The Problem:

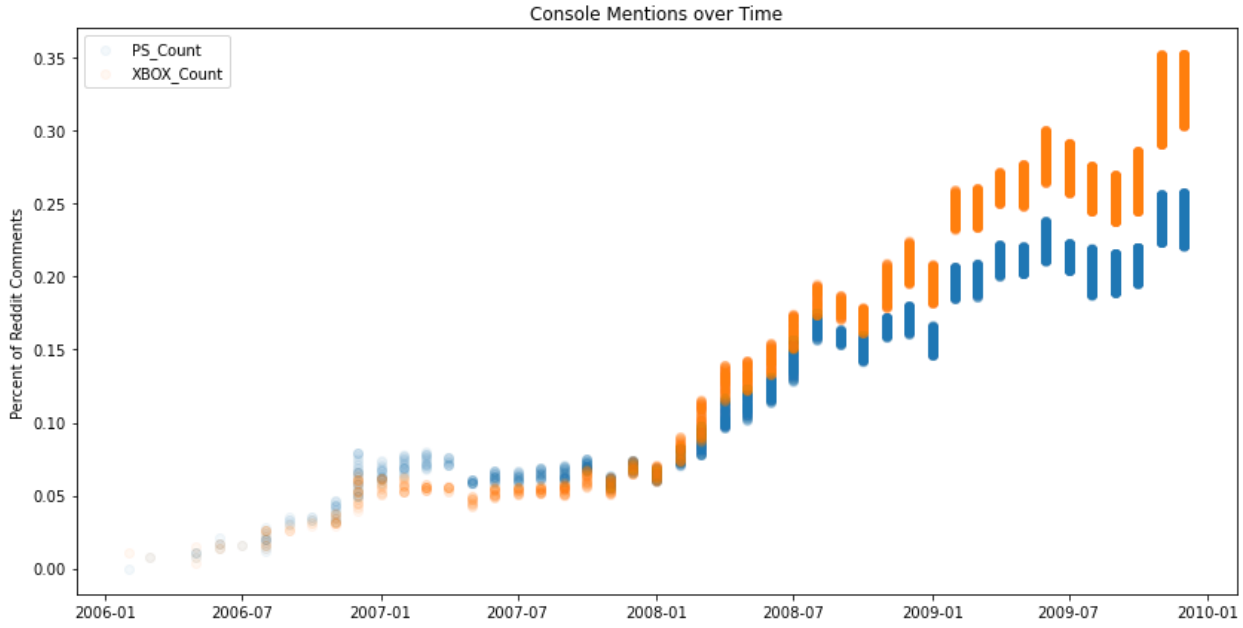
With information being spread faster than ever as different forms of social media gain popularity, it's important for companies to be able to monitor public opinion of their products and services, even on websites that aren't actively product review platforms. Keeping this in mind, I've decided to perform machine learning on social media data to try and gauge public opinion on two competing products, in this case the gaming consoles XBOX and Playstation.

Approach:

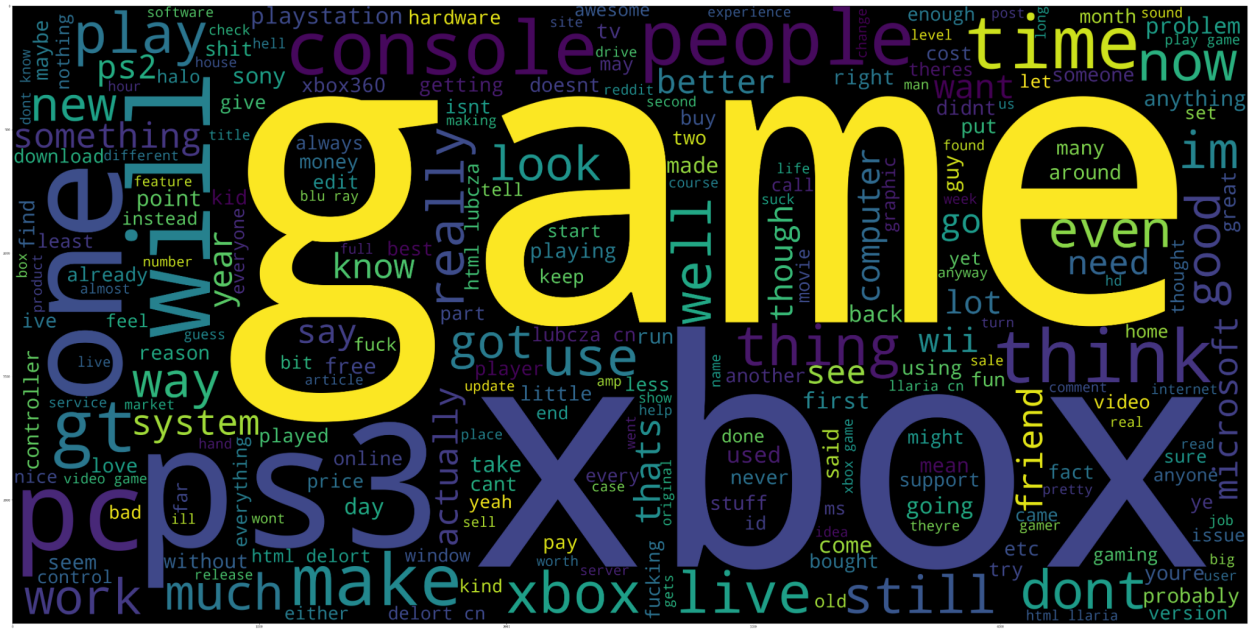
I found a large, open source data set of Reddit comments that was created by reddit users u/Dewarim and u/Stuck_in_the_Matrix. This dataset contained not only the comments themselves, but also included data such as score of the post, subreddit, and author. After gathering a few years' worth of posts I inspected the data, but it was surprisingly clean of null values already. As a result, most of my preparations involved picking which columns to keep and filtering out posts that did not contain information on my selected products, resulting in me keeping about 15,000 posts to work with.

Exploring the Data:

During the EDA step of this project, I examined the relationships between different variables, but noticed a distinct lack of correlations, partially due to the number of categorical columns such as subreddit and body, as well as boolean columns such as PS and XBOX. The only exceptions were variables related to time, such as the year the post was published, the running total of posts mentioning PS or XBOX respectively, and the number of posts from that year. It was interesting to find out that these products started becoming a higher percentage of the posts on Reddit over the years, as seen below.



Before moving from EDA, I made a word cloud of body paragraph text from each post to get an idea of what words would play a significant role in predicting sentiment. As you can see below, a lot of the words within it don't add any meaningful information to decipher emotion from a post, so I made sure to handle text in a better way once I was ready to preprocess.

