the simple nature of Elman RNN, it becomes really difficult to train them to use information distant from the current time step. Despite encoding preceding computations in a hidden state, the information tends to be fairly local and can only capture recent computations more effectively. This limits them from being used in cases, such as Rating Prediction, where we may need distant information to make a more guided decision. Moreover, Elman RNNs also suffer from the vanishing gradients problem, which drives the gradient to zero, hence making the backward pass during training ineffective.

3.6. Long Short-Term Memory (LSTM) Network

To address the weaknesses of Elman RNN, more complex networks have been built that can maintain relevant information for a longer period of time by forgetting information that is no longer needed. One such commonly used extension to an Elman RNN is LSTM [12].

An LSTM processes information in the same way an Elman RNN does. However, to maintain relevant information for a longer period of time, it employs the concept of a 'memory cell'. It also uses different gates, such as forget gate, update gate, and output gate, that allow more control over the gradient flow and effectively deals with the vanishing gradient problem. Fig 8 gives an accurate visualization of how all the computations occurs.

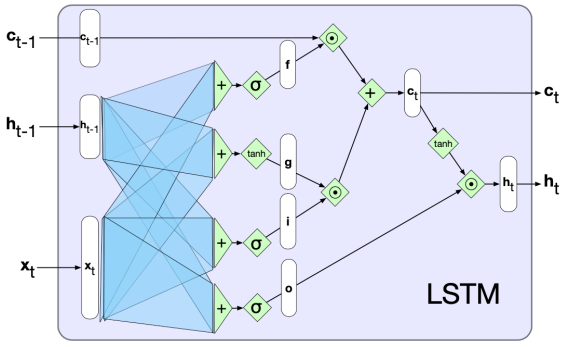

Figure 8. A single LSTM unit. Inputs consist of current input x_t, previous hidden state h_{t-1}, and previous cell state c_{t-1} [3]

Rating Prediction

The overall text processing remains exactly the same as we saw in section 3.3.1, but with one key difference. We replace the Elman RNN unit in Fig 7 with an LSTM unit.

3.7. Gated Recurrent Unit (GRU) Network

Although LSTMs work well for processing long sequences of texts while maintaining meaningful context, they demand long training times due to their complex architecture. To deal with this, we use GRUs [10] which require lesser training time and memory.

The flow of information in GRUs is similar to that of LSTMs, but with a small modification. It does not use a 'memory unit', and uses two gates (namely update and reset gate) instead of three. Despite its less complex architecture, its performance is comparable to that of LSTM, while being more computationally efficient. Fig 9 demonstrates how a GRU unit works.

Rating Prediction

The overall text processing remains exactly the same as we saw in section 3.3.1, but instead of using an Elman RNN unit, we use a GRU unit.

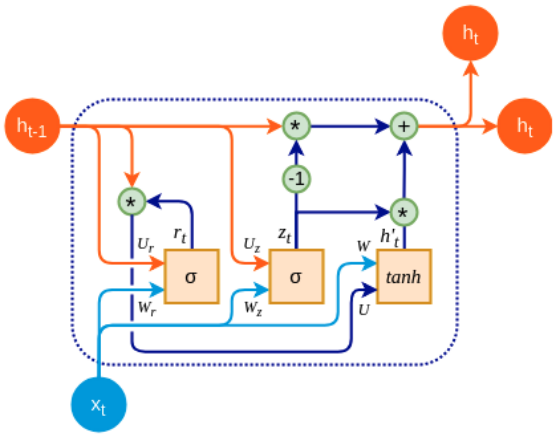

Figure 9. A single GRU unit. Inputs consist of current input x_t, previous hidden state h_{t-1} [9]

4. Pertinent Literature

Most of the recent work related to Rating Predictions tasks involves using sentiment analysis to extract meaningful features from the review text. Qu et al. [11] address this problem from Amazon.com reviews by introducing a novel feature extraction method called bag-of-opinions. Under this technique, they first extract all the opinions from a given review. Each opinion is made up of three components - a root word, a set of modifiers from the review, and one or more negation words. To predict the rating of a review, each of the opinions is assigned a score (which is learned on a large independent dataset of reviews, for e.g., all Amazon reviews), which are then aggregated and combined using a domain-dependent unigram model.

