Sentiment Analysis - Bahar Khouban

Sentiment Analysis یکی از کاربردهای مهم NLP است که هدف آن تشخیص و استخراج حالات و احساسات بیان‌شده در متون است. این تسک شامل شناسایی و دسته‌بندی متون به احساسات مثبت، منفی و یا خنثی می‌باشد.

برخی از کاربردهای رایج Sentiment Analysis شامل موارد زیر است:

بررسی بازخورد مشتریان: تحلیل نظرات و بازخوردهای مشتریان در مورد محصولات یا خدمات.

رصد شبکه‌های اجتماعی: پایش و تحلیل احساسات بیان‌شده در پست‌ها و نظرات کاربران در شبکه‌های اجتماعی.

تحلیل بازار: تحلیل احساسات سرمایه‌گذاران و تحلیلگران در مورد سهام یا بازارهای مالی.

مدیریت شهرت: رصد و تحلیل افکار عمومی در مورد برندها و افراد مشهور.

استفاده از دیتاست SST-2

دیتاست SST-2 یکی از معتبرترین و پرکاربردترین دیتاست‌ها در زمینه تحلیل احساسات است. این دیتاست شامل جملات انگلیسی از بررسی‌های فیلم‌ها و برچسب‌های احساسی مربوط به هر جمله است. SST-2 نسخه‌ای از دیتاست SST است که در آن جملات به دو دسته احساسات مثبت و منفی دسته‌بندی شده‌اند.

استفاده از دیتاست SST-2 به دلیل کیفیت و اعتبار آن، پوشش گسترده احساسات، تعادل در داده‌ها، و استفاده گسترده در پژوهش‌های علمی، انتخاب مناسبی برای پروژه‌های تحلیل احساسات است. این دیتاست به ما کمک می‌کند تا مدل‌های دقیقی برای تشخیص احساسات در متون آموزش دهیم و نتایج قابل اعتمادی به دست آوریم.

مدل BERT

BERT یکی از مدل‌های پیشرفته و پرکاربرد در زمینه پردازش زبان طبیعی است که بر اساس معماری ترنسفورمرها ساخته شده است. دلایل انتخاب BERT برای این تسک شامل موارد زیر می شود:

- Fine Tuning ، مدل های بسیار زیادی روی برت وجود دارند که با داده های بسیار زیادی آموزش داده شده اند.
- مدل‌های مبتنی بر برت در بسیاری از تسک‌های NLP نتایج خوبی به دست آورده‌اند و عملکرد بهتری نسبت به مدل‌های قبلی مانند LSTM و GRU نشان داده‌اند.
- کامیونیتی بزرگ: BERT توسط جامعه بزرگی از پژوهشگران و توسعه‌دهندگان پشتیبانی می‌شود و بسیاری از پیاده‌سازی‌ها و مدل‌های پیش‌آموزش دیده در دسترس هستند که استفاده از آن را آسان‌تر می‌کند.

پیاده سازی پروژه

```
from google.colab import drive
 drive.mount('/content/drive')
```

متصل شده به اکانت گوگل درایو برای دسترسی به دیتاست ها و ذخیره کردن مدل. همچنین می توان این کار را با دریافت داده ها به صورت مستقیم از hugging face و لاگین شدن از طریق نوت بوک و push to hub در هاگینگ فیس منتشر کرد.

```
pip install transformers!
```

سپس کتابخانه transoferms رو با این دستور نصب میکنیم. از مزایای آنها میتوان اشاره کرد به مدل‌های پیش‌آموزش دیده مختلفی مانند DistilBERT، RoBERTa، GPT، BERT و بسیاری دیگر است که می‌توانند برای تسک‌های مختلف NLP استفاده شوند. این کتابخانه رابط کاربری ساده و کاربرپسندی دارد که امکان استفاده از مدل‌های پیچیده را فراهم می‌کند. همچنین از تسک‌های مختلفی مانند text generator، classification، translation، پاسخ به سوالات و sentiment analysis پشتیبانی می‌کند.

```
from transformers import pipeline

sentiment_pipeline = pipeline("sentiment-analysis",
model="distilbert-base-uncased")



texts = [

    "I love this book!",

    "This is the worst book I have ever read."

]



results = sentiment_pipeline(texts)



for text, result in zip(texts, results):
```

```
    print(f"Text: {text}Sentiment: {result['label']}, Score:
{result['score']}")
```

