Sentiment Analysis on Shopee Reviews

CMSC 197: Machine Learning Project

2 Contributors

Introduction

In the growing field of e-commerce, delivering an exceptional online shopping experience to the public is important. **Shopee** has played an important role in the growth of e-commerce in Southeast Asia. Its number of users has continually been increasing since its release. As students and as users of the app, we recognize the significance of understanding customer sentiments in aiming for customer satisfaction. For this project, we conduct a sentiment analysis for 6,000 **Shopee** reviews in the Google Play Store app.

The primary objective is to understand and categorize customer sentiments expressed in the reviews by distinguishing them between Positive and Negative.

Through the use of Natural Language Processing (NLP) techniques, we aim to extract the valuable insights of the users from their reviews that can help refine ang enhance the shopping experience for a diverse customer base.

Problem Statement

Shopee has played an important role in the growth of e-commerce in Southeast Asia. The mobile app has surpassed millions of users as of 2022 and the number of users are increasing till today.

As such, it is important to enhance Shopee's online shopping experience by identifying the sentiments of the customers and understanding the users' pain points.

why does it matter?

Identifying sentiments in customer reviews helps **Shopee** understand the experience of the users. Positive sentiments highlights the aspects that the customers appreciate and find amiable. Meanwhile negative sentiments point to areas where improvements may be needed.

It helps in enhancing the overall user experience.

It could also be used as a basis for a competitive analysis against **Shopee**'s leading competing brand. It provides them insights on how they are performing in relative to their competitors.

Data Collection

Dataset: BwandoWando. (2023).Shopee App Reviews from Google Store [Data set]. Kaggle. https://doi.org/10.34740/KAGGLE/DS/3960350

Data was collected from Kaggle where reviews were scraped from the Google Play Store between 2015 - 2023. The original dataset contained features such as:

- user reviews
- user ratings (1–5)
- review likes (the amount of users that agreed with a particular review)
- App Version
- Author Id
- Author Name
- Review Date

Data Collection

Only these were utilized for this sentiment analysis:

- user reviews
- user ratings (1–5)
- review likes (the amount of users that agreed with a particular review)

Data Cleaning

Data Cleaning

[9]:
```
shopee_df.shape
```

[9]:
```
(6049, 3)
```

[10]:
```
shopee_df[shopee_df.duplicated(['review_text'])].shape
```

[10]:
```
(1542, 3)
```

[11]:
```
# Drop duplicates as we only want unique reviews
shopee_df.drop_duplicates(['review_text'], inplace=True)
```

[12]:
```
# Reindex the dataframe
shopee_df.reset_index(drop=True, inplace=True)
```

[13]:
```
shopee_df.shape
```

[13]:
```
(4507, 3)
```

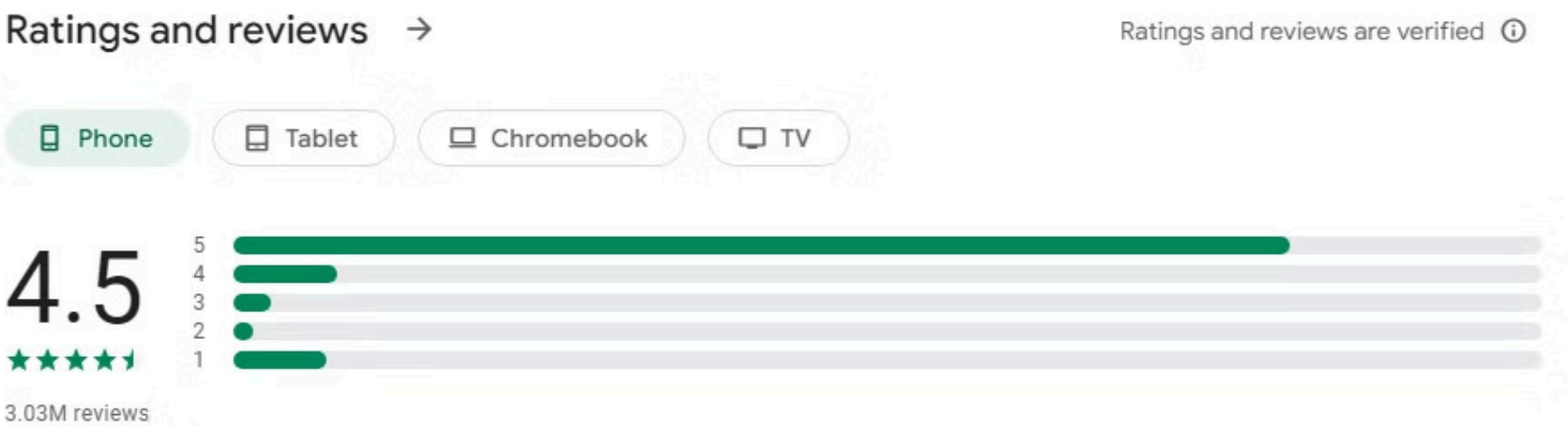
```
# Check how many reviews we have for each score after dropping the duplicates
shopee_df['review_rating'].value_counts().sort_index()
```

```
review_rating
1     705
2     124
3     177
4     301
5    3200
Name: count, dtype: int64
```

[ ]
```
shopee_df.columns
```

```
Index(['review_text', 'review_rating', 'review_likes'], dtype='object')
```

