COVID-19 Tweets Sentiment Analysis Report

Matan Blaich   Oriya Menahem Zabari

July 2025

Contents

1 Introduction

The COVID-19 pandemic was not only a global health emergency, but also a profound emotional and social event. As governments imposed lockdowns and uncertainty grew, millions of people turned to social media—especially Twitter—to share their thoughts, fears, hopes, and frustrations in real time. These tweets formed a rich and dynamic record of public sentiment during a crisis, providing researchers with a unique opportunity to analyze emotional reactions at scale.

In this project, we aim to automatically classify the sentiment of COVID-19-related tweets into three categories: positive, negative, or neutral. This task is framed as a multiclass text classification problem. The informal language, short-mid length, and subjective tone of tweets make this a challenging yet highly meaningful problem in natural language processing (NLP).

We began by organizing and preprocessing our dataset to ensure it was structured in a way that would allow for optimal understanding and interpretation of the data. Following that, we conducted exploratory data analysis to uncover potential patterns and relationships—such as distributions and correlations—that might influence model performance.

With the cleaned and structured dataset in hand, we proceeded to train and evaluate several classification models, including Logistic Regression, Support Vector Machines (SVM), BERT, K-Nearest Neighbors (KNN), Adaboost and Random Forest.

2 Dataset Description

This dataset contains about 45,000 tweets related to the COVID-19 pandemic, and each tweet has been manually labeled based on the sentiment it conveys — helping models learn to classify text as positive, negative, or neutral in the context of a global crisis.

Perform Text Classification on the data. The tweets have been pulled from Twitter and manual tagging has been done then. The names and usernames have been given codes to avoid any privacy concerns.

Columns:

1. Location - Country/City
2. Tweet At - Date

3. Original Tweet
4. Label - Positive/Negative/Neutral
5. UserName - UserID

2.1 Preliminary Challenges

Text data is raw and unprocessed, which provides a realistic challenge for NLP preprocessing.

3 Key Research Questions

1. What is the overall distribution of sentiments in the dataset? Are people more likely to express positive, negative, or neutral sentiments during the pandemic?
2. How are sentiment labels distributed across the days of the week? Are there specific days with more emotional tweets?
3. Which countries tweet the most during the COVID-19 pandemic? And does sentiment vary by country?
4. What are the most common words used in each sentiment class? Do positive, negative, and neutral tweets use clearly distinct vocabularies?
5. Is the length of the tweets is related the sentiment? Is tweet length correlated with classification accuracy?
6. Which vectorization method (BoW, TF-IDF, Word2Vec) works best for this task?
7. Which classification models perform best on this type of noisy, real-world data?
8. Does the tweet's metadata (like date or location) help improve classification accuracy?
9. Does using pre-trained embeddings (like Google News Word2Vec) help more than training on our own dataset?
10. What is the impact of using a deep contextual model like BERT compared to traditional ML models?
11. Which words receive the highest or lowest weights in each vectorization method?

4 Exploratory Data Analysis (EDA)

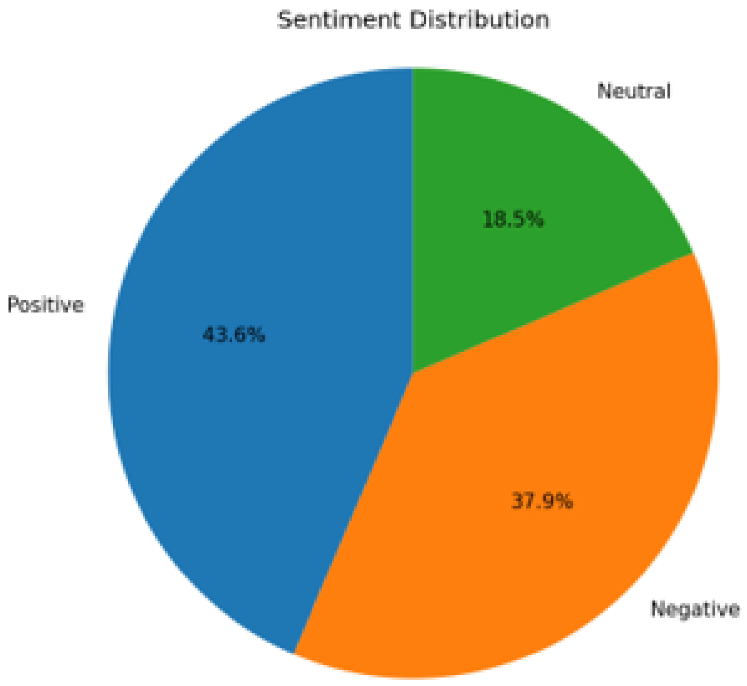
After conducting a thorough review of the dataset, we observed that the "Location" field was highly inconsistent. In some cases, it included abbreviations such as "NYC," while in others it contained only the city name, the country name, or a combination of both. We concluded that such inconsistencies could lead the model to interpret the same location as multiple distinct ones, potentially degrading performance.

To address this, we created a new column that standardizes the location information by including only the country name. Additionally, we found that the dataset spans approximately one month, which suggests that temporal variations over longer periods are unlikely to be significant. Instead, we hypothesized that the day of the week on which a tweet was posted might have some influence. To explore this, we added a new column indicating the day of the week for each tweet.

Finally, we noted that each UserName appears only once in the dataset, meaning that every person contributed exactly one tweet. As a result, we concluded that the UserName field is not relevant for modeling and can be excluded from further analysis. Our final dataset includes the following columns:

1. OriginalTweet
2. Sentiment
3. TweetDay
4. Country
5. TweetLength

4.1 Sentiment Distribution



A significant portion of the tweets (43.6%) are classified as positive. This is somewhat surprising considering the context of a global crisis. It suggests that many users used the platform to spread hope, humor, support, or share uplifting information.Negative tweets also form a substantial portion (37.9%), reflecting the widespread fear, frustration, criticism, and uncertainty experienced during the pandemic. Only 18.5% of the tweets are neutral, indicating that the majority of users expressed some form of emotional sentiment rather than remaining neutral or purely informative.

The classification model must be particularly effective in distinguishing between positive and negative sentiments, as they dominate the dataset. Since the classes are imbalanced (especially with a smaller neutral category), it necessary to apply resampling techniques to ensure fair model training.

We searched for the best suited technique to our data. Eventually, we chose Class Weight Balanced.

```
model = LogisticRegression(max_iter=1000, solver='lbfgs',
multi_class='multinomial', class_weight='balanced')
```

4.2 Day of the Week Distribution

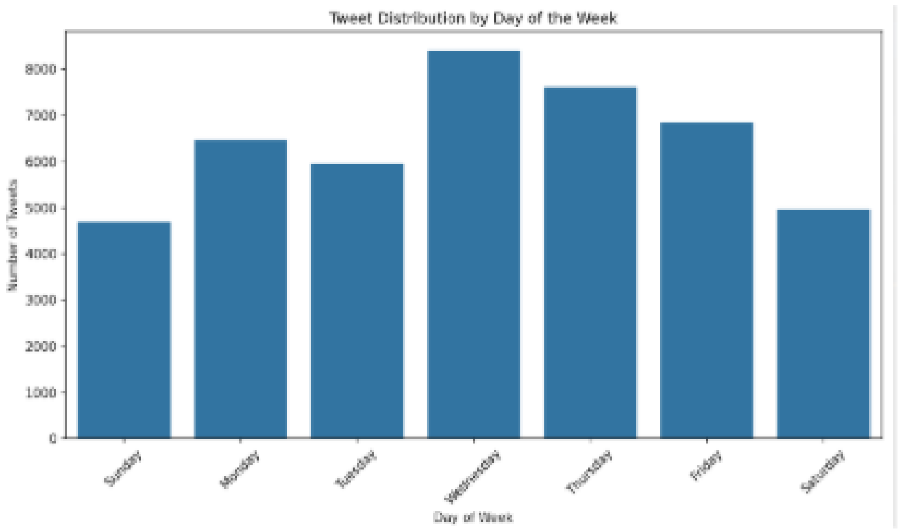


Figure 1: Tweet Distribution By Week

Tweets peak in the middle of the week—especially on Wednesday. This likely reflects higher online activity or increased engagement with midweek events.

Weekends (Saturday and Sunday) show lower tweet activity—probably due to vacations, family time, or reduced social media usage.

There is a gradual increase in activity from Sunday to Wednesday, followed by a decline toward Saturday—a typical pattern of user behavior during the workweek.

4.3 Tweets Distribution By Country

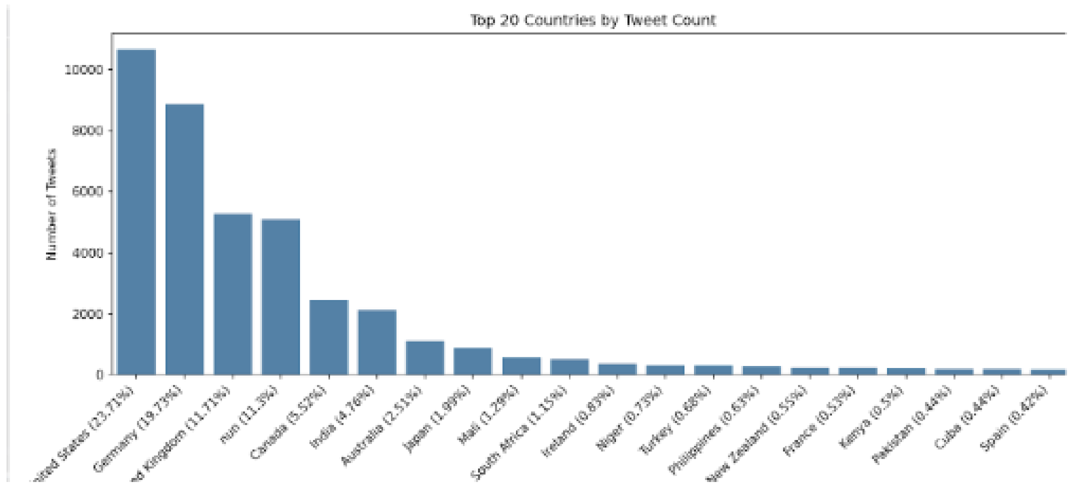


Figure 2: Top 20 Countries Distribution

We can see that USA and Germany are main countries and actually we don't have a balanced distribution on this data. We also noticed that we have a lot (11.3%) that are "nun" which are for the countries that we couldn't recognize.

4.4 Tweet Length Distribution

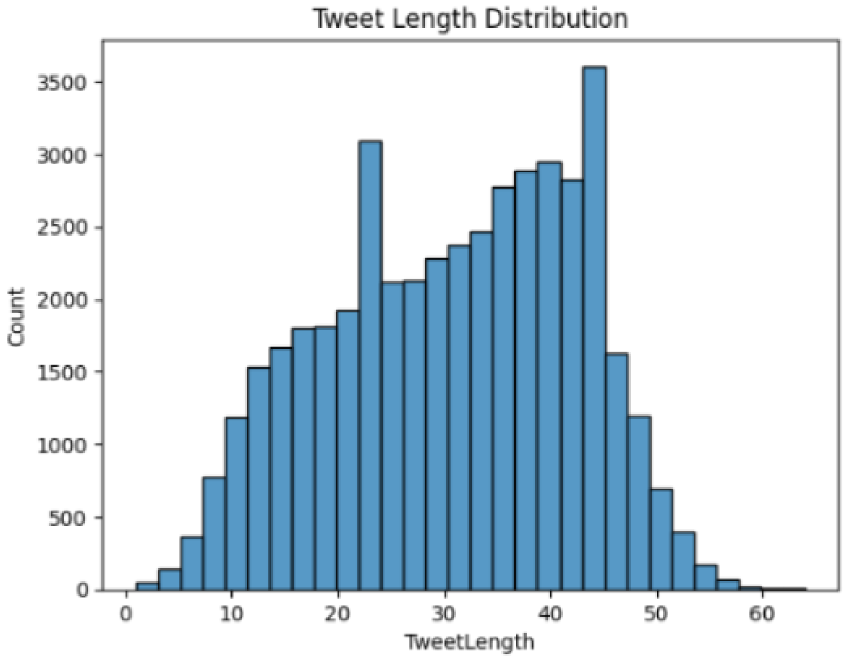
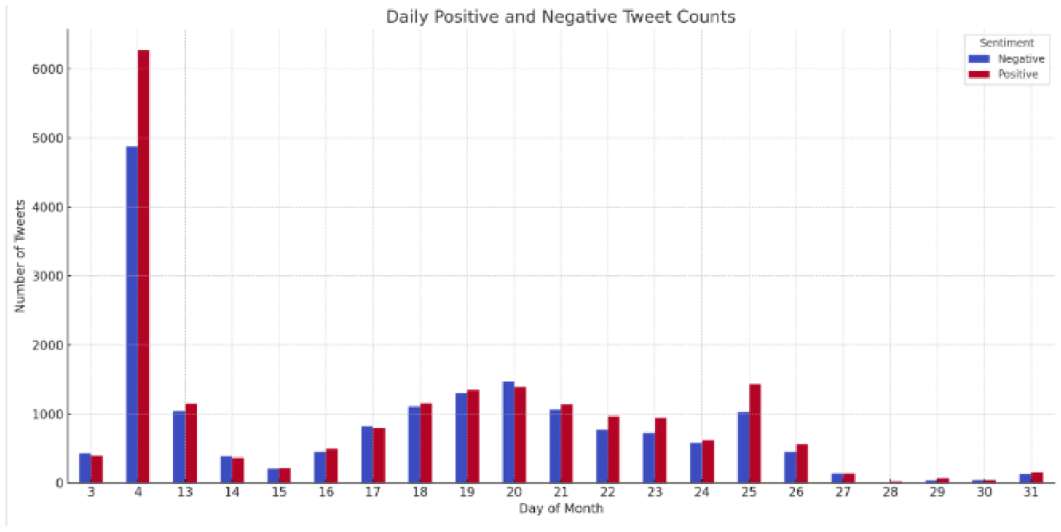


Figure 3: Tweet Length Distribution

Our tweets are mostly "mid length" which should affect the number of our dimensions - this insight can give us a lot of pre-info with choosing our best models.

