Final Project
Steam Game Reviews

Kevin Hansen, Ivan Pan
IST-664 Natural Language Processing

Introduction

The digital gaming industry is booming, generating significant revenue and capturing a large portion of consumers' time. In the United States, the industry is valued at over $83 billion, with household spending on video games increasing by 36.7% in 2020. Game developers now earn revenue not only from initial game sales but also from in-game purchases and subscription services. For example, Activision Blizzard made $5.9 billion from in-game purchases and subscriptions in 2022, accounting for 79% of their total revenue. Understanding the factors that influence consumer engagement and playtime is crucial for game developers aiming to maximize post-purchase revenues.

Various game elements, such as colors, shapes, lights, rewards, and sound effects, can enhance a game’s addictiveness. Additionally, the game's content, social interaction, immersion, and achievement importance play critical roles in shaping player engagement and time spent playing. These modifications are not limited to gaming but can also be observed in other software products like social media. While internal factors are controlled by the producing firm, external factors, such as product reviews, also need to be examined for a comprehensive understanding of what influences usage. Although product reviews are well-accepted as influential for potential consumers pre-purchase, the extent to which they affect existing consumers post-purchase, specifically with video game usage, warrants further investigation.

This project aims to enhance datasets for machine learning-based text classification, specifically for game review recommendations, by leveraging various natural language processing (NLP) features. Additionally, the project will employ advanced recurrent neural network (RNN) models to perform text classification through sentiment analysis. These insights will enable users to scrape game reviews and determine whether they are net positive or negative based on the review text.

Problem statement

Can we predict from a game review whether the video game is recommended or not?

Dataset

This dataset contains over 990,000 rows of data scraped from the Steam platform, focusing on game reviews, rankings, and game-related information across various genres. The data was collected from the top 40 games in sales, revenue, and reviews within six core genres on Steam. The dataset includes 242 games for player reviews and 290 games for genre rankings and descriptions. The steam game reviews file contains the following columns seen in Figure 1.

steam_game_reviews.csv:

This file contains player reviews for the games.

Columns:

- review: The content of the player's review
- hours_played: Total hours the player has spent on the game
- helpful: Number of users who found the review helpful
- funny: Number of users who found the review funny
- recommendation: Whether the player recommended or did not recommend the game
- date: Date of the review
- game_name: Name of the game being reviewed
- username: Username of the player who wrote the review

Figure 1. Steam game review file column descriptions

Exploratory data analysis

The exploratory data analysis performed involved mostly basic statistical findings about the data. For example Figure 2 shows the breakdown between reviews that recommend or do not recommend the video games. Clearly there is a large disparity between the two and this would need to be addressed during filtering and processing. We also showed the min/max and average of the hours each reviewer played the games, helpful score the review received, funny score the review received, and the number of reviews broken down by video game seen in Figure 3.

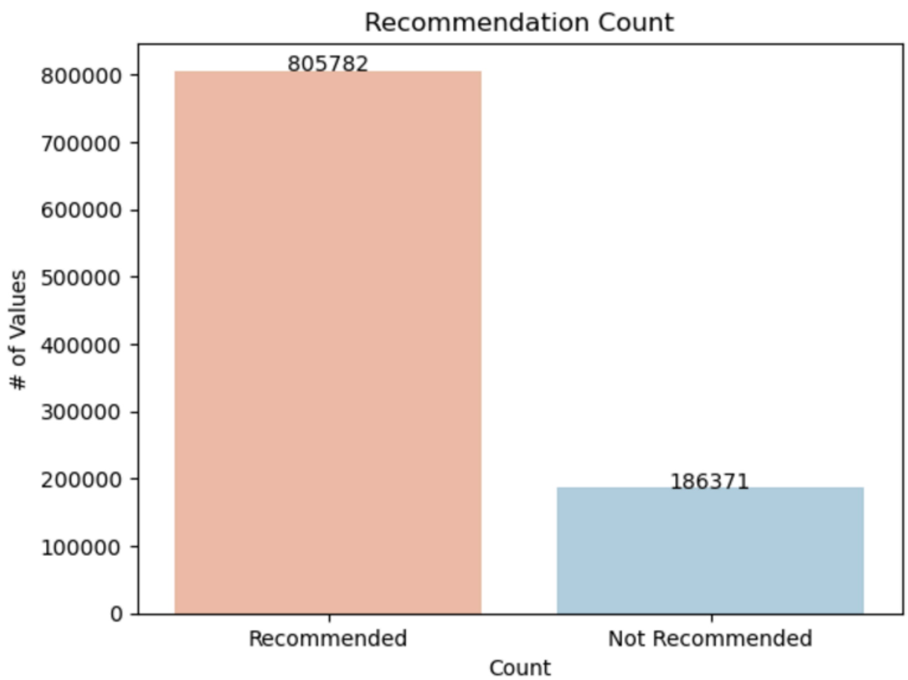


Figure 2. Count of Recommended vs. Not recommended Reviews

	Game Name	Number of Reviews
0	Apex Legends™	4999
1	Black Myth: Wukong	3367
2	Call of Duty®	5010
3	Call of Duty®: Black Ops III	4999
4	Counter-Strike 2	3386
5	Dead Rising Deluxe Remaster	1239
6	Dead by Daylight	4999
7	Destiny 2	5008
8	Diablo® IV	5006
9	Dragon's Dogma 2	4998
10	ELDEN RING	3360
11	FINAL FANTASY VII REMAKE INTERGRADE	5005
12	FINAL FANTASY XIV Online	5010
13	Grand Theft Auto V	5010
14	Gunfire Reborn	5008
15	LOCKDOWN Protocol	2568
16	Left 4 Dead 2	5010
17	Lies of P	4993
18	NARAKA: BLADEPOINT	5000
19	Noita	5009
20	Once Human	4993
21	Party Animals	4989
22	Portal 2	3333
23	Red Dead Redemption 2	5010
24	Risk of Rain 2	5009
25	Sea of Thieves: 2024 Edition	5000
26	Team Fortress 2	5010
27	Terraria	5010
28	The Crew™ 2	5008
29	The First Descendant	5006
30	The Forest	5000
31	Tom Clancy's Rainbow Six® Siege	5010
32	Total War: WARHAMMER III	4999
33	Wallpaper Engine	5010
34	Warframe	5010
35	Warhammer 40,000: Space Marine 2	3400

Figure 3.Number of reviews per game

Last part of the exploratory data analysis involved creating word clouds and outputting basic statistics of the “Recommended” reviews (Figure 4) and “Not Recommended” reviews (Figure 5). Since there is such a large difference in the number of reviews in each category it is difficult to make direct interpretations of the data between the two categories. However, it does make sense that the negative reviews are longer, typically this is the pattern seen across all industries. Negative reviews are usually longer because people take more time to express their frustrations and go out of their way to complain in hopes of seeing change.

Figure 4. Recommended Reviews Word Cloud

Figure 5. Not Recommended Reviews word cloud

Data preparation

Pre-processing steps first involved removing all columns from the dataset except for the review and recommendation columns since those are the focus of the project. Next was the removal of any row that had a missing or null value in either column. Recommended column was then converted to a binary label with ‘0’ being ‘Not Recommended’ and ‘1’ equals ‘Recommended’. Then the dataset was balanced at different values depending on computational efficiency of some of the test features and spacy pre-processing on the reviews text was performed to better tokenize social media verbiage before using sklearn’s CountVectorizer for tokenization. For example, the RNN used 50,000 recommended reviews and 50,000 not recommended reviews while the Bag-Of-Words feature used 200,000 total reviews.

Feature sets

Bag-Of-Words

Bag-of-words (BOW) or unigram features were extracted using sklearn's CountVectorizer, including tokenization, frequency counts, filtering tokens appearing in at least 5 reviews, removing English stopwords, and converting words to lowercase. The 2000 most common tokens were selected as the baseline feature set for comparison.

```
Bag-of-Words DataFrame shape: (200000, 2000)
```

POS features

POS features were extracted using NLTK's default POS tagger and the Stanford tagger. A custom function counted nouns, verbs, adjectives, and adverbs for feature representation. Sample of the dataset below:

```
print(POS_featuresDF.head())
```

	nouns	verbs	adjectives	adverbs
0	48	40	13	15
1	4	7	2	2
2	26	16	8	6
3	3	3	1	2
4	42	2	3	1

Emoji features

Emoji features were extracted using the emoji library, with a custom function identifying and counting emoji labels in the text. A total of 638 emoji features were extracted, with the top 5 most used emojis listed below:

```
:heart_suit:                125097.0
:check_box_with_check:        6300.0
:black_square_button:         2346.0
:trade_mark:                   732.0
:white_square_button:          634.0
```

Document stats features

Document statistics were extracted using a custom function to calculate the percentage of capital letters, numeric characters, and non-alphanumeric characters, as well as the text length for each review.

```
print(percentages_featuresDF.head())
```

```
   capital_percentage  number_percentage  non_alphanumeric_percentage  \
0            0.052246           0.019248                     0.235564
1            0.057692           0.000000                     0.250000
2            0.032720           0.022495                     0.249489
3            0.012821           0.000000                     0.243590
4            0.000000           0.011331                     0.144476

   text_len
0      1091
1       104
2       489
3        78
4       353
```

Named entity recognition features

Named entity recognition (NER) features were extracted using spaCy's NER pipeline, with a custom function identifying entity labels and their counts. A total of 18 NER features were extracted.

```
print(NER_featuresDF.head())

   CARDINAL_count  DATE_count  EVENT_count  FAC_count  GPE_count  \
0               2           2            0          0          2
1               0           0            0          0          0
2               1           1            0          0          0
3               0           0            0          0          0
4               1           0            0          0          0

   LANGUAGE_count  LAW_count  LOC_count  MONEY_count  NORP_count  \
0               0          1          1            0           0
1               0          0          0            0           0
2               0          0          0            0           0
3               0          0          0            0           0
4               0          0          0            0           0

   ORDINAL_count  ORG_count  PERCENT_count  PERSON_count  PRODUCT_count
0              0          0              0             0              0
1              0          0              0             0              0
2              1          5              1             0              0
3              0          0              0             0              0
4              0          2              0             5              0

   QUANTITY_count  TIME_count  WORK_OF_ART_count
0               0           2                  1
1               0           0                  0
2               0           0                  0
3               0           0                  0
4               0           0                  0
```

Vader sentiments features

Sentiment features were extracted using the VADER sentiment analyzer, with a custom function calculating positive, negative, neutral, and compound sentiment scores for each text.

```
print(Vader_featuresDF.head())

     neg    neu    pos  compound
0  0.046  0.846  0.108    0.8021
1  0.134  0.866  0.000   -0.5255
2  0.107  0.786  0.108   -0.2232
3  0.302  0.621  0.078   -0.7227
4  0.000  0.930  0.070    0.5777
```

Bigram features

Bigram features were extracted using sklearn's CountVectorizer with the same parameters as the bag-of-words feature set, selecting the top 500 most common bigrams. We see bigrams such as:

- bad game

- bad bad
- awesome game
- beautiful game
- better game
- boss fights
- breaking bugs
- worth price
- worth time

RNN features

The RNN dataset was reduced to 100K, same balanced between the classifications, for performance reasons. The data was then tokenized in different ways for feature extraction and for the RNN. Tensorflow was used to tokenize the data for the RNN model. It works by splitting the input text into smaller units such as words, characters, or subwords, and then assigning each unit a unique integer identifier. By transforming text into a numerical format, the TensorFlow tokenizer enables the text to be fed into machine learning models, facilitating tasks like text classification, sentiment analysis, and machine translation.

Model evaluation

Performing text classification using various machine learning models against our feature sets we can evaluate and determine the best model and feature sets to use for predicting game review recommendations.

Feature set names and description

Feature sets	Feature Counts	Description
Bag of Words	2000	Freq counts, most common, Stopwords removed, lowercase, at least in 5 reviews
POS counts	4	NLTK default POS tagger, counts for noun, verb, adjective, adverb
Emoji counts	638	emoji library, identify emoji label and freq counts
DocStats	4	percentage of capital letters, numeric characters, non-alphanumeric, and text length
NER counts	18	SpaCy NER label and counts
Vader Sentiment scores	4	Vader library, counts for neg, neu, pos, and compound
Bigrams	500	Freq counts, most common, Stopwords removed, lowercase, at least in 5 reviews

Model selection

Multinomial Naive Bayes

Multinomial Naive Bayes (MNB) is effective for text classification because it works well with discrete features, like word counts or frequencies, assuming the features are independent. It's computationally efficient and performs well with a large vocabulary.

Using BOW we see in our cross validation, 5 folds, and our hold out test we're seeing 82% accuracy, and similar average precision, recall, and F-1 score of 82%. Making this model a very strong baseline model. As we add more feature sets we're seeing a decrease in performance, with all features (everything) the accuracies dropped to 79%. The extra features appear to introduce redundancy and the MNB suffers for it due to its probabilistic output, confusing it.

