گزارش پروژه NLP

قاسم یوسفی – 99243101

برای پیاده سازی این پروژه دو مدل مورد بررسی قرار گرفته است

- **مدل XGBoost** (تقویت گرادیان شدید) یک الگوریتم تقویت گرادیان قدرتمند و کارآمد است که به طور گسترده برای کارهای طبقه بندی و رگرسیون استفاده می شود. برای طبقه‌بندی چند برچسبی، XGBoost را می‌توان با استفاده از استراتژی‌هایی مانند ارتباط باینری یا تبدیل مشکل تطبیق داد. در رویکرد ارتباط باینری، طبقه‌بندی‌کننده‌های جداگانه XGBoost برای هر برچسب آموزش داده می‌شوند و هر کدام را به عنوان یک مشکل طبقه‌بندی باینری مستقل در نظر می‌گیرند. روش دیگر، روش‌های تبدیل مسئله مانند زنجیره‌های طبقه‌بندی، وابستگی‌های برچسب را با آموزش طبقه‌بندی‌کننده‌ها به‌طور متوالی در نظر می‌گیرند، جایی که پیش‌بینی‌های مدل‌های قبلی به عنوان ویژگی‌های اضافی عمل می‌کنند. توانایی XGBoost در مدیریت مقادیر از دست رفته، ویژگی‌های منظم‌سازی و اجرای کارآمد آن، آن را به یک کاندیدای قوی برای وظایف طبقه‌بندی چند برچسبی، به‌ویژه زمانی که با داده‌های جدولی ساختاریافته سروکار دارید، تبدیل می‌کند. تنظیم فراپارامتر مناسب برای بهینه سازی عملکرد آن در سناریوهای چند برچسبی ضروری است.

- **شبکه های عصبی (NN)** با لایه های مختلف، مانند LSTM (حافظه کوتاه مدت طولانی)، برای کارهای طبقه بندی چند برچسبی، به ویژه در داده های متوالی مانند متن و سری های زمانی بسیار موثر هستند. یک معماری NN برای این منظور معمولاً شامل یک لایه جاسازی برای نمایش کلمات به عنوان بردارهای متراکم و به دنبال آن لایه‌های LSTM برای ثبت وابستگی‌های دوربرد و الگوهای متوالی در ورودی است. LSTM های دو طرفه می توانند با پردازش توالی ها در هر دو جهت جلو و عقب، عملکرد را بیشتر افزایش دهند. لایه‌های Dropout با صفر کردن تصادفی واحدها در طول تمرین، به جلوگیری از بیش‌برازش کمک می‌کنند. لایه‌های کاملاً متصل (متراکم) ویژگی‌های آموخته‌شده را به خروجی‌های مربوط به برچسب‌های متعدد تبدیل می‌کنند، با لایه نهایی از تابع فعال‌سازی سیگموئید برای پیش‌بینی احتمالات برای هر برچسب به طور مستقل استفاده می‌کند. ترکیب LSTM با لایه‌های اضافی مانند شبکه‌های عصبی کانولوشن (CNN) یا مکانیسم‌های توجه می‌تواند با ثبت الگوهای محلی و جهانی، عملکرد را بیشتر بهبود بخشد. تنظیم هایپرپارامتر مناسب و توابع از دست دادن مانند آنتروپی متقاطع باینری برای پیش بینی های چند برچسبی دقیق ضروری هستند.

I. کتابخانه ها و ابزارهای مورد استفاده

- Pandas: دستکاری و پیش پردازش داده ها.
- NumPy: محاسبات عددی.
- Matplotlib: نمایش داده ها.
- TensorFlow/Keras: ساخت و آموزش مدل‌های یادگیری ماشین.
- NLTK: جعبه ابزار زبان طبیعی برای پردازش متن.
- scikit-learn: تقسیم و پیش پردازش داده ها.
- re: عملیات تطبیق عبارات منظم را ارائه می دهد و به شما امکان جستجو، تطبیق و دستکاری رشته ها را می دهد.

- pickle: یک ماژول پایتون برای سریال‌سازی و سریال‌زدایی اشیا، که به شما امکان می‌دهد اشیاء پایتون را ذخیره و بارگذاری کنید.
- jieba: یک کتابخانه تقسیم‌بندی متن چینی، که معمولاً برای توکن کردن متن چینی به کلمات استفاده می‌شود.
- fastapi: یک چارچوب وب مدرن و سریع (با کارایی بالا) برای ساخت API با Python 3.7+.
- pydantic: یک کتابخانه مدیریت تنظیمات و اعتبارسنجی داده برای پایتون، که معمولاً با FastAPI برای اعتبارسنجی ورودی استفاده می‌شود.
- nest_asyncio: به شما امکان می‌دهد حلقه‌های رویداد ناهمزمان را تودرتو کنید، و آنها را قادر می‌سازد در حلقه‌های در حال اجرا اجرا شوند (مفید برای نوت‌بوک‌های Jupyter).
- uvicorn: یک سرور ASGI سریع برای پایتون که معمولاً برای اجرای برنامه های FastAPI استفاده می شود.
- starlette.requests: بخشی از چارچوب Starlette، کلاس Request را برای رسیدگی به درخواست های HTTP در برنامه های FastAPI ارائه می دهد.
- pyngrok: یک پوشش پایتون برای ngrok، که تونل های امنی را برای میزبان محلی فراهم می کند، که اغلب برای نمایش سرورهای محلی در وب استفاده می شود.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from keras.utils import pad_sequences
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
from keras.models import Sequential, Model
from keras.layers import Dense,Dropout,Input,Embedding,Flatten,TextVectorization,Conv1D,GlobalMaxPooling1D,
from keras.initializers import Constant
from keras.layers import Dense,LSTM,Bidirectional,Attention,Concatenate,GRU,BatchNormalization
import nltk
from nltk.corpus import stopwords
import re
nltk.download('stopwords')
import pickle
import pandas as pd
import jieba
nltk.download('punkt')
nltk.download('wordnet')

from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()
```

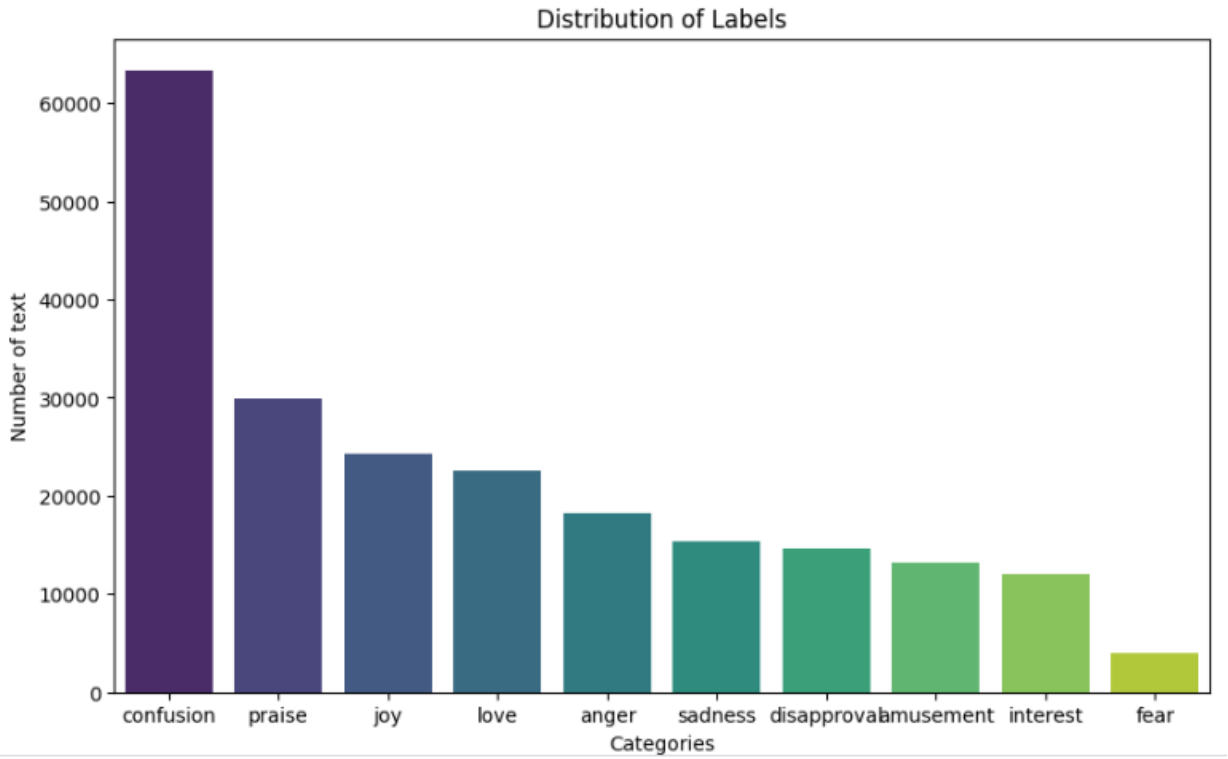
```
from fastapi import FastAPI
from pydantic import BaseModel
import nest_asyncio
import uvicorn
from fastapi.responses import HTMLResponse
from fastapi.templating import Jinja2Templates
from starlette.requests import Request