4.5 Daily Positive/Negative Tweets Counts



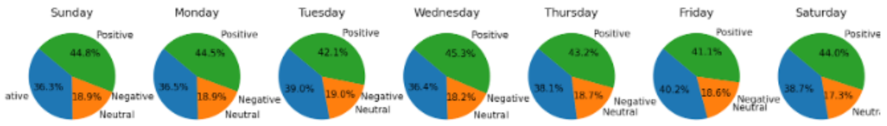
Due to the unique distribution patterns observed in the dataset, several interesting research questions emerged — for example, whether there is a significant variation in sentiment distribution (positive vs. negative) across different days of the month. To explore this, a bar chart was constructed showing the number of positive and negative tweets for each calendar day.

However, the analysis reveals that there is no substantial difference in sentiment distribution on any specific day compared to the overall dataset. While certain dates (such as the 3rd and 4th of the month) exhibit a higher volume of tweets, the ratio between positive and negative sentiments remains relatively stable throughout the month.

This finding suggests that relying solely on the date of a tweet may not be sufficient for effective sentiment analysis or prediction. Instead, incorporating additional features — such as geographic location, tweet topic, or day of the week — may offer more meaningful insights into users' emotional expressions.

4.6 Sentiment Distribution By Day Of The Week

Does sentiment vary by day of the week?

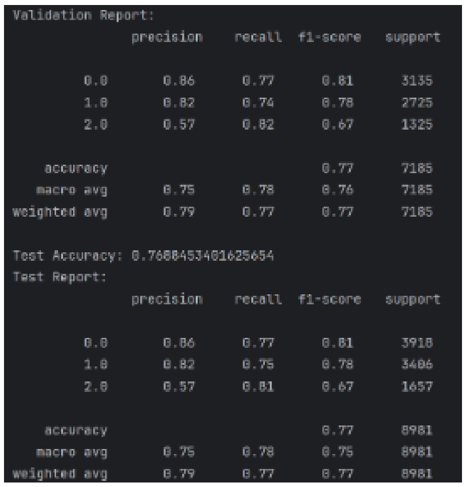


Using the datetime function, we created a new column containing only the day of the week, where 1 represents Sunday and 7 represents Saturday.

Here too, we observed a sentiment distribution for each day that is very similar to the overall sentiment distribution. Therefore, no significant correlation can be concluded between the day of the week and the type of sentiment expressed in the tweets.

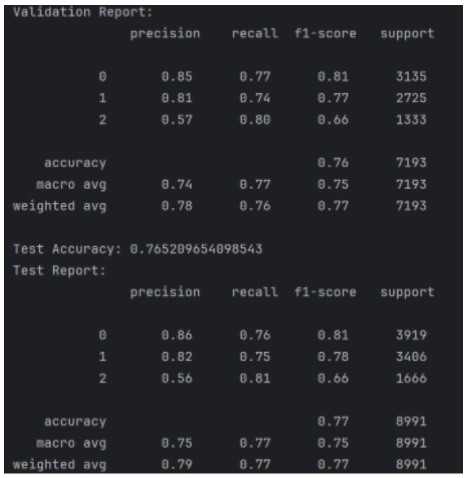
In other words, the distribution of positive, negative, and neutral tweets remains consistent across all days of the week, suggesting that the day of the week does not have a meaningful impact on the sentiment expressed by Twitter users during the analyzed period.

We conducted an experiment using Logistic Regression with the TF-IDF method. In the first run, we included the column indicating the day of the week, and in the second run, we excluded it.



```
Validation Report:
              precision    recall  f1-score   support

         0.0       0.86      0.77      0.81      3135
         1.0       0.82      0.74      0.78      2725
         2.0       0.57      0.82      0.67      1325

    accuracy                           0.77      7185
   macro avg       0.75      0.78      0.76      7185
weighted avg       0.79      0.77      0.77      7185

Test Accuracy: 0.7688453401625654
Test Report:
              precision    recall  f1-score   support

         0.0       0.86      0.77      0.81      3918
         1.0       0.82      0.75      0.78      3406
         2.0       0.57      0.81      0.67      1657

    accuracy                           0.77      8981
   macro avg       0.75      0.78      0.75      8981
weighted avg       0.79      0.77      0.77      8981
```

Figure 4: Without Days



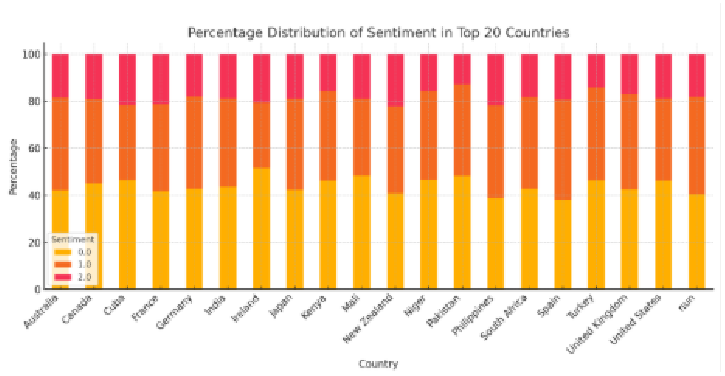
```
Validation Report:
              precision    recall  f1-score   support

           0       0.85      0.77      0.81      3135
           1       0.81      0.74      0.77      2725
           2       0.57      0.80      0.66      1333

    accuracy                           0.76      7193
   macro avg       0.74      0.77      0.75      7193
weighted avg       0.78      0.76      0.77      7193

Test Accuracy: 0.765209654098543
Test Report:
              precision    recall  f1-score   support

           0       0.86      0.76      0.81      3919
           1       0.82      0.75      0.78      3406
           2       0.56      0.81      0.66      1666

    accuracy                           0.77      8991
   macro avg       0.75      0.77      0.75      8991
weighted avg       0.79      0.77      0.77      8991
```

Figure 5: With Days

As shown, there was no significant difference in accuracy between the two runs. Therefore, we concluded that the day-of-week column does not contribute meaningfully to the model's learning.

4.7 Combined Day Countries-Sentiment Analysis

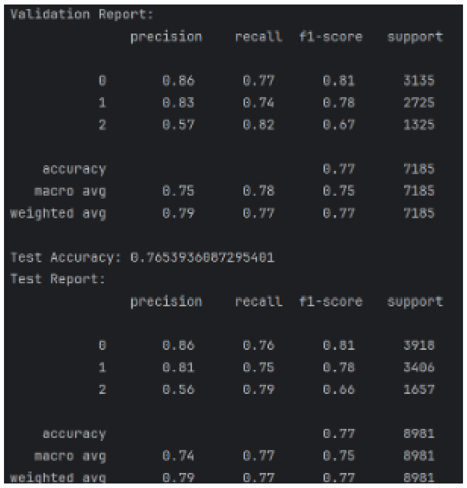
Does sentiment differ across countries?



Although there are slight differences between countries, the overall distribution of sentiments remains relatively consistent.

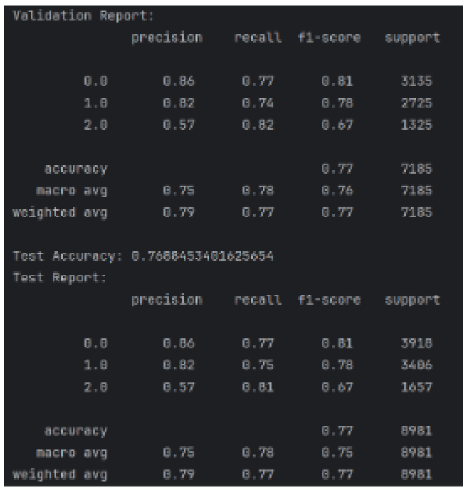
It is possible that cultural, political, or media-related factors contribute to certain countries being more positive in nature (Japan), while others show a more neutral or negative tone (Spain).

The findings suggest that the country of residence does not have a significant impact on the sentiment expressed in tweets.



```
Validation Report:
              precision    recall  f1-score   support

           0       0.86      0.77      0.81      3135
           1       0.83      0.74      0.78      2725
           2       0.57      0.82      0.67      1325

    accuracy                           0.77      7185
   macro avg       0.75      0.78      0.75      7185
weighted avg       0.79      0.77      0.77      7185

Test Accuracy: 0.7653936087295401
Test Report:
              precision    recall  f1-score   support

           0       0.86      0.76      0.81      3918
           1       0.81      0.75      0.78      3406
           2       0.56      0.79      0.66      1657

    accuracy                           0.77      8981
   macro avg       0.74      0.77      0.75      8981
weighted avg       0.79      0.77      0.77      8981
```

Figure 6: With Country

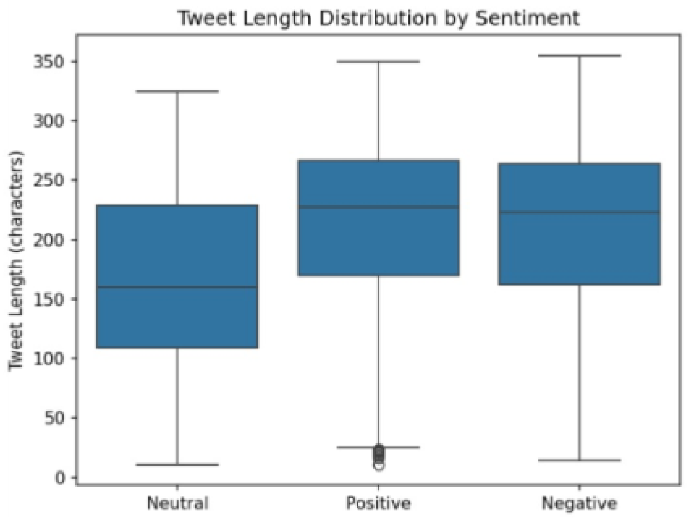


```
Validation Report:
              precision    recall  f1-score   support

         0.0       0.86      0.77      0.81      3135
         1.0       0.82      0.74      0.78      2725
         2.0       0.57      0.82      0.67      1325

    accuracy                           0.77      7185
   macro avg       0.75      0.78      0.76      7185
weighted avg       0.79      0.77      0.77      7185

Test Accuracy: 0.7688453401625654
Test Report:
              precision    recall  f1-score   support

         0.0       0.86      0.77      0.81      3918
         1.0       0.82      0.75      0.78      3406
         2.0       0.57      0.81      0.67      1657

    accuracy                           0.77      8981
   macro avg       0.75      0.78      0.75      8981
weighted avg       0.79      0.77      0.77      8981
```

Figure 7: Without Country

4.8 Combined Tweet Length-Sentiment Analysis

Is there a relationship between tweet length and sentiment?



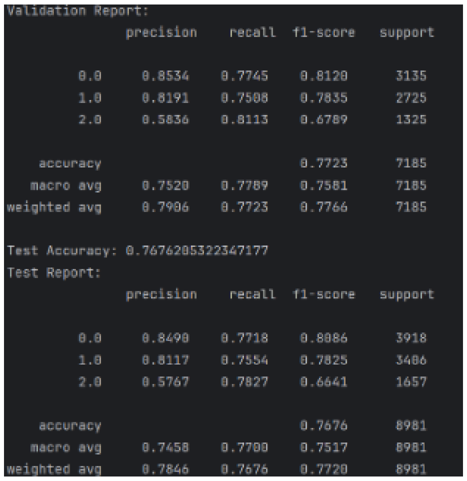
An analysis of tweet length distribution by sentiment reveals that tweets with positive or negative sentiment tend to be longer than neutral ones. The median (represented by the horizontal line inside the box) is higher for both positive and negative tweets, and the interquartile range (IQR) — the box itself — is positioned further up the y-axis, indicating greater length overall. In contrast, neutral tweets tend to be shorter on average, with a lower median and a more compact spread.Additionally, positive tweets contain a noticeable number of short outliers, represented by dots below the box.

These likely correspond to brief, upbeat expressions such as ”Yay!”, ”Thanks!”, or ”Great!”.When tweet length is explicitly included as a feature in a model, it may learn useful patterns — for example: ”If a tweet is particularly short → it is more likely to be positive.”

While one might expect that if there is a meaningful relationship between tweet length and sentiment classification, a model would be able to learn this pattern inherently—regardless of whether a tweet length feature is explicitly provided—this is not always the case. Because for example TF-IDF captures word frequency patterns, but it does not preserve the total length of the text.

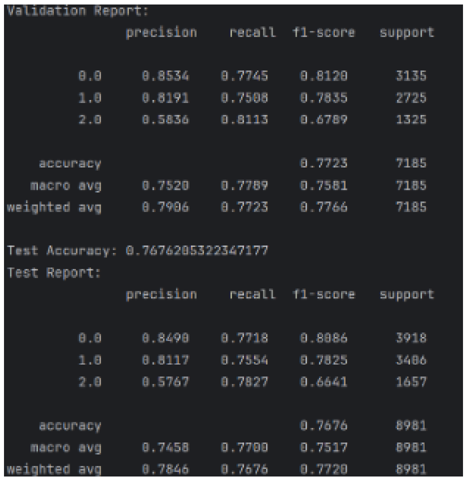
For example, two tweets that both contain a specific keyword—one consisting of 3 words and the other of 30—could appear identical in their TF-IDF representation if the remaining words are not part of the vocabulary. In such cases, potentially valuable information about the text's overall length is lost.