تابع pipleline از کتابخانه transformers، یک رابط ساده و کاربرپسند برای انجام تسک‌های مختلف NLP با استفاده از مدل‌های پیش‌آموزش دیده هست.

```
sentiment_pipeline = pipeline("sentiment-analysis",
model="distilbert-base-uncased")
```

در اینجا، یک پایپ‌لاین برای تسک مورد نظر ایجاد می‌کنیم. پارامتر اول، نوع تسک را مشخص می‌کند که در این مورد sentiment-analysis است. پارامتر دوم، مدل مورد استفاده را تعیین می‌کند که در اینجا distilbert-base-uncased انتخاب شده است که این مدل نسخه‌ی کوچکتر و سریعتر مدل BERT است که برای تسک‌های مشابه دقت بالایی ارائه می‌دهد.

```
texts = [
    "I love this book!",
    "This is the worst book I have ever read."
]

results = sentiment_pipeline(texts)

for text, result in zip(texts, results):
    print(f"Text: {text}\nSentiment: {result['label']}, Score:
{result['score']}\n")
```

در این بخش، برای تست کردن مدل distilbert دو جمله به عنوان ورودی تعریف شده‌اند. یکی جمله‌ای با احساس مثبت و دیگری جمله‌ای با احساس منفی.
که خروجی ها مطابق زیر میباشند:

```
Text: I love this book!
Sentiment: LABEL_1, Score: 0.5055715441703796

Text: This is the worst book I have ever read.
Sentiment: LABEL_1, Score: 0.5226098895072937
```

```
import json
import pandas as pd
import re
import string
from sklearn.preprocessing import LabelEncoder
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

with open('/content/drive/My Drive/NLP Challenge/SST2_Train.json', 'r') as
file:
    data = json.load(file)

df = pd.DataFrame(data)

def preprocess_text(text):
    text = text.lower()
    text = ''.join(text)
    return text

df['text'] = df['text'].apply(preprocess_text)
label_encoder = LabelEncoder()
df['encoded_label'] = label_encoder.fit_transform(df['label'])
df['label'] =
label_encoder.inverse_transform(label_encoder.fit_transform(df['label']))
df.to_csv('/content/drive/My Drive/NLP Challenge/PreProcessed_Train.csv',
index=False)
print(df.head())
```

json برای خواندن داده‌های JSON.

pandas برای دستکاری و تحلیل داده‌ها.

LabelEncoder از sklearn برای رمزگذاری برچسب‌ها.

```
with open('/content/drive/My Drive/NLP Challenge/SST2_Train.json', 'r') as
file:
```

```
    data = json.load(file)

df = pd.DataFrame(data)
```

این بخش از کد، فایل JSON که فایل Trainهست را از Google Drive می‌خواند و آن را در متغیر data ذخیره می‌کند. سپس آن را تبدیل به دیتافریم میکند به کمک pandas

```
def preprocess_text(text):
    text = text.lower()
    text = ''.join(text)
    return text

df['text'] = df['text'].apply(preprocess_text)
label_encoder = LabelEncoder()
df['encoded_label'] = label_encoder.fit_transform(df['label'])
df['label'] =
label_encoder.inverse_transform(label_encoder.fit_transform(df['label']))
df.to_csv('/content/drive/My Drive/NLP Challenge/PreProcessed_Train.csv',
index=False)
print(df.head())
```

تابع preprocess_text برای پیش پردازش متن تعریف شده است.

متن ها lowercas میشوند زیرا

- کلمات با حروف بزرگ و کوچک به صورت یکسان در نظر گرفته شوند و مدل بتواند به طور یکسان با آنها کار کند.
- با تبدیل تمام کلمات به حروف کوچک، تعداد توکن‌های یکتا کاهش می‌یابد که باعث می‌شود مدل به شکلی کارآمدتر عمل کند.