Cross validation, 5 fold	Hold out test
=== BOW === Accuracy: 0.820438 Precision: 0.822832 Recall: 0.820438 F1-Score: 0.820104	=== BOW Test Metrics === Accuracy: 0.821700 Precision: 0.824298 Recall: 0.821700 F1-Score: 0.821342
=== BOW + everything === Accuracy: 0.799444 Precision: 0.807585 Recall: 0.799444 F1-Score: 0.798108	Accuracy: 0.795600 Precision: 0.803997 Recall: 0.795600 F1-Score: 0.794179

Random Forest

Random Forest (RF) is strong in text classification due to its ability to handle high-dimensional feature spaces and its robustness against overfitting by averaging multiple decision trees. Using 150 decision trees (50 more than the default), we're seeing a slightly improved performance for BOW, 82.2%. As we include all features (everything) the improvement jumps to 83%, this is reflected in both the cross validation and hold out testing.

Cross validation, 5 fold	Hold out test

=== RF: BOW === Accuracy: 0.822169 Precision: 0.822514 Recall: 0.822169 F1-Score: 0.822121	=== RF: BOW Test Metrics === Accuracy: 0.820325 Precision: 0.820628 Recall: 0.820325 F1-Score: 0.820283
=== RF: everything === Accuracy: 0.830575 Precision: 0.831286 Recall: 0.830575 F1-Score: 0.830484	=== RF: everything test Metrics === Accuracy: 0.829600 Precision: 0.830449 Recall: 0.829600 F1-Score: 0.829490

Logistic Regression

Logistic Regression (LR) is popular for text classification because of its simplicity and efficiency in handling linear relationships between features, making it suitable for tasks with many features like text data. We are interested to find out if sentiment alone, positive or negative, is enough to determine if the product will be recommended or not. For this simple model we only use the Vader sentiment feature set. Both our cross validation and hold out test shows about 66.8% accuracy. This shows us sentiment alone cannot determine if a reviewer will recommend the game or not.

Upon further review, there are instances of positive reviews and the reviewer still refuses to recommend the game. There are glowing reviews about the game's potential to be great but it's not ready for prime time yet, these are likely early access games where the games are in alpha or beta testing and not in its final release.

Cross validation, 5 fold	Hold out test
=== LR: Vader === Accuracy: 0.667538 Precision: 0.667575 Recall: 0.667538 F1-Score: 0.667519	=== LR: Vader TEST metric === Accuracy: 0.667575 Precision: 0.667610 Recall: 0.667575 F1-Score: 0.667558

XGBoost

XGBoost (XGB) excels in text classification because of its ability to handle complex relationships, scale well with large datasets, and its advanced regularization techniques that improve accuracy and prevent overfitting. We do see a similar 82.2% same as the RF for BOW feature set, when we include everything we're seeing an improvement of 84%. 2% gain from

MNB with BOW and 1% gain from RF also with everything. XGB appears to have the best results according to our cross validation and hold out testing.

Cross validation, 5 fold	Hold out test
=== XGBoost: BOW === Accuracy: 0.822706 Precision: 0.827366 Recall: 0.822706 F1-Score: 0.822073	=== XGBoost: BOW TEST metric === Accuracy: 0.820475 Precision: 0.825657 Recall: 0.820475 F1-Score: 0.819758
=== XGBoost: everything === Accuracy: 0.839819 Precision: 0.839866 Recall: 0.839819 F1-Score: 0.839813	=== XGBoost: everything TEST metric === Accuracy: 0.837925 Precision: 0.837953 Recall: 0.837925 F1-Score: 0.837922

Recurrent Neural Network (RNN) - Long Short-Term Memory(LSTM)

RNN's are a class of artificial neural networks designed to recognize patterns in sequences of data, such as time series or natural language. They are particularly useful for tasks where the context of previous inputs significantly influences the current input, making them well-suited for sequential data processing and temporal dynamics. LSTM networks are a special kind of RNN, capable of learning long-term dependencies.

This model contains seven layers in total in the following order: Input Layer, Embedding Layer, LSTM Layer, Second Input Layer, Concatenate Layer, Dense Layer, Output Layer. The input layer was set to take in each input sequence at 100 tokens. The embedding layer converts the input tokens into dense vectors and outputs them at a fixed dimension size of 128. The LSTM layer has 128 units and includes dropout regularization at a fraction of 0.2.. Second input layer defines an additional input for the sentiment data, which is a single value (sentiment score).Concatenate layer combines the output from the LSTM layer with the sentiment input to create a single tensor for subsequent processing. This dense layer has 64 units and uses the ReLU activation function to introduce non-linearity into the model. The output layer has a single unit with a sigmoid activation function.

Results

Summary of experiments

	Model	CV 5 fold results		Hold Out	
Feature Set Name	Model	Accuracy	others	Accuracy	others
BOW	MNB	0.82044	Precision: 0.822832 Recall: 0.820438 F1-Score: 0.820104	0.8217	Precision: 0.824298 Recall: 0.821700 F1-Score: 0.821342
BOW + POS	MNB	0.81858	Precision: 0.820439 Recall: 0.818575 F1-Score: 0.818311	0.819	Precision: 0.821024 Recall: 0.819000 F1-Score: 0.818714
BOW + Emoji	MNB	0.81971	Precision: 0.821580 Recall: 0.819706 F1-Score: 0.819443	0.82008	Precision: 0.822083 Recall: 0.820075 F1-Score: 0.819794
BOW + Stats	MNB	0.82003	Precision: 0.825560 Recall: 0.820031 F1-Score: 0.819264	0.82108	Precision: 0.826879 Recall: 0.821075 F1-Score: 0.820277
BOW + NER	MNB	0.81953	Precision: 0.821778 Recall: 0.819531 F1-Score: 0.819216	0.82135	Precision: 0.823801 Recall: 0.821350 F1-Score: 0.821011
BOW + Vader	MNB	0.80588	Precision: 0.820609 Recall: 0.805875 F1-Score: 0.803619	0.8067	Precision: 0.821550 Recall: 0.806700 F1-Score: 0.804442
BOW + Bigrams	MNB	0.81479	Precision: 0.815258 Recall: 0.814787 F1-Score: 0.814718	0.81583	Precision: 0.816348 Recall: 0.815825 F1-Score: 0.815749
everything	MNB	0.79944	Precision: 0.807585 Recall: 0.799444 F1-Score: 0.798108	0.7956	Precision: 0.803997 Recall: 0.795600 F1-Score: 0.794179
everything, no BOW	MNB	0.69671	Precision: 0.707824 Recall: 0.696712 F1-Score: 0.692604	0.69355	Precision: 0.703717 Recall: 0.693550 F1-Score: 0.689678
BOW	RF	0.82217	Precision: 0.822514 Recall: 0.822169 F1-Score: 0.822121	0.82033	Precision: 0.820628 Recall: 0.820325 F1-Score: 0.820283
everything	RF	0.83058	Precision: 0.831286 Recall: 0.830575 F1-Score: 0.830484	0.8296	Precision: 0.830449 Recall: 0.829600 F1-Score: 0.829490
everything, no BOW	RF	0.79174	Precision: 0.792550 Recall: 0.791744 F1-Score: 0.791600	0.78358	Precision: 0.784511 Recall: 0.783575 F1-Score: 0.783397
everything, no BOW/Bigrams	RF	0.75906	Precision: 0.760206 Recall: 0.759056 F1-Score: 0.758790	0.78358	Precision: 0.784511 Recall: 0.783575 F1-Score: 0.783397
Vader only	LR	0.66754	Precision: 0.667575 Recall: 0.667538 F1-Score: 0.667519	0.66758	Precision: 0.667610 Recall: 0.667575 F1-Score: 0.667558
BOW	XGB	0.82271	Precision: 0.827366 Recall: 0.822706 F1-Score: 0.822073	0.82048	Precision: 0.825657 Recall: 0.820475 F1-Score: 0.819758
everything	XGB	0.83982	Precision: 0.839866 Recall: 0.839819 F1-Score: 0.839813	0.83793	Precision: 0.837953 Recall: 0.837925 F1-Score: 0.837922

Per our experiment results above. Multinomial Naive Bayes (MNB) performs best with Bag of Words (BOW), but extra features introduce redundancy, causing a performance drop. Random Forest (RF) with 150 trees and default settings benefits from these additional features, slightly outperforming MNB. Logistic Regression (LR) was tested to determine if sentiment alone could predict whether a gamer would recommend a game, but it fails, as some non-recommendations are still positive reviews. XGBoost (XGB) shows strong performance with added NLP features, achieving the best results for classical ML models at around 84%, with hold-out results matching cross-validation outcomes.

Recurrent Neural Network (RNN)

The model was trained and evaluated using a training dataset and a testing dataset. These were broken up in an 80/20 split. The model was trained over 10 epochs. The results of the training can be seen in Figure 6. As can be seen the training was going well on the training and validation sides up until epoch 7. At this point the Training accuracy was increasing and loss decreasing, but the validation accuracy was staying consistent and validation loss was rising after this point. This is evidence of possible overfitting. Figure 7 is the output results of the model predictions using the test set and the evaluation of the models prediction.

```
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 341s 269ms/step - accuracy: 0.6719 - loss: 0.6082 - val_accuracy: 0.8508 - val_loss: 0.3563
Epoch 2/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 382s 270ms/step - accuracy: 0.8623 - loss: 0.3277 - val_accuracy: 0.8734 - val_loss: 0.3059
Epoch 3/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 389s 275ms/step - accuracy: 0.8968 - loss: 0.2573 - val_accuracy: 0.8730 - val_loss: 0.3085
Epoch 4/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 365s 262ms/step - accuracy: 0.9121 - loss: 0.2235 - val_accuracy: 0.8707 - val_loss: 0.3270
Epoch 5/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 383s 263ms/step - accuracy: 0.9234 - loss: 0.1976 - val_accuracy: 0.8690 - val_loss: 0.3339
Epoch 6/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 332s 266ms/step - accuracy: 0.9337 - loss: 0.1737 - val_accuracy: 0.8654 - val_loss: 0.3591
Epoch 7/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 381s 265ms/step - accuracy: 0.9417 - loss: 0.1543 - val_accuracy: 0.8607 - val_loss: 0.3900
Epoch 8/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 394s 275ms/step - accuracy: 0.9501 - loss: 0.1355 - val_accuracy: 0.8605 - val_loss: 0.4465
Epoch 9/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 364s 261ms/step - accuracy: 0.9545 - loss: 0.1210 - val_accuracy: 0.8575 - val_loss: 0.4673
Epoch 10/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 387s 266ms/step - accuracy: 0.9587 - loss: 0.1119 - val_accuracy: 0.8554 - val_loss: 0.5146
625/625 ━━━━━━━━━━━━━━━━━━━━ 38s 61ms/step - accuracy: 0.8549 - loss: 0.5290
Test accuracy: 0.86
```

Figure 6. RNN Model training results

	precision	recall	f1-score	support
class 0	0.86	0.85	0.85	9959
class 1	0.85	0.86	0.86	10041
accuracy			0.86	20000
macro avg	0.86	0.86	0.86	20000
weighted avg	0.86	0.86	0.86	20000

Figure 7. Model prediction and evaluation

Conclusions

Text mining using bag-of-words combined with natural language processing feature engineering enhances datasets for machine learning-based text classification tasks, such as game review recommendations. These techniques transform raw text into structured data that machine learning models can effectively analyze, thereby improving the accuracy and relevance of the classification outputs.

On the other hand, advanced models like Long Short-Term Memory networks and Recurrent Neural Networks can achieve even better text classification results. However, these sophisticated models come with a trade-off: they demand significantly more computational resources and processing time. This makes them more suitable for applications where the utmost accuracy is essential, and the cost of additional resources is justified.

For applications where a 1-3% performance gain is critical, investing in NLP feature engineering and advanced LSTM/RNN models may be worthwhile. The marginal improvement in accuracy can be crucial in scenarios where even a slight edge can lead to better decision-making or user experience.

