

Milestone 2 Project Report: Video Game Reviews

Team Members: Sachiko Uchikoshi, Jonathan Harper, Weifong Chou

Github: https://github.com/jonathanharper3/steam_sentiment_analysis

Introduction

Steam is a video game digital distribution service, available on a variety of platforms with over 50,000 games and 120 million monthly users. Steam's store homepage always has game suggestions and while the game's description and trailer can sound interesting, it may not always live up to expectations. Not all games have demos, so the players turn to the ratings and reviews before they make their purchasing decisions. We are interested in exploring these reviews through a sentiment analysis to classify reviews as positive, negative, or neutral as well as identify topics that can point towards game improvements. By performing this analysis we can give developers insight into the audience's response towards their game and can point towards any potential improvements that can be made for future iterations. We were motivated to do this project because of our interest in the topic and our desire to grow our natural language processing skills.

For supervised learning, we explored several different methods and through cross-validation and evaluating the different metrics, concluded that LinearSVC performed the best with our dataset. Compared to the related projects, we evaluated more models and a bigger percent of the dataset, which helped us find the best performing model that could generalize well to newer data. We also had a class imbalance that may not be found in other datasets. We used Latent Dirichlet Allocation (LDA) and Latent Semantic Indexing (LSI) as unsupervised learning methods over the three different sentiments and over the dataset as a whole, which is an approach that we had not seen done in related sentiment analysis projects. We found through this approach that a higher topic count sometimes provides better insight when compared to the optimal coherence value.

Related work

The first example is from a previous semester's project. The group performed a sentiment analysis on Airbnb data. They used Airbnb customer reviews and the underlying emotions of those reviews to compute a review score and help prospective guests determine if they want to stay at the listing or not. Likewise, they identified key themes and topics in the reviews to help hosts learn more about their customer and improve their listing. Our project is similar, as we are pursuing a similar goal of determining sentiment and creating product recommendations based on those sentiments. Our project differs in the industry we are working in and the approach we are taking to determine our suggestions. Specifically, the previous semester's team only reviewed positive reviews in their unsupervised learning approach. We leveraged reviews from all sentiments and came away with novel suggestions due to this approach.

Link: https://docs.google.com/document/d/1A709B7YUc_r_qutelQq0k_S2qa_R8EhTBkdIcXdYTvE

Two other examples come from the code section of the Kaggle data set.

The second example of a similar project is a sentiment analysis performed by Daniel Beltsazar Marpaung in August 2022. Daniel focuses on the provided review recommendations (1 for positive, -1 for negative) and the text accompanying those review recommendations to determine hidden sentiments. This is similar to our project, as we are also trying to use the text to determine the sentiment of the review. He is also utilizing word clouds on the entire data set to better understand the most common words, to hopefully point towards an understanding of the customer. Our analysis differs from Daniel's in a few ways. We are dropping the customer-provided recommendation scores (1, -1) in favor of a VADER analysis that assigns a positive, negative, or neutral sentiment based on the text in the review. We view this as an improvement on the project, as it provides more consistency, since some reviewers would include "ruined my life" in

their game review but still give it a 1 recommendation score. Daniel is also using a random forest classifier to determine hidden sentiments, which gives him an 88.7% accuracy rating. We are exploring a number of classifiers and regressors and are seeing accuracy scores above 90%, which we view as an improvement.

Link: <https://www.kaggle.com/code/danielbeltsazar/steam-games-reviews-analysis-sentiment-analysis>

The third example of a similar project is a sentiment classification performed by Pegah Pooya one year ago on the same data set. Pegah used a BERT tokenizer to analyze 2% of the data set and determine the review score (1 for positive, -1 for negative) based on the review text. This is similar to our project, as we are also trying to understand the review score / sentiment based on the text. This project differs from ours in two ways. We are analyzing approximately 60 times more reviews than Pegah, which we view as an advantage and is helping us return much higher precision scores. We are also balancing our data to reflect a significantly higher amount of positive sentiments vs. negative sentiments. We do not see any effort to do this in Pegah's code.

Link: <https://www.kaggle.com/code/pegahpooya/steam-reviews-sentiment-classification>

Data Source

We obtained our data from Kaggle. Kaggle states that the reviews compiled in this data set are publicly available reviews pulled from the Steam Reviews portion of the Steam PC store, which is run by Valve. This information can be found [here](#). This data set was downloaded from Kaggle in a csv format, and it included the following variables:

Column	Format	Description	Used
app_id	id	Game ID	Yes
app_name	string	Game Name	Yes
review_text	string	Review Text	Yes
review_score	int	Review Sentiment: whether the review recommends the game	No - used sentiment from VADER
review_votes	int	Review Vote: whether the review was recommended by another user	No

This dataset originally included 6.4 million reviews over 9,972 games. Our team only used 610,398 reviews. These reviews came from the top 20 games, measured by review volume. The exact time frame in which these reviews were made is unclear, but the data set was uploaded to Kaggle in 2017, so it is safe to assume that all reviews were made before or during 2017.

We used spaCy for text preprocessing and allowed POS tags for nouns, adjectives, verbs, and adverbs so as to yield better results during tokenization. Alongside this, we dropped null values. This helped us handle noisy or missing data.

Data Pipeline and Preprocessing

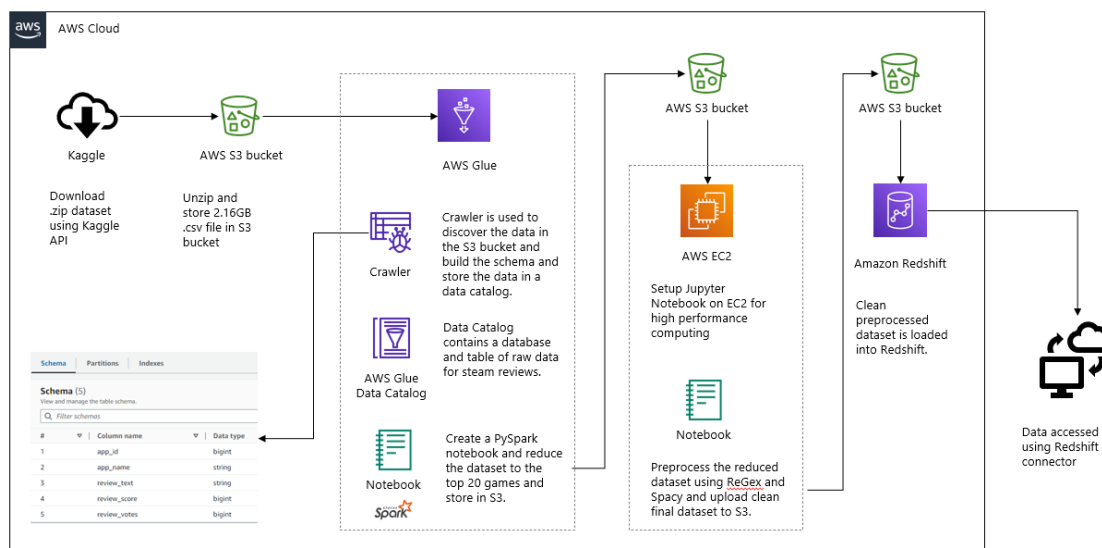
AWS was leveraged to build a data pipeline that enabled efficient ETL processing and stored the data in a Redshift data warehouse. Instead of working with files on individual machines, making use of a data warehouse from which to store and read data ensured data integrity. Jupyter Notebook was installed on

an EC2 virtual machine and data was collected using the Kaggle API. The unzipped raw dataset contained a 2.16 GB csv file, which was stored in an S3 bucket.

