AI Textual Emotion Recognition System

Report and code developed By:
Loai Bitawi

Supervisor:
Dr. Geovanni De Gasperis

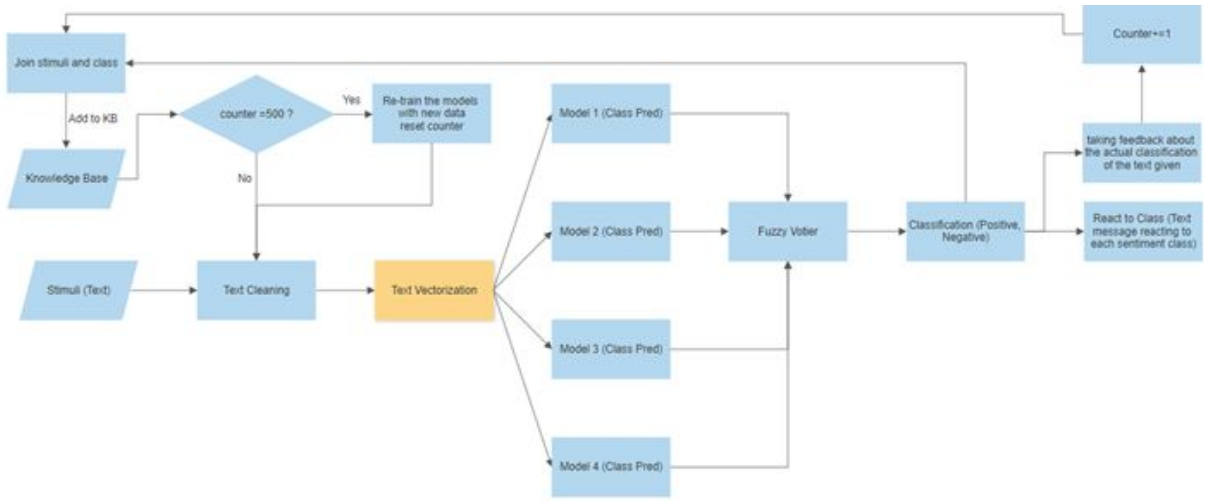
This project is submitted as a requirement fulfillment for Artificial Intelligence course at Arab American University, Palestine

Abstract:

The project will design an AI agent that uses different algorithms for sentiment analysis to classify the text whether it is positive ,neutral, or negative. The agent's components are:

- **Goals:** to correctly classify the text given as a stimulus and react to it
- Abilities: the agent is able to take a text with any length, vectorize it, and classify it. It can react to the sentiment given by sending a text message to the environment.
- **Stimuli:** input text to be analyzed and classified
- **Action:** text reaction to classification result (positive, negative). The agent also requests validation from environment about the actual class of the text
- **Controller:** four pre-trained supervised models to classify the stimuli text, then results (in the form of a probability) are input to the voting function for the final decision. The voting uses fuzzy logic to define the positive, neutral, and negative regions, and calculates the centroid of the area results from interacting with classification algorithm results.
- **Knowledge Base:** the knowledge that the agent uses was taken from "Sentiment140" dataset that contains 1.6 million tweets with its sentiment (positive or negative). The data is sampled and used for models training. (the ratio used for proof of concept is 1% of the whole dataset. However, this ratio can be increased with high computational power machines.
- **Past Experience:** a set of previously classified stimuli, and the actual classifications of these stimuli. The data is fed to the knowledge base. For each new 500 entries, the agent will re-train and test its model with new dataset, which would result in better performance

The following figure represents the agent's work flow:



The code:

Initialization of the Agent and Data Preprocessing:

```
# Imports
import pandas as pd
import re
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
import numpy as np
import skfuzzy as fuzz
```