Codes and screenshots

Python Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from collections import Counter
from nltk.corpus import stopwords
import nltk
# more libraries
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.preprocessing import normalize
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVC
import emoji
import spacy
nlp = spacy.load("en_core_web_sm")
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import cross_val_predict, StratifiedKFold
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
import xgboost as xgb
from sklearn.model_selection import GridSearchCV
import torch
from transformers import DistilBertTokenizer, DistilBertForSequenceClassification, Trainer, TrainingArguments
from torch.utils.data import Dataset
import os
```

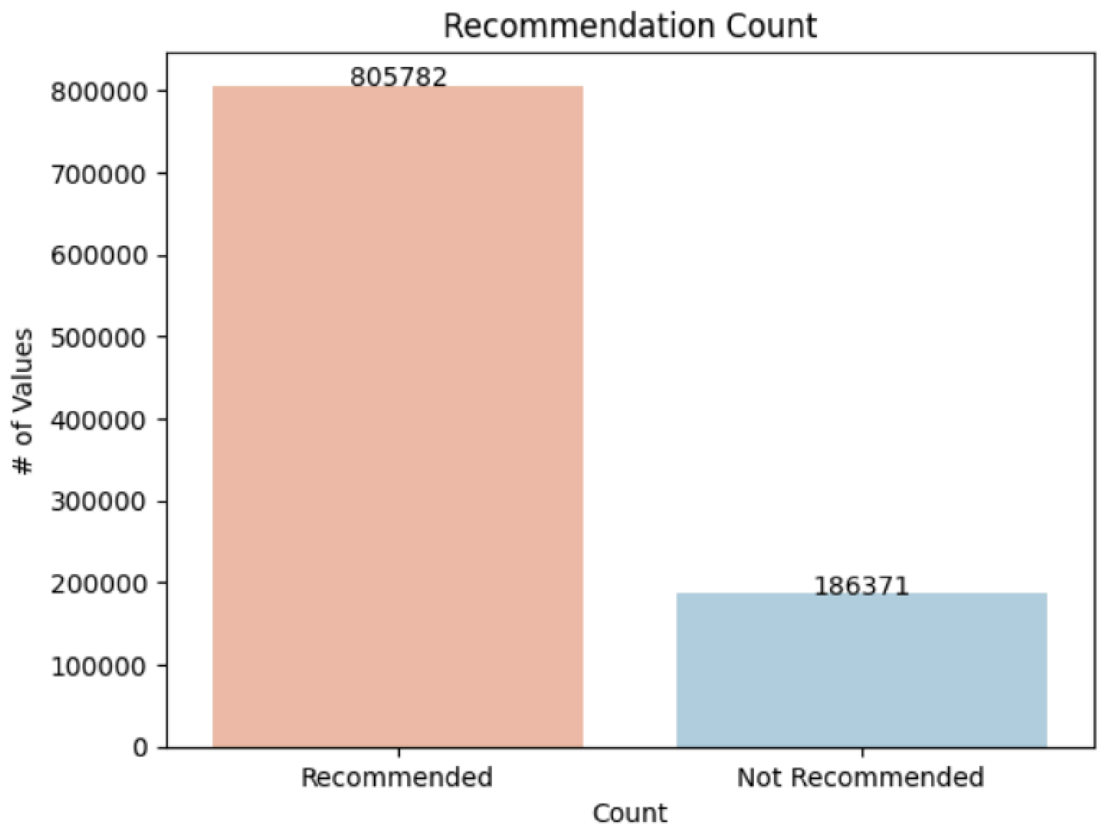
Data Prep and EDA

```
# Load the dataset
reviewDF = pd.read_csv('/content/steam_game_reviews.csv')

# Display the first few rows of the dataframe
print(reviewDF.head())

# Data frame summary
reviewDF.shape
reviewDF.columns
```

```
<ipython-input-2-09ffced2e928>:2: DtypeWarning: Columns (2,3) have mixed types. Specify dtype option on import or set low_memory=False.
  reviewDF = pd.read_csv('/content/steam_game_reviews.csv')
                                              review hours_played helpful  \
0  The game itself is also super fun. The PvP and...         39.9   1,152
1  Never cared much about Warhammer until this ga...         91.5     712
2  A salute to all the fallen battle brothers who...         43.3     492
3  this game feels like it was made in the mid 20...         16.8     661
4  Reminds me of something I've lost. A genuine g...         24.0     557

  funny recommendation          date                       game_name  \
0    13    Recommended  14 September  Warhammer 40,000: Space Marine 2
1   116    Recommended  13 September  Warhammer 40,000: Space Marine 2
2    33    Recommended  14 September  Warhammer 40,000: Space Marine 2
3    15    Recommended  14 September  Warhammer 40,000: Space Marine 2
4     4    Recommended  12 September  Warhammer 40,000: Space Marine 2

                             username
0  Sentinowl\n224 products in account
1    userpig\n248 products in account
2  Imparat0r\n112 products in account
3                      Fattest_falcon
4        Jek\n410 products in account
Index(['review', 'hours_played', 'helpful', 'funny', 'recommendation', 'date',
       'game_name', 'username'],
      dtype='object')
```

```
# Count of recommended responses
rec_counts = reviewDF['recommendation'].value_counts()
print(rec_counts)

# Recommendation Visualization
import matplotlib.pyplot as plt
import seaborn as sns

sns.barplot(x=rec_counts.index, y=rec_counts.values, palette = 'RdBu')
plt.title('Recommendation Count')
plt.xlabel('Count')
plt.ylabel('# of Values')
for i, v in enumerate(rec_counts.values):
  plt.text(i, v + 0.1, str(v), ha='center')
plt.show()
```

```
recommendation
Recommended        805782
Not Recommended    186371
Name: count, dtype: int64
```


Recommendation Count
805782
186371
800000
700000
600000
500000
400000
300000
200000
100000
0
# of Values
Recommended
Not Recommended
Count

Word Cloud and stats functions

```
# library
from wordcloud import WordCloud

# function to filter the df by status and anything else...
def filter_reviews(df, recommendation_status):
    return df[df['recommendation'] == recommendation_status]['review'].dropna().astype(str) # remove NA's


# create wordcloud function
def generate_wordcloud(reviews, label="Reviews", top_n=50):
    # Load English stopwords
    stop_words = set(stopwords.words('english'))

    # Ensure reviews are strings and combine them into one text
    text = " ".join(str(review) for review in reviews if isinstance(review, str))

    # Remove stopwords
    words = [word for word in text.split() if word.lower() not in stop_words]

    # Count word frequencies
    word_counts = Counter(words)
    top_words = dict(word_counts.most_common(top_n))

    # Generate the word cloud using only the top `n` words
    wordcloud = WordCloud(width=800, height=400, background_color='white').generate_from_frequencies(top_words)

    # Convert top words to a DataFrame for display
    top_words_df = pd.DataFrame(top_words.items(), columns=["Word", "Count"]).sort_values(by="Count", ascending=False)

    # Plot the word cloud
    plt.figure(figsize=(10, 5))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis('off')
    plt.title(f"Word Cloud for Top {top_n} Words in '{label}'", fontsize=16)
    plt.show()

    # Display the top words
    print(f"Top {top_n} Words and Their Counts (stopwords removed):")
    print(top_words_df.to_string(index=False))

# function to get descriptive statistics for reviews
def review_statistics(reviews):
    word_counts = reviews.str.split().str.len()  # Vectorized word count
    char_counts = reviews.str.len()             # Vectorized character count

    stats = {
        'Total Reviews': reviews.size,
        'Average Word Count': word_counts.mean(),
        'Median Word Count': word_counts.median(),
        'Max Word Count': word_counts.max(),
        'Min Word Count': word_counts.min(),
        'Average Character Count': char_counts.mean(),
        'Median Character Count': char_counts.median(),
        'Max Character Count': char_counts.max(),
        'Min Character Count': char_counts.min()
    }
    return stats
```

```python
print('#######  Recommended  #######')
filtered_recm_yes = filter_reviews(reviewDF, 'Recommended')
generate_wordcloud(filtered_recm_yes,label="Recommended", top_n=50)
stats_recm_yes = review_statistics(filtered_recm_yes)
#print(stats_recm_yes)
for key, value in stats_recm_yes.items():
    print(f"{key}: {value}")
```

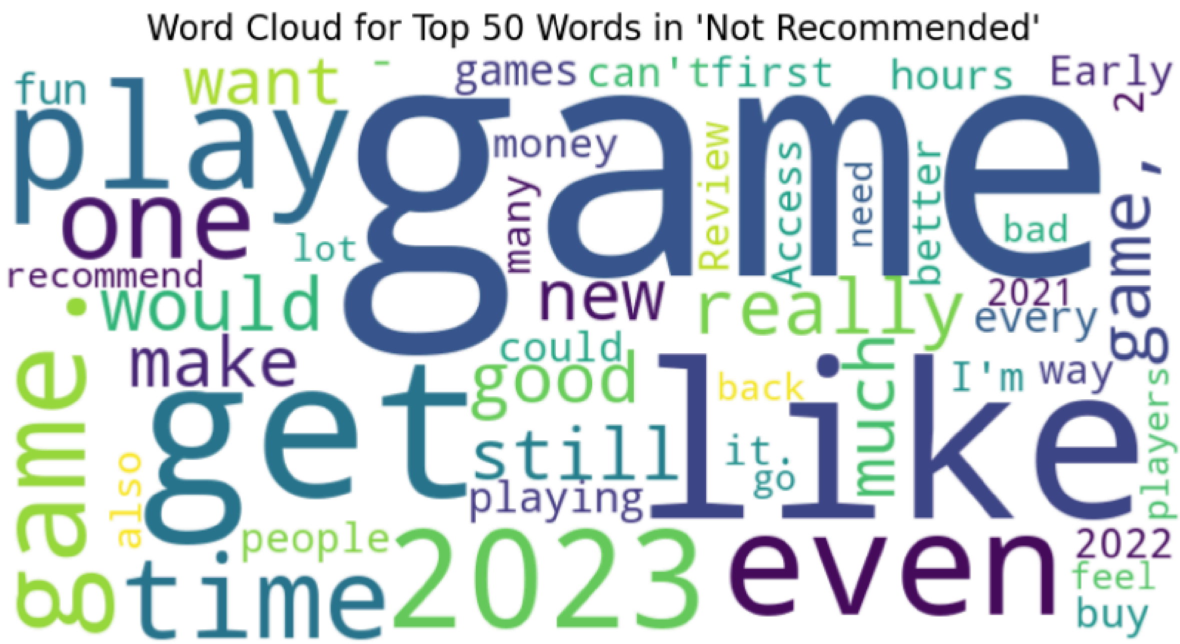
```
#######  Recommended  #######
```

Word Cloud for Top 50 Words in 'Recommended'


```
Top 50 Words and Their Counts (stopwords removed):
     Word  Count
     game 722299
     like 214727
    Early 150611
   Access 148504
   Review 147317
      get 145715
     play 140032
     2023 133863
     good 131729
      one 118027
   really 117881
      fun 115515
```

```python
print('#######  Not Recommended  #######')
filtered_recm_no = filter_reviews(reviewDF, 'Not Recommended')
generate_wordcloud(filtered_recm_no,label="Not Recommended", top_n=50)
stats_recm_no = review_statistics(filtered_recm_no)
# print(stats_recm_no)
for key, value in stats_recm_no.items():
    print(f"{key}: {value}")
```

```
#######  Not Recommended  #######
```




```
Top 50 Words and Their Counts (stopwords removed):
     Word   Count
     game  281116
     like   85017
      get   68415
     play   59553
     even   57473
     2023   51233
    game.   40929
     time   40733
      one   38743
   really   38465
    would   36244
            -----
```

Pre-processing and rebalancing

```
# Processing Data

# Make data set only contain review and recommendation columns
reviewDF = reviewDF[['review', 'recommendation']]

# Remove rows containing null values
null_counts = reviewDF.isnull().sum()
print(null_counts)
reviewDF = reviewDF.dropna()
null_counts = reviewDF.isnull().sum()
print(null_counts)
```

```
review            503
recommendation      0
dtype: int64
review            0
recommendation    0
dtype: int64
```

```python
# Convert recommendation label to binary label
from sklearn.preprocessing import LabelEncoder
labeler = LabelEncoder()
reviewDF['recommendation'] = labeler.fit_transform(reviewDF['recommendation'])

# Count number of 0's and 1's
count_class_0 = len(reviewDF[reviewDF['recommendation'] == 0])
count_class_1 = len(reviewDF[reviewDF['recommendation'] == 1])
print(f'Not Recommended - 0 = {count_class_0}')
print(f'Recommended - 1 = {count_class_1}')

# Balance the Data set to train model
# Create different class variables
class0 = reviewDF[reviewDF['recommendation'] == 0]
class1 = reviewDF[reviewDF['recommendation'] == 1]

# Undersample both class0 and class1 to 100k each
random_seed = 42
class0_under = class0.sample(n=100000, random_state=random_seed)
class1_under = class1.sample(n=100000, random_state=random_seed)
print(f'Not Recommended - 0 = {len(class0_under)}')
print(f'Recommended - 1 = {len(class1_under)}')

# Combine classes to create a new balanced dataset
reviewDF_balanced = pd.concat([class0_under, class1_under], axis=0)

# Shuffle the balanced dataset
reviewDF_balanced = reviewDF_balanced.sample(frac=1).reset_index(drop=True)
print('reviewDF_balanced shape:',reviewDF_balanced.shape)

# Create test dataset with the remaining data
remaining_class0 = class0.drop(class0_under.index)
remaining_class1 = class1.drop(class1_under.index)
reviewDF_test = pd.concat([remaining_class0, remaining_class1], axis=0)

# Shuffle the test dataset
reviewDF_test = reviewDF_test.sample(frac=1).reset_index(drop=True)
print(f"Test dataset shape: {reviewDF_test.shape}")
```

```
Not Recommended - 0 = 186335
Recommended - 1 = 805315
Not Recommended - 0 = 100000
Recommended - 1 = 100000
reviewDF_balanced shape: (200000, 2)
Test dataset shape: (791650, 2)
```

```python
# top 5 records and last 5 records
print(reviewDF_balanced.head())
print(reviewDF_balanced.tail())
```

```
                                              review  recommendation
0  Product refunded Early Access Review Not worth...               0
1  2023 Not sure if it's a cool racing game like ...               0
2  the rating formerly known as overwhelmingly po...               0
3  2019 I've got about 300 races doing both Road ...               0
4                           2014 Early Access Review /               1
                                                   review  recommendation
199995  2020 L4D2 is a fantastic game that had a vibra...               0
199996  2021 Adorable strategy game where the strategy...               0
199997  Would I recommend? No, not unless you're a tru...               0
199998  No one will read this so I'll run one mile per...               1
199999  2022 This game is awesome. Even after 7 years ...               1
```

```python
import spacy
nlp = spacy.load("en_core_web_sm")

