CSE 158/258 - Predicting Amazon Ratings

Exploratory Data Analysis

We decided to explore the Video Games reviews dataset from the Amazon Review Dataset provided by Julian McAuley and Jianmo Ni. The full dataset contains approximately two and a half million reviews.

The data has several columns in a json format: image, overall, vote, verified, reviewTime, reviewerID, asin, style, reviewerName, reviewText, summary and unixReviewTime.

Our goal with this dataset is to predict the overall rating of the review based on the other data provided. We chose to exclude redundant columns such as style which we felt would be too excessive to be included due to the individuality of different video games, image because almost no video game reviews had any image posted by the reviewer due to them being a piece of software and unixReviewTime due to the over specificity of the column; we chose to use reviewTime instead, which is in the DD/MM/YYYY if we are to implement a temporal feature.

We then opted to use the 5-core dataset for video games provided by Jianmo Ni, which means that there are at least 5 reviews for each asin (amazon product ID) and each reviewer. We chose this not only to get rid of noise in the data, but also to enable us to better use techniques such as Jaccard similarity and cosine similarity in our model.

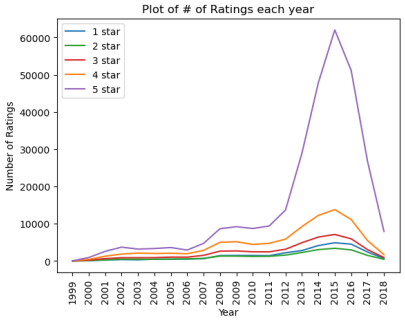
The Dataset we picked contains around 500,000 reviews which is large enough to run any model we discussed in the class.

We noticed that some potentially useful information was missing in our data such as the length of each review and also number of words in each review which is even more useful.

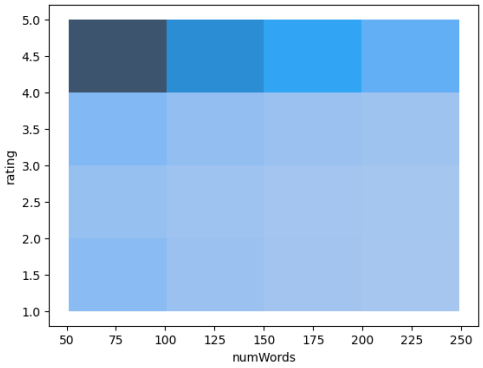
	rating	vote	reviewYear	reviewLength	numWords
count	497343.000000	497343.000000	497343.000000	497343.000000	497343.000000
mean	4.220329	2.309040	2012.851979	670.318963	122.701429
std	1.185478	17.137017	3.899879	1266.191356	227.005962
min	1.000000	0.000000	1999.000000	1.000000	1.000000
25%	4.000000	0.000000	2011.000000	57.000000	11.000000
50%	5.000000	0.000000	2014.000000	210.000000	40.000000
75%	5.000000	0.000000	2016.000000	710.000000	132.000000
max	5.000000	2474.000000	2018.000000	32721.000000	5928.000000

From the data above we noticed that the 25th percentile of numWords is 11 words. That's barely one sentence on average and a pretty barebones review to leave. On the other hand there were reviews that had over 500 words which is basically an essay. That's too much information so we decided to filter the words to be between the 25th percentile of numWords and 85th Percentile. This resulted in a nicer distribution of the numerical values we had.

	rating	vote	reviewYear	reviewLength	numWords
count	306127.000000	306127.000000	306127.000000	306127.000000	306127.000000
mean	4.121701	1.431354	2012.458127	396.839648	74.207126
std	1.252762	7.380440	3.914801	331.224020	60.611877
min	1.000000	0.000000	1999.000000	38.000000	11.000000
25%	4.000000	0.000000	2011.000000	136.000000	26.000000
50%	5.000000	0.000000	2014.000000	272.000000	51.000000
75%	5.000000	0.000000	2015.000000	572.000000	107.000000
max	5.000000	741.000000	2018.000000	2952.000000	249.000000



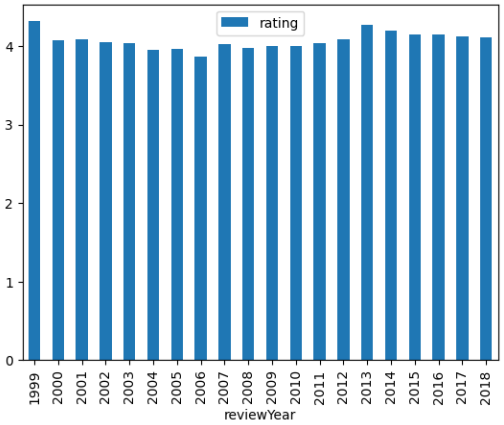
From the above plot, we can see that the number of reviews has increased over time, and also that 5 star ratings are the most popular ratings of all ratings.