Based on this observation, we trained a Logistic Regression model under several configurations, both with and without the TweetLength feature. While the theoretical reasoning supports including length as a feature, we observed slight change in model performance - about 0.1 improvment, suggesting that tweet length alone may not have strong predictive power, or that similar information is already captured implicitly through other correlated features.



```
Validation Report:
              precision    recall  f1-score   support

         0.0     0.8534    0.7745    0.8120      3135
         1.0     0.8191    0.7508    0.7835      2725
         2.0     0.5836    0.8113    0.6789      1325

    accuracy                         0.7723      7185
   macro avg     0.7520    0.7789    0.7581      7185
weighted avg     0.7906    0.7723    0.7766      7185

Test Accuracy: 0.7676205322347177
Test Report:
              precision    recall  f1-score   support

         0.0     0.8490    0.7718    0.8086      3918
         1.0     0.8117    0.7554    0.7825      3406
         2.0     0.5767    0.7827    0.6641      1657

    accuracy                         0.7676      8981
   macro avg     0.7458    0.7700    0.7517      8981
weighted avg     0.7846    0.7676    0.7720      8981
```

Figure 8: Without Length



```
Validation Report:
              precision    recall  f1-score   support

         0.0     0.8534    0.7745    0.8120      3135
         1.0     0.8191    0.7508    0.7835      2725
         2.0     0.5836    0.8113    0.6789      1325

    accuracy                         0.7723      7185
   macro avg     0.7520    0.7789    0.7581      7185
weighted avg     0.7906    0.7723    0.7766      7185

Test Accuracy: 0.7676205322347177
Test Report:
              precision    recall  f1-score   support

         0.0     0.8490    0.7718    0.8086      3918
         1.0     0.8117    0.7554    0.7825      3406
         2.0     0.5767    0.7827    0.6641      1657

    accuracy                         0.7676      8981
   macro avg     0.7458    0.7700    0.7517      8981
weighted avg     0.7846    0.7676    0.7720      8981
```

Figure 9: With Length

4.9 Sentiment vs. Words (Word Clouds)

Which words appear most frequently in each sentiment category?

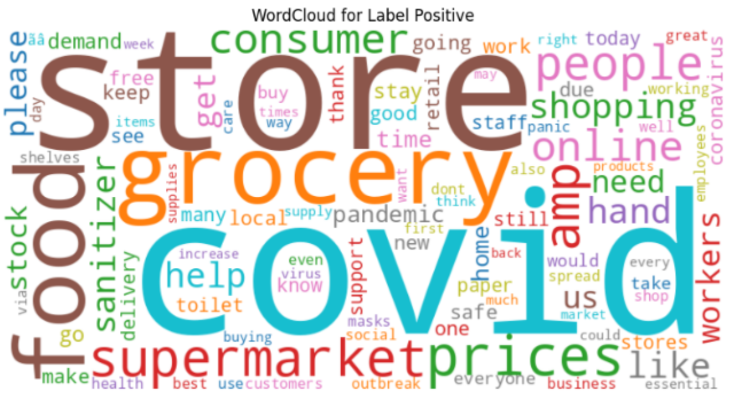


Figure 10: WordCloud for Positive Label

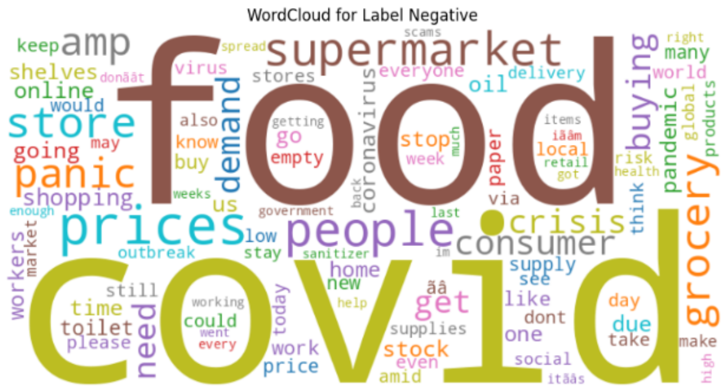


Figure 11: WordCloud for Negative Label

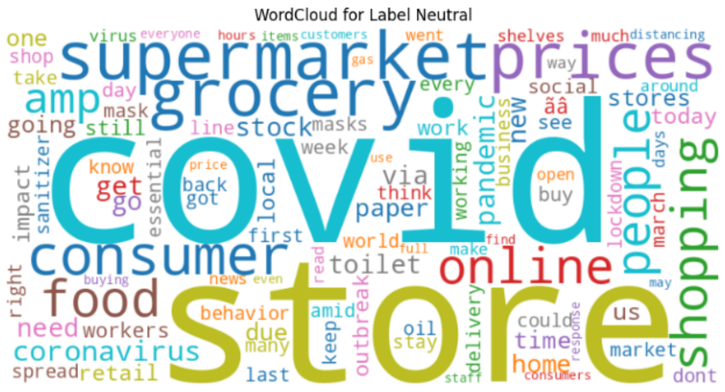


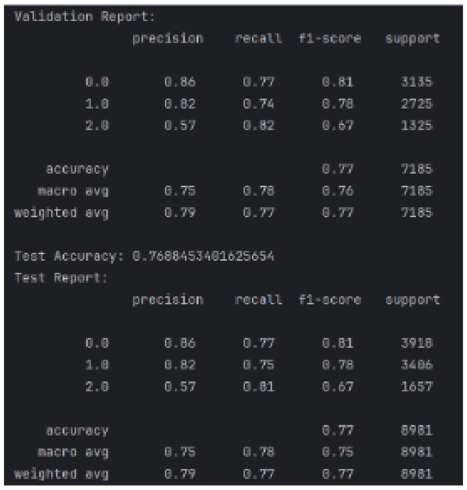
Figure 12: WordCloud for Neutral Label

From the analysis of the word clouds for each sentiment label, we can identify recurring patterns. Words like "store," "covid," "grocery," "prices," "food," "consumer," "supermarket," "online," "shopping," and "people" appear across all three sentiment categories. This indicates that these are general terms related to the central topic of the tweets—shopping, consumer behavior, and the COVID-19 pandemic. Since these words are very common and do not distinguish between sentiments, they are less helpful for the sentiment classification model.

On the other hand, certain words appear predominantly within specific sentiment categories and can serve as strong indicators. In the positive sentiment category, words like "thank," "help," "support," and "safe" stand out—terms that express satisfaction or positivity. In the negative sentiment group, words such as "panic," "crisis," "stop," "empty," and "outbreak" are prominent, conveying worry, scarcity, or threat. The neutral sentiment group is mostly characterized by informative words like "shopping," "consumer," and "online"—terms that describe events or facts without strong emotional tone. From this, we can conclude that the model might struggle to distinguish between sentiments when overly common words dominate the text.

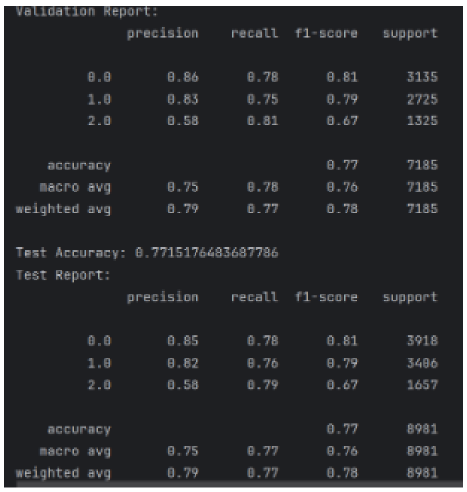
4.10 TF-IDF Max_df Filtering Impact

A possible improvement would be to filter out overly frequent words using the max df parameter in the TF-IDF vectorizer. This way, more distinctive and sentiment-relevant words would carry greater weight and contribute to more accurate sentiment classification.



```
Validation Report:
              precision    recall  f1-score   support

         0.0       0.86      0.77      0.81      3135
         1.0       0.82      0.74      0.78      2725
         2.0       0.57      0.82      0.67      1325

    accuracy                           0.77      7185
   macro avg       0.75      0.78      0.76      7185
weighted avg       0.79      0.77      0.77      7185

Test Accuracy: 0.7688453401625654
Test Report:
              precision    recall  f1-score   support

         0.0       0.86      0.77      0.81      3918
         1.0       0.82      0.75      0.78      3406
         2.0       0.57      0.81      0.67      1657

    accuracy                           0.77      8981
   macro avg       0.75      0.78      0.75      8981
weighted avg       0.79      0.77      0.77      8981
```

Figure 13: Without 80



```
Validation Report:
              precision    recall  f1-score   support

         0.0       0.86      0.78      0.81      3135
         1.0       0.83      0.75      0.79      2725
         2.0       0.58      0.81      0.67      1325

    accuracy                           0.77      7185
   macro avg       0.75      0.78      0.76      7185
weighted avg       0.79      0.77      0.78      7185

Test Accuracy: 0.7715176483687786
Test Report:
              precision    recall  f1-score   support

         0.0       0.85      0.78      0.81      3918
         1.0       0.82      0.76      0.79      3406
         2.0       0.58      0.79      0.67      1657

    accuracy                           0.77      8981
   macro avg       0.75      0.77      0.76      8981
weighted avg       0.79      0.77      0.78      8981
```

Figure 14: With 80

In this experiment, we implemented a Logistic Regression mode againl. In the first trial, we used standard TF-IDF, while in the second, we added filtering of overly common words using the parameter max df=0.8, which excludes terms that appear in more than 80% of the documents—typically words like “please,” “people,” or “amp” that contribute little to sentiment discrimination.

Despite our expectations, the overall classification performance between the two settings—standard TF-IDF versus TF-IDF with max df=0.8—showed only a marginal difference in accuracy and macro-F1 score. This suggests that removing overly frequent words, while conceptually sound, did not significantly improve the model's ability to distinguish between sentiment classes.

A possible explanation may be that the Logistic Regression model, being linear and regularized, is already quite robust to noisy or frequent terms. It learns to assign near-zero coefficients to words that appear in all classes and are uninformative for prediction. Thus, even if some common words remain, the model is capable of minimizing their influence.

5 Preprocessing Pipeline

Preprocessing was a crucial step in preparing the Twitter data for classification. Since tweets are short-mid, noisy, and often contain informal or unstructured text, we applied a full pipeline to clean and normalize the text while preserving the semantic content relevant for sentiment classification.

5.1 Text Cleaning

5.1.1 Lowercasing

All characters in the tweet were converted to lowercase. This standardization ensures that "COVID", "Covid", and "covid" are treated the same during tokenization and vectorization.

5.1.2 Removing URLs and Mentions

Hyperlinks (e.g., http://...) and mentions (e.g., @user) were removed with regex patterns. These components typically don't add meaningful sentiment and often introduce noise.

5.1.3 Removing Punctuation and Digits

We removed all non-alphanumeric characters and digits. Punctuation usually carries little sentiment value in tweets, and numbers were not deemed relevant for classification.

5.2 Tokenization

The cleaned text was tokenized using word tokenize from NLTK. This breaks the tweet into a list of words, which is required for further processing like stopword removal and lemmatization.

5.3 Stopword Removal

Common English stopwords (e.g., “the”, “is”, “and”) were removed using NLTK’s stopwords list. This helps reduce dimensionality and removes frequent words that typically don’t contribute to sentiment.

5.4 Lemmatization

Words were lemmatized using WordNetLemmatizer. Lemmatization reduces each word to its base form (e.g., “running” $\rightarrow$ “run”, “better” $\rightarrow$ “good”), helping models learn better generalizations without inflating the vocabulary.

5.5 Minimum Length Filter

We excluded very short tokens (less than 2 characters). This removes residual tokens like “rt”, “ok”, or typos that often don’t carry useful meaning.

6 Vectorization Techniques

Before feeding text into machine learning models, it must be converted into a numerical format that algorithms can process. This transformation is called vectorization. In this project, we explored three main vectorization techniques: Bag-of-Words (BoW), TF-IDF, and Word2Vec.

These techniques differ in how they represent textual data, their ability to capture context or semantics, and their impact on model performance.

6.1 Bag-of-Words (BoW)

The Bag-of-Words model is one of the simplest and most widely used text vectorization techniques. It works by:

Building a vocabulary of all unique words in the corpus.

Representing each document (or tweet) as a vector, where each element indicates the count (or presence) of a word from the vocabulary.

Example:

> If the vocabulary is [”mask”, ”virus”, ”lockdown”], then the tweet ”mask lockdown mask” would be represented as [2, 0, 1].

Pros:

- Simple and easy to implement.
- Works well with linear models like Logistic Regression and SVM.

Cons:

- Ignores word order and grammar.
- Treats all words equally, regardless of informativeness.
- Results in high-dimensional, sparse vectors.

6.2 TF-IDF

TF-IDF improves upon BoW by assigning weights to words based on their importance in the document and across the entire corpus.

Term Frequency (TF) measures how often a word appears in a document.

Inverse Document Frequency (IDF) down-weights common words that appear in many documents (like “the”, “and”) and up-weights rare, informative words.

Pros:

- Highlights important, rare words.
- Performs better than BoW for many text classification tasks.

Cons:

- Still ignores word order and semantics.
- Sensitive to vocabulary noise or stopwords unless cleaned properly.

6.3 Word2Vec (Trained and Pretrained)

Word2Vec is a neural network-based embedding model that represents each word as a dense, fixed-size vector. These vectors capture semantic relationships between words, such as:

$$\text{vector}(\text{"queen"}) \approx \text{vector}(\text{"woman"}) + \text{vector}(\text{"man"}) - \text{vector}(\text{"king"})$$

In our project, we explored both:

A locally trained Word2Vec model on our dataset.

Pretrained Word2Vec (Google News) embeddings trained on 100 billion words.

To represent a sentence or tweet: We averaged the vectors of all words in the tweet to get a single fixed-length vector.

Pros:

- Captures semantic similarity between words.
- Dense, low-dimensional vectors (typically 300D).

Cons:

- Ignores word order and sentence structure.
- Averaging can lose contextual and syntactic information.
- Requires pretrained embeddings or large data to train well.