# Bag of words using freq count and spacy
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score

# Tokenize in batches using spaCy
def preprocess_reviews(reviews):
    docs = nlp.pipe(reviews, batch_size=1000)
    return [" ".join([token.text for token in doc]) for doc in docs]

# Preprocess the 'review' column
reviewDF_balanced['processed_review'] = preprocess_reviews(reviewDF_balanced['review'])

reviewDF_balanced.head()
```

	review	recommendation	processed_review
0	2016 It's finally time to rate this game. Ulti...	0	2016 It 's finally time to rate this game . Ul...
1	Copy and Paste form previous game. Just racing...	0	Copy and Paste form previous game . Just racin...
2	2020 A very good and relaxing game ,if you lov...	0	2020 A very good and relaxing game , if you lo...
3	I like the game , but the stuttering is just t...	0	I like the game , but the stuttering is just t...
4	2022 game was so good i wrote a beatbox:ba dum...	1	2022 game was so good i wrote a beatbox : ba d...

Feature sets

```
# Use CountVectorizer on the preprocessed text, remove english stop words
vectorizer = CountVectorizer(binary=False, min_df=5, stop_words='english', max_features=2000, lowercase=True)
X = vectorizer.fit_transform(reviewDF_balanced['processed_review'])

# Convert to DataFrame for readability
bag_of_words_df = pd.DataFrame(X.toarray(), columns=vectorizer.get_feature_names_out())
print("Bag-of-Words DataFrame shape:", bag_of_words_df.shape)
```

```
Bag-of-Words DataFrame shape: (200000, 2000)
```

```
bag_of_words_df.head()
```

	000	10	100	1000	11	12	13	14	15	16	...	yeah	year	years	yes	youtube	zero	zombie	zombies	zone	LABEL
0	0	0	2	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1

5 rows × 2001 columns

```python
## create POS count features, taken from wk9 lab, modify to not need word features and take in string:
# this function takes a document list of words and returns a feature dictionary
# it runs the default pos tagger (the Stanford tagger) on the document
#   and counts 4 types of pos tags to use as features
def POS_features(text):
    # Tokenize the text into words
    words = nltk.word_tokenize(text)
    # Perform POS tagging
    tagged_words = nltk.pos_tag(words)

    # Initialize counts for each POS category
    numNoun = 0
    numVerb = 0
    numAdj = 0
    numAdverb = 0

    # Count the POS tags
    for (_, tag) in tagged_words:
        if tag.startswith('N'): numNoun += 1
        if tag.startswith('V'): numVerb += 1
        if tag.startswith('J'): numAdj += 1
        if tag.startswith('R'): numAdverb += 1

    # Create and return the features dictionary
    features = {
        'nouns': numNoun,
        'verbs': numVerb,
        'adjectives': numAdj,
        'adverbs': numAdverb
    }
    return features
```

```python
# loop thru reviewDF_balanced, "processed_review" on the POS_features and save to POS_featuresDF

# Process each row in the DataFrame
features_list = []
for _, row in reviewDF_balanced.iterrows():
    processed_review = row['processed_review']
    features = POS_features(processed_review)
    features_list.append(features)

# Create a new DataFrame to hold the features
POS_featuresDF = pd.DataFrame(features_list)

print(POS_featuresDF.head())
```

```
   nouns  verbs  adjectives  adverbs
0     48     40          13       15
1      4      7           2        2
2     26     16           8        6
3      3      3           1        2
4     42      2           3        1
```

```python
# save
#POS_featuresDF.to_csv("POS_featuresDF.csv", index=False)
```

```python
def demojize_with_count(text):
    # Extract emojis from the text using emoji.emoji_list()
    emoji_list = emoji.emoji_list(text)

    # Demojize each emoji and store the demojized version
    demojized_labels = [emoji.demojize(e['emoji']) for e in emoji_list]

    # Count the occurrences of each demojized label
    label_count = Counter(demojized_labels)

    return label_count

text = "I love Python ❤️ and coding 💻! More ❤️!"
result = demojize_with_count(text)
print(result)
```

```
Counter({':red_heart:': 2, ':laptop:': 1})
```

```python
# Loop thru function for each processed review:
processed_results = []

for index, row in reviewDF_balanced.iterrows():
    # Get the processed_review text
    processed_review_text = row["processed_review"]

    # Apply demojize_with_count function
    emoji_counts = demojize_with_count(processed_review_text)

    # Store the results as a dictionary for each record
    processed_results.append({
        "original_text": processed_review_text,
        "emoji_counts": emoji_counts
    })

# Convert processed results into a new dataframe
emji_featuresDF = pd.DataFrame(processed_results)

print(emji_featuresDF.head())
```

```
                                       original_text emoji_counts
0  2016 It 's finally time to rate this game . Ul...           {}
1  Copy and Paste form previous game . Just racin...           {}
2  2020 A very good and relaxing game , if you lo...           {}
3  I like the game , but the stuttering is just t...           {}
4  2022 game was so good i wrote a beatbox : ba d...           {}
```

```python
# Filter rows where 'emoji_counts' is populated (non-null and non-empty)
populated_records = emji_featuresDF[emji_featuresDF['emoji_counts'].notnull() &
                                    emji_featuresDF['emoji_counts'].apply(lambda x: len(x) > 0)]

# Display the top 50 populated records
top_50_populated = populated_records.head(50)

# Show the result
print(top_50_populated)
```

```
                                         original_text  \
19   Playing a Korean , looter - shooter game , i a...
22   2019 ♥ ♥ ♥ ♥ ♥ ♥ ♥ ♥ game than originally anti...
24   2019 The game is good in terms of its macro el...
74   2022 Arma 3 is one of those games that is pret...
78   2023 Early Access Review Fix your ♥ ♥ ♥ ♥ ♥ ♥ ...
84   this is the worst balanced game i 've played ,...
92           2022 I found a ♥ ♥ ♥ ♥ ♥ ♥ hacker ! ! ! !
112  2023 Early Access Review Vroom Crash jump cool...
128  2022 STAY AWAYThis game was great . Intriguing...
141  Early Access Review Originally bought this gam...
142  2021 1hp which I 'm sure everyone knows about ...
165  2019 Сейчас в сети интернета очень много спорo...
185  Pros : The best NASCAR / Road Course simulator...
189  Kept Writing Fake Memo 's Saying i ♥ ♥ ♥ ♥ ♥ ♥...
190  2018 ---{Graphics}--- ☐ You forget what realit...
205  2017 Finished Life is Strange yesterday . Peew...
224  2018 Enjoyed the game quite a bit , have nt pl...
231  The game used to be great , it still could be ...
```

```python
#show top 10 by counts emojis

emoji_columns = [col for col in NewEmoji_featuresDF.columns if col != 'original_text']  # not the original_text

# Sum the emoji counts
emoji_totals = NewEmoji_featuresDF[emoji_columns].sum()

# Sort the sums in descending order and get the top 10
top_10_emoji_counts = emoji_totals.sort_values(ascending=False).head(20)
print(top_10_emoji_counts)
```

```
:heart_suit:             125097.0
:check_box_with_check:     6300.0
:black_square_button:      2346.0
:trade_mark:                732.0
:white_square_button:       634.0
:check_mark_button:         581.0
:black_large_square:        545.0
:star:                      514.0
:thumbs_up:                 430.0
:cross_mark:                393.0
:white_large_square:        355.0
:check_mark:                298.0
:multiply:                  297.0
:red_square:                273.0
:blue_square:               266.0
:frog:                      246.0
:face_with_tears_of_joy:    190.0
:red_heart:                 178.0
:red_circle:                159.0
:hundred_points:            153.0
dtype: float64
```

```python
# function to count capital letters in text and return % of capital letters in text in relation to total text

def capital_number_non_alphanumeric_percentage(text):
    # Count the number of capital letters
    capital_count = sum(1 for char in text if char.isupper())

    # Count the number of numbers
    number_count = sum(1 for char in text if char.isdigit())

    # Count the number of non-alphanumeric characters
    non_alphanumeric_count = sum(1 for char in text if not char.isalnum())

    # Calculate the total number of characters
    total_count = len(text)

    # Avoid division by zero if the text is empty
    if total_count == 0:
        return {'capital_percentage': 0.0, 'number_percentage': 0.0, 'non_alphanumeric_percentage': 0.0}

    # Calculate the percentage of capital letters, numbers, and non-alphanumeric characters
    capital_percentage = round((capital_count / total_count),6)
    number_percentage = round((number_count / total_count),6)
    non_alphanumeric_percentage = round((non_alphanumeric_count / total_count),6)

    return {
        'capital_percentage': capital_percentage,
        'number_percentage': number_percentage,
        'non_alphanumeric_percentage': non_alphanumeric_percentage,
        'text_len': total_count
    }

# Example usage
text = "Hello 123 World! #@-=++"
result = capital_number_non_alphanumeric_percentage(text)
print(f"Percentage of capital letters: {result['capital_percentage']}")
print(f"Percentage of numbers: {result['number_percentage']}")
print(f"Percentage of non-alphanumeric characters: {result['non_alphanumeric_percentage']}")
print(f"text_len: {result['text_len']}")
```

```
Percentage of capital letters: 0.086957
Percentage of numbers: 0.130435
Percentage of non-alphanumeric characters: 0.434783
text_len: 23
```

```python
# loop thru reviewDF_balanced, "processed_review" on the capital_number_non_alphanumeric_percentage

# Process each row in the DataFrame
features_list = []
for _, row in reviewDF_balanced.iterrows():
    processed_review = row['processed_review']
    features = capital_number_non_alphanumeric_percentage(processed_review)
    features_list.append(features)

# Create a new DataFrame to hold the features
percentages_featuresDF = pd.DataFrame(features_list)

print(percentages_featuresDF.head())
```

```
   capital_percentage  number_percentage  non_alphanumeric_percentage  \
0            0.052246           0.019248                     0.235564
1            0.057692           0.000000                     0.250000
2            0.032720           0.022495                     0.249489
3            0.012821           0.000000                     0.243590
4            0.000000           0.011331                     0.144476

   text_len
0      1091
1       104
2       489
3        78
4       353
```

```python
# NER counts using spacy

text = "Apple is looking at buying U.K. startup for $1 billion in June."

# Process the text
doc = nlp(text)

# Print entities and their categories
for ent in doc.ents:
    print(f"Entity: {ent.text}, Category: {ent.label_}")
```

```
Entity: Apple, Category: ORG
Entity: U.K., Category: GPE
Entity: $1 billion, Category: MONEY
Entity: June, Category: DATE
```

```python
def extract_ner_counts(text):
    # Process the text with spaCy
    doc = nlp(text)

    # Extract NER counts
    ner_counts = Counter(ent.label_ for ent in doc.ents)

    # Define all possible categories
    categories = nlp.get_pipe("ner").labels

    # Create a feature set with 1 field per category
    features = {f"{category}_count": ner_counts.get(category, 0) for category in categories}

    return features

#text = "Apple is looking at buying a startup in the U.K. for $1 billion."
text = "Delta Flight 862, enroute to London, returned to JFK airport for emergency repairs. The engineers onsite thinks it was a miracle the plane was al
ner_feature_set = extract_ner_counts(text)
print(ner_feature_set)
```

```
{'CARDINAL_count': 0, 'DATE_count': 0, 'EVENT_count': 0, 'FAC_count': 0, 'GPE_count': 1, 'LANGUAGE_count': 0, 'LAW_count': 0, 'LOC_count': 0, 'MONEY_cou
nt': 1, 'NORP_count': 0, 'ORDINAL_count': 0, 'ORG_count': 2, 'PERCENT_count': 0, 'PERSON_count': 1, 'PRODUCT_count': 0, 'QUANTITY_count': 0, 'TIME_coun
t': 0, 'WORK_OF_ART_count': 0}
```

```python
# loop thru reviewDF_balanced, "processed_review" on the extract_ner_counts

# Process each row in the DataFrame
features_list = []
for _, row in reviewDF_balanced.iterrows():
    processed_review = row['processed_review']
    features = extract_ner_counts(processed_review)
    features_list.append(features)

# Create a new DataFrame to hold the features
NER_featuresDF = pd.DataFrame(features_list)

print(NER_featuresDF.head())
```

```
   CARDINAL_count  DATE_count  EVENT_count  FAC_count  GPE_count  \
0               2           2            0          0          2
1               0           0            0          0          0
2               1           1            0          0          0
3               0           0            0          0          0
4               1           0            0          0          0

   LANGUAGE_count  LAW_count  LOC_count  MONEY_count  NORP_count  \
0               0          1          1            0           0
1               0          0          0            0           0
2               0          0          0            0           0
3               0          0          0            0           0
4               0          0          0            0           0

   ORDINAL_count  ORG_count  PERCENT_count  PERSON_count  PRODUCT_count  \
0              0          0              0             0              0
1              0          0              0             0              0
2              1          5              1             0              0
3              0          0              0             0              0
4              0          2              0             5              0

   QUANTITY_count  TIME_count  WORK_OF_ART_count
0               0           2                  1
1               0           0                  0
2               0           0                  0
3               0           0                  0
4               0           0                  0
```

```python
# save
#NER_featuresDF.to_csv("NER_featuresDF.csv", index=False)
#LOAD FROM CSV, POS_featuresDF
NER_featuresDF = pd.read_csv("NER_featuresDF.csv")
print(NER_featuresDF.shape)
```