After storing the raw data in S3, the initial preprocessing of the data was performed in Glue. A crawler was used to access the raw data stored in S3, extract the metadata, and populate the data into a table that is stored in a data catalog. The benefit of using a crawler is that as data is added to S3 it is automatically added to the data catalog. Since we were dealing with 6.4 million records, an ETL job was created in Glue using PySpark where data was read from the data catalog and then preprocessed. The PySpark preprocessing job dropped null values, trimmed the review text by removing whitespace, and dropped reviews that contained “Early Access Review” since these reviews were considered noise.

The data was further reduced to the top 20 games measured by review volume and the review text was lowercased before uploading into S3. Additional AWS info can be found in Appendix 4.

AWS Service	Description	Use Case
S3	Object storage service	Used to store all data
EC2	Elastic virtualized computing service	Used for computationally expensive processing and model training
Glue	ETL service for data preparation and integration	Used to create a run ETL job and prepare data for analytics
Redshift	Cloud data warehouse	Used to store clean processed data

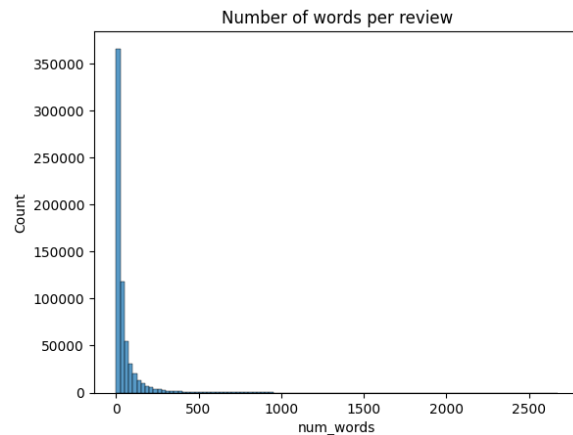


Feature engineering

After performing the ETL process in Glue by using PySpark to reduce the data to the top 20 games by review volume, an EC2 virtual machine was setup with an instance size of C5a.4xlarge which provided 16 virtual CPUs and 32 GB of memory. Being able to make use of a virtual machine with additional processing power was an upgrade over Deepnote’s standard virtual machine that has 2 vCPUs and 5 GB of memory.

The AWS wrangler Python library was used to clean tokenized data, which stood at 846,712 rows. Regex was applied to remove punctuation from the review text, which resulted in 10,422 rows with empty review text after applying regex. These rows were dropped and reviews were then checked for duplicates. 183,821 reviews were found to be duplicates and after dropping these rows, the dataset stood at 652,453 records.

Given that reviews vary in length, the number of words per review was plotted along with the number of words for various quantiles. It was found that 5% of reviews had 4 words or less and 95% of reviews had 197 words or less. In order to remove noise from the dataset, we focused on reviews between a length of 4 and 197 words, which reduced the dataset to 610K reviews. A boxenplot was created to visualize a nonparametric representation of the distribution of words per review for each game. The boxenplot can be found in Appendix 5.



The language processing pipeline was set up using spaCy using the `en_core_web_sm` pipeline which is their smallest English language model at 12MB. Since we were processing 610K records, we made use of spaCy's `nlp.pipe` method which processes batches of text as a stream instead of one-by-one. Stop words were removed and words were then tokenized and lemmatized. In order to minimize the time to tokenize each review, Joblib was utilized in order to parallelize the tokenization process. Before implementing this parallelization job, it took 35 minutes to tokenize each review but with parallelization it took 12 minutes.

After we downloaded the tokenized data from Redshift, we performed a VADER analysis on it to determine sentiment. VADER stands for Valence Aware Dictionary and sEntiment Reasoner, and "it is a lexicon and rule-based sentiment analysis tool that is specifically attuned to sentiments expressed in social media" (Cjhutto). Each review was assigned a normalized, weighted composite score, which is the normalized sum of the valence scores of each word in the lexicon. This score sits between -1 and 1. A score greater than or equal to 0.05 was assigned a positive sentiment. A score less than or equal to -0.05 was assigned a negative sentiment. Any score between -0.05 and 0.05 was assigned a neutral sentiment. This is in line with the VADER documentation's suggestions.

Supervised Learning

With the data preprocessed, the first step of the supervised learning workflow was the term frequency-inverse document frequency (TF-IDF) vectorizer. As we are working with text data, the text needs to be represented as a numerical feature for us to process it. The vectorizer combines the term frequency and document frequency to produce a weight for each word in the dataset that is high for rare terms and low for common terms, which was used as the features for our supervised learning methods. Additionally, through the vectorizer, we were able to set some parameters, such as `max_df`, which allowed us to remove corpus-specific stop words, and `min_df`, which allowed us to remove infrequent words that may be misspelled words or just rarely used words.

To begin, we used Logistic Regression as our baseline model with a balanced class weight. This classifier attempts to fit a logistic function to find the relationship between the variables, which are the values from the vectorizer. As there was an imbalance to our classes (more than 70% was positive sentiment), the class weight parameter penalizes mistakes in the samples of the class with the class weight, so misclassifying an observation from a minority class is more heavily penalized than misclassifying an observation from a majority class. This parameter is available in a couple of the models that were used, but for the other parameters, the defaults were used. Continuing with the deterministic models, the next

method that was tested was Support Vector Machines (SVM), which creates decision boundaries with hyperplanes to separate the data into classes. These hyperplanes should fit more closely to the data than a logistic function would. We used the linear kernel first as it was the default, but it performed pretty well, so other kernels weren't tested. Default parameters were used for the initial exploration, but with a balanced class weight. Switching to a probabilistic model, we built a model with Multinomial Naive Bayes, which focuses on the counts for each individual word in each review, as Naive Bayes assumes that the features are conditionally independent with little to no correlation to each other, given the class. The last three classifiers in the initial exploration were tree-based: Decision Tree, Random Forest, and Gradient Boosting Decision Trees. Decision Tree uses a series of conditional steps to make its decision, however, while it is usually good at making decisions, it has a tendency to overfit. This is why we also tested Random Forest, which uses multiple Decision Trees independently and aggregates their results into a single final result, which can help with the overfitting issue. Similarly, Gradient Boosting Decision Trees also use multiple Decision Trees to arrive at a final result, but these trees are built to improve upon the previous trees and minimize the loss function. However, they are also prone to overfitting.

To evaluate each of these models, several metrics can be computed from the 3 by 3 confusion matrix. Additionally, for some of the metrics, because multiclass evaluation was not available, we used the one-vs-rest (OVR) strategy, where the classification is converted to a binary decision. For example, it would become positive vs. not positive instead of positive vs. neutral vs. negative. The first metric is accuracy, where the predicted class matches the true class, which is done with micro averaging. Unfortunately, accuracy can sometimes be a misleading metric because it does not account for class imbalance. With our data, if the classifier labeled all the samples with positive sentiment, we would achieve 84% accuracy. The other metrics are done with weighted averaging, meaning it is a mean of the metrics between all classes computed individually and weighted to account for the class imbalance. Precision is the proportion of predicted positives that are truly positive and recall is the proportion of actual positives that are correctly classified. Depending on the use of the classifiers, one of these metrics may be more important than others, as precision and recall are in a trade-off relationship. Fortunately, the F1 score combines these two metrics, minimizing both the false positives and false negatives in an imbalance classification and is the score that was mainly used to pick our final model.

