CSE158 Project

Table of Contents

Part 1: Dataset Description

This dataset consists of reviews of video games from amazon, which is an updated version of the Amazon review dataset released in 2014. It includes reviews (ratings, text, reviewerID etc.), along with the product metadata. The data date spans a period of 22 years, from May 1996 to Oct 2018. Reviews include the overall(rating), verified, review Time, reviewer ID, asin(item id), reviewer Name, review Text, summary, and unixReviewTime. The original dataset from "https://nijianmo.github.io/amazon/index.html" contains 233.1 million reviews, which has 34 GB of raw review data. Due to the scope of this study, computing power, and ram availability, dealing with this amount of raw data is obviously impossible. To reduce the data size without losing the significance of the review data, we decide to use 5 cores for the category Video Games; a subset of the data in which all users and items have exactly 5 reviews.

Data includes:
Reviews from May 1996 to Oct 2018
497,577 reviews
17408 reviewed items
55223 users

We load the dataset and give it a name. Then we check data fields by a sample. From the sample, we can see the data contains the following basic fields:

- reviewerID - ID of the reviewer, e.g. A1HP7NVNPFMA4N
- Verified - True if the user actually purchases the item, eg. True,False
- asin - ID of the product, e.g. 0700026657
- reviewerName - name of the reviewer
- reviewText - text of the review
- overall - rating of the product
- summary - summary of the review
- unixReviewTime - time of the review (unix time)
- reviewTime - time of the review (raw)

Convert data to data frame to have a better look. And we found there are 497577 rows of data. The overall rating and unixReviewTime are numeric, and others are strings. The overall rating can be used for rating prediction tasks.
We check the info of the dataset.

Data columns (total 12 columns):
 # Column Non-Null Count Dtype
--- ------ -------------- -----
 0 overall 497577 non-null float64
 1 verified 497577 non-null bool
 2 reviewTime 497577 non-null object
 3 reviewerID 497577 non-null object
 4 asin 497577 non-null object
 5 reviewerName 497501 non-null object
 6 reviewText 497419 non-null object
 7 summary 497468 non-null object
 8 unixReviewTime 497577 non-null int64
 9 vote 107793 non-null object
 10 style 289237 non-null object
 11 image 3634 non-null object
dtypes: bool(1), float64(1), int64(1), object(9)

The data also contains other fields such as vote, style, image, which have a lot of missing values, but contribute few meanings. These fields will be ignored. Since the data have been filtered by 5-core, each item/user has exactly 5 data.

