Sentiment Analysis

Phase Four Project

Author: ***Winfred Karimi***

TM: ***Samuel Karu***

The project will CRISP-DM Criteria

Business understanding
Data Understanding
Data preparation
Modeling
Evaluation
Deployment

Business Understanding

Business Problem

Businesses face the challenge of analyzing large volumes of unstructured text data, such as customer reviews and social media posts, to understand sentiment. Manual analysis is time-consuming and inefficient, creating a need for an automated solution to classify text into positive, negative, or neutral sentiments. This will help businesses make data-driven decisions and improve customer satisfaction.

Business Overview

Sentiment analysis is vital across industries like retail, hospitality, and finance. It helps monitor brand reputation, identify customer pain points, and tailor marketing strategies. For example, analyzing product reviews or social media feedback enables companies to enhance customer experiences and address issues promptly, driving growth and improving brand loyalty.

Objective of the Project

The project aims to build a sentiment analysis model to classify text into positive, negative, or neutral sentiments. It involves preprocessing text data, extracting features, training machine learning or deep learning models, and evaluating performance. The final goal is to create a tool that automates sentiment analysis, helping businesses analyze text data efficiently and make informed decisions.

Data Understanding

Data repository

The dataset, known as "Tweet Sentiment Analysis", was downloaded from Kaggle . It contains text data from tweets, where each tweet is labeled with a sentiment: positive, negative, or neutral. The dataset can be Download Here

Data overview

The dataset provided contains text samples with four columns: textID, text, selected_text, and sentiment. Each row represents a unique text entry, where:

1. textID is a unique identifier for each text.
2. text contains the full sentence or phrase.
3. selected_text highlights the specific part of the text that reflects the sentiment.
4. sentiment labels the text as positive, negative, or neutral.

Data Preparation

```
# import libraries
import pandas as pd
import numpy as np

import re
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords

from sklearn.model_selection import train_test_split

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D,
GlobalMaxPooling1D, Dense


# Download stopwords
nltk.download('wordnet')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\USER\AppData\Roaming\nltk_data...
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\USER\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!

True
```

```python
# load the dataset
df = pd.read_csv("Data\Tweets.csv")
df.head()
```

```
       textID                                               text  \
0  cb774db0d1                I`d have responded, if I were going
1  549e992a42      Sooo SAD I will miss you here in San Diego!!!
2  088c60f138                          my boss is bullying me...
3  9642c003ef                     what interview! leave me alone
4  358bd9e861   Sons of ****, why couldn`t they put them on t...

                        selected_text sentiment
0  I`d have responded, if I were going   neutral
1                            Sooo SAD  negative
2                         bullying me  negative
3                      leave me alone  negative
4                      Sons of ****,  negative
```

```python
# preview the dataset
display(df.head(10))
display(df.tail(10))
```

```
       textID                                               text  \
0  cb774db0d1                I`d have responded, if I were going
1  549e992a42      Sooo SAD I will miss you here in San Diego!!!
2  088c60f138                          my boss is bullying me...
3  9642c003ef                     what interview! leave me alone
4  358bd9e861   Sons of ****, why couldn`t they put them on t...
5  28b57f3990  http://www.dothebouncy.com/smf - some shameles...
6  6e0c6d75b1  2am feedings for the baby are fun when he is a...
7  50e14c0bb8                                         Soooo high
8  e050245fbd                                        Both of you
9  fc2cbefa9d   Journey!? Wow... u just became cooler.  hehe....

                                       selected_text sentiment
0                I`d have responded, if I were going   neutral
1                                           Sooo SAD  negative
2                                        bullying me  negative
3                                     leave me alone  negative
4                                      Sons of ****,  negative
5  http://www.dothebouncy.com/smf - some shameles...   neutral
6                                                fun  positive
7                                         Soooo high   neutral
8                                        Both of you   neutral
9                        Wow... u just became cooler.  positive

           textID
text  \
27471  15bb120f57  i`m defying gravity. and nobody in alll of oz,...
```

```
27472  8f5adc47ec  http://twitpic.com/663vr - Wanted to visit the...
27473  a208770a32   in spoke to you yesterday and u didnt respond...
27474  8f14bb2715  So I get up early and I feel good about the da...
27475  b78ec00df5                                     enjoy ur night
27476  4eac33d1c0   wish we could come see u on Denver  husband l...
27477  4f4c4fc327   I`ve wondered about rake to.  The client has ...
27478  f67aae2310   Yay good for both of you. Enjoy the break - y...
27479  ed167662a5                        But it was worth it  ****.
27480  6f7127d9d7     All this flirting going on - The ATG smiles...


                                           selected_text sentiment
27471  i`m defying gravity. and nobody in alll of oz,...   neutral
27472                                      were too late  negative
27473  in spoke to you yesterday and u didnt respond ...   neutral
27474                                     I feel good ab  positive
27475                                              enjoy  positive
27476                                             d lost  negative
27477                                      , don`t force  negative
27478                          Yay good for both of you.  positive
27479                         But it was worth it  ****.  positive
27480  All this flirting going on - The ATG smiles. Y...   neutral
```

```python
# check info of the data
print(f"The shape indicates that the dataset has {df.shape[0]} rows and {df.shape[1]} Columns")
```

```
The shape indicates that the dataset has 27481 rows and 4 Columns
```

```python
#Check more info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27481 entries, 0 to 27480
Data columns (total 4 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   textID         27481 non-null  object
 1   text           27480 non-null  object
 2   selected_text  27480 non-null  object
 3   sentiment      27481 non-null  object
dtypes: object(4)
memory usage: 858.9+ KB
```

