Name: Mohamed Ashraf Rabie

ID: 21100854

Name: Mohamed Ahmed Salah

ID: 21100806

Project: Sentiment classification

The models that were chosen are:

- Support Vector Machines (SVM)

  - SVM can be effective for sentiment classification when you have a binary classification task, such as distinguishing between positive and negative sentiments.
  - In this case, you can train two SVM classifiers: one to distinguish positive sentiments from neutral/negative, and another to distinguish negative sentiments from neutral/positive.
  - SVM's ability to find an optimal hyperplane for separating classes can be useful when the sentiment categories are well-defined and distinct.

- Naive Bayes (NB)

  - NB can be useful for sentiment classification even with multiple output categories.
  - You can train a multiclass NB classifier that estimates the probability of each sentiment category based on the input features.
  - NB's assumption of independence between features may not hold perfectly in sentiment analysis, but it can still provide reasonable results and be computationally efficient.

- Decision Trees (DT)

  - DT can also be used for sentiment classification with multiple output categories.
  - You can train a decision tree classifier that can split the data based on different sentiment features to classify the sentiments into positive, neutral, or negative categories.
  - DT's ability to capture non-linear relationships and handle both categorical and numerical data can be advantageous for sentiment analysis tasks.

Importing the needed Libraries:

```
import tkinter as tk
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
```

tkinter is for creating a GUI for the classifier.

Pandas is for creating the data frame and managing the data set.

From **sklearn** we import:

-CountVectorizer

That will be used to allow us to handle data in the form of strings.

-MultinomialNB

-SVC

-DecisionTreeClassifier

That will be used to create models that will predict the sentiment.

- train_test_split

That will be used to split the data into training and testing.

-f1_score

That will be used to calculate the f1 scores of the models.

- confusion_matrix

That will be used to create the confusion matrix for the used models.

Reading the dataset:

```
# Load the Sentiment140 dataset
df = pd.read_csv('largeData.csv', encoding='ISO-8859-1', header=None,
                 names=['target', 'id', 'date', 'flag', 'user', 'text'])
df = df.sample(n=20000, random_state=42)
```

Loads the Sentiment140 dataset from a CSV file called 'largeData.csv'. The dataset contains columns with the following names: 'target', 'id', 'date', 'flag', 'user', 'text'.

Uses the pd.read_csv() function from the pandas library to read the CSV file.

The encoding='ISO-8859-1' parameter is used to specify the character encoding of the CSV file.

The header=None parameter indicates that the CSV file does not contain a header row, and names parameter provides a list of column names explicitly.

The df.sample() function randomly samples rows from the dataframe df. In this case, it samples 20,000 rows from the dataframe using the n=20000 parameter and sets the random seed to 42 using the random_state=42 parameter. (to insure that the same data is chosen every time)

Overall, this code loads a dataset from a CSV file, specifies column names, and randomly samples 20,000 rows from the dataset for further processing or analysis.

Preprocess the data:

```
# Preprocess the data
df['text'] = df['text'].str.lower()
df['text'] = df['text'].str.replace('[^\w\s]', '', regex=True)
df['text'] = df['text'].str.replace('\d+', '', regex=True)
df['text'] = df['text'].str.strip()
```

df['text'] = df['text'].str.lower(): Converts all the text in the 'text' column to lowercase. This is done to ensure that capitalization variations do not affect the analysis and to normalize the text.

df['text'] = df['text'].str.replace('[^\w\s]', '', regex=True): Removes all non-alphanumeric characters and punctuation from the text in the 'text' column. The regular expression [^\w\s] matches any character that is not a word character (alphanumeric) or whitespace. By replacing these characters with an empty string, it effectively removes them from the text.

df['text'] = df['text'].str.replace('\d+', '', regex=True): Removes all numeric digits from the text in the 'text' column. The regular expression \d+ matches one or more consecutive digits. By replacing them with an empty string, it removes the numeric digits from the text.

df['text'] = df['text'].str.strip(): Removes leading and trailing whitespaces from the text in the 'text' column. This step ensures that any extra spaces at the beginning or end of the text are eliminated.