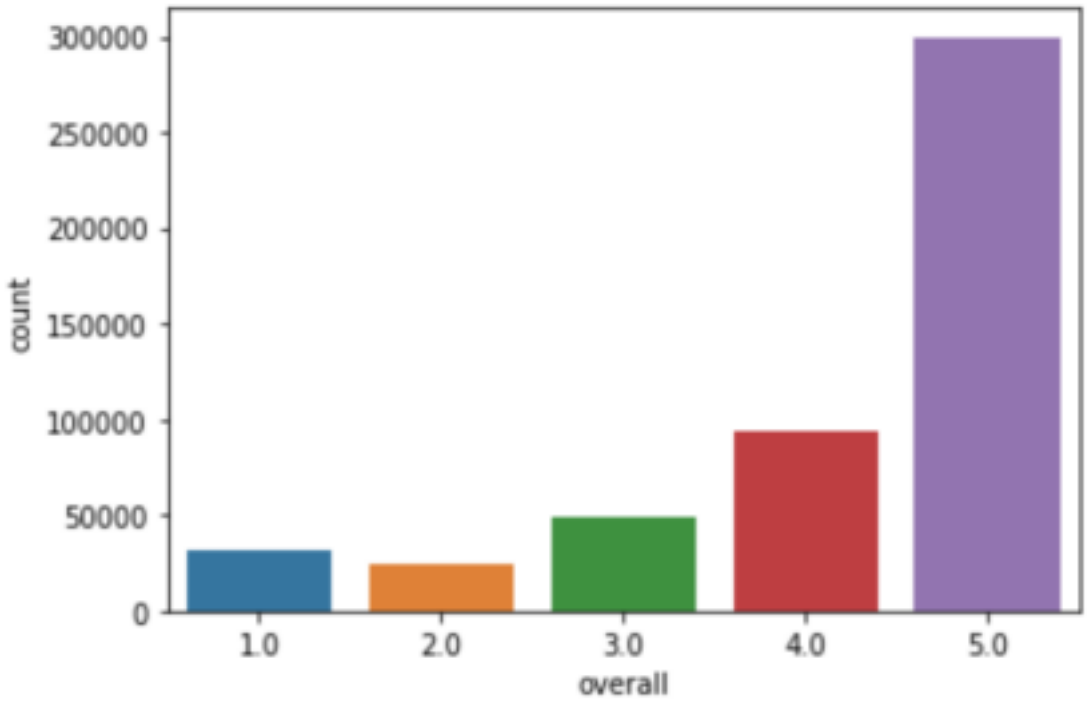
Exploratory Analysis

To begin with, we extra raw review data from json.gz file to list objects. Then, we build some data structures for future processes: usersPerItem(set), itemsPerUser(set), reviewsPerUser(list), reviewsPerItem(list), ratingDict(dict). By assuming the 'overall' rating for each review and dividing it by the length of total reviews, we got the rating mean of 4.22. There are 31715 reviews that give a star rating higher than the rating mean, and 23508 reviews that give a star rating below the rating mean. Amazon uses a 5-star rating system, 4.22 stars out of 5 stars, and most of the reviews tend to give a higher rating than the mean. This looks reasonable, the item sales on amazon that has at least 5 reviews should include more high-rating items, otherwise, that item would not have enough reviews to occur in this 5-core dataset. Other than the overall rating, we also have an eye on the review text. The review text length mean is 59.37. There are 43687 reviews that have review text lengths greater than the mean, while 11536 reviews' review text lengths are less than the mean. It is interesting to observe the amount of review text length that is greater than the mean is 32151 more than those below the mean. Reviews that only have a few words might not be helpful for our prediction task, we may filter out those outliers during the next section.

Overall Distribution

For checking the overall distribution of all the reviewers and only verified reviewers's overall distribution. We made two plots for comparison.

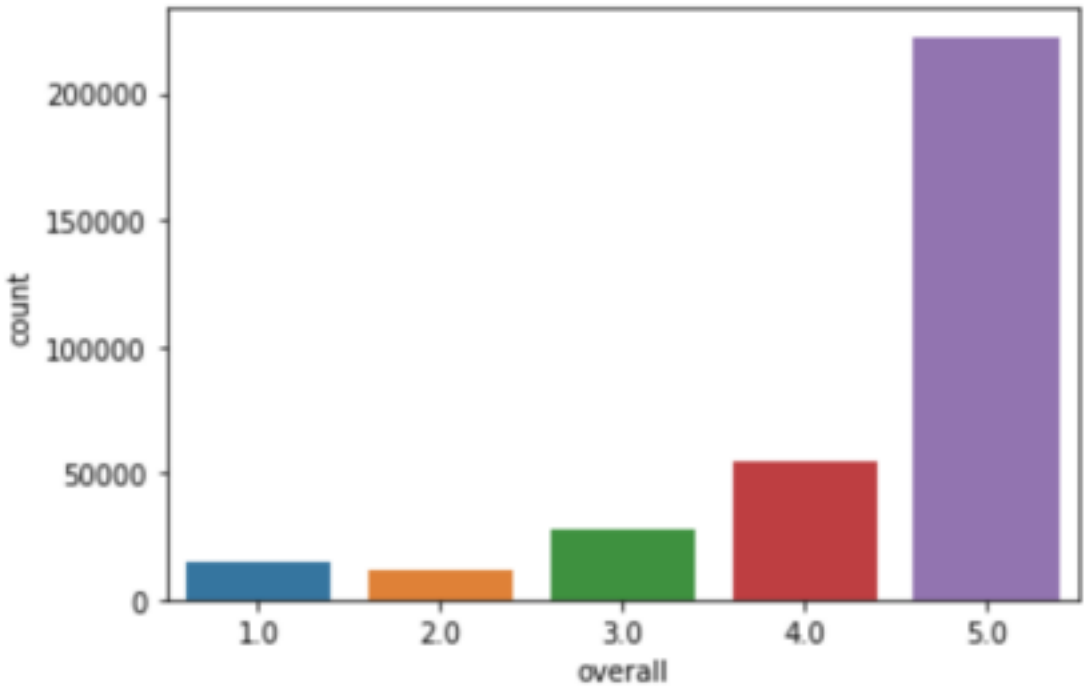
Overall Distribution Ignored Verified



The above plot shows the overall distribution ignored.