Leung et al. [1] proposed a new approach of combining Collaborative Filtering (CF) models and the Sentiment Analysis approach to make more guided rating predictions. They do this by first constructing an opinion dictionary that contains opinion words, their sentiment orientation, and the strength of that orientation. The strength of a word with respect to a sentiment orientation is the relative frequency of its occurrence in that particular class. Basically, the opinion dictionary answers the following question: "Given an opinion word, how likely is it used with a positive sentiment, and how likely it is used with a negative sentiment" [1]. Through this computation, they found out that although similar words imply similar meanings, they are not used with similar sentiments. Given a review, they aggregate the scores of all the opinion words in the review and combine them with common CF algorithms to generate a final prediction. For their research, they used the **MovieLens** dataset which contains user ratings for 1,692 movies. They then crawled IMDb to download user reviews for each movie and generated a dataset with approximately 30k reviews for 1,477 movies, provided by 1,065 users.

Fan et al. [6] employ a similar purely text-based approach as ours to predict a Business's star rating on Yelp based on user reviews. For this, they experiment with variations of bag-of-words models and use supervised machine learning algorithms like Linear Regression, Support Vector Regression, and Decision Tree for predictions. They used **Yelp** dataset containing 1,000 restaurants with a total of 35,645 text reviews written about them.

The **Epinions** dataset we used in this paper was directly downloaded from Prof. Julian McAuley's repository [2] [13] and can be easily collected using basic crawling tools from epinions.com. In McAuley et al. [2], the authors use this dataset to address the cold-start problem in recommender systems. They improve upon the matrix factorization-based techniques by incorporating signals from social relationships and sequences of recent activities. Zhao et al. [13] propose a novel personalized feature projection method for one-class recommendation problem. By using this novel approach, they were able to better model users' preferences over items as compared to the simple latent factor models.

Li et al. [5] proposed a novel solution for the Rating Prediction task on **Epinions** dataset (crawled independently) where instead of just using reviews to make predictions, which is our aim in this paper, they incorporate reviewer and product preferences into a text-based learner to predict the rating a user will give to a product. They also employ a tensor factorization technique to address the user-item interaction sparsity and discover latent factors among the different features. Their dataset contained 5,806 users, 13,269 items, and 220,654 total reviews. Furthermore, Fan et al. [7], came up with a novel graph neural network framework that exploits social trust networks and user-item interaction graphs for learning users' and items' latent factors. These latent representations are concatenated and fed into a multi-layer perceptron for rating prediction. They ran their experiments on an independently

collected epinions dataset that contains 18,088 unique users, 261,669 unique items, and 764,352 ratings, and was much bigger than the one we use in this paper.

In this paper, our main goal is to use the semantics of the review text to make better rating predictions and compare their performance with the models covered in the course. Therefore, we do not concern ourselves with sentiment analysis or graph-based methods. Furthermore, we find that existing works that rely on text-based approaches arrive at similar conclusions as we do. One such common conclusion is that deep learning-based methods almost always do a better job of predicting ratings.

5. Results

In this section we demonstrate the effectiveness of the various models we have built for the Rating Prediction task.

5.1. Similarity Based Model

In the case of the similarity-based model, we estimate the ratings based on two different rating prediction functions and six different similarity measures.

The resulting MSEs on the test set in the case of the rating prediction function (11) are reported in the Table 3.

Similarity Measure	User-Based	Item-Based
Jaccard	2.444	1.952
Cosine	2.446	1.952
Pearson	2.530	1.963
ACosine	2.445	1.966
MSD	2.441	1.937
ITR	2.430	1.934

Table 3. Evaluation of different similarity measures for the rating prediction function (11)

As can be seen from Table 3, the item-based approach significantly outperforms the user-based approach, regardless of the similarity measure used. The accuracy of the similarity-based methods depends mostly on the ratio between the number of users and items in the system. In cases where the number of users is much greater than the number of items, as in the case of our dataset, item-based methods produce more accurate recommendations. Moreover, better scalability makes the item-based approach more favorable in most cases. The number of items is in most cases much less dynamic than the number of users so the item-item similarities can be computed offline and accessed when needed. In addition, item-item methods are more appropriate for explaining the reasoning behind predictions. This is due to the fact that users are more familiar with items they have previously rated, but not those allegedly similar users. Finally, item properties show permanence, as opposed to users, whose opinions may change over time.

We tried to address the issue of sparse data by implementing the similarity-based model using the rating prediction function (12) and the resulting MSEs on the test set are reported in Table 4.

