

Exploring the Sentiment of Steam Reviews with Naive Bayes and Support Vector Machine

Puttisan "Ninja" Mukneam

March 25, 2023

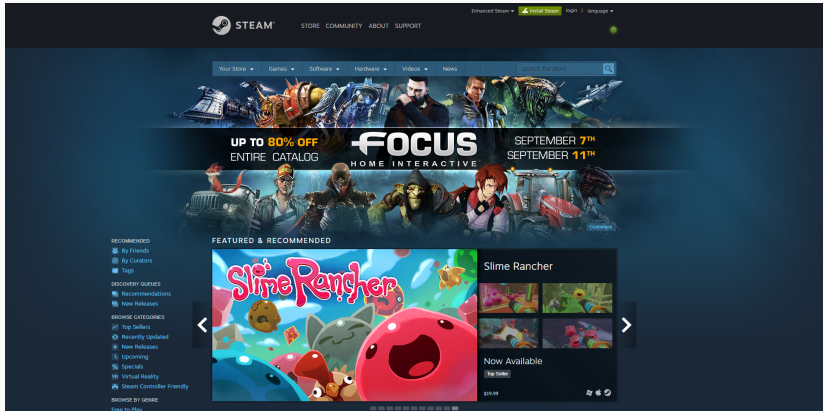
Pitzer College

Introduction

NLP: Sentiment Analysis

Sentiment analysis, also known as opinion mining, is a subfield of natural language processing (NLP) that focuses on extracting subjective information from text data. The primary goal of sentiment analysis is to determine the sentiment or emotional tone behind a series of words, which can be used to understand the attitudes, opinions, and emotions expressed by the author. Sentiment analysis is widely applied in various domains, including social media monitoring, product reviews, customer feedback analysis, and market research, among others.

- Steam platform: largest digital distribution platform for video games



Steam Review



Not Recommended

10.3 hrs on record (7.4 hrs at review time)



POSTED: JUNE 14

I am bad at the game. But instead of taking accountability and improve my skills, I will blame the game instead.

Was this review helpful?



Yes



No



Funny



Award

239 people found this review helpful

300 people found this review funny



4



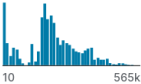


2



10

Steam Review Dataset [1]

- Approximately 6.8 million reviews.

🔗 app_id Game ID	📄 app_name Game Name	📄 review_text Review text	# review_score Review Sentiment: whether the game the review recommends the game or not.	# review_votes Review vote: whether the review was recommended by another user or not.
	<div><div>[null]3%</div><div>PAYDAY 21%</div><div>Other (6144899)96%</div></div>	<div><div>Early Access Review16%</div><div>Early Access Review0%</div><div>Other (5392421)84%</div></div>		
10	Counter-Strike	Ruined my life.	1	0
10	Counter-Strike	This will be more of a ''my experience with this game'' type of review, because saying things like ''...	1	1
10	Counter-Strike	This game saved my virginity.	1	0
10	Counter-Strike	• Do you like original games? • Do you like games that don't lag? • Do you like games you can run on...	1	0

Methods

Data Collection and Preprocessing

- First, 1M rows from the dataset.
- Only interested in review_text and review_score column
- Performed text preprocessing: Remove remove NAs, duplicates rows, trailing space, special characters, stopwords.
- Example: Before

"I hecking love Dota :3 (LoL is a terrible game btw.)"

- After

(w/ stopwords)"i hacking love dota lol is a terrible game btw"

(w/o stopwords):"hecking love dota lol terrible game btw"

Text Vectorization: Bags of Words vs TFIDF

- Importance weighting: Frequency, rarity, Importance
- Reducing the impact of stopwords

TF-IDF is the short for term frequency inverse document frequency. It is a vectorizer and TFIDF-transformed data can be used directly for the classifier. The term frequency is calculated for each term in the review as

$$TF(t, d) = \frac{\text{number of times terms } t \text{ appears in document } d}{\text{total number of terms in document } d}$$

where t is the term and d is the document.

The Inverse Document Frequency is calculated by

$$IDF(t, d) = \log\left(\frac{\text{total number of documents } D}{\text{number of documents with the term in it}}\right)$$

After calculating TF and IDF with the two formulas above, the TFIDF is calculated by

$$TFIDF(t, d, D) = TF(t, d) \times IDF(t, D)$$

Multinomial Naive Bayes

Given a document d and a class label c , Bayes' theorem can be expressed as:

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)} \quad (1)$$

$$P(d|c)P(c) = P(w_1, w_2, \dots, w_n|c)P(c) \quad (2)$$

where w_1, w_2, \dots, w_n are the words in the document.

