Univerzitet Donja Gorica

Fakultet za informacione sisteme i tehnologije

Podgorica

Emotion recognition in the speech

AI4S3 courses

Mentor:
Ivan Jovović

Oleksandr Galushko 23/066

Contents

I. Introduction

Python 3 has emerged as a preferred language for a multitude of applications, including machine learning and natural language processing (NLP). Its popularity can be attributed to several factors that make it well-suited for these advanced projects.

Readability and Expressiveness: Python's syntax is clean and readable, making it easier for developers to express complex ideas concisely. This readability is crucial for projects like machine learning and NLP, where intricate algorithms and data manipulations are common.

Extensive Libraries and Frameworks: Python boasts a rich ecosystem of libraries and frameworks specifically designed for machine learning and NLP. Libraries like TensorFlow, PyTorch, and scikit-learn offer powerful tools for building and deploying machine learning models. In NLP, spaCy and NLTK are popular choices for text processing and analysis.

Data Science Capabilities: Python's popularity in data science aligns well with the requirements of machine learning and NLP projects. Libraries like Pandas facilitate data manipulation and analysis, while Jupyter Notebooks provide an interactive and collaborative environment for experimentation and analysis. Strong Support for AI and Machine Learning Research: Python is a preferred language for AI and machine learning research, with many academic institutions adopting it as the primary language for research projects. This has led to a wealth of research-focused libraries and resources that benefit practitioners in the field. State-of-the-Art NLP Libraries: For NLP projects, Python offers state-of-the-art libraries like spaCy and NLTK, which provide pre-trained models, efficient text processing capabilities, and tools for building custom models.

The project of SpeakUp contains a very promising idea of working with people and being a mental assistent to them. The ability to recieve the user’s speech to transcribe it and use the NLP model together with the list of examples to get the emotion prediction is what makes the project special in a certain way. In the future

more advanced models will be able to communicate with people in a deep way considering the feelings the person has at the moment. The computer will make his way out to people's hearts and consult them on their problem. That is more advanced idea, and for now there is emotion recognition which will work solely with the text from the speech.

II. Materials

The materials used in the project include such as libraries, datasets and additional sources are following:

Libraries:

Pandas:

Pandas is a powerful data manipulation and analysis library in Python. It provides data structures like DataFrame and Series, making it easy to handle and analyze structured data.

Pandas is used to read and manipulate the emotion dataset. It helps in loading data, creating a DataFrame, and performing operations such as value counts and data preprocessing.

NumPy:

NumPy is a fundamental package for scientific computing with Python. It provides support for large, multi-dimensional arrays and matrices, along with mathematical functions to operate on these arrays.

While not explicitly referenced in the code, NumPy is often used implicitly through other libraries like Pandas and Scikit-learn for numerical operations on data.

Scikit-learn:

Scikit-learn is a machine learning library that provides simple and efficient tools for data analysis and modeling, including classification, regression, clustering, and more.

Scikit-learn is utilized for building a machine learning pipeline. The RandomForestClassifier is used for training and predicting emotions based on the processed text data.

Spacy:

spaCy is a natural language processing (NLP) library that simplifies tasks such as tokenization, part-of-speech tagging, and entity recognition.

spaCy is employed for text preprocessing in the project. It tokenizes and lemmatizes(the process of grouping together different inflected forms of the same word) the text data, extracting essential features for the machine learning model.

Whisper:

Whisper is a library for automatic speech recognition (ASR). It converts spoken language into written text, facilitating the processing of audio data. In the project, Whisper is used for transcribing audio data. This is particularly valuable for scenarios where emotions are conveyed through spoken words.

Sounddevice, PyDub:

Sounddevice and PyDub are libraries for working with audio data. Sounddevice is used for recording audio, while PyDub provides tools for audio file manipulation.

Usage: These libraries are applied in the project for capturing and processing audio input. This includes recording spoken sentences, saving the audio as an MP3 file, and exporting audio segments for further analysis.