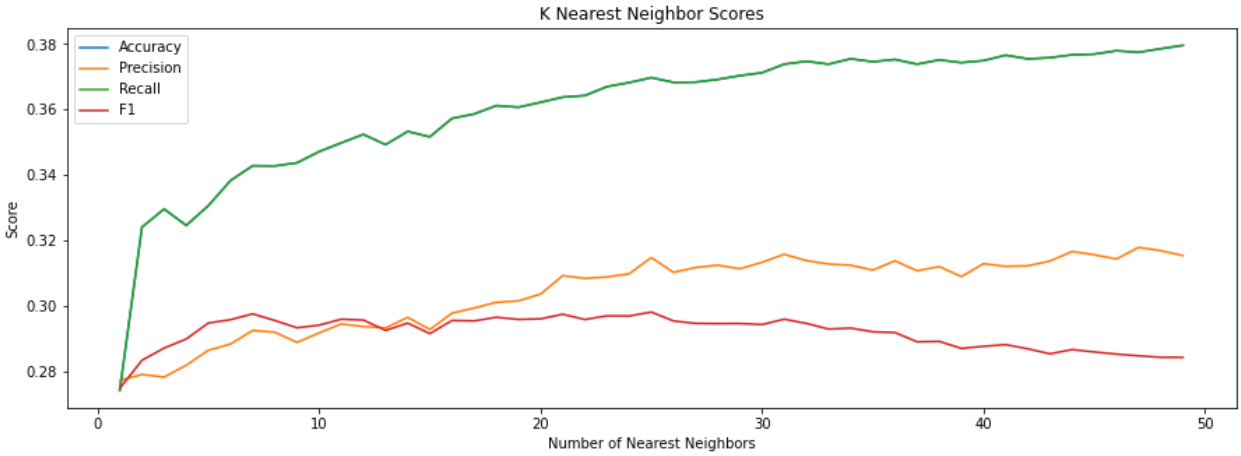


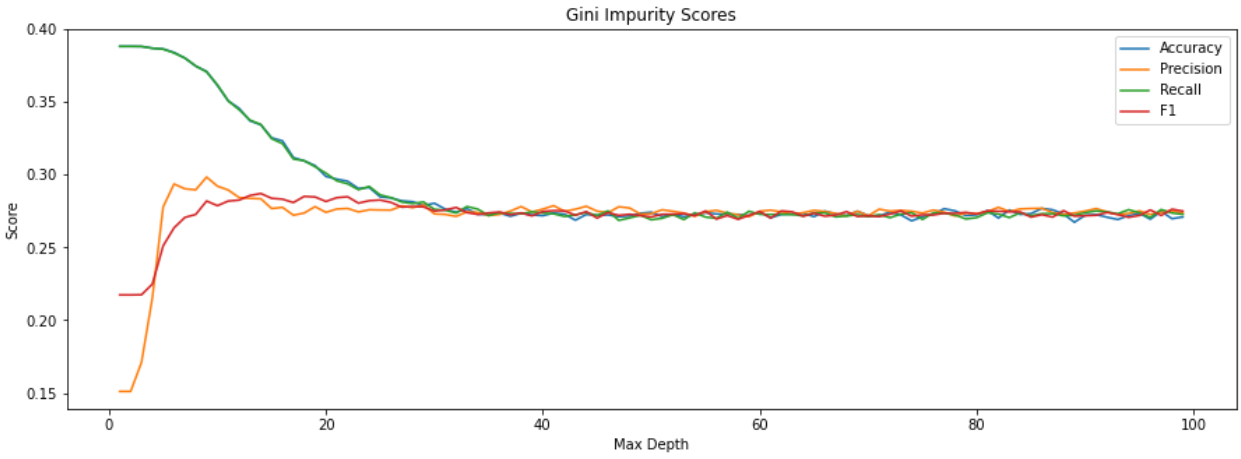
Preprocessing:

One thing that made this project tricky is that a lot of my critical information is derived from text data, rather than numbers. To start off, I needed a prediction variable to run as my y value for training and testing models, so I ran each body column through a Bert sentiment analysis pipeline. After that, I created dummies for the subreddit column and dropped columns that I had only added in EDA for improving my graphs visually. Lastly, I attempted to run my body paragraphs through a TF-IDF program that essentially takes tokens of the paragraphs and grants a score based on how frequently it appears in that paragraph compared to how frequently it appears in the entire database. Unfortunately, this became non-functional when trying to merge it with the rest of my data, as it added too many columns to fit into memory, the results of which are in the file UnguidedCapstonePreprocessingAlternate. Instead, I ended up writing my own TF-IDF program to help me find the most important tokens to score, and create columns for those instead of for every token.