Multinomial Naive Bayes continue

$$P(w_1, w_2, \dots, w_n | c) P(c) = P(c) \prod_{i=1}^n P(w_i | c) \quad (3)$$

To classify a document, we choose the class c^* with the highest probability:

$$c^* = \arg \max_c P(c) \prod_{i=1}^n P(w_i | c) \quad (4)$$

Laplacian Smoothing

In practice, some words in the test data may not be present in the training data, leading to a zero probability for $P(w_i|c)$, which can cause issues in classification. To address this, Laplacian smoothing (also known as additive smoothing) is applied, which assigns a small non-zero probability to unseen words.

Given a word w_i and a class c , the smoothed probability is calculated as:

$$P(w_i|c) = \frac{f_{w_i,c} + \alpha}{\sum_{w \in V} (f_{w,c} + \alpha)} \quad (5)$$

where $f_{w_i,c}$ is the frequency of word w_i in class c , V is the vocabulary, and α is the smoothing parameter. A common choice for α is 1 (Laplace smoothing) or values between 0 and 1 (Lidstone smoothing).

Linear SVM

Goal: to find the optimal hyperplane that separates the data points of different classes with the maximum margin

The decision function is a linear combination of the input features:

$$f(x) = w^T x + b \quad (6)$$

where w is the weight vector, x is the input feature vector, and b is the bias term.

The goal is to minimize the following objective function:

$$\frac{1}{2}|w|^2 + C \sum_{i=1}^n \xi_i \quad (7)$$

subject to the some constrain function.

Results and Discussion

Model Comparison MNB

We trained the Multinomial Naive Bayes classifier using various parameters for Laplacian smoothing. The MultinomialNB class from the scikit-learn library was utilized for this purpose. The results of the training process are presented below:

Smoothing Parameter (α)	Mean Squared Error	Accuracy	F-1 Score
0.001	0.53193	86.7018	0.92718
0.01	0.53193	86.7018	0.92718
0.1	0.53196	86.7011	0.92718
0.5	0.53211	86.6972	0.92716
1	0.53227	86.6933	0.92714
2	0.53248	86.6879	0.92712
3	0.53273	86.6818	0.92709

Table 1: Multinomial Naive Bayes performance for different Laplacian smoothing parameters

Smoothing Parameter (α)	Mean Squared Error	Accuracy	F-1 Score
0.001	0.49789	87.5528	0.93152
0.01	0.49789	87.5528	0.93152
0.1	0.49789	87.5528	0.93152
0.5	0.49814	87.5466	0.93149
1	0.49826	87.5435	0.93148
2	0.49854	87.5366	0.93144
3	0.49881	87.5296	0.93141

Table 2: Multinomial Naive Bayes performance for different Laplacian smoothing parameters (removed stopwords)

Model Comparison SVM

We trained a Linear Support Vector Machine (SVM) classifier with various regularization parameter C values. The implementation was performed using the LinearSVC function from the scikit-learn library. The results of the training process are presented below:

Regularization Parameter (C)	Mean Squared Error	Accuracy	F-1 Score
0.001	0.50156	87.461	0.93096
0.01	0.40936	89.766	0.94211
0.1	0.39620	90.095	0.94355
1	0.39531	90.117	0.94362
10	0.39534	90.117	0.94361
100	0.39528	90.118	0.94362
1000	0.39528	90.118	0.94362

Table 3: Results for Linear SVM with various regularization parameter C values

Regularization Parameter (C)	Mean Squared Error	Accuracy	F-1 Score
0.001	0.48479	87.880	0.93323
0.01	0.41010	89.747	0.94216
0.1	0.40291	89.927	0.94287
1	0.40238	89.940	0.94290
10	0.40235	89.941	0.94290
100	0.40235	89.941	0.94290
1000	0.40238	89.940	0.94290

Table 4: Results for Linear SVM with various regularization parameter C values (removed stopwords)

Testing

we decided to pick the Linear SVM with $C=100$ as the best model based on the previous results. We then used this model to test with our testing dataset. Here is the result:

Mean Squared Error	Accuracy	F-1 Score
0.40112	89.97176	0.94282

Table 5: Testing SVM with regularization parameter $C = 100$

Mean Squared Error	Accuracy	F-1 Score
0.40473	89.88157	0.94251

Table 6: Testing SVM with regularization parameter $C = 100$ (removed stopwords)

Future Work

- Deep Learning Algorithm: CNN, BERT, etc.
- Balancing the training and testing data set
- Try different SVM kernel
- Game Recommending system
- Hidden Markov Model