To explore further, we also calculated the Area Under the Receiver Operating Characteristic Curve (ROC AUC) score, which can be explained as the average of F1 Scores evaluated at various thresholds. However it does not handle class imbalance well. An additional metric that we used for evaluating hyperparameter tuning is Log Loss. It is an error function, so a classifier that minimizes the log function would be better. For the initial exploration comparison, a 5-fold cross-validation was used to gather the metrics instead of using a single train/test split to better evaluate the model's performance and generalization ability. From the following table, we can see the mean for each metric as well as the standard deviation from performing the cross-validation. The standard deviation seems small, so the scores from each of the data set splits were not too far from each other.

	Accuracy Mean	Accuracy Std	Precision Mean	Precision Std	Recall Mean	Recall Std	F1 Mean	F1 Std	ROC AUC Mean	ROC AUC Std
LogisticRegression	0.859	0.024	0.892	0.012	0.859	0.024	0.868	0.021	0.958	0.012
LinearSVC	0.901	0.020	0.901	0.017	0.901	0.020	0.899	0.018	0.958	0.013
MultinomialNB	0.753	0.028	0.735	0.034	0.753	0.028	0.696	0.020	0.852	0.035
DecisionTreeClassifier	0.764	0.018	0.824	0.008	0.764	0.018	0.781	0.015	0.841	0.014
RandomForestClassifier	0.780	0.020	0.838	0.012	0.780	0.020	0.795	0.017	0.897	0.011
GradientBoostingClassifier	0.853	0.016	0.848	0.014	0.853	0.016	0.848	0.014	0.926	0.015

After our initial exploration, we decided to hyperparameter tune the LinearSVC model as it had the highest F1 Score and also the highest value in the other metrics as well. Different combinations of the penalty, loss, and class_weight were tested and scored. For the penalty, L2 is the standard used in SVC and is the default, while L1 leads to coef_ vectors that are sparse. For the loss function that measures the

quality, hinge is the standard SVM loss. However, squared_hinge, which is the square of the hinge loss, is the default. Interestingly, the combination of L1 and hinge was not supported and the combination of L1 and squared_hinge required the dual parameter to be false, which determines whether the algorithm solves the dual or primal optimization problem. Otherwise, the dual parameter was kept default. For each of these combinations, we tested with multiple C values (0.01, 0.1, 1, 10, 100), which is the regularization parameter where the strength of the regularization is inversely proportional to C.

	Accuracy	Precision	Recall	F1 Score	ROC AUC Score	Log Loss
L2 Hinge 10	0.942	0.941	0.942	0.941	0.966	0.287
L2 Hinge 100	0.942	0.941	0.942	0.941	0.966	0.288
L2 Hinge 1	0.941	0.940	0.941	0.940	0.966	0.287
L2 Hinge Balanced 100	0.936	0.935	0.936	0.935	0.970	0.284
L2 Hinge Balanced 10	0.936	0.934	0.936	0.935	0.970	0.284
L2 Hinge Balanced 1	0.935	0.934	0.935	0.935	0.970	0.283
L1 Squared 1	0.934	0.933	0.934	0.933	0.970	0.274
L2 Squared 1	0.934	0.932	0.934	0.933	0.970	0.275
L1 Squared 10	0.933	0.932	0.933	0.932	0.970	0.275
L2 Squared 10	0.933	0.932	0.933	0.932	0.970	0.275
L2 Squared 100	0.933	0.932	0.933	0.932	0.970	0.275
L1 Squared 100	0.933	0.932	0.933	0.932	0.970	0.276
L1 Squared 0.1	0.933	0.932	0.933	0.932	0.970	0.276
L2 Squared 0.1	0.933	0.931	0.933	0.932	0.970	0.276
L1 Squared Balanced 0.1	0.931	0.930	0.931	0.930	0.971	0.277
L1 Squared Balanced 1	0.931	0.930	0.931	0.930	0.971	0.277
L2 Squared Balanced 100	0.931	0.929	0.931	0.930	0.970	0.280
L2 Squared Balanced 1	0.930	0.929	0.930	0.929	0.971	0.278
L2 Squared Balanced 0.1	0.930	0.929	0.930	0.929	0.971	0.277
L1 Squared Balanced 10	0.930	0.928	0.930	0.929	0.971	0.279
L2 Squared Balanced 10	0.930	0.928	0.930	0.929	0.971	0.279
L1 Squared Balanced 100	0.930	0.928	0.930	0.929	0.971	0.279
L2 Hinge 0.1	0.930	0.928	0.930	0.928	0.964	0.294
L2 Hinge Balanced 0.1	0.926	0.925	0.926	0.925	0.968	0.290
L2 Squared 0.01	0.917	0.915	0.917	0.915	0.965	0.296
L2 Squared Balanced 0.01	0.916	0.915	0.916	0.915	0.967	0.294
L1 Squared Balanced 0.01	0.904	0.904	0.904	0.902	0.960	0.319
L1 Squared 0.01	0.903	0.903	0.903	0.900	0.957	0.329
L2 Hinge 0.01	0.903	0.900	0.903	0.899	0.955	0.328
L2 Hinge Balanced 0.01	0.897	0.896	0.897	0.894	0.959	0.322

With the combinations of all these parameters, it also allows us to test the sensitivity of our model and see how each of the metrics change with each parameter change. While tuning the parameters does affect the results, it seems that the regularization parameter C has a greater effect than the penalty, loss, and class_weight. This parameter tells the SVM optimization how much to avoid misclassifying each training sample. For larger values of C, the optimizer chooses a smaller margin hyperplane if that hyperplane

performs better, while smaller values will have the optimizer choose a larger margin hyperplane even if the hyperplane misclassifies more points.

As we are dealing with reviews, which is text based data, the features of our model from the TF-IDF vectorizer are the words from the data set. With the given parameters, the vectorizer had a total of 4110 words, or features. Retrieving the feature names from the vectorizer, we can match up the weights assigned to each of the features from the classifier and retrieve the list of words with the largest and smallest weight for each of the sentiments. The words furthest from zero have a bigger effect on the classifier than the ones closer to zero. It's interesting to note that most of the words with the largest positive weight for the negative sentiment have the largest negative weight for the positive sentiment and vice versa. It's also interesting to see that the largest positive weights for the neutral sentiment have pretty small scores compared to the negative and positive sentiments. Looking at the list of words, these words have a mix of positive and negative connotations. However, for the negative weights for the neutral sentiment, there are a few words that are from the positive sentiment's largest weighted words.

Through the evaluation, there were a few noticeable tradeoffs. The first is the effect of the C parameter on speed. With larger C values, especially when C = 100, the classifier consistently took longer to run than when C is small. It was also noticed that models with the balanced class_weight often took longer to run than the models without the class_weight set. Additionally, within the tree-based models, there appears to be a tradeoff between speed and the performance of the models. Decision Tree ran quickest, then Random Forest, then Gradient Boosting and that is also the order of worst to best metrics for accuracy, precision, recall, F1, and ROC AUC scores. There is a known tradeoff relationship between precision and recall. However, it was not observed in our models, though this could possibly be due to the class imbalance, as we used the weighted precision and weighted recall instead of the macro weighted one. These would have given equal weight to each of the three classes.

Even with perfect parameters, natural language can at times be difficult to classify. After looking through the reviews, there are a few main categories of failures. The first is irrelevance, where the review does not say anything about the gameplay, instead commenting on something else that is possibly related to the game or comparing it to something/someone. The second is sarcasm, where the review sounds like one sentiment, but is actually not the intended sentiment. The last issue that occurs frequently is misspellings or abbreviations, where the word may have a strong weight, but is instead disregarded due to the TF-IDF vectorizer. Though some issues, such as the irrelevance or sarcasm, are hard to fix, one of the easier to fix issues would be a spellcheck on the reviews for the words to be assigned a correct weight for the classifiers. Here are a few examples of where our classifier failed.