from pyngrok import ngrok
```

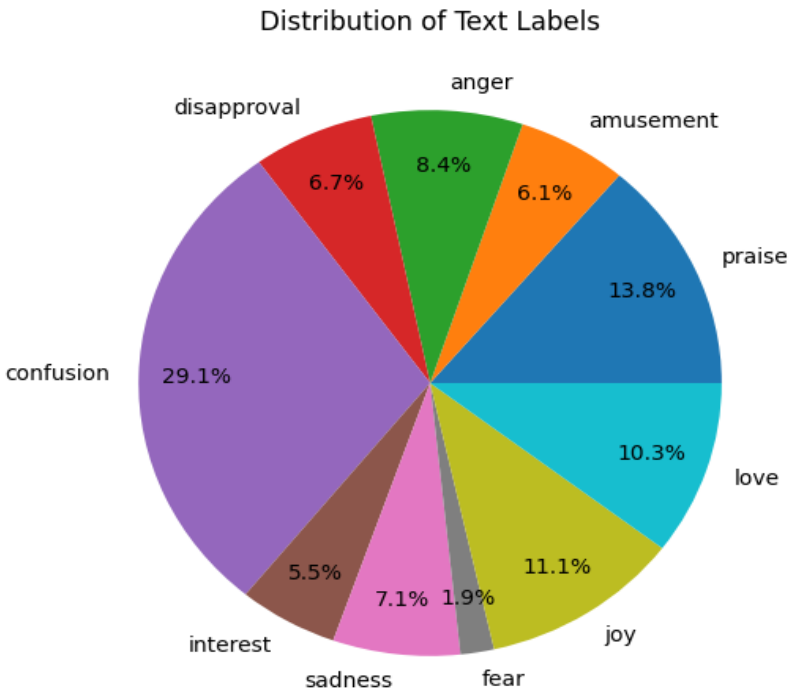
II. بررسی دیتاست

- نسبت و تناسب هر کلاس را بدست میاوریم تا بتوانیم دیدی بهتری از داده داشته باشیم

```
import matplotlib.pyplot as plt
import seaborn as sns
list_classes = ['praise', 'amusement', 'anger', 'disapproval', 'confusion', 'interest', 'sadness',

for column in list_classes:
    print(data_train[column].value_counts())
    print('-' * 50)

y_train = data_train[list_classes].values

df = pd.read_csv(train_path)
# Count total occurences of each label
label_counts = df[list_classes].sum().sort_values(ascending=False)

# Plot the label distribution
plt.figure(figsize=(10,6))
sns.barplot(x=label_counts.index, y=label_counts.values, palette="viridis")
plt.title("Distribution of Labels")
plt.xlabel("Categories")
plt.ylabel("Number of texts")
plt.show()
```

با استفاده از Seaborn برای تجسم توزیع برچسب های متنی، که در آن محور x نشان دهنده دسته ها و محور y تعداد متون در هر برچسب را نشان می دهد.





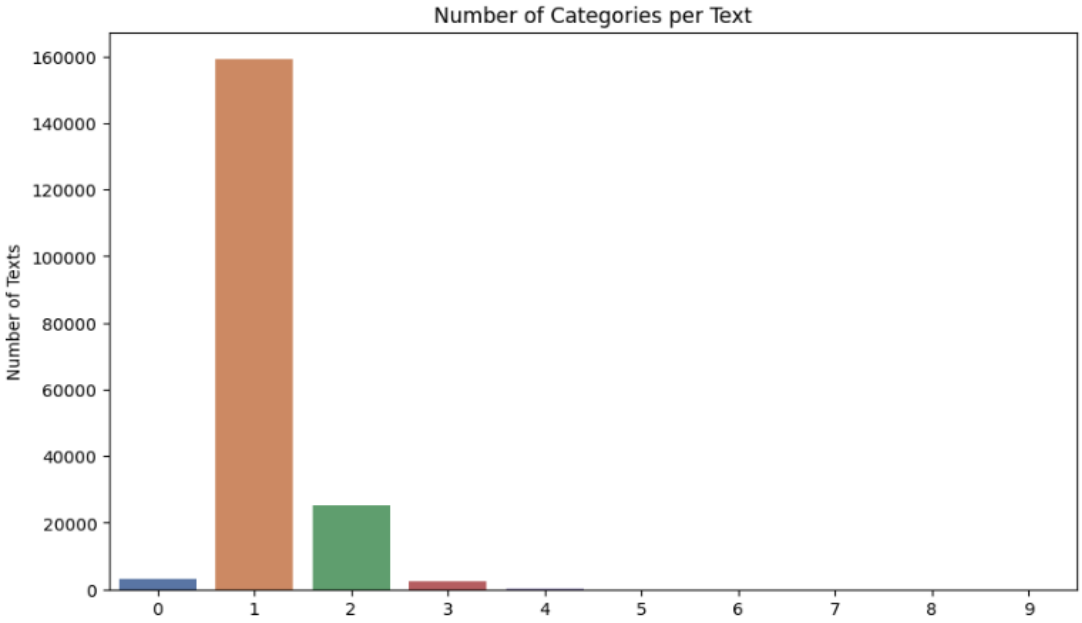
- بررسی مقادیر null

[6]:
```
# Check for missing values
print("\nMissing Values:")
print(df.isnull().sum())
```

```
Missing Values:
Unnamed: 0     0
text           0
praise         0
amusement      0
anger          0
disapproval    0
confusion      0
interest       0
sadness        0
fear           0
joy            0
love           0
dtype: int64
```

- توزیع label های هر متن

```
# Check the number of multi-label Text
df['multi_lable_count'] = df[Label_columns].sum(axis=1)
multi_label_distribution = df['multi_lable_count'].value_counts().sort_index()

