Hyperiondev

PROJECT ON

NLP Applications

BY: Yonas Ashagrie
29/02/2024

OBJECTIVE

The objective is to conduct a sentiment analysis and text similarity within a corpus of reviews for a range of products offered through Amazon. This refers to being able to discover semantics from the reviews, such as views of how they feel in relation to some product, and whether there are enormous similarities between sets of reviews emanating the same sentiments. These representations of knowledge will be classified based on their polarity, and sentiment analysis will be done to determine what category a particular review falls into: Positive, Negative, or Neutral. Finally, it is supposed to text similarity estimation, in order to estimate the degree of resemblance between pairs of reviews, with sentiment labels taken into consideration. Summing up, this project aims to assist companies in deriving actionable insights into decision-making processes—referring to product development, marketing strategy, and enhancing customer satisfaction—out of Amazon reviews based on sentiments expression.

SUMMARY OF DATA

Building on this, the following paper gives a presentation of sentiment analysis and an analysis of product information and user-generated reviews for the Amazon Kindle E-Reader 6" Wifi (8th Generation, 2016). The set includes structured data with the ASIN identification number down to the details of a product image: brand, manufacturer, categories, date added, date updated, and more. Besides, it contains some user-given reviews in text format that are accompanied by positive, negative comments, or complaints about ratings.

The data includes an overview of sentiments, preferences, and experiences of customers concerning the Kindle E-reader, for which sentiment analysis is possibly undertaken to have a view of overall customer satisfaction and hence find whether there are any opportunity areas existing.

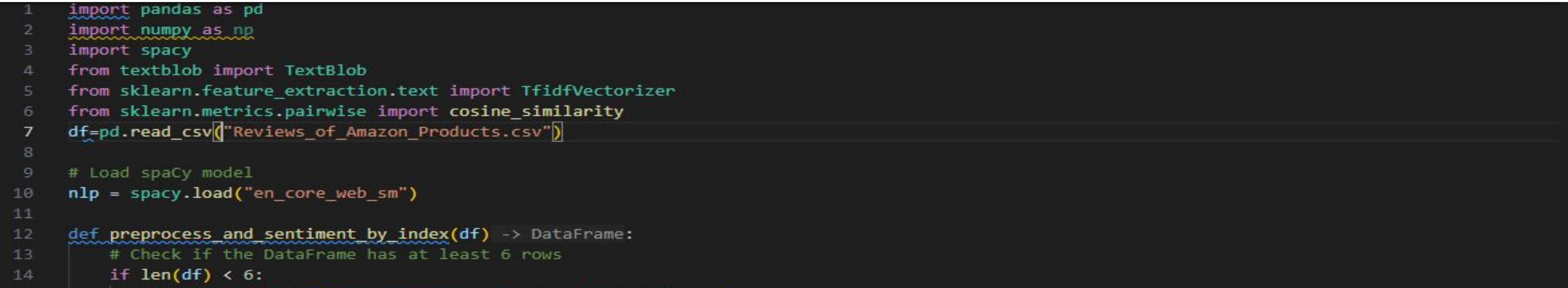
The data is most helpful in understanding customer sentiments about the product "Kindle E-Reader" so as to help decision-makers, primarily in product developmental processes, choice of marketing strategies, avenues for customer support initiatives, among others.