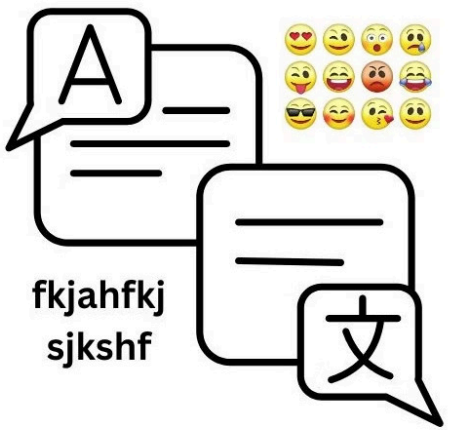
Target Variable



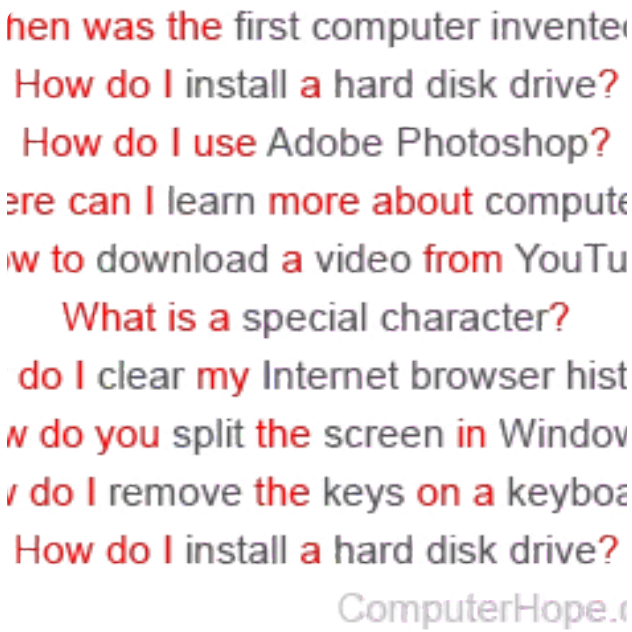
1-3 stars: **Negative sentiment**, class 1

4-5 stars: **Positive sentiment**, class 0

Key data pre-processing steps that improved accuracy



Removed gibberish words



Removed stopwords stemmed words to their root form using NLTK

Removed frequently occurring words that both appeared in positive and negative reviews

```
[ ] # This is the original text of the first review in our dataset
    shopee_df.loc[0]['review_text']
```

```
'napaka bulok an tagal mag load ng tracking order page aabot pa ata ng isang taon bago mag load bwisit, ayusin nyo naman to mga bugok'
```

```
# This is how the text looks like after stemming
shopee_df.loc[0]['content_stem']
```

```
'bulok tagal mag load track order page aabot ata taon mag load bwisit ayusin bugok'
```

```
[27]: # View reviews that do not have any meaningful words
      shopee_df[shopee_df['content_clean_len']==0]['review_text']
```

```
[27]: 38                                   😊😊😊😍😍😍
      80                                      👍🤞🤞
      134                  this app is not for me
      162                                 😍😍😍😍😍
      178                                  😋😋😋😋
                           ...
      4242                                 😁😁😁😁😁
      4371    🔰😑💞🐂👺😢💞🐂👺😆🐂😆😍🐂♓⏳♉♑🔝⏳🚰🔝♉♓🚹♉♓🚹
      4406                                 👏👏👏👏
      4436                                  😊😊😊
      4437                                  🥰😘
      Name: review_text, Length: 135, dtype: object
```

```
[28]: # Drop these reviews that do not have any meaningful words
      shopee_df = shopee_df.drop(shopee_df[shopee_df['content_clean_len']==0].index)
```

Tagalog Stopwords

```
]: # Load Filipino stopwords from a .txt file taken from
   # https://github.com/explosion/spaCy/blob/master/spacy/lang/tl/stop_words.py and
   # https://github.com/stopwords-iso/stopwords-tl

   # Initialize an empty list for Filipino stopwords
   filipino_stopwords = []

   # Load Filipino stopwords from the first .txt file
   with open('genediazjr-tagalog.txt', 'r', encoding='utf-8') as file1:
       filipino_stopwords.extend([line.strip() for line in file1])

   # Load Filipino stopwords from the second .txt file
   with open('stopwords-tl.txt', 'r', encoding='utf-8') as file2:
       filipino_stopwords.extend([line.strip() for line in file2])

   print(filipino_stopwords)
```

```
['ako', 'sa', 'akin', 'ko', 'aking', 'sarili', 'kami', 'atin', 'ang', 'aming', 'amin', 'ating', 'ka', 'iyong', 'iyo', 'inyong', 'siya', 'kanya', 'mismo',
'ito', 'nito', 'kanyang', 'sila', 'nila', 'kanila', 'kanilang', 'kung', 'ano', 'alin', 'sino', 'kanino', 'na', 'mga', 'iyon', 'am', 'ay', 'maging', 'nagi
ng', 'mayroon', 'may', 'nagkaroon', 'pagkakaroon', 'gumawa', 'ginagawa', 'ginawa', 'paggawa', 'ibig', 'dapat', 'maaari', 'marapat', 'kong', 'ikaw', 'tay
o', 'hindi', 'namin', 'gusto', 'nais', 'niyang', 'nilang', 'niya', 'huwag', 'ginawang', 'gagawin', 'maaaring', 'sabihin', 'narito', 'kapag', 'ni', 'nasaa
n', 'bakit', 'paano', 'kailangan', 'walang', 'katiyakan', 'isang', 'at', 'pero', 'o', 'dahil', 'bilang', 'hanggang', 'habang', 'ng', 'pamamagitan', 'par
a', 'tungkol', 'laban', 'pagitan', 'panahon', 'bago', 'pagkatapos', 'itaas', 'ibaba', 'mula', 'pataas', 'pababa', 'palabas', 'ibabaw', 'ilalim', 'muli',
'pa', 'minsan', 'dito', 'doon', 'saan', 'lahat', 'anumang', 'kapwa', 'bawat', 'ilan', 'karamihan', 'iba', 'tulad', 'lamang', 'pareho', 'kaya', 'kaysa',
'masyado', 'napaka', 'isa', 'bababa', 'kulang', 'marami', 'ngayon', 'kailanman', 'sabi', 'nabanggit', 'din', 'kumuha', 'pumunta', 'pumupunta', 'ilagay',
'makita', 'nakita', 'katulad', 'mahusay', 'likod', 'kahit', 'paraan', 'noon', 'gayunman', 'dalawa', 'tatlo', 'apat', 'lima', 'una', 'pangalawa', 'akin',
'aking', 'ako', 'alin', 'am', 'amin', 'aming', 'ang', 'ano', 'anumang', 'apat', 'at', 'atin', 'ating', 'ay', 'bababa', 'bago', 'bakit', 'bawat', 'bilan
g', 'dahil', 'dalawa', 'dapat', 'din', 'dito', 'doon', 'gagawin', 'gayunman', 'ginagawa', 'ginawa', 'ginawang', 'gumawa', 'gusto', 'habang', 'hanggang',
'hindi', 'huwag', 'iba', 'ibaba', 'ibabaw', 'ibig', 'ikaw', 'ilagay', 'ilalim', 'ilan', 'inyong', 'isa', 'isang', 'itaas', 'ito', 'iyo', 'iyon', 'iyong',
'ka', 'kahit', 'kailangan', 'kailanman', 'kami', 'kanila', 'kanilang', 'kanino', 'kanya', 'kanyang', 'kapag', 'kapwa', 'karamihan', 'katiyakan', 'katula
d', 'kaya', 'kaysa', 'ko', 'kong', 'kulang', 'kumuha', 'kung', 'laban', 'lahat', 'lamang', 'likod', 'lima', 'maaari', 'maaaring', 'maging', 'mahusay', 'm
```

```
# Adding on stopwords that were appearing frequently in both positive and negative reviews
stops.update(['app','shopee','shoppee','item','items','seller','sellers'])
```

