Introduction

This report aims to describe all information about the classification emotion function.

It contains 4 parts which are:

1). How to process the text input.

This part will include how to read the text file and the data preprocessing steps.

2). How to classify the emotions.

This part will explain which machine learning model are using in this task and how can the machine learning model can perform the result.

3). The results on the test set.

This part will show the result of the model and evaluate the result.

4). The UI system.

This part will describe the function and the workflow of the web application.

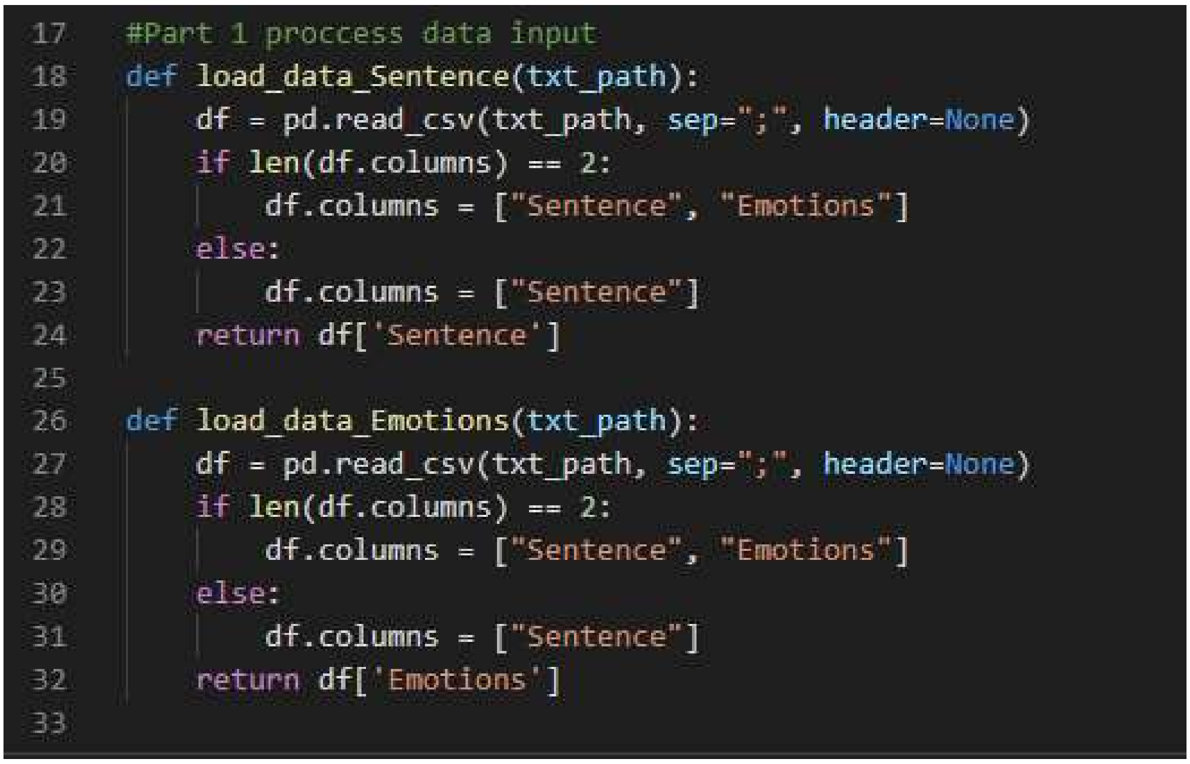
1. Process the Text Input

1.1 Read the text file

The text files input is basic on “pandas” data frame. In the training dataset, we can see the file contain text and emotion label which are in the same link and connected by “:”. So, we need to split these two texts into two columns which named “Sentence” column and “Emotion” column.

Besides, the testing dataset only contains sentence. Therefore, we cannot only design a function which always split two columns in a standardize way. We need to design a function that can split the dataset into one or two columns which satisfy the requirement of training and testing dataset. Therefore, we can use an If condition command to achieve our purpose.

Below are the code about text file input.