Negative				
	Largest	Weight (L)	Smallest	Weight (S)
0	kill	11.82	great	-12.99
1	hell	9.30	love	-10.68
2	bad	8.68	amazing	-9.97
3	murder	8.31	fun	-9.60
4	die	7.93	awesome	-9.18
5	dead	7.68	good	-9.16
6	cancer	7.32	friend	-7.08
7	terrorist	7.16	win	-6.84
8	war	7.10	perfectly	-6.75
9	ruin	7.01	beautiful	-6.68
Neutral				
	Largest	Weight (L)	Smallest	Weight (S)
0	overlook	2.47	good	-10.98
1	manner	2.19	play	-10.70
2	topic	2.10	great	-9.68
3	accuracy	2.05	fun	-9.30
4	accomplishment	1.91	love	-8.98
5	vibe	1.91	like	-8.36
6	instance	1.84	amazing	-7.62
7	royale	1.78	awesome	-7.49
8	discourage	1.75	well	-7.14
9	fleet	1.68	recommend	-7.02
Positive				
	Largest	Weight (L)	Smallest	Weight (S)
0	great	18.17	kill	-12.45
1	love	15.31	hell	-10.12
2	good	14.18	dead	-8.50
3	fun	13.88	die	-8.22
4	amazing	13.31	murder	-8.09
5	awesome	13.30	enemy	-8.08
6	play	9.52	bad	-7.99
7	friend	9.39	terrorist	-7.76
8	beautiful	8.95	cancer	-7.38
9	fantastic	8.59	hatred	-7.13

Example: “love the game - hate the updates... i have a mac (haters don't hate) and every time gmod decides to make an update, i can't install it. basically, half the time i log in to play, i end up not being able to play. why? Why?!?!?!?”

The sentiment assigned by VADER is positive, yet the classifier assigned negative. This example had both positive and negative words, but is an overall positive review, however, it has the word hate in it multiple times, which may have caused the classifier to assign a negative sentiment.

Example: “i can't walk into banks like a normal person anymore...”

The sentiment assigned by VADER is negative, yet the classifier assigned positive. This review is possibly more specific to the game and requires more context on the game, though it is not really a game review. The classifier may have classified it as positive because of the word like.

Example: “despite all my rage, this game is better then nicholas cage.”

The sentiment assigned by VADER is positive, yet the classifier assigned negative. After the text preprocessing, the remaining words that may have a larger impact could be rage and better. There are 17 samples that contain Nicholas Cage and neither word was a feature from the vectorizer.

Unsupervised Learning Methods

We chose to use topic modeling as our feature representation to understand the underlying topics or themes in our corpus of text. We chose this feature representation because we wanted to understand the most important aspects of the reviews we were looking at, and topic modeling can make this understanding possible by transforming high-dimensional text data into a lower-dimensional space.

Specifically, we used Latent Dirichlet Allocation (LDA) through the sklearn and Gensim libraries and Latent Semantic Indexing (LSI) through the Gensim library. LDA gives a topic modeling, probabilistic approach, while LSI gives a dimensionality reduction, non-probabilistic approach. We chose these two approaches because of their usefulness in identifying the underlying semantic structure of a corpus. By doing this, we can identify keywords and topics that may point towards product enhancements.

Latent Dirichlet Allocation (LDA)

We used both the sklearn library and the Gensim library when performing our LDA.

For the sklearn version of this exploration, we first created a model using sklearn's TF-IDF vectorizer. This model was enhanced using parameters like `strip_accents` (unicode), `ngram_range` (1, 2), `min_df` (3), and `max_features` (5000). The `ngram_range` was used to cover the bigram extraction, while `min_df` was used to avoid words with a frequency less than 3. All other parameters were kept in their default format. This model was fit and transformed using our cleaned tokens.

After this TF-IDF vectorizer matrix was built, we created an LDA model and fit it to the matrix. The LDA model included parameters `n_components` (14), `n_jobs` (-1), and `learning_method` (online). The `n_components` parameter was used to determine how many topics the model needs to retrieve and was challenging to initially set, as there wasn't an easy way to find the optimal topic amount. We used an evaluation metric function in the Gensim library called coherence to determine the number of topics that would provide the most interpretable and quality topics. This evaluation showed 14 as the optimal topic count when it concerns the entire corpus, and we fine-tuned this parameter accordingly.

With our model tuned appropriately and the TF-IDF matrix fit, we used the `get_feature_names()` function on the TF-IDF matrix and the `components_` attribute on the LDA model to retrieve, sort, and print the topics that are seen below.


```

Topic #1:
game play not good like great fun time amazing story

Topic #2:
buy not minecraft like minecraft game stop buy game play bad ball

Topic #3:
good good game game play game play game good favorite favorite game addict time

Topic #4:
ninja like masterpiece like game toby space ninja game like beat new update age

Topic #5:
boss ninjas kill space adventure awesome car to soccer space ninjas

Topic #6:
skeleton review wonderful gwent tear game great possibly write rip great gameplay

Topic #7:
life hour pay win valve win hour fun real goty minute ill

Topic #8:
grind need update repetitive dead run thing game forever thing flower

Topic #9:
nice amazing addictive nice game recommend word game highly game amazing highly recommend

Topic #10:
awesome love love game awesome game cry game awesome game game love dota far

Topic #11:
fun great great game friend game play friend fun game play fun play lot

Topic #12:
terraria ve ve play game ve music good play good game game wow

Topic #13:
undertale witcher ing video video game yes suck penny xd geralt

Topic #14:
cool well pretty worth game fun game steam game well steam pretty good simulator

```

These topics are difficult to interpret, and so we chose to utilize the Gensim package to supplement the LDA we derived using sklearn. We also chose to analyze the corpuses according to their sentiment to build results that are easier to interpret. This meant that we added three sub-datasets that were split according to sentiment.

For the Gensim version of our exploration, we created a dictionary using the corpora.dictionary function, filtered out tokens that appeared in two or less documents, and converted the text to a document-term matrix using the doc2bow function. Using this matrix, we built an LdaModel with the matrix passed under the corpus parameter and an id2word tool that mapped each ID to the corresponding word in our dictionary. Other parameters included in the matrix were chunksize (2000), iterations (400), passes (20), eval_every (None), num_topics (6), alpha (auto), and eta (auto). These parameter values, aside from num_topics, were referenced from the Airbnb sentiment analysis project referenced earlier, and we found them to perform optimally compared to other tested values. Num_topics was determined through our coherence score.

An attempt was made to see if the LDA model could be improved through the use of bigrams and trigrams by using gensim's Phraser to extract bigrams and then extract trigrams using min_count=10 and a threshold of 20. After extracting bigrams and trigrams, gensim's TF-IDF model was used to remove words with low importance using a threshold of 0.03 to ensure that the most common words don't show up as key words. After TF-IDF was used to update the corpus, the number of topics was determined through the CV coherence score.

Latent Semantic Indexing (LSI)

We used the Gensim library while performing our LSI.

Much of the pre-preparation steps we took for LSI were similar to the steps taken for our Gensim version of LDA. We created a dictionary using the corpora.dictionary function, filtered out tokens that appeared in two or less documents, and converted the text to a document-term matrix using the doc2bow function. Then, using this matrix and dictionary, we built an LsiModel with num_topics and chunksize as the only

adjusted parameter. Both of these parameters are identical to the parameters that were used for our Gensim LDA analysis and are used for the same reason, that being that they were shown through testing to perform optimally compared to other parameter values.

