Video Game Rating Classification using Text Analysis

Pranay Tulugu Anthony Limon

0. **Abstract**

This assignment analyzes video game product reviews from Amazon. Through univariate and bivariate analyses of the data set, further exploratory data analysis and data visualization, we attempt to implement a model that classifies video game ratings from textual features. By building a LogisticRegression model pipeline and employing feature generation techniques from textual data, our goal is to maximize accuracy in predicting video game ratings from reviews. We compare this model to SOTA models such as XLNet for text classification.

1. **Dataset Analysis**

We are using Professor McAuley's updated (2018) Amazon product review dataset (He and McAuley). The entire dataset consists of 233.1 million reviews, composed of reviews from May 1996 - October 2018. For the purpose of this assignment, we will only be using a subset of this data, Video Game reviews. The Video Game subset consists of 497,577 reviews. Initially, when looking for datasets, we performed a minor exploratory data analysis on the datasets to examine which potential target features were the most balanced. For example, one of the review features we had in mind was the overall rating. We examined the counts of each rating, and compared the distributions across other datasets in order to choose one relatively balanced. After identifying several options for datasets we began thinking about which predictive tasks might be better suited for the features in the data and began generating baseline models for each dataset that we felt could best be improved upon. Below is the exploratory data analysis for the video game review dataset.

Table 1: Dataset Features

	feature	description
1	overall	rating of the product
2	verified	boolean
3	reviewTime	time of the review (raw)
4	reviewerID	ID of reviewer
5	asin	ID of product
6	reviewerName	name of reviewer
7	reviewText	text of review
8	summary	summary of review
9	unixReviewTime	time of review (unix time)
10	vote	# of helpful votes of review
11	style	dictionary of product metadata
12	image	image that users post after receiving product

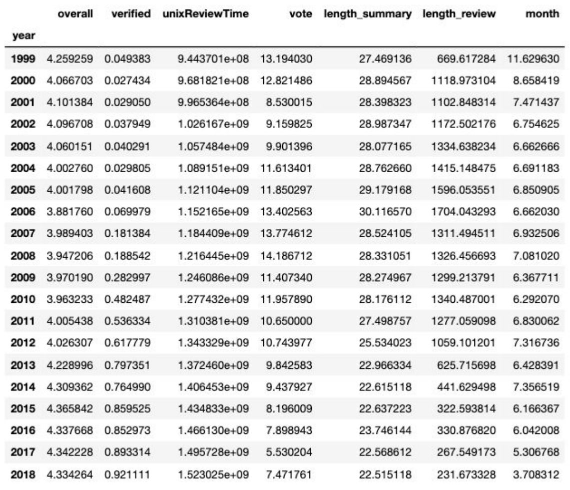
Table 2 below shows basic stats of all numerical variables.

Table 2: Descriptive statistics

	overall	unixReviewTime	vote	length_summary	length_review
count	497577.000000	4.975770e+05	107793.000000	497468.000000	497419.000000
mean	4.220456	1.367848e+09	10.655117	24.494878	670.288642
std	1.185424	1.224113e+08	35.582078	18.895582	1266.122219
min	1.000000	9.398592e+08	2.000000	1.000000	1.000000
25%	4.000000	1.316563e+09	2.000000	10.000000	57.000000
50%	5.000000	1.410221e+09	4.000000	18.000000	210.000000
75%	5.000000	1.452384e+09	8.000000	34.000000	710.000000
max	5.000000	1.538438e+09	2474.000000	282.000000	32721.000000

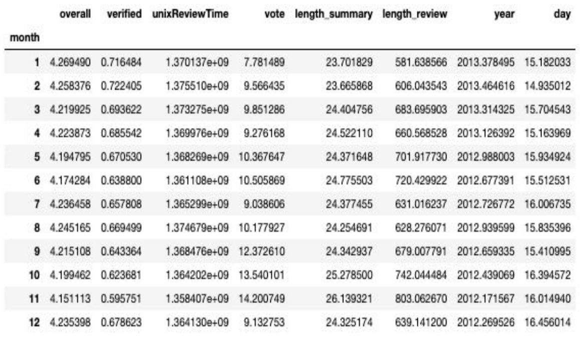
We also wanted to try to utilize temporal features, and see if particular time periods in which a review was written depict a pattern or relationship with other features. Table 3 and Table 4 below show the averages across numerical variables, grouped by year and grouped by month, respectively. Some of the observations from the tables were that certain averages varied per year, for example, from 2000-2012 the average length of reviews were 2-7 times larger than the length of reviews from 1999 and 2013-2018, depending on which two years were compared.