DATA CLEANING AND PREPROCESSING

- **LOADING LIBRARIES AND DATA :**

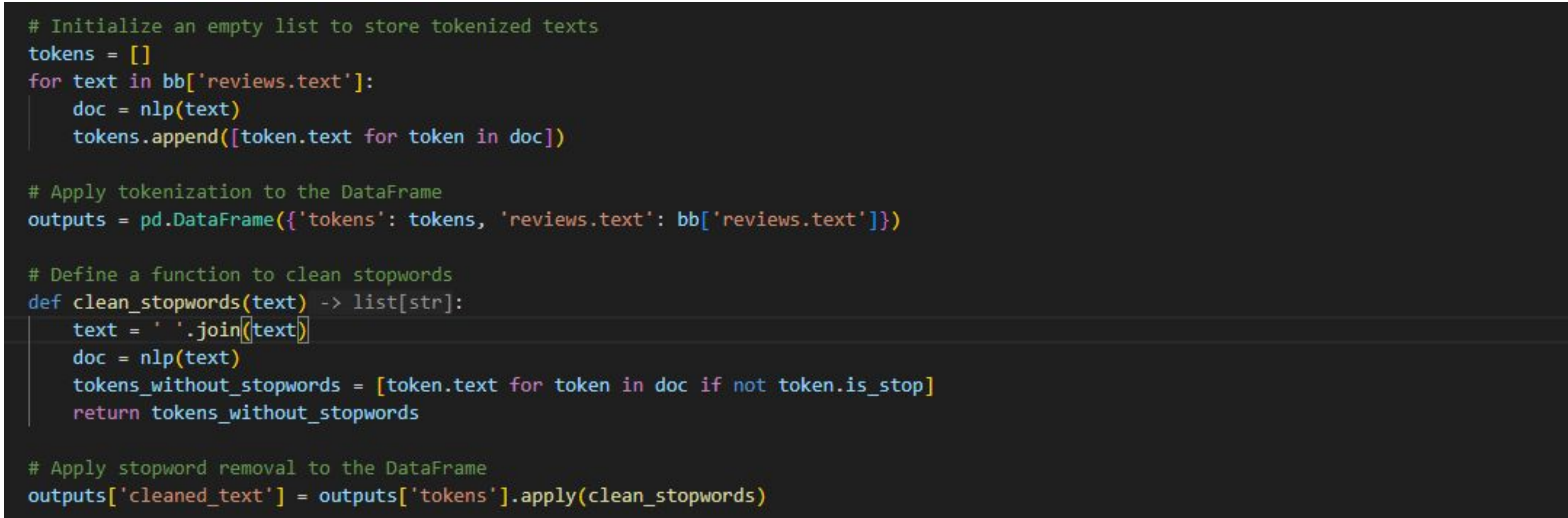
  Libraries such as Pandas, NumPy, spaCy, TextBlob, and scikit-learn are imported.
  The spaCy English language model ("en_core_web_sm") is loaded.
  Data is read from the CSV file "Reviews_of_Amazon_Products.csv" into a Pandas DataFrame named df.
  Rows with missing values in the 'reviews.text' column are dropped using the dropna()

  All preprocessing and sentiment analysis tasks are encapsulated within the preprocess_and_sentiment_by_index function. function to ensure only non-null reviews are considered for analysis.



```
import pandas as pd
import numpy as np
import spacy
from textblob import TextBlob
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
df=pd.read_csv("Reviews_of_Amazon_Products.csv")

# Load spaCy model
nlp = spacy.load("en_core_web_sm")

def preprocess_and_sentiment_by_index(df) -> DataFrame:
    # Check if the DataFrame has at least 6 rows
    if len(df) < 6:
```

- **TOKENIZATION:**
  Tokenization is performed using the spaCy library within the preprocess_and_sentiment_by_index function to break down each review text into individual tokens.
  Stop words are also removed during this process using a custom function named clean_stopwords.

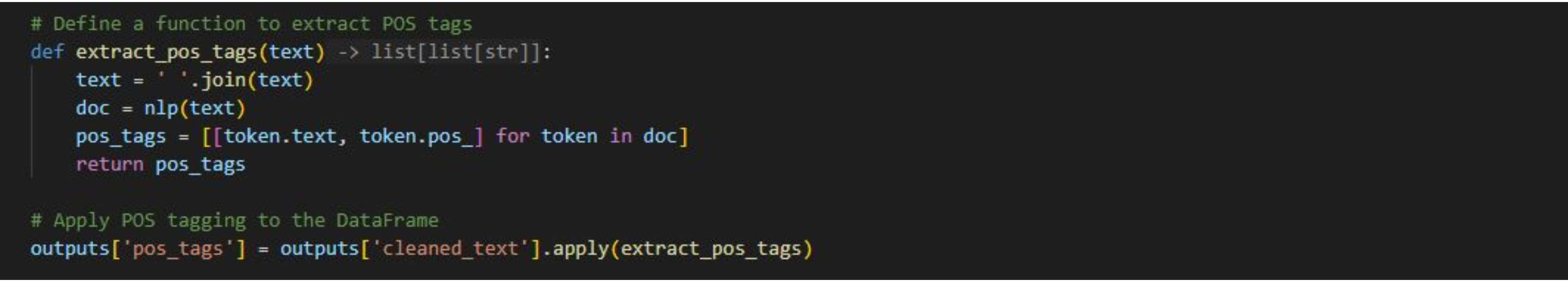


```python
# Initialize an empty list to store tokenized texts
tokens = []
for text in bb['reviews.text']:
    doc = nlp(text)
    tokens.append([token.text for token in doc])

# Apply tokenization to the DataFrame
outputs = pd.DataFrame({'tokens': tokens, 'reviews.text': bb['reviews.text']})

# Define a function to clean stopwords
def clean_stopwords(text) -> list[str]:
    text = ' '.join(text)
    doc = nlp(text)
    tokens_without_stopwords = [token.text for token in doc if not token.is_stop]
    return tokens_without_stopwords

# Apply stopword removal to the DataFrame
outputs['cleaned_text'] = outputs['tokens'].apply(clean_stopwords)
```