Similarity Measure	User-Based	Item-Based
Jaccard	2.454	1.966
Cosine	2.467	1.976
Pearson	2.541	1.980
ACosine	2.556	1.978
MSD	2.438	1.933
ITR	2.429	1.932

Table 4. Evaluation of the different similarity measures for the rating prediction function (12)

As before, we can conclude from the results of Table 4 the item-based approach performs considerably better than the user-based approach for the rating prediction function (12). In the cases of the MSD and ITR similarity measures, the reported MSE is slightly lower as compared to the first rating prediction function (12). In the rest of the cases, the use of the rating prediction function (11) renders a lower MSE.

Overall, from the results of Tables 3 and 4, it may be concluded that different combinations of rating prediction functions with different similarity measures can have a significant impact on the reduction of the MSE. However, the reported MSEs in all of the examined cases are still far from the desired.

5.2. Latent Factor Model

In the case of the bias-only model, we minimize the loss function (14) using L-BFGS-B. The maximum number of iterations was set to 100. The MSE on the validation set using different regularization constant values for the bias-only model can be seen in Table 5.

As can be seen from Table 5, the MSE is minimized using a regularization constant of $\lambda = 0.00001$. The

Regularization Constant λ	MSE
0.01	2.203
0.005	2.186
0.001	2.096
0.0005	2.030
0.0001	1.842
0.00005	1.775
0.00001	1.745
0.000005	1.792

Table 5. Evaluation of the different regularization constant values on the validation set for the bias-only model

reported MSE value on the test set in this case is 1.773. We can conclude that the bias-only model performs substantially better than the similarity-based models we examined previously.

We then further improve the MSE with the complete latent factor model. In this case, we use $K = 2, 3, 4, 5$ factors. The loss function is minimized using L-BFGS-B. The MSE on the test set using different regularization constant values for each number of factors for the complete latent-factor model can be seen in Table 6.

Regularization Constant λ	Factors K			
	2	3	4	5
0.01	2.203	2.203	2.203	2.203
0.005	2.186	2.186	2.186	2.186
0.001	2.096	2.096	2.096	2.096
0.0005	2.030	2.030	2.030	2.030
0.0001	1.842	1.842	1.842	1.842
0.00005	1.779	1.777	1.775	1.773
0.00001	1.831	1.812	1.788	1.798
0.000005	1.907	1.817	1.809	1.837

Table 6. Evaluation of the different regularization constant values on the validation set for the complete latent-factor model with $K = 2, 3, 4, 5$ factors

As can be seen from Table 6, the MSE is minimized using $K = 5$ factors and regularization constant of $\lambda = 0.00005$. The reported MSE on the test set in this case is 1.797. From the results above, we can deduce that the bias-only model performs better for the given dataset as compared to the complete latent factor model. However, both of the latent factor models achieved a lower MSE than the baseline similarity-based models.

Model	Dataset	MSE
Bag-of-Words	Training	1.582
	Validation	1.581
	Testing	1.601
TF-IDF	Training	1.383
	Validation	1.382
	Testing	1.403
Elman RNN	Training	1.422
	Validation	1.428
	Testing	1.431
LSTM	Training	1.033
	Validation	1.113
	Testing	1.110
GRU	Training	0.965
	Validation	1.064
	Testing	1.082

Table 7. Evaluation of Text-based Methods

Model	Feature Size	Validation MSE
Bag-of-Words	250	1.785
	500	1.581
	750	1.633
	1,000	1.601
TF-IDF	250	1.642
	500	1.509
	750	1.451
	1,000	1.382

Table 8. Feature sizes vs. MSE in BoW and TF-IDF

5.3. Bag-Of-Word Model

We tried different numbers of features and based on the computational tractability and overall performance, we decided to use the top 1,000 words as features. Table 8 shows MSEs on the validation set for different feature sizes.

Using the training samples, we trained a linear regression model provided in the sklearn.linear_model package. The MSE on the training, validation, and test dataset are reported in Table 7. In Table 9, we mention words and the coefficients the model learned for them. As we can see, the model does a good job of assigning positive words with positive coefficients, neutral words with near-zero coefficients, and negative words with negative coefficients.

To further optimize the model, we fit a Regularized Linear Regression model, but it did not affect the overall performance and gave almost similar results. This may be because the features are sparse.