Table 3: Averages grouped by Year



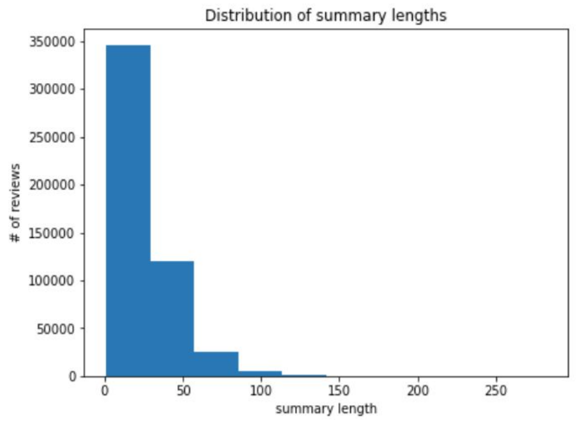
year	overall	verified	unixReviewTime	vote	length_summary	length_review	month
1999	4.259259	0.049383	9.443701e+08	13.194030	27.469136	669.617284	11.629630
2000	4.066703	0.027434	9.681821e+08	12.821486	28.894567	1118.973104	8.658419
2001	4.101384	0.029050	9.965364e+08	8.530015	28.398323	1102.848314	7.471437
2002	4.096708	0.037949	1.026167e+09	9.159825	28.987347	1172.502176	6.754625
2003	4.060151	0.040291	1.057484e+09	9.901396	28.077165	1334.638234	6.662666
2004	4.002760	0.029805	1.089151e+09	11.613401	28.762660	1415.148475	6.691183
2005	4.001798	0.041608	1.121104e+09	11.850297	29.179168	1596.053551	6.850905
2006	3.881760	0.069979	1.152165e+09	13.402563	30.116570	1704.043293	6.662030
2007	3.989403	0.181384	1.184409e+09	13.774612	28.524105	1311.494511	6.932506
2008	3.947206	0.188542	1.216445e+09	14.186712	28.331051	1326.456693	7.081020
2009	3.970190	0.282997	1.246086e+09	11.407340	28.274967	1299.213791	6.367711
2010	3.963233	0.482487	1.277432e+09	11.957890	28.176112	1340.487001	6.292070
2011	4.005438	0.536334	1.310381e+09	10.650000	27.498757	1277.059098	6.830062
2012	4.026307	0.617779	1.343329e+09	10.743977	25.534023	1059.101201	7.316736
2013	4.228996	0.797351	1.372460e+09	9.842583	22.966334	625.715698	6.428391
2014	4.309362	0.764990	1.406453e+09	9.437927	22.615118	441.629498	7.356519
2015	4.365842	0.859525	1.434833e+09	8.196009	22.637223	322.593814	6.166367
2016	4.337668	0.852973	1.466130e+09	7.898943	23.746144	330.876820	6.042008
2017	4.342228	0.893314	1.495728e+09	5.530204	22.568612	267.549173	5.306768
2018	4.334264	0.921111	1.523025e+09	7.471761	22.515118	231.673328	3.708312