Dataset:

The information for training the project's algorithm came from a place called Kaggle, and specifically from a dataset named "Emotions Dataset for NLP." I used a part of the dataset called "train.txt." You can find this dataset on Kaggle's website at this link: https://www.kaggle.com/datasets/praveengovi/emotions-dataset-for-nlp. The dataset is like a collection of information stored in a special file, and I looked at it using a tool called Pandas. By using this dataset, I could carefully look at and get the data ready for teaching the computer to understand emotions in text.

```
df = pd.read_csv("archive/train.txt",sep=";",
    names=["Description","Emotion"])
```

To get the simplest information about the whole dataset “train.txt” I used Pandas function called “.info()”. As it will be seen, that output shows us that there are 16000 entries and 2 columns. And to give a clear view of what we are dealing here with we

are using another function to display top 5 first values of the dataset in the table ".head()".

```
df.head(5)
```

✓ 0.1s

	Description	Emotion
0	i didnt feel humiliated	sadness
1	i can go from feeling so hopeless to so damned...	sadness
2	im grabbing a minute to post i feel greedy wrong	anger
3	i am ever feeling nostalgic about the fireplac...	love
4	i am feeling grouchy	anger

To see the total count for beautiful representation and once more to see the abilities of the Pandas library the following code will present the output:

```
df['Emotion'].value_counts()
```

```
Emotion
joy         5362
sadness     4666
anger       2159
fear        1937
love        1304
surprise     572
Name: count, dtype: int64
```

Machine learning algorithms typically work with numerical data. Assigning numerical values to emotions allows the model to process and analyze the data more effectively. And to organize our code for the future learning we must add the specific numeration for each of the emotions. The numerical representation makes it easier to evaluate the model's performance using metrics like accuracy, precision, recall, and confusion matrices. It also simplifies the interpretation of model predictions. This is an incredibly crucial part of the whole process.

```
df['label_num'] = df['Emotion'].map({
    'joy' : 0,
    'sadness': 1,
    'anger': 2,
    'fear': 3,
    'love': 4,
    'surprise':5
})

df.head(5)
```

	Description	Emotion	label_num
0	i didnt feel humiliated	sadness	1
1	i can go from feeling so hopeless to so damned...	sadness	1
2	im grabbing a minute to post i feel greedy wrong	anger	2
3	i am ever feeling nostalgic about the fireplac...	love	4
4	i am feeling grouchy	anger	2

III. Model Training

Collecting the data

This process starts with collecting the training data. Here is where the Scikit-learn(sklearn) library comes in help. In this code, df.Description represents the input features, which are the textual descriptions, and df.label_num represents the target variable, which is the numerical representation of emotions assigned to each description and the *test_size* shows the 20% used for testing and 80% of the dataset for training accordingly.

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(df.Description,df.label_num,test_size=0.2)
```

The train_test_split function is used to split the dataset into training and testing sets. The training data (X_train and y_train) are used to train the machine learning model, while the testing data (X_test and y_test) are used to evaluate the model's performance.

```
import spacy
nlp = spacy.load("en_core_web_sm")
```

import spacy: This line imports the spaCy library into your Python script or Jupyter Notebook. spaCy is a natural language processing (NLP) library that provides pre-trained models and tools for processing and understanding human language.

nlp = spacy.load("en_core_web_sm"): This line initializes a spaCy language processing pipeline using the English language model named "en_core_web_sm." The "sm" in the model name stands for "small," indicating that it's a smaller and faster model suitable for basic NLP tasks.