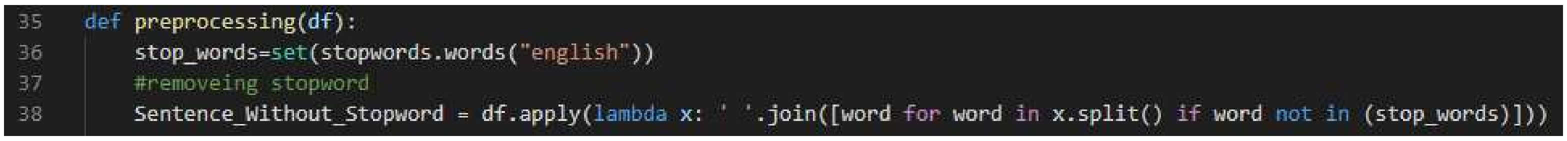


```python
#Part 1 proccess data input
def load_data_Sentence(txt_path):
    df = pd.read_csv(txt_path, sep=";", header=None)
    if len(df.columns) == 2:
        df.columns = ["Sentence", "Emotions"]
    else:
        df.columns = ["Sentence"]
    return df['Sentence']

def load_data_Emotions(txt_path):
    df = pd.read_csv(txt_path, sep=";", header=None)
    if len(df.columns) == 2:
        df.columns = ["Sentence", "Emotions"]
    else:
        df.columns = ["Sentence"]
    return df['Emotions']

```

The function will finally output two data frame which are df[‘Sentence’] and df[‘Emotion’]. df[‘Sentence’] is the data frame only contains sentence. On the other hand, df[‘Emotion’] is the data frame only contains emotion label.

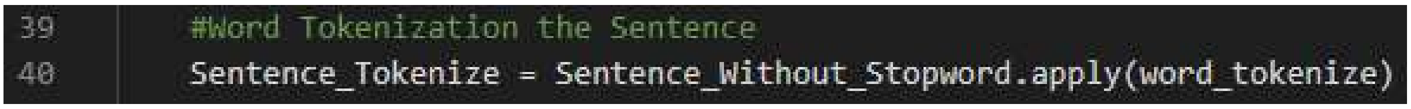
1.2 Data preprocessing

To process the classification, I firstly define a preprocessing function named "preprocessing". Then, we need to remove the stopwords which are the words need to filter out. The result is that stopwords is the noise for performing data analysis. For every rows in the data frame, the stopwords is deleted and then join all another words.



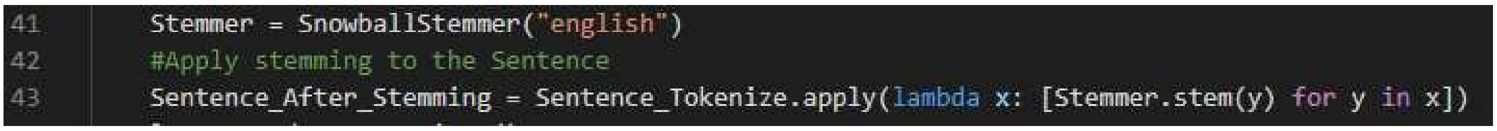
```python
def preprocessing(df):
    stop_words=set(stopwords.words("english"))
    #removeing stopword
    Sentence_Without_Stopword = df.apply(lambda x: ' '.join([word for word in x.split() if word not in (stop_words)]))
```

The next step is to tokenize the sentence which means we split the sentence words by words. This process is useful for the later step: Stemming and Lemmatization.



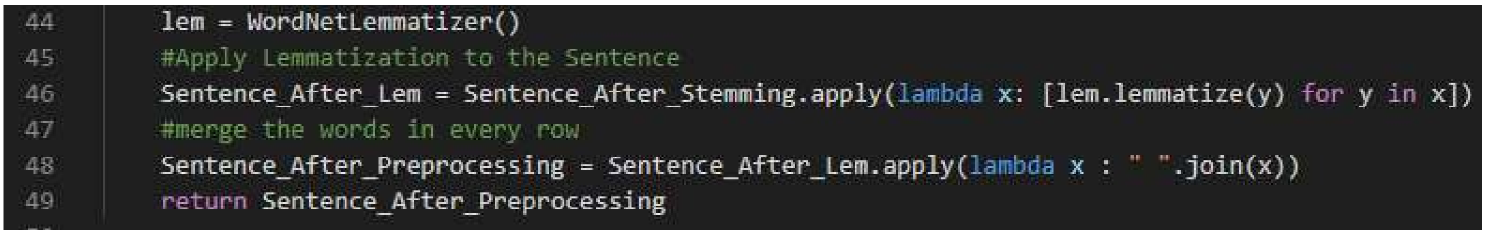
```python
    #Word Tokenization the Sentence
    Sentence_Tokenize = Sentence_Without_Stopword.apply(word_tokenize)
```