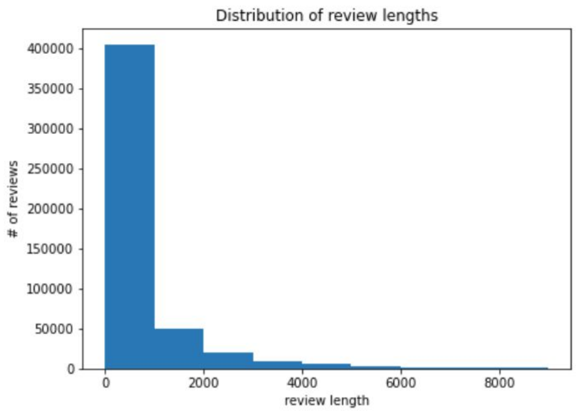
Table 4: Averages grouped by month



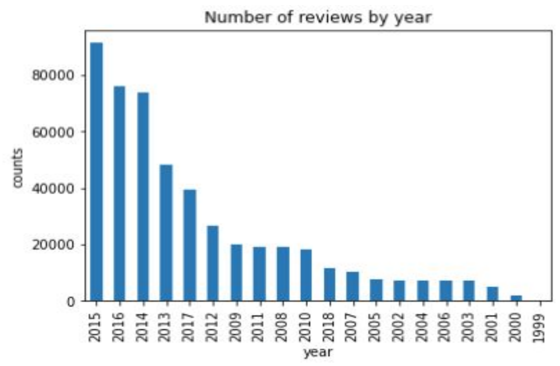
month	overall	verified	unixReviewTime	vote	length_summary	length_review	year	day
1	4.269490	0.716484	1.370137e+09	7.781489	23.701829	581.638566	2013.378495	15.182033
2	4.258376	0.722405	1.375510e+09	9.566435	23.665868	606.043543	2013.464616	14.935012
3	4.219925	0.693622	1.373275e+09	9.851286	24.404756	683.695903	2013.314325	15.704543
4	4.223873	0.685542	1.369976e+09	9.276168	24.522110	660.568528	2013.126392	15.163969
5	4.194795	0.670530	1.368269e+09	10.367647	24.371648	701.917730	2012.988003	15.934924
6	4.174284	0.638800	1.361108e+09	10.505869	24.775503	720.429922	2012.677391	15.512531
7	4.236458	0.657808	1.365299e+09	9.038606	24.377455	631.016237	2012.726772	16.006735
8	4.245165	0.669499	1.374679e+09	10.177927	24.254691	628.276071	2012.939599	15.835396
9	4.215108	0.643364	1.368476e+09	12.372610	24.342937	679.007791	2012.659335	15.410995
10	4.199462	0.623681	1.364202e+09	13.540101	25.278500	742.044484	2012.439069	16.394572
11	4.151113	0.595751	1.358407e+09	14.200749	26.139321	803.062670	2012.171567	16.014940
12	4.235398	0.678623	1.364130e+09	9.132753	24.325174	639.141200	2012.269526	16.456014

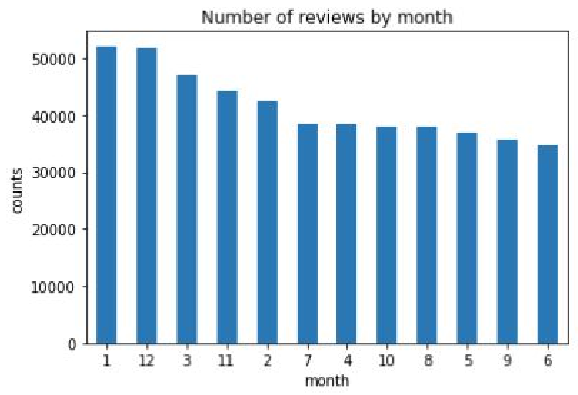
Associated plots show the distribution of summary and review lengths, findings were that most summaries were less than 100 characters, and most reviews were less than 1000 characters.

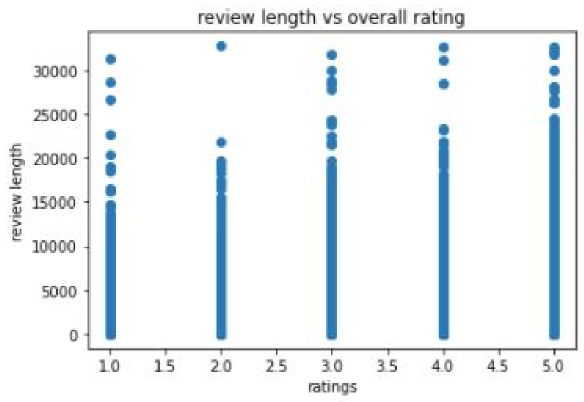




Interestingly, there were also more reviews from 2013-2017 than from 2000-2012 (328,468 vs 157,214). This can be seen in the Number of Reviews per Year barplot.

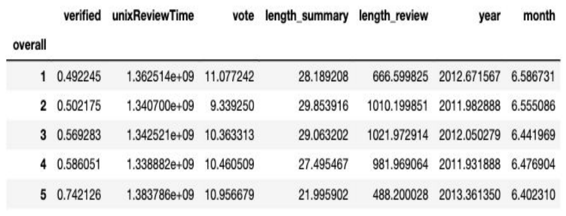




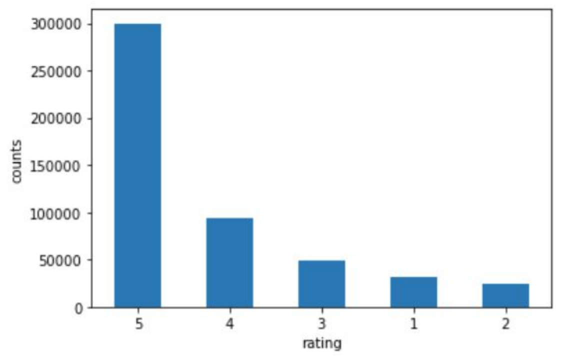


In Table 5 we have average data grouped by ratings, and an associated barplot showing the number of each rating in the reviews. From the table you can see that the average review length was higher for reviews that were rated 2 or 3 compared to all other ratings. From the plot you can see that there are more '5' ratings than all other ratings combined.

