UNIVERSITÀ DI PISA

Data Mining and Machine Learning

Year 2022/2023

DEPRESSION HUNTER

An API to detect depression in text

BASMA MOHAMED AHMED GABER ADAWY

INDEX

1. Introduction

Depression is a mood disorder that causes a persistent feeling of sadness and loss of interest. Also called major depressive disorder or clinical depression, it affects how you feel, think and behave and can lead to a variety of emotional and physical problems. Many of people end their life as a result of depression, so this may help them to discover their illness and save their life.

Depression hunter is an API that allows the user to check whether he is depressed or no. Only by typing some thoughts or feelings, the api will detect and tell the user if he needs to visit a therapist.

The purpose of the project is to apply machine learning techniques to solve a problem. We used different classification models on text datasets.

The implementation of the application can be found in the following git repository:

https://github.com/BasmaAdawy/DepressionHunter

2. Datasets

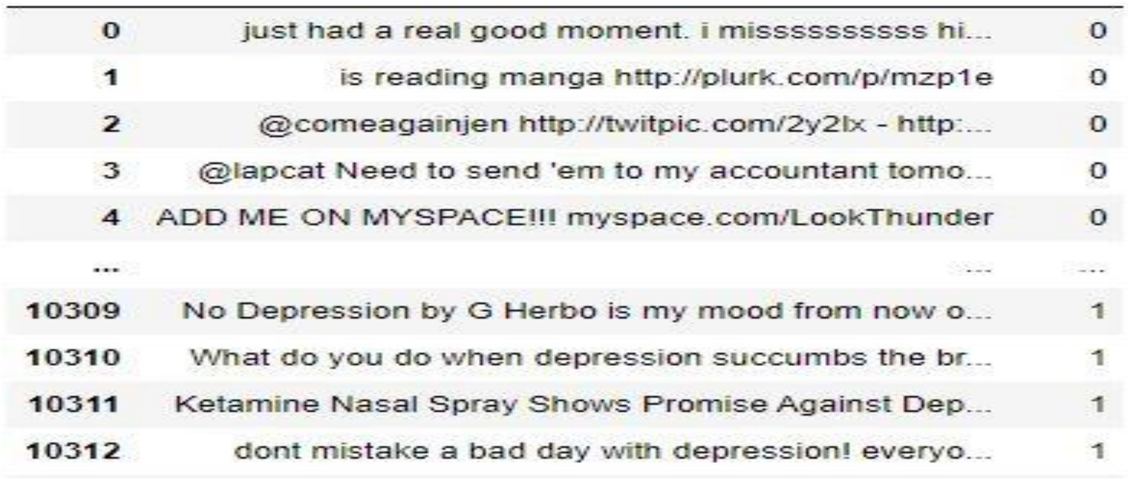
Datasets used to train the model are collected from two different sources, one is collected through web scrapping subreddits and the other one is from tweets.

The first dataset is 7731 rows, the second is 10314 rows.

Finally, we have 17933 rows.

we had two attributes for each instance: (text and label)

This is a sample from the data



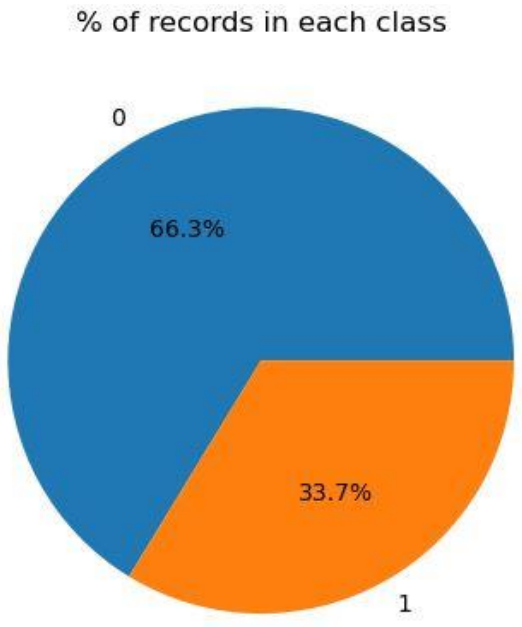
0	just had a real good moment. i missssssssss hi...	0
1	is reading manga http://plurk.com/p/mzp1e	0
2	@comeagainjen http://twitpic.com/2y2lx - http:...	0
3	@lapcat Need to send 'em to my accountant tomo...	0
4	ADD ME ON MYSPACE!!! myspace.com/LookThunder	0
...	...	...
10309	No Depression by G Herbo is my mood from now o...	1
10310	What do you do when depression succumbs the br...	1
10311	Ketamine Nasal Spray Shows Promise Against Dep...	1
10312	dont mistake a bad day with depression! everyo...	1

3. Data Pre-processing

We used a lot of different python functions to get useful information about our data. We found that it is imbalanced.

