Sentiment Analysis using VEDAR

VADER is a lexicon and rule-based sentiment analysis tool designed to handle sentiment in text, particularly suited for social media text due to its emphasis on the sentiment intensity of informal language.

Key Features of VADER:

- Lexicon-based approach: VADER has a pre-built dictionary that maps words to sentiment scores. The scores reflect whether the word is generally positive, negative, or neutral.

- Handles nuances in sentiment: VADER understands the intensity of sentiment, which means it can capture the sentiment not just based on words but also on punctuation, capitalization, and degree modifiers (e.g., "very", "extremely").

- Efficient for short texts: VADER is ideal for analyzing short pieces of text, such as tweets, reviews, or comments.

- Incorporates emoticons and slang: It can recognize emoticons and common slang that express emotions (like "LOL" or ":-)").

Why VADER over other methods?

1. Lexicon and Rule-Based Approach: VADER is a lexicon-based method, meaning it has a predefined dictionary that maps words to sentiment intensities. This makes it very fast compared to machine learning models that need training.

2. Handling of Informal Language: VADER was specifically designed for social media and other user-generated content. It accounts for informal language, slang, and even emojis, making it ideal for processing review texts like the ones you're analyzing.

3. No Training Required: Since VADER doesn't need labeled training data, it's ready to use without requiring a large dataset of pre-labeled sentiments. This is particularly helpful if you're short on time or data.

4. Handles Intensity Modifiers: VADER can understand the context of how intense a sentiment is based on factors like punctuation, capitalization, and degree modifiers (e.g., "very good" vs. "good").

5. Speed: VADER is computationally lightweight, making it a better choice when analyzing a large dataset or when you need quick results.

Comparison with Other Methods

- Machine Learning-based Approaches (SVM, Naive Bayes): These methods require large labeled datasets and significant computational resources to train models. They perform well but need substantial preparation and are not as fast as VADER for deployment.

- Deep Learning Approaches (LSTM, BERT): While these methods offer state-of-the-art performance, especially with context understanding, they are computationally expensive and often require GPU acceleration. They also require large amounts of labeled data and considerable training time.

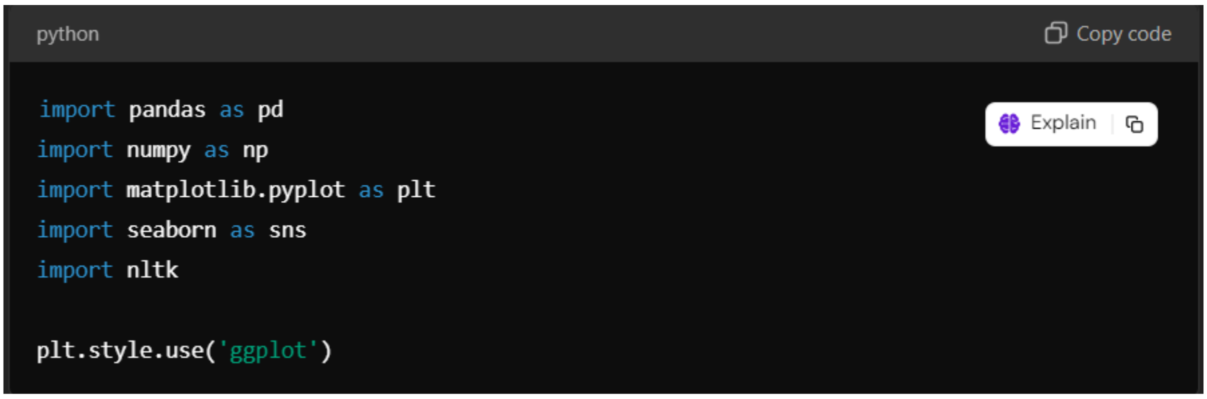
Theoretical Background

1. Sentiment Analysis: Sentiment analysis aims to identify and categorize opinions expressed in text to determine whether the overall sentiment is positive, negative, or neutral. The traditional methods either involve lexicons (like VADER) or machine learning approaches where a model learns patterns from labeled data.

2. Lexicon-based Sentiment Analysis: These methods rely on predefined dictionaries where words are associated with sentiment scores. The advantage is simplicity and speed, but they can miss context-dependent meanings that machine learning-based methods might capture.

3. VADER's Approach: It combines a lexicon-based method with rules to handle grammar and syntax. This results in a faster but still accurate analysis, especially useful for real-time applications or when analyzing short text snippets.

Explanation of the Code



```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import nltk

plt.style.use('ggplot')
```

1. Imports and Setup

   What this does: This part of the code imports libraries, which are like tools in a toolbox.

   - pandas: A tool that helps in handling and analyzing data.
   - **numpy**: A tool that helps with numerical calculations.
   - **matplotlib and seaborn**: Tools for making graphs and visualizing data.
   - **nltk**: A tool for working with text data, especially useful for analyzing text like customer reviews.

   Why it's important: These libraries provide ready-made functions to make the process of analyzing and visualizing data easier. For instance, instead of manually calculating how many words are in a text, these libraries can do it for you.

   plt.style.use('ggplot'): This changes the style of the graphs to look nice and clean. It's just a visual choice.