Word	Coef.	Word	Coef.
glad	0.313	amazing	0.278
highly	0.267	satisfied	0.260
fantastic	0.257	toyota	-0.00017
enough	0.000339	actually	-0.00040
able	-0.00072	worse	-0.323
disappointed	-0.370	poor	-0.378
horrible	-0.434	worst	-0.790

Table 9. Coefficients of Bag-of-Words Model

Word	Coef.	Word	Coef.
glad	1.878	amazing	2.049
highly	1.859	satisfied	1.177
fantastic	1.511	toyota	0.176
enough	0.031	actually	0.304
able	0.027	worse	-2.640
disappointed	-1.877	poor	-2.605
horrible	-2.483	worst	-4.573

Table 10. Coefficients of TF-IDF Model

5.4. TF-IDF

We use the TfidfVectorizer tool provided in the sklearn.feature_extraction.text package for build our model. After experimenting with the number of features to include, we decided to choose a maximum of 1000 features. Table 8 shows MSEs on the validation set for different feature sizes. We could not run experiments for a greater value of feature size because of limited computational power.

Using the TF-IDF representation of the reviews, along with their ratings, we train a linear regression model provided in the sklearn.linear_model. The MSE on the training, validation, and test dataset are reported in Table 8. In Table 10, we have mentioned the same words as Table 9 and the coefficients the TF-IDF model learned for them. As we can see, the TF-IDF does a better job than BoW of assigning words' coefficients based on their meaning.

We tried optimizing the model with Regularized Linear Regression, but due to sparsity issues, we could not get a better performance.

5.5. Elman RNN

In our experiments, we randomly initialize the word embedding from a standard normal distribution. We choose an embedding dimension D of 256 and a hidden state size of 256. We train our model on the training dataset for 10 epochs using the Adam optimizer with a learning rate of 0.001. The training loss and validation loss for each epoch are plotted in Fig 10. We can see that the model starts overfitting after 8^{th} epoch. We save the model parameters at 8^{th} epoch to test its performance on the test dataset. The performance of the model is mentioned in Table 7.

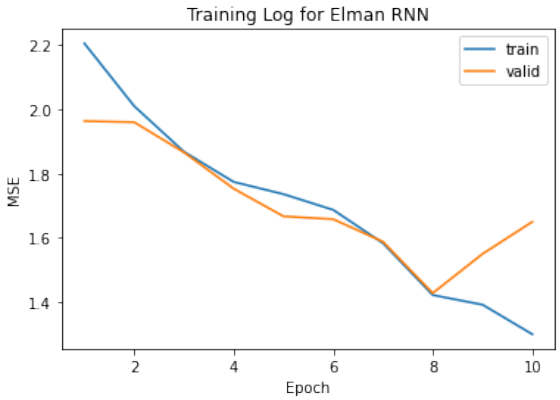

Figure 10. Training Log for Elman RNN

5.6. LSTM and GRU

We choose the same hyperparameters and optimizer as the Elman RNN model and trained an LSTM and a GRU model for 10 epochs. The training and validation loss for each epoch and model is plotted in Fig 11.

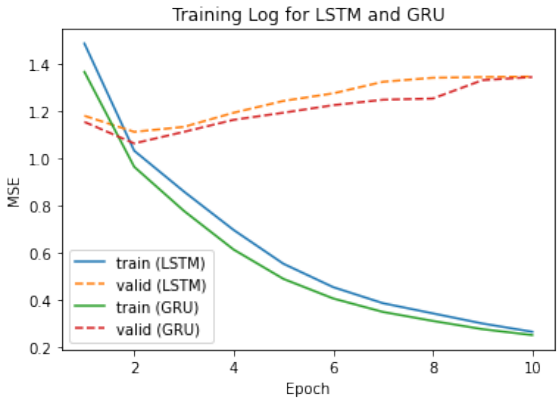

Figure 11. Training Log for LSTM and GRU

It is clear that both the models give similar performance, with GRU being slightly better than LSTM. The training, validation, and test errors for both these models at 2^{nd} epoch are mentioned in Table 7.

Interestingly, comparing Fig 10 and Fig 11 show how powerful LSTMs and GRUs are as compared to

Elman RNN in learning the underlying structure of the text to predict rating. Elman RNN took 8 epochs to give the optimal performance, whereas both LSTM and GRU took only 2 epochs to do the same.