The model was run on all four sentiment options (positive, neutral, negative, all) and the resulting topics were printed for analysis. However, in a text format, these topics were difficult to interpret, so we chose to utilize the WordCloud library to visualize the data and make it easier for interpretation.

Unsupervised Learning Evaluation

Evaluation Metrics

With LDA and LSI models, coherence scores can be calculated to evaluate the effectiveness of your methods and choice of feature representation.

Coherence measures the interpretability and quality of the topics in our topic model by measuring the degree of semantic similarity between high probability words within a topic. A higher coherence score corresponds with a more coherent topic. This function works well for our purposes, as we are wanting to form focused topic clusters that lead to useful recommendations.

To build our coherence model, we used the CoherenceModel function that is found in the Gensim library. We measured coherence using an LsiModel that was pre-trained using our doc2bow matrix, the GloVe 300d model, and the c_v coherence measurement. The GloVe model was added to combine the strengths of our domain-specific model with something more broad and meant for general purposes. We used a c_v measurement in favor of u_mass or other coherence measurements because we had the pre-trained, domain-specific word embedding model available.

Our topic coherence functions were used to evaluate our model and fine tune its parameters. Since the coherence score is largely dependent on the number of topics retrieved, we used a function that includes an iterable “for loop”, which allows us to measure coherence values with varying levels of topic numbers. These values were plotted using matplotlib and show the number of topics that is optimal.

Sentiment	2 Topic Coherence Value	4	6	8	10	12	14	16	18
Positive	.5577	.5978	.5617	.5922	.5916	.5958	.5866	.5796	.5767
Negative	.6007	.6065	.6017	.6171	.6105	.6022	.6162	.6008	.6124
Neutral	.4323	.4267	.4973	.5306	.5578	.5592	.5618	.5446	.5492
All	.6070	.5603	.5672	.5742	.5369	.5729	.5745	.5662	.5510

The values shown in green are the topic value counts we chose to use for our analysis. Some of these sentiments show a smaller number of topics with a higher coherence value. These values are shown in yellow. For example, we see a .6070 coherence value for two topics over all sentiments. We chose to favor the second highest coherence score, at 14 topics, in this instance with the thought that more topics would lead to more product enhancement options. Line graphs showing these same values can be found in appendix 2.

Sensitivity Analysis

As is shown, the sensitivity of our model varies based on the sentiment and the number of topics. This influences our hyper-parameters, as we chose our number of topics based on which topic amount returned the highest coherence value. We can see how sensitive our model is to the number of topics hyper-parameter by changing it. We chose to test this sensitivity by building LSI word clouds visualizing

our corpus in two, eight, fourteen, and sixteen topics models. These represent the two lowest coherence scores (2 - .6007, 16 - .6008) and the two highest coherence scores (8 - .6171, 14 - .6162).



Since these coherence scores aren't very different from each other, the model may not be too sensitive to different topic number parameters. Mostly, we see the benefit of more topics, as the word clouds for the 2 topic number parameter show the exact same keywords in both clouds and isn't very informative. In contrast, the larger topic number parameters show some overlap but also show topics with unique keywords, like "connection", "buy", and "weapon".

Other Potential Evaluation Metrics

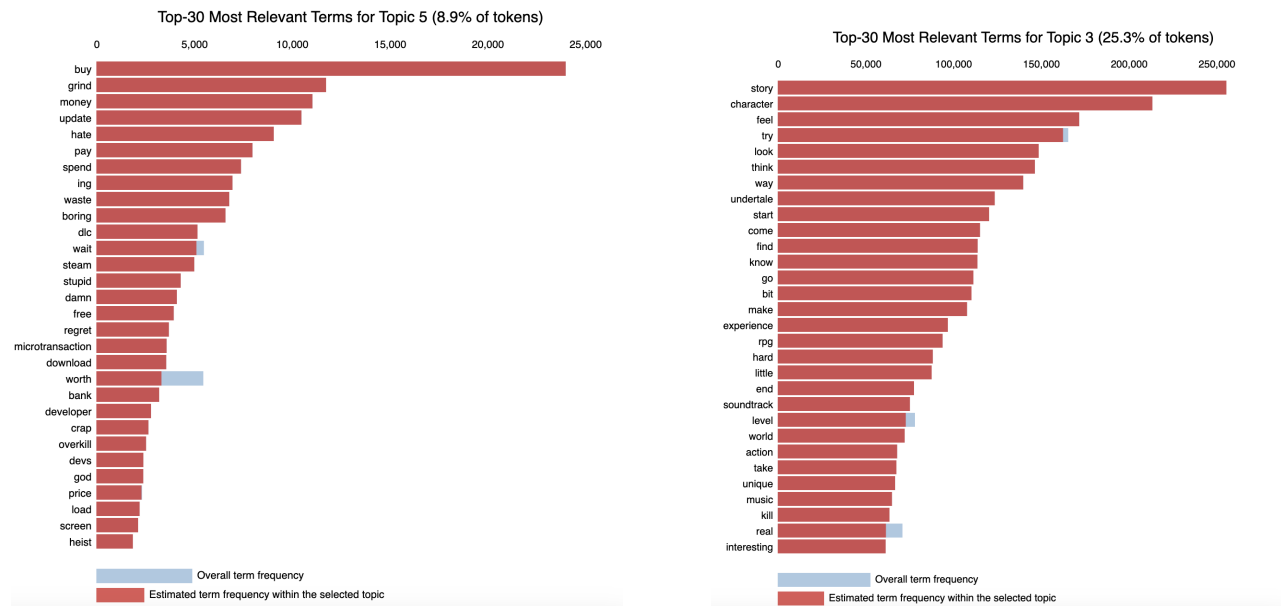
There are other evaluation metrics that could also be used for LDA and LSI models. We will discuss two here and explain our reasoning for not pursuing these metrics.

Perplexity measures how well the model predicts the observed data or test set and is used to evaluate the ability of a topic model to generalize to new data. The lower the perplexity, the better the model is at predicting the observed data or test set. A high perplexity indicates that the model is not able to predict the test set well, which can be a sign of overfitting or poor model selection. In our project, we are not trying to predict a test set of data. Instead, we are trying to observe the data that is already available. Therefore, a perplexity score wouldn't aid our evaluation.

One other method to evaluate the quality of LDA and LSI is through human evaluation. This may include human annotators assessing the interpretability of a topic in the model through surveys, interviews, etc. This evaluation metric would have made sense and potentially been helpful for our purposes, but we did not have the resources available to pursue it.

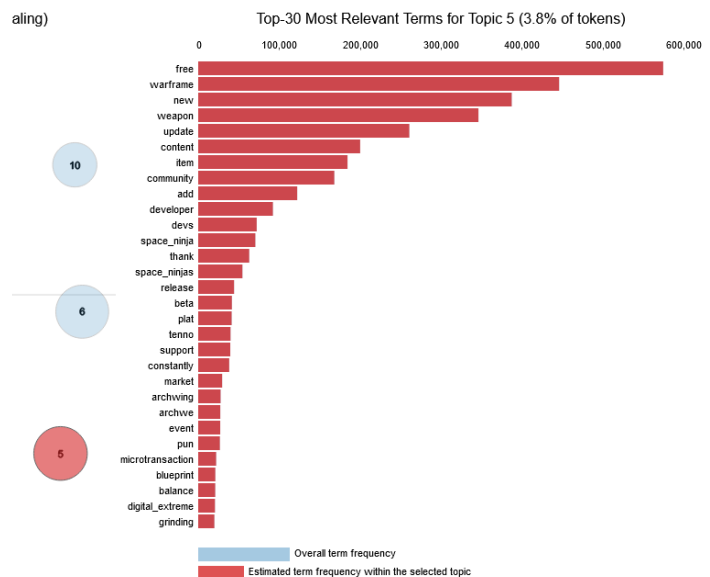
LDA Results Reporting

Top topics from the corpus were printed and words like "like", "good", "play", and "free" were shown to be top salient terms for positive review topics. However, these results were difficult to interpret in a printed, textual form, so we used the pyLDAvis library as a tool to better interpret our results.