Create Train & Test Dataset

3497 reviews for training and **875** reviews in our test set.

The class representation is consistent across the train and test set

- **77%** of the data belonging to class 0 (positive sentiment)
- **23%** belonging to class 1 (negative sentiment)

```
[ ]  # As we would like to stratify our target variable, we will need to first assign X and y
     X = shopee_df[[cols for cols in shopee_df.columns if cols != 'target']]
     y = shopee_df['target']
```

```
[ ]  # Perform a train_test_split to create a train and test set
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
```

```
[ ]  # Merge X_train and y_train back together using index
     train = pd.merge(X_train, y_train, left_index=True, right_index=True)

     # Merge X_test and y_test back together using index
     test = pd.merge(X_test, y_test, left_index=True, right_index=True)
```

```
[ ]  # Reindex the train and test set
     train.reset_index(drop=True, inplace=True)
     test.reset_index(drop=True, inplace=True)
```

```
     # 3497 documents in our training set
     train.shape
```

```
     (3497, 6)
```

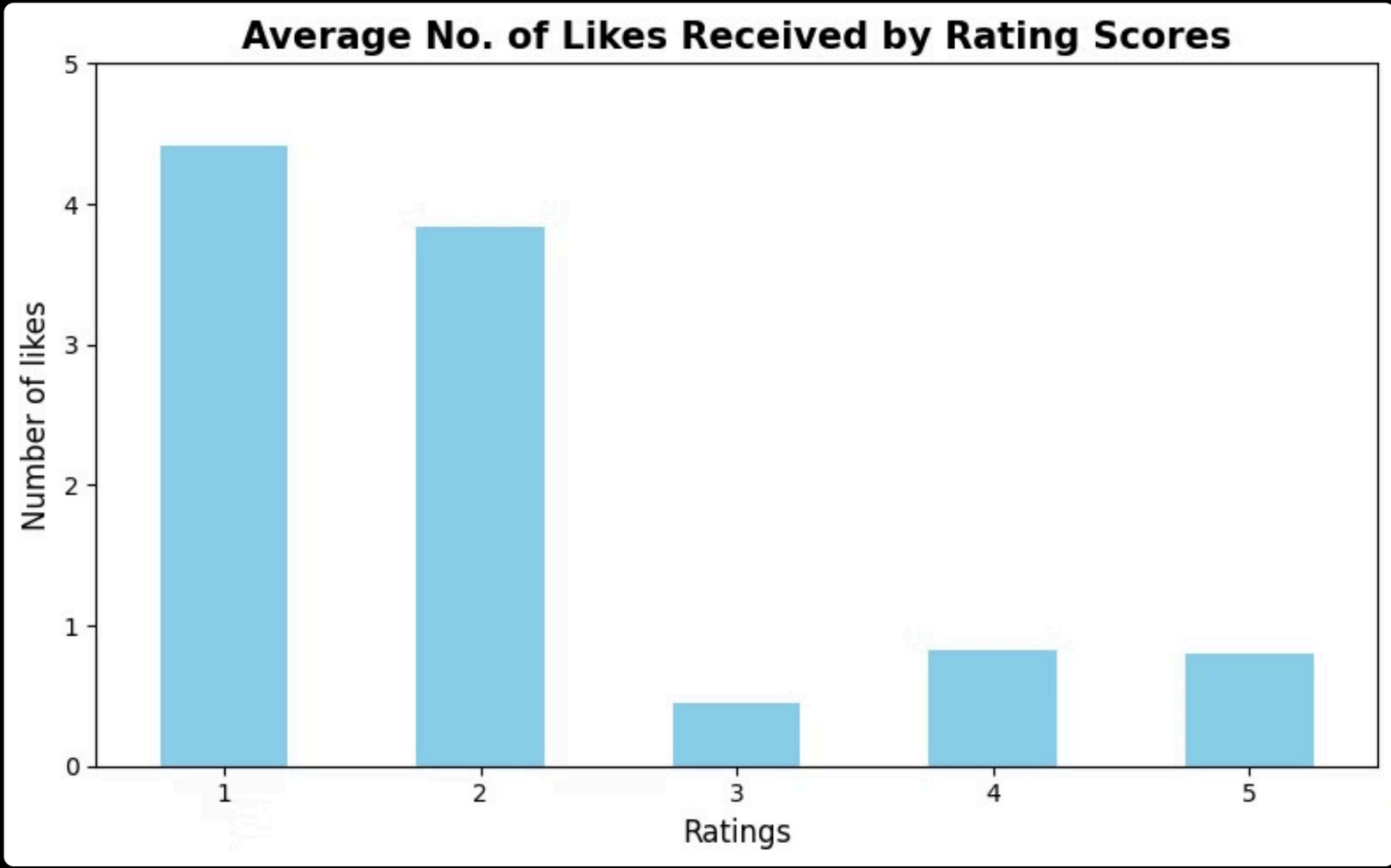
```
     # 875 documents in our test set
     test.shape
```

```
     (875, 6)
```