```
(200000, 18)
```

```python
# vader sentiment function
def get_vader_sentiment_scores(text):
    # Initialize VADER sentiment analyzer
    analyzer = SentimentIntensityAnalyzer()

    # Analyze the text
    sentiment_scores = analyzer.polarity_scores(text)

    return sentiment_scores


#text = "I absolutely love this product! It's amazing and works perfectly."
text = "I absolutely love this product, NOT! It's amazing and works perfectly if I was dumb."
sentiment_scores = get_vader_sentiment_scores(text)
print(sentiment_scores)
```

```
{'neg': 0.242, 'neu': 0.417, 'pos': 0.341, 'compound': 0.5593}
```

```python
# loop thru reviewDF_balanced, "processed_review" on the get_vader_sentiment_scores

# Process each row in the DataFrame
features_list = []
for _, row in reviewDF_balanced.iterrows():
    processed_review = row['processed_review']
    features = get_vader_sentiment_scores(processed_review)
    features_list.append(features)

# Create a new DataFrame to hold the features
Vader_featuresDF = pd.DataFrame(features_list)

print(Vader_featuresDF.head())
```

```
     neg    neu    pos  compound
0  0.046  0.846  0.108    0.8021
1  0.134  0.866  0.000   -0.5255
2  0.107  0.786  0.108   -0.2232
3  0.302  0.621  0.078   -0.7227
4  0.000  0.930  0.070    0.5777
```

```python
# save
#Vader_featuresDF.to_csv("Vader_featuresDF.csv", index=False)
#LOAD FROM CSV, POS_featuresDF
Vader_featuresDF = pd.read_csv("Vader_featuresDF.csv")
print(Vader_featuresDF.shape)
```

```
(200000, 4)
```

```
# get top 500 bigrams, most common, remove stop words, min 5x
bigram_vectorizer = CountVectorizer(binary=False, min_df=5, stop_words='english', max_features=500, lowercase=True,ngram_range=(2, 2))  # For bigrams
X = bigram_vectorizer.fit_transform(reviewDF_balanced['processed_review'])
feature_names = bigram_vectorizer.get_feature_names_out()

# Convert to DataFrame for readability
bigrams500_df = pd.DataFrame(X.toarray(), columns=bigram_vectorizer.get_feature_names_out())
print("bigram DataFrame shape:", bigrams500_df.shape)
```

```
bigram DataFrame shape: (200000, 500)
```

```
bigrams500_df.head()
```

	10 10	10 game	10 hours	10 minutes	10 years	100 hours	20 hours	2014 early	2015 early	2016 early	...	worth money	worth playing	worth price	worth time	year old	years ago	years game	years old	yes yes	youtube video
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 500 columns

```
# save
#bigrams500_df.to_csv("bigrams500_df.csv", index=False)
#LOAD FROM CSV, bigrams500_df
bigrams500_df = pd.read_csv("bigrams500_df.csv")
print(bigrams500_df.shape)
```

```
(200000, 500)
```

```
##################################### start experiments using cv5
```

Experiments

BOW

```
#split train/test - 80/20
X = bag_of_words_df.drop(columns='LABEL')
y = bag_of_words_df['LABEL']

# Perform the train/test split with stratification
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, stratify=y, random_state=42)
# Print the shapes of the resulting sets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (160000, 2000)
X_test shape: (40000, 2000)
y_train shape: (160000,)
y_test shape: (40000,)
```

```
# Initialize the classifier
mnb = MultinomialNB()
cv = StratifiedKFold(n_splits=5)

# Perform cross-validation predictions
y_pred = cross_val_predict(mnb, X_train, y_train, cv=cv)

# Calculate metrics
accuracy = accuracy_score(y_train, y_pred)
precision = precision_score(y_train, y_pred, average='macro')
recall = recall_score(y_train, y_pred, average='macro')
f1 = f1_score(y_train, y_pred, average='macro')

print('=== BOW ===')
print(f"Accuracy: {accuracy:.6f}")
print(f"Precision: {precision:.6f}")
print(f"Recall: {recall:.6f}")
print(f"F1-Score: {f1:.6f}")
```

```
=== BOW ===
Accuracy: 0.820438
Precision: 0.822832
Recall: 0.820438
F1-Score: 0.820104
```

```python
# BOW hold out
# Initialize the classifier
mnb = MultinomialNB()

# Train the best model on the entire training set
mnb.fit(X_train, y_train)

# Evaluate the best model on the training set
y_pred_train = mnb.predict(X_train)
train_accuracy = accuracy_score(y_train, y_pred_train)

# Evaluate the best model on the test set
y_pred_test = mnb.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred_test)

print("Model Accuracy (on training set):", round(train_accuracy,6))
print("Model Accuracy (on test set):", round(test_accuracy,6))

# Calculate metrics on the test set
test_accuracy = accuracy_score(y_test, y_pred_test)
test_precision = precision_score(y_test, y_pred_test, average='macro')
test_recall = recall_score(y_test, y_pred_test, average='macro')
test_f1 = f1_score(y_test, y_pred_test, average='macro')

print('=== BOW Test Metrics ===')
print(f"Accuracy: {test_accuracy:.6f}")
print(f"Precision: {test_precision:.6f}")
print(f"Recall: {test_recall:.6f}")
print(f"F1-Score: {test_f1:.6f}")
```

```
Model Accuracy (on training set): 0.821762
Model Accuracy (on test set): 0.8217
=== BOW Test Metrics ===
Accuracy: 0.821700
Precision: 0.824298
Recall: 0.821700
F1-Score: 0.821342
```

```python
# add POS_featuresDF to BOW
new_df = pd.concat([bag_of_words_df, POS_featuresDF], axis=1)
new_df.shape

#split train/test - 80/20
X = new_df.drop(columns='LABEL')
y = new_df['LABEL']

# Perform the train/test split with stratification
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, stratify=y, random_state=42)
# Print the shapes of the resulting sets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (160000, 2004)
X_test shape: (40000, 2004)
y_train shape: (160000,)
y_test shape: (40000,)
```

```python
# Initialize the classifier
mnb = MultinomialNB()
cv = StratifiedKFold(n_splits=5)

# Perform cross-validation predictions
y_pred = cross_val_predict(mnb, X_train, y_train, cv=cv)

# Calculate metrics
accuracy = accuracy_score(y_train, y_pred)
precision = precision_score(y_train, y_pred, average='macro')
recall = recall_score(y_train, y_pred, average='macro')
f1 = f1_score(y_train, y_pred, average='macro')

print('=== BOW + POS ===')
print(f"Accuracy: {accuracy:.6f}")
print(f"Precision: {precision:.6f}")
print(f"Recall: {recall:.6f}")
print(f"F1-Score: {f1:.6f}")
```

```
=== BOW + POS ===
Accuracy: 0.818575
Precision: 0.820439
Recall: 0.818575
F1-Score: 0.818311
```

```python
# BOW hold out
# Initialize the classifier
mnb = MultinomialNB()

# Train the best model on the entire training set
mnb.fit(X_train, y_train)

# Evaluate the best model on the training set
y_pred_train = mnb.predict(X_train)
train_accuracy = accuracy_score(y_train, y_pred_train)

# Evaluate the best model on the test set
y_pred_test = mnb.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred_test)

print("Model Accuracy (on training set):", round(train_accuracy,6))
print("Model Accuracy (on test set):", round(test_accuracy,6))

# Calculate metrics on the test set
test_accuracy = accuracy_score(y_test, y_pred_test)
test_precision = precision_score(y_test, y_pred_test, average='macro')
test_recall = recall_score(y_test, y_pred_test, average='macro')
test_f1 = f1_score(y_test, y_pred_test, average='macro')

print('=== BOW + POS Test Metrics ===')
print(f"Accuracy: {test_accuracy:.6f}")
print(f"Precision: {test_precision:.6f}")
print(f"Recall: {test_recall:.6f}")
print(f"F1-Score: {test_f1:.6f}")
```

```
Model Accuracy (on training set): 0.820119
Model Accuracy (on test set): 0.819
=== BOW + POS Test Metrics ===
Accuracy: 0.819000
Precision: 0.821024
Recall: 0.819000
F1-Score: 0.818714
```

```python
# add NewEmoji_featuresDF to BOW
new_df = pd.concat([bag_of_words_df, NewEmoji_featuresDF], axis=1)
# Drop the 'original_text' column
new_df = new_df.drop(columns=['original_text'])

new_df.shape

#split train/test - 80/20
X = new_df.drop(columns='LABEL')
y = new_df['LABEL']

# Perform the train/test split with stratification
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, stratify=y, random_state=42)
# Print the shapes of the resulting sets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (160000, 2638)
X_test shape: (40000, 2638)
y_train shape: (160000,)
y_test shape: (40000,)
```

```python
# Initialize the classifier
mnb = MultinomialNB()
cv = StratifiedKFold(n_splits=5)

# Perform cross-validation predictions
y_pred = cross_val_predict(mnb, X_train, y_train, cv=cv)

# Calculate metrics
accuracy = accuracy_score(y_train, y_pred)
precision = precision_score(y_train, y_pred, average='macro')
recall = recall_score(y_train, y_pred, average='macro')
f1 = f1_score(y_train, y_pred, average='macro')

print('=== BOW + Emoji ===')
print(f"Accuracy: {accuracy:.6f}")
print(f"Precision: {precision:.6f}")
print(f"Recall: {recall:.6f}")
print(f"F1-Score: {f1:.6f}")
```

```
=== BOW + Emoji ===
Accuracy: 0.819706
Precision: 0.821580
Recall: 0.819706
F1-Score: 0.819443
```

```python
# BOW hold out
# Initialize the classifier
mnb = MultinomialNB()

# Train the best model on the entire training set
mnb.fit(X_train, y_train)

# Evaluate the best model on the training set
y_pred_train = mnb.predict(X_train)
train_accuracy = accuracy_score(y_train, y_pred_train)

# Evaluate the best model on the test set
y_pred_test = mnb.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred_test)

print("Model Accuracy (on training set):", round(train_accuracy,6))
print("Model Accuracy (on test set):", round(test_accuracy,6))

# Calculate metrics on the test set
test_accuracy = accuracy_score(y_test, y_pred_test)
test_precision = precision_score(y_test, y_pred_test, average='macro')
test_recall = recall_score(y_test, y_pred_test, average='macro')
test_f1 = f1_score(y_test, y_pred_test, average='macro')

print('=== BOW + Emoji Test Metrics ===')
print(f"Accuracy: {test_accuracy:.6f}")
print(f"Precision: {test_precision:.6f}")
print(f"Recall: {test_recall:.6f}")
print(f"F1-Score: {test_f1:.6f}")
```

```
Model Accuracy (on training set): 0.821019
Model Accuracy (on test set): 0.820075
=== BOW + Emoji Test Metrics ===
Accuracy: 0.820075
Precision: 0.822083
Recall: 0.820075
F1-Score: 0.819794
```

```python
# add percentages_featuresDF to BOW
new_df = pd.concat([bag_of_words_df, percentages_featuresDF], axis=1)

new_df.shape

#split train/test - 80/20
X = new_df.drop(columns='LABEL')
y = new_df['LABEL']

# Perform the train/test split with stratification
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, stratify=y, random_state=42)
# Print the shapes of the resulting sets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (160000, 2004)
X_test shape: (40000, 2004)
y_train shape: (160000,)
y_test shape: (40000,)
```

```python
# Initialize the classifier
mnb = MultinomialNB()
cv = StratifiedKFold(n_splits=5)

# Perform cross-validation predictions
y_pred = cross_val_predict(mnb, X_train, y_train, cv=cv)

# Calculate metrics
accuracy = accuracy_score(y_train, y_pred)
precision = precision_score(y_train, y_pred, average='macro')
recall = recall_score(y_train, y_pred, average='macro')
f1 = f1_score(y_train, y_pred, average='macro')

print('=== BOW + stats ===')
print(f"Accuracy: {accuracy:.6f}")
print(f"Precision: {precision:.6f}")
print(f"Recall: {recall:.6f}")
print(f"F1-Score: {f1:.6f}")
```

```
=== BOW + stats ===
Accuracy: 0.820031
Precision: 0.825560
Recall: 0.820031
F1-Score: 0.819264
```

```python
# BOW hold out
# Initialize the classifier
mnb = MultinomialNB()

# Train the best model on the entire training set
mnb.fit(X_train, y_train)

# Evaluate the best model on the training set
y_pred_train = mnb.predict(X_train)
train_accuracy = accuracy_score(y_train, y_pred_train)

# Evaluate the best model on the test set
y_pred_test = mnb.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred_test)

print("Model Accuracy (on training set):", round(train_accuracy,6))
print("Model Accuracy (on test set):", round(test_accuracy,6))

# Calculate metrics on the test set
test_accuracy = accuracy_score(y_test, y_pred_test)
test_precision = precision_score(y_test, y_pred_test, average='macro')
test_recall = recall_score(y_test, y_pred_test, average='macro')
test_f1 = f1_score(y_test, y_pred_test, average='macro')

print('=== BOW + stats Test Metrics ===')
print(f"Accuracy: {test_accuracy:.6f}")
print(f"Precision: {test_precision:.6f}")
print(f"Recall: {test_recall:.6f}")
print(f"F1-Score: {test_f1:.6f}")
```

