**Stock Market Sentiment Analysis Using Preprocessing Techniques and Naive Bayes Classification**

**Introduction**

In this project, stock market sentiment data was gathered from multiple Twitter handles that focus on economic news. The collected tweets were manually categorized into two sentiment classes: positive (1) and negative (-1). The dataset consists of 2,106 negative samples and 3,685 positive samples. To prepare the data for analysis, various word normalization techniques such as case folding, contraction, tokenization, stop word removal, punctuation removal, Stemming and Lemmatization were applied. After preprocessing, a Multinomial Naive Bayes classifier was implemented to predict the sentiment of new, unseen tweets. This approach aims to efficiently classify economic news sentiments, which can assist in stock market trend analysis and decision-making.

**Dataset Description**

We worked with the Stock Market Sentiment Dataset from Kaggle. This dataset has around 5800+ entries. Each entry contains a stock-related text like a news headline or a tweet. Each text is labeled with a sentiment: Positive, Negative, or Neutral.

The goal of using this dataset is to predict the sentiment behind stock market news and social media posts. It helps to understand how people feel about the market and different companies.

The dataset has two main columns:

* **Text**: Stock-related news or posts.
* **Sentiment**: The sentiment label.

**Task Implementation**

**1. Mount Google Drive and Read Dataset**

We connected our Google Drive to Colab and loaded the dataset using pandas. This way we can use the file in our code.

from google.colab import drive

drive.mount('/content/drive')

stock\_df = pd.read\_csv("/content/drive/MyDrive/stock\_data.csv")

**2. Case Folding**

We changed all letters in the text to small letters. This helps to make the data uniform because "Stock" and "stock" will be treated the same.

stock\_df['Text\_nlp'] = stock\_df['Text'].str.lower()

**3. Expand Contractions**

We expanded short forms like "can't" to "cannot". This helps the computer understand the text better.

import contractions

stock\_df['Text\_nlp'] = stock\_df['Text\_nlp'].apply(contractions.fix)

**4. Punctuation Removal**

We removed symbols like commas, periods, and underscores. This makes the text cleaner and easier to process.

import re

stock\_df['Text\_nlp'] = stock\_df['Text\_nlp'].apply(lambda x: re.sub(r"[\_-]", " ", x))

def punctuationRemoved(x):

pRemoved = []

for word in x:

if word not in string.punctuation:

pRemoved.append(word)

return pRemoved

**5. Tokenization**

We split the sentences into individual words. This makes it easier to work with each word separately.

from nltk.tokenize import word\_tokenize

stock\_df['Tokens'] = stock\_df['Text\_nlp'].apply(word\_tokenize)

**6. Stop Words Removal**

We removed very common words like "the", "is", "on", etc. These words do not give much useful information for sentiment analysis.

from nltk.corpus import stopwords

stop\_words = set(stopwords.words('english'))

def stopWordRemove(x):

swRemoved = []

for word in x:

if word not in stop\_words:

swRemoved.append(word)

return swRemoved

stock\_df['Tokens'] = stock\_df['Tokens'].apply(stopWordRemove)

**7. Synonym Substitution (Optional Step)**

Synonym substitution is replacing a word with another word that has a similar meaning. We did not apply synonym substitution directly in this project, but it can make models better by replacing words like "buy" with "purchase", making the data more uniform.

**8. Stemming**

We reduced words to their base form. For example, "playing", "played", "plays" become "play". This reduces the number of unique words.

from nltk.stem import PorterStemmer

stemmer = PorterStemmer()

def stemming(x):

wStemming = []

for word in x:

wStemming.append(stemmer.stem(word))

return wStemming

stock\_df['Tokens'] = stock\_df['Tokens'].apply(stemming)

**9. Lemmatization**

We further processed the words to make them their dictionary form. It is smarter than stemming. For example, "better" becomes "good".

from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

def lemmatization(x):

lemmaWord = []

for word in x:

lemmaWord.append(lemmatizer.lemmatize(word))

return lemmaWord

stock\_df['Tokens'] = stock\_df['Tokens'].apply(lemmatization)

**10. Vector Semantics (TF-IDF Vectorization)**

We changed the words into numbers because machine learning models work with numbers. TF-IDF measures how important a word is in the text.

from sklearn.feature\_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer()

stock\_df['Processed\_Text'] = stock\_df['Tokens'].apply(lambda x: ' '.join(x))

X = vectorizer.fit\_transform(stock\_df['Processed\_Text'])

y = stock\_df['Sentiment']

**11. Model Training with Multinomial Naïve Bayes**

We split the data into training and testing sets. Then, we trained the Naïve Bayes model to learn from the data.

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

from sklearn.naive\_bayes import MultinomialNB

model = MultinomialNB()

model.fit(X\_train, y\_train)

**12. Model Evaluation**

We tested the model to see how well it worked. We also made a confusion matrix to show results.

from sklearn.metrics import classification\_report, confusion\_matrix

import matplotlib.pyplot as plt

import seaborn as sns

y\_pred = model.predict(X\_test)

print(classification\_report(y\_test, y\_pred))

Classification Report:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Metric** | **Class -1** | **Class 1** | **Accuracy** | **Macro Avg** | **Weighted Avg** |
| Precision | 0.84 | 0.72 |  | 0.78 | 0.76 |
| Recall | 0.34 | 0.96 |  | 0.65 | 0.73 |
| F1-Score | 0.49 | 0.82 |  | 0.65 | 0.70 |
| Support | 427 | 732 | 1159 | 1159 | 1159 |
| Accuracy |  |  | 0.73 |  |  |

Confusion Matrix:

cm = confusion\_matrix(y\_test, y\_pred)

plt.figure(figsize=(6,4))

sns.heatmap(cm, annot=True, fmt='.1f', cmap='Greens')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

A graph showing a number of confusion matrix

AI-generated content may be incorrect.

Fig: Confusion Matrix

The confusion matrix shows that the model is very good at finding class 1. Out of 732 real class 1 examples, it got 703 correct and only made 29 mistakes. But for class -1, the model has more problems. Out of 427 real class -1 examples, it only got 147 correct and made 280 mistakes by calling them class 1. So, the model is much better at finding class 1 than class -1.

**Conclusion**

In this project, we successfully used several natural language processing techniques to prepare stock market-related text data for sentiment analysis. We applied steps like case folding, tokenization, stop words removal, punctuation removal, stemming, and lemmatization. After processing the text, we used TF-IDF to transform the data into numbers and trained a Multinomial Naïve Bayes model. Our model was able to classify the text into positive, negative, and neutral sentiments. This project helped us learn how different NLP methods work together to clean, process, and make predictions from real-world text data.