An imbalanced dataset in Natural Language Processing is a dataset whose number of data samples is not the same in the different classes. One class has more data samples than the other class.

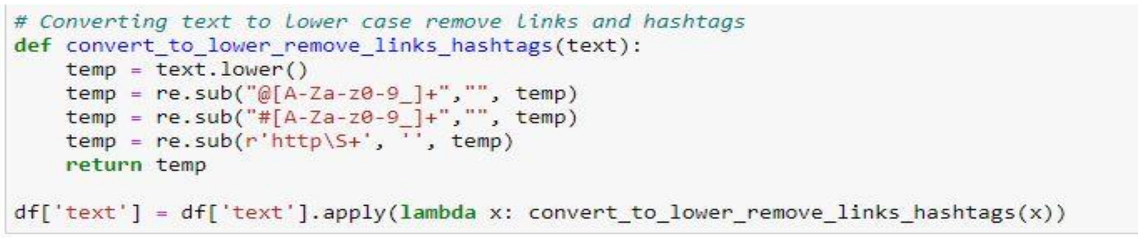
When we train a model with an imbalanced dataset, the model will be biased towards the majority class. The model may make wrong predictions and give inaccurate results.



Let's go step by step in cleaning our data:

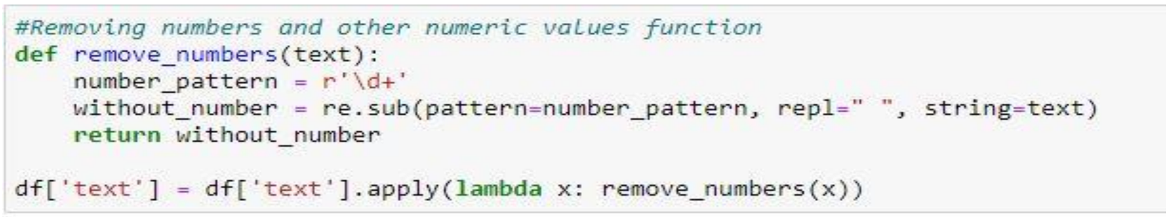
Many text cleaning steps will format the text. It includes removing unnecessary words, punctuation, stop words, white spaces, and unnecessary symbols from the text dataset.

- Converting all the text to lower case and removing links and hashtags: this ensures that we have a uniform dataset.



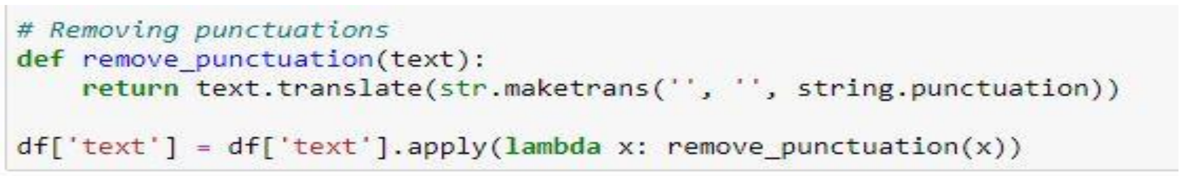
```python
# Converting text to lower case remove links and hashtags
def convert_to_lower_remove_links_hashtags(text):
    temp = text.lower()
    temp = re.sub("@[A-Za-z0-9_]+","", temp)
    temp = re.sub("#[A-Za-z0-9_]+","", temp)
    temp = re.sub(r'http\S+', '', temp)
    return temp

df['text'] = df['text'].apply(lambda x: convert_to_lower_remove_links_hashtags(x))
```

- Removing numbers and other numeric values: It ensures that only text that remains in the dataset adds value to the model.