Checking for missing values

```
# checking missing values
df.isnull().sum()
```

```
textID           0
text             1
selected_text    1
sentiment        0
dtype: int64
```

```
# droping the missing values
df.dropna(inplace=True)
print(f"Now the dataset has {df.isnull().sum().sum()} missing texts or values")
```

```
Now the dataset has 0 missing texts or values
```

```
# Checkinf for duplicated texts
print(f"This dataset contains {df.duplicated().sum()} duplicated rows")
```

```
This dataset contains 0 duplicated rows
```

Working on columns

```
df.columns
```

```
Index(['textID', 'text', 'selected_text', 'sentiment'],
dtype='object')
```

The dataset contains 4 columns, lets review the importance of each column as illustrated in the table below

Column name	Description	Status
textID	A unique identifier for each tweet.	Drop
text	The full original text of the tweet.	Keep
selected_text	Text extract that shows the sentiment.	Drop
sentiment	The sentiment label	Keep

```
# drop columns
df.drop(columns=['textID', 'selected_text'], inplace=True)
#print columns
df.columns
```

```
Index(['text', 'sentiment'], dtype='object')
```

Text processing

This process involves standardizing text data that is by:

1. Converting to lowercase
2. Removing special characters, numbers, and punctuation
3. Removing stopwords
4. Lemmatization that is reducing words to their root form

```
# Initialize lemmatizer and stopwords
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

""""
Since stopword removal can eliminate important words like "not,"
sentiment can be misinterpreted.
To counter this, we define a set of common negation words to preserve
and detect sentiment reversals.
This includes standard negations and contractions like "not," "no,"
"never," "n't," "can't," etc.
"""
negation_words = {"not", "no", "never", "n't", "can't", "won't",
"shouldn't", "isn't", "wasn't", "couldn't"}
# define a function
def preprocess_text(text):
    # standardize text to lowercased
    text = text.lower()
    # Remove HTML tags
    text = re.sub(r'<.*?>', '', text)
    # Remove special characters
    text = re.sub(r'[^a-z\s]', '', text)

    words = text.split()

    # Negation handling
    processed_words = []
    negate = False

    for word in words:
        if word in negation_words:
            negate = True
            processed_words.append(word)
        elif negate:
            processed_words.append(f"not_{word}")
            negate = False
        else:
            processed_words.append(word)

    # Remove stopwords but keep negation words
    processed_words = [word for word in processed_words if word not in
stop_words or word in negation_words]
    # Lemmatization
    processed_words = [lemmatizer.lemmatize(word) for word in
processed_words]
```

```
    return " ".join(processed_words)
# Apply preprocessing
df['cleaned_text'] = df['text'].apply(preprocess_text)

# preview the text processed data
print(f"This is the orginal text before text processing:
{df[['text']].head()}")
print("_"*100)
# After text processing
print(f"This is the new text after text processing:
{df[['cleaned_text']].head()}")
```

```
This is the orginal text before text processing:
text
0                  I`d have responded, if I were going
1       Sooo SAD I will miss you here in San Diego!!!
2                                my boss is bullying me...
3                           what interview! leave me alone
4    Sons of ****, why couldn`t they put them on t...
____________________________________________________________________________________________________
This is the new text after text processing:
cleaned_text
0                       id responded going
1                  sooo sad miss san diego
2                             bos bullying
3                    interview leave alone
4   son couldnt put release already bought
```