For overall at 1.0, the probability is 0.045700371266665665
For overall at 2.0, the probability is 0.03643523876805604
For overall at 3.0, the probability is 0.08410768236408182
For overall at 4.0, the probability is 0.16499872236167687
For overall at 5.0, the probability is 0.6687579852395196

Overall Distribution Verified Reviewers



The above plot shows the overall distribution only from verified reviewers.

For overall at 1.0, the probability is 0.062066775594531096
For overall at 2.0, the probability is 0.0485050554989479
For overall at 3.0, the probability is 0.0987706425337184
For overall at 4.0, the probability is 0.18822011467571853
For overall at 5.0, the probability is 0.6024374116970841

One-star WordCloud Two-star WordCloud

Three-star WordCloud Four-star WordCloud

Five-star WordCloud

The WordClouds above shows the most popular words appearing in each rating category. For example, there is a big “great” in the review data with a rating of 5.0, showing that the word “great” is one of the most popular words commented by the buyers who rate 5.0 on the selected products.

Part 2: Predictive Task

Predict rating ('overall' attribute) based on 'reviewText'

To ensure the validity of our model’s predictions, we split data into 3 sets, 80% train set, 10% validation set, and 10% test set. All models are trained with the training set, and tuned with the validation set. To avoid fine-tuned hyperparameters overfit the validation set, the final accuracy is calculated based on the test set. We only use verified reviews for predictions.

Baseline 1 - predicting mean

In this model, we always use the rating mean as the predicted value. By summing all ratings in the training set and then dividing by the length of the training set, and rounding(since all ratings are integers) we got mean = 4. Accuracy is calculated by dividing the amount of rating in the test set that matches the predicted value by the total comparison counter. The accuracy is 0.157. This model performs poorly, always predicting rating means is obviously not a good choice.

Baseline 2 - user mean

Since an overall rating mean can not fit everyone, we use user means to make predictions that fit every user's bias. To obtain the user mean data, we iterate through the training set, for each user, calculate his/her rating mean with rounding(to integer), and add this user to the user's mean dictionary. Then, iterate through the test set, use user mean as a prediction if the current user is in the training user's mean dictionary, otherwise use rating mean as the prediction. The accuracy is 0.559. The user means the model is significantly better than only predicting the mean, which means users bias plays an important role in predicting its rating.

Baseline 3 - bag-of-words model

Data Cleaning

Before we created the Bag of Words, we first cleaned the review data as we first cleaned the texts in the following steps as shown below. Then we created the Bag of Words using the word tokens generated from the review text data.

1. **Remove hyperlink in the review text**
2. **Tokenize the review texts into words**
3. **Remove stop words**
4. **Lemmatize words**
5. **Remove Punctuation**

Also, we observed that all ratings in the dataset are integer numbers from 1 to 5, which can be classified as 5 categories. Thus, we use the bag of words model in this section. To begin with, we break down each review text into words, count the occurrences of each word, and store it in a dictionary. Dump each word and corresponding words counts into a temporary value for sorting. Then, take the first 1000 high-frequency words as a words dictionary.

Each feature for X is obtained by iterating through the data array, breaking down sentences into words, and counting the occurrences of words that are in the words dictionary. Y feature is the rating of each review. Repeat the above process for Xtrain, Xvalid, and Xtest. After that, use LogisticRegression with C=1 to train on the training data. Use the model to predict the validation set and calculate the accuracy. Tune the hyperparameters to increase the accuracy of the validation set. The accuracy of the validation set is 0.643. Finally, run the prediction model on the test set, the accuracy is 0.642. The accuracy on the test set is lower than the validation set, this might be caused by overfitting, but the bag of words models has improved the accuracy compared to baseline models.