The third step for preprocessing is about stemming. Stemming is a naive version of morphological analysis which simply strip off affixes. This help for the machine learning process because we can classify the emotion label with the words although the word had different form.



```python
    Stemmer = SnowballStemmer("english")
    #Apply stemming to the Sentence
    Sentence_After_Stemming = Sentence_Tokenize.apply(lambda x: [Stemmer.stem(y) for y in x])
```

The final step for preprocessing is lemmatization. We use lemmatization to reduce inflectional forms and derivationally related forms of a word to a common lemmatized form. Lemmatization also have the same effect of stemming.



```python
    lem = WordNetLemmatizer()
    #Apply Lemmatization to the Sentence
    Sentence_After_Lem = Sentence_After_Stemming.apply(lambda x: [lem.lemmatize(y) for y in x])
    #merge the words in every row
    Sentence_After_Preprocessing = Sentence_After_Lem.apply(lambda x : " ".join(x))
    return Sentence_After_Preprocessing
```

Finally, before we preform machine learning modelling, we need to merge the word in every row. So, we can run the modelling step. The function will output a sentence without stopwords and every word in the sentence are the basic form.

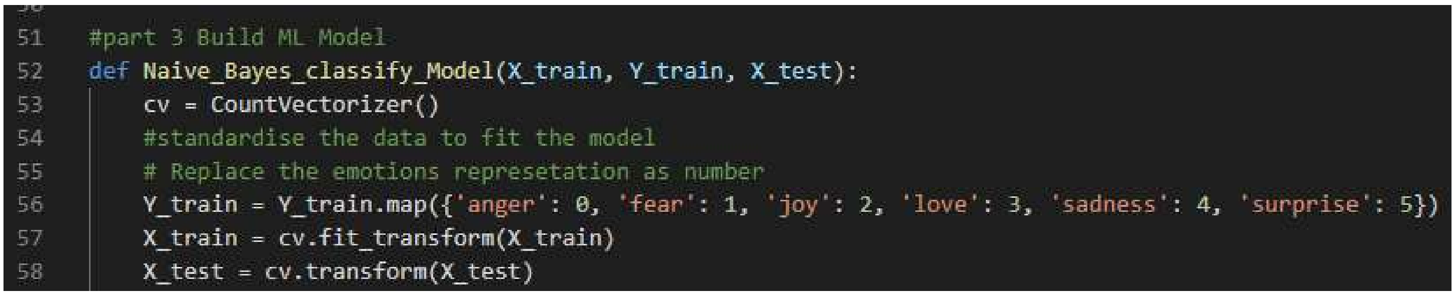
2. Classify the Emotions

This part is to fit a machine learning model to perform classification. We have two datasets which are sentence (X_train) and emotion (Y_train).

We firstly need to standardize the datasets. A function named CountVectorizer is defined. CountVectorizer aims to convert a collection of text documents to a matrix of token counts.

For the emotion (Y_train), since this dataset are expressed in word forms which are anger, fear, joy, love, sadness, or surprise. This expression cannot be performed machine learning model. Therefore, the expressions are converted from word to a number. That is 'anger' to 0, 'fear' to 1, 'joy' to 2, 'love' to 3, 'sadness' to 4 and 'surprise' to 5.