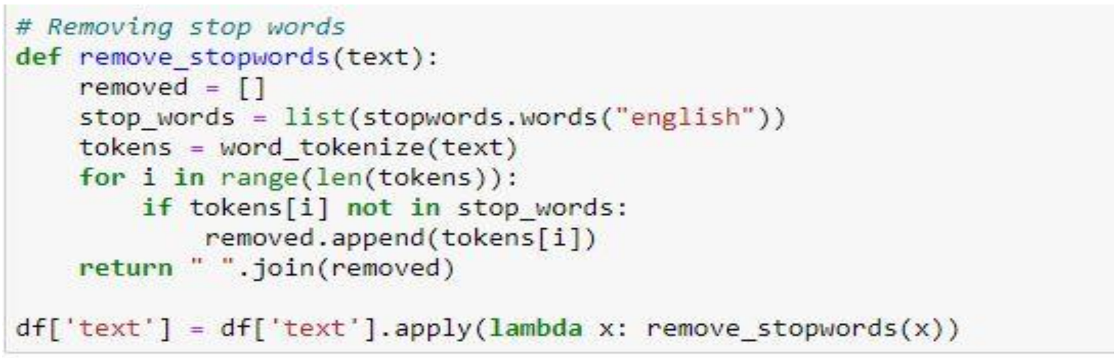
```python
#Removing numbers and other numeric values function
def remove_numbers(text):
    number_pattern = r'\d+'
    without_number = re.sub(pattern=number_pattern, repl=" ", string=text)
    return without_number

df['text'] = df['text'].apply(lambda x: remove_numbers(x))
```

- Removing punctuation involves removing full stops and other punctuation marks. These are the unnecessary symbols in the dataset.



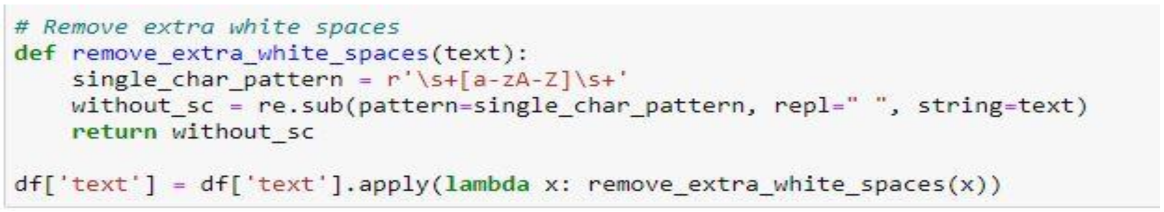
```python
# Removing punctuations
def remove_punctuation(text):
    return text.translate(str.maketrans('', '', string.punctuation))

df['text'] = df['text'].apply(lambda x: remove_punctuation(x))
```

- Removing stop words, stop words do not contribute to the meaning of a sentence since they are common in a language. Stop words for the English language are pronouns, conjunctions, and articles. Removing stop words enables the NLP model to focus on unique words in the SMS messages that will add value.



```python
# Removing stop words
def remove_stopwords(text):
    removed = []
    stop_words = list(stopwords.words("english"))
    tokens = word_tokenize(text)
    for i in range(len(tokens)):
        if tokens[i] not in stop_words:
            removed.append(tokens[i])
    return " ".join(removed)

df['text'] = df['text'].apply(lambda x: remove_stopwords(x))
```

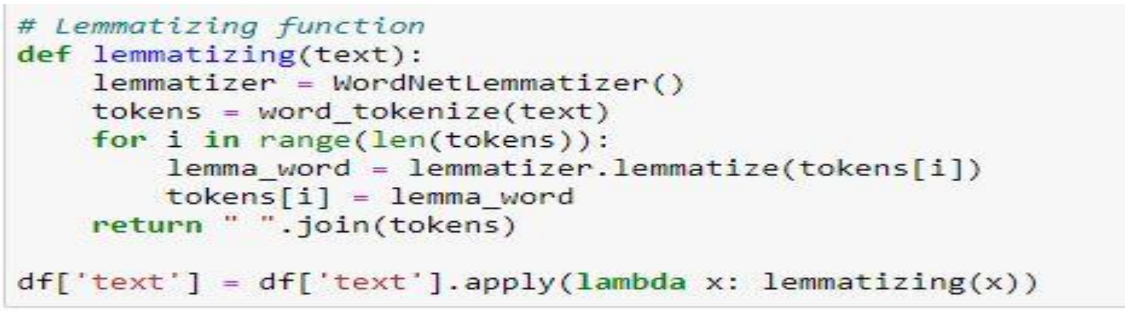
- Removing extra white spaces: White space occupies the dataset, but they do not carry information. Removing the extra white spaces ensures we only remain with the text that the model will use.



```python
# Remove extra white spaces
def remove_extra_white_spaces(text):
    single_char_pattern = r'\s+[a-zA-Z]\s+'
    without_sc = re.sub(pattern=single_char_pattern, repl=" ", string=text)
    return without_sc

df['text'] = df['text'].apply(lambda x: remove_extra_white_spaces(x))
```