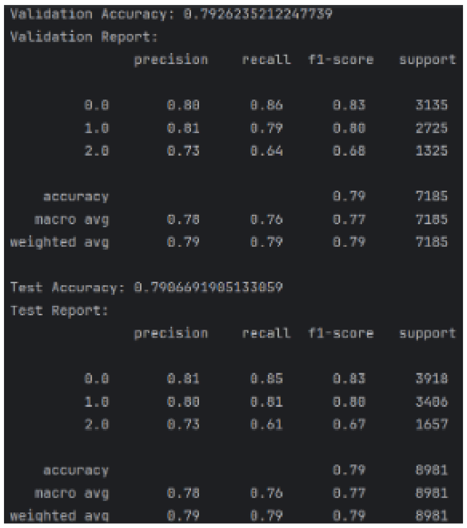
7 Classification Models

7.1 Logistic Regression

Logistic Regression is a widely used model for classification tasks, especially suitable for problems involving categorical outcomes. Unlike linear regression, which predicts continuous values, logistic regression estimates the probability of an instance belonging to a particular class using a logistic (sigmoid or softmax) function. In our case, the dataset consists of tweets labeled with three possible sentiments.

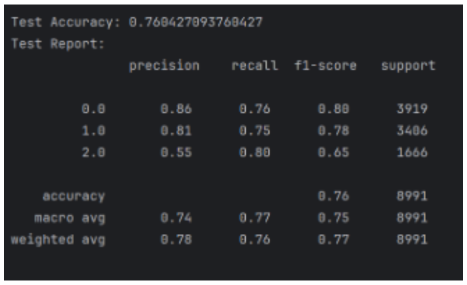
Since the tweets have been transformed into numerical representations such as TF-IDF vectors, logistic regression is particularly effective, as it performs well with sparse and high-dimensional data. It trains quickly even on large datasets and offers interpretable results, helping us understand which features influence predictions the most. For these reasons, logistic regression is a strong and practical baseline model for sentiment classification in our dataset.

7.1.1 Handling Class Imbalance



```
Validation Accuracy: 0.7926235212247739
Validation Report:
              precision    recall  f1-score   support

         0.0       0.80      0.86      0.83      3135
         1.0       0.81      0.79      0.80      2725
         2.0       0.73      0.64      0.68      1325

    accuracy                           0.79      7185
   macro avg       0.78      0.76      0.77      7185
weighted avg       0.79      0.79      0.79      7185

Test Accuracy: 0.7986691985133859
Test Report:
              precision    recall  f1-score   support

         0.0       0.81      0.85      0.83      3918
         1.0       0.80      0.81      0.80      3406
         2.0       0.73      0.61      0.67      1657

    accuracy                           0.79      8981
   macro avg       0.78      0.76      0.77      8981
weighted avg       0.79      0.79      0.79      8981
```

Figure 15: Unbalanced Weight



```
Test Accuracy: 0.7604270937680427
Test Report:
              precision    recall  f1-score   support

         0.0       0.86      0.76      0.80      3919
         1.0       0.81      0.75      0.78      3406
         2.0       0.55      0.80      0.65      1666

    accuracy                           0.76      8991
   macro avg       0.74      0.77      0.75      8991
weighted avg       0.78      0.76      0.77      8991
```

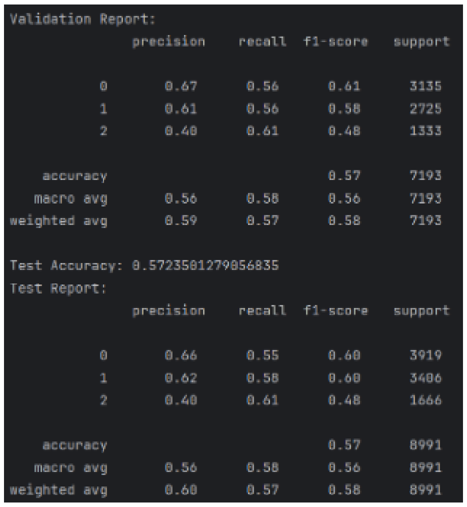
Figure 16: Balanced Weight

The Impact of Using class weight='balanced' on Model Performance When not using the class weight='balanced' parameter, the model tends to favor predicting the majority classes (such as 0.0 for positive or 1.0 for negative), since they appear more frequently in the dataset. In this case, the model rarely predicts the minority class 2.0 (neutral), resulting in high precision (when it does predict 2.0, it's usually correct) but low recall (it misses most of the actual neutral tweets).

In contrast, when class weight='balanced' is applied, the model adjusts for class imbalance by giving more weight to the underrepresented classes. As a result, the model attempts to predict more neutral tweets (2.0) and successfully identifies more of them — increasing recall. However, this also leads to more false positives (incorrectly predicting 2.0), so precision decreases.

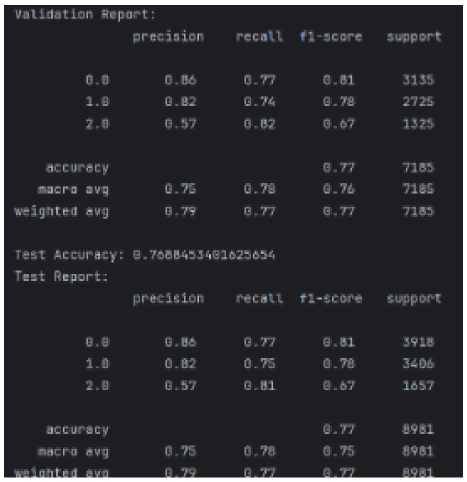
In short, using balanced improves recall for rare classes but at the cost of lower precision — a classic precision-recall tradeoff. This is the classic precision–recall trade-off: we must decide what matters more for our task—avoiding missed neutral tweets (high recall) or being certain when you label something as neutral (high precision). We chose the first option.

7.1.2 Comparison Across Vectorizations



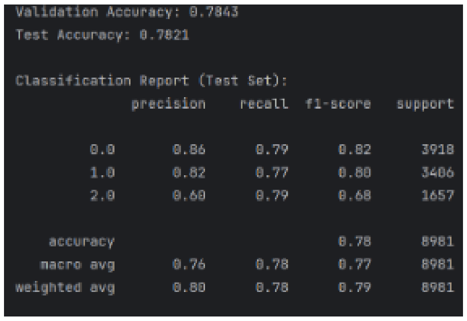
```
Validation Report:
              precision    recall  f1-score   support

           0       0.67      0.56      0.61      3135
           1       0.61      0.56      0.58      2725
           2       0.40      0.61      0.48      1333

    accuracy                           0.57      7193
   macro avg       0.56      0.58      0.56      7193
weighted avg       0.59      0.57      0.58      7193

Test Accuracy: 0.5723501279056835
Test Report:
              precision    recall  f1-score   support

           0       0.66      0.55      0.60      3919
           1       0.62      0.58      0.60      3406
           2       0.40      0.61      0.48      1666

    accuracy                           0.57      8991
   macro avg       0.56      0.58      0.56      8991
weighted avg       0.60      0.57      0.58      8991
```

Figure 17: Word2Vec Results



```
Validation Report:
              precision    recall  f1-score   support

         0.0       0.86      0.77      0.81      3135
         1.0       0.82      0.74      0.78      2725
         2.0       0.57      0.82      0.67      1325

    accuracy                           0.77      7185
   macro avg       0.75      0.78      0.76      7185
weighted avg       0.79      0.77      0.77      7185

Test Accuracy: 0.7688453481625654
Test Report:
              precision    recall  f1-score   support

         0.0       0.86      0.77      0.81      3918
         1.0       0.82      0.75      0.78      3406
         2.0       0.57      0.81      0.67      1657

    accuracy                           0.77      8981
   macro avg       0.75      0.78      0.75      8981
weighted avg       0.79      0.77      0.77      8981
```

Figure 18: TFIDF Results



```
Validation Accuracy: 0.7843
Test Accuracy: 0.7821

Classification Report (Test Set):
              precision    recall  f1-score   support

         0.0       0.86      0.79      0.82      3918
         1.0       0.82      0.77      0.80      3406
         2.0       0.60      0.79      0.68      1657

    accuracy                           0.78      8981
   macro avg       0.76      0.78      0.77      8981
weighted avg       0.80      0.78      0.79      8981
```

Figure 19: Bag Of Words Results

7.1.3 Results Overview

1. **CountVectorizer (Bag-of-Words):**
   - Validation Accuracy: 0.7843
   - Test Accuracy: 0.7821
2. **TF-IDF**
   - Validation Accuracy: 0.7692
   - Test Accuracy: 0.7688
3. **Word2Vec**:
   - Validation Accuracy: 0.57
   - Test Accuracy: 0.5723

7.1.4 Conclusions

- **CountVectorizer(BoW)** achieved the best performance overall, with the highest accuracy and weighted F1-score.
- **TF-IDF** produced close results to CountVectorizer, with a negligible difference in performance.
- **Word2Vec** had the weakest performance, with significantly lower accuracy and F1-score.

7.1.5 Why Did TF-IDF and BoW Outperform Word2Vec?

Despite Word2Vec's popularity for capturing semantic meaning, in our experiments both **TF-IDF** and **BoW** yielded better classification performance. Several factors explain this outcome:

Preservation of Discriminative Words

- **BoW and TF-IDF** maintain **explicit word presence** information.
- In sentiment analysis, specific words like "safe", "panic", "love", or "risk" often serve as **strong predictors** of sentiment.
- TF-IDF even goes further by **boosting rare but important words**, which are common in emotional expressions.
- In contrast, **We use Word2Vec with average of all word vectors**, treating each word equally unless a downstream model can learn subtle patterns — which simple classifiers like Logistic Regression cannot do well with dense embeddings. Sentences like "not good" can give us a neutral classification.

  "A few sentiment-bearing words are diluted by many neutral ones." An average tweet in your data contains about33 words, yet perhaps only 2–3 carry clear sentiment. The remaining about 30 neutral words dominate the arithmetic mean, so the influence of those sentiment-charged terms is strongly reduced.

Model Compatibility

- **BoW and TF-IDF are sparse and high-dimensional**, making them a **good match for linear models** like Logistic Regression
- These models rely on individual word features and their weights.

- **Word2Vec**, on the other hand, produces **dense, low-dimensional vectors** that are better suited for **deep or nonlinear models** (e.g., RNNs, transformers), which we did not use in combination with Word2Vec.

Corpus Size and Embedding Quality

- Word2Vec **requires a large corpus** to learn meaningful embeddings.
- Our dataset has about 45,000 tweets — relatively small for training high-quality vectors.
- The **pretrained Word2Vec (Google News)** embeddings were not aligned with our domain:
  - They were trained on formal news articles.
  - Our tweets are informal, often sarcastic, and COVID-specific.

Result: the embeddings were **less meaningful**, and out-of-vocabulary terms were frequent.

TF-IDF Provides Better Handling of Noisy Text

- Twitter data is **noisy**, full of slang, typos, and abbreviations.
- TF-IDF naturally **filters out common, generic words** (like "the", "and", "people") and focuses on content-specific words.
- Word2Vec, unless fine-tuned on similar noise, does not do this by default.

"Vector averaging flattens opposing directions." A positive-polarity vector like great points in one direction of the embedding space, while a negative one like terrible points in (roughly) the opposite. When you average ⟨great, terrible, store, prices⟩, those opposite directions cancel each other out, so the resulting vector lands near the origin—erasing the sentiment signal.

TF-IDF produced close results to CountVectorizer, because the tweets are almost binary anyway.Each tweet contains only a few dozen words, so every term appears 0or1times.CountVectorizer therefore behaves like a binary "word-present" matrix, while TF-IDF just multiplies each 0/1 by a small IDF factor (typically about1–2). The raw TF part adds almost no new information, so both matrices encode nearly the same signal.

7.2 K-Nearest Neighbors (KNN)

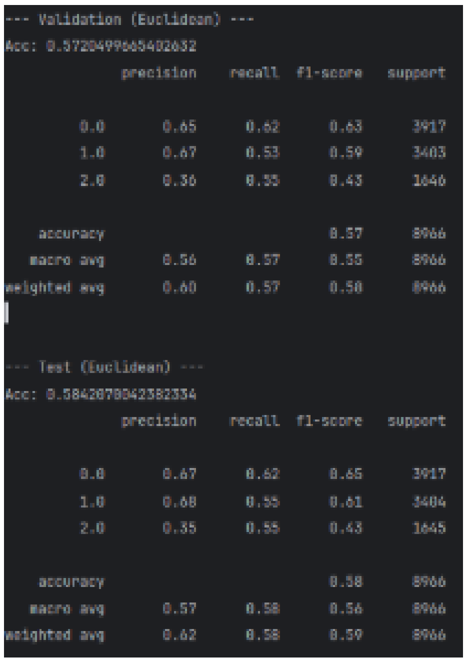
k-NearestNeighbors (k-NN) is a non-parametric algorithm that classifies a new sample by looking at the k closest neighbors in feature space: it computes a distance metric (Euclidean, cosine.) to every training example, selects the k nearest ones, and assigns the most common class among them (or the average value in regression).

k-NN is generally a poor fit for our sentiment dataset because the TF-IDF representation produces a sparse, high-dimensional space (thousands of features). In such dimensionality, distances tend to become nearly uniform, making it hard for the model to distinguish tweets—an effect known as the "curse of dimensionality." Moreover, the algorithm must store all training examples in memory and compute distances to each of them at prediction time, which becomes heavy and slow when dealing with tens of thousands of tweets. Finally, k-NN lacks an internal mechanism for class balancing, so it naturally gravitates toward the majority classes.

We applied normalization because, in KNN (or any distance-based algorithm), you need to bring all samples to the same scale; otherwise, distances will be driven by vector length rather than by actual content.