Table 5: Averages grouped by ratings



	verified	unixReviewTime	vote	length_summary	length_review	year	month
overall							
1	0.492245	1.362514e+09	11.077242	28.189208	666.599825	2012.671567	6.586731
2	0.502175	1.340700e+09	9.339250	29.853916	1010.199851	2011.982888	6.555086
3	0.569283	1.342521e+09	10.363313	29.063202	1021.972914	2012.050279	6.441969
4	0.586051	1.338882e+09	10.460509	27.495467	981.969064	2011.931888	6.476904
5	0.742126	1.383786e+09	10.956679	21.995902	488.200028	2013.361350	6.402310



Finally, we plotted the relationship between ratings and review lengths in a scatter plot. Unfortunately, the correlation is unclear.

2. **Predictive Task**

The task is to predict the rating ('overall') of a given review. We can look at this in two ways, as a regressive task and a classification task. Let us consider the regressive task. We consider a baseline MSE of always predicting the mean.

$$MSE = 1.4137$$

We can compare this to a similarity based model such as Jaccard similarity based prediction. Given a user, item pair, we look at all the items the user has used, and find the item - item similarity to predict the mean rating

$$MSE = 1.3181$$

This is better, and can be further improved upon. However, consider that these ratings are integers, not real valued, so it seems better to treat this as a classification task. Since this dataset is highly skewed let us consider the following baseline: always predict 5. We measure validity using the accuracy metric.

$$Accuracy = 0.6044$$

The first method to consider is simply using the real valued Jaccard similarity model above and rounding our values to make a prediction.

Accuracy = 0.5472

Our accuracy is worse than the baseline, so lets evaluate our dataset to consider how:
(1) data is organized such that classification is possible using simple features (linearly separable)
(2) imbalances and outliers are present.
We see, given in section 1, that our dataset is highly imbalanced and skewed towards '5'. It is also not linearly separable using simple feature representation.
Let us consider a simple feature representation. These were the simple parameters present in dataset, not including reviewText:

[Review Length, Verified, Vote]

We can try to use simple logistic regression:

Accuracy = 0.4814

Let us try Support Vector Machines and Random Forests to make classifications. Let us look at the accuracies of these using our feature vector

RandomForestClassifier Acc. = 0.595
RBF Kernel SVM Acc. = 0.6

We see that even for rbf kernels in SVM, our model doesn't tend to go above the baseline threshold.
This is because we see from section 1 that our data is not separable and balanced using the simple features in the dataset, so we have to do something using text analysis for this classification.

We can start by implementing a simple bag of words model, with a dictionary size of 500. The only preprocessing we do is convert all words to lowercase and remove punctuation. Each word in the feature vector is represented as a frequency encoding and we pass this through a logistic regression.

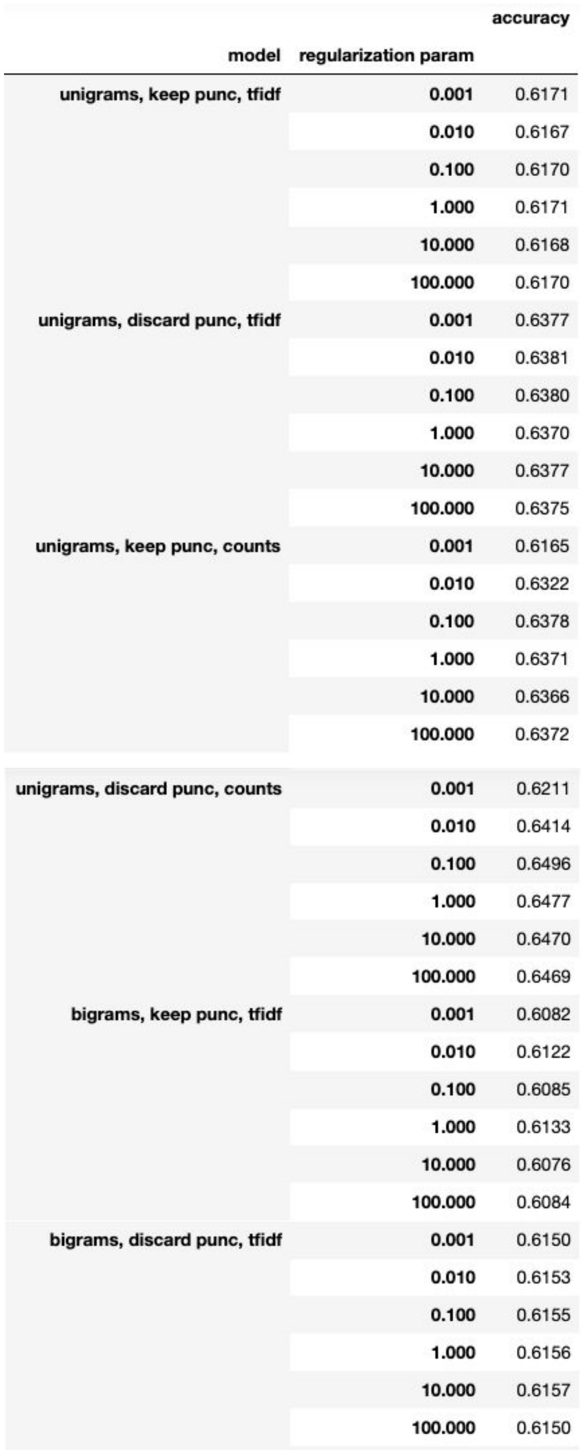
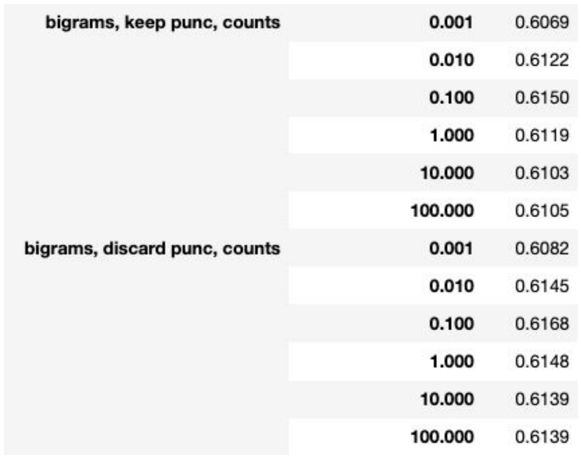
Accuracy = 0.61