```
def preprocess(text):
    doc = nlp(text)
    filtered_tokens = []
    for token in doc:
        if token.is_stop or token.is_punct:
            continue
        else:
            filtered_tokens.append(token.lemma_)
    return " ".join(filtered_tokens)


df['processed_text'] = df["Description"].apply(preprocess)
```

Tokenization with spaCy:

doc = nlp(text): The input text is processed using spaCy's language model (nlp). This step tokenizes the text, meaning it breaks it down into individual words or tokens and performs other linguistic analyses.

Filtering Stop Words and Punctuation:

The function then iterates through each token in the processed document (doc).

if token.is_stop or token.is_punct:: Checks if the token is a stop word or punctuation. If it is, the token is skipped (not included in the filtered output).

Lemmatization:

filtered_tokens.append(token.lemma_): If the token is not a stop word or punctuation, its lemma (base form) is appended to the list of filtered_tokens.

Joining Tokens:

return " ".join(filtered_tokens): The list of lemmatized tokens is joined into a single string, where tokens are separated by spaces. The resulting string is then returned as the preprocessed text.

In summary, the preprocess function takes an input text, processes it with spaCy, removes stop words and punctuation, lemmatizes the remaining tokens, and returns

the preprocessed text as a string. This kind of preprocessing is common in NLP tasks to clean and standardize the text data before further analysis or modeling. And in the next lines of code we're taking our raw text descriptions, cleaning them up with the preprocess function, and saving the cleaned-up versions in a new column called "processed_text." This cleaned-up text is often used as input for machine learning models.

Training the model.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import classification_report


clf = Pipeline([
     ('vectorizer_tfidf',TfidfVectorizer()),
     ('Random Forest', RandomForestClassifier())
])

clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
y_pred_rf = clf.predict(X_test)

print(classification_report(y_test, y_pred))
```

- First we import needed libraries from Scikit-learn.

Then goes clf:

- The first step is TfidfVectorizer, which converts a collection of raw documents to a matrix of TF-IDF features. *“TF-IDF stands for Term Frequency Inverse Document Frequency of records. It can be defined as the calculation of how relevant a word in a series or corpus is to a text. The meaning increases proportionally to the number of times in the text a word appears but is compensated by the word frequency in the data-set”.* It transforms the text data into numerical features suitable for machine learning.
- The second step is RandomForestClassifier, which is an ensemble of decision trees used for classification. *“The Random forest classifier creates a set of decision trees from a randomly selected subset of the training set. It is basically a set of decision trees (DT) from a randomly selected subset of the training set and then It collects the votes from different decision trees to decide the final prediction.”*

The training with function .fit():

- Training the model on the training data (X_train is the processed text and y_train is the corresponding emotion labels).
- And the predictions are made.
-

	precision	recall	f1-score	support
0	0.81	0.94	0.87	1065
1	0.92	0.89	0.90	921
2	0.90	0.83	0.86	423
3	0.88	0.84	0.86	421
4	0.85	0.63	0.73	249
5	0.84	0.70	0.77	121
accuracy			0.86	3200
macro avg	0.87	0.80	0.83	3200
weighted avg	0.87	0.86	0.86	3200

At last there is classification_report, which provides metrics such as precision, recall, and F1-score to evaluate the performance of the model on the test data.

As it can be seen, the overall accuracy was 86%. The Recall shows how many in the data collection that was completely true the model identified correctly. And as the report shows the recall was great for class 0 and 1.

Confusion Matrix

The confusion matrix provides insights into where the model is making correct predictions and where it may be making errors, helping you understand its strengths and weaknesses.