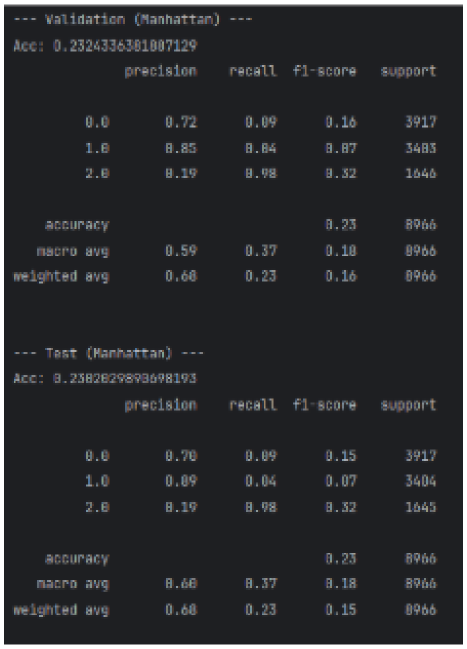
7.2.1 Distance Metrics (Cosine, L1, L2)

Using k=20 and same hyperparameters



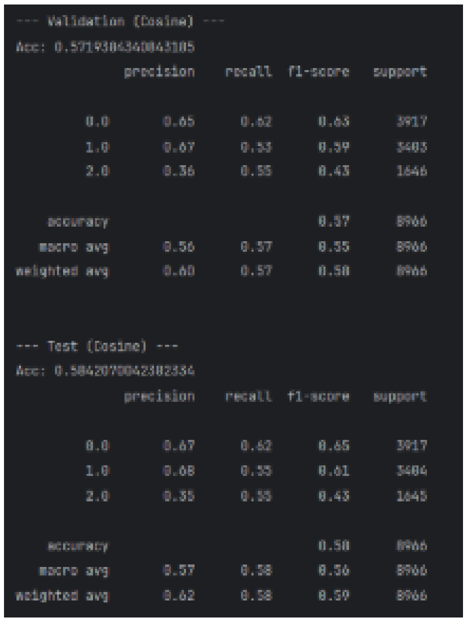
```
--- Validation (Euclidean) ---
Acc: 0.5720499665402632
              precision    recall  f1-score   support

         0.0       0.65      0.62      0.63      3917
         1.0       0.67      0.53      0.59      3403
         2.0       0.36      0.55      0.43      1646

    accuracy                           0.57      8966
   macro avg       0.56      0.57      0.55      8966
weighted avg       0.60      0.57      0.58      8966


--- Test (Euclidean) ---
Acc: 0.5842070042382334
              precision    recall  f1-score   support

         0.0       0.67      0.62      0.65      3917
         1.0       0.68      0.55      0.61      3404
         2.0       0.35      0.55      0.43      1645

    accuracy                           0.58      8966
   macro avg       0.57      0.58      0.56      8966
weighted avg       0.62      0.58      0.59      8966
```

Figure 20: Euclidean Results



```
--- Validation (Manhattan) ---
Acc: 0.2324336301007129
              precision    recall  f1-score   support

         0.0       0.72      0.09      0.16      3917
         1.0       0.85      0.04      0.07      3403
         2.0       0.19      0.98      0.32      1646

    accuracy                           0.23      8966
   macro avg       0.59      0.37      0.18      8966
weighted avg       0.68      0.23      0.16      8966


--- Test (Manhattan) ---
Acc: 0.2382829898698193
              precision    recall  f1-score   support

         0.0       0.70      0.09      0.15      3917
         1.0       0.89      0.04      0.07      3404
         2.0       0.19      0.98      0.32      1645

    accuracy                           0.23      8966
   macro avg       0.60      0.37      0.18      8966
weighted avg       0.68      0.23      0.15      8966
```

Figure 21: Manhattan Results



```
--- Validation (Cosine) ---
Acc: 0.5719384340843105
              precision    recall  f1-score   support

         0.0       0.65      0.62      0.63      3917
         1.0       0.67      0.53      0.59      3403
         2.0       0.36      0.55      0.43      1646

    accuracy                           0.57      8966
   macro avg       0.56      0.57      0.55      8966
weighted avg       0.60      0.57      0.58      8966


--- Test (Cosine) ---
Acc: 0.5842070042382334
              precision    recall  f1-score   support

         0.0       0.67      0.62      0.65      3917
         1.0       0.68      0.55      0.61      3404
         2.0       0.35      0.55      0.43      1645

    accuracy                           0.58      8966
   macro avg       0.57      0.58      0.56      8966
weighted avg       0.62      0.58      0.59      8966
```

Figure 22: Cosine Results

Cosine Distance Looks only at the angle between two TF-IDF vectors, ignoring their length. it's compares word-usage patterns rather than tweet length, which is ideal for short, sparse text documents. As soon as two tweets share even one word, cosine distance drops below 1, giving meaningful nearest neighbours for every class.

- Test accuracy =0.5842
- Best overall results; reasonably balanced across the three sentiment classes.

Euclidean (L2) Distance Validation/Test accuracy =0.5842.

Manhattan (L1) Distance Adds absolute gaps across all dimensions. In high-dimensional sparse space, most pairs end up at roughly the same distance. Neighbourhood structure is lost, so k-NN can hardly distinguish classes.

- Validation/Test accuracy 0.2302.
- Hardly detects Positive or Negative tweets at all—the neighbourhood structure collapses.

Why Manhattan distance performs poorly here ? When we build TF-IDF vectors and apply the default L2 normalization, each tweet is rescaled so that its Euclidean length equals1—but the sum of its TF-IDF weights (its L1 "mass") still varies. Longer tweets or those with a richer vocabulary have many non-zero terms, so even after L2 scaling they keep a larger L1 total.

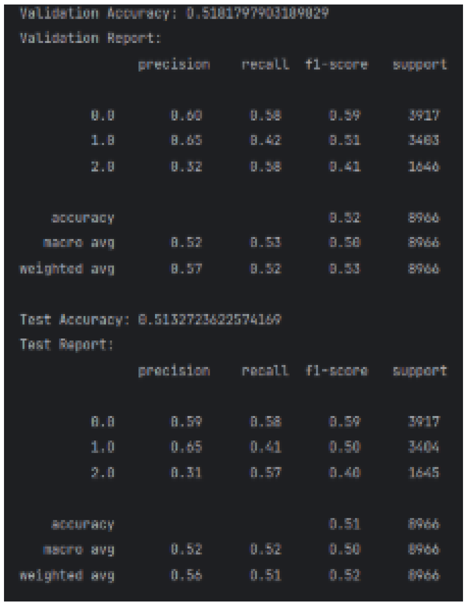
Manhattan distance (L1) simply adds up absolute differences between corresponding weights. Because its value depends directly on those L1 totals, it is strongly influenced by tweet length and vocabulary size. Two tweets that share the same sentiment words can therefore appear far apart if one is much longer: identical terms receive slightly different TF-IDF scores (due to different term frequencies and different L2 divisors), and those tiny per-term discrepancies accumulate across thousands of dimensions, pushing the Manhattan distance upward. In practice, K-NN then chooses neighbours based on document length or word-count richness rather than true topical similarity, which degrades classification accuracy.

Why are the cosine and Euclidean results so similar? Although the formulas for cosine similarity and Euclidean distance look different, once every TF-IDF vector is L-normalized to unit length, both metrics end up measuring the same notion of "closeness" — the angle between vectors. Cosine similarity directly compares that angle, while Euclidean distance between points on the unit sphere also depends solely on how far apart their directions are.

Consequently, they rank nearest neighbors in exactly the same order under KNN, producing nearly identical predictions.

7.2.2 Effect of K Value

7.2.3 K=3

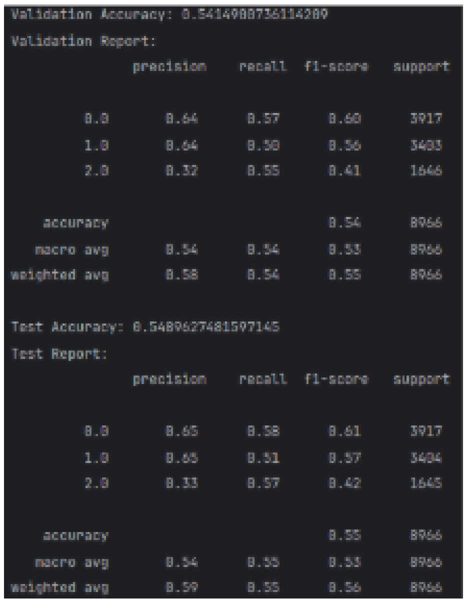


```
Validation Accuracy: 0.5181797903189029
Validation Report:
              precision    recall  f1-score   support

         0.0       0.60      0.58      0.59      3917
         1.0       0.65      0.42      0.51      3403
         2.0       0.32      0.58      0.41      1646

    accuracy                           0.52      8966
   macro avg       0.52      0.53      0.50      8966
weighted avg       0.57      0.52      0.53      8966

Test Accuracy: 0.5132723622574169
Test Report:
              precision    recall  f1-score   support

         0.0       0.59      0.58      0.59      3917
         1.0       0.65      0.41      0.50      3404
         2.0       0.31      0.57      0.40      1645

    accuracy                           0.51      8966
   macro avg       0.52      0.52      0.50      8966
weighted avg       0.56      0.51      0.52      8966
```

Figure 23: K=3

- Validation Accuracy: 0.5182
- Test Accuracy: 0.5133

7.2.4 K=7

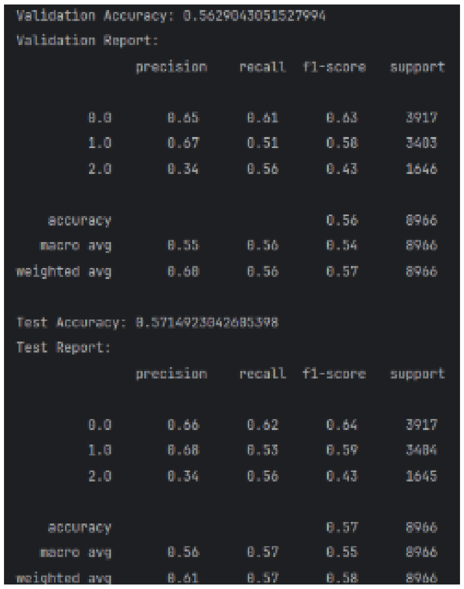


```
Validation Accuracy: 0.5414980736114289
Validation Report:
              precision    recall  f1-score   support

         0.0       0.64      0.57      0.60      3917
         1.0       0.64      0.50      0.56      3403
         2.0       0.32      0.55      0.41      1646

    accuracy                           0.54      8966
   macro avg       0.54      0.54      0.53      8966
weighted avg       0.58      0.54      0.55      8966

Test Accuracy: 0.5489627481597145
Test Report:
              precision    recall  f1-score   support

         0.0       0.65      0.58      0.61      3917
         1.0       0.65      0.51      0.57      3404
         2.0       0.33      0.57      0.42      1645

    accuracy                           0.55      8966
   macro avg       0.54      0.55      0.53      8966
weighted avg       0.59      0.55      0.56      8966
```

Figure 24: K=7

- Validation Accuracy: 0.5415
- Test Accuracy: 0.5490

7.2.5 K=15

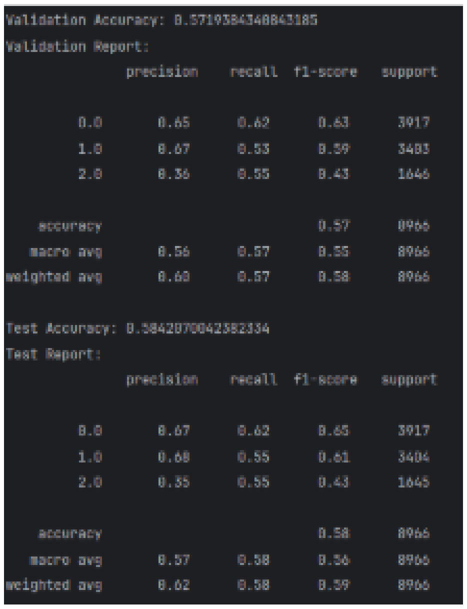


```
Validation Accuracy: 0.5629043051527994
Validation Report:
              precision    recall  f1-score   support

         0.0       0.65      0.61      0.63      3917
         1.0       0.67      0.51      0.58      3403
         2.0       0.34      0.56      0.43      1646

    accuracy                           0.56      8966
   macro avg       0.55      0.56      0.54      8966
weighted avg       0.60      0.56      0.57      8966

Test Accuracy: 0.5714923042685398
Test Report:
              precision    recall  f1-score   support

         0.0       0.66      0.62      0.64      3917
         1.0       0.68      0.53      0.59      3404
         2.0       0.34      0.56      0.43      1645

    accuracy                           0.57      8966
   macro avg       0.56      0.57      0.55      8966
weighted avg       0.61      0.57      0.58      8966
```

Figure 25: K=15

- Validation Accuracy: 0.5629
- Test Accuracy: 0.5715

7.2.6 K=20

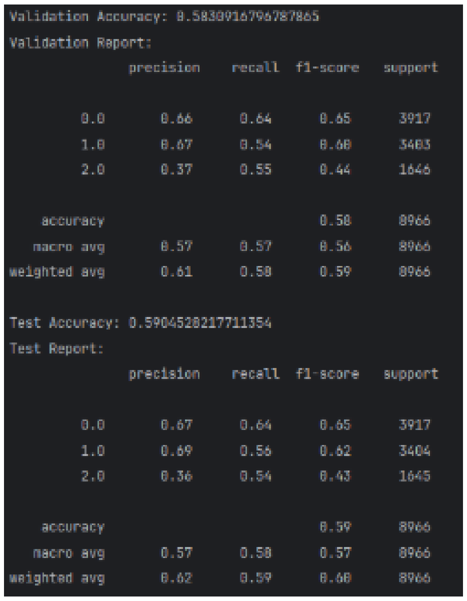


```
Validation Accuracy: 0.5719384348843185
Validation Report:
              precision    recall  f1-score   support

         0.0       0.65      0.62      0.63      3917
         1.0       0.67      0.53      0.59      3403
         2.0       0.36      0.55      0.43      1646

    accuracy                           0.57      8966
   macro avg       0.56      0.57      0.55      8966
weighted avg       0.60      0.57      0.58      8966

Test Accuracy: 0.5842070042382334
Test Report:
              precision    recall  f1-score   support

         0.0       0.67      0.62      0.65      3917
         1.0       0.68      0.55      0.61      3404
         2.0       0.35      0.55      0.43      1645

    accuracy                           0.58      8966
   macro avg       0.57      0.58      0.56      8966
weighted avg       0.62      0.58      0.59      8966
```