We saw specifically through topic 5 that around 9% of negative reviews were financial concerns, and they may have felt they were wasting money on features that didn't necessarily meet their expectations. This can be compared to topic 3 that shows that 25% of positive reviews came from strong stories, characters, and interesting experiences. With this in mind, game developers can take inspiration from these storylines and add that inspiration to aspects of their paid features.

The plot on the right is from the LDA model using bigrams and trigrams. For this model we used 10 topics and words rarely overlapped between topics. It was difficult to determine a topic that summarized a group of reviews but it was interesting to see that bigrams were found in some topics. For instance, bigrams found in topic 5 included `space_ninja`, `space_ninjas`, and `digital_extreme`. Topic 9 contained the bigrams `farming_simulator`, `bullet_sponge`, `hand_down`, and the trigram `co_op_shooter`. Perhaps if more than 20 games were analyzed it would be easier to find a collection of words that define a specific genre of games. Links to the full pyLDAvis charts can be found in Appendix 3 of this report.



LSI Results Reporting

These topic models, specifically in the negative sentiments, show that “kill”, “cut”, “shoot”, and “cheat” are keywords that show up, suggesting that these are areas for improvement. Likewise, in the positive sentiments topic model, we see keywords like “story”, “friend”, “arma”, and “realistic”. These point towards features or games that customers enjoy, which may help direct future enhancements. Larger topic model visualizations can be seen in Appendix 1.

Unsupervised Learning Discussion

Surprises

We were surprised by the impact of the storyline of the game. We have varying experiences with video games, but we originally viewed action, graphics, and ability to interact with other gamers as key aspects of a successful video game.

Challenges

With textual data, and especially with the unsupervised portion of this project, we recognized that proper pre-processing of data is critical, and this turned out to be a challenge. We initially removed punctuation from our textual data but found that this led to incorrect tokenization that made results difficult to understand. We responded to this challenge by researching part-of-speech tagging and re-processing our data with this new information.

Extend solution with more time / resources

There are two different areas where we think we could extend our analysis if we had more time and resources.

First, there were ~ six million reviews in the original dataset, but we chose to only analyze 610,000 reviews. This reduction of the dataset helped with our task, as 610,000 reviews still took a considerable time to compute. However, specifically with our topic modeling task, the reduction limited our ability to be effective, and we think that we could have provided better recommendations if we had analyzed more data.

Secondly, we would want to make our analysis more granular. We had originally pulled reviews from only the top twenty games, and our intention was to make recommendations based on the game and the sentiment and not based on the sentiment. However, this proved to be a little more time-consuming than we originally anticipated, and we made recommendations based on the sentiment alone. Given more time, we think this extension would be absolutely possible and useful.

Ethical Considerations

We run into the potential for bias as the reviews we've collected may not be representative of the population at large. This is particularly true of reviews, as they are voluntary and often take more time than those who aren't strong supporters or detractors would care to spend. If we focus too heavily on biased data, we may make changes to a game that would reflect this bias.

Providing a solution also opens the door for misuse. This misuse could be illegal and include something like racial or religious profiling, or it could be legal but unethical, like building a game that purposefully targets addictive behaviors. If the overall sentiment for the reviews of a game were published, reviewers can also leave many reviews and impact the sentiment, potentially damaging the reputation of a game.

Statement of Work

- Sachiko: AWS Data Pipeline and Data Preparation
- Jonathan: Determine sentiments through VADER
- Weifong: TF-IDF Vectorizer and Supervised Learning
- Sachiko, Jonathan: Build and evaluate LDA and LSA tools to identify the most discussed topics.
- Sachiko, Jonathan, Weifong: Project report and visualizations

References

Cjhutto. (n.d.). *CJHUTTO/Vadersentiment: Vader sentiment analysis. vader (valence aware dictionary and sentiment reasoner) is a lexicon and rule-based sentiment analysis tool that is specifically attuned to sentiments expressed in social media, and works well on texts from other domains*. GitHub. Retrieved February 28, 2023, from <https://github.com/cjhutto/vaderSentiment>

Larxel. (2021, November 29). *Steam reviews*. Kaggle. Retrieved February 28, 2023, from <https://www.kaggle.com/datasets/andrewmvd/steam-reviews>

Learn: Machine learning in python - scikit-learn 0.16.1 documentation. scikit. (n.d.). Retrieved February 28, 2023, from <https://scikit-learn.org/>

Michwynn. (n.d.). *Michwynn/London-Airbnb-analysis---2: Milestone II*. GitHub. Retrieved February 28, 2023, from <https://github.com/Michwynn/London-Airbnb-Analysis---2>

SIADS 542: Supervised Learning. Coursera. (n.d.). Retrieved February 28, 2023, from <https://www.coursera.org/learn/siads542/home/week/1>

SIADS 543: Unsupervised Learning. Coursera. (n.d.). Retrieved February 28, 2023, from <https://www.coursera.org/learn/siads543/home/week/1>

Appendix

Appendix 1

Word cloud with all sentiments combined

Topic 0
buy like get
play hour
good fun
great time
friend

Topic 2
people
get great
new time
like buy
thing
feel fun

Topic 4
amazing
great
gameplay like
love story
play bioshock
character

Topic 6
story get
hour character
fps time
run world
kill new

Topic 8
amazing love
arma
community
mod new
military
dlc weapon life

Topic 10
love arma
feel like
new hey
time get
weapon mod

Topic 12
interrupt
weapon
connection
minecraft bos
fast friend
terrarium time
warframe

Topic 1
like story
buy
feel graphic thing
good
lot great
gameplay

Topic 3
mod graphic amazing
great
time lot
fun story
gameplay friend

Topic 5
hour time
love
buy want
dlc go money
worth

Topic 7
run get fps
zombie
great
server
arma mod pc
player

Topic 9
pc hour
buy time
online issue
crash
fps like run

Topic 11
dlc arma
terrarium minecraft
hour like
time get
update amazing

Topic 13
zombie space
ninja shooter
free
car fast
warframe
arma crash

Word cloud with only positive sentiment reviews

Topic 0
time buy like
play
friend story
good fun
hour great

Topic 2
great lot
buy fun
feel thing
like
new love time

Topic 4
play amazing
character story
feel like graphic
great
bioshock love

Topic 6
friend zombie pay
dlc **buy**
good
great
money mod
community

Topic 8
buy story
infinite time
pay
dlc money
gameplay
bioshock

Topic 10
like bioshock
time
feel
dota new mod
weapon
fallout dlc

Topic 1
character
like story
gameplay
buy
pretty
good
graphic great
thing

Topic 3
minecraft
buy u
like
feel people
good play
want

Topic 5
love story
buy new
amazing want hour
get
time
dlc

Topic 7
update
mod new
terrarium dota
time
player weapon
lot
community

Topic 9
fallout
amazing new
mod vega
garrys
well arma story
community

Topic 11
shooter graphic
bioshock mission
military fps
community
run arma
realistic

Word cloud with only negative sentiment reviews

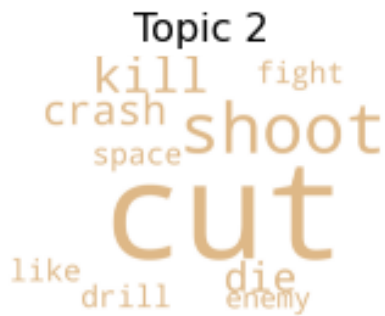
Topic 0



die good
play
people get
bad like buy
kill time

This word cloud for Topic 0 features the word 'play' in a large, bold, purple font at the top. Below it, 'like' is also in purple and slightly smaller. Other words in purple include 'people', 'get', 'bad', 'buy', 'kill', and 'time'. Smaller words in a lighter purple include 'die' and 'good'.