از آنجایی که تصمیم داریم از مدل Bert استفاده کنیم حذف stopwordها و کاراکترها و توکنایز کردن به ارزیابی لطمه میزند. مدل BERT خودش به طور داخلی توکن‌سازی را انجام می‌دهد. BERT از یک توکنایزر مخصوص به خود به نام "WordPiece" استفاده می‌کند که توانایی تجزیه wordsبه subwordsرا دارد. این روش به BERT اجازه می‌دهد تا با کلمات ناشناخته کار کند و نیازی به توکن‌سازی مجزا در مرحله پیش‌پردازش نیست. برخلاف برخی از مدل‌های ساده‌تر، BERT به Stop words نیز توجه می‌کند. کلمات توقف مانند "in", "the", "is" ممکن است حاوی اطلاعات مهمی در مورد ساختار جملات باشند و حذف آن‌ها می‌تواند به از دست دادن اطلاعات مفید منجر شود. BERT از تمام کلمات برای درک کامل معنای جملات و کانتکست آن استفاده می‌کند. پانکچوئیشن‌ها

نیز مانند نقطه، ویرگول و علامت سوال می‌توانند در فهم ساختار و معنای جملات بسیار مهم باشند. BERT به گونه‌ای طراحی شده که نشانه‌گذاری‌ها را در نظر بگیرد و آن‌ها را به عنوان بخشی از ورودی پردازش کند. حذف این نشانه‌گذاری‌ها می‌تواند منجر به از دست دادن اطلاعات ارزشمندی شود که برای درک بهتر جملات ضروری است.

در بخش بعد از انکودر برای لیبل ها استفاده میکنیم تا مدل‌های یادگیری ماشین بتوانند با آن‌ها کار کنند.
سپس آن را تبدیل به csv میکنیم و ذخیره میکنیم.

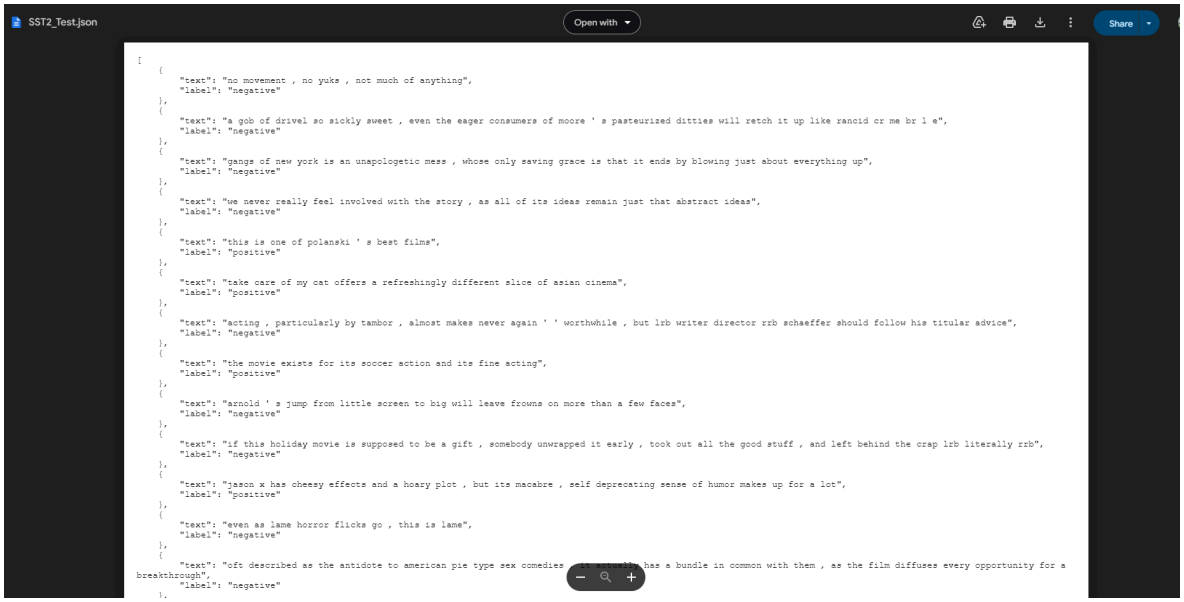
دیتاست

به دیتاست sst2 می توان به صورت مستقیم از datasetهای huggingface دسترسی داشت ولی در اینجا از دیتاست لینک شده در متن پروژه استفاده شده که به صورت زیر میباشد:

Train:


```
[
    {
        "text": "a stirring , funny and finally transporting re - imagining of beauty and the beast and 1930s horror films",
        "label": "positive"
    },
    {
        "text": "apparently reassembled from the cutting - room floor of any given daytime soap .",
        "label": "negative"
    },
    {
        "text": "they presume their audience wo n ' t sit still for a sociology lesson , however entertainingly presented , so they trot out the conventional science - fiction
elements of bug - eyed monsters and futuristic women in skimpy clothes .",
        "label": "negative"
    },
    {
        "text": "this is a visually stunning rumination on love , memory , history and the war between art and commerce .",
        "label": "positive"
    },
    {
        "text": "jonathan parker ' s bartleby should have been the be - all - end - all of the modern - office anomie films .",
        "label": "positive"
    },
    {
        "text": "campanella gets the tone just right - - funny in the middle of sad in the middle of hopeful .",
        "label": "positive"
    },
    {
        "text": "a fan film that for the uninitiated plays better on video with the sound turned down .",
        "label": "negative"
    },
    {
        "text": "b [UNK] art and berling are both superb , while huppert . . . is magnificent .",
        "label": "positive"
    },
    {
        "text": "a little less extreme than in the past , with longer exposition sequences between them , and with fewer gags to break the tedium .",
        "label": "negative"
    },
    {
        "text": "the film is strictly routine .",
        "label": "negative"
    },
    {
        "text": "a lyrical metaphor for cultural and personal self - discovery and a picaresque view of a little - remembered world .",
        "label": "positive"
    },
    {
        "text": "the most repugnant adaptation of a classic text since roland joff [UNK] and demi moore ' s the scarlet letter .",
        "label": "negative"
    },
    {
        "text": "for something as splendid - looking as this particular film , the viewer expects something special but instead gets - lrb - sci - fi - rrb - rehash .",
        "label": "negative"
```


Test:



دیتاست شامل حدود 7500 داده برای train و valid و 1800 رکورد برای تست ست هستش که به فرمت JSON هستند.

```
import pandas as pd

df = pd.read_csv('/content/drive/My Drive/NLP Challenge/PreProcessed_Train.csv')
random_seed = 42
validation_df = df.sample(frac=0.1, random_state=random_seed)
train_df = df.drop(validation_df.index)
train_file_path = '/content/drive/My Drive/NLP Challenge/PreProcessed_TrainFinal.csv'
validation_file_path = '/content/drive/My Drive/NLP Challenge/PreProcessed_ValFinal.csv'
train_df.to_csv(train_file_path, index=False)
validation_df.to_csv(validation_file_path, index=False)
train_df.head(), validation_df.head(), train_file_path, validation_file_path
```

```
random_seed = 42
```

این دستور یک random seed تصادفی تعیین می‌کند. استفاده از random seed باعث می‌شود که نتایج تقسیم داده‌ها همیشه یکسان باشند و امکان reproducibility نتایج فراهم شود.

```
validation_df = df.sample(frac=0.1, random_state=random_seed)
```

۱۰٪ از دیتاها به صورت تصادفی و با استفاده از random seed مشخص شده و به عنوان مجموعه validation انتخاب می‌شوند. تابع sample از pandas برای این کار استفاده می‌شود و پارامتر frac=0.1 به معنی انتخاب ۱۰٪ از داده‌ها است.

```
train_df = df.drop(validation_df.index)
train_file_path = '/content/drive/My Drive/NLP
Challenge/PreProcessed_TrainFinal.csv'
validation_file_path = '/content/drive/My Drive/NLP
Challenge/PreProcessed_ValFinal.csv'
train_df.to_csv(train_file_path, index=False)
validation_df.to_csv(validation_file_path, index=False)
train_df.head(), validation_df.head(), train_file_path,
validation_file_path
```

۹۰٪ باقی دیتاها به عنوان train set در نظر گرفته میشود. سپس در دایرکتوری مد نظر سیو میشود.

```
testdf['text'] = testdf['text'].apply(preprocess_text)
label_encoder = LabelEncoder()
testdf['encoded_label'] = label_encoder.fit_transform(testdf['label'])
testdf['label'] =
label_encoder.inverse_transform(label_encoder.fit_transform(testdf['label'
]))
testdf.to_csv('/content/drive/My Drive/NLP
Challenge/PreProcessed_TestFinal.csv', index=False)
print(testdf.head())
```

سپس همین مراحل را برای test set data نیز اجرا می‌کنیم. Preprocessing را انجام می دهیم و encoded label رو اضافه میکنیم.

```
import torch

!pip install datasets transformers huggingface_hub
```

این دستور کتابخانه torch را وارد می‌کند که توسط PyTorch توسعه داده شده است.