# Plot multi-label distribution
plt.figure(figsize=(10, 6))
sns.barplot(x=multi_label_distribution.index, y=multi_label_distribution.values,
plt.title("Number of Categories per Text")
plt.xlabel("Number of Labels")
plt.ylabel("Number of Texts")
plt.show()
```

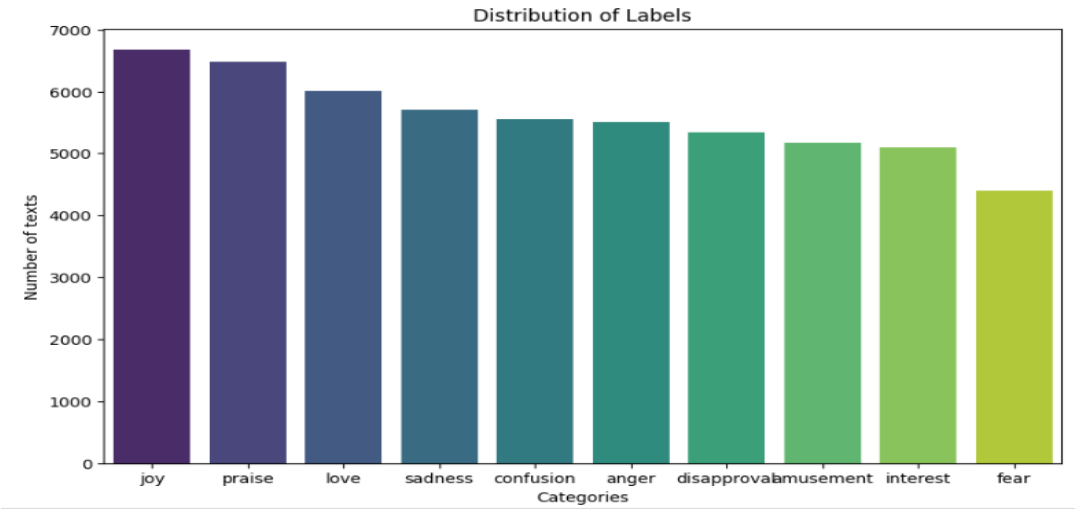


در پیش پردازش از این اطلاعات جهت پردازش dataset ها استفاده می شود

III. پیش پردازش و پاکسازی

از اطلاعات در دست داشته این چنین به بنظر می رسد که داده ها نا متعادل هستند
حجم این دیتا نیز زیاد بوده پس:

- انجام متعادل سازی با undersampling



هر چند این مورد زیاد کار ساز نبوده و بعدا از آن صرف نظر شده است

- پاک سازی متن ها

متن را برای یکنواختی به حروف کوچک تبدیل می نماییم.
هر کاراکتر غیر الفبایی (مانند علائم نگارشی) را با استفاده از یک عبارت منظم حذف نموده.
متن را به کلمات مجزا تبدیل می کند.
کلمات توقف (کلمات رایج مانند ""is ،the""، و غیره) را حذف می کند تا روی عبارات مهم تمرکز کند.
کلمات را به صورت لماتیزه می کند (آنها را به شکل اصلی خود تقلیل می دهد، به عنوان مثال، "running" به "run"). سپس متن پاک شده به صورت رشته ای از کلمات پردازش شده برگردانده می شود.

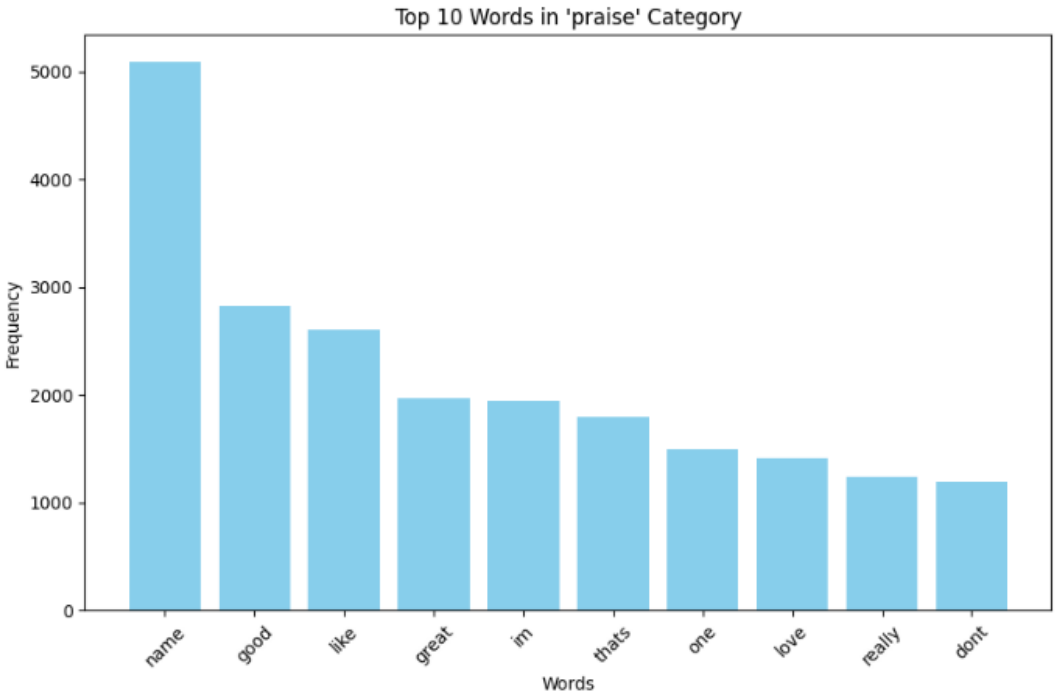
```
# Text Preprocessing
def clean(text):
    text = text.lower()
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text)
    words = word_tokenize(text)
    words = [lemmatizer.lemmatize(word) for word in words if word not in stop_words]
    return ' '.join(words)
```

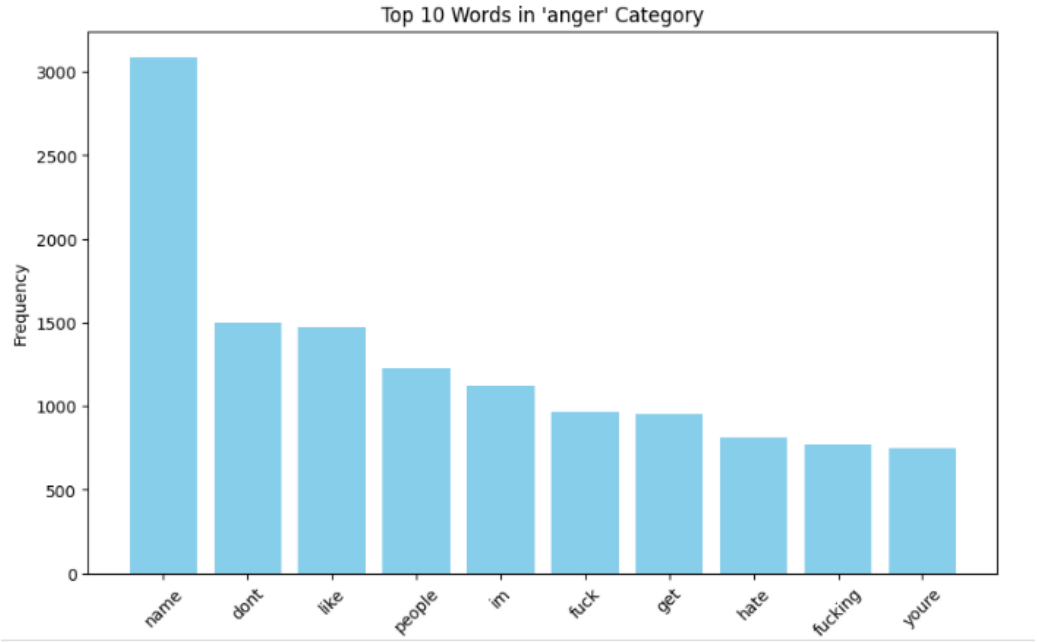
```
Is there some scripture you could quote me? I'd like to read up on it just to be sure for myself
scripture could quote id like read sure
```

IV. بررسی متن و دیتا بعد پاکسازی

برای اطمینان پیش تر از کارکرد دیتاست باید نقاط ضعف داده را حذف نمود

- تعداد کلمات منحصر به فرد را در متن در دسته خاص
- نمایش 10 کلمه برتر در هر کلاس





بطور نمونه می توان دید که کلمات مانند name و im در تمام کلاس ها بطور مشترک وجود دارد

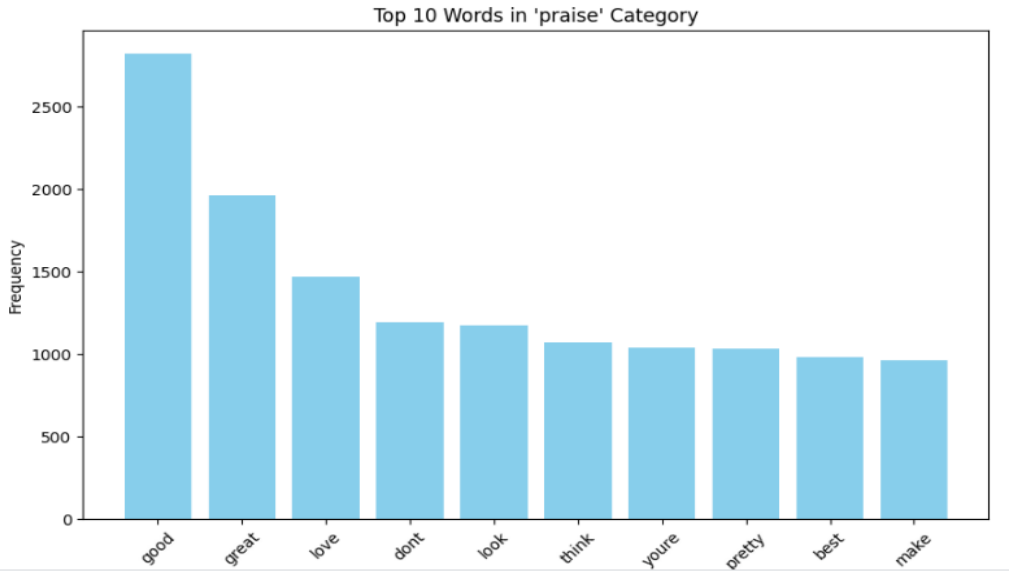
- بدست آوردن کلمات مشترک بین تمام کلاس ها و کلمات که موثر نخواهد بود