PART OF SPEECH (POS) TAGGING:

POS tagging is applied to the tokenized text using another custom function named extract_pos_tags.

This step provides insight into the syntactic structure of the text.

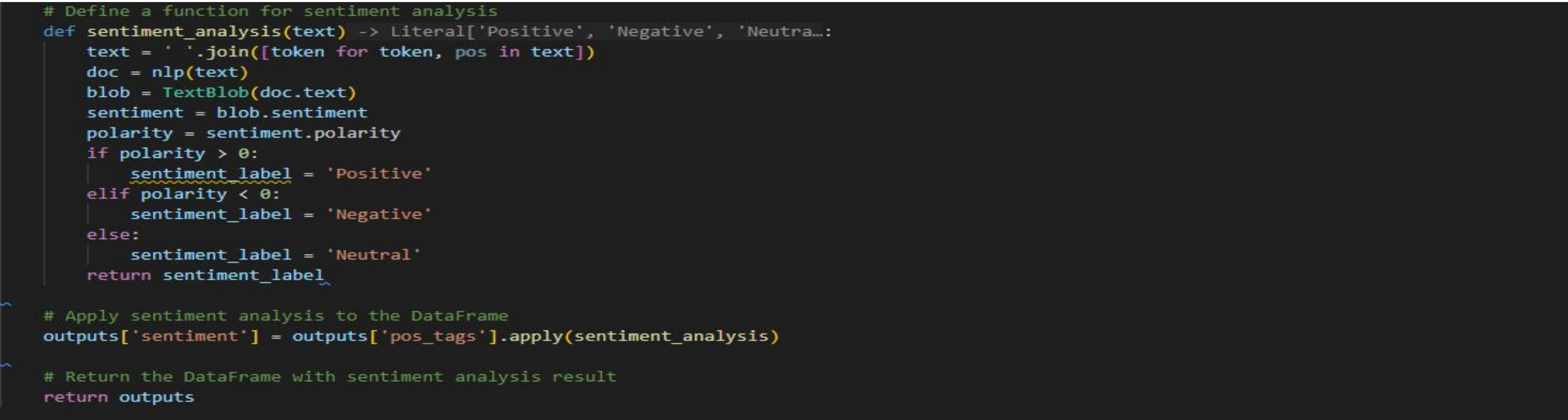


```python
# Define a function to extract POS tags
def extract_pos_tags(text) -> list[list[str]]:
    text = ' '.join(text)
    doc = nlp(text)
    pos_tags = [[token.text, token.pos_] for token in doc]
    return pos_tags

# Apply POS tagging to the DataFrame
outputs['pos_tags'] = outputs['cleaned_text'].apply(extract_pos_tags)
```

SENTIMENT ANALYSIS

Sentiment analysis is conducted within the preprocess_and_sentiment_by_index function using TextBlob, which assigns a polarity score to each review text.

Based on the polarity score, each review is categorized as Positive, Negative, or Neutral.