Baseline 4 - improved bag-of-words model

For rating prediction, we also use the ACC to measure the model since it is a regression problem. The rating prediction may be used to show ranking of the product, for item recommendation etc. Under these prediction scenarios, although products in category 5 might all mean 'Good', but ratings of that category such as rating 4.8, 4.9 (both can be seen as category 5 as 'Good') is different in terms of ranking.

Correspondingly, the baseline model is revised to use ACC for assessment metric. Then an improved version of the Bag of Words model is used to predict the ratings. Firstly, build the word counter dictionary. The data set is also parsed by removing punctuation and stop words, to improve the learning efficiency. Then, define the feature function along with a pipeline to select the best size of word set to use.

For selecting the size of words, we determined to choose a larger size than our previous bag-of-words models for reducing the ACC. After running for different sizes of words, it shows size of 3000 is the best size with minimum ACC to use. We use this model for the test data set. The result is better than the previous bag-of-words model baseline. We have also tried to use bigrams, but the result is not as good as unigram.

Part 3: Proposed Model

We found that if we applied sentiment analysis on the review text (Bag of words method), it would produce a higher accuracy. However, Bag of Words transformation would give common words which are not very representative to the review a higher weight, thus would carry ineffective information to its review representation. To improve this, we keep the idea of sentiment analysis but change the representation of review from the Bag of Words model to TF-IDF (term frequency–inverse document frequency). By doing so,

words that are more representative to the review itself would carry a higher weight, and theoretically would produce a higher accuracy on the classification task. We thus implement a prediction model with TF-IDF and Linear SVM classifer (Support vector machine). Before converting review texts into TF-IDF vetcors, we first clean the texts using the same way as we did in the Bag of Words baseline.

Now we are going to use the SVM model other than the Logistic Regression used in the Bag of Words approach. Our baseline has a C value of C = 1. We have the accuracy by stars as follow:

```
1 :   0.5177016831108532
2 :   0.12965722801788376
3 :   0.21565495207667731
4 :   0.19691487335745572
5 :   0.9325842696629213
overall accuracy:       0.6950166912273316
```

We could find that the accuracy for 5 stars is extremely large. We think that the reason is that the users are more willing to rate 5 instead of other stars when they feel satisfied with the video games. We could assume that most users would feel good about the video games since they would not buy the games they are not interested in. In this case, the prediction of 5 stars would be more accurate than the others. Due to this imbalance of the dataset, we further found that the f1 score for certain classes (eg, rating of 2.0) is extremely low compared to other classes. Taking into account, we adjusted the class weight of this classifier by setting class_weight = 'balanced' and tried other C values to find a better result.

Change C values:

We are going to use 0.001 to 1000 as our C values of the SVM (or called SVC in Sklearn) model. Our test shows that the larger the max_feature value in the TF-IDF transformer, the more accurate our prediction. In this case, we have max_feature as 150000 and ngram_range = (1,4). The table for our accuracies is shown in Figure 3.1 below.

C value	train	valid	test
0.001	0.69	0.71	0.68
0.01	0.72	0.71	0.69
0.1	0.81	0.71	0.69
1	0.91	0.69	0.67
10	0.95	0.66	0.64
100	0.96	0.64	0.63
1000	0.95	0.61	0.60

Figure 3.1 Accuracies by C value

We can have a more direct view with a line plot in Figure 3.2 below:

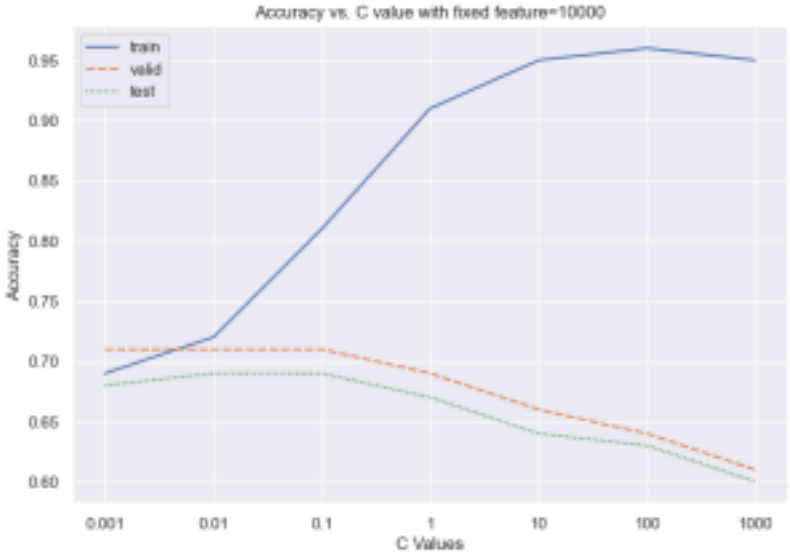


Figure 3.2 Line plot of C values vs Accuracies

As we can see in the above plot, the training accuracy will grow when the C value increases; however, the accuracies for validation set and test set are decreasing when C value increases since C = 1. Furthermore, due to the training data being not balanced as mentioned previously such that it contains an unproportional number of 5.0 ratings than other ratings, we need to consider other metrics to evaluate the performance of the model. We chose F-1 score as the extra metric as it focuses on both precision and recall. Based on the accuracies and macro (unweighted) average F-1 score on the test data shown in the Figure 3.3, we clearly observed that the highest accuracy on the test data is produced by C value of 0.1, and coincidentally this model also produced a highest macro average F-1 score of 0.46. Great! One thing worth noting is that the train accuracy for SVM performs extremely well (accuracy of 95%) , but the test accuracy drops to around 68% when predicting on validation data and test data. Based on this observation, we concluded that our model is overfitting the training data. The strength of the SVM model is it is effective in high dimensional spaces, but it trains data very slowly and needs plenty of time to train the model.

C value	train	valid	test	f1 score test avg
0.001	0.69	0.71	0.68	0.36
0.01	0.72	0.71	0.69	0.45
0.1	0.81	0.71	0.69	0.46
1	0.91	0.69	0.67	0.44
10	0.95	0.66	0.64	0.41
100	0.96	0.64	0.63	0.39
1000	0.95	0.61	0.60	0.37

Figure 3.3 C value, Test Accuracy, and F1 Score Avg

The results are summarized in the table below (See Figure 3.4). The model proposed by us outperformed other baseline models, and a significant improvement of accuracy compared to Baseline 1.

Model Accuracy on testing set
Baseline 1: predicting mean 0.157
Baseline 2: user mean 0.559
Baseline 3: bag-of-words model 0.642
Baseline 4: improved bag-of-words model 0.674
Proposed Model 0.690

Figure 3.4 Comparison of Accuracies

Part4: Related Literature

Jianmo Ni and Julian McAuley's study of expanding phrases to full reviews in 2018 [1] focuses on building an assistive system that helps users to write reviews. In terms of dataset, the study focuses on the Amazon review dataset released in 2014. In natural language processing, contextual (or "data-to-text") natural language generation is a very essential task that could be applied in a lot of different fields. This study applied this task within recommender systems to estimate personal thinking about a certain product. The model generates complete and coherent reviews with an encoder-decoder framework that

could expand short phrases from the input system. The encoder-decoder framework considers three encoders: a sequence encoder, an attribute encoder, and an aspect encoder into one decoder. The sequence encoder uses a gated recurrent unit (GRU) network to encode text information; the attribute encoder learns how to express the users' opinion onto items; and the aspect encoder finds an aspect-aware representation of users and items, which reflects user-aspect preferences and item-aspect relationships. The decoder attempts to generate full reviews that are consistent with the input phrases belonging to the most relevant aspects.

Jianmo Ni and Julian McAuley's further study in 2018 [2] builds a review generation model which could fulfill short clauses into complete and subjective reviews. This study also applies the Amazon review dataset released in 2014. Many review texts written by existing generators could not precisely reflect users' decision-making. As a result, the model that learns directly from reviews does not capture crucial information that explains users' purchases. In this case, this further study in 2019 seeks to introduce new datasets and methods to address this recommendation justification task so that it could generate convincing and diverse justifications. The study first proposes an 'extractive' approach to identify review segments that justify users' opinions. This approach is then used to distantly label massive review corpora and construct large-scale personalized recommendation justification datasets. The study designs two personalized generation models: (1) a reference-based Seq to Seq model with aspect planning which can generate justifications covering different aspects, and (2) an aspect-conditional masked language model which can generate diverse justifications based on templates extracted from justification histories.