```
Model Accuracy (on training set): 0.821088
Model Accuracy (on test set): 0.821075
=== BOW + stats Test Metrics ===
Accuracy: 0.821075
Precision: 0.826879
Recall: 0.821075
F1-Score: 0.820277
```

```python
# add NER_featuresDF to BOW
new_df = pd.concat([bag_of_words_df, NER_featuresDF], axis=1)

new_df.shape

#split train/test - 80/20
X = new_df.drop(columns='LABEL')
y = new_df['LABEL']

# Perform the train/test split with stratification
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, stratify=y, random_state=42)
# Print the shapes of the resulting sets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (160000, 2018)
X_test shape: (40000, 2018)
y_train shape: (160000,)
y_test shape: (40000,)
```

```python
# Initialize the classifier
mnb = MultinomialNB()
cv = StratifiedKFold(n_splits=5)

# Perform cross-validation predictions
y_pred = cross_val_predict(mnb, X_train, y_train, cv=cv)

# Calculate metrics
accuracy = accuracy_score(y_train, y_pred)
precision = precision_score(y_train, y_pred, average='macro')
recall = recall_score(y_train, y_pred, average='macro')
f1 = f1_score(y_train, y_pred, average='macro')

print('=== BOW + NER ===')
print(f"Accuracy: {accuracy:.6f}")
print(f"Precision: {precision:.6f}")
print(f"Recall: {recall:.6f}")
print(f"F1-Score: {f1:.6f}")
```

```
=== BOW + NER ===
Accuracy: 0.819531
Precision: 0.821778
Recall: 0.819531
F1-Score: 0.819216
```

```python
# BOW hold out
# Initialize the classifier
mnb = MultinomialNB()

# Train the best model on the entire training set
mnb.fit(X_train, y_train)

# Evaluate the best model on the training set
y_pred_train = mnb.predict(X_train)
train_accuracy = accuracy_score(y_train, y_pred_train)

# Evaluate the best model on the test set
y_pred_test = mnb.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred_test)

print("Model Accuracy (on training set):", round(train_accuracy,6))
print("Model Accuracy (on test set):", round(test_accuracy,6))

# Calculate metrics on the test set
test_accuracy = accuracy_score(y_test, y_pred_test)
test_precision = precision_score(y_test, y_pred_test, average='macro')
test_recall = recall_score(y_test, y_pred_test, average='macro')
test_f1 = f1_score(y_test, y_pred_test, average='macro')

print('=== BOW + NER  Test Metrics ===')
print(f"Accuracy: {test_accuracy:.6f}")
print(f"Precision: {test_precision:.6f}")
print(f"Recall: {test_recall:.6f}")
print(f"F1-Score: {test_f1:.6f}")
```

```
Model Accuracy (on training set): 0.820944
Model Accuracy (on test set): 0.82135
=== BOW + NER  Test Metrics ===
Accuracy: 0.821350
Precision: 0.823801
Recall: 0.821350
F1-Score: 0.821011
```

```python
# add Vader_featuresDF to BOW
new_df = pd.concat([bag_of_words_df, Vader_featuresDF], axis=1)

#mnb cannot take negative values, +1 to Vader's compound value
new_df['compound'] = new_df['compound'] + 1

new_df.shape

#split train/test - 80/20
X = new_df.drop(columns='LABEL')
y = new_df['LABEL']

# Perform the train/test split with stratification
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, stratify=y, random_state=42)
# Print the shapes of the resulting sets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (160000, 2004)
X_test shape: (40000, 2004)
y_train shape: (160000,)
y_test shape: (40000,)
```

```python
# Initialize the classifier
mnb = MultinomialNB()
cv = StratifiedKFold(n_splits=5)

# Perform cross-validation predictions
y_pred = cross_val_predict(mnb, X_train, y_train, cv=cv)

# Calculate metrics
accuracy = accuracy_score(y_train, y_pred)
precision = precision_score(y_train, y_pred, average='macro')
recall = recall_score(y_train, y_pred, average='macro')
f1 = f1_score(y_train, y_pred, average='macro')

print('=== BOW + Vader ===')
print(f"Accuracy: {accuracy:.6f}")
print(f"Precision: {precision:.6f}")
print(f"Recall: {recall:.6f}")
print(f"F1-Score: {f1:.6f}")
```

```
=== BOW + Vader ===
Accuracy: 0.805875
Precision: 0.820609
Recall: 0.805875
F1-Score: 0.803619
```

```python
# BOW hold out
# Initialize the classifier
mnb = MultinomialNB()

# Train the best model on the entire training set
mnb.fit(X_train, y_train)

# Evaluate the best model on the training set
y_pred_train = mnb.predict(X_train)
train_accuracy = accuracy_score(y_train, y_pred_train)

# Evaluate the best model on the test set
y_pred_test = mnb.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred_test)

print("Model Accuracy (on training set):", round(train_accuracy,6))
print("Model Accuracy (on test set):", round(test_accuracy,6))

# Calculate metrics on the test set
test_accuracy = accuracy_score(y_test, y_pred_test)
test_precision = precision_score(y_test, y_pred_test, average='macro')
test_recall = recall_score(y_test, y_pred_test, average='macro')
test_f1 = f1_score(y_test, y_pred_test, average='macro')

print('=== BOW + Vader  Test Metrics ===')
print(f"Accuracy: {test_accuracy:.6f}")
print(f"Precision: {test_precision:.6f}")
print(f"Recall: {test_recall:.6f}")
print(f"F1-Score: {test_f1:.6f}")
```

```
Model Accuracy (on training set): 0.807025
Model Accuracy (on test set): 0.8067
=== BOW + Vader  Test Metrics ===
Accuracy: 0.806700
Precision: 0.821550
Recall: 0.806700
F1-Score: 0.804442
```

```python
# add bigrams500_df to BOW
new_df = pd.concat([bag_of_words_df, bigrams500_df], axis=1)

new_df.shape

#split train/test - 80/20
X = new_df.drop(columns='LABEL')
y = new_df['LABEL']

# Perform the train/test split with stratification
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, stratify=y, random_state=42)
# Print the shapes of the resulting sets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (160000, 2500)
X_test shape: (40000, 2500)
y_train shape: (160000,)
y_test shape: (40000,)
```

```python
# Initialize the classifier
mnb = MultinomialNB()
cv = StratifiedKFold(n_splits=5)

# Perform cross-validation predictions
y_pred = cross_val_predict(mnb, X_train, y_train, cv=cv)

# Calculate metrics
accuracy = accuracy_score(y_train, y_pred)
precision = precision_score(y_train, y_pred, average='macro')
recall = recall_score(y_train, y_pred, average='macro')
f1 = f1_score(y_train, y_pred, average='macro')

print('=== BOW + Bigrams ===')
print(f"Accuracy: {accuracy:.6f}")
print(f"Precision: {precision:.6f}")
print(f"Recall: {recall:.6f}")
print(f"F1-Score: {f1:.6f}")
```

```
=== BOW + Bigrams ===
Accuracy: 0.814787
Precision: 0.815258
Recall: 0.814787
F1-Score: 0.814718
```

```python
# BOW hold out
# Initialize the classifier
mnb = MultinomialNB()

# Train the best model on the entire training set
mnb.fit(X_train, y_train)

# Evaluate the best model on the training set
y_pred_train = mnb.predict(X_train)
train_accuracy = accuracy_score(y_train, y_pred_train)

# Evaluate the best model on the test set
y_pred_test = mnb.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred_test)

print("Model Accuracy (on training set):", round(train_accuracy,6))
print("Model Accuracy (on test set):", round(test_accuracy,6))

# Calculate metrics on the test set
test_accuracy = accuracy_score(y_test, y_pred_test)
test_precision = precision_score(y_test, y_pred_test, average='macro')
test_recall = recall_score(y_test, y_pred_test, average='macro')
test_f1 = f1_score(y_test, y_pred_test, average='macro')

print('=== BOW + Bigrams  Test Metrics ===')
print(f"Accuracy: {test_accuracy:.6f}")
print(f"Precision: {test_precision:.6f}")
print(f"Recall: {test_recall:.6f}")
print(f"F1-Score: {test_f1:.6f}")
```

```
Model Accuracy (on training set): 0.816344
Model Accuracy (on test set): 0.815825
=== BOW + Bigrams  Test Metrics ===
Accuracy: 0.815825
Precision: 0.816348
Recall: 0.815825
F1-Score: 0.815749
```

```python
# add everything
new_df = pd.concat([bag_of_words_df, POS_featuresDF, NewEmoji_featuresDF, bigrams500_df, percentages_featuresDF, NER_featuresDF, Vader_featuresDF], axis

# Drop the 'original_text' column
new_df = new_df.drop(columns=['original_text'])

#mnb cannot take negative values, +1 to Vader's compound value
new_df['compound'] = new_df['compound'] + 1

new_df.shape

#split train/test - 80/20
X = new_df.drop(columns='LABEL')
y = new_df['LABEL']

# Perform the train/test split with stratification
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, stratify=y, random_state=42)
# Print the shapes of the resulting sets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (160000, 3168)
X_test shape: (40000, 3168)
y_train shape: (160000,)
y_test shape: (40000,)
```

```python
# Initialize the classifier
mnb = MultinomialNB()
cv = StratifiedKFold(n_splits=5)

# Perform cross-validation predictions
y_pred = cross_val_predict(mnb, X_train, y_train, cv=cv)

# Calculate metrics
accuracy = accuracy_score(y_train, y_pred)
precision = precision_score(y_train, y_pred, average='macro')
recall = recall_score(y_train, y_pred, average='macro')
f1 = f1_score(y_train, y_pred, average='macro')

print('=== BOW + everything ===')
print(f"Accuracy: {accuracy:.6f}")
print(f"Precision: {precision:.6f}")
print(f"Recall: {recall:.6f}")
print(f"F1-Score: {f1:.6f}")
```

```
=== BOW + everything ===
Accuracy: 0.799444
Precision: 0.807585
Recall: 0.799444
F1-Score: 0.798108
```

```python
# BOW hold out
# Initialize the classifier
mnb = MultinomialNB()

# Train the best model on the entire training set
mnb.fit(X_train, y_train)

# Evaluate the best model on the training set
y_pred_train = mnb.predict(X_train)
train_accuracy = accuracy_score(y_train, y_pred_train)

# Evaluate the best model on the test set
y_pred_test = mnb.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred_test)

print("Model Accuracy (on training set):", round(train_accuracy,6))
print("Model Accuracy (on test set):", round(test_accuracy,6))

# Calculate metrics on the test set
test_accuracy = accuracy_score(y_test, y_pred_test)
test_precision = precision_score(y_test, y_pred_test, average='macro')
test_recall = recall_score(y_test, y_pred_test, average='macro')
test_f1 = f1_score(y_test, y_pred_test, average='macro')

print('=== BOW + Bigrams  Test Metrics ===')
print(f"Accuracy: {test_accuracy:.6f}")
print(f"Precision: {test_precision:.6f}")
print(f"Recall: {test_recall:.6f}")
print(f"F1-Score: {test_f1:.6f}")
```

```
Model Accuracy (on training set): 0.800338
Model Accuracy (on test set): 0.7956
=== BOW + Bigrams  Test Metrics ===
Accuracy: 0.795600
Precision: 0.803997
Recall: 0.795600
F1-Score: 0.794179
```

```python
# add everything, no BOW
new_df = pd.concat([POS_featuresDF, NewEmoji_featuresDF, bigrams500_df, percentages_featuresDF, NER_featuresDF, Vader_featuresDF], axis=1)

# Drop the 'original_text' column
new_df = new_df.drop(columns=['original_text'])

#mnb cannot take negative values, +1 to Vader's compound value
new_df['compound'] = new_df['compound'] + 1

new_df.shape

#split train/test - 80/20
X = new_df #.drop(columns='LABEL')
y = bag_of_words_df['LABEL'] #label in BOW

# Perform the train/test split with stratification
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, stratify=y, random_state=42)
# Print the shapes of the resulting sets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (160000, 1168)
X_test shape: (40000, 1168)
y_train shape: (160000,)
y_test shape: (40000,)
```

```python
# Initialize the classifier
mnb = MultinomialNB()
cv = StratifiedKFold(n_splits=5)

# Perform cross-validation predictions
y_pred = cross_val_predict(mnb, X_train, y_train, cv=cv)

# Calculate metrics
accuracy = accuracy_score(y_train, y_pred)
precision = precision_score(y_train, y_pred, average='macro')
recall = recall_score(y_train, y_pred, average='macro')
f1 = f1_score(y_train, y_pred, average='macro')

print('=== everything, no BOW ===')
print(f"Accuracy: {accuracy:.6f}")
print(f"Precision: {precision:.6f}")
print(f"Recall: {recall:.6f}")
print(f"F1-Score: {f1:.6f}")
```

```
=== everything, no BOW ===
Accuracy: 0.696712
Precision: 0.707824
Recall: 0.696712
F1-Score: 0.692604
```

```python
# BOW hold out
# Initialize the classifier
mnb = MultinomialNB()

# Train the best model on the entire training set
mnb.fit(X_train, y_train)

# Evaluate the best model on the training set
y_pred_train = mnb.predict(X_train)
train_accuracy = accuracy_score(y_train, y_pred_train)

# Evaluate the best model on the test set
y_pred_test = mnb.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred_test)

print("Model Accuracy (on training set):", round(train_accuracy,6))
print("Model Accuracy (on test set):", round(test_accuracy,6))