Optimizing GRU

Since GRU gave the best performance among all the other models, we spent some time experimenting with different embeddings, and architecture settings to see if we can get better performance. Due to limited computational power and time, we could not do an exhaustive search, but here are some of the settings we tried:

1. **Using Glove Embeddings**
   Instead of learning the embeddings on the go, we use pre-trained word embeddings. One of the most common embeddings used by researchers is Glove Embedding. We freeze the embeddings during training to save time and tried different embedding dimensions. The results, and corresponding settings, are mentioned in Table 11.

2. **Using Dropout**
   Dropout is a regularization technique used in neural networks to reduce overfitting. It accomplishes that by randomly dropping out nodes during the training process. We apply dropout layers in the final feed-forward network layer in our model (refer to Fig 7). We train our model for different values of dropout, and the results, along with the corresponding dropout value, are described in table 11.

As we can see from Table 11, using pre-trained Glove Embedding with an embedding dimension of 300 produces the best MSE. Interestingly, it can be seen that as we keep on increasing the embedding dimension, the MSE keeps decreasing. This is because bigger embeddings carry more information which helps the model "understand" more.

Interpreting Word Embeddings

During the training process, our original (base setting) GRU model learned word embeddings. Now we turn our discussion to see how our model uses these embeddings to encode a review and form a decision. For this, we will run our model on different reviews, check the similarity of the generated encoding, and the closeness between the predicted rating.

Setting	Dataset	MSE
Glove Embedding (D=50)	Training	1.106
	Validation	1.117
	Testing	1.128
Glove Embedding (D=200)	Training	0.968
	Validation	1.021
	Testing	1.092
Glove Embedding (D=300)	Training	0.882
	Validation	0.989
	Testing	0.994
Dropout (p=0.2)	Training	1.038
	Validation	1.105
	Testing	1.124
Dropout (p=0.4)	Training	0.948
	Validation	1.100
	Testing	1.124
Dropout (p=0.6)	Training	1.055
	Validation	1.093
	Testing	1.105

Table 11. Evaluating GRU under different settings

Encoding a review is simple. We pass the review through our trained GRU, and simply use the final hidden state (i.e h_T) as the review's encoded representation. We now take two reviews, encode them, check the cosine similarity between the encoding and see what predictions the model generated for them. Given below is an example for two reviews who carry similar tone and similar ratings:

1. Sentence 1: "i bought this dishwasher because i had good service from a maytag dishwasher in a previous house boy have their products changed". Rating: 1.0. Prediction: 1.77

2. Sentence 2: "... please dont waste your time on this like i have to i hope this helps people make better choices good luck i wish someone could have forecasted this for me". Rating: 1.0. Prediction 1.03

3. Cosine Similarity between Encodings: 0.4697

As we can clearly see, both reviews carry a negative tone and hence have high similarity scores for their encodings. Consequently, the model accurately predicted their ratings. Given below is another example of two ratings that carry similar tones, but dissimilar ratings:

1. Sentence 1: “bought it in dec 2001 with 48k miles on it have since put on 120k additional miles in all that time ive had to replace several batteries this car seems hard on them ...”. Rating: 5.0. Prediction: 2.34

2. Sentence 2: “i have had this machine for about 2 years while it is a moderately decent printer though slow it is frequently in trouble needs to rest or reboot or whatever makes it happy it uses ink fast is hard to set for black only printing insists on the long way around the block when doing anything it seems”. Rating: 2.0. Prediction 2.53

3. Cosine Similarity between Encodings: 0.5001

It can be seen that both users are disappointed with their products, but still, the first user gives a 5.0 rating (when the item seems worthy of just 2.0 or 3.0) while the other one gives a 2.0. However, our GRU model was still able to generate similar encodings for both reviews and assign them similar ratings. Next, given below is an example of two ratings with different tones, and different ratings.

1. Sentence 1: “i recieved a bop it when i was around nine years old i was soo excited about it my siblings and i loved it i was a great way for us to spend time together and not be at each others throats it has a lot of great features”. Rating: 4.0. Prediction: 4.35

2. Sentence 2: “i recieved a bop it when i was around nine years old i was soo excited about it my siblings and i loved it i was a great way for us to spend time together and not be at each others throats it has a lot of great features...”. Rating: 1.0. Prediction 1.41

3. Cosine Similarity between Encodings: 0.0884

The first review carries a positive tone, whereas the second review carries a negative tone. As we would expect, the cosine similarity between the encodings of the two reviews is low. Consequently, the model gave accurate predictions for the ratings.