```python
def common_words(*word_lists):
    if not word_lists:
        return set()

    common_set = set(word_lists[0])
    for word_list in word_lists[1:]:
        common_set.intersection_update(word_list)

    return common_set

result = common_words(*list_words)
print("Words in all classes:", result)
```

```
Words in all classes: {'name', 'like', 'im'}
```

- حذف این کلمات (پاکسازی مرحله دوم)

```python
res = list(result)
res.append('people')
res.append('thats')
res.append('get')
res.append('one')
res.append('would')
res.append('really')
x_train_cleaned1 = x_train_cleaned.apply(lambda text: clean1(text, res))

words = words_freq(x_train_cleaned1)
```



مدل می تواند دید بهتری از هر کلاس داشته باشد.

V. توکن سازی و پدینگ

توکن‌سازی فرآیند تجزیه متن به واحدهای کوچک‌تر، معمولاً کلمات یا زیرکلمه‌ها است که به عنوان نشانه‌ها شناخته می‌شوند. در پردازش زبان طبیعی (NLP)، توکن‌سازی برای تبدیل متن خام به قالبی که توسط مدل‌های یادگیری ماشین قابل درک و پردازش باشد، بسیار مهم است. به عنوان مثال، جمله "I love NLP" ممکن است به نشانه های ["I"، "Love"، "NLP""] تبدیل شود. از سوی دیگر، padding فرآیند استانداردسازی طول توالی های نشانه گذاری شده است، به ویژه زمانی که طول جملات ورودی متفاوت است. از آنجایی که بسیاری از مدل‌های یادگیری ماشین (مانند مدل‌هایی که در NLP استفاده می‌شوند) نیاز به ورودی‌ها دارند که اندازه ثابتی داشته باشند، از padding برای ایجاد طول یکسانی همه دنباله‌ها استفاده می‌شود.

```
x_train_cleaned = x_train_cleaned1
# Tokenize the text data
tokenizer = Tokenizer()
tokenizer.fit_on_texts(x_train_cleaned)

X_train_sequences = tokenizer.texts_to_sequences(x_train_cleaned)

word_index = tokenizer.word_index
total_words = len(word_index)
print(total_words) #same as length of vocabulary

emb_len=len(tokenizer.index_word)+1
#getting the length of the maximum sequence in the dataset
max_length = max([len(w) for w in X_train_sequences])
print(max_length)

# Padding the sequences
X_train_padded = pad_sequences(X_train_sequences, maxlen=100, padding='post', truncating='post')
```

ابتدا، یک Tokenizer را مقداردهی اولیه می کند و آن را به متن آموزشی تمیز می کند تا یک فهرست کلمه ایجاد کند، و هر کلمه منحصربه فرد را به یک عدد صحیح منحصر به فرد نگاشت می‌کند. سپس، متن پاک شده را با استفاده از این فهرست کلمه به دنباله ای از اعداد صحیح تبدیل می کند. این تعداد کل کلمات منحصر به فرد در واژگان و حداکثر طول توالی در تمام داده های ورودی را محاسبه می کند. در نهایت، توالی‌های توکن‌شده را به طول ثابت ۱۰۰ ژتون اضافه می‌کند، و اطمینان می‌دهد که توالی‌های کوتاه‌تر با صفر در انتها (پس از لایه برداری) و دنباله‌های طولانی‌تر در انتها (پس از برش دادن) کوتاه می‌شوند، و داده‌ها را برای سازگاری آماده می‌کند.

VI. مدل (آموزش و بررسی)

همانطوری که در ابتدا اشاره شده دو مدل را بررسی می نماییم

```
# Train the XGBoost model
    model = xgb.XGBClassifier(
        objective='binary:logistic',
        eval_metric='logloss',
        use_label_encoder=False,
        max_depth=6,
        learning_rate=0.1,
        n_estimators=100,
        random_state=42
    )
```

این مدل برای طبقه‌بندی باینری با هدف تنظیم شده روی «دودویی: لجستیک» پیکربندی شده است، به این معنی که احتمالات را برای دو کلاس با استفاده از رگرسیون لجستیک پیش‌بینی می‌کند. معیار ارزیابی روی "logloss" تنظیم شده است که عملکرد طبقه‌بندی را بر حسب آنتروپی متقابل اندازه‌گیری می‌کند. پارامتر max_depth بر روی 6 تنظیم می شود و عمق درخت های تصمیم مورد استفاده در مدل را کنترل می کند و نرخ_آموزش روی 0.1 تنظیم می شود که اندازه گام را در هر تکرار تقویتی تعیین می کند. مدل از 100 دور تقویت (n_estimators=100) استفاده خواهد کرد. random_state=42 تکرارپذیری نتایج را تضمین می کند. علاوه بر این، use_label_encoder=False از هشدار مربوط به رمزگذاری برچسب جلوگیری می کند. این پیکربندی برای کارهای طبقه بندی باینری با مدل کنترل شده مناسب است.

در ابتدا هر برای تمام کلاس ها آموزش داده شده و تست می شود

```
Class distribution after undersampling: Counter({0: 25350, 1: 25350})
Classification Report for praise:
              precision    recall  f1-score   support

           0       0.90      0.87      0.88     33795
           1       0.41      0.49      0.45      6338

    accuracy                           0.81     40133
   macro avg       0.65      0.68      0.66     40133
weighted avg       0.82      0.81      0.81     40133


Training model for amusement...

Class distribution after undersampling: Counter({0: 11203, 1: 11203})
Classification Report for amusement:
              precision    recall  f1-score   support

           0       0.97      0.93      0.95     37332
           1       0.39      0.64      0.49      2801

    accuracy                           0.91     40133
   macro avg       0.68      0.78      0.72     40133
weighted avg       0.93      0.91      0.92     40133


Training model for anger...

Class distribution after undersampling: Counter({0: 15438, 1: 15438})
Classification Report for anger:
              precision    recall  f1-score   support

           0       0.94      0.86      0.90     36273
           1       0.28      0.49      0.36      3860

    accuracy                           0.83     40133
   macro avg       0.61      0.68      0.63     40133
weighted avg       0.88      0.83      0.85     40133


Training model for disapproval...

Class distribution after undersampling: Counter({0: 12348, 1: 12348})
Classification Report for disapproval:
              precision    recall  f1-score   support

           0       0.95      0.81      0.87     37046
           1       0.19      0.54      0.28      3087

    accuracy                           0.78     40133
   macro avg       0.57      0.67      0.58     40133
weighted avg       0.90      0.78      0.83     40133
```

```
Training model for confusion...

Class distribution after undersampling: Counter({0: 53582, 1: 53582})
Classification Report for confusion:
              precision    recall  f1-score   support

           0       0.85      0.45      0.59     26737
           1       0.43      0.84      0.57     13396

    accuracy                           0.58     40133
   macro avg       0.64      0.65      0.58     40133
weighted avg       0.71      0.58      0.59     40133


Training model for interest...

Class distribution after undersampling: Counter({0: 10115, 1: 10115})
Classification Report for interest:
              precision    recall  f1-score   support

           0       0.96      0.83      0.89     37604
           1       0.15      0.46      0.23      2529

    accuracy                           0.81     40133
   macro avg       0.56      0.65      0.56     40133
weighted avg       0.91      0.81      0.85     40133



  Training model for love...

  Class distribution after undersampling: Counter({0: 19002, 1: 19002})
  Classification Report for love:
                precision    recall  f1-score   support

             0       0.96      0.94      0.95     35383
             1       0.61      0.68      0.64      4750

      accuracy                           0.91     40133
     macro avg       0.78      0.81      0.79     40133
  weighted avg       0.91      0.91      0.91     40133


  Overall Classification Report:
                precision    recall  f1-score   support

             0       0.95      0.85      0.90    355347
             1       0.35      0.62      0.45     45983

      accuracy                           0.83    401330
     macro avg       0.65      0.74      0.67    401330
  weighted avg       0.88      0.83      0.84    401330
```

دقت نصبتا خوبی برای هر کلاس بشکل جداگانه می توان بدست آورد

ولی در نهایت برای multilabel عملکردی خوبی ارائه نمیدهد

این دقت در حدود 20% الی 30% در صد می باشد