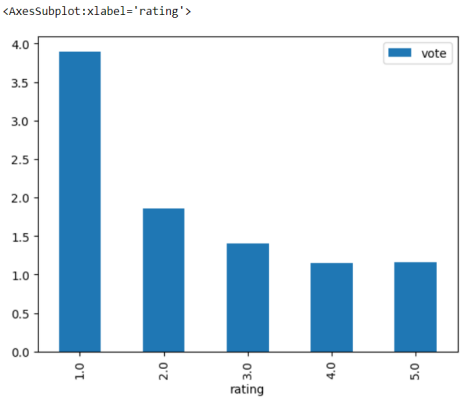
Above is a heatmap of rating based on number of words. It is clear that 5 star ratings is still the most common even as the number of words increases.

<u>Interesting findings</u> -

- there are more ratings with value five than ratings with the other four values combined.
- The review with the maximum length was 6000 words but 75th percentile of word lengths was length 350.
- Ratings with values 1 and 5 have lower words per rating than ratings with values 3,4 and 5.
- Ratings in the year 2018 had the lowest number of words per rating on average
- Ratings with value 1 received more votes than any other ratings.



Average rating of all reviews through the years. As shown, it does not fluctuate greatly and remains around a rating slightly above 4.



Average number of helpful votes for each rating. It is interesting to see that ratings with 1 have the most amount of votes on average.

Predictive Task

As we decided to choose to predict the rating of the reviews (Rating predictions), we looked to find the features that would result in the most drastic shifts in ratings. Our predictive task is to predict Amazon review ratings using review texts for (review, user) pairs. Accuracy will be measured in terms of the mean-squared error (MSE).

We primarily used pandas to process the data after importing the data using tools and functions provided in the class. Firstly, we used the python library pandas to clean the data which included filling nan values appropriately. For example filling the missing vote values with 0 and converting them to integers as 0 vote values were filled with nan, and the number values were typed as strings. We then used pandas to drop any irrelevant columns, shape data, and also to engineer relevant features.

```
smallDfRAT = mainDf.groupby(['rating']).mean()
smallDfRAT
```

rating	verified	vote	reviewYear	reviewLength	numWords
1.0	0.482230	3.897074	2012.521931	433.011677	80.744937
2.0	0.542719	1.851644	2012.118089	493.936249	91.889001
3.0	0.621118	1.399037	2012.124193	465.231770	86.956553
4.0	0.626187	1.145156	2011.918217	452.814153	84.609580
5.0	0.705344	1.161104	2012.730505	350.561237	65.659425

Above is a data frame we processed with pandas of the 5 ratings, and average values when grouping by the rating. We see that on average, the higher the rating the more likely the user is verified. Also interestingly, the unfiltered words are lowest for five star ratings and highest for 2 stars.

The features we used to do the predictive task are reviewID, UserID and the rating.

We dropped the rest of the features since we decided not to do sentiment analysis for our predictive task.

For the baseline model we chose to trivially use the average rating of all ratings the user had made formerly, and if said user did not exist within the training data, the global average would be used instead for the prediction. To achieve this we used an iterative lambda function and restricted the number of iterations to 10 iterations.

We achieved a MSE of 1.546 following this procedure while using a lambda value of 1. We tried various lambda values in order to optimize the baseline. We are able to change the values of the lambda because it is the constant result of an integration. After testing multiple lambda values, we found that the lambda value of 3 was the best for the baseline model.

Using a lambda value of 3, we were able to achieve an MSE value of 1.487. Another baseline that we tried was to simply use the average of the ratings from a particular year. However, this model had a very high MSE and was relatively super easy to beat, so it was not seriously considered.

Model

We attempted many different models in our quest to achieve the best MSE possible. Amongst these we tried a Singular Value Decomposition model, Simple Bias Only Latent Factor Models, Complete Latent Factor Model, and Complete Latent Factor Model Using Tensorflow.

$$f(u,i) = \alpha + \beta_u + \beta_i$$

The simplest model we could use as described before is simply using the mean to predict the next rating. The next most simple would be adding B_u and B_i. B_u is the magnitude of which the user tends to rate items above their respective global means and B_i is how much the item tends to be rated above the mean rating of all items. This is the Simple Latent Factor Model.

$$f(u,i) = \alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i$$

The Complete Latent Factor Model adds as well the user preference, y_u, and item preference, y_i, and takes the dot product into account to do dimensionality reduction. As a result theoretically the Complete Latent Factor Model should be the best model we do however on a previous dataset it was found depending on how you picked values, the Simple Latent Factor Model could be better.

Before trying to optimize any model we compared the default Simple Latent Factor Model and Complete Latent Factor Model. This meant that in our optimization, we left the derivative to converge all the variables as the default and didn't put the best derivative given in class for the respective Simple Latent Factor Model and Complete Latent Factor Model.

