Emotion Detection of Text

Tayyub Naveed (P19-0108)

Usman Manzoor (P19-0068)

April 16, 2023

Abstract

This NLP project aims to detect emotions in textual data using various techniques. The project involves data cleaning to remove stop words and special characters, followed by sentiment analysis to add the corresponding emotions to the dataset. The next step involves extracting words to determine their relevance and importance to the sentiment of the text. Finally, a Naive Bayes model is used to predict the sentiment of the text, and the performance of the model is evaluated using a confusion matrix. Overall, this project demonstrates the application of NLP techniques in emotion detection, which can be useful in various fields such as customer service, social media analysis, and market research.

1 Code Explanation

```
df = pd.read_csv("emotion_dataset.csv")
```

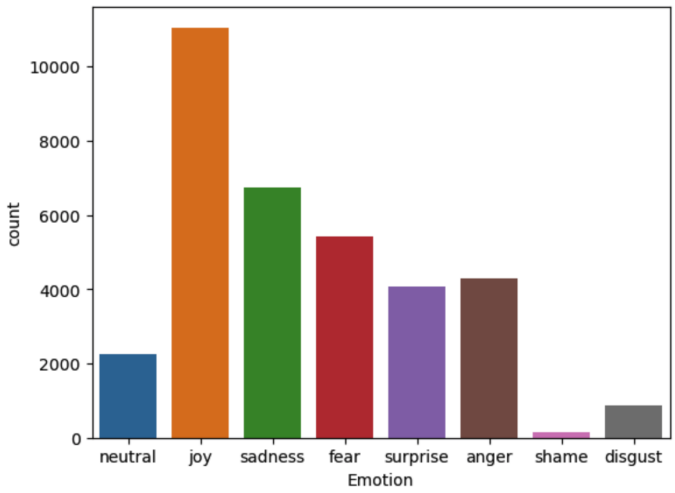
```
df.head()
```

	Emotion	Text
0	neutral	Why ?
1	joy	Sage Act upgrade on my to do list for tommorow.
2	sadness	ON THE WAY TO MY HOMEGIRL BABY FUNERAL!!! MAN ...
3	joy	Such an eye ! The true hazel eye-and so brill...
4	joy	@lluvmiasantos ugh babe.. hugggzzz for u .! b...

using the pandas library, reading a CSV file called "emotion-dataset.csv" and saving it into a pandas DataFrame object called df and displaying the first few rows of the DataFrame using the head() method.

```
sns.countplot(x='Emotion',data=df)
```

```
<Axes: xlabel='Emotion', ylabel='count'>
```



using the seaborn library, display the Emotion column in the df DataFrame. The x-axis will show the different categories of emotions present in the dataset, and the y-axis will show the count of each category.

```
from textblob import TextBlob
```

```
def get_sentiment(text):
    blob = TextBlob(text)
    sentiment = blob.sentiment.polarity
    if sentiment > 0:
        result = "Positive"
    elif sentiment < 0:
        result = "Negative"
    else:
        result  = "Neutral"
    return result
```

```
get_sentiment("I love coding")
```

```
'Positive'
```

The TextBlob class is used to create a TextBlob object, which represents a text as a sequence of words. The sentiment property of a TextBlob object returns a tuple of two values:

- polarity
- subjectivity

The polarity is a float value between -1 and 1 that indicates the sentiment of the text, with negative values indicating negative sentiment, positive values indicating positive sentiment, and 0 indicating neutral sentiment.

The function get-sentiment(text) takes a string text as input and returns a string indicating the sentiment of the text. If the sentiment is greater than 0, the function returns "Positive". If the sentiment is less than 0, the function returns "Negative". Otherwise, the function returns "Neutral".

```
df['Sentiment'] = df['Text'].apply(get_sentiment)
```

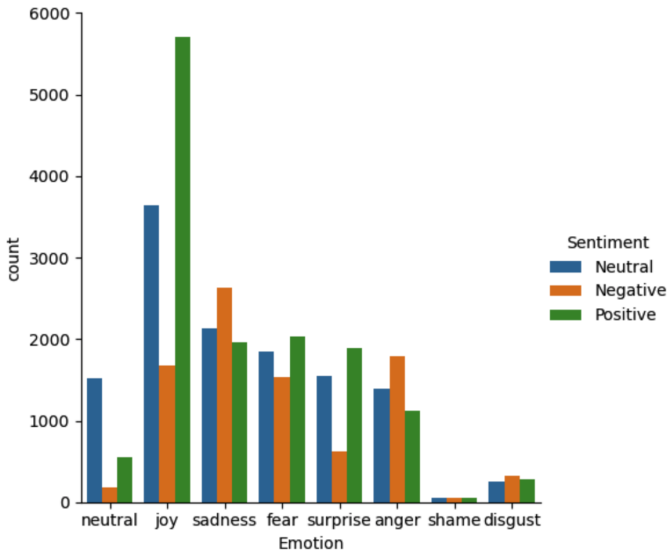
```
df.head()
```

	Emotion	Text	Sentiment
0	neutral	Why ?	Neutral
1	joy	Sage Act upgrade on my to do list for tommorow.	Neutral
2	sadness	ON THE WAY TO MY HOMEGIRL BABY FUNERAL!!! MAN ...	Negative
3	joy	Such an eye ! The true hazel eye-and so brill...	Positive
4	joy	@lluvmiasantos ugh babe.. hugggzzz for u .! b...	Neutral

