Predicting Ratings from Mystery-Thriller-Crime Goodreads Reviews

CSE 158

Courtney Cheung | c6cheung@ucsd.edu

1. Dataset and EDA

In this project, I analyzed Goodreads reviews within the Mystery-Thriller-Crime genre (data: https://sites.google.com/eng.ucsd.edu/ucsdbookgraph/home). Goodreads is a website in which users can browse a database of books, quotes, and reviews. Users can also create libraries, have discussions, and leave their own reviews.

Out of the 1,849,236 detailed reviews from the original Mystery-Thriller-Crime genre dataset, I loaded in 150,000. I split the data into three sections of length 50,000 to use as training, validation, and testing sets. After the first attempt, I observed that there were few overlaps in users between the sets because the original data was sorted by users. This was a problem because I wanted to eventually create some models that factored in user history, and these would be trained on the training set and tested on the test set. To allow for a more accurate evaluation of the efficacy of my models, I shuffled the data to create more overlap between users in the test set, users in the validation set, and users in the training set:

```
#importing data
dataset = []
z = gzip.open("goodreads_reviews_mystery_thriller_crime.json.gz")
for l in z.readlines():
    dataset.append(eval(l))
    if len(dataset) == 150000:
        break
random.seed(1)
random.shuffle(dataset)
```

After shuffling, the validation set and test set had over 6300 users that were also in the training set:

```
trainUsers = set([d['user_id'] for d in train])
validUsers = set([d['user_id'] for d in valid])
testUsers = set([d['user_id'] for d in test])
```

```
len(trainUsers.intersection(validUsers))
```

```
6398
```

```
len(trainUsers.intersection(testUsers))
```

```
6406
```

Each review has the attributes: ‘user_id’, ‘book_id’, ‘review_id’, ‘rating’, ‘review_text’, ‘date_added’, date_updated’, ‘read_at’, ‘started_at’, ‘n_votes’, ‘n_comments’. An example of a review datapoint is as follows:

{'user_id': '8842281e1d1347389f2ab93d60773d4d',
'book_id': '6392944',
'review_id': '5e212a62bced17b4dbe41150e5bb9037',
'rating': 3,
'review_text': "I haven't read a fun mystery book in a while and not sure I've ever read Poirot. Was looking for a fun read set in France while I was on holiday there and this didn't disappoint! Fast paced and good mystery. \n One that struck me was how similar Poirot is to Sherlock. They are both detectives, have a ex-military sidekick who is telling the story, and solve mysteries using their superior wit. Poirot seems like a French Sherlock. I'm curious if he was inspired by Sherlock.",
'date_added': 'Mon Jul 24 02:48:17 -0700 2017',
'date_updated': 'Sun Jul 30 09:28:03 -0700 2017',
'read_at': 'Tue Jul 25 00:00:00 -0700 2017',
'started_at': 'Mon Jul 24 00:00:00 -0700 2017',
'n_votes': 6,
'n_comments': 0}

I was most interested in analyzing the relationships between user id (uniquely identifies a Goodreads user), book id (uniquely identifies a book), star rating, and review text. To get a better sense of what the training data looks like with regard to these attributes and potential prediction features, I performed the following exploratory data analysis.

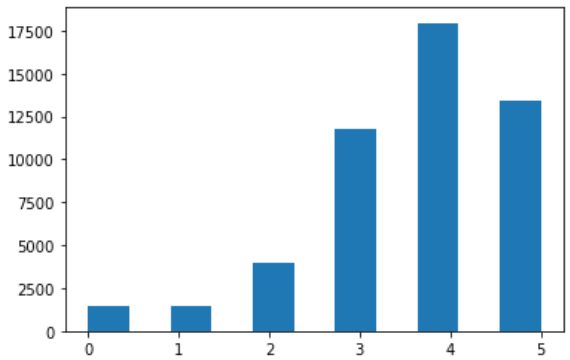

Figure 1: Distribution of Ratings

The star ratings ranged from 0 to 5. The most common rating was 4 stars, at around 17500 reviews. There were less than 5000 ratings of 0, 1, and 2 stars each, while there were more than 10000 ratings of 3 and 5 stars. The distribution indicates that the training data was slightly unbalanced in frequency of each rating.