```python
def multilabel_accuracy():
    r1 = []
    y1 = []
    c = 0
    for i in range(n):
        temp = []
        temp2 = []
        for label in Label_columns:
            temp.append(p[label][i])
            temp2.append(r[label][i])
        if(temp == temp2):
            c+=1
        y1.append(temp)
        r1.append(temp2)
    c = c / n
    return r1,y1,c

r1,y1,ress = multilabel_accuracy()
print(f"multilabel accuracy:{ress}")
```

```
multilabel accuracy:0.20687369816322665
```

مدل دوم (NN)

مدل توسعه یافته برای طبقه بندی احساسات با استفاده از Keras طراحی شده است و شامل اجزای زیر است:

- لایه ورودی: دنباله های متنی توکن شده را پردازش می کند.
- لایه جاسازی: کلمات را به بردارهای متراکم با اندازه ثابت تبدیل می کند. لایه Embedding() به ثبت روابط معنایی بین کلمات کمک می کند.
- لایه LSTM دوطرفه: یک لایه تکرارشونده (Bidirectional(LSTM())) که توالی های ورودی را از هر دو جهت رو به جلو و عقب پردازش می کند و جذب وابستگی های طولانی مدت در متن را افزایش می دهد.
- مکانیسم attention: لایه‌های توجه سفارشی (attention) برای تمرکز بر مرتبط‌ترین بخش‌های دنباله ورودی اضافه شدند.
- لایه‌های Dropout: با استفاده از Dropout() برای تنظیم تصادفی کسری از ورودی‌ها بر روی صفر در طول آموزش، پیاده‌سازی می‌شود که به جلوگیری از بیش‌برازش کمک می‌کند.

- لایه های متراکم: لایه های کاملاً متصل (Dense()) ویژگی ها را تغییر می دهند و خروجی طبقه بندی نهایی را ارائه می دهند.

- ExponentialDecay به تدریج نرخ یادگیری را با استفاده از واپاشی نمایی بر اساس مراحل و نرخ فروپاشی مشخص شده کاهش می دهد.

$$\text{learning\_rate} = \text{initial\_learning\_rate} \times (\text{decay\_rate})^{\left\lfloor \frac{\text{step}}{\text{decay\_steps}} \right\rfloor}$$

```
def create_model(lr, decay_step, decay_rate):

    # Define a decaying learning rate
    lr_schedule = ExponentialDecay(
        initial_learning_rate=lr,
        decay_steps=decay_step,
        decay_rate= decay_rate,
        staircase=True  # Apply decay in discrete intervals
    )
    # Input layer
    input_layer = Input(shape=(100,))  # Specify max_len as the maximum sequence length

    # Embedding layer
    embedding_layer = Embedding(emb_len, 128)(input_layer)

    # Bidirectional LSTM layer replaced with Attention layer
    lstm_layer = Bidirectional(LSTM(128, return_sequences=True, dropout=0.2, recurrent_dropout=0.3))(
    attention = Attention()([lstm_layer, lstm_layer])  # Attention layer

    # 1D Convolutional layer
    conv1d_layer = Conv1D(64, kernel_size=3, activation='relu')(attention)
    global_max_pooling_layer = GlobalMaxPooling1D()(conv1d_layer)

    # Dense layers
    dense_layer_1 = Dense(128, activation='relu')(global_max_pooling_layer)
    ######
    # dense_layer_1 = Dropout(0.3)(dense_layer_1)  # Prevent overfitting
    # dense_layer_2 = Dense(128, activation='relu')(dense_layer_1)
    output_layer = Dense(10, activation='sigmoid')(dense_layer_1)

    # Model creation
    model = Model(inputs=input_layer, outputs=output_layer)
    # custom_adam = Adam(learning_rate=0.0001)
    custom_adam = Adam(learning_rate=lr_schedule)

    model.compile(optimizer=custom_adam,loss='binary_crossentropy',metrics=['accuracy'])

    # model.compile(optimizer=custom_adam, loss=focal_loss(), metrics=['accuracy'])
    return model
```

Layer (type)	Output Shape	Param #	Connected to
input_layer_5 (InputLayer)	(None, 100)	0	-
embedding_5 (Embedding)	(None, 100, 128)	3,781,120	input_layer_5[0][0]
bidirectional_5 (Bidirectional)	(None, 100, 256)	263,168	embedding_5[0][0]
attention_5 (Attention)	(None, 100, 256)	0	bidirectional_5[0][0], bidirectional_5[0][0]
conv1d_5 (Conv1D)	(None, 98, 64)	49,216	attention_5[0][0]
global_max_pooling1d_5 (GlobalMaxPooling1D)	(None, 64)	0	conv1d_5[0][0]
dense_11 (Dense)	(None, 128)	8,320	global_max_pooling1d_…
dense_12 (Dense)	(None, 10)	1,290	dense_11[0][0]

آموزش

با کمک earlystopping و modelcheck point در زمان آموزش بهترین مدل ذخیره شده و بعد در زمان نخمین و حدس می توان آنرا بازیابی نمود

```
checkpoint_path = '/kaggle/working/best_model.keras'  # Change the extension to .keras
checkpoint_dir = os.path.dirname(checkpoint_path)

# Make sure the directory exists
if not os.path.exists(checkpoint_dir):
    os.makedirs(checkpoint_dir)

model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_path,
    monitor='val_accuracy',
    mode='max',
    save_best_only=True
)
early_stop = EarlyStopping(monitor='val_loss',patience=2,verbose=True)

#Trainig the model
history = model.fit(X_train_padded,y_train,batch_size=128,epochs=4,validation_split=0.1,callbacks=[mc
model.save('/kaggle/working/final_model.keras')
```

ارزیابی

برای آسانی کار با مدل می توان از مدل ذخیره شده استفاده نمود

```python
def pred_evaluate(data_test):
    model = load_model('/kaggle/working/final_model.keras')
    model.load_weights('/kaggle/working/best_model.keras')

    list_classes = ['praise', 'amusement', 'anger', 'disapproval', 'confusion', 'interest', 'sadnes
    x_test=data_test['text']
    label_test= data_test[list_classes]
    y_test = label_test[list_classes].values
    #Load best model
    x_test_cleaned = x_test.apply(clean)
    x_test_cleaned = x_test_cleaned.apply(lambda text: clean1(text, res))
    X_test_sequences = tokenizer.texts_to_sequences(x_test_cleaned)
    X_test_padded = pad_sequences(X_test_sequences, maxlen=100, padding='post', truncating='post')

    ## Evaluate Model

    loss, accuracy = model.evaluate(X_test_padded, y_test)
    print(f'Validation Loss: {loss}, Validation Accuracy: {accuracy}')
```

```python
    ## Evaluate Model

    loss, accuracy = model.evaluate(X_test_padded, y_test)
    print(f'Validation Loss: {loss}, Validation Accuracy: {accuracy}')

    # Predictions
    predictions = model.predict(X_test_padded)
    print(predictions)
    target_columns = list_classes
    predictions = pd.DataFrame(predictions)
    output = pd.DataFrame(data={'praise': predictions[0], 'amusement': predictions[1],
                                'anger': predictions[2], 'disapproval': predictions[3],  'confusion': predictions[4],
                                'interest': predictions[5],'sadness': predictions[6],'fear': predictions[7],
                                'joy': predictions[8],'love': predictions[9]})
    output.to_csv('result.csv', index=False)

    predicted = output.values
    return predicted

val_path = '../input/emotrainval/EmoVal.csv'
data_test = pd.read_csv(val_path)

predicted = pred_evaluate(data_test)
```

برای ارزیابی مدل نیاز به threshold می باشد تا label های واقعی مقایسه شوند

```python
# Find best threshold per class
for i in range(num_classes):
    if np.sum(real_labels[:, i]) == 0:
        # No positive samples, default threshold to 0.5
        best_thresholds[i] = 0.5
        binarized_preds[:, i] = (predicted[:, i] >= best_thresholds[i]).astype(int)
    else:
        precision, recall, thresholds = precision_recall_curve(real_labels[:, i], predicted[:, i])
        f1_scores = 2 * (precision * recall) / (precision + recall + 1e-10)
        best_index = np.nanargmax(f1_scores)

        if best_index < len(thresholds):
            best_thresholds[i] = thresholds[best_index]
        else:
            best_thresholds[i] = 0.5  # Fallback

        binarized_preds[:, i] = (predicted[:, i] >= best_thresholds[i]).astype(int)
print(binarized_preds)
# Print the best thresholds
print("Best thresholds for each class:", best_thresholds)
```

مدل با مقادیر مختلف هایپر پارامتر تست شده بعضی آنها بشکل زیر نماش داده شده است

Learning rate: 0.001 #epoch:4 inputsize:100 pathsize:128 decaystep:10000 …….