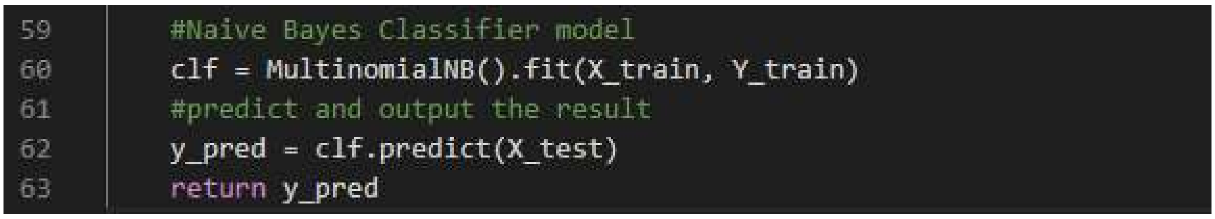
For the sentence (X_train), we convert the word into vector expression. That means the word dataset will convert to a matrix form. In the matrix form, each word are represented by a vector.



```
#part 3 Build ML Model
def Naive_Bayes_classify_Model(X_train, Y_train, X_test):
    cv = CountVectorizer()
    #standardise the data to fit the model
    # Replace the emotions represetation as number
    Y_train = Y_train.map({'anger': 0, 'fear': 1, 'joy': 2, 'love': 3, 'sadness': 4, 'surprise': 5})
    X_train = cv.fit_transform(X_train)
    X_test = cv.transform(X_test)
```

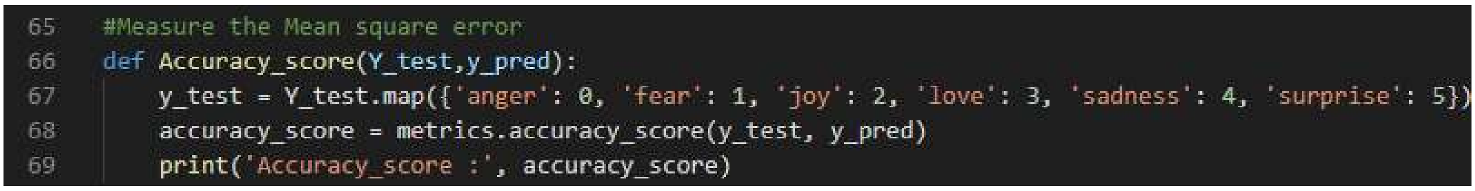
The machine learning model is naïve bayes classifier model which is suitable for classification with discrete features. That means naïve bayes is suitable for our dataset. And finally we output the predicted emotion label number.

The package we use is sklearn, Sklearn is a python package provides the machine learning model. Therefore, we can directly import the naïve bayes classifier model function and apply in our program.



```
    #Naive Bayes Classifier model
    clf = MultinomialNB().fit(X_train, Y_train)
    #predict and output the result
    y_pred = clf.predict(X_test)
    return y_pred
```

Furthermore, we use accuracy score to evaluate the model. Accuracy score is a statistics tool to computes the accuracy different between result data set and validation data set. The highest the score, the better the model.



```
#Measure the Mean square error
def Accuracy_score(Y_test,y_pred):
    y_test = Y_test.map({'anger': 0, 'fear': 1, 'joy': 2, 'love': 3, 'sadness': 4, 'surprise': 5})
    accuracy_score = metrics.accuracy_score(y_test, y_pred)
    print('Accuracy_score :', accuracy_score)
```