# Calculate metrics on the test set
test_accuracy = accuracy_score(y_test, y_pred_test)
test_precision = precision_score(y_test, y_pred_test, average='macro')
test_recall = recall_score(y_test, y_pred_test, average='macro')
test_f1 = f1_score(y_test, y_pred_test, average='macro')

print('=== everything, no BOW Test Metrics ===')
print(f"Accuracy: {test_accuracy:.6f}")
print(f"Precision: {test_precision:.6f}")
print(f"Recall: {test_recall:.6f}")
print(f"F1-Score: {test_f1:.6f}")
```

```
Model Accuracy (on training set): 0.697369
Model Accuracy (on test set): 0.69355
=== everything, no BOW Test Metrics ===
Accuracy: 0.693550
Precision: 0.703717
Recall: 0.693550
F1-Score: 0.689678
```

```python
### random forest - BOW
#split train/test - 80/20
X = bag_of_words_df.drop(columns='LABEL')
y = bag_of_words_df['LABEL']

# Perform the train/test split with stratification
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, stratify=y, random_state=42)
# Print the shapes of the resulting sets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)

# Initialize the classifier
rf = RandomForestClassifier(n_estimators=150,random_state=42)
cv = StratifiedKFold(n_splits=5)

# Perform cross-validation predictions
y_pred = cross_val_predict(rf, X_train, y_train, cv=cv)

# Calculate metrics
accuracy = accuracy_score(y_train, y_pred)
precision = precision_score(y_train, y_pred, average='macro')
recall = recall_score(y_train, y_pred, average='macro')
f1 = f1_score(y_train, y_pred, average='macro')

print('=== RF: BOW ===')
print(f"Accuracy: {accuracy:.6f}")
print(f"Precision: {precision:.6f}")
print(f"Recall: {recall:.6f}")
print(f"F1-Score: {f1:.6f}")
```

```
X_train shape: (160000, 2000)
X_test shape: (40000, 2000)
y_train shape: (160000,)
y_test shape: (40000,)
=== RF: BOW ===
Accuracy: 0.822169
Precision: 0.822514
Recall: 0.822169
F1-Score: 0.822121
```

```python
# continue, dont' need to retrain again, from above. code break point

# Evaluate the best model on the training set
y_pred_train = rf.predict(X_train)
train_accuracy = accuracy_score(y_train, y_pred_train)

# Evaluate the best model on the test set
y_pred_test = rf.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred_test)

print("Model Accuracy (on training set):", round(train_accuracy,6))
print("Model Accuracy (on test set):", round(test_accuracy,6))

# Calculate metrics on the test set
test_accuracy = accuracy_score(y_test, y_pred_test)
test_precision = precision_score(y_test, y_pred_test, average='macro')
test_recall = recall_score(y_test, y_pred_test, average='macro')
test_f1 = f1_score(y_test, y_pred_test, average='macro')

print('=== RF: BOW Test Metrics ===')
print(f"Accuracy: {test_accuracy:.6f}")
print(f"Precision: {test_precision:.6f}")
print(f"Recall: {test_recall:.6f}")
print(f"F1-Score: {test_f1:.6f}")
```

```
Model Accuracy (on training set): 0.975412
Model Accuracy (on test set): 0.820325
=== RF: BOW Test Metrics ===
Accuracy: 0.820325
Precision: 0.820628
Recall: 0.820325
F1-Score: 0.820283
```

```python
### random forest - everything
new_df = pd.concat([bag_of_words_df, POS_featuresDF, NewEmoji_featuresDF, bigrams500_df, percentages_featuresDF, NER_featuresDF, Vader_featuresDF], axis=

# Drop the 'original_text' column
new_df = new_df.drop(columns=['original_text'])

#mnb cannot take negative values, +1 to Vader's compound value
new_df['compound'] = new_df['compound'] + 1

new_df.shape

#split train/test - 80/20
X = new_df.drop(columns='LABEL')
y = new_df['LABEL']

# Perform the train/test split with stratification
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, stratify=y, random_state=42)
# Print the shapes of the resulting sets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
# Initialize the classifier
rf = RandomForestClassifier(n_estimators=150,random_state=42)
cv = StratifiedKFold(n_splits=5)

# Perform cross-validation predictions
y_pred = cross_val_predict(rf, X_train, y_train, cv=cv)

# Calculate metrics
accuracy = accuracy_score(y_train, y_pred)
precision = precision_score(y_train, y_pred, average='macro')
recall = recall_score(y_train, y_pred, average='macro')
f1 = f1_score(y_train, y_pred, average='macro')

print('=== RF: everything ===')
print(f"Accuracy: {accuracy:.6f}")
print(f"Precision: {precision:.6f}")
print(f"Recall: {recall:.6f}")
print(f"F1-Score: {f1:.6f}")
```

```
X_train shape: (160000, 3668)
X_test shape: (40000, 3668)
y_train shape: (160000,)
y_test shape: (40000,)
=== RF: everything ===
Accuracy: 0.830575
Precision: 0.831286
Recall: 0.830575
F1-Score: 0.830484
```

```python
new_df = pd.concat([bag_of_words_df, POS_featuresDF, NewEmoji_featuresDF, bigrams500_df, percentages_featuresDF, NER_featuresDF, Vader_featuresDF], axis

# Drop the 'original_text' column
new_df = new_df.drop(columns=['original_text'])

#mnb cannot take negative values, +1 to Vader's compound value
new_df['compound'] = new_df['compound'] + 1

new_df.shape

#split train/test - 80/20
X = new_df.drop(columns='LABEL')
y = new_df['LABEL']

# Perform the train/test split with stratification
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, stratify=y, random_state=42)
# Print the shapes of the resulting sets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)

# Initialize the classifier
rf = RandomForestClassifier(n_estimators=150,random_state=42)

# Train the best model on the entire training set
rf.fit(X_train, y_train)

# Evaluate the best model on the training set
y_pred_train = rf.predict(X_train)
train_accuracy = accuracy_score(y_train, y_pred_train)

# Evaluate the best model on the test set
y_pred_test = rf.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred_test)

print("Model Accuracy (on training set):", round(train_accuracy,6))
print("Model Accuracy (on test set):", round(test_accuracy,6))

# Calculate metrics on the test set
test_accuracy = accuracy_score(y_test, y_pred_test)
test_precision = precision_score(y_test, y_pred_test, average='macro')
test_recall = recall_score(y_test, y_pred_test, average='macro')
test_f1 = f1_score(y_test, y_pred_test, average='macro')

print('=== RF: everything test Metrics ===')
print(f"Accuracy: {test_accuracy:.6f}")
print(f"Precision: {test_precision:.6f}")
print(f"Recall: {test_recall:.6f}")
print(f"F1-Score: {test_f1:.6f}")

os.system('echo -e "\a"') #play a sound to alert me
```

```
X_train shape: (160000, 3168)
X_test shape: (40000, 3168)
y_train shape: (160000,)
y_test shape: (40000,)
Model Accuracy (on training set): 0.995762
Model Accuracy (on test set): 0.8296
=== RF: everything test Metrics ===
Accuracy: 0.829600
Precision: 0.830449
Recall: 0.829600
F1-Score: 0.829490
```

```python
### random forest - everything, no BOW
new_df = pd.concat([POS_featuresDF, NewEmoji_featuresDF, bigrams500_df, percentages_featuresDF, NER_featuresDF, Vader_featuresDF], axis=1)

# Drop the 'original_text' column
new_df = new_df.drop(columns=['original_text'])

#mnb cannot take negative values, +1 to Vader's compound value
new_df['compound'] = new_df['compound'] + 1

new_df.shape

#split train/test - 80/20
X = new_df #.drop(columns='LABEL')
y = bag_of_words_df['LABEL'] #label in BOW

# Perform the train/test split with stratification
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, stratify=y, random_state=42)
# Print the shapes of the resulting sets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)

# Initialize the classifier
rf = RandomForestClassifier(n_estimators=150,random_state=42)
cv = StratifiedKFold(n_splits=5)

# Perform cross-validation predictions
y_pred = cross_val_predict(rf, X_train, y_train, cv=cv)

# Calculate metrics
accuracy = accuracy_score(y_train, y_pred)
precision = precision_score(y_train, y_pred, average='macro')
recall = recall_score(y_train, y_pred, average='macro')
f1 = f1_score(y_train, y_pred, average='macro')

print('=== RF: everything, no BOW ===')
print(f"Accuracy: {accuracy:.6f}")
print(f"Precision: {precision:.6f}")
print(f"Recall: {recall:.6f}")
print(f"F1-Score: {f1:.6f}")
```

```
X_train shape: (160000, 1668)
X_test shape: (40000, 1668)
y_train shape: (160000,)
y_test shape: (40000,)
=== RF: BOW ===
Accuracy: 0.791744
Precision: 0.792550
Recall: 0.791744
F1-Score: 0.791600
```

```python
### random forest - everything, no BOW
new_df = pd.concat([POS_featuresDF, NewEmoji_featuresDF, bigrams500_df, percentages_featuresDF, NER_featuresDF, Vader_featuresDF], axis=1)

# Drop the 'original_text' column
new_df = new_df.drop(columns=['original_text'])

#mnb cannot take negative values, +1 to Vader's compound value
new_df['compound'] = new_df['compound'] + 1

new_df.shape

#split train/test - 80/20
X = new_df #.drop(columns='LABEL')
y = bag_of_words_df['LABEL'] #label in BOW

# Perform the train/test split with stratification
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, stratify=y, random_state=42)
# Print the shapes of the resulting sets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)

# Initialize the classifier
rf = RandomForestClassifier(n_estimators=150,random_state=42)

# Train the best model on the entire training set
rf.fit(X_train, y_train)

# Evaluate the best model on the training set
y_pred_train = rf.predict(X_train)
train_accuracy = accuracy_score(y_train, y_pred_train)

# Evaluate the best model on the test set
y_pred_test = rf.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred_test)

print("Model Accuracy (on training set):", round(train_accuracy,6))
print("Model Accuracy (on test set):", round(test_accuracy,6))

# Calculate metrics on the test set
test_accuracy = accuracy_score(y_test, y_pred_test)
test_precision = precision_score(y_test, y_pred_test, average='macro')
test_recall = recall_score(y_test, y_pred_test, average='macro')
test_f1 = f1_score(y_test, y_pred_test, average='macro')

print('=== RF: everything, no BOW test Metrics ===')
print(f"Accuracy: {test_accuracy:.6f}")
print(f"Precision: {test_precision:.6f}")
print(f"Recall: {test_recall:.6f}")
print(f"F1-Score: {test_f1:.6f}")

os.system('echo -e "\a"') #play a sound to alert me
```

```
X_train shape: (160000, 1168)
X_test shape: (40000, 1168)
y_train shape: (160000,)
y_test shape: (40000,)
Model Accuracy (on training set): 0.99135
Model Accuracy (on test set): 0.783575
=== RF: everything, no BOW test Metrics ===
Accuracy: 0.783575
Precision: 0.784511
Recall: 0.783575
F1-Score: 0.783397
```

```python
### random forest - POS, Emoji, percent, NER, Vader
new_df = pd.concat([POS_featuresDF, NewEmoji_featuresDF, percentages_featuresDF, NER_featuresDF, Vader_featuresDF], axis=1)

# Drop the 'original_text' column
new_df = new_df.drop(columns=['original_text'])

new_df.shape

#split train/test - 80/20
X = new_df #.drop(columns='LABEL')
y = bag_of_words_df['LABEL'] #label in BOW

# Perform the train/test split with stratification
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, stratify=y, random_state=42)
# Print the shapes of the resulting sets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)

# Initialize the classifier
rf = RandomForestClassifier(n_estimators=150,random_state=42)
cv = StratifiedKFold(n_splits=5)

# Perform cross-validation predictions
y_pred = cross_val_predict(rf, X_train, y_train, cv=cv)

# Calculate metrics
accuracy = accuracy_score(y_train, y_pred)
precision = precision_score(y_train, y_pred, average='macro')
recall = recall_score(y_train, y_pred, average='macro')
f1 = f1_score(y_train, y_pred, average='macro')

print('=== RF: POS, Emoji, percent, NER, Vader ===')
print(f"Accuracy: {accuracy:.6f}")
print(f"Precision: {precision:.6f}")
print(f"Recall: {recall:.6f}")
print(f"F1-Score: {f1:.6f}")
```

```
X_train shape: (160000, 668)
X_test shape: (40000, 668)
y_train shape: (160000,)
y_test shape: (40000,)
=== RF: POS, Emoji, percent, NER, Vader ===
Accuracy: 0.759056
Precision: 0.760206
Recall: 0.759056
F1-Score: 0.758790
```

```python
new_df = pd.concat([POS_featuresDF, NewEmoji_featuresDF, bigrams500_df, percentages_featuresDF, NER_featuresDF, Vader_featuresDF], axis=1)

# Drop the 'original_text' column
new_df = new_df.drop(columns=['original_text'])

#mnb cannot take negative values, +1 to Vader's compound value
new_df['compound'] = new_df['compound'] + 1

new_df.shape

#split train/test - 80/20
X = new_df #.drop(columns='LABEL')
y = bag_of_words_df['LABEL'] #label in BOW