Figure 26: K=20

- Validation Accuracy: 0.5719
- Test Accuracy: 0.5842

7.2.7 K=25



```
Validation Accuracy: 0.5830916796787865
Validation Report:
              precision    recall  f1-score   support


         0.0       0.66      0.64      0.65      3917
         1.0       0.67      0.54      0.60      3403
         2.0       0.37      0.55      0.44      1646

    accuracy                           0.58      8966
   macro avg       0.57      0.57      0.56      8966
weighted avg       0.61      0.58      0.59      8966

Test Accuracy: 0.5904528217711354
Test Report:
              precision    recall  f1-score   support


         0.0       0.67      0.64      0.65      3917
         1.0       0.69      0.56      0.62      3404
         2.0       0.36      0.54      0.43      1645

    accuracy                           0.59      8966
   macro avg       0.57      0.58      0.57      8966
weighted avg       0.62      0.59      0.60      8966
```

Figure 27: K=25

- Validation Accuracy: 0.5831
- Test Accuracy: 0.5905

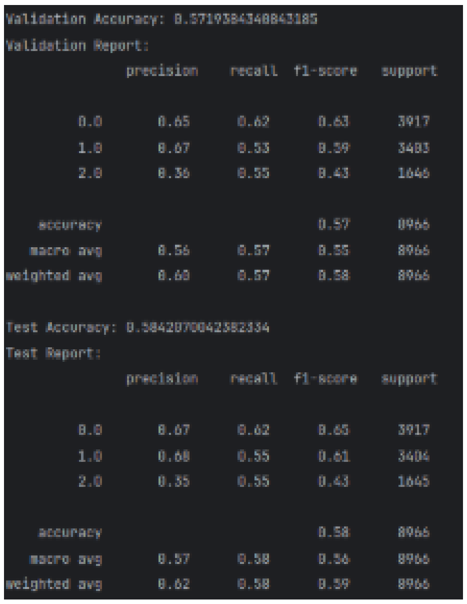
Result Analysis (English)

- **Small k (3-7)** – Each tweet is classified based on just a handful of neighbours. A single noisy tweet or a rare-class sample can flip the majority vote, leading to high variance and lower accuracy
- **Medium-to-largek (15-25)** – The model consults more neighbours, random noise is averaged out, and the vote better reflects the dominant class in the local region. The decision boundary becomes smoother and overall accuracy rises.
- **k=25** yields the highest validation and test accuracies, and both the macro-F1 and weighted-F1 scores peak at this value as well.
- The additional neighbours are increasingly far away in TF-IDF space and mostly belong to the majority classes 0 and 1. They "drown out" class 2 (the minority), so the recall for class2 stops improving beyond k=20-25.

- **Is it worth trying an even larger k?** Probably not. Moving from k=20 to25 gave only about 1% extra accuracy. Raising k to 30-40 is likely to add at most a few tenths of a percent—and may even hurt performance because class2 will be swamped further. Computation time also grows linearly withk.

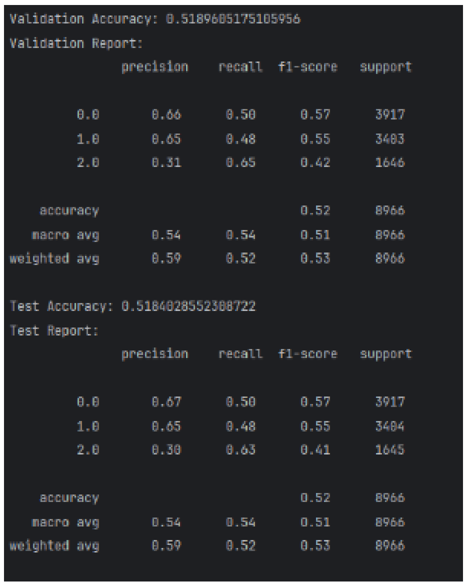
Conclusions A value of k around 20-25 is the sweet spot for this dataset: enough neighbours to smooth out noise, but not so many that the signal from the minority class is lost.

7.2.8 Comparison Across Vectorizations (using cosine)



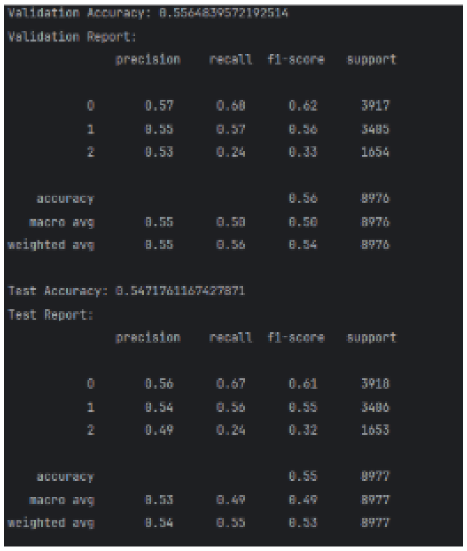
```
Validation Accuracy: 0.5719384340843185
Validation Report:
              precision    recall  f1-score   support

         0.0       0.65      0.62      0.63      3917
         1.0       0.67      0.53      0.59      3403
         2.0       0.36      0.55      0.43      1646

    accuracy                           0.57      8966
   macro avg       0.56      0.57      0.55      8966
weighted avg       0.60      0.57      0.58      8966

Test Accuracy: 0.5842070042382334
Test Report:
              precision    recall  f1-score   support

         0.0       0.67      0.62      0.65      3917
         1.0       0.68      0.55      0.61      3404
         2.0       0.35      0.55      0.43      1645

    accuracy                           0.58      8966
   macro avg       0.57      0.58      0.56      8966
weighted avg       0.62      0.58      0.59      8966
```

Figure 28: TF-IDF Results



```
Validation Accuracy: 0.5189605175105956
Validation Report:
              precision    recall  f1-score   support

         0.0       0.66      0.50      0.57      3917
         1.0       0.65      0.48      0.55      3403
         2.0       0.31      0.65      0.42      1646

    accuracy                           0.52      8966
   macro avg       0.54      0.54      0.51      8966
weighted avg       0.59      0.52      0.53      8966

Test Accuracy: 0.5184028552308722
Test Report:
              precision    recall  f1-score   support

         0.0       0.67      0.50      0.57      3917
         1.0       0.65      0.48      0.55      3404
         2.0       0.30      0.63      0.41      1645

    accuracy                           0.52      8966
   macro avg       0.54      0.54      0.51      8966
weighted avg       0.59      0.52      0.53      8966
```

Figure 29: Bag Of Words Results



```
Validation Accuracy: 0.5564839572192514
Validation Report:
              precision    recall  f1-score   support

           0       0.57      0.68      0.62      3917
           1       0.55      0.57      0.56      3485
           2       0.53      0.24      0.33      1654

    accuracy                           0.56      8976
   macro avg       0.55      0.50      0.50      8976
weighted avg       0.55      0.56      0.54      8976

Test Accuracy: 0.5471761167427871
Test Report:
              precision    recall  f1-score   support

           0       0.56      0.67      0.61      3918
           1       0.54      0.56      0.55      3406
           2       0.49      0.24      0.32      1653

    accuracy                           0.55      8977
   macro avg       0.53      0.49      0.49      8977
weighted avg       0.54      0.55      0.53      8977
```

Figure 30: Word2Vec Results

TF-IDF + Cosine

- Validation Accuracy: 0.5719

- Test Accuracy: 0.5842

Bag-of-Words (CountVectorizer) + Cosine

- Validation Accuracy: 0.5189
- Test Accuracy: 0.5184

Word2Vec + Cosine

- Validation Accuracy: 0.5565
- Test Accuracy: 0.5472

7.2.9 Results Analysis

Why TF-IDF gave the best performance ?

TF-IDF weights each word by its importance (IDF down-weights very common terms) and normalizes every document vector to a constant length, so cosine similarity focuses on the words that truly distinguish sentiment classes. In KNN's high-dimensional space, this produces a much cleaner separation of the emotion categories. Moreover, the IDF component (Inverse Document Frequency) further suppresses ubiquitous words across the corpus and amplifies unique terms, while L normalization ensures that cosine distance measures only the mixture of those informative words.

Why Word2Vec outperformed Bag-of-Words here ? Rather than raw counts, Word2Vec maps words to dense vectors capturing semantic relationships, and averaging those vectors "smooths" synonyms together (for example, "happy" and "joyful"). Compared to sparse BoW counts, this reduces noise and minor vocabulary differences, helping KNN pick more meaningful neighbors.

Why BoW TF-IDF in Logistic (but not in KNN)? Parametric classifiers like SVM or logistic regression learn feature weights during training, so they can adjust raw BoW counts to emphasize informative words—often yielding performance on par with or slightly above TF-IDF. Word2Vec averaging can dilute strong signals there, leading to lower results. In contrast, KNN does not learn weights: it computes distances directly on the input vectors. Thus TF-IDF's built-in weighting and normalization give it a big edge, Word2Vec brings some semantic smoothing, and raw BoW suffers the most.

7.3 Support Vector Machine (SVM)

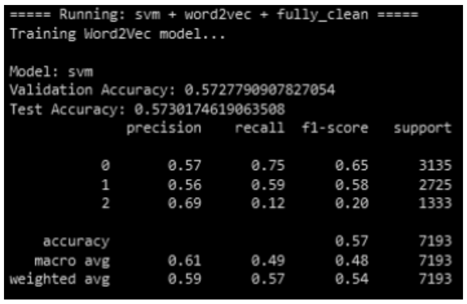
Support Vector Machine (SVM) is a powerful linear or kernel-based classifier designed to find the optimal separating hyperplane between classes by maximizing the margin. It is particularly effective in high-dimensional spaces and is well-suited for text classification problems due to its ability to handle sparse input vectors efficiently. When used with a linear kernel, SVM often performs very well on problems where classes are linearly separable or nearly so.

7.3.1 Suitability of SVM for Our Dataset and Task

Since the data is high-dimensional (especially with TF-IDF or BoW), and the input features are mostly sparse (each tweet has a limited set of meaningful words), SVM is naturally a strong candidate. Its ability to generalize well in sparse vector spaces and focus on margin-based separation aligns well with the nature of our problem.

7.3.2 Results

7.3.3 Word2Vec

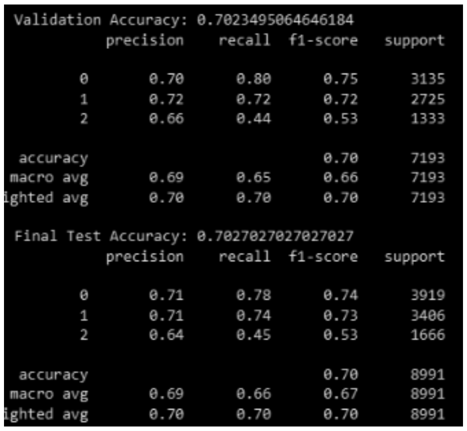


```
===== Running: svm + word2vec + fully_clean =====
Training Word2Vec model...

Model: svm
Validation Accuracy: 0.5727790907827054
Test Accuracy: 0.5730174619063508
              precision    recall  f1-score   support

           0       0.57      0.75      0.65      3135
           1       0.56      0.59      0.58      2725
           2       0.69      0.12      0.20      1333

    accuracy                           0.57      7193
   macro avg       0.61      0.49      0.48      7193
weighted avg       0.59      0.57      0.54      7193
```

Figure 31: Word2Vec Results

When using Word2Vec embeddings with SVM, we observed low performance (accuracy: 57.3%), especially in detecting the Neutral class (class 2). This is likely because averaging Word2Vec vectors removes critical context — such as word importance, negations, and phrase structure — leading to sentence representations that are too generic. As SVM relies on strong, discriminative features to separate classes, the lack of signal in these averaged embeddings limited its effectiveness. This highlights the importance of using richer or context-aware embeddings, or pairing Word2Vec with deep models rather than linear classifiers.

7.3.4 Google Word2Vec



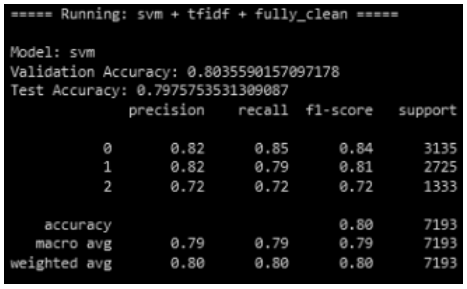
```
Validation Accuracy: 0.7023495064646184
           precision    recall  f1-score   support

        0       0.70      0.80      0.75      3135
        1       0.72      0.72      0.72      2725
        2       0.66      0.44      0.53      1333

 accuracy                           0.70      7193
macro avg       0.69      0.65      0.66      7193
ighted avg      0.70      0.70      0.70      7193

Final Test Accuracy: 0.7027027027027027
           precision    recall  f1-score   support

        0       0.71      0.78      0.74      3919
        1       0.71      0.74      0.73      3406
        2       0.64      0.45      0.53      1666

 accuracy                           0.70      8991
macro avg       0.69      0.66      0.67      8991
ighted avg      0.70      0.70      0.70      8991
```

Figure 32: Google Word2Vec Results

When switching from a locally trained Word2Vec model to Google's pre-trained Word2Vec embeddings, we observed a significant improvement in classification performance, with test accuracy rising from approximately 57% to 70%. This enhancement is attributed to the superior quality and semantic richness of the Google vectors, which were trained on a massive corpus of over 100 billion words from Google News. These embeddings capture nuanced relationships between words, including sentiment-driven similarities (e.g., "awful" and "terrible") that the local model, limited by a much smaller dataset, failed to learn effectively.

Although Word2Vec does not account for word order or syntactic structure, the strength of the pre-trained vectors compensates for these weaknesses by providing more meaningful sentence-level representations through vector averaging. This result emphasizes the value of leveraging large-scale, general-purpose embeddings in NLP tasks, particularly when working with short and noisy texts like tweets.