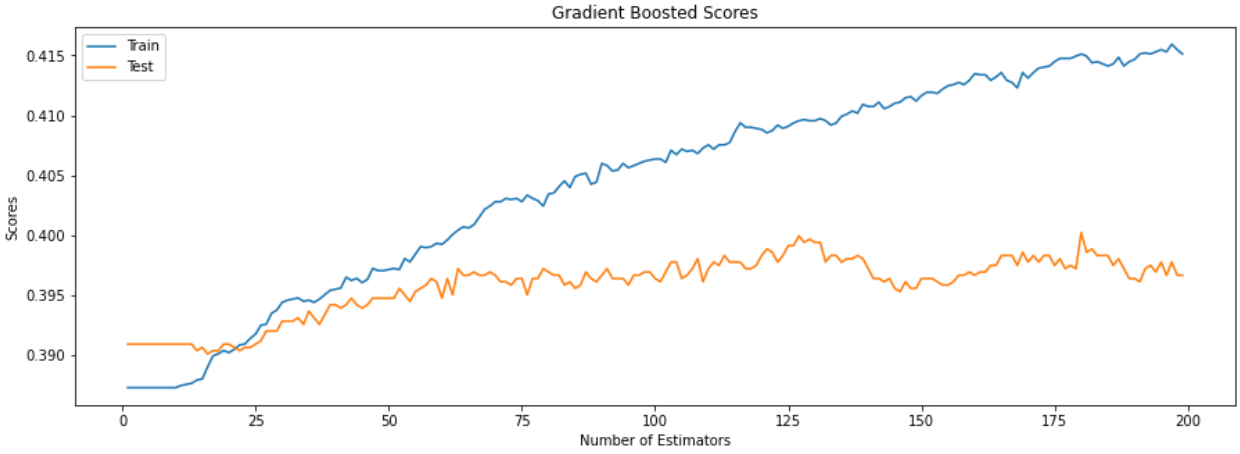
The modeling for this dataset went about as well as it could for uncorrelated data. I attempted using both a K Nearest Neighbor classifier and a Gini-Impurity Tree and tested a range

of hyperparameters for each, using 5-fold cross validation to maximize my chances of getting a decent prediction model. The two pictures below reflect the results of each model. Notice that both of them get pretty close to a maximum score of 0.38 in recall. This is because the Bert sentiment analysis I used for my y value grants a score from 1 to 5 stars, and the score of 1 occurs about 38% of the time, leading models that take a lot of neighborly input like KNN to predict 1 on nearly every post, and Gini getting its highest recall score at max_depth=1, where it can only choose one of two scores. In some of my testing I did reduce the number of posts with bert_label equal to 1, but they managed to perform even worse than before. F1 and Accuracy in the below graphs seemed to be calculated in a similar manner, and as a result overlap.





Since those two models didn't work very well, I ended up using a Gradient Boosting Classifier to make a model that can learn from previous mistakes, ideally granting a buffer against only voting for 1. It performed marginally better than the other two models at around 39.39% after it had been run through a Bayesian optimizer to find the best hyperparameters. Below is an example of this model with one of its hyperparameters on the x axis.



Findings:

With low correlations between given data and a non-optimal use of TF-IDF for scoring tokens, the model is not able to reliably predict sentiment. To see if the models were just overfitting for y=1, I tried cutting down the instances of that y value down to 3,000, which is closer to the other values while still leading, and then retraining all of the models on the smaller data set. This ultimately led to worse performance for all 3 models, with gradient boost still performing the best. Interestingly the precision of each model actually went up to about 0.6 in this experiment, but the recall, accuracy, and F1 dropped to less than 0.2. This indicated to me that given the columns I had as selector variables, I had not chosen a combination that would result in a predictable model.

Ideas for future:

Since I was unable to build a very accurate model, the main idea I have for the future is finding a less memory-intensive way of adding a TF-IDF to the dataset so that I have more potentially useful predictor columns. One thing that might help with this is finding ways to reduce the dimensionality of the TF-IDF, either by removing columns with little-to-no value or finding a way to merge columns while limiting information loss. Another useful tool to add is a pipeline that can properly clean and modify the data to work as a general-case Reddit sentiment analysis, rather than just for gaming consoles.

Client Recommendations:

There are a few things a client could do with the code and models presented here. For example, using the tokenizer and word cloud generator to see what words are most frequently being used when people talk about their products. Another possibility is to get a general idea about interest in a particular product after it has been announced to the public, to see what features people are most looking forward to. Finally, the client could run this model on both

their product and their competition to see which products are getting better received by the public.