```
Epoch 1/4
1337/1337 ━━━━━━━━━━━━━━━━━━━━ 433s 321ms/step - accuracy: 0.3475 - loss: 0.3248 - val_accuracy:
0.4423 - val_loss: 0.2787
Epoch 2/4
1337/1337 ━━━━━━━━━━━━━━━━━━━━ 428s 320ms/step - accuracy: 0.4607 - loss: 0.2688 - val_accuracy:
0.4634 - val_loss: 0.2651
Epoch 3/4
1337/1337 ━━━━━━━━━━━━━━━━━━━━ 429s 321ms/step - accuracy: 0.4935 - loss: 0.2491 - val_accuracy:
0.4545 - val_loss: 0.2634
Epoch 4/4
 575/1337 ━━━━━━━━━━━━━━━━━━━━ 4:00 316ms/step - accuracy: 0.5177 - loss: 0.2352
```

Learning rate: 0.01 #epoch:4 inputsize:100 pathsize:128 decaystep:10000 …….

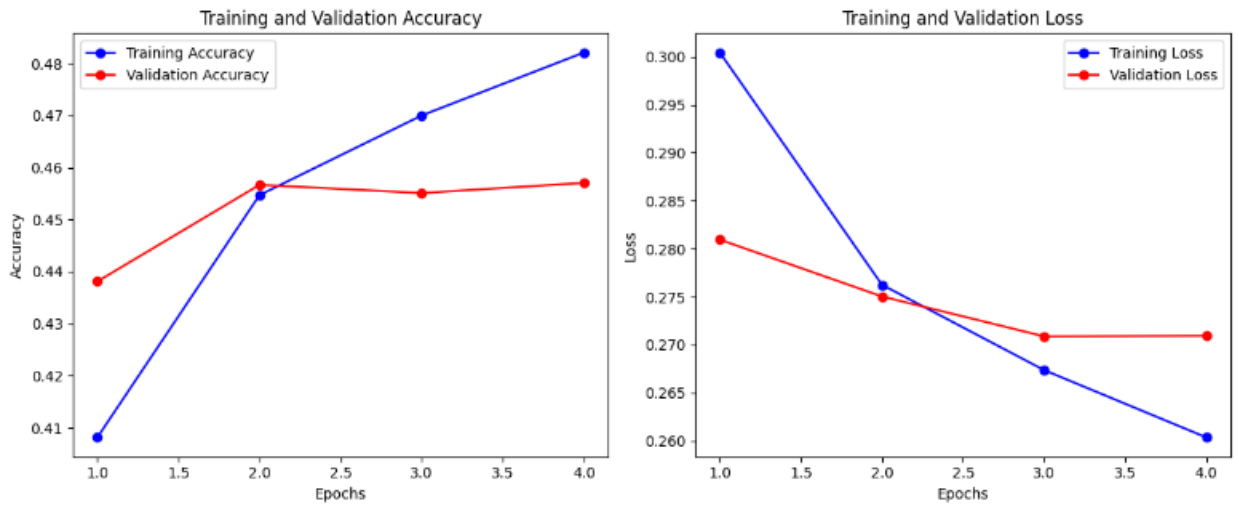
```
Epoch 1/4
1337/1337 ━━━━━━━━━━━━━━━━━━━━ 296s 215ms/step - accuracy: 0.3486 - loss: 0.3226 - val_accuracy: 0.4400 - val_l
oss: 0.2795
Epoch 2/4
1337/1337 ━━━━━━━━━━━━━━━━━━━━ 288s 215ms/step - accuracy: 0.4532 - loss: 0.2727 - val_accuracy: 0.4467 - val_l
oss: 0.2750
Epoch 3/4
1337/1337 ━━━━━━━━━━━━━━━━━━━━ 288s 216ms/step - accuracy: 0.4704 - loss: 0.2620 - val_accuracy: 0.4356 - val_l
oss: 0.2752
Epoch 4/4
1337/1337 ━━━━━━━━━━━━━━━━━━━━ 290s 217ms/step - accuracy: 0.4869 - loss: 0.2558 - val_accuracy: 0.4467 - val_l
oss: 0.2739
```

Learning rate: 0.0001 #epoch:10 inputsize:100 pathsize:128 decaystep:10000 …….

```
Epoch 1/10
1337/1337 ━━━━━━━━━━━━━━━━━━━━ 272s 201ms/step - accuracy: 0.3176 - loss: 0.3604 - val_accuracy: 0.3179 - val_loss: 0.3286
Epoch 2/10
1337/1337 ━━━━━━━━━━━━━━━━━━━━ 265s 198ms/step - accuracy: 0.3242 - loss: 0.3281 - val_accuracy: 0.3179 - val_loss: 0.3286
Epoch 3/10
1337/1337 ━━━━━━━━━━━━━━━━━━━━ 266s 199ms/step - accuracy: 0.3207 - loss: 0.3280 - val_accuracy: 0.3589 - val_loss: 0.3079
Epoch 4/10
1337/1337 ━━━━━━━━━━━━━━━━━━━━ 265s 198ms/step - accuracy: 0.3622 - loss: 0.3036 - val_accuracy: 0.3736 - val_loss: 0.2966
Epoch 5/10
1337/1337 ━━━━━━━━━━━━━━━━━━━━ 269s 201ms/step - accuracy: 0.3866 - loss: 0.2932 - val_accuracy: 0.3855 - val_loss: 0.2925
Epoch 6/10
1337/1337 ━━━━━━━━━━━━━━━━━━━━ 271s 202ms/step - accuracy: 0.4059 - loss: 0.2865 - val_accuracy: 0.3962 - val_loss: 0.2885
Epoch 7/10
1337/1337 ━━━━━━━━━━━━━━━━━━━━ 271s 203ms/step - accuracy: 0.4226 - loss: 0.2806 - val_accuracy: 0.4047 - val_loss: 0.2863
Epoch 8/10
1337/1337 ━━━━━━━━━━━━━━━━━━━━ 268s 200ms/step - accuracy: 0.4302 - loss: 0.2763 - val_accuracy: 0.4073 - val_loss: 0.2840
Epoch 9/10
1337/1337 ━━━━━━━━━━━━━━━━━━━━ 269s 201ms/step - accuracy: 0.4385 - loss: 0.2726 - val_accuracy: 0.4159 - val_loss: 0.2826
Epoch 10/10
1337/1337 ━━━━━━━━━━━━━━━━━━━━ 269s 201ms/step - accuracy: 0.4493 - loss: 0.2689 - val_accuracy: 0.4168 - val_loss: 0.2824
```

در شکل زیر می توان متوجه شد که:

- شکاف فزاینده بین دقت آموزش و اعتبارسنجی، و همچنین تلفات واگرا، نشانه ای از تطبیق بیش از حد پس از دوره دوم را نشان می دهد.
- این مدل ممکن است داده های آموزشی را خیلی خوب یاد بگیرد و به طور موثر به داده های اعتبار سنجی تعمیم ندهد.



مدل دقت حدود 50% را دارا می باشد

نکته: افزایش تعداد لایه ها بر مدل تاثیری مثبتی نداشت برای مثال: BatchNormaliztion بین conv1d_layeو global_max_pooling_layer

گزارش Evaluation data روی مدل نهایی بدست آمده :

```
Best thresholds for each class: [0.24885656 0.249157   0.24167001 0.24031973 0.27781302 0.10143355
 0.26373702 0.13712554 0.16975009 0.43053997]

Classification Report:
              precision    recall  f1-score   support

      praise       0.45      0.45      0.45      1708
   amusement       0.47      0.51      0.49       727
       anger       0.38      0.46      0.42      1010
disapproval       0.25      0.31      0.27       771
   confusion       0.50      0.72      0.59      3586
    interest       0.27      0.19      0.22       634
     sadness       0.49      0.38      0.42       831
        fear       0.44      0.33      0.38       248
         joy       0.26      0.49      0.34      1354
        love       0.71      0.62      0.66      1227

   micro avg       0.43      0.52      0.47     12096
   macro avg       0.42      0.45      0.42     12096
weighted avg       0.44      0.52      0.47     12096
 samples avg       0.46      0.53      0.48     12096


Accuracy: 0.49746070819920474
```

سایر موارد

- Layer freezing و انواع sampling روی مدل تست شده است
- focal_loss نیز در مدل تست شده است

```python
def focal_loss(alpha=0.25, gamma=2.0):
    def loss(y_true, y_pred):
        bce = K.binary_crossentropy(y_true, y_pred)
        p_t = y_true * y_pred + (1 - y_true) * (1 - y_pred)
        return K.mean(alpha * (1 - p_t) ** gamma * bce)
    return loss
```

- Tunning هایپر پارامتر ها با مقادیر مختلف بررسی شده است نمونه آنها در بالا ذکر شده باقی موارد در اشکال زیر قابل مشاهده می باشد