# Perform the train/test split with stratification
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, stratify=y, random_state=42)
# Print the shapes of the resulting sets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)

# Initialize the classifier
rf = RandomForestClassifier(n_estimators=150,random_state=42)

# Train the best model on the entire training set
rf.fit(X_train, y_train)

# Evaluate the best model on the training set
y_pred_train = rf.predict(X_train)
train_accuracy = accuracy_score(y_train, y_pred_train)

# Evaluate the best model on the test set
y_pred_test = rf.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred_test)

print("Model Accuracy (on training set):", round(train_accuracy,6))
print("Model Accuracy (on test set):", round(test_accuracy,6))

# Calculate metrics on the test set
test_accuracy = accuracy_score(y_test, y_pred_test)
test_precision = precision_score(y_test, y_pred_test, average='macro')
test_recall = recall_score(y_test, y_pred_test, average='macro')
test_f1 = f1_score(y_test, y_pred_test, average='macro')

print('=== RF: POS, Emoji, percent, NER, Vader TEST metric ===')
print(f"Accuracy: {test_accuracy:.6f}")
print(f"Precision: {test_precision:.6f}")
print(f"Recall: {test_recall:.6f}")
print(f"F1-Score: {test_f1:.6f}")

os.system('echo -e "\a"') #play a sound to alert me
```

```
X_train shape: (160000, 1168)
X_test shape: (40000, 1168)
y_train shape: (160000,)
y_test shape: (40000,)
Model Accuracy (on training set): 0.99135
Model Accuracy (on test set): 0.783575
=== RF: POS, Emoji, percent, NER, Vader TEST metric ===
Accuracy: 0.783575
Precision: 0.784511
Recall: 0.783575
F1-Score: 0.783397
```

```
### random forest - POS, Emoji, percent, NER, Vader
new_df = pd.concat([POS_featuresDF, NewEmoji_featuresDF, percentages_featuresDF, NER_featuresDF, Vader_featuresDF], axis=1)

# Drop the 'original_text' column
new_df = new_df.drop(columns=['original_text'])

new_df.shape

#split train/test - 80/20
X = new_df #.drop(columns='LABEL')
y = bag_of_words_df['LABEL'] #label in BOW

# Perform the train/test split with stratification
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, stratify=y, random_state=42)
# Print the shapes of the resulting sets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)

# Initialize the classifier
rf = RandomForestClassifier(n_estimators=150,random_state=42)
cv = StratifiedKFold(n_splits=5)

# Perform cross-validation predictions
y_pred = cross_val_predict(rf, X_train, y_train, cv=cv)

# Calculate metrics
accuracy = accuracy_score(y_train, y_pred)
precision = precision_score(y_train, y_pred, average='macro')
recall = recall_score(y_train, y_pred, average='macro')
f1 = f1_score(y_train, y_pred, average='macro')

print('=== RF: POS, Emoji, percent, NER, Vader ===')
print(f"Accuracy: {accuracy:.6f}")
print(f"Precision: {precision:.6f}")
print(f"Recall: {recall:.6f}")
print(f"F1-Score: {f1:.6f}")
```

```
X_train shape: (160000, 668)
X_test shape: (40000, 668)
y_train shape: (160000,)
y_test shape: (40000,)
=== RF: POS, Emoji, percent, NER, Vader ===
Accuracy: 0.759056
Precision: 0.760206
Recall: 0.759056
F1-Score: 0.758790
```

```python
# try a logic regression? on vader alone

### LR - vader only
new_df = Vader_featuresDF #pd.concat([Vader_featuresDF, bigrams500_df], axis=1)

new_df.shape

#split train/test - 80/20
X = new_df #.drop(columns='LABEL')
y = bag_of_words_df['LABEL'] #label in BOW

# Perform the train/test split with stratification
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, stratify=y, random_state=42)
# Print the shapes of the resulting sets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)

# Initialize the classifier
lr = LogisticRegression(random_state=42)
cv = StratifiedKFold(n_splits=5)

# Perform cross-validation predictions
y_pred = cross_val_predict(lr, X_train, y_train, cv=cv)

# Calculate metrics
accuracy = accuracy_score(y_train, y_pred)
precision = precision_score(y_train, y_pred, average='macro')
recall = recall_score(y_train, y_pred, average='macro')
f1 = f1_score(y_train, y_pred, average='macro')

print('=== LR: Vader ===')
print(f"Accuracy: {accuracy:.6f}")
print(f"Precision: {precision:.6f}")
print(f"Recall: {recall:.6f}")
print(f"F1-Score: {f1:.6f}")
```

```
X_train shape: (160000, 4)
X_test shape: (40000, 4)
y_train shape: (160000,)
y_test shape: (40000,)
=== LR: Vader ===
Accuracy: 0.667538
Precision: 0.667575
Recall: 0.667538
F1-Score: 0.667519
```

```python
### LR - vader only
new_df = Vader_featuresDF #pd.concat([Vader_featuresDF, bigrams500_df], axis=1)

new_df.shape

#split train/test - 80/20
X = new_df #.drop(columns='LABEL')
y = bag_of_words_df['LABEL'] #label in BOW

# Perform the train/test split with stratification
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, stratify=y, random_state=42)
# Print the shapes of the resulting sets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)

# Initialize the classifier
lr = LogisticRegression(random_state=42)

# Train the best model on the entire training set
lr.fit(X_train, y_train)

# Evaluate the best model on the training set
y_pred_train = lr.predict(X_train)
train_accuracy = accuracy_score(y_train, y_pred_train)

# Evaluate the best model on the test set
y_pred_test = lr.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred_test)

print("Model Accuracy (on training set):", round(train_accuracy,6))
print("Model Accuracy (on test set):", round(test_accuracy,6))

# Calculate metrics on the test set
test_accuracy = accuracy_score(y_test, y_pred_test)
test_precision = precision_score(y_test, y_pred_test, average='macro')
test_recall = recall_score(y_test, y_pred_test, average='macro')
test_f1 = f1_score(y_test, y_pred_test, average='macro')

print('=== LR: Vader TEST metric ===')
print(f"Accuracy: {test_accuracy:.6f}")
print(f"Precision: {test_precision:.6f}")
print(f"Recall: {test_recall:.6f}")
print(f"F1-Score: {test_f1:.6f}")

os.system('echo -e "\a"') #play a sound to alert me
```

```
X_train shape: (160000, 4)
X_test shape: (40000, 4)
y_train shape: (160000,)
y_test shape: (40000,)
Model Accuracy (on training set): 0.667619
Model Accuracy (on test set): 0.667575
=== LR: Vader TEST metric ===
Accuracy: 0.667575
Precision: 0.667610
Recall: 0.667575
F1-Score: 0.667558
```

```python
## XGB
#split train/test - 80/20
X = bag_of_words_df.drop(columns='LABEL')
y = bag_of_words_df['LABEL']

# Perform the train/test split with stratification
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, stratify=y, random_state=42)
# Print the shapes of the resulting sets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)

# Initialize the classifier
xgb_model = xgb.XGBClassifier(random_state=42)

# Perform cross-validation predictions
cv = StratifiedKFold(n_splits=5)
y_pred = cross_val_predict(xgb_model, X_train, y_train, cv=cv)

# Calculate metrics
accuracy = accuracy_score(y_train, y_pred)
precision = precision_score(y_train, y_pred, average='macro')
recall = recall_score(y_train, y_pred, average='macro')
f1 = f1_score(y_train, y_pred, average='macro')

print('=== XGBoost: BOW ===')
print(f"Accuracy: {accuracy:.6f}")
print(f"Precision: {precision:.6f}")
print(f"Recall: {recall:.6f}")
print(f"F1-Score: {f1:.6f}")
```

```
X_train shape: (160000, 2000)
X_test shape: (40000, 2000)
y_train shape: (160000,)
y_test shape: (40000,)
=== XGBoost: BOW ===
Accuracy: 0.822706
Precision: 0.827366
Recall: 0.822706
F1-Score: 0.822073
```

```python
#hold out
#split train/test - 80/20
X = bag_of_words_df.drop(columns='LABEL')
y = bag_of_words_df['LABEL']

# Perform the train/test split with stratification
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, stratify=y, random_state=42)
# Print the shapes of the resulting sets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)

# Initialize the classifier
xgb_model = xgb.XGBClassifier(random_state=42)

# Train the best model on the entire training set
xgb_model.fit(X_train, y_train)

# Evaluate the best model on the training set
y_pred_train = xgb_model.predict(X_train)
train_accuracy = accuracy_score(y_train, y_pred_train)

# Evaluate the best model on the test set
y_pred_test = xgb_model.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred_test)

print("Model Accuracy (on training set):", round(train_accuracy,6))
print("Model Accuracy (on test set):", round(test_accuracy,6))

# Calculate metrics on the test set
test_accuracy = accuracy_score(y_test, y_pred_test)
test_precision = precision_score(y_test, y_pred_test, average='macro')
test_recall = recall_score(y_test, y_pred_test, average='macro')
test_f1 = f1_score(y_test, y_pred_test, average='macro')

print('=== XGBoost: BOW TEST metric ===')
print(f"Accuracy: {test_accuracy:.6f}")
print(f"Precision: {test_precision:.6f}")
print(f"Recall: {test_recall:.6f}")
print(f"F1-Score: {test_f1:.6f}")

os.system('echo -e "\a"') #play a sound to alert me
```

```
X_train shape: (160000, 2000)
X_test shape: (40000, 2000)
y_train shape: (160000,)
y_test shape: (40000,)
Model Accuracy (on training set): 0.83865
Model Accuracy (on test set): 0.820475
=== XGBoost: BOW TEST metric ===
Accuracy: 0.820475
Precision: 0.825657
Recall: 0.820475
F1-Score: 0.819758
```

```
## XGB
new_df = pd.concat([bag_of_words_df, POS_featuresDF, NewEmoji_featuresDF, bigrams500_df, percentages_featuresDF, NER_featuresDF, Vader_featuresDF], axis

# Drop the 'original_text' column
new_df = new_df.drop(columns=['original_text'])

new_df.shape

#split train/test - 80/20
X = new_df.drop(columns='LABEL')
y = new_df['LABEL']

# Perform the train/test split with stratification
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, stratify=y, random_state=42)
# Print the shapes of the resulting sets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)

# Initialize the classifier
xgb_model = xgb.XGBClassifier(random_state=42)

# Perform cross-validation predictions
cv = StratifiedKFold(n_splits=5)
y_pred = cross_val_predict(xgb_model, X_train, y_train, cv=cv)

# Calculate metrics
accuracy = accuracy_score(y_train, y_pred)
precision = precision_score(y_train, y_pred, average='macro')
recall = recall_score(y_train, y_pred, average='macro')
f1 = f1_score(y_train, y_pred, average='macro')

print('=== XGBoost: everything ===')
print(f"Accuracy: {accuracy:.6f}")
print(f"Precision: {precision:.6f}")
print(f"Recall: {recall:.6f}")
print(f"F1-Score: {f1:.6f}")
```

```
X_train shape: (160000, 3168)
X_test shape: (40000, 3168)
y_train shape: (160000,)
y_test shape: (40000,)
=== XGBoost: everything ===
Accuracy: 0.839819
Precision: 0.839866
Recall: 0.839819
F1-Score: 0.839813
```

```
## hold out
## XGB
new_df = pd.concat([bag_of_words_df, POS_featuresDF, NewEmoji_featuresDF, bigrams500_df, percentages_featuresDF, NER_featuresDF, Vader_featuresDF], axis

# Drop the 'original_text' column
new_df = new_df.drop(columns=['original_text'])

new_df.shape

#split train/test - 80/20
X = new_df.drop(columns='LABEL')
y = new_df['LABEL']

# Perform the train/test split with stratification
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, stratify=y, random_state=42)
# Print the shapes of the resulting sets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)

# Initialize the classifier
xgb_model = xgb.XGBClassifier(random_state=42)

# Train the best model on the entire training set
xgb_model.fit(X_train, y_train)

# Evaluate the best model on the training set
y_pred_train = xgb_model.predict(X_train)
train_accuracy = accuracy_score(y_train, y_pred_train)

# Evaluate the best model on the test set
y_pred_test = xgb_model.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred_test)

print("Model Accuracy (on training set):", round(train_accuracy,6))
print("Model Accuracy (on test set):", round(test_accuracy,6))

# Calculate metrics on the test set
test_accuracy = accuracy_score(y_test, y_pred_test)
test_precision = precision_score(y_test, y_pred_test, average='macro')
test_recall = recall_score(y_test, y_pred_test, average='macro')
test_f1 = f1_score(y_test, y_pred_test, average='macro')

print('=== XGBoost: everything TEST metric ===')
print(f"Accuracy: {test_accuracy:.6f}")
print(f"Precision: {test_precision:.6f}")
print(f"Recall: {test_recall:.6f}")
print(f"F1-Score: {test_f1:.6f}")

os.system('echo -e "\a"') #play a sound to alert me
```

```
X_train shape: (160000, 3168)
X_test shape: (40000, 3168)
y_train shape: (160000,)
y_test shape: (40000,)
Model Accuracy (on training set): 0.858862
Model Accuracy (on test set): 0.837925
=== XGBoost: everything TEST metric ===
Accuracy: 0.837925
Precision: 0.837953
Recall: 0.837925
F1-Score: 0.837922
```