Modelling

Tokenization and padding

The process of tokenization and padding involves converting text into numerical format for machine learning models. Tokenization breaks text into words or subwords and maps them to unique numerical indices. Padding ensures that all sequences have the same length by adding zeros or placeholders to shorter sequences. This standardization allows models to process text efficiently.

```
# Tokenize the text
tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(df['cleaned_text'])
X = tokenizer.texts_to_sequences(df['cleaned_text'])
# Pad sequences to a fixed length
# a twitter comment usully is of about a mean of 25 words
# doubled it
max_len = 50
X = pad_sequences(X, maxlen=max_len)
```

```python
# Map sentiment labels to numerical values
y = df['sentiment'].map({'negative': 0, 'neutral': 1, 'positive': 2})

# intiate the sequential model
model = Sequential()
# Embedding layer to remove input_length
model.add(Embedding(input_dim=5001, output_dim=128))
# 1D Convolutional layer
model.add(Conv1D(filters=128, kernel_size=3, activation='relu'))
# Global Max Pooling
model.add(GlobalMaxPooling1D())
# Fully connected layers
model.add(Dense(10, activation='relu'))
model.add(Dense(3, activation='softmax'))  # 3 classes: negative,
neutral, positive
# Compile the model
model.compile(loss='sparse_categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])

# checking the class distribution before splittng the data
print("Original class distribution:\n", y.value_counts())
```

```
Original class distribution:
 1    11117
2     8582
0     7781
Name: sentiment, dtype: int64
```

```python
# spliting the data
# using test size of 20% and random state of 42 and parameter
strarify= y
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)

#checking the class distribution after spliting the data
print("Training set class distribution:\n", y_train.value_counts())
print("Test set class distribution:\n", y_test.value_counts())
print("The dataset contains class imbalances, but the diffrence
between them is not huge, hence no need of balancing them using
SMOTE")
```

```
Training set class distribution:
 1    8893
2    6866
0    6225
Name: sentiment, dtype: int64
Test set class distribution:
 1    2224
2    1716
0    1556
Name: sentiment, dtype: int64
```

```
The dataset contains class imbalances, but the diffrence between them
is not huge, hence no need of balancing them using SMOTE
```

```python
# Train the sentiment analysis model for 5 epochs with a batch size of
64, using training data.
# Validate performance on the test set after each epoch.
sent_model = model.fit(X_train, y_train, epochs=5, batch_size=64,
validation_data=(X_test, y_test))
```

```
Epoch 1/5
344/344 [==============================] - 13s 37ms/step - loss:
0.8126 - accuracy: 0.6286 - val_loss: 0.6779 - val_accuracy: 0.7198
Epoch 2/5
344/344 [==============================] - 9s 27ms/step - loss: 0.5745
- accuracy: 0.7698 - val_loss: 0.6840 - val_accuracy: 0.7185
Epoch 3/5
344/344 [==============================] - 9s 25ms/step - loss: 0.4384
- accuracy: 0.8354 - val_loss: 0.7491 - val_accuracy: 0.7051
Epoch 4/5
344/344 [==============================] - 8s 23ms/step - loss: 0.3103
- accuracy: 0.8932 - val_loss: 0.8614 - val_accuracy: 0.6890
Epoch 5/5
344/344 [==============================] - 8s 24ms/step - loss: 0.2034
- accuracy: 0.9358 - val_loss: 1.0075 - val_accuracy: 0.6778
```

```python
# Retrieve the training history, including accuracy and loss for both
training and validation sets.
sent_model.history
```

```
{'loss': [0.8125553727149963,
  0.5744954347610474,
  0.4383648931980133,
  0.3103049695491791,
  0.20340023934841156],
 'accuracy': [0.6286389827728271,
  0.7698326110839844,
  0.8353802561759949,
  0.8932405114173889,
  0.9357714653015137],
 'val_loss': [0.6779490113258362,
  0.6839694380760193,
  0.7490654587745667,
  0.8614413142204285,
  1.0075197219848633],
 'val_accuracy': [0.7197962403297424,
  0.7185225486755371,
  0.705058217048645,
  0.6890465617179871,
  0.6777656674385071]}
```

```
# Evaluate the trained model
test_loss, test_accuracy = model.evaluate(X_test, y_test)
# measure its performance.
print(f"Test Accuracy: {test_accuracy:.4f}")
print(f"Test Loss: {test_loss:.4f}")
```

```
172/172 [==============================] - 1s 4ms/step - loss: 1.0075
- accuracy: 0.6778
Test Accuracy: 0.6778
Test Loss: 1.0075
```

Create function to predict sentiments

```
# Mapping labels
sentiment_labels = {0: "Negative", 1: "Neutral", 2: "Positive"}
#define a function
def predict_sentiment():
    user_text = input("Enter a tweet: ")
    # Preprocess the input text
    processed_text = preprocess_text(user_text)
    # Tokenize and pad the sequence
    sequence = tokenizer.texts_to_sequences([processed_text])
    padded_sequence = pad_sequences(sequence, maxlen=max_len)
    # Predict sentiment
    prediction = model.predict(padded_sequence)
    predicted_class = prediction.argmax(axis=1)[0]  # Get class with
highest probability

    print(f"\nTweet: {user_text}")
    print(f"Predicted Sentiment: {sentiment_labels[predicted_class]}\
n")
```

Some examples of predicted sentiments

This is how it works the user inputs a tweep comment and the system predicts if it is postive, neutral or negative

```
predict_sentiment()


Tweet: i really hate that hotel, its a bad one
Predicted Sentiment: Negative


predict_sentiment()


Tweet: she has have very good vibe she is good at her job
Predicted Sentiment: Positive
```

```
predict_sentiment()

Tweet: The  movie was so great i like it
Predicted Sentiment: Positive

predict_sentiment()

Tweet: allow me not to comment on this
Predicted Sentiment: Neutral

predict_sentiment()

Tweet: im not sure if i will make to come
Predicted Sentiment: Neutral
```