Exploratory Data Analysis

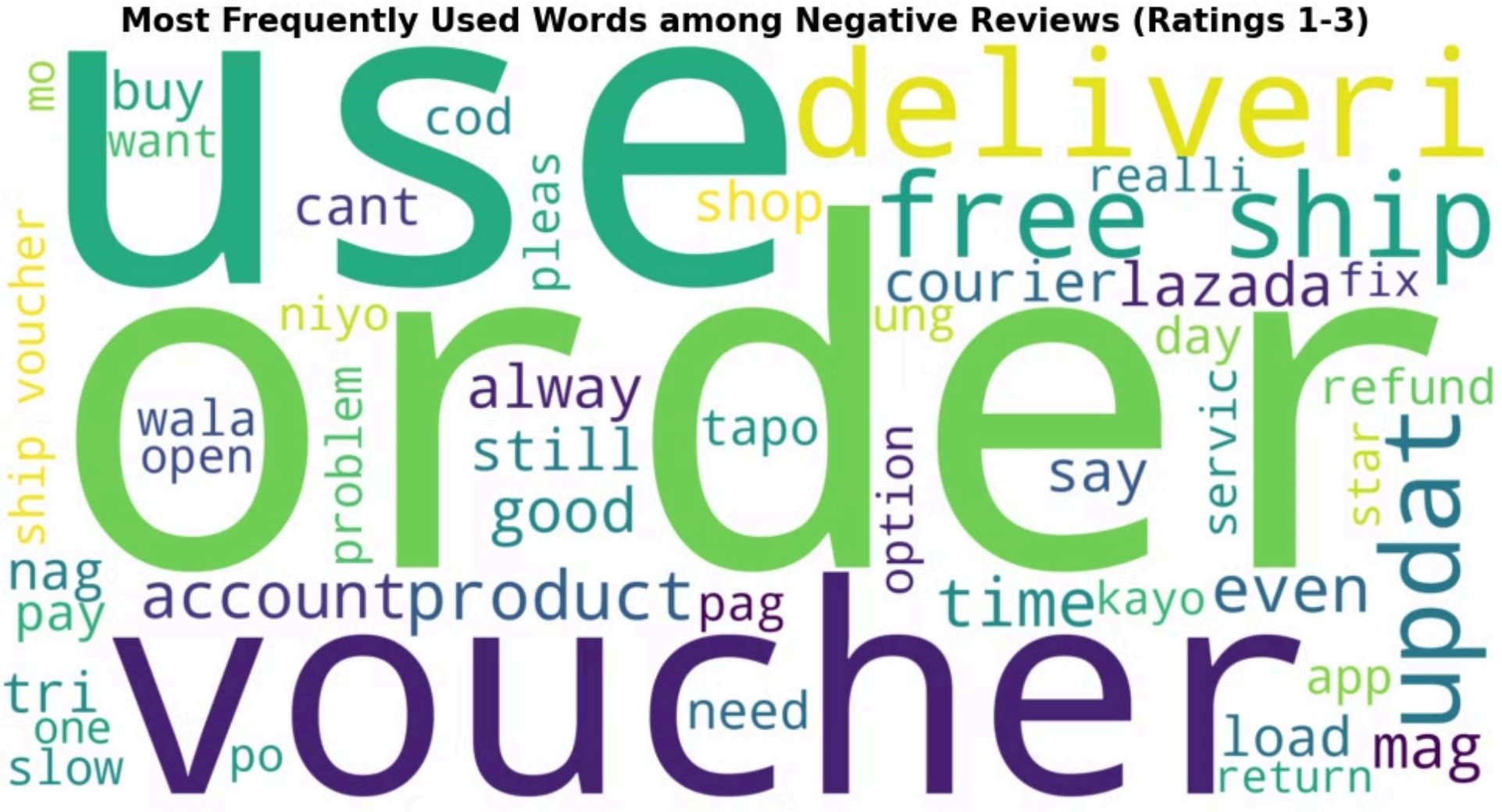
A review of **1 star** has the **highest** likes received.

5 star reviews has the **lowest** likes



Negative reviews with 1 or 2-star ratings receive more thumbs up on average, than positive reviews. This may suggest that several others face the same issues as those who have written these negative reviews.

Word cloud of the **50 most frequently occurring words** among **negative reviews**



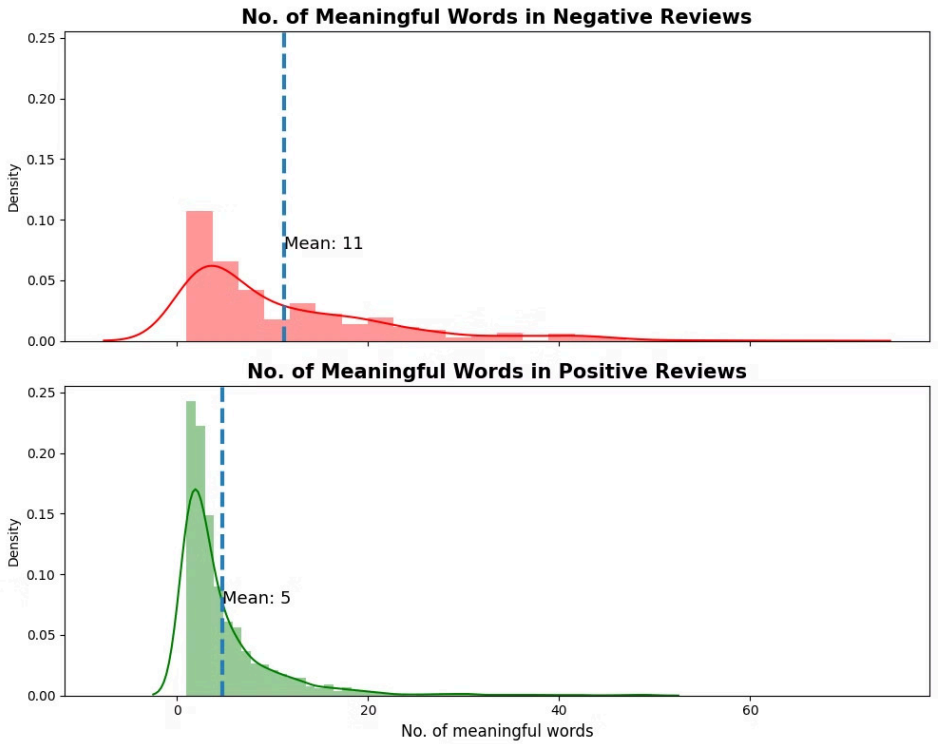
A **negative** review with the word **''Order'**

Word cloud of the 50 most frequently occurring words among positive reviews

"good", "nice", & "thank"

A **positive** review with the word "good'

Number of Meaningful Words



Distribution: **Right Skewed**

The average number of meaningful words in a negative review (11 words) is higher than that in a positive review (5 words).