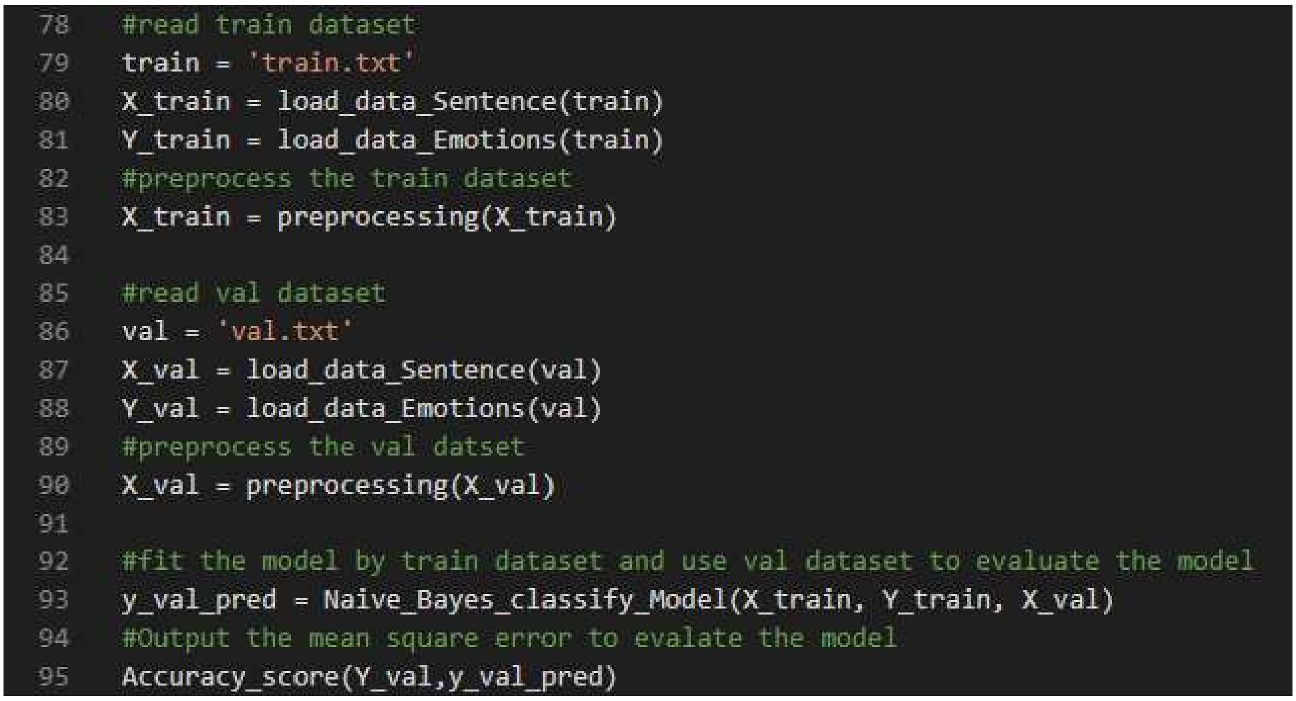
3. Results on the test set

After processing all steps that performed data input step, data preprocessing step, performed the naïve bayes classifier model, we use the training dataset and validation dataset to perform the accuracy score.

In our case, we firstly spilt the training dataset and validation dataset into two data frame Sentence (X_train, X_val) and Emotion (Y_train, Y_val) respectively.

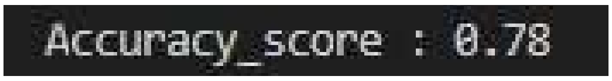
Then we preprocess the Sentence column of training dataset and validation dataset (X_train, X_val). We use the data frames (X_train, X_val and Y_train) to process naïve bayes classifier model. The model will output an emotion label about the validation dataset (X_val). So, we get a prediction of the validation dataset (y_val_pred).

Finally, we can test the model accuracy by using two set of data (y_val, y_val_pred).



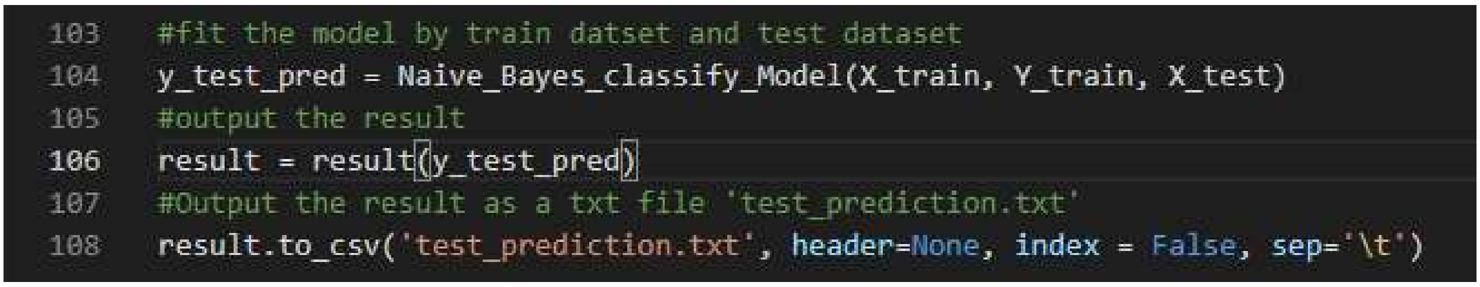
```
#read train dataset
train = 'train.txt'
X_train = load_data_Sentence(train)
Y_train = load_data_Emotions(train)
#preprocess the train dataset
X_train = preprocessing(X_train)

#read val dataset
val = 'val.txt'
X_val = load_data_Sentence(val)
Y_val = load_data_Emotions(val)
#preprocess the val datset
X_val = preprocessing(X_val)

#fit the model by train dataset and use val dataset to evaluate the model
y_val_pred = Naive_Bayes_classify_Model(X_train, Y_train, X_val)
#Output the mean square error to evalate the model
Accuracy_score(Y_val,y_val_pred)
```