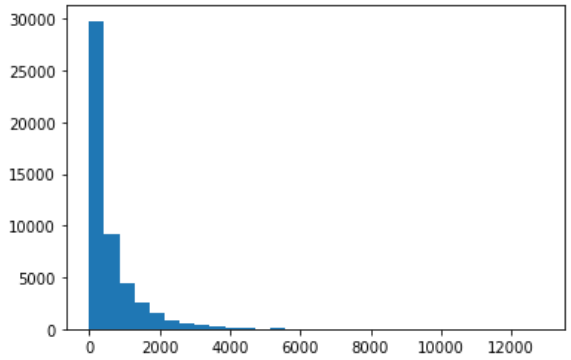

Figure 2: Distribution of Review Lengths

The majority of reviews in the training data contained less than 2000 characters. Some outlier reviews had up to 12861 characters.

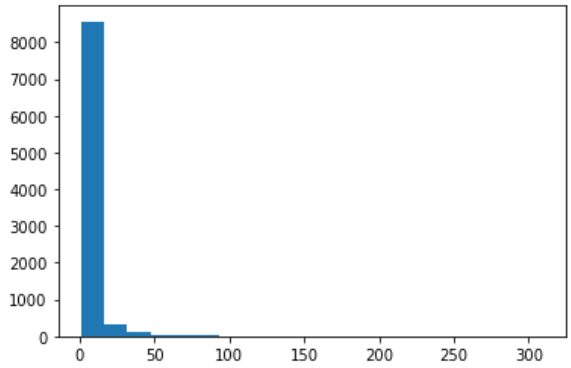

Figure 3: Distribution of Number of Reviews Per User

Most users reviewed less than 100 books, while some outliers had reviewed up to 309.

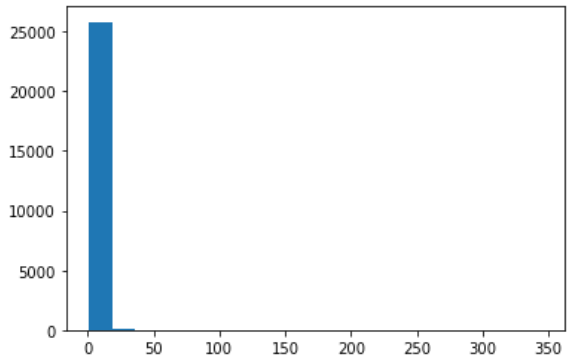

Figure 4: Distribution of Number of Reviews Per Book

Most books had less than 30 reviews, while some outliers had up to 345 reviews.

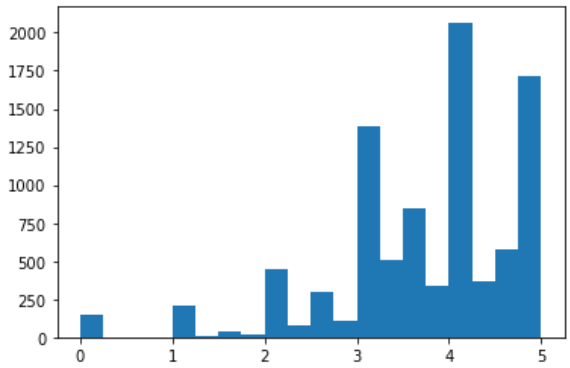

Figure 5: Distribution of User Rating Averages

The most common average user rating was around 4 stars, while the global average of the user ratings was 3.673.

Common Unigrams	Common N-grams
'The', 'and', 'a', 'to', 'i', 'of', 'is', 'in', 'it','this', 'was', 'that', 'book', 'but', 'for', 'with', 'as', 'her', 'read', 'not', 'on', 'you', 'one', 'story', 'be', 'have', 'are','his', 'me', 'so'	'the', 'and', 'a', 'to', 'i', 'of', 'is', 'in', 'it', 'this', 'was', 'that', 'book', 'but', 'for', 'with', 'as', 'her', 'of the', 'read', 'not', 'on', 'you', 'one', 'story', 'in the', 'be', 'have', 'are', 'his'

Table 1: Common Unigrams and N-grams

Lastly, I calculated the 1000 most common unigrams, as well as the 1000 most common N-grams ranging from N=1 to N=5. A subset of those are listed in Table 1.

2. Predictive Task

For a given user-book pair in the mystery-thriller-crime genre, I predicted the star rating using various models.