دستور بعدی transfomers, datasets, huggingface_hub را نصب میکند. dataset توسط huggingface راه اندازی شده است و امکان دسترسی به دیتاست‌های مختلف را فراهم میکند. Huggingface_hub ابزارهای ارتباط با هاب مدل‌های Hugging Face را فراهم می‌کند.

```
!pip install pyarrow==15.0.2
```

pyarrow یک بسته‌ی Python برای Apache Arrow است. Apache Arrow یک قالب درون حافظه‌ای ست که به منظور بهینه‌سازی تجزیه و تحلیل داده‌ها توسعه داده شده است. برای ذخیره و پردازش داده‌های بزرگ به صورت کارآمد طراحی شده است.

مثال دیتا فریم valid:

	text	label	encoded_label
0	unlike most surf movies , blue crush thrilling...	positive	1
1	all leather pants & augmented boobs , hawn is ...	positive	1
2	detox is ultimately a pointless endeavor .	negative	0
3	in between the icy stunts , the actors spout h...	positive	1
4	a funny and touching film that is gorgeously a...	positive	1
...	...	...	...
740	at the end , when the now computerized yoda fi...	positive	1
741	if they broke out into elaborate choreography ...	negative	0
742	the holes in this film remain agape - - holes ...	negative	0

743 however it may please those who love movies th... negative 0

744 it reaffirms life as it looks in the face of d... positive 1

745 rows × 3 columns

```
from datasets import load_dataset, Features, Value, DatasetDict, Dataset

import pandas as pd



train_df['idx'] = range(len(train_df))

test_df['idx'] = range(len(test_df))

valid_df['idx'] = range(len(valid_df))

train_df = train_df.drop(columns=['label'])

test_df = test_df.drop(columns=['label'])

valid_df = valid_df.drop(columns=['label'])

train_df = train_df.rename(columns={'text': 'sentence', 'encoded_label':
'label'})

test_df = test_df.rename(columns={'text': 'sentence', 'encoded_label':
'label'})

valid_df = valid_df.rename(columns={'text': 'sentence', 'encoded_label':
'label'})
```

طبق ساختار مدل لازم است تغییراتی روی دیتاست اعمال شود، ستون idx اضافه شود، این دستورات یک ستون جدید به نام idx به هر DataFrame اضافه می‌کنند که شامل ایندکس‌های منحصر به فرد برای هر ردیف است.
همچنین لازم است ستون label که به صورت استرینگ هست حذف شود و encoded label به label تغییر کند
0: negative
1: positive

اسم ستون text نیز به sentence تغییر پیدا میکند.

DatasetDict({

```
    train: Dataset({
        features: ['idx', 'sentence', 'label'],
        num_rows: 6702
    })
    test: Dataset({
        features: ['idx', 'sentence', 'label'],
        num_rows: 1821
    })
    valid: Dataset({
        features: ['idx', 'sentence', 'label'],
        num_rows: 745
    })
})
Dataset({
    features: ['idx', 'sentence', 'label'],
    num_rows: 6702
})
```

```
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")
```

توکنایزرهای PreTrain شده تبدیل متن خام به توکن‌های قابل پردازش توسط مدل‌های یادگیری عمیق را بر عهده دارند. با استفاده از این دستور، توکنایزر مرتبط با مدل DistilBERT آماده استفاده می‌شود. این توکنایزر متن‌ها را به توکن‌های مناسب برای ورودی به مدل‌های DistilBERT تبدیل می‌کند.

```
def preprocess_function(examples):
    return tokenizer(examples['sentence'], truncation=True, padding=True)


tokenized_train = dataset['train'].map(preprocess_function, batched=True)
tokenized_test = dataset['test'].map(preprocess_function, batched=True)
tokenized_valid = dataset['valid'].map(preprocess_function, batched=True)
```

examples['sentence']: این پارامتر شامل جملات ورودی است که باید توکنایز شوند.

tokenizer: توکنایزرDistilBERT برای تبدیل متن خام به توکن‌ها استفاده می‌شود.

truncation=True: عمل truncate انجام میشود و جملات خیلی بلند کوتاه میشوند.

padding=True: این باعث می‌شود که جملات کوتاه‌تر با استفاده از پدینگ (padding) به طول یکسانی برسند.

```
from transformers import DataCollatorWithPadding
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
data_collator
```