- Tokenization: It is the splitting/breaking of the raw texts into smaller words or phrases known as tokens. We will implement the text cleaning steps using Natural Language Toolkit (NLTK).

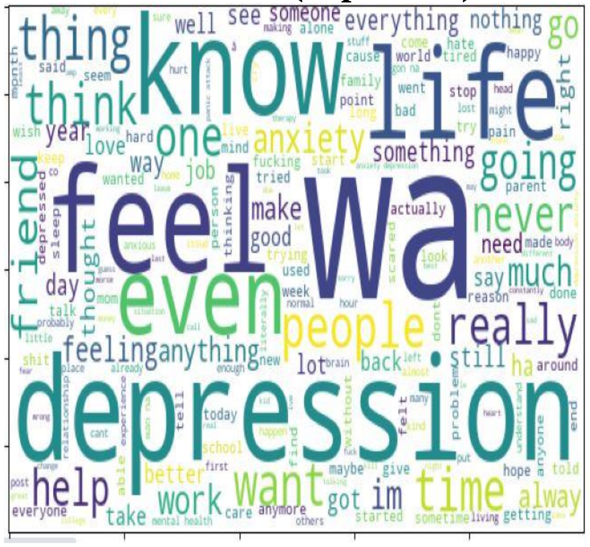
- Lemmatizing the texts: Stemming reduces inflected forms of a text/word into its lemma or dictionary form. For example, the words/texts “running”, “ran”, and “runs” are all reduced to the root form “run”.



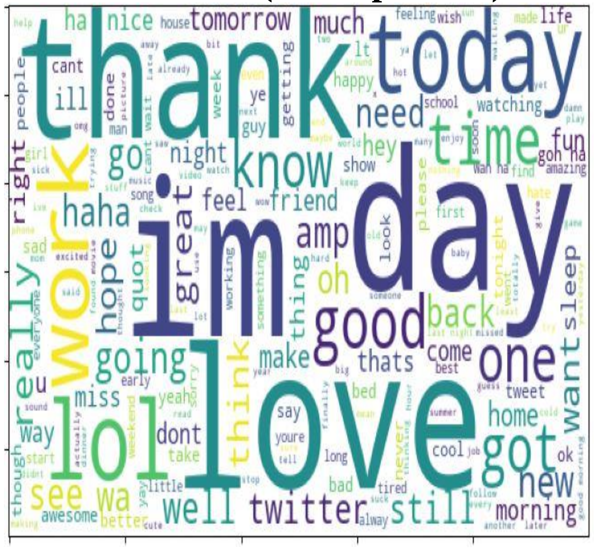
```
# Lemmatizing function
def lemmatizing(text):
    lemmatizer = WordNetLemmatizer()
    tokens = word_tokenize(text)
    for i in range(len(tokens)):
        lemma_word = lemmatizer.lemmatize(tokens[i])
        tokens[i] = lemma_word
    return " ".join(tokens)

df['text'] = df['text'].apply(lambda x: lemmatizing(x))
```

After cleaning the data we made some visualization techniques using **wordcloud** library

Label = 1 (depressed)



Label = 0 (not depressed)



4. Classification

▪ Dataset Splitting

We decided to test different classification algorithms against our dataset, in order to see which one was the best.

we had to split it into a training and a test set: our approach was to test each classifier both splitting the dataset using a 10-folds cross validation and a percentage split (70% of the initial dataset has been used to generate the training set, while the other 30% has been used as test set).

▪ Data Vectorization

It converts the raw text into a format the NLP model can understand and use. Vectorization will create a numerical representation of the text strings called a sparse matrix or word vectors. The model works with numbers and not raw text.

CountVectorizer simply counts the number of times a word appears in a document (using a bag-of-words approach).

TF-IDF Vectorizer takes into account not only how many times a word appears in a document but also how important that word is to the whole corpus.