At first, the initialization of the agent includes define needed variables, constructing the knowledge base, data preprocessing, and models initialization:

```
# Agent Class
class Emotion_Agent(object):
    # Initialization
    def __init__(self):
        self.Text=[]
        self.counter=0
        self.data=open('Data.txt').readlines()
        self.features=[]
        self.labels=[]
        for i in range(len(self.data)):
self.features.append(re.compile('".*",*').search(self.data[i]).group(0))
self.labels.append(re.compile('";.\n').search(self.data[i]).group(0))
self.processed_features = self.data_cleaner(self.features)
# Data Cleaning
self.processed_labels=self.data_cleaner(self.labels)
self.Processed_data=pd.DataFrame(data=[self.processed_features,self.process
ed_labels]).T
self.Processed_data.columns=['Text','Sentiment']
self.Processed_data['Sentiment']=self.Processed_data['Sentiment'].astype(in
t)
# sampling the data. I took 0.005 of the dataset due to memory limitations
self.sampled_data=self.Processed_data.sample(frac=0.005).reset_index(drop=T
rue)
self.initializer(self.sampled_data)
```

Data cleaner function performs a set of actions that involve removing special characters, single characters, multiple spaces, and converting to lower case for the analysis:

```
# Data Cleaning Function
def data_cleaner(self,Data):
processed_features = []
for sentence in range(0, len(Data)):
# Remove all the special characters
processed_feature = re.sub(r'\W', ' ', str(Data[sentence]))
# remove all single characters
processed_feature= re.sub(r'\s+[a-zA-Z]\s+', ' ', processed_feature)
# Remove single characters from the start
processed_feature = re.sub(r'\^[a-zA-Z]\s+', ' ', processed_feature)
# Substituting multiple spaces with single space
processed_feature = re.sub(r'\s+', ' ', processed_feature, flags=re.I)
# Removing prefixed 'b'
processed_feature = re.sub(r'^b\s+', '', processed_feature)
# Converting to Lowercase
processed_feature = processed_feature.lower()
processed_features.append(processed_feature)
return processed_features
```

Initializer function performs vectorization to the knowledge base using Term Frequency, Inverse Document Frequency (TFIDF vectorizer), then splitting the knowledge base to training and testing set, then initializing and training classification algorithms:

```
 def initializer(self,sampled_d):
        # Vectorizing the Data
        self.vectorizer = TfidfVectorizer
(max_features=2000,min_df=7,max_df=0.8,
stop_words=stopwords.words('english'))
        self.vectorizer.fit(sampled_d['Text'])
        self.vectorized_features=self.vectorization(sampled_d['Text'])
        # Training Models
        self.X_train, self.X_test, self.y_train, self.y_test =
train_test_split(self.vectorized_features , self.sampled_data['Sentiment'],
test_size=0.2, random_state=0)
        self.model_1=self.model_1_init(self.X_train, self.X_test,
self.y_train, self.y_test)
```

```
        self.model_2=self.model_2_init(self.X_train, self.X_test,
self.y_train, self.y_test)
        self.model_3=self.model_3_init(self.X_train, self.X_test,
self.y_train, self.y_test)
        self.model_4=self.model_4_init(self.X_train, self.X_test,
self.y_train, self.y_test)
        return
```