The code adds a new column called 'Sentiment' to the df DataFrame. The values in this column are computed by applying the get-sentiment() function to the 'Text' column of the DataFrame.

```
sns.catplot(x='Emotion', hue='Sentiment', data=df,kind='count')
```

```
<seaborn.axisgrid.FacetGrid at 0x7fcde2b64100>
```



using the seaborn library, creates a categorical plot of the count of each sentiment in each emotion category.

```
import neattext.functions as nfx
```

```
df['Text']
```

```
0                                                  Why ?
1            Sage Act upgrade on my to do list for tommorow.
2          ON THE WAY TO MY HOMEGIRL BABY FUNERAL!!! MAN ...
3           Such an eye ! The true hazel eye-and so brill...
4          @Iluvmiasantos ugh babe.. hugggzzz for u .!  b...
                               ...
34787      @MichelGW have you gift! Hope you like it! It'...
34788      The world didnt give it to me..so the world MO...
34789                                A man robbed me today .
34790      Youu call it JEALOUSY, I call it of #Losing YO...
34791      I think about you baby, and I dream about you ...
Name: Text, Length: 34792, dtype: object
```

```
df['Clean_Text'] = df['Text'].apply(nfx.remove_stopwords)
```

```
df['Clean_Text'] = df['Text'].apply(nfx.remove_punctuations)
```

```
df['Clean_Text'] = df['Text'].apply(nfx.remove_userhandles)
```

using "neattext" library to perform text preprocessing on a DataFrame column called 'Text'. We then apply the different text preprocessing functions from "neattext" to the 'Text' column of the DataFrame using the apply() method. Each function is applied one after the other using the apply() method again.
The output of each preprocessing step is then used as input to the next preprocessing step. The output of the last step is assigned to a new column called 'Clean-Text'.

```
from collections import Counter
```

```
def extract_keywords(text, num=50):
    tokens = [tok for tok in text.split()]
    most_common_tokens = Counter(tokens).most_common(num)
    return dict(most_common_tokens)
```

```
emotion_list = df['Emotion'].unique().tolist()
```

```
emotion_list
```

```
['neutral', 'joy', 'sadness', 'fear', 'surprise', 'anger', 'shame', 'disgust']
```

```
joy_list = df[df['Emotion']=='joy']['Clean_Text'].tolist()
```

```
joy_docx = ''.join(joy_list)
```

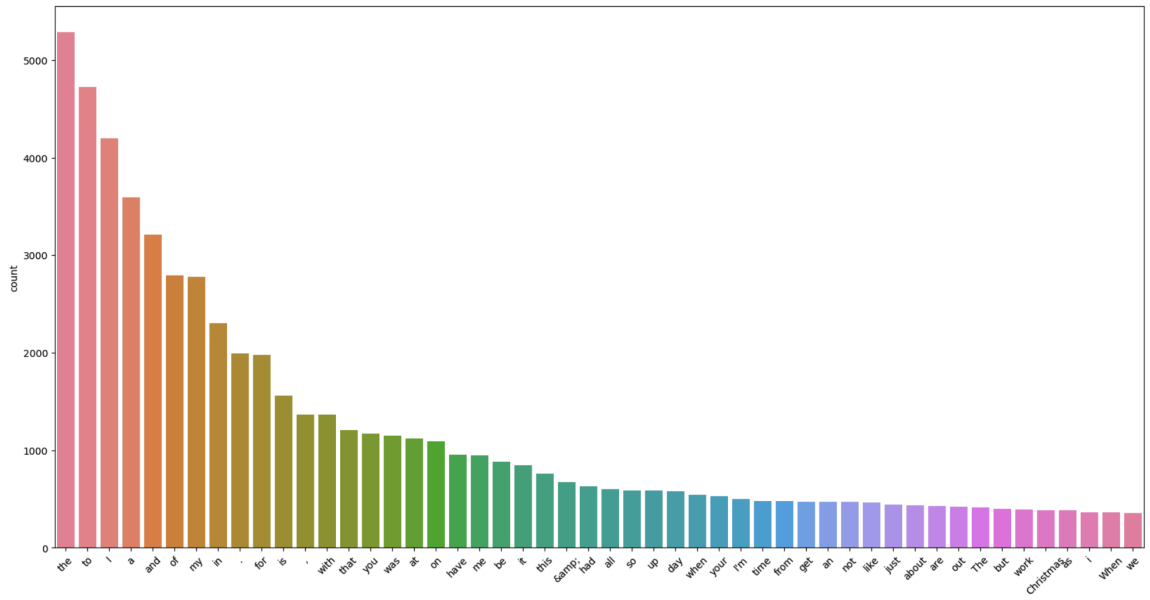
```
keyword_joy = extract_keywords(joy_docx)
```

In this code, we first import the "Counter" class from the "collections" module and the "neattext" library. We then define a function called "extract-keywords" that takes a text input and a number num (default value 50), and returns a dictionary of the num most common words in the input text.