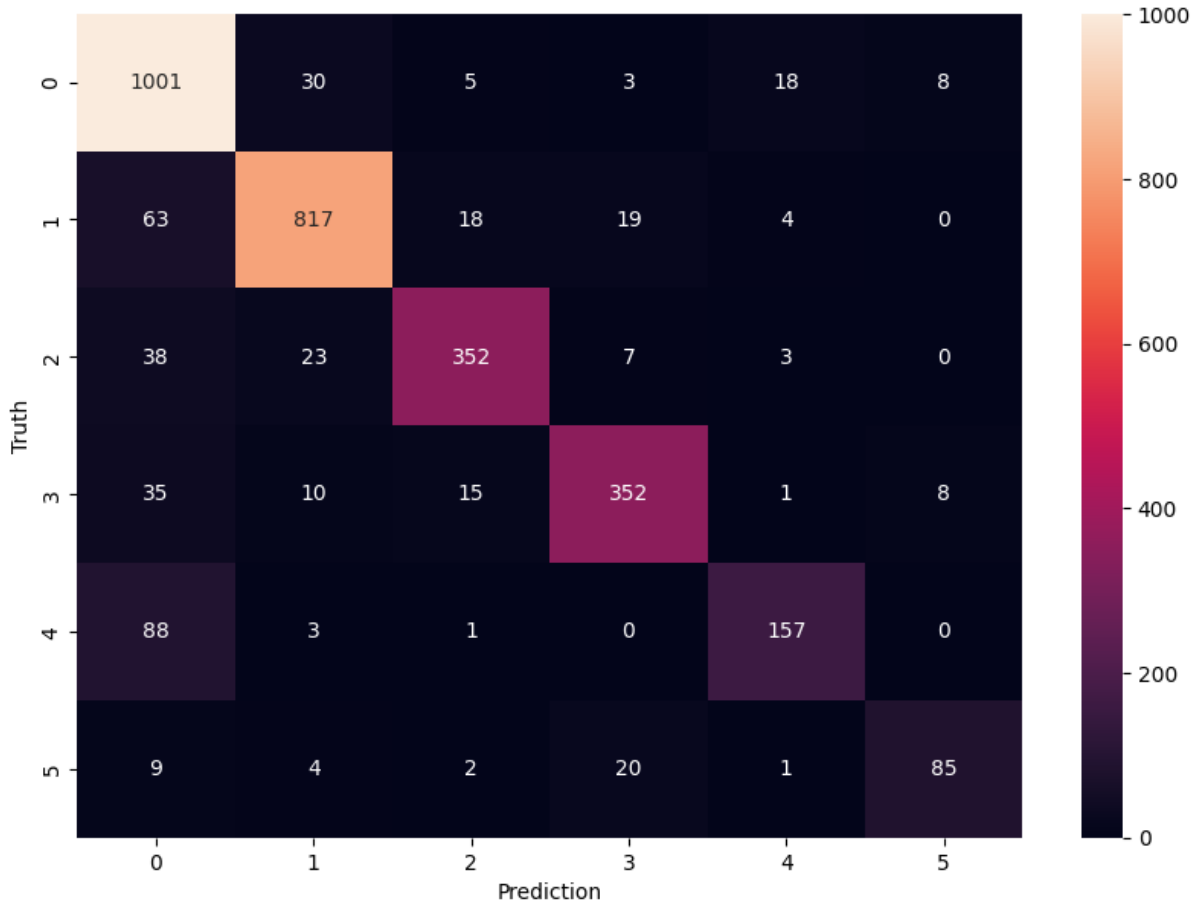
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

```
array([[1001,   30,    5,    3,   18,    8],
       [  63,  817,   18,   19,    4,    0],
       [  38,   23,  352,    7,    3,    0],
       [  35,   10,   15,  352,    1,    8],
       [  88,    3,    1,    0,  157,    0],
       [   9,    4,    2,   20,    1,   85]])
```

With the following code we are getting graph representation of the confusion matrix that was shown before.

```
from matplotlib import pyplot as plt
import seaborn as sn
plt.figure(figsize = (10,7))
sn.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Prediction')
plt.ylabel('Truth')
```

And the result is as follows.



IV. Work with Audio

The whole point of the project lies in this important part. So the concept is as follows: when the programm runs it asks you for how long you want to speak first. The future fixes will wait until the silence arrives, but in this beta version the program runs this way. When the string “Recording..” is printed the person must speak and the libraries that work with the microphone will catch the input and the audio will be saved to the temporary file inside of the project. Then the OpenAI whisper library is used to get this .mp3 file and transcribe it to the text.