I tuned hyperparameters by minimizing mean squared error on the validations set, and evaluated models by comparing their mean squared errors on the training and test sets.

First, I created a baseline model which made predictions based on the user's average rating. If the user was not seen in the training set, the model would predict the global average (average of all ratings in the training set).

To create an improved model, I tried out combinations of the following strategies.

Feature vectors for ridge regression:

1. Bag-of-words counts for the 1000 most common unigrams
    a. Data processing: (1) Iterating through and splitting up training reviews into individual words in order to update a word-count dictionary, (2) identifying common unigrams, (3) processing training reviews through a feature function that returns a list of counts for each common unigram.
2. Bag-of-words counts for the 1000 most common N-grams (N=1 to N=5)
    a. Data processing: (1) Iterating through and splitting up training reviews into N-grams, where N={1, 2, 3, 4, 5}, order to update an N-gram-count dictionary, (2) identifying common N-grams, (3) processing training reviews through a feature function that returns a list of counts for each common N-gram.
3. Term Frequency - Inverse Document Frequency (TF-IDF) values for the 1000 most common unigrams. TF-IDF was defined as the number of times a term appeared in the review, multiplied by the log of the inverse of the proportion of documents containing the term.
    a. Data processing: (1) Iterating through and splitting up training reviews into individual words in order to update a word-count (term frequency) dictionary, (2) identifying common unigrams, (3) iterating through and creating a set of words in the training reviews in order to update a document frequency dictionary, (4) processing training reviews through a feature function that returns a list of TF-IDF values for each common unigram.
4. Popularity, defined as the number of reviews for an item, divided by the number of total reviews. If the item was not seen in the training data, the model would use the global average popularity (average popularity of training items).
    a. Data processing: (1) creating a popularity-per-item dictionary by iterating through item ratings, calculating the number of ratings divided by the total for each, (2) calculating average popularity using popularity-per-item dictionary, (3) processing training reviews through a feature function that returns popularity.
5. Star Rating Proportions. If the item was not seen in the training data, the model would use the global average popularity (average popularity of training items).
    a. Data processing: (1) creating a rating proportion dictionary by iterating through item ratings, creating a list of proportions of 1, 2, 3, 4, and 5 star ratings for each, (2) calculating average rating proportions by iterating through rating proportion dictionary, (3) processing training reviews through a feature function that returns a list of rating proportions.

Other heuristics using user-item interactions:

1. Rating associated with the most similar review (cosine similarity of TF-IDF vector of 10 most common unigrams)
2. $$r(u,i) = \bar{R}_i + \frac{\sum_{j \in I_u \setminus \{i\}} (R_{u,j} - \bar{R}_j) \cdot \text{Sim}(i,j)}{\sum_{j \in I_u \setminus \{i\}} \text{Sim}(i,j)}$$
   where Sim = Jaccard Similarity between items, defined as the number of users who reviewed both i and j, divided by the sum of the number of users who reviewed i and the number of users who reviewed j.
3. $$r(u,i) = \bar{R}_i + \frac{\sum_{j \in I_u \setminus \{i\}} (R_{u,j} - \bar{R}_j) \cdot \text{Sim}(i,j)}{\sum_{j \in I_u \setminus \{i\}} \text{Sim}(i,j)}$$,
   where Sim(i, j) is the Cosine Similarity between Item2Vec item representations, defined as the cosine of the angle between the vectors. The vectors representing items are composed of similarity scores of other items, determined using the Word2Vec embedding from Gensim. In other words, “. Just as word2vec discovers which words appear in the same context in a sentence (essentially ‘synonyms’), item2vec learns item representations γi that are capable of predicting which items occur in the same context in an interaction sequence” (Personalized Machine Learning, pg. 120).

3. Model

While the baseline model was not computationally expensive, it relied heavily on how many users are shared between reviews in the test set and the training set.

To improve upon this model, I tried out strategies that utilized both sentiment analysis and user-item interactions.

I started by performing ridge regression using Bag-of-words and TF-IDF feature vectors. Using the 1000 most common

words (unigrams), I created a feature vector composed of counts of how many times each unigram appeared in a review. Then I tuned the hyperparameter, lambda, against the validation set, which came out to be 279, indicating the amount of bias in the model. Similarly, I tried out a feature vector of counts using the 1000 most common N-grams ranging from N=1 to N=5. The purpose was to see if unigrams or a combination of N-grams had more predictive qualities. The N-grams turned out to be more computationally expensive, and higher in MSE.