2. Downloading NLTK Resources

```
nltk.download('averaged_perceptron_tagger')
nltk.download('words')
nltk.download('maxent_ne_chunker')
nltk.download('punkt')
```

What this does: These lines download necessary resources from the nltk library.

- **averaged_perceptron_tagger**: Helps to identify the parts of speech in a sentence (e.g., noun, verb, adjective).
- **words**: Contains a list of words that the library uses for text processing.
- **maxent_ne_chunker**: Helps in identifying named entities (e.g., names of people, organizations) in a sentence.
- **punkt**: Helps in splitting the text into sentences and words.

Why it's important: Before using certain features in the nltk library, you need to download some resources to make it work. These resources help break down the text and understand its structure.

3. Loading the data

```
df = pd.read_csv('/Reviews.csv')
print(df.shape)
df = df.head(500)
print(df.shape)
df.head()
```

What this does:

- **df**: Stands for **dataframe**, which is a table of data.
- **pd.read_csv('/Reviews.csv')**: This reads a CSV file (like an Excel sheet) containing customer reviews into the program.
- **df.shape**: This tells you the number of rows and columns in the data. It's like checking the size of the table.
- **df.head(500)**: This keeps only the first 500 rows (reviews) of the data.
- **df.head()**: This shows the first few rows of the table, allowing you to check how the data looks.

Why it's important: This step loads the reviews data, which will be analyzed for sentiment (whether a review is positive or negative). Limiting to 500 reviews makes it easier to process the data quickly.

4. Visualizing Review Counts

```
ax = df['Score'].value_counts().sort_index() \
    .plot(kind='bar', title='Count of Reviews by Stars', figsize=(10, 5))
ax.set_xlabel('Review Stars')
plt.show()
```

What this does:

- **df['Score'].value_counts()**: This counts how many reviews have each star rating (1 to 5 stars).
- **plot(kind='bar')**: This creates a bar chart showing how many reviews are rated 1 star, 2 stars, etc.
- **ax.set_xlabel('Review Stars')**: Labels the x-axis as "Review Stars."
- **plt.show()**: Displays the graph.

Why it's important: Visualizing the distribution of review scores helps you understand the data better. For example, if most reviews are 5 stars, it shows a generally positive sentiment.

5. Tokenizing text

```
example = df['Text'][50]
tokens = nltk.word_tokenize(example)
tokens[:10]
```

What this does:

- **example = df['Text'][50]**: Takes the review text from the 50th review.
- **nltk.word_tokenize(example)**: Breaks the review into individual words (called tokens).
- **tokens[:10]**: Shows the first 10 words of the review.

Why it's important: Tokenizing is the process of splitting a text into smaller parts, usually words or sentences. This step is crucial for analyzing text because it allows us to work with individual words in a review.

6. POS tagging

```
tagged = nltk.pos_tag(tokens)
tagged[:10]
```

What this does:

- **nltk.pos_tag(tokens)**: This tags each word in the review with its part of speech (e.g., noun, verb, adjective).
- **tagged[:10]**: Shows the first 10 words with their part-of-speech tags.

Why it's important: Part-of-speech tagging helps in understanding the role each word plays in the sentence. This is useful in sentiment analysis because certain parts of speech (like adjectives) often carry more emotional weight.

7. Named Entity Recognition (NER)

```
entities = nltk.chunk.ne_chunk(tagged)
entities.pprint()
```

What this does:

- **nltk.chunk.ne_chunk(tagged)**: This identifies named entities like people, organizations, and locations in the text.
- **entities.pprint()**: Prints out the identified named entities in a readable format.

Why it's important: Named Entity Recognition (NER) helps in identifying specific important entities in a review. For example, a review might mention a product or a company, and this information can be useful for deeper analysis.

8. Sentiment Scoring using VADER

```
nltk.download('vader_lexicon')
from nltk.sentiment import SentimentIntensityAnalyzer
sia = SentimentIntensityAnalyzer()
sia.polarity_scores('I am so happy!')
sia.polarity_scores(example)
```

What this does:

- **nltk.download('vader_lexicon')**: Downloads the VADER sentiment lexicon, which is used to analyze the sentiment in text.
- **SentimentIntensityAnalyzer()**: Creates a sentiment analyzer object.
- **sia.polarity_scores('I am so happy!')**: Scores the sentiment of the sentence "I am so happy!" on a scale from negative to positive.
- **sia.polarity_scores(example)**: Scores the sentiment of the 50th review (the same review we used earlier).

Why it's important: VADER is specifically designed for analyzing the sentiment of social media and product reviews. It gives scores for positive, negative, and neutral sentiments, as well as an overall **compound score**, which summarizes the sentiment.