But before that there are other steps. The following code trains the model *‘clf’*. The function ‘predict_emotion’ takes the model and user input and predicts the emotion there. The user input is gotten from the function get_user_input which calls for other functions which listen to audio, transcribe and return the text to the predict_emotion The whole output is printed.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline


X = df['processed_text']
y = df['label_num']

clf = Pipeline([
    ('vectorizer_tfidf', TfidfVectorizer()),
    ('Random Forest', RandomForestClassifier())
])

clf.fit(X, y)

def predict_emotion(sentence):
    processed_sentence = preprocess(sentence)

    emotion_label = clf.predict([processed_sentence])[0]
    emotion_mapping = {0: 'joy', 1: 'sadness', 2: 'anger', 3: 'fear', 4: 'love', 5: 'surprise'}
    predicted_emotion = emotion_mapping[emotion_label]

    return predicted_emotion

def get_user_input():
    duration = int(input("put the length of your speech in seconds (e.g. 5): "))
    audio_data = record_audio(duration)
    save_audio_mp3(audio_data)
    text = transcribe()
    return str(text['text'])


user_input = get_user_input()
predicted_emotion = predict_emotion(user_input)
print(user_input)
print(f"The predicted emotion for the sentence is: {predicted_emotion}")
```

The audio functions look the following way:

```python
import whisper
import sounddevice as sd
import numpy as np
from pydub import AudioSegment

def record_audio(duration, samplerate=44100):
    print("Recording...")
    audio_data = sd.rec(int(samplerate * duration), samplerate=samplerate, channels=1, dtype='int16')
    sd.wait()
    print("Recording complete.")
    return audio_data.flatten()

def save_audio_mp3(data, filename='collection/output.mp3', samplerate=44100):
    print(f"Saving audio to {filename}...")
    # Normalize the audio data to the range [-1, 1]
    normalized_data = data / np.max(np.abs(data), axis=0)
    # Convert to 16-bit PCM format
    pcm_data = (normalized_data * 32767).astype(np.int16)
    # Create an AudioSegment from the PCM data
    audio_segment = AudioSegment(pcm_data.tobytes(), frame_rate=samplerate, sample_width=2, channels=1)
    # Export as MP3
    audio_segment.export(filename, format="mp3")
    print("Audio saved.")

def transcribe():
    model = whisper.load_model("base")
    result = model.transcribe("collection/output.mp3")
    return result
```

The record_audio records the microphone and returns the audio file. The same audio is saved in format of .mp3 and later transcribed using the whisper. Which is used with the ‘base’ model. It is one of the smallest but very powerful. It is also multilingual but for the purposes of the program finishing successfully we are talking in English as the whole dataset is in that language. As the idea for future fixes is the consideration for the translator implemented. The whole text can be translated from Russian to English with ChatGPT as an example.

V. Result

The last function as I said gives the result as emotion that is recognized in the speech you do. It was fun.

Here is the example of it to work:

```
Recording...
Recording complete.
Saving audio to collection/output.mp3...
Audio saved.
/Users/sanekng/Documents/SpeakUp/venv/lib/python3.11/site-packages/whisper/transcribe.py:115:
  warnings.warn("FP16 is not supported on CPU; using FP32 instead")
 I'm happy to know that I finished my project. It was fun. I love it.
The predicted emotion for the sentence is: joy
```

VI. Literature

dataset:
https://www.kaggle.com/datasets/abdallahwagih/emotion-dataset
explanation:
https://www.geeksforgeeks.org/understanding-tf-idf-term-frequency-inverse-document-frequency/
https://www.geeksforgeeks.org/random-forest-classifier-using-scikit-learn/
https://python-sounddevice.readthedocs.io/en/0.4.6/
https://www.youtube.com/watch?v=Pk29TV4VLEg&t=147s
https://www.w3schools.com/python/pandas/pandas_csv.asp
https://www.datacamp.com/tutorial/pandas-read-csv
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
https://spacy.io/models/en
https://github.com/openai/whisper