Because the N-gram model was not as effective, I built upon the unigram model by creating a unigram feature vector using TF-IDF values rather than counts. The TF-IDF approach differs from the counts approach because TF-IDF factors in the context and importance of each term. Terms that appear frequently in a review are only important if they are rare overall. This decreases the impact of common, yet non informative words, such as “and” and “the”. Overall, this performed slightly better than the counts approach on the training data, but worse on the testing data.

Next, I created two more features: popularity and star rating proportions. To optimize MSE, I tested out different combinations of features out of the baseline predictions, Bag-of-words unigram counts, unigram TF-IDF values, popularity, and star rating proportions. I will touch on strengths and weaknesses of this later on.

Next, I tried out Heuristic 1. The idea was that, given a user-book pair and its review text, the model would locate the review containing the most similar text to the given one. Then the model would use its rating as the prediction. Similarity between review texts was calculated with Cosine Similarity, using the vectors containing TF-IDF values of the 10 most common unigrams as the text representations. This model was not successful, as it was inefficient and performed worse than the baseline in both training and testing. One strength was that there was only a small discrepancy between training and testing results.

While I was previously able to improve upon the baseline model by incorporating a combination of features, I wanted to see how user-item interaction heuristics performed in comparison. Heuristic 2 utilized a user-based Jaccard similarity score to measure similarity between items. It performed well in comparison to the baseline model on the training data. However, it performed the worst on the testing data. This indicates that this model has a tendency for overfitting. Similar to the baseline model, a weakness of heuristic 2 is that it relied too heavily on the interactions in the seen data, causing it not to generalize well to unseen data.

Lastly, I tested Heuristic 3. This model used Item2Vec item representations, which is a variation of Word2Vec in which “documents” are items. Initially, this model performed poorly because of negative rating prediction values. An improvement I made to this model was replacing the negative similarity scores with a value of .01. I will touch on the strengths and weaknesses of this later on.

Model	**MSE (training)**	**MSE (testing)**
Baseline (User Averages)	0.810	1.258
Bag-of-words Unigrams	1.119	1.183
Baseline + Bag-of-words Unigrams	0.688	1.074
Bag-of-words N-grams	1.128	1.191
TF-IDF Unigrams	1.114	1.187
Baseline + Bag-of-words Unigrams + TF-IDF Unigrams	0.686	1.071
Baseline + Bag-of-words Unigrams + TF-IDF Unigrams + Popularity	0.686	1.071
Baseline + Bag-of-words Unigrams + TF-IDF Unigrams + Star Rating Proportions	.684	1.065
Heuristic 1 (Rating of Most Similar Review)	3.527	3.61
Heuristic 2 (Jaccard Similarity)	0.640	1.732
Heuristic 3 (Item2Vec)	.821	.825

Table 2: Training and Testing Performance of Models

Table 2 shows the MSE of the training and testing set on each combination I tried out. Overall, there were only issues with overfitting, as all of the testing MSE values were lower than their corresponding training MSE values. Highlighted are the best performing models.

The ridge regression model with a feature vector containing user averages (or global average), counts and TF-IDF values of the 1000 most common unigrams, and star rating proportions, had the best performance on the training set and the second best performance on the test set. The third

heuristic, which used Item2Vec, had the best testing performance, and the second lowest training performance. While performance on testing data is important, the third heuristic relied on having seen some of the testing data, while the ridge regression model relied solely on training data. On the other hand, the ridge regression model contains the largest feature vector, showing that it has a tendency for overfitting. Overall, however, I would consider the ridge regression model to be my final model because of its performance on unseen data.

4. Literature

The dataset I used was collected in 2017 by scraping public shelves on goodreads. User id's and review id's were anonymized. This dataset was originally used in the paper, "Item Recommendation on Monotonic Behavior Chains" to analyze the spectrum of explicit and implicit feedback in recommender systems ranging from "clicks and purchases, to ratings and reviews", and to see if implicit signals might help predict explicit signals (Item Recommendation on Monotonic Behavior Chains, pg. 1).