```
lr 4.816525004971233e-05, Epoch 1/8, Train Loss: 0.3436532951593399
Validation Accuracy: 0.2306, F1 Score: 0.4524730804845848, Best_threshold:0.5748448371887207
lr 4.816525004971233e-05, Epoch 2/8, Train Loss: 0.29602280781269075
Validation Accuracy: 0.2276, F1 Score: 0.4592890004237399, Best_threshold:0.5019522309303284
lr 4.816525004971233e-05, Epoch 3/8, Train Loss: 0.27074915018081663
Validation Accuracy: 0.2332, F1 Score: 0.4642292203293826, Best_threshold:0.6257055997848511
lr 4.816525004971233e-05, Epoch 4/8, Train Loss: 0.2485907057762146
Validation Accuracy: 0.261, F1 Score: 0.4606644165716386, Best_threshold:0.5979422330856323
lr 4.816525004971233e-05, Epoch 5/8, Train Loss: 0.22110454186201095
Validation Accuracy: 0.2196, F1 Score: 0.4575443130100119, Best_threshold:0.492294579744339
lr 4.816525004971233e-05, Epoch 6/8, Train Loss: 0.19578207646608353
Validation Accuracy: 0.1712, F1 Score: 0.4421042264652947, Best_threshold:0.5870124697685242
lr 4.816525004971233e-05, Epoch 7/8, Train Loss: 0.17016463284492492
Validation Accuracy: 0.1748, F1 Score: 0.43696792406299007, Best_threshold:0.4928027093410492
lr 4.816525004971233e-05, Epoch 8/8, Train Loss: 0.14823489870429038
Validation Accuracy: 0.1622, F1 Score: 0.43011982136412474, Best_threshold:0.7155377864837646
```

```
lr 4.521099668346832e-05, Epoch 1/6, Train Loss: 0.3450117794990539
Validation Accuracy: 0.1964, F1 Score: 0.4607133120984117, Best_threshold:0.4590798020362854
lr 4.521099668346832e-05, Epoch 2/6, Train Loss: 0.293466538977623
Validation Accuracy: 0.2284, F1 Score: 0.47705992054219043, Best_threshold:0.5334787964820862
lr 4.521099668346832e-05, Epoch 3/6, Train Loss: 0.2688123738527298
Validation Accuracy: 0.1906, F1 Score: 0.46010116371499643, Best_threshold:0.2230965793132782
lr 4.521099668346832e-05, Epoch 4/6, Train Loss: 0.24201123020648957
Validation Accuracy: 0.1578, F1 Score: 0.44978791658100403, Best_threshold:0.6456204056739807
lr 4.521099668346832e-05, Epoch 5/6, Train Loss: 0.21596418390274047
Validation Accuracy: 0.2484, F1 Score: 0.4507790576991866, Best_threshold:0.6094832420349121
lr 4.521099668346832e-05, Epoch 6/6, Train Loss: 0.18764299721717834
Validation Accuracy: 0.2178, F1 Score: 0.4447650992985621, Best_threshold:0.9160576462745667
```

```
lr 2.5038909189822072e-05, Epoch 1/5, Train Loss: 0.35213318898677826
Validation Accuracy: 0.2182, F1 Score: 0.4642931978563274, Best_threshold:0.5842392444610596
lr 2.5038909189822072e-05, Epoch 2/5, Train Loss: 0.2943249342441559
Validation Accuracy: 0.2104, F1 Score: 0.4630716888533006, Best_threshold:0.5278831720352173
lr 2.5038909189822072e-05, Epoch 3/5, Train Loss: 0.26611732289791107
Validation Accuracy: 0.221, F1 Score: 0.46144044464821093, Best_threshold:0.6579594612121582
lr 2.5038909189822072e-05, Epoch 4/5, Train Loss: 0.2384158760547638
Validation Accuracy: 0.2436, F1 Score: 0.4650051136814673, Best_threshold:0.5505707859992981
lr 2.5038909189822072e-05, Epoch 5/5, Train Loss: 0.20808387701511383
Validation Accuracy: 0.1822, F1 Score: 0.44752569535587056, Best_threshold:0.6404014229774475
```

```
lr 8.366245098532627e-05, Epoch 1/6, Train Loss: 0.3754875840187073
Validation Accuracy: 0.0676, F1 Score: 0.38092449668365325, Best_threshold:0.8266355395317078
lr 8.366245098532627e-05, Epoch 2/6, Train Loss: 0.327436577129364
Validation Accuracy: 0.1698, F1 Score: 0.4444204119933953, Best_threshold:0.5106073617935181
lr 8.366245098532627e-05, Epoch 3/6, Train Loss: 0.3048241612195969
Validation Accuracy: 0.192, F1 Score: 0.4428220944469895, Best_threshold:0.45036762952804565
lr 8.366245098532627e-05, Epoch 4/6, Train Loss: 0.2859416137218475
Validation Accuracy: 0.214, F1 Score: 0.4483203046707939, Best_threshold:0.5428503155708313
lr 8.366245098532627e-05, Epoch 5/6, Train Loss: 0.27004332525730135
Validation Accuracy: 0.2156, F1 Score: 0.4427638105081595, Best_threshold:0.3411511778831482
lr 8.366245098532627e-05, Epoch 6/6, Train Loss: 0.3122679966926575
Validation Accuracy: 0.0, F1 Score: 0.20950350434780396, Best_threshold:0.1280585527420044
lr 0.00013808708509165357, Epoch 1/9, Train Loss: 0.38518665811886044
Validation Accuracy: 0.0762, F1 Score: 0.3494137729211522, Best_threshold:0.4582127034664154
lr 0.00013808708509165357, Epoch 2/9, Train Loss: 0.3617896378611604
Validation Accuracy: 0.1286, F1 Score: 0.37623927628185805, Best_threshold:0.8640435338020325
lr 0.00013808708509165357, Epoch 3/9, Train Loss: 0.360603840396808
Validation Accuracy: 0.1558, F1 Score: 0.3651305406165235, Best_threshold:0.5837913751602173
lr 0.00013808708509165357, Epoch 4/9, Train Loss: 0.3497696943557301
Validation Accuracy: 0.1302, F1 Score: 0.377091887354874, Best_threshold:0.12158820778131485
lr 0.00013808708509165357, Epoch 5/9, Train Loss: 0.34225492384106204
Validation Accuracy: 0.1214, F1 Score: 0.38269666115549256, Best_threshold:0.7195218205451965
lr 0.00013808708509165357, Epoch 6/9, Train Loss: 0.35026314578498136
Validation Accuracy: 0.0464, F1 Score: 0.3194724690446903, Best_threshold:0.47862884402275085
lr 0.00013808708509165357, Epoch 7/9, Train Loss: 0.33975787570301336
Validation Accuracy: 0.0972, F1 Score: 0.37218920919151943, Best_threshold:0.7671162486076355
lr 0.00013808708509165357, Epoch 8/9, Train Loss: 0.3416533396838191
Validation Accuracy: 0.0992, F1 Score: 0.34979524341947726, Best_threshold:0.722552478313446
lr 0.00013808708509165357, Epoch 9/9, Train Loss: 0.35182587074014704
Validation Accuracy: 0.0, F1 Score: 0.24045405257361435, Best_threshold:0.1570020318031311
```

VII. API Interface

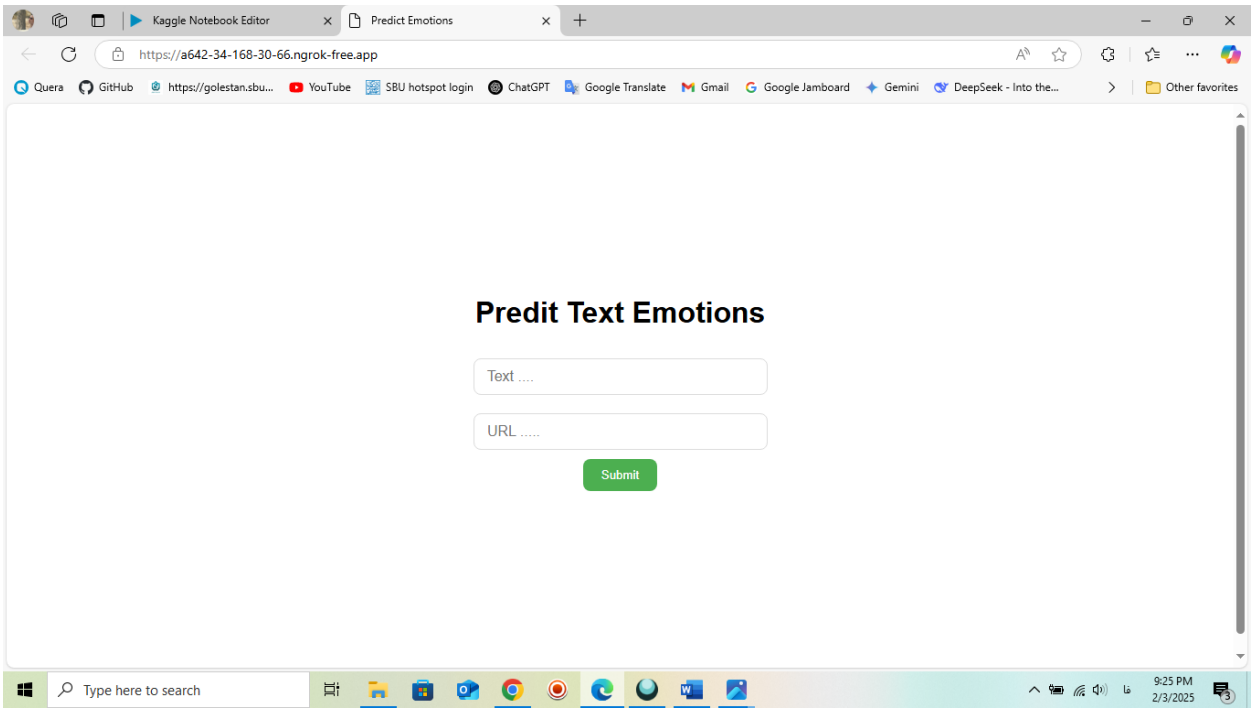
- از کتابخانه ngrok برای ایجاد public URL کمکی ایجاد شده است که یکبار مصرف می باشد و بعد از قطع هر ارتباط قابل استفاده نمی باشد.