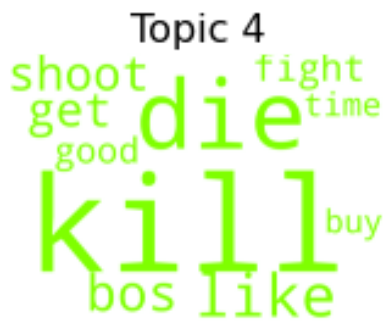
Topic 2



kill fight
crash shoot
space
cut
like drill die enemy

This word cloud for Topic 2 is primarily orange and brown. The word 'cut' is the largest and most prominent in the center. Other large words include 'shoot' and 'kill'. Smaller words include 'fight', 'crash', 'space', 'like', 'drill', 'die', and 'enemy'.

Topic 4



shoot fight
get die time
good
kill buy
bos like

This word cloud for Topic 4 is green. The word 'kill' is the largest and most central. Other large words include 'die' and 'shoot'. Smaller words include 'fight', 'time', 'good', 'buy', 'bos', and 'like'.

Topic 6



dlc weapon
buy like
people good
bad money
interrupt connection

This word cloud for Topic 6 is orange and red. The word 'bad' is the largest and most prominent. Other large words include 'buy' and 'connection'. Smaller words include 'weapon', 'like', 'people', 'good', 'money', and 'interrupt'.

Topic 1



good enemy bos
fight zombie
kill
die like shoot
weapon

This word cloud for Topic 1 is red. The word 'kill' is the largest and most central. Other large words include 'die' and 'like'. Smaller words include 'enemy', 'bos', 'fight', 'zombie', 'shoot', and 'weapon'.


Topic 3



bos crash
kill
fight enemy like
die get shoot
weapon

This word cloud for Topic 3 is teal. The word 'kill' is the largest and most central. Other large words include 'die' and 'like'. Smaller words include 'enemy', 'fight', 'get', 'shoot', 'weapon', and 'bos'.

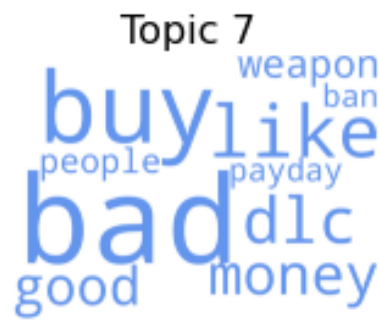
Topic 5



soul kill
play
die hard bos
hour enemy fight dark

This word cloud for Topic 5 is orange. The word 'play' is the largest and most central. Other large words include 'die' and 'kill'. Smaller words include 'hard', 'bos', 'hour', 'enemy', 'fight', and 'dark'.

Topic 7



weapon ban
buy like
people payday
bad dlc
good money

This word cloud for Topic 7 is blue. The word 'buy' is the largest and most central. Other large words include 'like' and 'bad'. Smaller words include 'weapon', 'ban', 'people', 'payday', 'dlc', and 'money'.

Word cloud with only neutral sentiment reviews

Topic 0
anymore
hear
buck cost
account bit soul
grind dog
shooter

Topic 2
flake
play
temmie
time craft
learn dog need
like
hol

Topic 4
good
get
pond
skull
play time
hour
like life
drink

Topic 6
good
component
get
time
directx update
money
play
like hour

Topic 8
run like
directx
get play
hour money mod
update
component

Topic 10
good time
hour new
preal like
play
get content

Topic 12
directx like
play go
component try get
time good
story

Topic 1
like
time
hour
get
microtransaction
need play
drill
good

Topic 3
play hour
craft
fill need
time
like good
get
thing

Topic 5
good like
need
go play
money
get time
hour update

Topic 7
like
get money
time
component play good
update
hour
directx

Topic 9
time get
hour like
fun
play
money
good go need

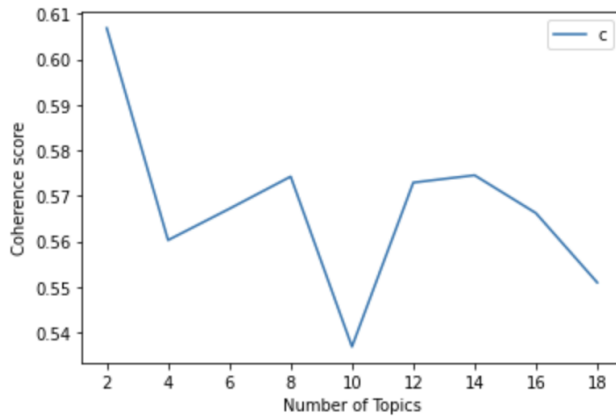
Topic 11
good like
content
money go
get real
play mod time

Topic 13
minecraft
fun play
content good
stop like go
kill real

Appendix 2

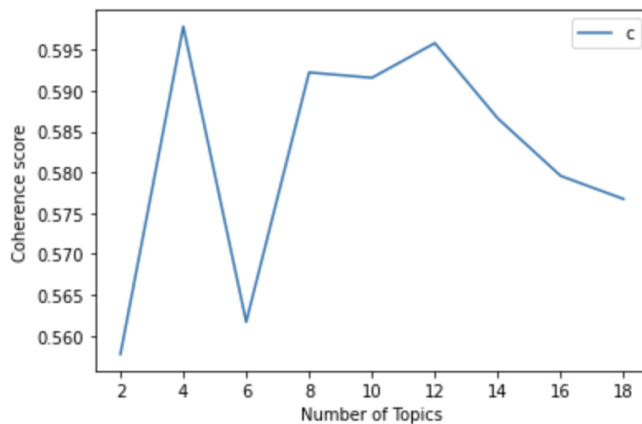
All sentiments

Num Topics = 2 has Coherence Value of 0.6069
Num Topics = 4 has Coherence Value of 0.5603
Num Topics = 6 has Coherence Value of 0.5672
Num Topics = 8 has Coherence Value of 0.5742
Num Topics = 10 has Coherence Value of 0.5369
Num Topics = 12 has Coherence Value of 0.5729
Num Topics = 14 has Coherence Value of 0.5745
Num Topics = 16 has Coherence Value of 0.5662
Num Topics = 18 has Coherence Value of 0.551



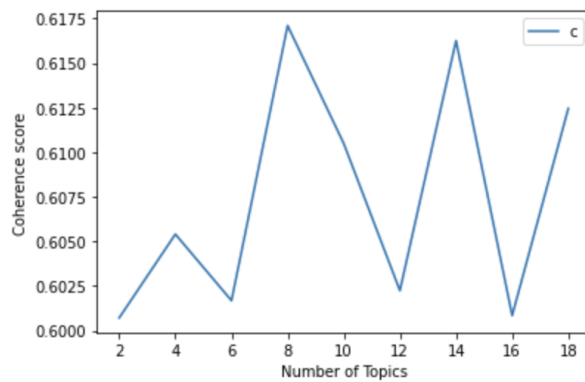
Positive sentiments

Num Topics = 2 has Coherence Value of 0.5577
Num Topics = 4 has Coherence Value of 0.5978
Num Topics = 6 has Coherence Value of 0.5617
Num Topics = 8 has Coherence Value of 0.5922
Num Topics = 10 has Coherence Value of 0.5916
Num Topics = 12 has Coherence Value of 0.5958
Num Topics = 14 has Coherence Value of 0.5866
Num Topics = 16 has Coherence Value of 0.5796
Num Topics = 18 has Coherence Value of 0.5767