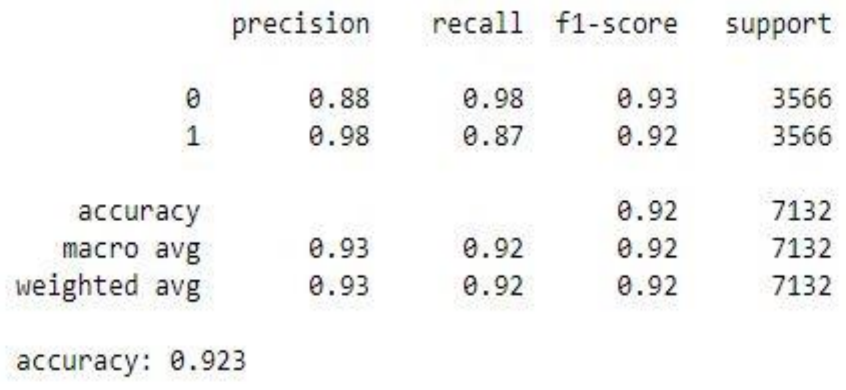
For more information: see the notebook

- **Data Balancing**

We solved this problem by using SMOTE (synthetic minority oversampling technique) is one of the most commonly used oversampling methods to solve the imbalance problem. It aims to balance class distribution by randomly increasing minority class examples by replicating them. SMOTE synthesises new minority instances between existing minority instances.

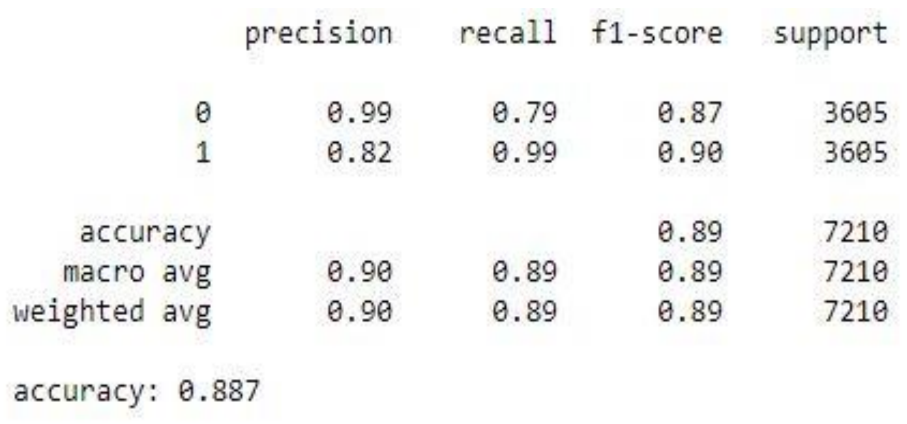
- **Model Building and Comparison between models**

SVC, or Support Vector Classifier, is a supervised machine learning algorithm typically used for classification tasks. SVC works by mapping data points to a high-dimensional space and then finding the optimal hyperplane that divides the data into two classes



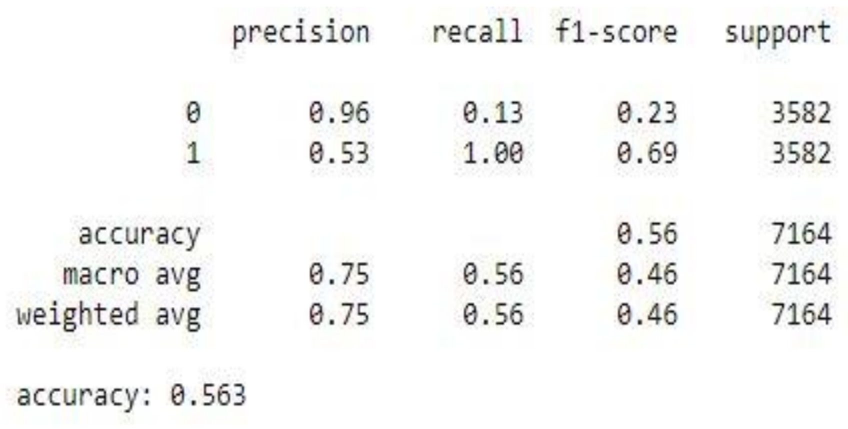
```
              precision    recall  f1-score   support

           0       0.88      0.98      0.93      3566
           1       0.98      0.87      0.92      3566

    accuracy                           0.92      7132
   macro avg       0.93      0.92      0.92      7132
weighted avg       0.93      0.92      0.92      7132

accuracy: 0.923
```

