Gary Gau Ku

**Video Game Review Sentiment Analysis**

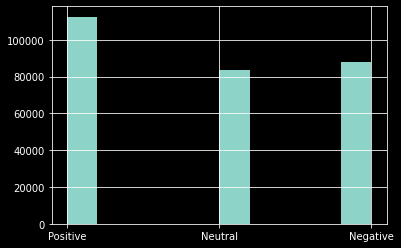
**Problem Statement**

Sentiment analysis is a tool that can potentially play a huge part in connecting a company to their audience/customer base. This is especially true for businesses whose products are non-commodities and sales are dependent on the perceived value of their products, as is the case with videogame developers. Normally, in order to get an understanding of general opinions towards a game without having to rely on a numerical value of a review score a reader would have to extract meaning from a tremendous volume of text. But what if there was a way to figure out what people think without having to read through piles of text? This would allow sentiments and feedback to be extracted from many more sources than numerically scored reviews, sources such as forum posts, public chats, social media comments, and much more. Thus, in this exercise I will attempt to apply the principles of sentiment analysis to several comments of games on Metacritic to see if a players sentiment towards a game can be predicted based on what they say about it!

(Link to the data set: <https://www.kaggle.com/dahlia25/metacritic-video-game-comments?select=metacritic_game_user_comments.csv>)

**Data Cleaning**

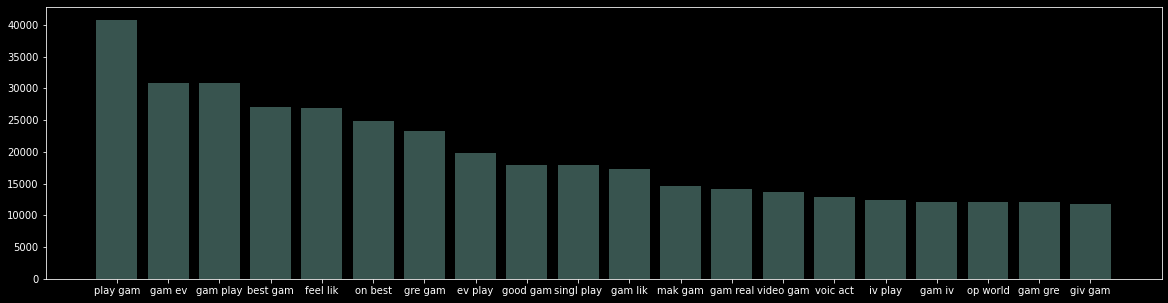
The original data set was taken from Kaggle and contained 283,983 rows and 6 columns with no missing values. The variable of interest, ‘Userscore’, was standardized then converted to the classes [‘Negative’, ‘Neutral’, ‘Positive’] using a lambda function. This results in a balanced dataset with similar numbers of rows in each class.



The ‘Comment’ column was the column that contained all of the text to be analyzed. The text was processed by removing punctuation and stopwords, stemming words to their root words, and vectorizing each word to individual frequency columns resulting in a dataframe with over 400,000 columns.

**Data Exploration**

Exploring the frequency of the top 20 bigrams present in the data yields some interesting observations that may be worth further investigation in a future project. It seems that the games that were talked about the most were single player games and open world games, and that the aspects of the games that were commented on the most were gameplay and voice acting.



**Dimension Reduction**

There were 3 methods used for dimension reduction purposes: filtering the columns by frequency, applying singular value decomposition, and applying clustering. This resulted in the creation of 3 dataframes to base models off of.

For the first method, words that appeared in less than .5% and more than 99.5% of rows were excluded resulting and a dataframe with 1,460 columns and 283,983 rows.

For the second method a dataframe with 6 columns and 283,983 rows was extracted with singular value decomposition.

For the third method the words were condensed into 42 cluster resulting in a dataframe with 42 columns and 283,983 rows.

**Models Applied**

After the data was cleaned and prepared for analysis, 4 types of predictive models were tested: KNN, Gaussian Naive Bayes’, Support Vector Classifiers, and a Neural Network. Each model was run with up to 3 different dataframes for each method used to reduce the dimensionality of the vectorized text data. All models were validated with a train-test split of 80/20. The results were as follows:

**KNN**

**with Frequency Filtered Dataframe**

**precision recall f1-score support**

Negative 0.614665 0.022265 0.042973 17696.000000

Neutral 0.441989 0.019141 0.036693 16718.000000

Positive 0.397803 0.985298 0.566776 22378.000000

**accuracy 0.400813 0.400813 0.400813 0.400813**

macro\_avg 0.484819 0.342235 0.215481 56792.000000

weighted\_avg 0.478383 0.400813 0.247521 56792.000000

**with SVD Dataframe**

**precision recall f1-score support**

Negative 0.604553 0.681284 0.640629 17696.000000

Neutral 0.468793 0.329764 0.387176 16718.000000

Positive 0.616142 0.690812 0.651344 22378.000000

**accuracy 0.581561 0.581561 0.581561 0.581561**

macro\_avg 0.563163 0.567287 0.559716 56792.000000

weighted\_avg 0.569155 0.581561 0.570242 56792.000000

**with Clustered Dataframe**

**precision recall f1-score support**

Negative 0.498493 0.738189 0.595112 17696.000000

Neutral 0.445102 0.197871 0.273954 16718.000000

Positive 0.593392 0.613996 0.603518 22378.000000

**accuracy 0.530198 0.530198 0.530198 0.530198**

macro\_avg 0.512329 0.516685 0.490862 56792.000000

weighted\_avg 0.520170 0.530198 0.503884 56792.000000

**Gaussian Naive Bayes’**

**with SVD Dataframe**

**precision recall f1-score support**

Negative 0.462539 0.862059 0.602048 17696.000000

Neutral 0.418567 0.247039 0.310702 16718.000000

Positive 0.698795 0.435428 0.536534 22378.000000

**accuracy 0.512907 0.512907 0.512907 0.512907**

macro\_avg 0.526634 0.514842 0.483095 56792.000000

weighted\_avg 0.542688 0.512907 0.490469 56792.000000

**with Clustered Dataframe**

**precision recall f1-score support**

Negative 0.444407 0.826910 0.578117 17696.000000

Neutral 0.387075 0.243989 0.299310 16718.000000

Positive 0.667817 0.397712 0.498530 22378.000000

**accuracy 0.486195 0.486195 0.486195 0.486195**

macro\_avg 0.499767 0.489537 0.458652 56792.000000

weighted\_avg 0.515562 0.486195 0.464684 56792.000000

**Support Vector Classifier**

**with SVD Dataframe**

**precision recall f1-score support**

Negative 0.562066 0.731295 0.635609 17696.000000

Neutral 0.483527 0.186984 0.269680 16718.000000

Positive 0.594074 0.724819 0.652966 22378.000000

**accuracy 0.568513 0.568513 0.568513 0.568513**

macro\_avg 0.546555 0.547699 0.519418 56792.000000

weighted\_avg 0.551558 0.568513 0.534729 56792.000000

**with Clustered Dataframe**

**precision recall f1-score support**

Negative 0.519477 0.746044 0.612480 17696.00000

Neutral 0.479084 0.180165 0.261856 16718.00000

Positive 0.601650 0.674591 0.636036 22378.00000

**accuracy 0.551310 0.551310 0.551310 0.55131**

macro\_avg 0.533404 0.533600 0.503457 56792.00000

weighted\_avg 0.539965 0.551310 0.518548 56792.00000

\**note: GNB and SVC models were not applied to the frequency filtered dataframe because they could not be applied to sparse dataframes*

**Neural Network**

|  |  |  |
| --- | --- | --- |
| **NN with Filtered Frequency Dataframe** | | |
|  | **Loss** | **Accuracy** |
| Epoch 1 | 0.8 | 0.64 |
| Epoch 2 | 0.68 | 0.7 |
| Epoch 3 | 0.6 | 0.74 |
| Epoch 4 | 0.52 | 0.78 |
| **Validation** | **0.74** | **0.68** |

|  |  |  |
| --- | --- | --- |
| **NN with SVD Dataframe** | | |
|  | **Loss** | **Accuracy** |
| Epoch 1 | 1.04 | 0.48 |
| Epoch 2 | 0.91 | 0.56 |
| Epoch 3 | 0.91 | 0.57 |
| Epoch 4 | 0.91 | 0.57 |
| **Validation** | **0.9** | **0.58** |

|  |  |  |
| --- | --- | --- |
| **NN with Clustered Dataframe** | | |
|  | **Loss** | **Accuracy** |
| Epoch 1 | 0.99 | 0.52 |
| Epoch 2 | 0.93 | 0.55 |
| Epoch 3 | 0.56 | 0.56 |
| Epoch 4 | 0.92 | 0.56 |
| **Validation** | **0.92** | **0.56** |

**Conclusion**

Of the models tested, with the exception of the best and worst models, the accuracy of the models tested seemed relatively similar ranging from around .48 to .57. The worst model was KNN classification trained on a frequency filtered dataframe scoring an accuracy of .40 while the best model was a neural network model trained on a frequency filtered dataframe scoring an accuracy of .68.

Overall, the best model tested during this project, with an accuracy of .68, performs significantly better than randomly guessing the classification of any given comment but is still far from reliable. For future revisions of this project, hyper parameter optimization and cross validation techniques can be used to optimize the accuracy of the models tested and additional models such as random forest can also be applied. These methods were excluded in the current project in order to meet a deadline but can potentially further the success of this project.