Two similar datasets are Yelp 2014 and IMDB as it also contains reviews text and item ratings. One remarkable research done on this dataset was conducted by Chen and his team, achieving the state-of-art benchmark on this dataset. Chen et al. [4] built a sequence model to study the Yelp items reviews text. To achieve this, he used a 1-d CNN (Convolutional neural network) to train the generated word embeddings; then, group by those word embeddings into user and item representations. One thing worth noting in this process is that Chen put the temporal relations into the consideration, so the output user representations and item representations will be ordered by the time when the review was created. Afterwards, train the product embeddings and user embeddings with RNN-GRU (Recurrent Neural Network with Gated Recurrent Unit). Finally, concatenate the trained embeddings (reviews embeddings, user embeddings, and product embeddings), and apply a classical SVM classfier on the embeddings to do the ratings predictions work.

Shrestha and Nasoz [3] built a web application that detects the mismatched reviews text and their actual ratings on the Amazon product reviews, eg, the review looks very positive but the rating is extremely low. Inspired by Chen, Shrestha also applied temporal relations of the review embeddings in his model. Shrestha and his team utilized a RNN-GRU to learn the Amazon review data with paragraph vector and product

embeddings, and consider the temporal relations in the data learning before converting data into product embeddings, and then applied a SVM classifier to detect the sentiment of the review (Positive, Neutral, and Negative). Their RNN-SVM model achieved a remarkable accuracy of 81% on the Amazon product reviews dataset. Overall, these two papers on sentiment analysis both used deep learning and SVM classification on the Amazon review datasets, which are different from our approach as we are using tf-idf to described the reviews text and doesn't use neural networks and doesn't consider the temporal relations in the dataset. Our dataset is a 5-core dataset, so the dataset only contains 5 reviews for each user and each product; thus, the temporal relations between reviews on a same product is not meaningful as there are too few numbers of reviews data of a single product.

Part 5: Conclusions

In this project, we tried different baselines and models to predict the overall rating of video games from Amazon. In order to accurately predict the rating, we have to go through a discovery journey on the 5-core amazon reviews data on the video game category. By exploring the data, we compared to using predicting mean, user mean, bag-of-words model and proposed model (TF-IDF + SVM), we firstly observed that doing a sentiment analysis on the review texts would perform well on the ratings; then, we tried different sentiment analysis approachs and implement different models. We finally came up with a proposed model which has the highest accuracy and an acceptable average F-1 score on the test dataset. The fact that the training accuracy is higher than the test accuracy may show the prediction may not reflect real-world performance. Due to the scope and computational budget of this study, we are unable to utilize the whole amazon dataset(34GB) to train, validate, and test our model. Further study can be done to further improve the proposed model.

References

[1] Jianmo Ni and Julian McAuley. 2018. “Personalized Review Generation By Expanding Phrases and Attending on Aspect-Aware Representations.” In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pages 706–711, Melbourne, Australia. Association for Computational Linguistics.

[2] Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. “Justifying Recommendations using SDistantly-Labeled Reviews and Fine-Grained Aspects.” In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pages 188–197, Hong Kong, China. Association for Computational Linguistics.

[3] Shrestha, Nishit, and Fatma Nasoz. "Deep Learning Sentiment Analysis of Amazon. Com Reviews and Ratings." International Journal on Soft Computing, Artificial Intelligence and Applications, 8(1), pp.01-15. (2019).

[4] T. Chen, R. Xu, Y. He, Y. Xia and X. Wang, "Learning User and Product Distributed Representations Using a Sequence Model for Sentiment Analysis," in IEEE Computational Intelligence Magazine, vol. 11, no. 3, pp. 34-44, Aug. 2016, doi: 10.1109/MCI.2016.2572539.