The multinomial naïve Bayes is widely used for assigning documents to classes based on the statistical analysis of their contents. It provides an alternative to the "heavy" AI-based semantic analysis and drastically simplifies textual data classification.



```
              precision    recall  f1-score   support

           0       0.99      0.79      0.87      3605
           1       0.82      0.99      0.90      3605

    accuracy                           0.89      7210
   macro avg       0.90      0.89      0.89      7210
weighted avg       0.90      0.89      0.89      7210

accuracy: 0.887
```

KNN (k-nearest neighbors algorithm) is a supervised classification algorithm that classifies new data points based on the nearest data points. On the other hand, K-means clustering is an unsupervised clustering algorithm that groups data into a K number of clusters



```
              precision    recall  f1-score   support

           0       0.96      0.13      0.23      3582
           1       0.53      1.00      0.69      3582

    accuracy                           0.56      7164
   macro avg       0.75      0.56      0.46      7164
weighted avg       0.75      0.56      0.46      7164

accuracy: 0.563
```

5. The Depression Hunter API

This is the final step to build an Api trained on machine learning classification algorithms to predict from the text if the user suffering from depression or not.

We built this API using FastAPI which is a modern, fast (high-performance), web framework for building APIs with Python 3.7+ based on standard Python type hints.

And Spyder which is an open-source cross-platform integrated development environment for scientific programming in the Python language.

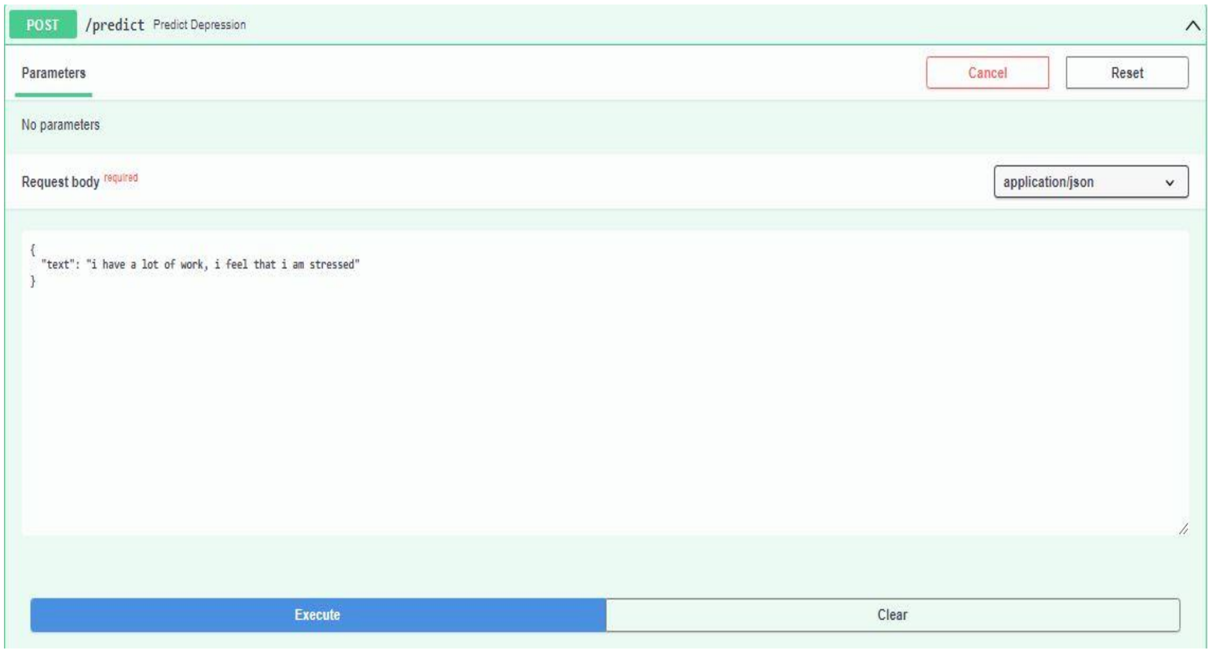
uvicorn is needed for the server that loads and serves your application.