7.3.5 TF-IDF

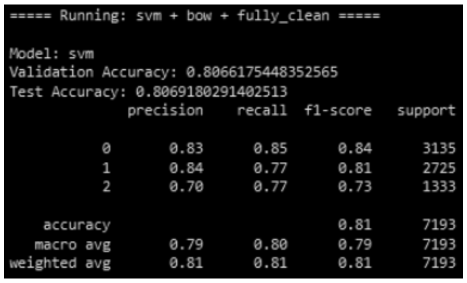


```
===== Running: svm + tfidf + fully_clean =====

Model: svm
Validation Accuracy: 0.8035590157097178
Test Accuracy: 0.7975753531309087
              precision    recall  f1-score   support

           0       0.82      0.85      0.84      3135
           1       0.82      0.79      0.81      2725
           2       0.72      0.72      0.72      1333

    accuracy                           0.80      7193
   macro avg       0.79      0.79      0.79      7193
weighted avg       0.80      0.80      0.80      7193
```

Figure 33: TF-IDF Results

With TF-IDF, performance improves significantly, reaching nearly 80% test accuracy. TF-IDF helps SVM by amplifying rare but informative terms while suppressing common ones. This allows SVM to construct better decision boundaries between sentiment categories, especially for short texts like tweets, where the presence of specific words carries strong meaning.

7.3.6 BoW - Bag Of Words



```
===== Running: svm + bow + fully_clean =====

Model: svm
Validation Accuracy: 0.8066175448352565
Test Accuracy: 0.8069180291402513
              precision    recall  f1-score   support

           0       0.83      0.85      0.84      3135
           1       0.84      0.77      0.81      2725
           2       0.70      0.77      0.73      1333

    accuracy                           0.81      7193
   macro avg       0.79      0.80      0.79      7193
weighted avg       0.81      0.81      0.81      7193
```

Figure 34: Bag Of Words Results

With Bag-of-Words, SVM performs the best, achieving 80.7% accuracy. Surprisingly, BoW outperforms TF-IDF. One likely reason is that in this dataset, some frequent words are still highly discriminative (e.g., words like “great”, “crisis”, “kill”) and TF-IDF may over-penalize them. BoW, in contrast, retains the raw frequency signal, which allows SVM to capture useful word patterns more effectively.

7.3.7 Final Remarks

SVM proves to be a strong model for this task, especially when paired with the right vectorization method. Word2Vec is not suitable in its current form due to loss of discriminative power. TF-IDF works well, and CountVectorized (BoW) works even better in this particular dataset because of how sentiment-bearing words repeat across tweets. The high-dimensional sparse nature of BoW plays directly into SVM's strengths, making it the best-performing setup for this model.

7.4 AdaBoost

AdaBoost is an ensemble method that builds a strong classifier by sequentially training weak learners, typically decision stumps. After each round, it increases the weight of misclassified samples so that subsequent learners focus more on difficult cases. The final output is a weighted majority vote across all learners.

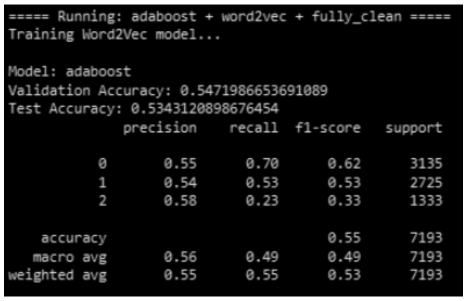
While AdaBoost performs well on structured datasets with strong, low-dimensional signals, its performance in high-dimensional NLP tasks is often limited. This is mainly because sparse features (like TF-IDF) provide weak decision rules for individual stumps, and the model can overfit on ambiguous or noisy text samples.

7.4.1 Suitability for Our Problem

Text sentiment classification, particularly with three classes (positive, negative, neutral), poses several challenges: semantic complexity, class imbalance, and the need to capture subtle linguistic cues. AdaBoost's reliance on very simple base learners (by default, depth-1 decision trees) limits its ability to model nuanced relationships in textual data.

7.4.2 Performance Analysis

7.4.3 Word2Vec



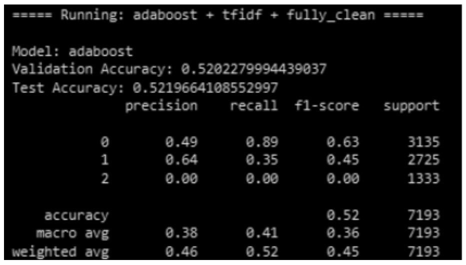
```
===== Running: adaboost + word2vec + fully_clean =====
Training Word2Vec model...

Model: adaboost
Validation Accuracy: 0.5471986653691089
Test Accuracy: 0.5343120898676454
              precision    recall  f1-score   support

           0       0.55      0.70      0.62      3135
           1       0.54      0.53      0.53      2725
           2       0.58      0.23      0.33      1333

    accuracy                           0.55      7193
   macro avg       0.56      0.49      0.49      7193
weighted avg       0.55      0.55      0.53      7193
```

Figure 35: Word2Vec Results

- Test Accuracy: 0.534
- F1-scores:
  - Positive: 0.62
  - Negative: 0.53
  - Neutral: 0.33

The model performs modestly on positive and negative tweets but fails to capture the neutral class effectively. Word2Vec embeddings provide a dense, semantic representation of words, but the weak learners in AdaBoost may not be powerful enough to exploit this representation.

7.4.4 TF-IDF



```
===== Running: adaboost + tfidf + fully_clean =====

Model: adaboost
Validation Accuracy: 0.5202279994439037
Test Accuracy: 0.5219664108552997
              precision    recall  f1-score   support

           0       0.49      0.89      0.63      3135
           1       0.64      0.35      0.45      2725
           2       0.00      0.00      0.00      1333

    accuracy                           0.52      7193
   macro avg       0.38      0.41      0.36      7193
weighted avg       0.46      0.52      0.45      7193
```

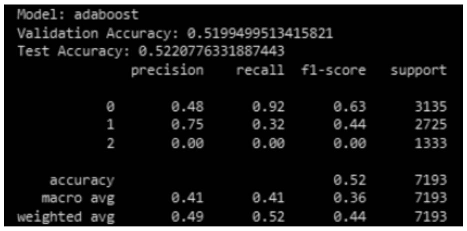
Figure 36: TF-IDF Results

- Test Accuracy: 0.522

- F1-scores:
  - Positive: 0.63 (Recall: 0.89)
  - Negative: 0.45
  - Neutral: **0.00**

Here, the model demonstrates extreme bias toward the positive class, achieving very high recall (0.89) while completely failing to classify any neutral samples. The precision-recall imbalance suggests the model heavily overfits dominant terms found in positive tweets.

7.4.5 Bag-of-Words



```
Model: adaboost
Validation Accuracy: 0.5199499513415821
Test Accuracy: 0.5220776331887443
              precision    recall  f1-score   support

           0       0.48      0.92      0.63      3135
           1       0.75      0.32      0.44      2725
           2       0.00      0.00      0.00      1333

    accuracy                           0.52      7193
   macro avg       0.41      0.41      0.36      7193
weighted avg       0.49      0.52      0.44      7193
```

Figure 37: Bag Of Words Results

- Test Accuracy: 0.522
- F1-scores:
  - Positive: 0.63 (Recall: 0.92)
  - Negative: 0.44
  - Neutral: **0.00**

Results mirror those of the TF-IDF setup. The model overly focuses on high-frequency patterns in class 0 (positive), leaving class 2 (neutral) entirely unrecognized. This is particularly concerning, as it indicates a systemic failure to detect neutral sentiment regardless of vectorization.

Conclusions and Observations

- **Weak Learner Limitation**: AdaBoost's default base learner (decision stumps) lacks the expressive power to capture the intricacies of text data, especially when using vector-based representations like Word2Vec or sparse encodings like TF-IDF/BOW.

- **Class Imbalance Sensitivity**: The model is highly sensitive to dominant classes, favoring positive sentiment while neglecting neutral altogether. This is consistent across all three vectorization methods.
- **Unsuitable in Current Form**: Given its consistent underperformance—particularly on the neutral class—AdaBoost with decision stumps appears ill-suited for this problem.
- While AdaBoost theoretically emphasizes underrepresented classes by increasing their weights, in practice it failed to classify the Neutral class when paired with TF-IDF or BoW.

  This is because sparse representations lack distinctive features for Neutral tweets, and the weak learners used in AdaBoost cannot form meaningful splits in such high-dimensional, sparse spaces. Word2Vec embeddings, by contrast, provided dense and semantically rich features that allowed the model to better cluster and separate minority-class samples, leading to improved recall.

7.4.6 Modified AdaBoost (Depth 3 Trees)

After modifying AdaBoost to use a stronger base learner (DecisionTreeClassifier(max depth=3)), we see noticeable performance improvements across all three vectorization methods.

TF-IDF This setup achieved the best performance among the three variants:

- Test Accuracy improved from 0.521 to 0.588
- Precision and recall for class 0 (the majority class) increased significantly
- However, class 2 still suffers from poor recall (0.12), indicating the model still struggles with the minority class

The stronger base learner allowed the model to capture more complex decision boundaries, especially beneficial for TF-IDF where term importance is nuanced.

Bag of Words

- Test Accuracy improved from 0.522 to 0.575
- Class 0 is still over-represented (high recall 0.83), while class 2 is completely ignored (recall 0.00)

- The macro average F1-score remains low (0.43), showing imbalance issues

This indicates that although the model captures more patterns with stronger trees, it may still overfit to frequent patterns from dominant classes.

Word2Vec

- Test Accuracy improved from 0.537 to 0.556
- A slight improvement in handling all classes, including class 2 (recall went from 0.23 to 0.30)
- Still weaker than BoW and TF-IDF overall, likely due to Word2Vec's dense representations not being well exploited by AdaBoost's ensemble structure

7.5 Random Forest

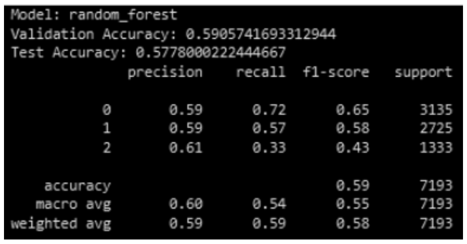
Random Forest is an ensemble model that builds multiple decision trees and combines their outputs to improve generalization and reduce overfitting. It performs well with high-dimensional, sparse data, and it is robust to noise and overfitting, especially when the number of estimators is sufficiently high. However, it doesn't perform well with dense, continuous vector spaces where the structure between features is crucial, like in Word2Vec embeddings.

7.5.1 Suitability for Our Problem

The tweet classification task involves high-dimensional, sparse text data that is preprocessed and transformed using different vectorization strategies. Since each tweet is short-mid and transformed into either a sparse binary-like vector (BoW, TF-IDF) or a dense embedding (Word2Vec), Random Forest is expected to work well with the former but not the latter. Its split-based logic aligns well with count-like features rather than continuous semantic spaces.

7.5.2 Performance Across Vectorizers

7.5.3 Word2Vec

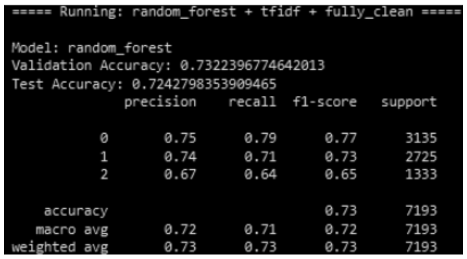


```
Model: random_forest
Validation Accuracy: 0.5905741693312944
Test Accuracy: 0.5778000222444667
              precision    recall  f1-score   support

           0       0.59      0.72      0.65      3135
           1       0.59      0.57      0.58      2725
           2       0.61      0.33      0.43      1333

    accuracy                           0.59      7193
   macro avg       0.60      0.54      0.55      7193
weighted avg       0.59      0.59      0.58      7193
```

Figure 38: Word2Vec Results

- Test Accuracy: 0.58
- The model performed poorly. Word2Vec generates dense, continuous embeddings, which are not ideal for Random Forest's decision boundary logic. Trees in the forest split based on feature thresholds, but Word2Vec features don't represent easily separable word-specific information. As a result, the model fails to capture meaningful splits and general patterns.

7.5.4 TF-IDF

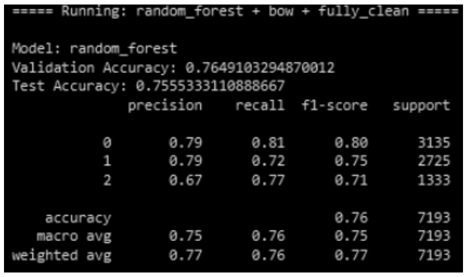


```
===== Running: random_forest + tfidf + fully_clean =====

Model: random_forest
Validation Accuracy: 0.7322396774642013
Test Accuracy: 0.7242798353909465
              precision    recall  f1-score   support

           0       0.75      0.79      0.77      3135
           1       0.74      0.71      0.73      2725
           2       0.67      0.64      0.65      1333

    accuracy                           0.73      7193
   macro avg       0.72      0.71      0.72      7193
weighted avg       0.73      0.73      0.73      7193
```

Figure 39: TF-IDF Results

- Test Accuracy: 0.72
- TF-IDF performed significantly better. This makes sense because TF-IDF vectors are sparse and high-dimensional, and Random Forest handles such structures well. The model can use the presence and importance (via the inverse document frequency) of particular words to split the data effectively.

7.5.5 Bag Of Words



```
===== Running: random_forest + bow + fully_clean =====

Model: random_forest
Validation Accuracy: 0.7649103294870012
Test Accuracy: 0.7555333110888667
              precision    recall  f1-score   support

           0       0.79      0.81      0.80      3135
           1       0.79      0.72      0.75      2725
           2       0.67      0.77      0.71      1333

    accuracy                           0.76      7193
   macro avg       0.75      0.76      0.75      7193
weighted avg       0.77      0.76      0.77      7193
```

Figure 40: Bag Of Words Results

- Test Accuracy: 0.76
- BoW outperformed TF-IDF in this case. Although BoW doesn't discount common words, it seems that for Random Forest, the raw frequency or presence of terms is enough to create useful splits. It may also suggest that the common words carry predictive power in this context, and penalizing them (as TF-IDF does) might remove useful information.