Overall, these preprocessing steps are aimed at cleaning and standardizing the text data in the 'text' column. By converting to lowercase, removing non-alphanumeric characters, punctuation, numeric digits, and leading/trailing whitespaces, the text is prepared for further analysis or modeling tasks

Vectorizing the text:

```
vectorizer = CountVectorizer(stop_words='english')
x = vectorizer.fit_transform(df['text'])
y = df['target']
```

vectorizer = CountVectorizer(stop_words='english'): Initializes a CountVectorizer object, which is used to convert the text data into a matrix of token counts. The stop_words='english' parameter specifies that common English words (e.g., "the", "is", "and") should be excluded from the tokenization process, as they often carry little or no additional meaning.

x = vectorizer.fit_transform(df['text']): Applies the fit_transform method of the CountVectorizer object to the 'text' column of the DataFrame (df['text']). This step performs two actions:

fit_transform: Learns the vocabulary of the corpus (text data) and returns a document-term matrix, where each row represents a document (text sample) and each column represents a word (token) in the vocabulary. The matrix contains the counts of how many times each word appears in each document.

Assigns the resulting matrix to the variable x.

y = df['target']: Assigns the 'target' column of the DataFrame (df['target']) to the variable y. This column typically represents the target variable or labels associated with the text data, such as sentiment labels or class labels.

After these operations, x represents the feature matrix, where each row corresponds to a text sample, and each column represents a word from the vocabulary with its count in the respective document. y represents the corresponding target variable associated with each text sample.

This vectorization process prepares the text data in a format suitable for various machine learning algorithms that require numerical inputs. The resulting x and y can be used for tasks like sentiment analysis, or any other analysis that involves numerical representations of text data.

Splitting the data:

```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=42)
```

train_test_split(x, y, test_size=0.2, random_state=42): This function splits the feature matrix (x) and the target variable (y) into separate training and testing sets. The parameters are:

x: The feature matrix, which typically represents the input data (text samples) for a machine learning model.

y: The target variable, which represents the corresponding labels or categories associated with the input data.

test_size=0.2: Specifies the proportion of the data that should be allocated for testing. In this case, 20% of the data will be used for testing, while the remaining 80% will be used for training.

random_state=42: Sets the random seed to 42, ensuring that the same split is obtained each time the code is run. This allows for reproducibility of the results.

X_train, X_test, y_train, y_test: Assigns the output of train_test_split to separate variables. The split data is assigned as follows:

X_train: The training set of the feature matrix (x), which will be used for model training.

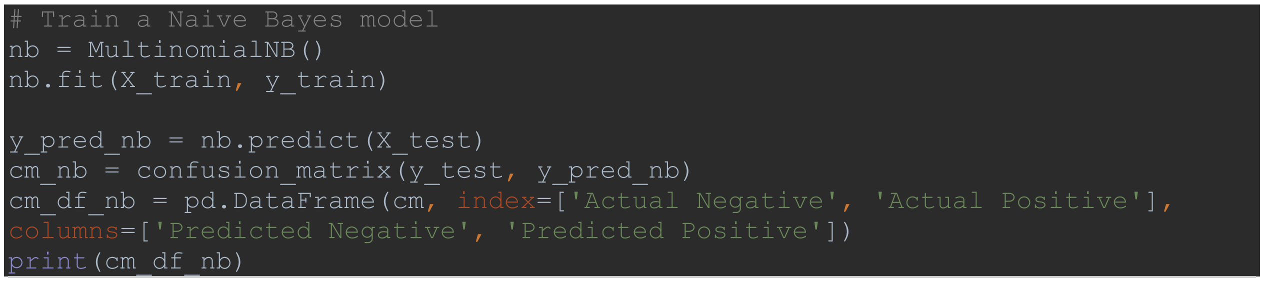
X_test: The testing set of the feature matrix (x), which will be used for evaluating the trained model's performance.

y_train: The training set of the target variable (y), corresponding to the labels/categories for the training data.

y_test: The testing set of the target variable (y), corresponding to the labels/categories for the testing data.