The classification models used are Gradient Boosting Classifier, MLP Classifier, K-Nearest Neighbor and Logistic Regression. The implementation code:

```
    # Model 1 Initialization (Gradient Boosting Classifier)
    def model_1_init (self,X_train, X_test, y_train, y_test):
        from sklearn.ensemble import GradientBoostingClassifier
        from sklearn.metrics import accuracy_score
        model_1 = GradientBoostingClassifier()
        model_1.fit(X_train, y_train)
        predictions = model_1.predict(X_test)
        from sklearn.metrics import classification_report, confusion_matrix
        print('Model 1 initialization (Gradient Boosting)')
        print(accuracy_score(y_test, predictions))
        print(confusion_matrix(y_test,predictions))
        print(classification_report(y_test,predictions))
        print('----------------------')
        return model_1

    # Model 2 Initialization (MLP classifier)
    def model_2_init (self,X_train, X_test, y_train, y_test):
        from sklearn.neural_network import MLPClassifier

model_2=MLPClassifier(hidden_layer_sizes=1,max_iter=100).fit(X_train,
y_train)
        model_2.score(X_test, y_test)
        print('Model 2 initialization (MLP Classifier)')
        print(model_2.score(X_test,y_test))
        print('----------------------')
        return model_2

    # Model 3 Initialization (SGD Classifier)
    def model_3_init (self,X_train, X_test, y_train, y_test):
        from sklearn.linear_model import SGDClassifier
```

```
        model_3= SGDClassifier(loss='log').fit(X_train,y_train)
        print('model 3 initialization (SGD Classifier)')
        print(model_3.score(X_test,y_test))
        print('----------------------')
        return model_3

    # Model 4 Initialization (Logistic Regression)
    def model_4_init (self,X_train, X_test, y_train, y_test):
        from sklearn.linear_model import LogisticRegression
        model_4=LogisticRegression(solver="lbfgs").fit(X_train,y_train)
        print('model 4 initialization (Logistic Regression)')
        print(model_4.score(X_test,y_test))
        print('----------------------')
        return model_4
```

After this step, the agent is ready to interact with the environment, the classification algorithms scores are printed:

```
Model 1 initialization (Gradient Boosting)
              precision    recall  f1-score   support

           0       0.62      0.87      0.72       825
           4       0.76      0.43      0.55       775

    accuracy                           0.66      1600
   macro avg       0.69      0.65      0.64      1600
weighted avg       0.69      0.66      0.64      1600

----------------------
Model 2 initialization (MLP Classifier)
0.72875
----------------------
model 3 initialization (KNN)
0.7225
----------------------
model 4 initialization (Logistic Regression)
0.7325
----------------------
```

The average accuracy of the model goes around 71%, which is acceptable. However, the models were implemented without any performance enhancement and using default hyperparameters. However, there is an opportunity to increase the performance.

Another function needed for initialization is the vectorization function. It uses the trained TFIDF model to transform the input text (knowledge base in the initialization phase, and the percept in the running phase) to convert the text into vector representation:

```
    # Vectorization Function
    def vectorization (self,Data):
        return self.vectorizer.transform(Data).toarray()
```