How did we build the API?

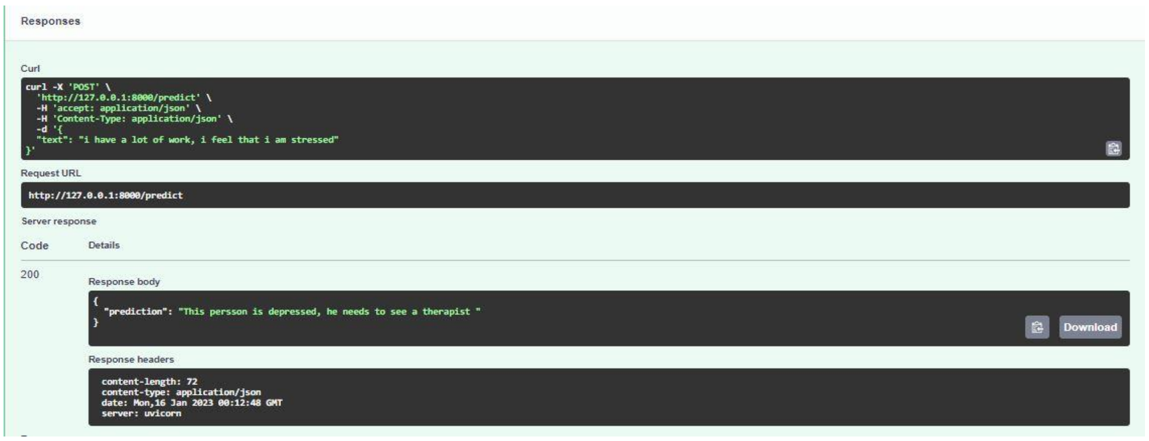
- Create the app object
- Index route, opens automatically on http://127.0.0.1:8000
- Expose the prediction functionality, make a prediction from the passed JSON data and return the predicted text
- Run the API with uvicorn

How it works?

The user enters a text in the post section and press on Execute button and waits for the result



```
{
  "text": "i have a lot of work, i feel that i am stressed"
}
```

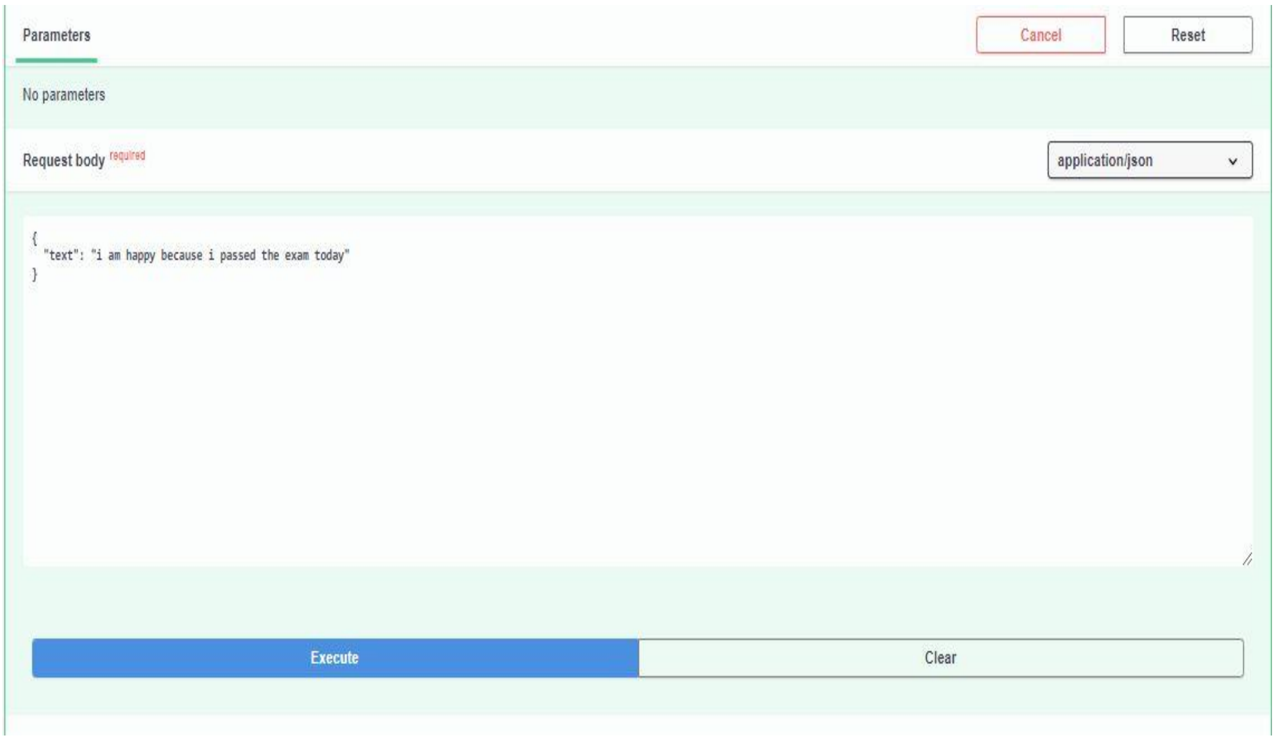


```
curl -X 'POST' \
  'http://127.0.0.1:8000/predict' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "text": "i have a lot of work, i feel that i am stressed"
}'
```