Negative Reviews have higher variance in the number of meaningful words meaning **dissatisfied customers** are more likely to write **longer reviews.**

Data Dictionary

Feature	Type	Description
review_text	obj	Raw text containing user reviews
content_stem	obj	Pre-processed text for modeling
review_rating	int	No. of star ratings the user gave (1-5)
target	int	Target variable Postive sentiment: 0 Negative sentiment: 1

Pre Modeling

```
[ ] # Perform train test split so that we can train, score and tune our models' hyperparameters
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
```

```
[ ] X_train.shape
```

```
(2797,)
```

```
[ ] X_val.shape
```

```
(700,)
```

```
[ ] # Assuming X_train and X_val are your training and validation sets
    X_train = X_train.dropna()
    y_train = y_train.loc[X_train.index]

    X_val = X_val.dropna()
    y_val = y_val.loc[X_val.index]
```

```
[ ] X_train = X_train.fillna('')
    X_val = X_val.fillna('')
```

```
[ ] # Use count vectorizer to check how many unique words there are
    cvec = CountVectorizer(stop_words='english')
    cvec_df = pd.DataFrame(cvec.fit_transform(X_train).todense(), columns=cvec.get_feature_names())
    cvec_df.shape
```

```
    # Write a function that takes in the actual y value and model predictions,
    # and prints out the confusion matrix and classification report
    # Dataset: Validation or test set

    def cmat(actual_y, predictions, dataset):

        # Create a classification report
        print('Classification report for', dataset)
        print(classification_report(actual_y, predictions))
        print('')

        # Create a confusion matrix
        cm = confusion_matrix(actual_y, predictions)
        cm_df = pd.DataFrame(cm, columns=['Predicted Positive Review','Predicted Negative Review'], index=['Actual Positive Review', 'Actual Negative Review'])
        print('Confusion matrix for', dataset)
        print(cm_df)
```

Modeling

Bag of Words (BoW) representation to extract features from the text.

Done through vectorization, specifically the **CountVectorizer and TF-IDF Vectorizer.**

CountVectorizer tokenizes and **counts the word occurrences** in the corpus.

Tokenization: **splits** the text into **single words**

1	2
“This is a nice app”	**[“This”, “is”, “a”, “nice”, “app"].**

Vectorization simply substitutes a **number** each time **a new word appears**

1	2
[“This”, “is”, “a”, “nice”, “app"].	**[0,1,2,3,4]**

TF-IDF tells us **which words are important** to one document, relative to all other documents.

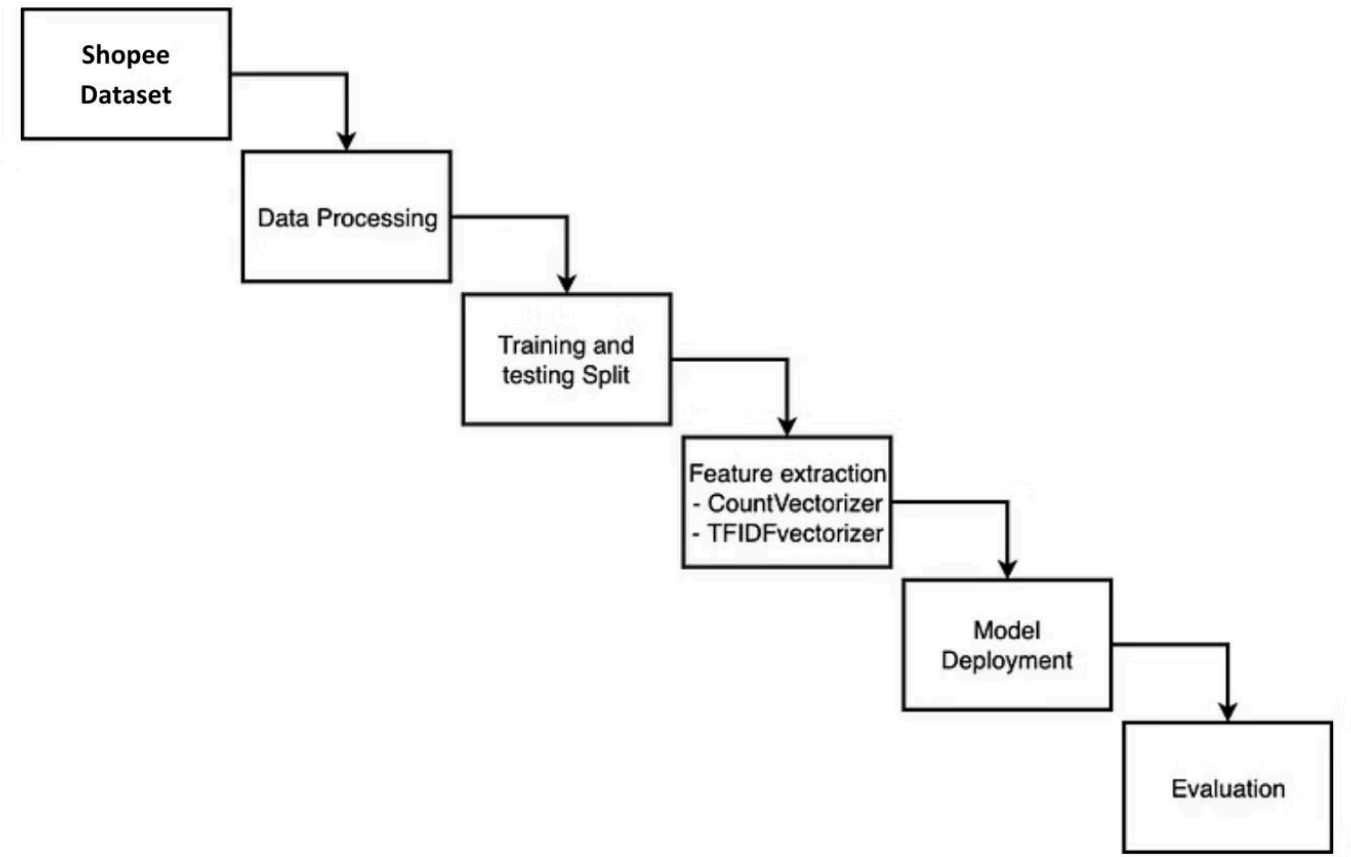
After vectorizing, we will fit a **Logistic Regression**, **Naive Bayes** and **Support Vector Machine** on the training data and evaluate the models' performance on the validation set.