This resulted in an MSE of 1.546 for Simple Latent Factor Model, which was the same as our unoptimized Baseline model. The Complete Latent Factor Model had an MSE of 1.482 as well which was better than the optimized Baseline Model with an MSE of 1.487.

$$\alpha = \frac{\sum_{u,i \in \text{train}} (R_{u,i} - (\beta_u + \beta_i))}{N_{\text{train}}}$$

$$\beta_u = \frac{\sum_{i \in I_u} R_{u,i} - (\alpha + \beta_i)}{\lambda + |I_u|}$$

$$\beta_i = \frac{\sum_{u \in U_i} R_{u,i} - (\alpha + \beta_u)}{\lambda + |U_i|}$$

$$\arg\min_{\alpha,\beta,\gamma} \underbrace{\sum_{u,i} (\alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i - R_{u,i})^2}_{\text{error}} + \underbrace{\lambda \left[\sum_u \beta_u^2 + \sum_i \beta_i^2 + \sum_i \|\gamma_i\|_2^2 + \sum_u \|\gamma_u\|_2^2 \right]}_{\text{regularizer}}$$

As a result we tried to optimize the Complete Latent Factor Model. This meant incorporating the derivatives covered in class which are above and improving the parameters wherever possible such as decreasing the step size and increasing the number of iterations to converge. After messing around with that and testing on the test set and validation set (Random 20 percent of the data). We ended up with an MSE of 0.997 and 1.07 respectively. Which were both very good values for an MSE

For both of the previous models there was no problem with scalability and no problems with missing values since we took care of all such things in EDA. One concern we did

have was overfitting our model. We trained and tested our model using a standard 80/20 split. However, in order to increase the accuracy of the test data, we changed multiple parameters in our model. Because these parameters were specifically adjusted for the test dataset, we put ourselves at a risk of overfitting. To overcome this, we used cross validation. We randomly resampled the original dataset from before the train/test split in order to simulate a new dataset.

After this we created an SVD model using the surprise module from Scikit-learn. Using lecture notes there wasn't much optimization we can do for this model so we left it as is. We kept to the usual data split of 80% for train and 20% for test. This resulted in an MSE of 1.19 which was the second best model so far and we know that there was no overfitting possible for the SVD model since we aren't doing any parameter optimization. Similarly there were no scalability issues.

The final model we created was a Complete Latent Factor Model using a Tensorflow model. The original Tensorflow model we created had an MSE of 1.948 which was far below even the baseline which can be considered an unsuccessful model. We tried to improve it by decreasing the step size and increasing the number of iterations but due to the time complexity of tensorflow models, it was taking nearly an hour to run only to improve to an MSE of 1.546 which we could consider a failure as well.

Models we also considered include trying to add sentiment analysis to our model. By doing this we can do Bayesian Personalized Ranking Model using Tensorflow and Instance Reweighting. This was very similar to the literature we have read and we tried a different approach to get a lower MSE.

Apart from that another model we considered was a temporal model which takes into account the time at which reviews happen and neural networks, however due to our inexperience with both of them, we decided not to do them since they were also out of scope of this class.

The strengths of both types of Latent Factor Models is that they reduce dimensionality while Complete Latent Factor also takes into account item preference and user preference as well making it a more complete model compared to simple latent factor model. However at the same time the information provided by the preferences.

The Complete Latent Factor Model that is from the Tensorflow library already uses Principal Component Analysis (PCA) within the library functions and models, and this dimensionality reduction, although would be useful to counter overfitting, causes loss of data that could otherwise be utilized by the model.

Literature

We encountered several studies and pieces of literature that studied the same dataset provided by Justin McAuley, and within those we found interesting findings of the dataset in general, and also several intriguing applications of machine learning techniques.

The first study and classifier we looked at made use of a ROC accuracy model and used NLTK library: stemming, n-grams, stop words, tokenizing TF-IDF for each of the stemmed and tokenized words and ngrams. The study experimented with several models including Gaussian Naive Bayes, Decision Tree, Random Forest, Stochastic Gradient Descent and Logistic Regression.

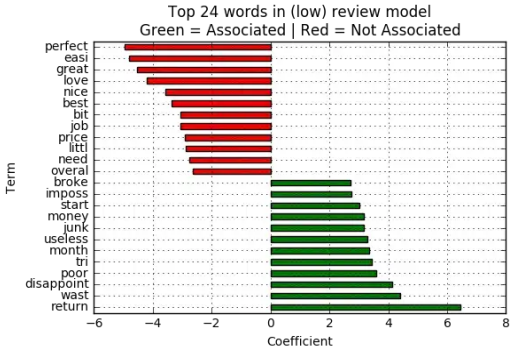
Interestingly, logistic regression was found to be the most accurate and most efficient in terms of prediction time, which goes to show that simple models that are less fit can sometimes predict better overall for unseen data. We found this to be the case for our models; using complex models such as deep learning did not increase our accuracy as much as simpler models. Grid search and cross validation was used to refine the model further.