We see a slight improvement, so we further work on this model

3. **Model Description**

After we identified our predictive task and established a baseline in the section above giving us an accuracy of ~0.6, we were able to further work on a model. It made sense to consider the review features of the dataset to extract meaning from the text reviews. Some of the considerations as far as text feature generation preprocessing for improving off the baseline model were stemming, tokenizing, (ngrams) and removing stopword and/or punctuation. We began reading in a subset (for runtime sake) of the reviews without punctuation or capitalization. Then we built a model utilizing bag of words feature vectors for the top 1000 most common unigrams and another model utilizing the top 1000 most common bigrams. We tried stemming vs non stemming and removing stopwords vs keeping stop words, however not stemming nor removing stop words helped our model's performance. Additionally, we then adapted the unigram and bigram models to use tf-idf scores. Eventually we realized that instead of tweaking each model it'd be a better idea to implement a pipeline that way we can also train models with varying regularization parameters. View the pipeline details below.

Table: Pipeline

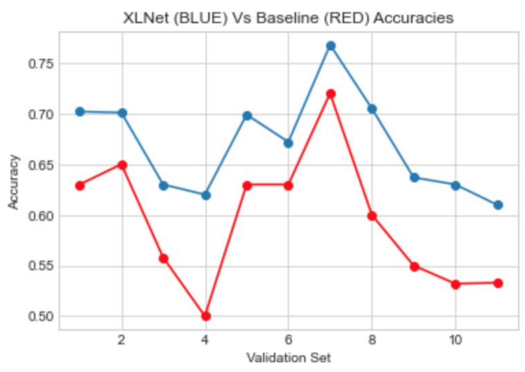



model	regularization param	accuracy
unigrams, keep punc, tfidf	0.001	0.6171
	0.010	0.6167
	0.100	0.6170
	1.000	0.6171
	10.000	0.6168
	100.000	0.6170
unigrams, discard punc, tfidf	0.001	0.6377
	0.010	0.6381
	0.100	0.6380
	1.000	0.6370
	10.000	0.6377
	100.000	0.6375
unigrams, keep punc, counts	0.001	0.6165
	0.010	0.6322
	0.100	0.6378
	1.000	0.6371
	10.000	0.6366
	100.000	0.6372
unigrams, discard punc, counts	0.001	0.6211
	0.010	0.6414
	0.100	0.6496
	1.000	0.6477
	10.000	0.6470
	100.000	0.6469
bigrams, keep punc, tfidf	0.001	0.6082
	0.010	0.6122
	0.100	0.6085
	1.000	0.6133
	10.000	0.6076
	100.000	0.6084
bigrams, discard punc, tfidf	0.001	0.6150
	0.010	0.6153
	0.100	0.6155
	1.000	0.6156
	10.000	0.6157
	100.000	0.6150
bigrams, keep punc, counts	0.001	0.6069
	0.010	0.6122
	0.100	0.6150
	1.000	0.6119
	10.000	0.6103
	100.000	0.6105
bigrams, discard punc, counts	0.001	0.6082
	0.010	0.6145
	0.100	0.6168
	1.000	0.6148
	10.000	0.6139
	100.000	0.6139