Modeling (6k Dataset)

	Accuracy on Training	Accuracy on Validation	Accuracy on Recall
Voting Classifier (TF-IDF Logistic Regression & TF-IDF Naive Bayes	0.908	0.868	0.51
TF-IDF & SVC	0.789	0.781	0.03
Count Vectorizer & Naïve Bayes	0.888	0.878	0.64
TF-IDF & Logistic Regression	0.902	0.858	0.48
TF-IDF & Naïve Bayes	**0.907**	**0.871**	**0.54**
Count Vectorizer & Logistic Regression	0.853	0.838	0.32
Count Vectorizer & SVC	0.872	0.842	0.36

TF-IDF & Naïve Bayes was selected as our production model as it achieved the highest accuracy and recall on the validation set.

TF-IDF & Naïve Bayes



1. Create a Pipeline with TF-IDF Vectorizer and Naive Bayes
    - **'tvec':** TF-IDF Vectorizer (`TfidfVectorizer`) with stop words removed.
    - **'nb':** Multinomial Naive Bayes classifier (`MultinomialNB`).
2. Define Hyperparameter Grid for Grid Search

```
[ ]  # Create a pipeline with TF-IDF and Naive Bayes
     pipe_tvec_nb = Pipeline([
         ('tvec', TfidfVectorizer(stop_words='english')),
         ('nb', MultinomialNB())
     ])

     # Search over the following values of hyperparameters:
     pipe_tvec_nb_params = {
         'tvec__max_features': [500], # Maximum number of features (vocabulary size)
         'tvec__min_df': [2,3], # Minimum document frequency for a word to be included
         'tvec__max_df': [.9,.95], # Maximum document frequency for a word to be included
         'tvec__ngram_range':[(1,1),(1,2)],   # Range for n-grams to be extracted
     }
```

Overall, this code is performing **hyperparameter tuning** using grid search for a Naive Bayes classifier with TF-IDF vectorization on text data. It aims to find the best combination of hyperparameters that maximizes performance on the validation set.

```
# Instantiate GridSearchCV
gs_tvec_nb = GridSearchCV(pipe_tvec_nb, # Objects to optimise
                          param_grid = pipe_tvec_nb_params, # Hyperparameters for tuning
                          cv=10) # 10-fold cross validation

# Fit model on to training data
gs_tvec_nb.fit(X_train, y_train)

# Generate predictions on validation set
tvec_nb_pred = gs_tvec_nb.predict(X_val)
```

Modeling (6k Dataset)

	Accuracy on Training	Accuracy on Validation	Accuracy on Recall
Voting Classifier (TF-IDF Logistic Regression & TF-IDF Naive Bayes	0.908	0.868	0.51
TF-IDF & SVC	0.789	0.781	0.03
Count Vectorizer & Naïve Bayes	0.888	0.878	0.64
TF-IDF & Logistic Regression	0.902	0.858	0.48
TF-IDF & Naïve Bayes	**0.907**	**0.871**	**0.54**
Count Vectorizer & Logistic Regression	0.853	0.838	0.32
Count Vectorizer & SVC	0.872	0.842	0.36

TF-IDF & Naïve Bayes was selected as our production model as it achieved the highest accuracy and recall on the validation set.

Evaluate Production Model on Test Set

```
[ ] # Read test set into a dataframe
    test = pd.read_csv('clean_test2.csv')
```

```
[ ] # There are 870 documents in our test set
    test.shape
```

```
(875, 4)
```

```
# The class representation in our test set looks similar to our training set as we used stratify
test['target'].value_counts(normalize=True)
```

```
target
0    0.774857
1    0.225143
Name: proportion, dtype: float64
```