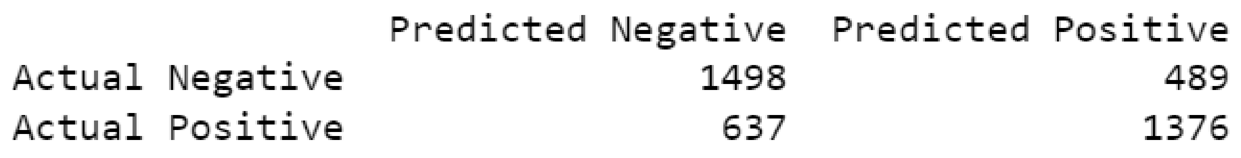
By splitting the data into training and testing sets, you can train a machine learning model on the training set and assess its performance on the unseen testing set. This approach helps evaluate the model's generalization ability and provides an estimate of its performance on new, unseen data.

Training the models and displaying the confusion matrix:

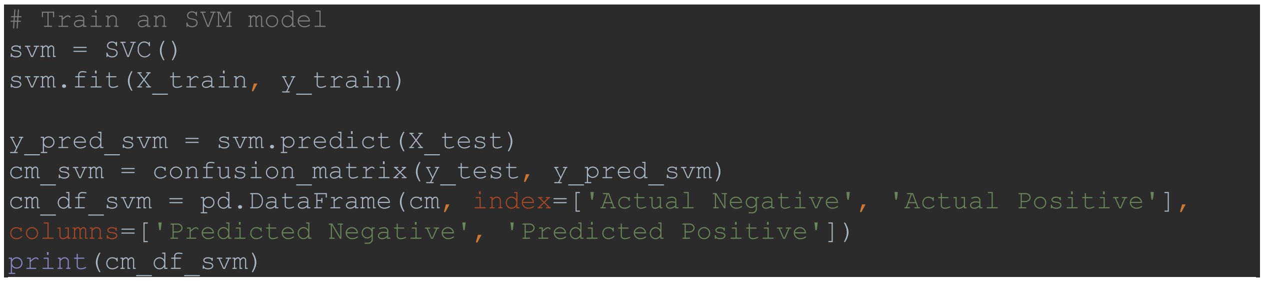


```python
# Train a Naive Bayes model
nb = MultinomialNB()
nb.fit(X_train, y_train)

y_pred_nb = nb.predict(X_test)
cm_nb = confusion_matrix(y_test, y_pred_nb)
cm_df_nb = pd.DataFrame(cm, index=['Actual Negative', 'Actual Positive'],
columns=['Predicted Negative', 'Predicted Positive'])
print(cm_df_nb)
```

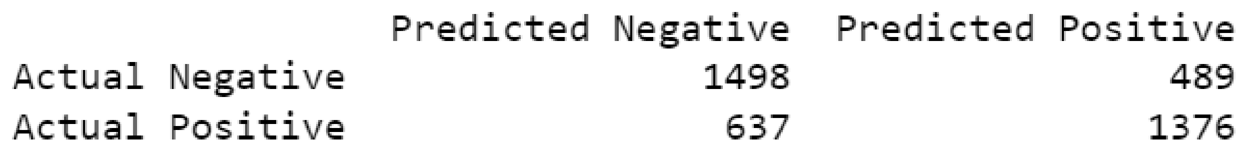


```
                 Predicted Negative  Predicted Positive
Actual Negative                1498                 489
Actual Positive                 637                1376
```

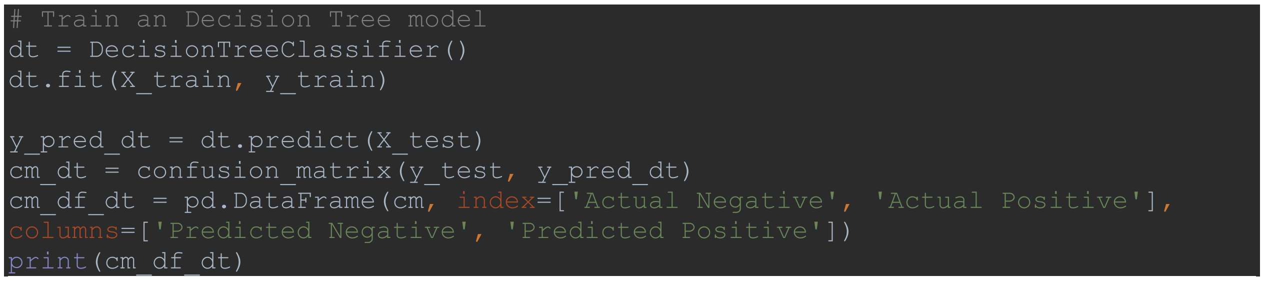


```python
# Train an SVM model
svm = SVC()
svm.fit(X_train, y_train)

y_pred_svm = svm.predict(X_test)
cm_svm = confusion_matrix(y_test, y_pred_svm)
cm_df_svm = pd.DataFrame(cm, index=['Actual Negative', 'Actual Positive'],
columns=['Predicted Negative', 'Predicted Positive'])
print(cm_df_svm)
```