The final step in the initialization phase is preparing the fuzzy voter. It uses fuzzy logic to take a decision about the text. Instead of providing either positive, neutral or negative sentiment, it provides the percentage of positivity or negativity in the text.

```
# Fuzzy Voter
    def voter (self,C1_prob,C2_prob,C3_prob,C4_prob):
        C1_val=C1_prob
        C2_val=C2_prob
        C3_val=C3_prob
        C4_val=C4_prob

        # Universe
        C1 = np.arange(0,1.05,0.05)
        C2 = np.arange(0,1.05,0.05)
        C3 = np.arange(0,1.05,0.05)
        C4 = np.arange(0,1.05,0.05)
        class_result= np.arange(0,1.1,0.1)
        # Membership Functions
        C1_pos= fuzz.trimf(C1, [0.5,1,1])
        C1_ntrl= fuzz.trimf(C1, [0.5,0.5,0.5])
        C1_neg= fuzz.trimf(C1, [0,0,0.5])

        C2_pos= fuzz.trimf(C2, [0.5,1,1])
        C2_ntrl= fuzz.trimf(C2, [0.5,0.5,0.5])
        C2_neg= fuzz.trimf(C2, [0,0,0.5])

        C3_pos= fuzz.trimf(C3, [0.5,1,1])
        C3_ntrl= fuzz.trimf(C3, [0.5,0.5,0.5])
        C3_neg= fuzz.trimf(C3, [0,0,0.5])
```

```
        C4_pos= fuzz.trimf(C4, [0.5,1,1])
        C4_ntrl= fuzz.trimf(C4, [0.5,0.5,0.5])
        C4_neg= fuzz.trimf(C4, [0,0,0.5])

        class_pos= fuzz.trimf(class_result, [0.5,1,1])
        class_ntrl= fuzz.trimf(class_result,[0.5,0.5,0.5])
        class_neg= fuzz.trimf(class_result, [0,0,0.5])


        # Membership Functions Interpretation With Given Values
        C1_level_neg = fuzz.interp_membership(C1, C1_neg, C1_val)
        C1_level_ntrl = fuzz.interp_membership(C1,C1_ntrl, C1_val)
        C1_level_pos = fuzz.interp_membership(C1, C1_pos, C1_val)

        C2_level_neg = fuzz.interp_membership(C2, C2_neg, C2_val)
        C2_level_ntrl = fuzz.interp_membership(C2,C2_ntrl, C2_val)
        C2_level_pos = fuzz.interp_membership(C2, C2_pos, C2_val)

        C3_level_neg = fuzz.interp_membership(C3, C3_neg, C3_val)
        C3_level_ntrl = fuzz.interp_membership(C3,C3_ntrl, C3_val)
        C3_level_pos = fuzz.interp_membership(C3, C3_pos, C3_val)

        C4_level_neg = fuzz.interp_membership(C4, C4_neg, C4_val)
        C4_level_ntrl = fuzz.interp_membership(C4, C4_ntrl, C4_val)
        C4_level_pos = fuzz.interp_membership(C4, C4_pos, C4_val)

        # Rule Creation
        active_rule_neg = np.fmax(C1_level_neg, np.fmax(C2_level_neg,
np.fmax(C3_level_neg,C4_level_neg)))
        Class_activation_neg = np.fmin(active_rule_neg,class_neg)

        active_rule_ntrl = np.fmax(C1_level_ntrl, np.fmax(C2_level_ntrl,
np.fmax(C3_level_ntrl,C4_level_ntrl)))
        Class_activation_ntrl = np.fmin(active_rule_ntrl,class_ntrl)

        active_rule_pos = np.fmax(C1_level_pos, np.fmax(C2_level_pos,
np.fmax(C3_level_pos,C4_level_pos)))
        Class_activation_pos = np.fmin(active_rule_pos, class_pos)

        aggregated = np.fmax(Class_activation_neg,
np.fmax(Class_activation_ntrl, Class_activation_pos))
```

```
        # Calculate Defuzzified Result
        classification = fuzz.defuzz(class_result, aggregated,'centroid' )
        return classification
```

Action and Memory Phase:

In this phase, the agent takes the input text as a stimuli, cleans it, vectorizes it, then predicts the sentiment through the 4 mentioned classification algorithms, then sends the result to the fuzzy voter to get the final decision.