- We then define a variable "emotion-list" that contains the unique emotion categories in the 'Emotion' column of the DataFrame.
- Next, we define a variable "joy-list" that contains all the clean text for rows where the 'Emotion' column is 'joy'.
- We then concatenate the clean text in "joy-list" using the join() method, and assign the result to a variable called "joy-docx".
- Finally, we apply the "extract-keywords" function to "joy-docx" and assign the result to a variable called "keyword-joy". This will give us a dictionary of the 50 most common words in the text associated with the 'joy' emotion category.

```
def plot_most_common_words(mydict):
    df_01=pd.DataFrame(mydict.items(), columns=['token','count'])
    plt.figure(figsize=(20,10))
    sns.barplot(x='token',y='count',data=df_01)
    plt.xticks(rotation=45)
    plt.show()
```

```
plot_most_common_words(keyword_joy)
```



A function "plot-most-common-words" takes a dictionary "mydict" and does the following:

- Converts the dictionary to a DataFrame using the items() method.
- Creates a bar plot using the sns.barplot() function, with the 'token' column on the x-axis and the 'count' column on the y-axis.
- Rotates the x-axis labels by 45 degrees using the plt.xticks() function.

```
cv = CountVectorizer()
X = cv.fit_transform(Xfeatures)
```

```
x_train,x_test,y_train,y_test = train_test_split(X,ylabels,test_size=0.3,random_state=42)
```

```
nv_model = MultinomialNB()
nv_model.fit(x_train, y_train)
```

```
MultinomialNB()
```

We create an instance of the "CountVectorizer" class and apply it to the 'Clean-Text' column of the DataFrame df using the fit-transform() method. The resulting vectorized data is assigned to the variable X. We also assign the target labels (i.e., the 'Emotion' column of the DataFrame df) to the variable y. Next, we split the vectorized data and the labels into training and testing sets using the train-test-split() function. We assign the training and testing sets for the vectorized data and the labels to the variables x-train, x-test, y-train, and y-test, respectively. Finally, we create an instance of the MultinomialNB class and fit it to the training data and labels using the fit() method. The resulting trained model is assigned to the variable nv-model.

```
sample_text = ["I love coding so much"]
```

```
vect = cv.transform(sample_text).toarray()
```

```
nv_model.predict(vect)
```

```
array(['joy'], dtype='<U8')
```

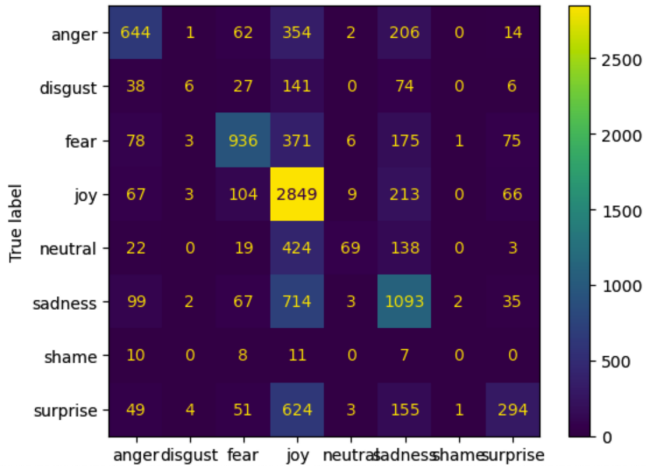
we first define the text sample "I love coding so much". We then transform the text sample into a vectorized form using the transform() method of the "CountVectorizer" object cv. The resulting vector is converted to an array using the toarray() method, and the resulting array is assigned to the variable vect.

Finally, we use the predict() method of the trained Naive Bayes model (nv-model) to predict the emotion label of the text sample, based on its vectorized form (vect). In this code, using plot-

```
plot_confusion_matrix(nv_model, x_test, y_test)
```

```
/opt/anaconda3/lib/python3.9/site-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function plot_confusion_ma
trix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the
class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fcde11a5490>
```



confusion-matrix() function (The confusion matrix is a useful tool for evaluating the performance of a classification model.) from the sklearn.metrics module. We then call this function with the trained Naive Bayes model (nv-model), the test set (x-test and y-test), which we obtained earlier using the train-test-split() function.