The optimal model utilized unigrams, used a bag of words feature vector rather than tf-idf scores, discarded punctuation and capitalization, and had a regularization parameter of 0.01. After running the pipeline, we took the optimal model and ran another model this time with an increase in dictionary size to the top 5000 most common unigrams. The model’s performance increased to an accuracy of 0.6603. We also ran the model with differing dictionary sizes, including the top 8000 most common unigrams(accuracy of 0.6624) and the top 10000 most common unigrams (accuracy of 0.6621), but the model’s performance and runtime began to suffer a bit with the increase to 10000. Due to lack of computational power we decided to cap the dictionary size at 8000. Another feature that stood out was the summary review text. We thought that since this variable summarizes the product review text, we could generate a more useful bag of words feature vectors for the top 50 unigrams in the text, without worrying about useless words that don’t provide as much meaningful information about the overall rating. From the EDA, ‘2’, ‘3’, and ‘4’ overall ratings had longer average review text lengths than ‘1’ or ‘5’ ratings. We used this intuition to incorporate this observation as a feature in our final model. As a test run we ran this model (with a dictionary size of 5000) on our subset train/validation split and achieved an accuracy of .6768. We were satisfied by the results and decided it was time to train our model on more data. The final model consists of a feature vector of length 8052, including the offset (bag of words feature vector of top 8000 most common unigrams in review text, bag of words feature vector of top 50 most common unigrams in summary text, review length, and offset). The final model's accuracy was 0.68589.

Identifying our problem lies in the scope of NLP, we can try to implement another model and compare it to our bag of words to see how well it's doing. One of the biggest drawbacks of bag of words is that it doesn't take into account contexts of sentences. To fix this let us look at the NLP model XLNet[1]. XLNet is an extension of the Transformer-XL model. It uses an autoregressive method to learn bidirectional contexts of sentences by maximizing the expected likelihood over all permutations of an input sequence. The reason we choose this instead of well the known Google BERT is because XLNet is known to outperform BERT because of its autoregressive formulation. (Yang 1)

Because implementing this model is complex, we found Simple Transformers, a library built off of Transformers by Hugging Face. This library allowed for a simple 'Initiate, Train, Evaluate' formula that allowed us to use this model without needing to know the complexities involved. The biggest issue we faced was how computationally intensive training this model was. Since we were unable to train on large datasets, we formed a representational dataset of 5000 entries, keeping note of rating frequencies in our new dataset, and trained on this data. To evaluate, we split our unseen data into multiple random iterations of 1000 data points and calculated model accuracies. These were the accuracies we got using default parameters (we were unable to tune because of computational intensity) compared to baseline accuracies:



Average Accuracy of XLNet = 0.673

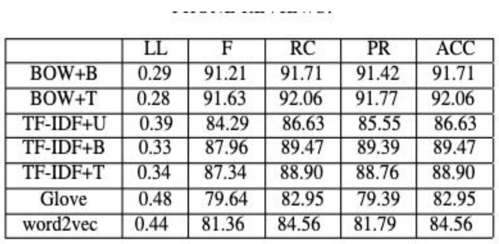
4. **Literature**

We are using Prof. McAuley Amazon dataset of Video Games (5 core). According to the professor's webpage, the larger amazon dataset seemed to have been used to model visual evolution of fashion trends, as well as image based recognition.

Consider this article (Aljuhani and Alghamdi) which uses Amazon mobile reviews for classification, similar to our task. In this case the data seems to be preprocessed by equalizing the number of positive and negative reviews, unlike our dataset which was highly skewed. They removed stop words but we did not as it did

[1] https://huggingface.co/transformers/model_doc/xlnet.html

not contribute to a better accuracy. They checked for spelling mistakes which we did not change. The labels were also changed to indicate sentiment as positive, negative or neutral, while we continued classification on star ratings, 1 to 5. After looking at the accuracies, they presented the following table



	LL	F	RC	PR	ACC
BOW+B	0.29	91.21	91.71	91.42	91.71
BOW+T	0.28	91.63	92.06	91.77	92.06
TF-IDF+U	0.39	84.29	86.63	85.55	86.63
TF-IDF+B	0.33	87.96	89.47	89.39	89.47
TF-IDF+T	0.34	87.34	88.90	88.76	88.90
Glove	0.48	79.64	82.95	79.39	82.95
word2vec	0.44	81.36	84.56	81.79	84.56

[2]

They achieved the best accuracy for BoW + trigrams as it seems that trigrams were able to capture word context to some extent. This seems intuitive as it should consider word contexts better than unigrams (Wallach 8). For our data set however, incorporating Bigrams or Trigrams further worsened the accuracy.
Looking at incorporation of tf-idf, we also saw a decrease in accuracy.