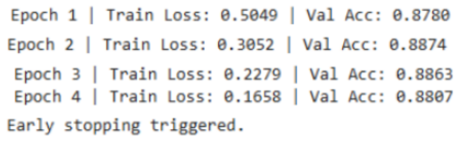
8 BERT Model

BERT (Bidirectional Encoder Representations from Transformers) is a deep transformer-based language model pre-trained on a large corpus using masked language modeling and next sentence prediction. Unlike traditional models, BERT understands the context of a word based on both its left and right surroundings. In this task, we fine-tuned the bert-base-uncased model for multi-class tweet sentiment classification.

8.1 Suitability for Our Problem

The problem of tweet sentiment classification is inherently complex due to the informal, brief, and often ambiguous nature of tweets. Despite this, BERT is particularly well-suited for such tasks because of its deep contextual understanding, ability to disambiguate meanings based on full-sentence context, and robustness to noisy text like slang, typos, and hashtags. Its pretraining on massive corpora allows it to generalize even with moderate downstream training data.

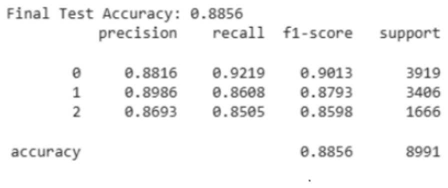
8.2 Training Details



```
Epoch 1 | Train Loss: 0.5049 | Val Acc: 0.8780
Epoch 2 | Train Loss: 0.3052 | Val Acc: 0.8874
Epoch 3 | Train Loss: 0.2279 | Val Acc: 0.8863
Epoch 4 | Train Loss: 0.1658 | Val Acc: 0.8807
Early stopping triggered.
```

Figure 41: Training Using Early Stopping

8.3 Test Results



```
Final Test Accuracy: 0.8856
             precision    recall  f1-score   support

           0     0.8816    0.9219    0.9013      3919
           1     0.8986    0.8608    0.8793      3406
           2     0.8693    0.8505    0.8598      1666

    accuracy                         0.8856      8991
```

Figure 42: Word2Vec Results

During training, the model showed steady improvements in validation accuracy, reaching a peak at 88.07% after four epochs before early stopping was triggered. This is a strong indicator of the model's ability to generalize well without overfitting. On the final test set, the model achieved an accuracy of 88.56%, significantly outperforming all previous models and vectorization approaches. The class-wise performance was also excellent:

- **Class 0**: Precision 0.8816, Recall 0.9219, F1 0.9013
- **Class 1**: Precision 0.8986, Recall 0.8608, F1 0.8793
- **Class 2**: Precision 0.8693, Recall 0.8505, F1 0.8598

8.4 Comparison to Classical Models

These metrics are highly balanced and demonstrate strong predictive capability across all sentiment categories, including the minority class (label 2), which was particularly difficult for classical models like Logistic Regression, AdaBoost, or Random Forest.

8.5 Insights and Conclusions

- BERT's ability to represent the semantic meaning of full sentences gives it a huge advantage in capturing the nuanced emotional tone of tweets.
- Unlike classical vectorizers (TF-IDF, BoW, or Word2Vec), which treat words more independently or rely on static embeddings, BERT dynamically generates embeddings for each input based on context.
- The performance gains are especially noticeable in the minority class, which was nearly ignored by some classical models. BERT, however, maintains solid recall and F1-score for this class as well.
- Overall, BERT delivers state-of-the-art performance on this sentiment classification task, and serves as a clear benchmark for future models to beat.

9 Model and Vectorizer Summary Comparison

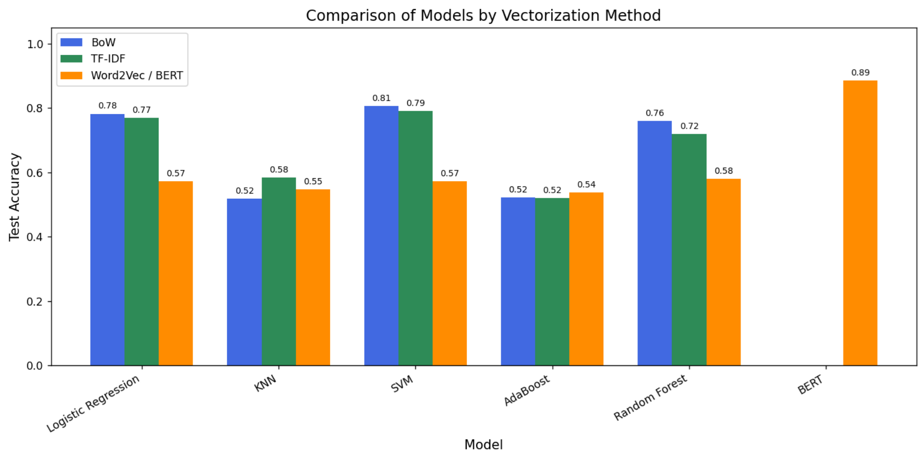


Figure 43: Overall Results

Across all tested models and vectorization methods, Support Vector Machines (SVM) and Logistic Regression achieved the highest classification performance, especially when combined with Bag-of-Words or TF-IDF vectorization.

SVM with BoW slightly outperformed all other combinations, achieving over 80% test accuracy, demonstrating its strength in handling sparse, high-dimensional text data.

Logistic Regression closely followed with comparable results. Random Forest performed reasonably well, particularly with BoW, but slightly lagged behind SVM and Logistic, possibly due to difficulty handling noisy features. Word2Vec underperformed in all models except when using Google's pretrained embeddings, which significantly improved results—indicating the importance of external semantic knowledge for short texts like tweets.

AdaBoost was the weakest model, failing to generalize well regardless of vectorizer, likely due to its sensitivity to noisy or imbalanced data and its limited base learner complexity. Overall, simpler linear models with sparse, frequency-based vectorizers proved most effective for this classification task.

10 Word Importance Analysis

The classifier reveals distinct linguistic patterns for each sentiment category based on the most influential words.

This analysis was used on Logistic Regression.

10.1 Top and Bottom Words per Class (TF-IDF BoW)

Top and bottom words by weight using WordCloud

10.1.1 Positive Tweets

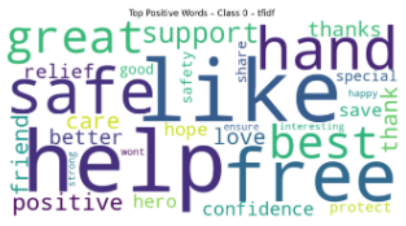


Figure 44: TF-IDF Strong Words

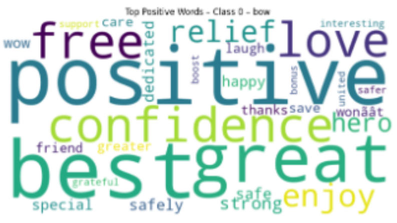


Figure 45: BoW Strong Words

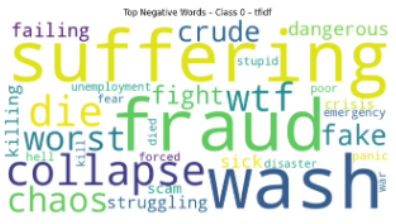


Figure 46: TF-IDF Weak Words

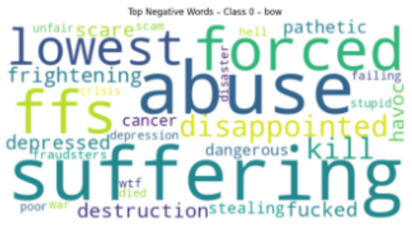


Figure 47: BoW Weak Words

Positive Tweets For positive tweets, the most representative words include "positive", "great", "love", "safe", "relief", "hero", "confidence", and "thanks". These words clearly express emotional support, appreciation, and optimism—reflecting gratitude or encouragement commonly seen in positive tweets during a crisis like COVID-19.

On the other end, the most negative-weighted words for this class are "crisis", "scam", "panic", "hell", "kill", "fraud", and "suffering". These words are highly emotional and destructive in tone, thus pulling classification away from the positive class

10.1.2 Negative Tweets

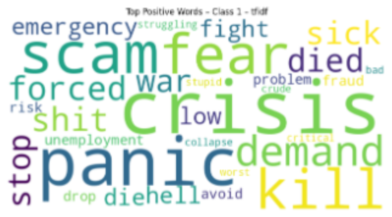


Figure 48: TF-IDF Strong Words

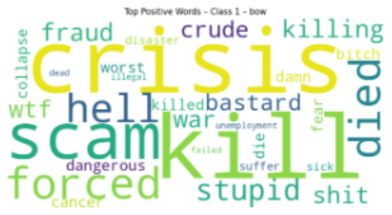


Figure 49: BoW Strong Words

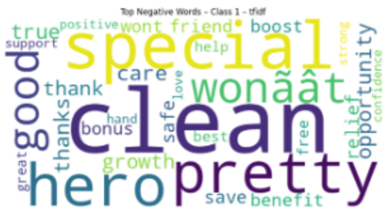


Figure 50: TF-IDF Weak Words

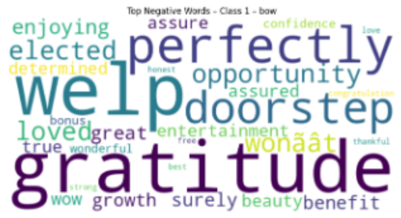


Figure 51: BoW Weak Words

Negative Tweets For negative tweets, the strongest positively associated words include “crisis”, “kill”, “panic”, “fear”, “died”, “hell”, “scam”, and “war”. These terms directly relate to danger, death, or societal instability—central themes in negative sentiment.

Conversely, negatively correlated words for this class (i.e., features that strongly indicate not being negative) are “help”, “love”, “support”, “safe”, “thank”, “great”, and “best”, all of which convey comfort, appreciation, or protection—values that contradict negative sentiment

10.1.3 Neutral Tweets

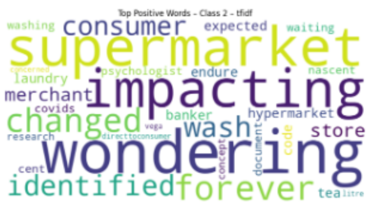


Figure 52: TF-IDF Strong Words

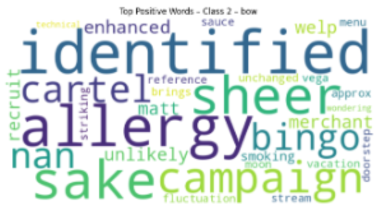


Figure 53: BoW Strong Words

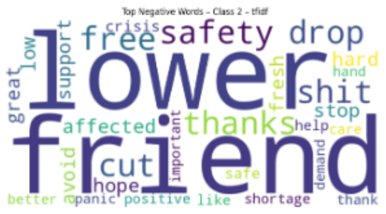


Figure 54: TF-IDF Weak Words

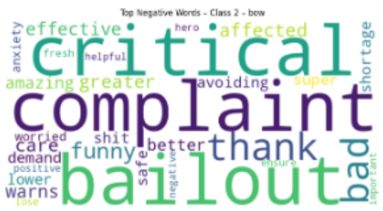


Figure 55: BoW Weak Words

Neutral Tweets For neutral tweets, the positively weighted words are generally informational or technical, such as “research”, “identified”, “merchant”, “consumer”, “store”, “document”, “code”, “technical”, and “nascent”. These terms lack emotional charge and reflect observation, analysis, or fact-reporting, which is appropriate for neutrality.

The lowest-weighted (most non-neutral) words include “positive”, “safe”, “thank”, “crisis”, “panic”, “support”, and “fear”—all of which carry strong emotional connotation, whether positive or negative, thus steering the classifier away from labeling them as neutral.

Overall, the classifiers successfully identify emotionally charged vocabulary for polarized classes, while selecting more analytical or context-based terms for neutrality. The results are coherent with expected sentiment structures and highlight that neutral tweets tend to revolve around technical, observational, or objective language rather than emotional expression

11 Conclusions

In this project, we tackled the challenge of classifying the sentiment of tweets related to COVID-19 into three categories: positive, negative, and neutral. Our goal was to explore a wide range of preprocessing, vectorization techniques, and classification models to understand what works best for this real-world, noisy text data.

11.1 What Worked Well

The best performing classical model was Support Vector Machine (SVM), particularly when paired with Bag-of-Words (BoW), achieving over 80% test accuracy. Logistic Regression also showed strong results with both BoW and TF-IDF, highlighting that linear models can be very effective when combined with appropriate vectorizers.

The highest overall performance came from fine-tuning the BERT model, which reached nearly 89% accuracy. BERT's contextual understanding allowed it to outperform all traditional methods, especially in handling the minority class (neutral sentiment).

11.2 What Didn't Work Well

K-Nearest Neighbors (KNN) consistently underperformed, especially with sparse vector spaces like TF-IDF and BoW. Its reliance on distance metrics in high-dimensional space proved ineffective due to the curse of dimensionality. AdaBoost also failed to generalize well, particularly with default weak learners and imbalanced classes.

Word2Vec (both trained and pretrained) generally gave lower results in classical models. While Google's pretrained vectors helped improve results, they still lacked the discriminative power compared to TF-IDF and BoW for linear models.

11.3 Insights and Challenges

- Class imbalance (especially with the neutral class) was a major challenge. Using class weighting helped mitigate this in linear models.
- Tweet length, metadata (like date and country), and word importance were investigated. Most metadata had minimal impact on model performance.
- Word clouds and weight analysis confirmed that sentiment classification is driven by emotionally charged vocabulary—many of which are consistent across models.

11.4 Traditional Models vs Deep Learning

Traditional models like SVM and Logistic Regression provided fast and interpretable baselines. However, BERT clearly surpassed them in all metrics, especially in recall for the underrepresented class. This highlights the power of transformer-based models even when trained on relatively small datasets.