Basic on the training dataset and validation dataset, our model gets the accuracy score result which is 0.78.



```
Accuracy_score : 0.78
```

Also, we get the emotion classification result of test dataset. The result of test dataset is stored in the file “test_prediction.txt”.



```python
#fit the model by train datset and test dataset
y_test_pred = Naive_Bayes_classify_Model(X_train, Y_train, X_test)
#output the result
result = result(y_test_pred)
#Output the result as a txt file 'test_prediction.txt'
result.to_csv('test_prediction.txt', header=None, index = False, sep='\t')
```

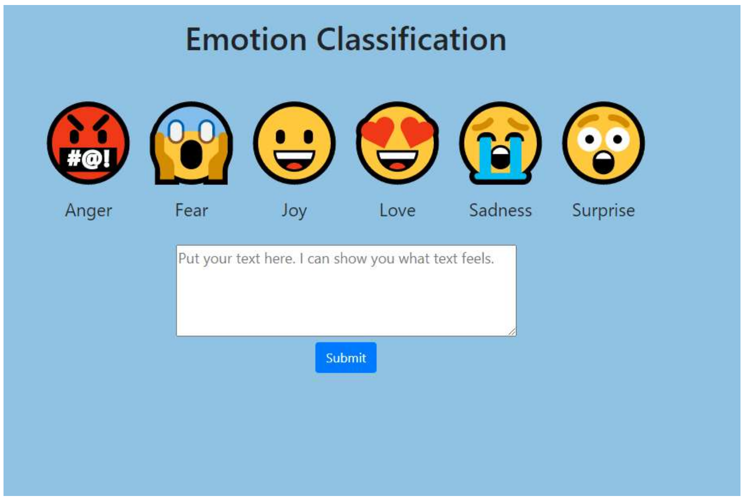
“test_prediction.txt” can refer to the enclosed zip file.

4. The UI system

A web application is developed to show the effect of classification emotion trained by our machine learning model. The UI system is developed by python flask and a web application framework (html and css).

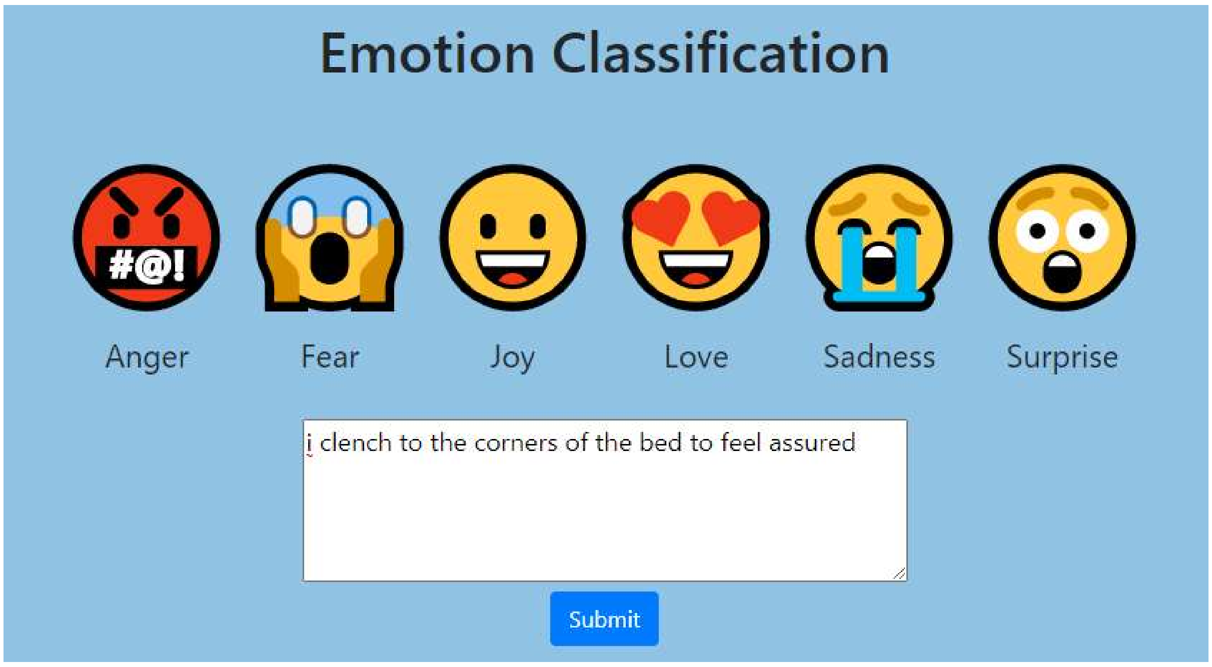
To run the web application, launch app.py and go to your endpoint http://localhost:5000/