BoW is one of the simplest models for classification, though effective, and it gave us good results compared to baseline. The biggest drawback is lack of context understanding. For instance ‘This is good’ and ‘Is this good’ have the same vector notations. (Kowsari 6)

Another powerful approach being used is Word Embedding. Research was done to solve the issue of synonyms like ‘plane’ and ‘airplane’ not to be orthogonal vectors. word2Vec is a good example of this, using neural networks to create a high dimensional vector for each word. (Kowsari, 7)

$$arg \max_{\theta} \prod_{w \in T} \left[\prod_{c \in c(w)} p(c \mid w; \theta) \right]$$

[3]

It tries to maximize probability above, where p(c|w) is probability of some context given some word, and theta is a parameter. There have been various other word embedding models such as GloVe and FastText.
One drawback of these models where they would consider sentence context but rarely document context. We see in the Amazon mobile phone paper above (Aljuhani and Alghamdi), word embedding techniques did not work so well, so we pursued more advanced models.

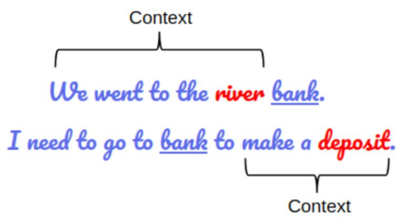
However, in the paper above, Convolutional Neural Networks with word2vec achieved an accuracy of 0.9. The biggest disadvantage however was that these models required a large amount of data to train, otherwise it would not likely outperform other models. Also they were very computationally expensive. (Kowsari 8)
We can see this in the XLNet accuracy graph in the section below, where we get consistent results with the BoW model but our XLNet accuracies are very choppy because of inability to train on large datasets. Also due to model complexity they are mostly black box models.

Google’s BERT debatably introduced NLP to a new phase. It bidirectionally trained on contexts in all layers. The result of this was

[2] Image: https://www.researchgate.net/publication/334185442_A_Comparison_of_Sentiment_Analysis_Methods_on_Amazon_Reviews_of_Mobile_Phones

[3] Image: https://arxiv.org/abs/1904.08067

that the model could be fined tuned with one additional output layer. (Devlin, 4)


Context
We went to the river bank.
I need to go to bank to make a deposit.
Context

4

The bidirectionality allows for context understanding such as the word ‘bank’ above.
In terms of text classification, we can use the model’s tokenization, and add a simple function on top as below

$$p(c|\mathbf{h}) = \mathrm{softmax}(W\mathbf{h}),$$

5

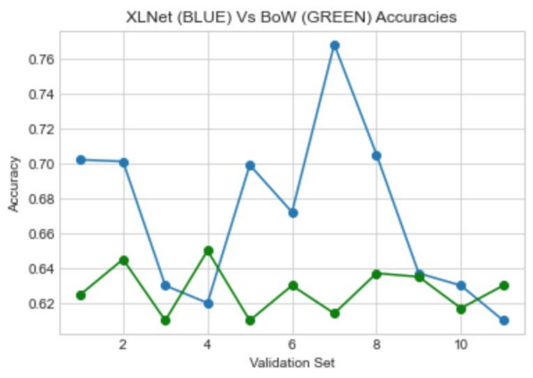
where c is the class, h is the final hidden state of token [CLS] which is a special classification embedding BERT does and W is a task specific parameter matrix. We can then take a maximum likehood estimation during fine tuning. (Sun and Qiu)
The model we used to compare to BoW model was XLNet, a SOTA model coming from ‘hugging face’ who provide frameworks for several other popular models that arrived after BERT such as roBERTa, XLM, etc[6].
For future studies, it may be interesting to study this data on systems like BERT and other SOTA models with fine tuned parameters on the entire data set. Also exploring deep learning models like CNN with word2vec like the paper referenced above could be interesting.