We then looked at a piece of literature that took a scalability approach instead. This approach looked at over 80 million reviews and made use of Apache Spark, a tool extremely useful and efficient when working with large datasets.

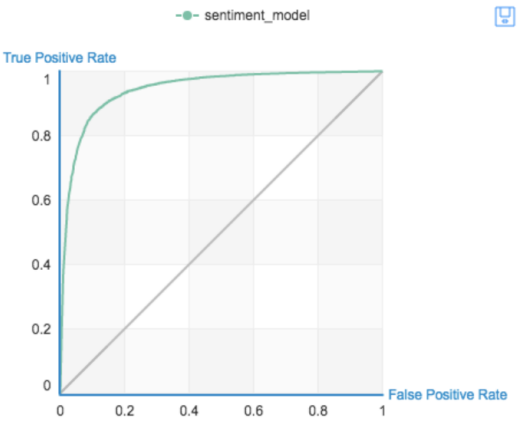
If we were to utilize the entire amazon dataset, using a tool like Apache Spark would be really helpful. It would also enable us to look into more advanced features such as using temporal features. Querying the data of a large dataset would be extremely time consuming if traditional methods were used.

The study found that distribution of nth reviews by a specific user are different. A user's first review is more likely to be a one star review, versus the subsequent reviews they make. This makes sense as most people are only going to go through the trouble of making a review if they have negative sentiment.

The next study we found focused on text analysis. There were several models, one of which functioned by clustering reviews by product. It identified specific words that are associated with certain ratings. For example ‘perfect’ was the most popular word for high ratings, ‘however’ was the most common for neutral ratings, and ‘return’ was the most common for low ratings. The chart below shows this relation



Finally, this article described a rather straightforward approach to classifying ratings. Instead of trying to predict the number of stars or numerical value of a rating, this model simply classified models as either positive or negative.



True Positive	False Negative	Accuracy	Precision
26521	**1455**	**0.916**	**0.952**
False Positive	True Negative	Recall	F1 Score
1327	**4001**	**0.948**	**0.95**

They used a logistic classifier and were able to create a very accurate model that was validating using actual numerical ratings from Amazon.

Results

Some machine learning techniques used from libraries such as tensorflow did not seem to improve accuracy, but having more time to experiment with the models and tune the parameters could have had potential in improving the accuracy of the model.

The lowest MSE we managed to achieve was 1.07 using our Complete Latent Factor Model. Some of the other models we used achieved the MSE of 1.19 (SVD model), 1.35 (Complete Latent Factor Model without optimization), 1.56 (TensorFlow Model).

The lower the MSE, the higher the accuracy of our model is, however our MSE was too good for one of our models which means we risked overfitting on the data, but since we got similar values on the Test and Validation set, that means we did not overfit. Our next best model was SVD (Singular Value Decomposition) model, which received a pretty low MSE and there is no chance of overfitting in that model since there are parameters to modify.

Some improvements that we could make would be using a larger dataset to gain more statistical power. The amazon dataset could potentially have millions of reviews, and utilizing this abundance would allow our model to be extremely well-fitted. This could however run into problems such as slow runtimes. However, utilizing tools such as the one mentioned above in one of the literatures, like apache spark, could enable us to use the entire dataset as it would speed up querying much more for bigger datasets.

Also using better refinement techniques such as Gridsearch and perhaps larger validation sets could enable us to optimize our choices for hyperparameters. We didn't get to fully utilize other libraries such as tensorflow and surprise, and the models we tried did not seem to yield great results. However, with more research, time and experimentation, we could've better engineered our features to suit the models, and potentially get better results accuracy wise.

We could also have added sentiment analysis to do other models such as Bayesian Personalized Ranking and Instance Reweighting but ended up not doing so.

Bibliography

https://t-lanigan.github.io/amazon-review-classifier/

https://minimaxir.com/2017/01/amazon-spark/

https://medium.com/jbencina/part-1-predicting-amazon-review-ratings-with-text-analytics-in-python-fa7c14e91464

https://towardsdatascience.com/predicting-sentiment-of-amazon-product-reviews-6370f466fa73

Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering
R. He, J. McAuley
WWW, 2016

Image-based recommendations on styles and substitutes
J. McAuley, C. Targett, J. Shi, A. van den Hengel
SIGIR, 2015

Justifying recommendations using distantly-labeled reviews and fined-grained aspects
Jianmo Ni, Jiacheng Li, Julian McAuley
Empirical Methods in Natural Language Processing (EMNLP), 2019