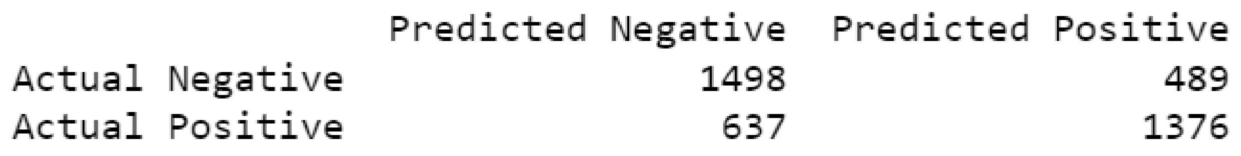


```
                 Predicted Negative  Predicted Positive
Actual Negative                1498                 489
Actual Positive                 637                1376
```



```python
# Train an Decision Tree model
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)

y_pred_dt = dt.predict(X_test)
cm_dt = confusion_matrix(y_test, y_pred_dt)
cm_df_dt = pd.DataFrame(cm, index=['Actual Negative', 'Actual Positive'],
columns=['Predicted Negative', 'Predicted Positive'])
print(cm_df_dt)
```



```
                 Predicted Negative  Predicted Positive
Actual Negative                1498                 489
Actual Positive                 637                1376
```

Creating the GUI:

```
root = tk.Tk()
root.geometry('600x200')
label = tk.Label(root, text='Enter some text to predict sentiment:')
label.pack(pady=10)
text_box = tk.Entry(root, width=50)
text_box.pack()
button = tk.Button(root, text='Predict', command=lambda:
predict_sentiment(text_box.get()))
button.pack(pady=10)
output_label = tk.Label(root, text='')
output_label.pack()
def predict_sentiment(text):
    # Vectorize the user input
    text_vec = vectorizer.transform([text])
    nb_sentiment = nb.predict(text_vec)[0]
    if nb_sentiment == 0:
        nb_sentiment_str = 'Negative'
    elif nb_sentiment == 2:
        nb_sentiment_str = 'Neutral'
    else:
        nb_sentiment_str = 'Positive'
    svm_sentiment = svm.predict(text_vec)[0]
    if svm_sentiment == 0:
        svm_sentiment_str = 'Negative'
    elif svm_sentiment == 2:
        svm_sentiment_str = 'Neutral'
    else:
        svm_sentiment_str = 'Positive'
    dt_sentiment = dt.predict(text_vec)[0]
    if dt_sentiment == 0:
        dt_sentiment_str = 'Negative'
    elif dt_sentiment == 2:
        dt_sentiment_str = 'Neutral'
    else:
        dt_sentiment_str = 'Positive'
    nbpred = nb.predict(X_test)
    nbf1 = f1_score(nbpred, y_test, average='weighted')
    svmpred = svm.predict(X_test)
    svmf1 = f1_score(svmpred, y_test, average='weighted')
    dtpred = dt.predict(X_test)
    dtf1 = f1_score(dtpred, y_test, average='weighted')
    if svmf1 > nbf1 and svmf1 > dtf1:
        model = 'Support Vector Machines (SVM)'
        prediction = svm_sentiment_str
    elif dtf1 > svmf1 and dtf1 > nbf1:
        model = 'Decision Tree'
        prediction = dt_sentiment_str
    else:
        model = 'Naive Bayes'
        prediction = nb_sentiment_str
    # Update the output label with the predicted sentiment and F1 score
    output_label.config(text=f'The model with the highest F1 score is
{model}, which predicted {prediction}.')
root.mainloop()
```