This is the user interface. Each emoji represents an emotion label. You can also see a textbox which is for inputting text and a submit button.


Emotion Classification

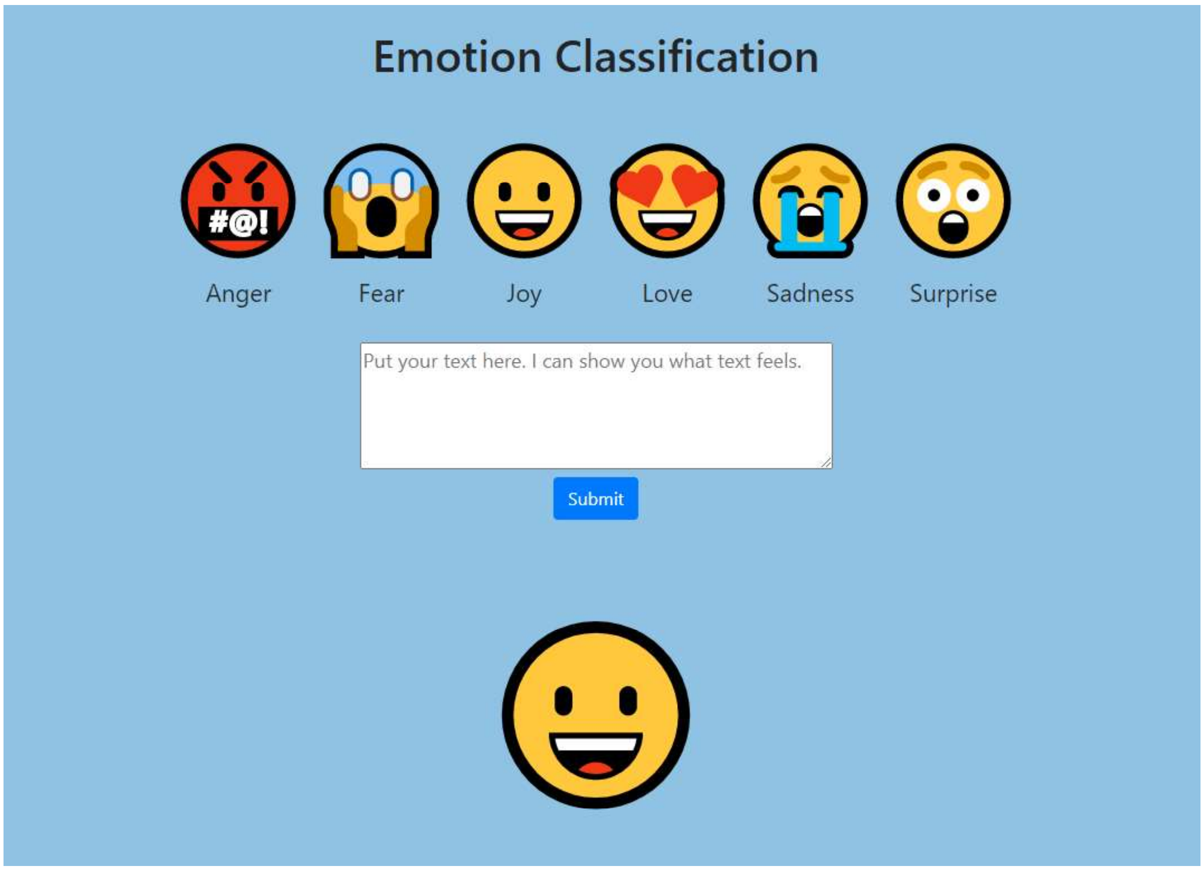
Put your text here. I can show you what text feels.