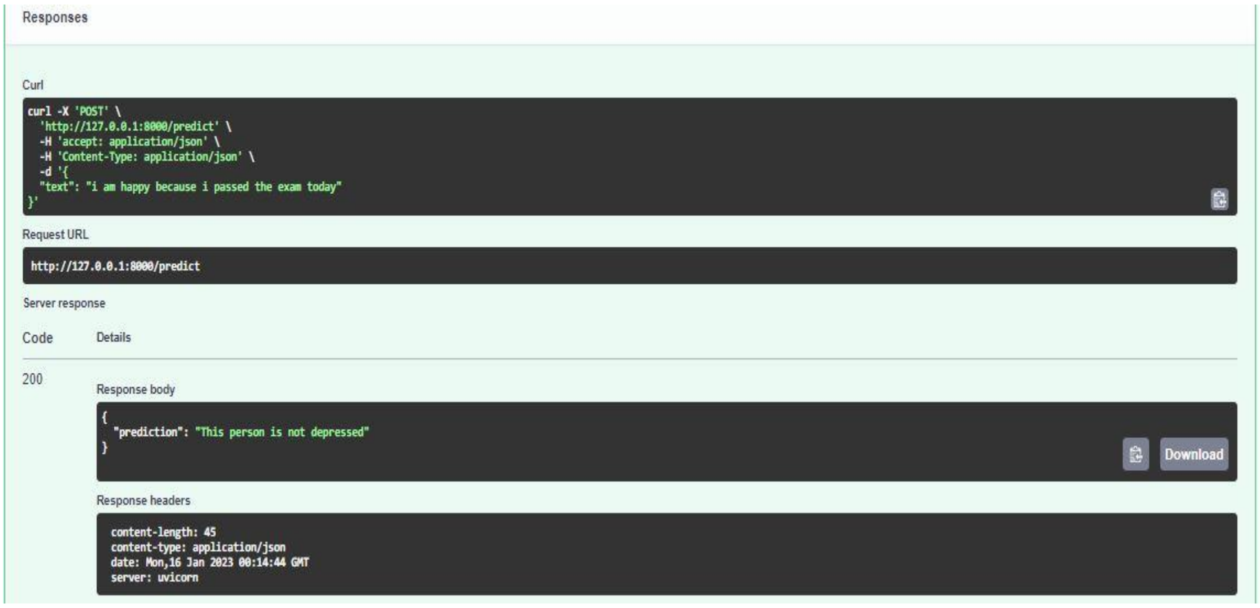
```
http://127.0.0.1:8000/predict
```

```
{
  "prediction": "This persson is depressed, he needs to see a therapist "
}
```

```
content-length: 72
content-type: application/json
date: Mon,16 Jan 2023 00:12:48 GMT
server: uvicorn
```



```
{
  "text": "i am happy because i passed the exam today"
}
```



```
curl -X 'POST' \
  'http://127.0.0.1:8000/predict' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "text": "i am happy because i passed the exam today"
}'
```

```
http://127.0.0.1:8000/predict
```

200

```
{
  "prediction": "This person is not depressed"
}
```

```
 content-length: 45
 content-type: application/json
 date: Mon,16 Jan 2023 00:14:44 GMT
 server: uvicorn
```

References

1. https://scikit-learn.org/stable/
2. https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html
3. https://fastapi.tiangolo.com/