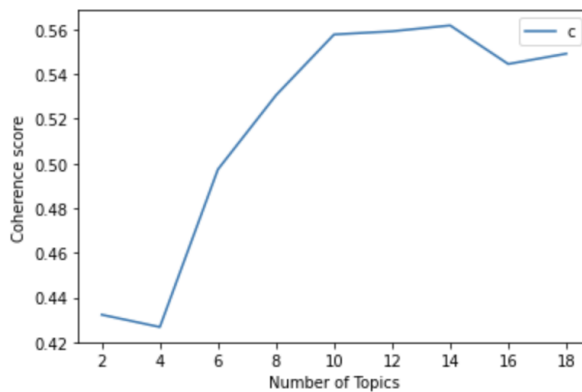
Negative sentiments

Num Topics = 2 has Coherence Value of 0.6007
Num Topics = 4 has Coherence Value of 0.6054
Num Topics = 6 has Coherence Value of 0.6017
Num Topics = 8 has Coherence Value of 0.6171
Num Topics = 10 has Coherence Value of 0.6105
Num Topics = 12 has Coherence Value of 0.6022
Num Topics = 14 has Coherence Value of 0.6162
Num Topics = 16 has Coherence Value of 0.6008
Num Topics = 18 has Coherence Value of 0.6124



Neutral sentiments

Num Topics = 2 has Coherence Value of 0.4323
Num Topics = 4 has Coherence Value of 0.4267
Num Topics = 6 has Coherence Value of 0.4973
Num Topics = 8 has Coherence Value of 0.5306
Num Topics = 10 has Coherence Value of 0.5578
Num Topics = 12 has Coherence Value of 0.5592
Num Topics = 14 has Coherence Value of 0.5618
Num Topics = 16 has Coherence Value of 0.5446
Num Topics = 18 has Coherence Value of 0.5492



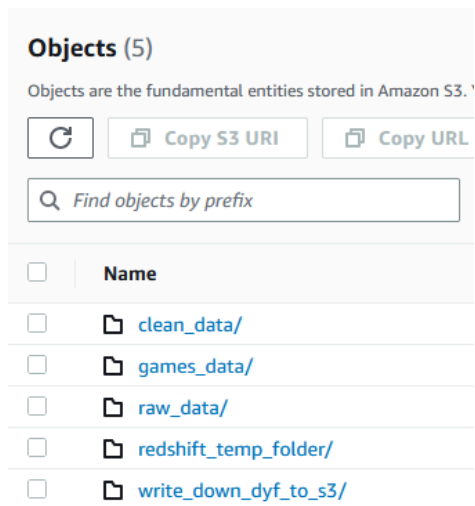
Appendix 3

Download files from drive to view pyLDavis .html file.

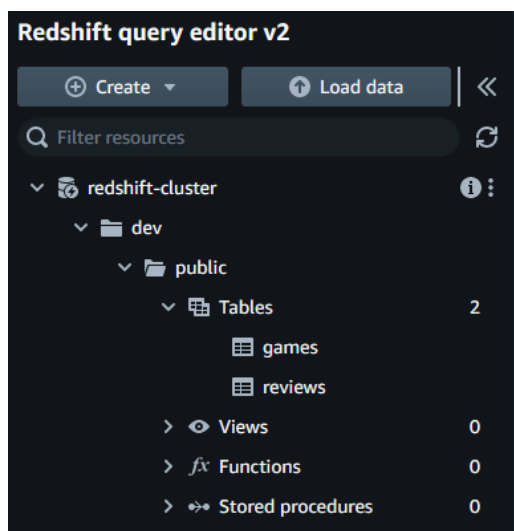
1. [pyLDavis for all sentiments](#)
2. [pyLDavis for positive sentiments](#)
3. [pyLDavis for negative sentiments](#)
4. [pyLDavis for neutral sentiments](#)

Appendix 4

AWS S3 bucket



Redshift data warehouse



AWS Glue Data Catalog

Schema						
Partitions						
Indexes						
<div>Schema (5)<div>View and manage the table schema.</div><div><div>Q Filter schemas</div><div>< 1 > ⚙</div></div></div>						
<div><div>Edit schema as JSON</div><div>Edit schema</div></div>						
#	Column name	Data type	Partition key	Comment		
1	app_id	bigint	-	-		
2	app_name	string	-	-		
3	review_text	string	-	-		
4	review_score	bigint	-	-		
5	review_votes	bigint	-	-		

AWS Crawler used to retrieve data from S3 and build data catalog in Glue

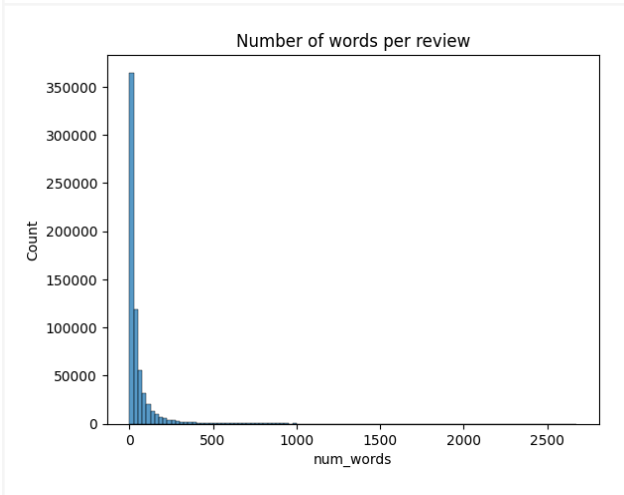
AWS Glue > Crawlers > reviews_cralwer			
reviews_cralwer <div>Last updated (UTC) February 26, 2023 at 09:11:07</div> <div><div>↺</div><div>Run crawler</div><div>Edit</div><div>Delete</div></div>			
<div>Crawler properties</div> <div><div><div>Name</div><div>reviews_cralwer</div></div><div><div>IAM role</div><div>GlueNotebookRole ↗</div></div><div><div>State</div><div>READY</div></div><div><div>Description</div><div>-</div></div></div> <div><div><div>Security configuration</div><div>-</div></div><div><div>Lake Formation configuration</div><div>-</div></div></div> <div><div>▶ Advanced settings</div></div>			
<div>Crawler runs</div> <div>Schedule Glue tables Classifiers Tags</div>			
<div>Crawler runs (1)<div>The list of crawler runs for this crawler.</div><div><div>Q Filter data</div><div>📅 Filter by a date and time range</div><div>< 1 > ⚙</div></div></div> <div><div><div>Start time (UTC)</div><div>▲</div><div>End time (UTC)</div><div>▼</div><div>Current/last duration</div><div>▼</div><div>Status</div><div>▼</div><div>DPU hours</div><div>▼</div><div>Table changes</div><div>▼</div></div><div><div><div>○</div><div>January 29, 2023 at 02:26:52</div><div>January 29, 2023 at 02:27:35</div><div>42 s</div><div>✔ Completed</div><div>0.041</div><div>1 table change, 0 partition changes</div></div></div></div>			

EC2 Instance

Instance ID	Instance state	Instance type	Status check
	✔ Running 🔍	c5a.4xlarge	✔ 2/2 checks passed

Appendix 5

Word distribution across entire dataset



Word distribution after filtering for reviews with words between 4 and 197