Evaluate Production Model on Test Set

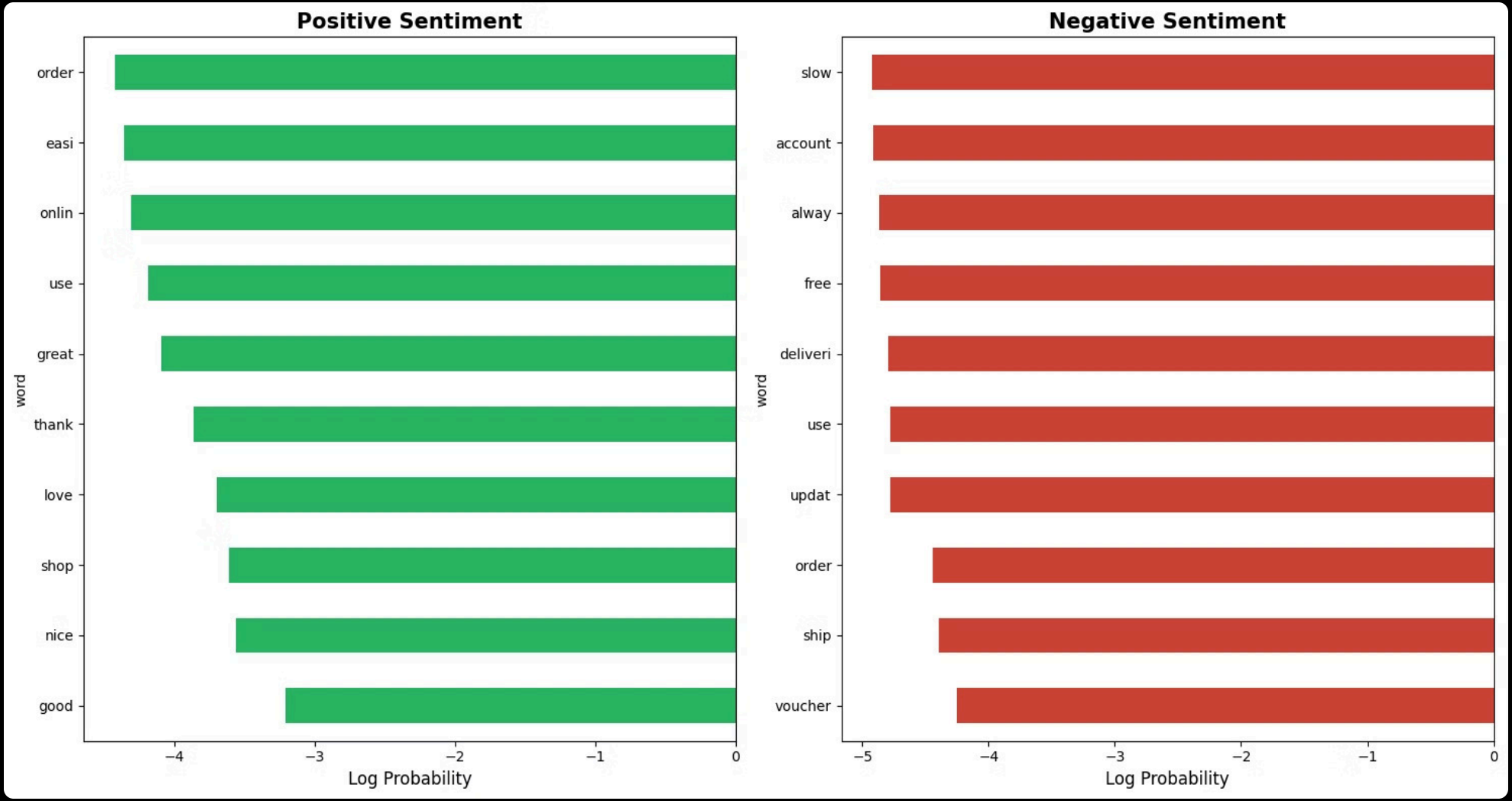
```
[ ]  # Establish our X and y variables
     X_test = test['content_stem']
     y_test = test['target']

[ ]  # Generate predictions on test set
     test_pred = gs_tvec_nb.predict(X_test)
```

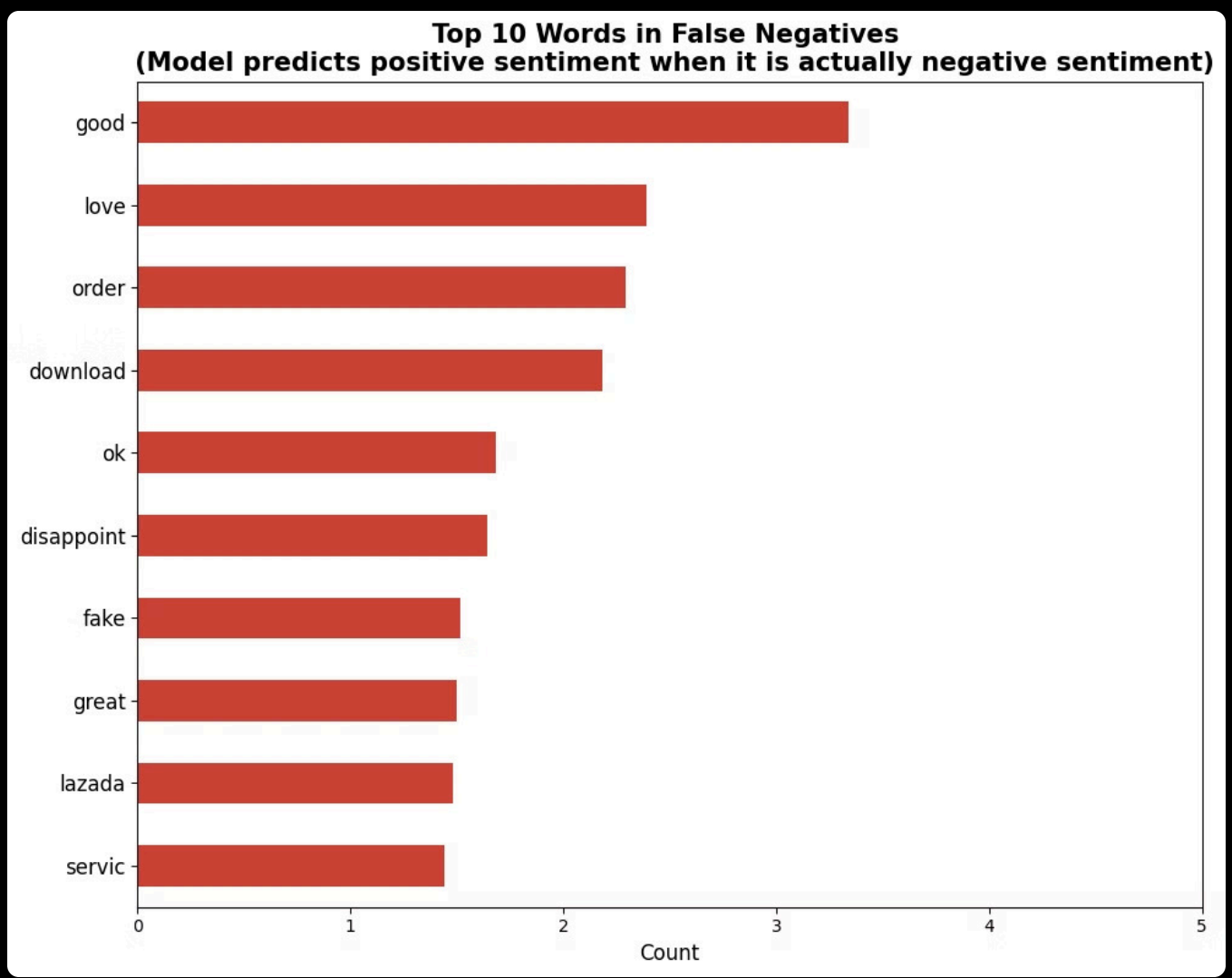
Production Model	Accuracy on Test Set	Recall on Test Set
TF-IDF & Naïve Bayes	0.875	0.56

