NLP: Text classification ¶

In [1]:

```
import pandas as pd
```

In [2]:

```
df=pd.read_csv('C:/Users/ahsan/Downloads/nlp_dataset.csv')
df
```

Out[2]:

	Comment	Emotion
0	i seriously hate one subject to death but now ...	fear
1	im so full of life i feel appalled	anger
2	i sit here to write i start to dig out my feel...	fear
3	ive been really angry with r and i feel like a...	joy
4	i feel suspicious if there is no one outside l...	fear
...	...	...
5932	i begun to feel distressed for you	fear
5933	i left feeling annoyed and angry thinking that...	anger
5934	i were to ever get married i d have everything...	joy
5935	i feel reluctant in applying there because i w...	fear
5936	i just wanted to apologize to you because i fe...	anger

5937 rows × 2 columns

In [3]:

```
df.isnull().sum()
```

Out[3]:

```
Comment    0
Emotion    0
dtype: int64
```

In [4]:

```
df.duplicated().sum()
```

Out[4]:

```
0
```

In [5]:

```
#preprocessing step
#1.tokenization
#2.lowercasing
#3.stopwordsremoval
#4.stemming
#5.lemmatization
```

Step-1 (preprocessing)

In [6]:

```python
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
nltk.download('stopwords')
from nltk import WordNetLemmatizer
from nltk import PorterStemmer
nltk.download('wordnet')
import re
```

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

In [7]:

```python
stop_words=set(stopwords.words('english'))
lemmer=WordNetLemmatizer()
```

In order to preprocessing the text we need to go through these following steps:

Lowercasing: Converting all characters to lowercase to ensure uniformity(e.g, "Text" and "text").

Punctuation Removal: Eliminates special characters and punctuation that do not contribute to semantic meaning, simplifying the dataset.

Tokenization: Splits text into individual words (tokens), making it easier to process for NLP models.

Stopword Removal:Eliminating the common words.(e.g,"is","the") usually do not contribute much meaning to the text

Lemmatization: Reduces words to their base form (e.g, "running"to"run"), helping to group similar terms and reduce dimensionality.

In [8]:

```python
def preprocessing_text(text):
    # Convert text to lowercase
    text=text.lower()
    # Remove punctuation and special characters
    text=re.sub(r'[^\w\s]', '', text)
    # Tokenize text
    tokens=word_tokenize(text)
    # Remove stopwords
    tokens=[i for i in tokens if i not in stop_words]
    # Lemmatize tokens
    tokens=[lemmer.lemmatize(i)for i in tokens]
    # Join tokens back to form a cleaned string
    return ' '.join(tokens)
```

In [9]:

```
# Apply the preprocessing function to the 'comment' column
df['Comment']=df['Comment'].apply(preprocessing_text)
```

In [10]:

```
df['Comment']
```

Out[10]:

```
0       seriously hate one subject death feel reluctan...
1                                im full life feel appalled
2       sit write start dig feeling think afraid accep...
3       ive really angry r feel like idiot trusting fi...
4       feel suspicious one outside like rapture happe...
                              ...                        
5932                                 begun feel distressed
5933    left feeling annoyed angry thinking center stu...
5934    ever get married everything ready offer got to...
5935    feel reluctant applying want able find company...
5936           wanted apologize feel like heartless bitch
Name: Comment, Length: 5937, dtype: object
```

These preprocessing steps are foundational for effective NLP tasks, including sentiment analysis, where a cleaner dataset leads to more reliable emotion classification.

Step-2 (Feature extraction)

In [11]:

```
#Convert the preprocessed text data in the Comment column into a numer
from sklearn.feature_extraction.text import CountVectorizer
cv=CountVectorizer()
BowedComment=cv.fit_transform(df['Comment'])
```

In [12]:

```
BowedComment
```

Out[12]:

```
<5937x7970 sparse matrix of type '<class 'numpy.int64'>'
        with 53718 stored elements in Compressed Sparse Row format>
```

The CountVectorizer method transforms text data into a numerical representation using the bag-of-words (BoW) model. It tokenizes the text, creates a vocabulary of unique words from the dataset, and generates a sparse matrix where each row corresponds to a text entry, and each column represents a word in the vocabulary. The values in the matrix are the word counts for each text entry. This process converts unstructured text into structured numerical features, enabling machine learning models to process and analyze the data effectively.

Step 3 (Model Building)

After transforming the text into a numerical format (BowedComment), you can proceed with model building.

In [13]:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

In [14]:

```
x=BowedComment
y=df['Emotion']
```

In [15]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,rando
```

In [16]:

```
x_train
```

Out[16]:

```
<4749x7970 sparse matrix of type '<class 'numpy.int64'>'
	with 42909 stored elements in Compressed Sparse Row format>
```

In [17]:

```
y_train
```

Out[17]:

```
4945      joy
5428     fear
1344    anger
1888    anger
2480     fear
        ...  
3772    anger
5191     fear
5226    anger
5390     fear
860      fear
Name: Emotion, Length: 4749, dtype: object
```

In [18]:

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC


# Initialize models
nb_model = MultinomialNB()      # Naive Bayes
svm_model = SVC(kernel='linear') # Support Vector Machine with Linear

# Train both models
nb_model.fit(x_train, y_train)
svm_model.fit(x_train, y_train)

# Predict using both models
y_pred_nb = nb_model.predict(x_test)    # Predictions from Naive Bayes
y_pred_svm = svm_model.predict(x_test)  # Predictions from SVM

# Evaluate both models
from sklearn.metrics import accuracy_score
accuracy_nb = accuracy_score(y_test, y_pred_nb)
accuracy_svm = accuracy_score(y_test, y_pred_svm)
print(f'Naive Bayes Accuracy: {accuracy_nb}')
print(f'SVM Accuracy: {accuracy_svm}')
```

```
Naive Bayes Accuracy: 0.8973063973063973
SVM Accuracy: 0.9503367003367004
```

Naive Bayes achieved an accuracy of 89.7%, demonstrating its suitability for text classification due to its simplicity and efficiency with sparse data. In contrast, SVM achieved a higher accuracy of 95.0%, showcasing its ability to handle high-dimensional text data and separate classes effectively with maximum margin. The superior performance of SVM indicates it is more reliable for capturing nuanced patterns in emotion classification compared to Naive Bayes.