Firstly, we need to input a sentence in the text area. For example, i clench to the corners of the bed to feel assured. Then, click submit.


Emotion Classification

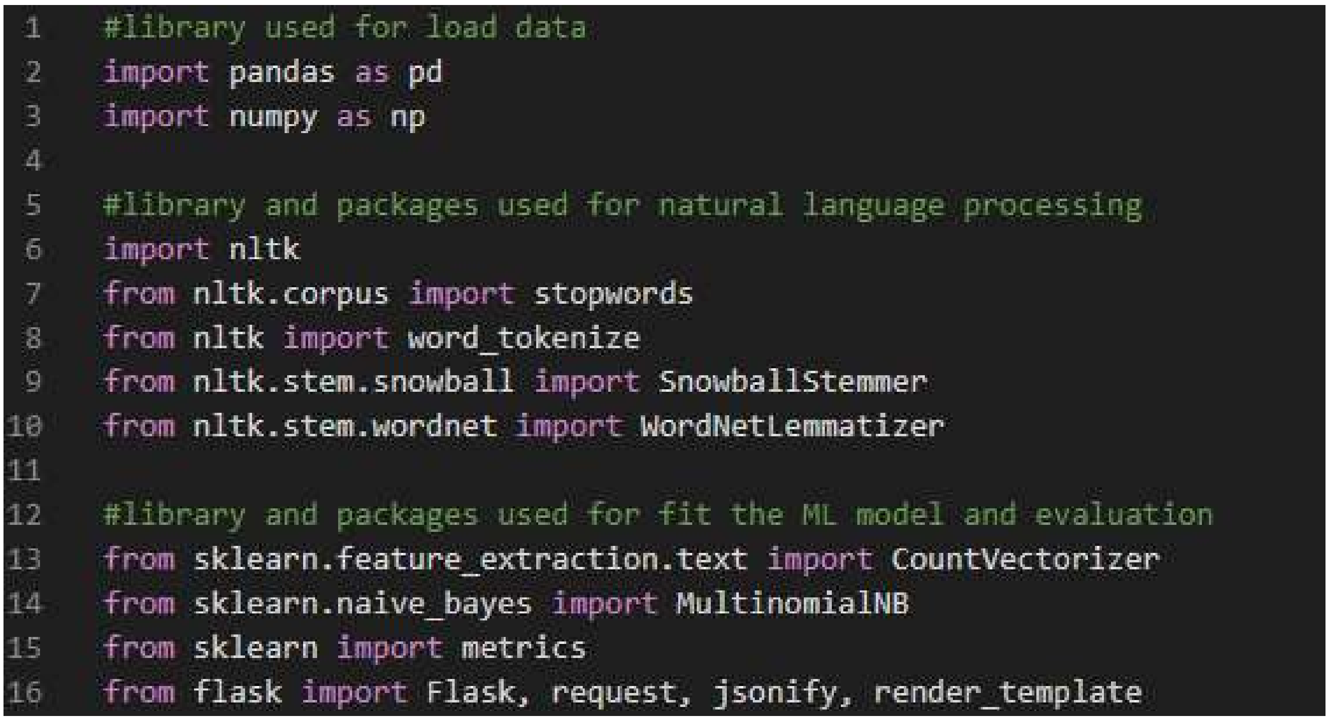
i clench to the corners of the bed to feel assured

The program will be ran and the input text will then perform naïve bayes classifier model. Finally, the web page output an emoji. You can refer the emoji with related emotion label. In our example, the emotional label is joy.



Appendix

The package input in the program



```python
#library used for load data
import pandas as pd
import numpy as np

#library and packages used for natural language processing
import nltk
from nltk.corpus import stopwords
from nltk import word_tokenize
from nltk.stem.snowball import SnowballStemmer
from nltk.stem.wordnet import WordNetLemmatizer

#library and packages used for fit the ML model and evaluation
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics
from flask import Flask, request, jsonify, render_template
```