Datasets similar to this one have been studied in the past in the context of rating predictions, by scraping Amazon, Yelp and other sites. I will explore some sentiment analysis strategies and state-of-the-art methods used for rating predictions and analysis of reviews, and discuss how they relate to my final model.

In particular, I will be referencing the paper, "Leveraging sentiment analysis at the aspects level to predict ratings of reviews", in which the authors develop a rating prediction model inspired by the observation that "many reviews that are spread across forums or on social media are written in plan text, which is not rated".

The first distinction between my model and the model developed in the paper, is that their "cumulative logit" model deals with ordinal labels, while my ridge regression interprets ratings as a sliding scale.
In this study, they first address some common state-of-the-art methods of classification, such as Naive Bayes and Support Vector Machine, as well as approaches that combine machine-learning methods and a lexicon to better incorporate the context of words into determining positive and negative reviews.

The authors first use reviews derived from e-commerce websites as a training set. They then developed a variant of the Conditional Random Field model, called SentiCRF, which obtains "a collection of term pairs that are attached to sentiment scores." These term pairs exclude stop words, which is something I could have also done in the preprocessing phase of my unigram and N-gram models. Next, they observed a class imbalance, in which the distribution of ratings was unbalanced, as were the ratings in the dataset I used. They addressed this by creating a heuristic resampling algorithm. This re-sampling improved accuracy on the Yelp dataset by about 7%. They made comparisons between the baseline CLM model and svm-based multiclass classifiers, among others. Lastly, they developed a rating prediction model for non-rated reviews, such as tweets, by extracting term pairs and mapping them to the term pairs generated by the SentiCRF.

A main difference between their takeaways and my conclusions, is that class imbalance should be addressed in preprocessing, as it greatly affects accuracy of the model. Overall, their lowest RMSE achieved was .860, which is close to my Item2Vec model. However, their baseline model outperformed my final ridge regression model.

5. Conclusion

My final ridge regression model had an MSE of .684 on the training set, and 1.065 on the test set. As shown in Table 2 of section 3, this performed better than 7 other combinations of feature vectors, as well as two other heuristics I tested out. The feature representations that worked well were the user averages, Bag-of-words unigrams, TF-IDF values for unigrams, and proportions of star ratings. The bag-of-words feature representation was similar in performance to the unigrams, but was more computationally expensive and slightly more erroneous. The popularity feature was also eliminated because it had no predictive effect on the model, as the MSE was the same with or without it. In comparison to the last heuristic, the final ridge regression model had a higher testing MSE, however, I eliminated this heuristic because it could not be accurately compared due to its utilization of testing data in training the model. Overall, the final model succeeded because it used the right combination of predictive features involving both sentiment analysis and user-item interactions, while other models did not utilize features or heuristics with the most predictive effect.

The final model's parameters show that the frequency of common words and the importance of common words in a review, as well as the user's average rating, and proportion of each type of rating for the particular book, can predict the user's rating of that book with an MSE of around 1.065 on unseen data.

To improve the final model in the future, I could try combining the sentiment analysis features and the Item2Vec results into one feature vector. Furthermore, I could create a heuristic using Item2Vec in which only training data is used, and items in the test set that were not seen in the training set could be dealt with by predicting an average of some kind. In addition, I could analyze how other attributes of the reviews could factor into rating prediction. For

example, I could add features such as review length, year, or time of day.

References

Jiangtao, Qiu, et al. “Leveraging Sentiment Analysis at the Aspects Level to Predict Ratings of Reviews.” *Information Sciences*, Elsevier, 7 Apr. 2018, https://www.sciencedirect.com/science/article/abs/pii/S0020025516317303?casa_token=l1fuE1x0hbkAAAAA%3Ar_j_TJcgpys12Y8aJ6vsoD2BbzEKPubCnlVGTM4jM3soLPLOh62bsyeevKRiKRezpflssqtV3g.

McAuley, Julian. *Personalized Machine Learning*. Cambridge University Press, 2022.

Mengting Wan, Julian McAuley, "Item Recommendation on Monotonic Behavior Chains", in RecSys'18.

Mengting Wan, Rishabh Misra, Ndapa Nakashole, Julian McAuley, "Fine-Grained Spoiler Detection from Large-Scale Review Corpora", in ACL'19.