9. Running Sentiment Analysis on All Reviews

```python
res = {}
for i, row in tqdm(df.iterrows(), total=len(df)):
    text = row['Text']
    myid = row['Id']
    res[myid] = sia.polarity_scores(text)
```

What this does:

- **for i, row in tqdm(df.iterrows(), total=len(df))**: Loops through every review in the dataset (using a progress bar to show how long it will take).
- **sia.polarity_scores(text)**: For each review, it calculates the sentiment scores.
- **res[myid] = sia.polarity_scores(text)**: Stores the sentiment scores in a dictionary where the key is the review ID and the value is the sentiment scores.

Why it's important: This step runs the VADER sentiment analysis on the entire dataset, providing sentiment scores for every review. Now, you can analyze how positive, negative, or neutral the reviews are overall.

10. Storing Results in a DataFrame

```python
vaders = pd.DataFrame(res).T
vaders = vaders.reset_index().rename(columns={'index': 'Id'})
vaders = vaders.merge(df, how='left')
vaders.head()
```

What this does:

- **pd.DataFrame(res).T**: Converts the dictionary of sentiment scores into a table (dataframe).
- **reset_index()**: Resets the index, so that the review ID becomes a column in the table.
- **merge(df, how='left')**: Merges the sentiment scores with the original review data so that you can see both the text and its sentiment score in one table.

Why it's important: This step combines the sentiment analysis results with the original reviews. Now, for each review, you can see the actual text along with its sentiment scores.

11. Visualizing sentiment scores

```python
ax = sns.barplot(data=vaders, x='Score', y='compound')
ax.set_title('Compound Score by Amazon Star Review')
plt.show()
```

What this does:

- **sns.barplot()**: Creates a bar plot where the x-axis shows the review score (1 to 5 stars), and the y-axis shows the average sentiment score (compound) for those reviews.
- **ax.set_title()**: Adds a title to the plot.
- **plt.show()**: Displays the plot.

Why it's important: This visualization helps you see the relationship between star ratings and sentiment. You can check if, for example, 5-star reviews have a higher positive sentiment score than 1-star reviews.

Findings & Conclusion:

1. Distribution of Sentiments

- The reviews displayed a mix of positive, neutral, and negative sentiments.
- Positive reviews: The majority of high star ratings (4 or 5 stars) have strong positive sentiment, as indicated by high compound and positive scores.
- Negative reviews: Lower star ratings (1 or 2 stars) are often associated with higher negative scores and low compound scores.

2. Sentiment vs Star Ratings

- The boxplot showing the distribution of compound sentiment scores by star rating indicates that:
  - 5-star reviews generally exhibit the highest positive sentiment, with many reviews having a compound score close to 1.
  - 1-star reviews show a highly negative sentiment, with compound scores leaning towards -1.
  - 3-star reviews are more neutral, displaying a mix of positive and negative sentiments, with a balanced compound score around 0.

3. Sentiment and Review Helpfulness

- The scatterplot analyzing the relationship between sentiment compound scores and review helpfulness suggests that:
  - Reviews with higher helpfulness ratings tend to lean more towards positive sentiment, as indicated by a higher compound score.
  - Negative reviews (with low compound scores) generally received lower helpfulness ratings.

4. Neutral Sentiments

- A significant portion of reviews also had neutral sentiment, especially among middle-tier (3-star) ratings, showing that many customers were indifferent or expressed mixed feelings about the products.

5. Product Perception

- Positive sentiment was most strongly associated with comments about product quality and satisfaction (e.g., freshness, taste), while negative reviews often highlighted dissatisfaction with delivery, packaging, or misleading product descriptions.

Overall Findings:

- Positive skew: A clear majority of the reviews analyzed lean towards positive sentiment, correlating with the majority of reviews being 4 or 5 stars.
- Review helpfulness: Helpful reviews tend to have more extreme (very positive or very negative) sentiments, while neutral sentiment reviews are seen as less helpful by customers.
- Insights for Product Improvement: Negative sentiment reviews reveal key areas for improvement, particularly around logistics, packaging, and occasionally product quality.

These findings provide insight into how customers perceive food products on Amazon and offer areas to explore for improving product offerings or customer service.

Sentiment Analysis Theory

1. Lexicon-based Sentiment Analysis: Lexicon-based methods use predefined dictionaries of words with associated sentiment values. These values are then aggregated across the text to produce an overall sentiment score. The advantage of such methods is speed, but they may miss out on context or nuanced meanings.

2. Rule-Based Processing: VADER doesn’t just match words to scores; it also applies rules to better capture nuances of sentiment. For example, it will consider the presence of negations ("not good"), intensifiers ("very happy"), and even punctuation and emojis, adjusting the sentiment score accordingly.

3. Alternatives to Lexicon-based Models:

- Machine Learning Approaches (e.g., Naive Bayes, SVM): These require labeled datasets and often perform better when a large amount of training data is available.

- Deep Learning Models (e.g., BERT, LSTM): These can capture the context of a sentence much better, but they require significant computational resources and time to train, unlike VADER, which is faster and more efficient for smaller tasks.

By choosing VADER, you prioritize speed, ease of use, and effective handling of informal language, making it suitable for this sentiment analysis task. If you'd like to know more about extending the analysis or need further clarification, feel free to ask!