After that, the agent requests feedback from the environment to check if the predicted sentiment is correct or not, and stores the feedback in the knowledge base for future learning. Due to computational limitations and the time needed for the model to train itself, I have set the retraining on the new entries to be after 500 new entries, and removing 500 entries from the knowledge base in order not to increase computation time.

```
# Action Function
    def Analyze(self,Text):
        self.cleaned_Text=self.data_cleaner(Text)
        self.vectorized_text=self.vectorization(self.cleaned_Text)
        self.C1=self.model_1.predict_proba(self.vectorized_text)
        self.C2=self.model_2.predict_proba(self.vectorized_text)
        self.C3=self.model_3.predict_proba(self.vectorized_text)
        self.C4=self.model_4.predict_proba(self.vectorized_text)
        print('probabilities are: ',self.C1,self.C2,self.C3,self.C4)

result=self.voter(self.C1[0][1],self.C2[0][1],self.C3[0][1],self.C4[0][1])
        print('---------------------------------')
        if result > 0.6:
            print ('The Text you gave me is ','{0:.0%}'.format(result),
'Positive. Keep up the positivity :) ')
        elif result <0.4:
            print("The text is only
","{0:.0%}".format(1-result),"Negative!. \n Why don't you cheer up a little
bit! ^_^")
        else:
            print("This is confusing!. The Text you gave me is
","{0:.0%}".format(result),"Positive. \n It's neutral to me. \n Anyway,
Don't lose positive Vibes")
```

```
#update the Knowledge Base to increase efficiency, actual classes are taken
as input
        while 1:
            self.Actual_class= input('what is the actual sentiment of the
text? (P: Positive, U: Neutral, N: Negative)').lower()
            if self.Actual_class not in ['p','n','u']:
                print('Wrong Input, Try Again!')
                continue
            elif
self.sampled_data['Text'].eq(str(self.cleaned_Text)).sum():
                break
            elif self.Actual_class =='p':
                self.Actual_class=4
                self.to_append=pd.DataFrame(data={'Sentiment':
int(self.Actual_class),'Text':str(self.cleaned_Text)},index=['0'])

self.sampled_data=self.sampled_data.append(self.to_append,ignore_index=True
)
                self.counter+=1
                break
            elif self.Actual_class == 'u':
                self.Actual_class=2
                self.to_append=pd.DataFrame(data={'Sentiment':
int(self.Actual_class),'Text':str(self.cleaned_Text)},index=['0'])

self.sampled_data=self.sampled_data.append(self.to_append,ignore_index=True
)
                self.counter+=1
                break
            elif self.Actual_class == 'n':
                self.Actual_class=0
                self.to_append=pd.DataFrame(data={'Sentiment':
int(self.Actual_class),'Text':str(self.cleaned_Text)},index=['0'])

self.sampled_data=self.sampled_data.append(self.to_append,ignore_index=True
)
                self.counter+=1
                break

        # Re-train The Model When There Are 500 New Samples
        if self.counter == 500 :
            print('New samples are being learned, Kindly wait... ')
```

```
            self.sampled_data.drop(labels=np.arange(0,500,1),inplace=True)
# to keep running performance the same, can be omitted
            self.initializer(self.sampled_data)
            self.counter=0
            print('Learning finished, back online!')


        return result
```

Environment:

The Environment is simple, just calling the Agent, and calling the analyze function with providing it with the text to analyze:

```
'---------- Environment ----------'
Agent=Emotion_Agent()
while 1:
    Text=[input('Enter the text to analyze: \n')]
    Agent.Analyze(Text)
```

The python console will ask for an input text, then it passes it to the agent through analyze function to get the result:

Example 1:

```
Please Enter the text you want to Analyze:
i just finished coding and the code works very well, glad to see good
results after hard working
probabilities are:  [[0.49521374 0.50478626]] [[0.51594646 0.48405354]]
[[0.19421531 0.80578469]] [[0.34473665 0.65526335]]
--------------------------------
The Text you gave me is  70% Positive. Keep up the positivity :)

What is the actual sentiment of the text? (P: Positive, U: Neutral, N:
Negative)p
```

Example 2:

```
Please Enter the text you want to Analyze:
i lost my friend. he died with COVID 19
probabilities are:  [[0.61986579 0.38013421]] [[0.67517322 0.32482678]]
[[0.73139041 0.26860959]] [[0.62722029 0.37277971]]
---------------------------------
The text is only  80% Negative!.
 Why don't you cheer up a little bit! ^_^

What is the actual sentiment of the text? (P: Positive, U: Neutral, N:
Negative)N
Out[18]: 0.19724343570930278
```

Example 3:

Please Enter the text you want to Analyze:

```
my name is ibrahim
probabilities are:  [[4.92935417e-14 1.00000000e+00 4.98922866e-14]]
[[0.57070638 0.19009252 0.23920109]] [[0.10422893 0.74456567 0.15120539]]
[[0.44096828 0.04110371 0.51792801]]
---------------------------------
This is confusing!. The Text you gave me is  50% Positive.
 It's neutral to me.
 Anyway, Don't lose positive Vibes

What is the actual sentiment of the text? (P: Positive, U: Neutral, N:
Negative)u
Out[23]: 0.5038681449212522
```

Discussion:
The agent in his initial state has a good performance, it is able to predict the emotions in the given text correctly up to 70% of the time. However, the performance will automatically be enhanced by the learning of the algorithm each time it retrains itself. Finally, enhancements on the classification algorithms can be made which would enhance the output of the algorithm.