5. **Results and Conclusions**

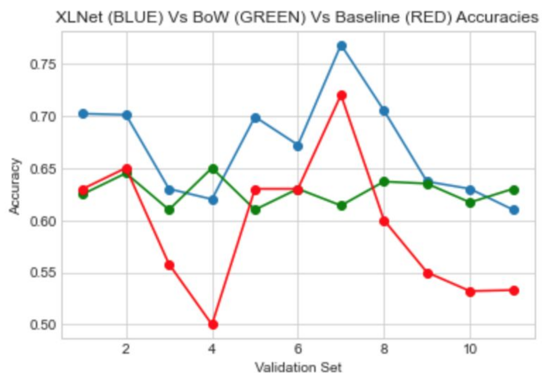
As hinted to in our model’s description, we trained over 48 Logistic Regression models. Our initial logit models were test runs to observe which features were most useful, whether larger or smaller dictionary size for a bag of words feature generation helped or hurt our model’s performance, whether stemming or not stemming, and removing stop words helped model performance, whether tf-idf scores were a better predictor of overall ratings, and whether stronger regularization also aided in our model’s performance vs weaker regularization. After implementing a pipeline, and using additional feature engineering to optimize our model’s performance our final model’s accuracy was 0.68589. One thing that stood out to us was how the bag of words models compared to the tf-idf models. One of the main differences between the two is that bag of words just creates a vector containing the occurrences/count of words in text, or in this case reviews, while tf-idf reflects the importance of words in text. It seems that a tf-idf model that can retain information about grammar and context would certainly outperform a bag of words model that cannot provide information about the semantics of a document. Especially when this classification task does depend on the semantics of text; negative reviews would be correlated with low ratings while positive ratings would be correlated with higher ratings. However, due to over 60% of the data containing positive reviews, the top 8000 most common words from which our bag of words feature vector is based off, could be biased towards more positive connotations/words, which is perhaps why

[4] Image: https://www.shangyexinzhi.com/article/285477.html
[5] Image: https://arxiv.org/abs/1810.04805
[6] https://huggingface.co/xlnet-base-cased?text=My+name+is+Thomas+and+my+main

the bag of words model performed better, and also why the best pipeline model had a strong regularization parameter. In order to penalize complexity and prevent our model from being subject to overfitting to this bias, a stronger parameter was needed. We also believe that a unigram feature performed better than bigrams exactly for this reason. The model could not generalize well to unseen bigrams. So as our n-gram feature length increases, the amount of times, in our case, that a bigram will be seen decreases due to data sparsity. In theory, while bigrams might contain more information about context (Wallach 8), it cannot generalize well to unseen data because the bigrams it will see in the training set decreases.Training the models also proved to be very costly, which is why we only used the 30,000 reviews for the training set, and 10,000 for the validation set in our pipeline. Our final model was trained on a training set of 300,000 and a test set of 100,000. We can say that our model generalized well on unseen data, our model's performance actually increased after testing on a larger portion of unseen data.
Let's compare this model to the XLNet model we implemented. The accuracies of the XLNet model seem high, considering the fact that we train only on 5000 random data points. The XLNet model beats the baseline, but let's compare it to the Bag of Words model. At first glance it seems like the XLNet model should outperform the Bag of Words model considering its complexity. To compare these results, we trained our BoW model on the same training data as XLNet and graphed the resulting accuracies.



In most cases, BoW does worse than XLNet considering it's trained on a very small portion of training data. Similarly, we can expect that XLNet performs better if it were to be trained on the entire training data and tune the parameters for the model in a pipeline.
We can plot all three accuracies on a graph to see how they do compared to the baseline.



Average XLNet accuracy = 0.673
Average BoW accuracy = 0.625
Average Baseline accuracy = 0.59

We can see that our XLNet model outperforms both models and BoW outperforms baseline. This shows how important directional context was in language modelling, which our BoW model did not incorporate.

References

He, Ruining, and Julian McAuley. "Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering." *proceedings of the 25th international conference on world wide web*. 2016.

McAuley, Julian, et al. "Image-based recommendations on styles and substitutes." *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*. 2015.

Wallach, Hanna M. "Topic modeling: beyond bag-of-words." Proceedings of the 23rd international conference on Machine learning. 2006.

“XLNet¶.” *XLNet - Transformers 4.0.0 Documentation*, huggingface.co/transformers/model_doc/xlnet.html.

Yang, Zhilin, et al. "Xlnet: Generalized autoregressive pretraining for language understanding." *Advances in neural information processing systems.* 2019.

Kowsari, Kamran & Jafari Meimandi, Kiana & Heidarysafa, Mojtaba & Mendu, Sanjana & Barnes, Laura & Brown, Donald & Id, Laura & Barnes,. (2019). Text Classification Algorithms: A Survey. Information (Switzerland). 10. 10.3390/info10040150.

Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." *arXiv preprint arXiv:1810.04805* (2018).

Sun, C., Qiu, X., Xu, Y. and Huang, X., 2020. *How To Fine-Tune BERT For Text Classification?*.

Aljuhani, Sara Ashour, and Norah Saleh. “A Comparison of Sentiment Analysis Methods on Amazon Reviews of Mobile Phones.” *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 6, 2019, doi:10.14569/ijacsa.2019.0100678.

“Xlnet-Base-Cased · Hugging Face.” *Hugging Face – On a Mission to Solve NLP, One Commit at a Time.*, huggingface.co/xlnet-base-cased?text=My+name+is+Thomas+and+my+main.