Naive Baye's Model

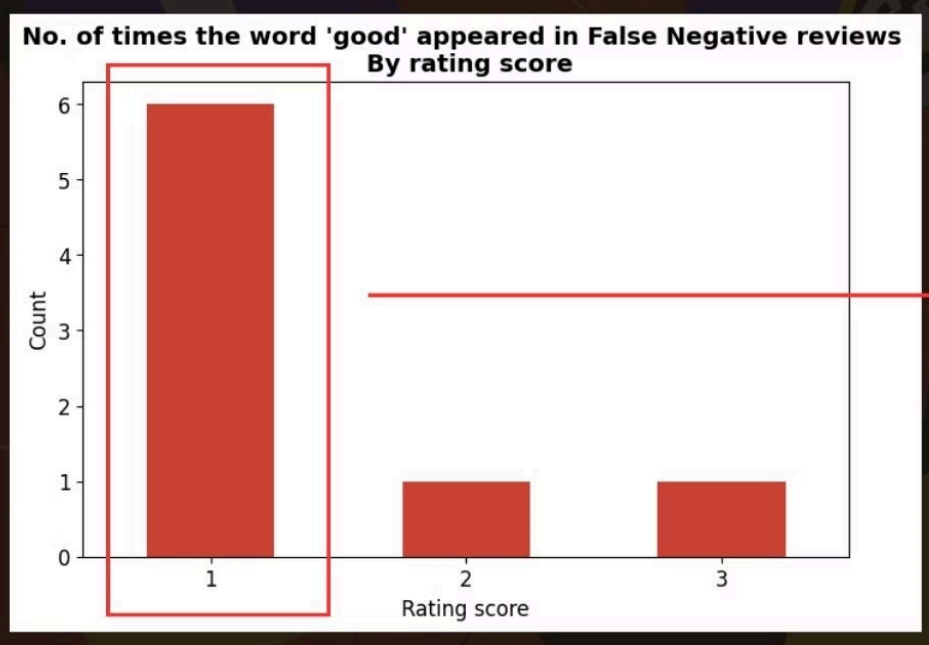
Most predictive words for each sentiment



Limitations: Misclassifications tend to occur when users write mixed reviews



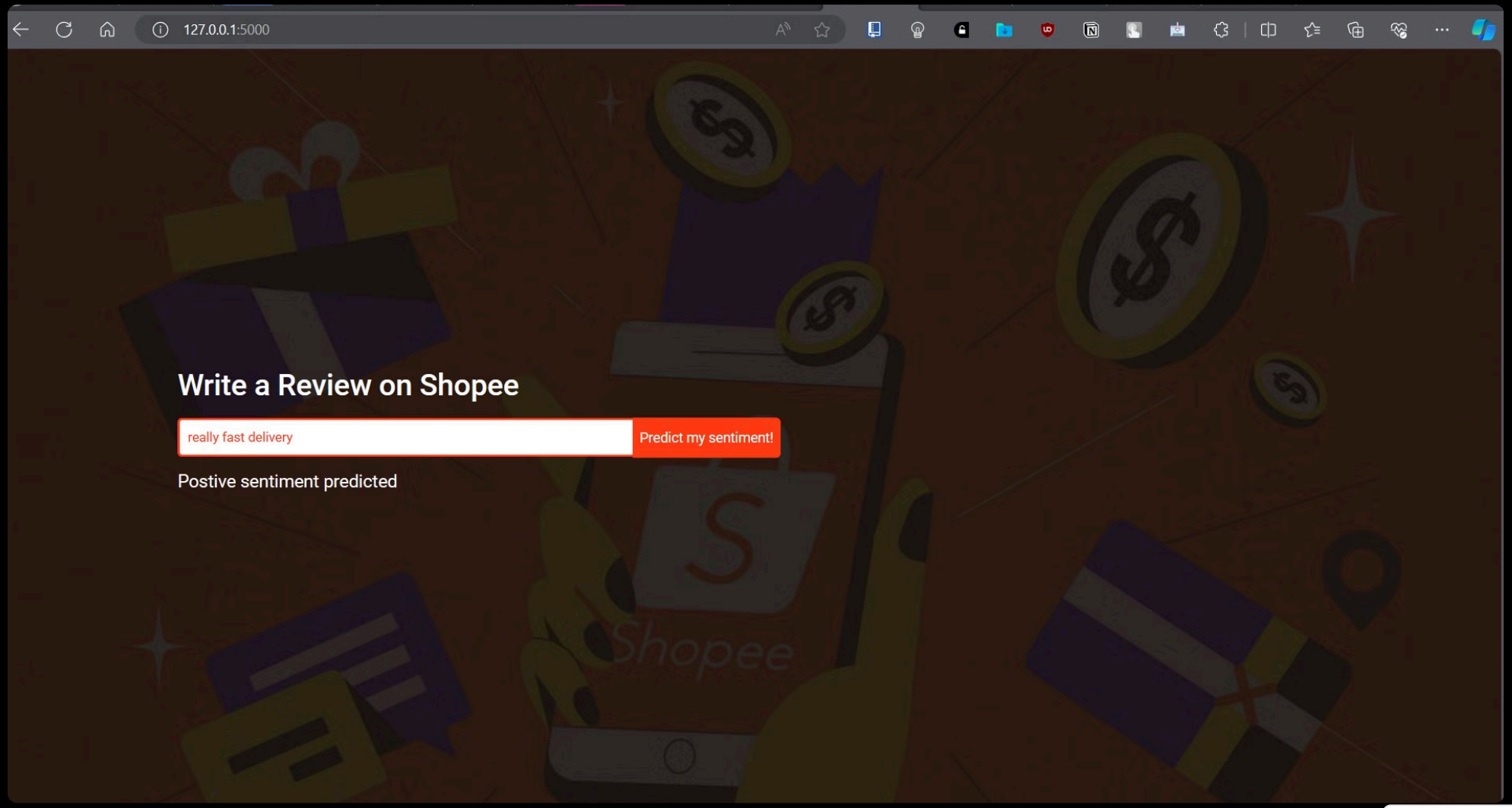
Sample False Negative that has the words **'good'** in it



"good in terms of items but in terms of quality of service? shopee itself... #poor"

Error uploading image.

Demo



Conclusion