By these examples, it can be safely assumed that if the dataset is cleaner and less “noisy”, our model may be able to give a better performance in terms of evaluation metrics. However, finding such clean and ”noiseless” is really difficult when dealing with the real world.

6. Conclusion

In this project, we looked at different models which can be used for the Ratings Prediction task. We started from the most basic models taught in the course, and gradually moved to more complex ones, and saw how text-based methods outperform the other. Based on the comprehensive analysis presented in this paper, here are some conclusions:

1. Similarity-based Rating Predictions fail when the user-item interactions are sparse. As we saw in section 4.1, using user-user similarity instead of item-item similarity performed poorly due to this exact reason.

2. Text-based methods, even the most basic one like BoW model, outperforms fairly complex methods like Latent-Factor Model.

3. In text-based methods, using word embeddings as features, as we did for RNNs, gave the best performance of all. Word embedding conveys more information and helps make better prediction as compared to simple feature representations like BoW. Additionally, word embeddings do not face the sparsity issues like BoW and can be used to represent any review in a low-dimensional space without worrying about loosing rarer, but more expressive, words.

Moreover, we saw how using pretrained Glove Embedding resulted in a better working model. Unfortunately, due to lack of enough computational resources, we could not experiment tuning the hyperparameters and doing an exhaustive search to reduce the MSE even more. However, we believe that trying out other domain-specific pre-trained embeddings, optimizers, hyperparameters (for e.g batch size, hidden size, learning rate, inner dimesions of FNN, etc) can improve our GRU model even further. We leave this as an avenue for future improvements.

References

[1] Stephen CF Chan Cane WK Leung and Fu lai Chung. Integrating collaborative filtering and sentiment analysis: A rating inference approach. *Proceedings of the ECAI 2006 workshop on recommender systems*, 2006. 10

[2] Julian McAuley Chenwei Cai, Ruining He. Spmc: Socially-aware personalized markov chains for sparse sequential recommendation. *IJCAI*, 2017. 1, 10

[3] James H. Martin Dan Jurafsky. Speech and language processing (3rd ed. draft). In *Prentice Hall PTR*, 2021. 8, 9

[4] J. L. Elman. Finding structure in time. *Cognitive science*, 1990. 8

[5] H. Jin K. Zhao Q. Yang F. Li, N. Liu and X. Zhu. Incorporating reviewer and product information for review rating prediction. *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, 2011. 10

[6] Mingming Fan and Maryam Khademi. Predicting a business star in yelp from its reviews text alone. *arXiv preprint arXiv:1401.0864*, 2014. 10

[7] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. *In The World Wide Web Conference (WWW '19). Association for Computing Machinery*, 2019. 10

[8] Fethi Fkih. Similarity measures for collaborative filtering-based recommender systems: Review and experimental comparison. *Journal of King Saud University - Computer and Information Sciences*, 2022. 5

[9] Gianmario Spacagna Peter Roelants Valentino Zocca Ivan Vasilev, Daniel Slater. Python deep learning - second edition. In *Packt Publishing*, 2019. 10

[10] KyungHyun Cho Yoshua Bengio Junyoung Chung, Caglar Gulcehre. Empirical evaluation of gated recurrent neural networks on sequence modeling. *Presented in NIPS 2014 Deep Learning and Representation Learning Workshop*, 2014. 9

[11] G. Ifrim L. Qu and G. Weikum. The bag-of-opinions method for review rating prediction from sparse text patterns. *Proceedings of the 23rd International Conference on Computational Linguistics*, 2010. 10

[12] Jürgen Schmidhuber Sepp Hochreiter. Long short-term memory. *Neural Computation*, 1997. 9

[13] Irwin King Tong Zhao, Julian McAuley. Improving latent factor models via personalized feature projection for one-class recommendation. *Conference on Information and Knowledge Management (CIKM)*, 2015. 1, 10

[14] Yan Wu Ziheng Cui Tingting Feng Wentao Zhao, Huanhuan Tian. A new item-based collaborative filtering algorithm to improve the accuracy of prediction in sparse data. *International Journal of Computational Intelligence Systems*, 2022. 5