```python
from pyngrok import ngrok

# Open a tunnel on port 8000
public_url = ngrok.connect(8000)
print(f"Public URL: {public_url}")
```

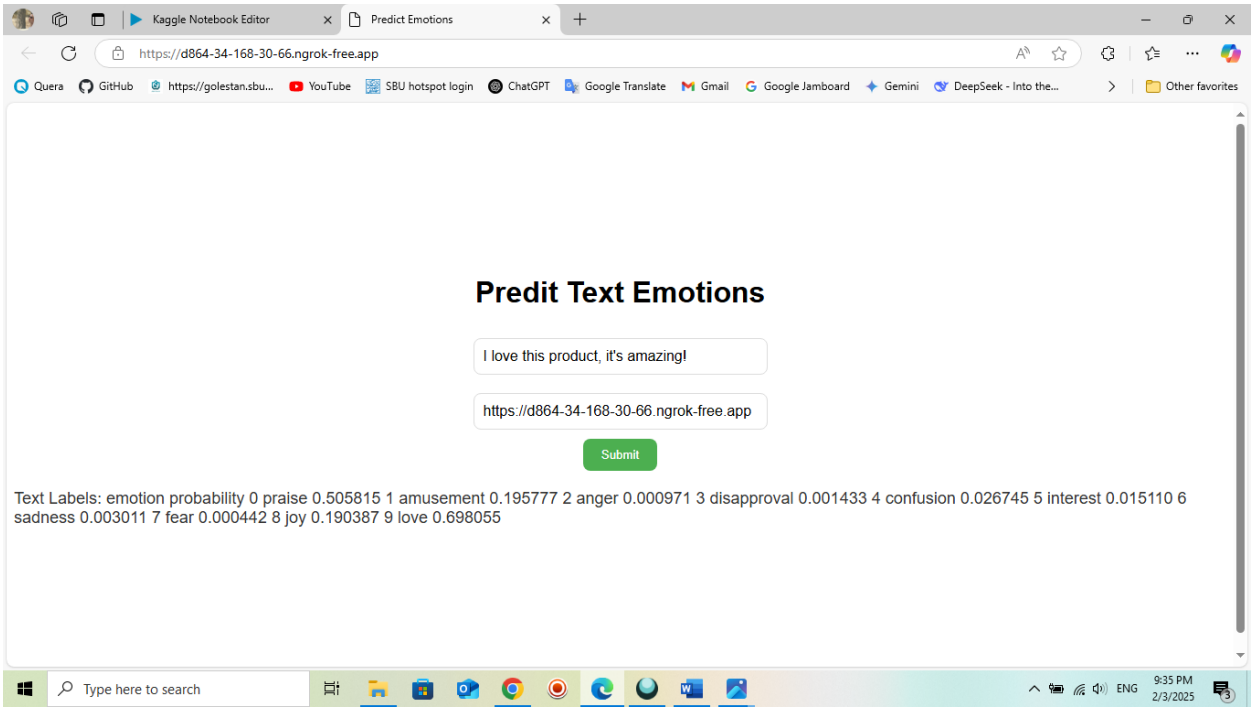
- از uvicorn هم برای اجرا روی سرور استفاده شده است
- برای برقراری ارتباط بین سرور و کلاینت نیاز به یک سری توابع GET و POST نیاز هست که قرار ذیل می باشد
- Home.html نیز برای سمت فرانت اند پیاده سازی شده است

```python
nest_asyncio.apply()

app = FastAPI()

# Set up templates directory
templates = Jinja2Templates(directory="../input/emotrainval")

@app.get("/", response_class=HTMLResponse)
async def home(request: Request):
    return templates.TemplateResponse("home.html", {"request": request})

# Define Pydantic Model
class TextRequest(BaseModel):
    text: str

@app.post("/process/")
async def process_text(request: TextRequest):
    result = predict_text(request.text)
    # result = request.text
    return {"response": f"Text Labels: {result}"}


uvicorn.run(app, host="0.0.0.0", port=8000)
```

```
INFO:     Started server process [31]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
```



- برای اینکه URL یکبار مصرف می باشد بهتر است آنرا بشکل دستی وارد نماییم(مشکل اصلی مرتبط با Kaggle است که با localhost کار نمی کند در خارج از Kaggle می توان هاست آدرس را بصورت ثابت وارد نمود و نیاز به ارسال URL نمی باشد)



VIII. تست مدل روی دیتای داده شده

```
Best thresholds for each class: [0.21425098 0.26231915 0.140062   0.1113196  0.32178858 0.08604681
 0.17121609 0.18891239 0.2784043  0.22402672]

Classification Report:
              precision    recall  f1-score   support

      praise       0.51      0.53      0.52       167
   amusement       0.55      0.60      0.58        68
       anger       0.36      0.61      0.45       100
disapproval       0.18      0.51      0.27        75
   confusion       0.47      0.77      0.59       310
    interest       0.18      0.30      0.22        69
     sadness       0.38      0.41      0.39        85
        fear       0.65      0.31      0.42        35
         joy       0.46      0.43      0.45       105
        love       0.67      0.65      0.66       127

   micro avg       0.42      0.58      0.49      1141
   macro avg       0.44      0.51      0.46      1141
weighted avg       0.46      0.58      0.50      1141
 samples avg       0.46      0.59      0.49      1141


Accuracy: 0.56
```

قابل ذکر است، LOVE (0.67 PRECISION, 0.65 RECALL, 0.66 F1-SCORE) و CONFUSION (0.47 PRECISION, 0.77 RECALL, 0.59 F1-SCORE)
از جمله برچسب های با بهترین عملکرد هستند، که نشان می دهد مدل ارسال شده به طور موثر این برچسب ها را ثبت می کند برعکس، DISAPPROVAL (0.18 PRECISION، 0.51 RECALL، 0.27 F1-SCORE) و INTEREST (0.18 PRECISION, 0.30 RECALL, 0.22 F1-SCORE) عملکرد ضعیفی دارند که نشان دهنده مشکل در تشخیص این احساسات است. میانگین 0.46 MACRO AVERAGE F1-Score عملکرد کلی متوسط را نشان می دهد، با عدم تعادل طبقاتی که بر برچسب های خاص تأثیر می گذارد. میانگین وزنی F1-Score 0.50 منعکس کننده تأثیر فرکانس برچسب بر عملکرد است. این مدل دارای یادآوری بالاتر (0.58 WEIGHTED AVG) از PRECISION (0.46 WEIGHTED AVG) است، به این معنی که به درستی موارد عاطفی واقعی بیشتری را شناسایی می کند اما به قیمت مثبت کاذب است.