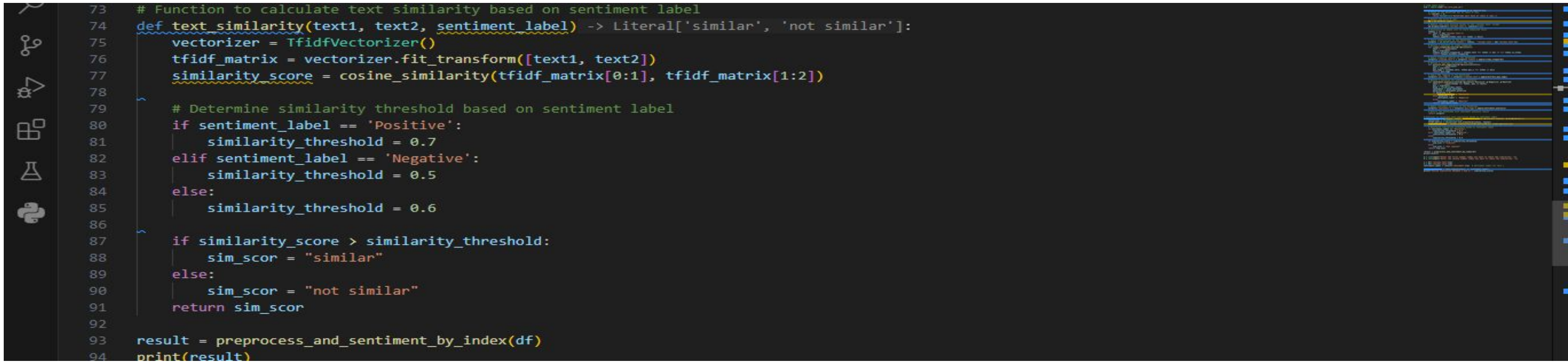
```
# Define a function for sentiment analysis
def sentiment_analysis(text) -> Literal['Positive', 'Negative', 'Neutra…:
    text = ' '.join([token for token, pos in text])
    doc = nlp(text)
    blob = TextBlob(doc.text)
    sentiment = blob.sentiment
    polarity = sentiment.polarity
    if polarity > 0:
        sentiment_label = 'Positive'
    elif polarity < 0:
        sentiment_label = 'Negative'
    else:
        sentiment_label = 'Neutral'
    return sentiment_label

# Apply sentiment analysis to the DataFrame
outputs['sentiment'] = outputs['pos_tags'].apply(sentiment_analysis)

# Return the DataFrame with sentiment analysis result
return outputs
```

- **TEXT SIMILARITY**

The text similarity between two review texts is determined using the cosine similarity metric. The similarity threshold is adjusted based on the sentiment label assigned to the first review text.

Cosine Similarity:

- Cosine similarity is computed using the TF-IDF representation of the review texts within the text_similarity function.
- If the cosine similarity score exceeds the predefined threshold, the texts are considered similar; otherwise, they are categorized as not similar.



```python
# Function to calculate text similarity based on sentiment label
def text_similarity(text1, text2, sentiment_label) -> Literal['similar', 'not similar']:
    vectorizer = TfidfVectorizer()
    tfidf_matrix = vectorizer.fit_transform([text1, text2])
    similarity_score = cosine_similarity(tfidf_matrix[0:1], tfidf_matrix[1:2])

    # Determine similarity threshold based on sentiment label
    if sentiment_label == 'Positive':
        similarity_threshold = 0.7
    elif sentiment_label == 'Negative':
        similarity_threshold = 0.5
    else:
        similarity_threshold = 0.6

    if similarity_score > similarity_threshold:
        sim_scor = "similar"
    else:
        sim_scor = "not similar"
    return sim_scor

result = preprocess_and_sentiment_by_index(df)
print(result)
```

EVALUATION

Text preprocessing includes tokenization, stop-word removal, and part-of-speech tagging. They serve to clean the text and extract features useful for further processing, mainly sentiment analysis and similarity measure calculation. Sentiment labels are most likely true for most of the reviews. For example, the first two are negative whereas the last four are positive.

The method for text similarity measurement in most test cases presented good agreement with manual similarity judgment.

MODEL'S STRENGTHS AND LIMITATIONS

- **STRENGTH**

  Pre-processing techniques in this model include: tokenization, stop-word removal, and part-of-speech tagging.

  It correctly classifies reviews into either the Positive, Negative or Neutral sentiment categories, though with an imperfect level of accuracy.

  The method builds on sophisticated natural language processing approaches, namely those that derive from term frequency-inverse document frequency vectorization and cosine similarity score calculations.

- **LIMITATION**

  The model's overall performance significantly depends on the quality and clarity of the input text. Clear reviews with no ambiguity will allow for more accurate sentiment analysis and calculation of the similarity of text. Applying some fixes—while using embeddings that are preserved by the model, inefficiently dealing with part of the spelling mistakes—will make the situation worse.

  Sentiment analysis is inherently subjective, and therefore it can vary with different individual perceptions. While the model may be trying to define sentiment in an objective way, different individuals may perceive the same sentiment in the text in different ways; therefore, there is even a chance of two classifications of sentiment not coinciding.

  The experimental performance of calculation of the measure depends on applied similarity thresholds.