DataCollatorWithPadding یک کلاس در transformers است که به صورت خودکار padding را به داده‌های ورودی اضافه می‌کند. این فرآیند ضروری است زیرا مدل‌های دیپ لرنینگ معمولاً نیاز دارند که تمامی نمونه‌های ورودی به طول یکسان باشند. DataCollatorWithPadding با استفاده از توکنایزر مشخص شده، داده‌های ورودی را به طول یکسان تبدیل می‌کند.

```
import numpy as np
from datasets import load_metric

def compute_metrics(eval_pred):
  load_accuracy = load_metric("accuracy")
  load_f1 = load_metric("f1")
  logits, labels = eval_pred
  predictions = np.argmax(logits, axis=-1)
  accuracy = load_accuracy.compute(predictions=predictions,
references=labels)["accuracy"]
  f1 = load_f1.compute(predictions=predictions, references=labels)["f1"]
  return {"accuracy": accuracy, "f1": f1}
```

این بخش تابع compute_metrics را تعریف می‌کند که برای محاسبه معیارهای ارزیابی عملکرد مدل استفاده می‌شود.

معیارهای accuracy و f1 با استفاده از تابع load_metric از کتابخانه datasets بارگذاری می‌شوند.

سپس logits (خروجی‌های مدل قبل از اعمال تابع softmax) و labels از ورودی eval_pred استخراج می‌شوند.

سپس، با استفاده از تابع np.argmax predictions تعیین می‌شوند.

Accuracy, F1-Score

Accuracy: معیاری است که نشان می‌دهد چه درصدی از پیش‌بینی‌های مدل درست بوده‌اند. دقت به صورت زیر محاسبه می‌شود:

Accuracy = تعداد پیش‌بینی‌های صحیح/ تعداد کل نمونه‌ها

دقت نسبت نمونه‌های صحیح پیش‌بینی شده به کل نمونه‌ها را نشان می‌دهد. این معیار به خصوص زمانی مفید است که تعداد کلاس‌ها یا دسته‌ها در داده‌ها متوازن باشند.

F1-Score: ترکیبی از Precision و Recall را در نظر می‌گیرد.

Precision: نشان می‌دهد که از میان تمام پیش‌بینی‌های مثبت مدل، چند درصد واقعاً مثبت بوده‌اند

Recall: نشان می‌دهد که از میان تمام نمونه‌های مثبت واقعی، چند درصد به درستی توسط مدل شناسایی شده‌اند.

F1-Score:

F1 Score= Precision * Recall / Precision + Recall

```
from transformers import TrainingArguments, Trainer

drive_output_dir = "/content/drive/My Drive/NLP
Challenge/finetuning-sentiment-model-sst2-samples"

training_args = TrainingArguments(
   output_dir=drive_output_dir,
   learning_rate=2e-5,
   per_device_train_batch_size=16,
   per_device_eval_batch_size=16,
   num_train_epochs=5,
   weight_decay=0.01,
   save_strategy="epoch"
)
```

این دستور کلاس‌های TrainingArguments و Trainer را transformers وارد می‌کند. TrainingArguments برای تنظیم پارامترهای آموزش و Trainer برای مدیریت فرآیند ترین کردن مدل استفاده می‌شود. سپس مسیر ذخیره کردن مدل رو مشخص میکنیم.

پارامترهای Training Arguments:

output_dir: مسیر ذخیره‌سازی نتایج آموزش مدل که قبلا مشخص شده است.

learning_rate: مقدار 2e-5 تنظیم شده است،تعیین می‌کند که مدل چقدر سریع یا آهسته به سمت optimiation حرکت کند.

per_device_train_batch_size: بچ batch size برای ترین مدل در دستگاه. مقدارش ۱۶ ست شده است.

per_device_eval_batch_size: اندازه‌ی بچ برای eval مدل در هر دستگاه. مقدار آن هم ۱۶ هست.

num_train_epochs: تعداد دوره‌های ترین (epochs) که ۵ بار ست شده است.

weight_decay: برای جلوگیری از overfitting استفاده می‌شود که 0.01 تنظیم شده است.

```
trainer = Trainer(
   model=model,
   args=training_args,
   train_dataset=tokenized_train,
   eval_dataset=tokenized_test,
   tokenizer=tokenizer,
   data_collator=data_collator,
   compute_metrics=compute_metrics,
)
```

سپس فرایند train شدن را تعریف میکنیم.

مدل distilbert را که تعریف کرده ایم. همچنین training args تعریف شده در قسمت قبل، train set و eval set دیتاست‌های لود شده هست که توکنایز شده‌اند. و compution metrics نیز تعریف شده است که F1 score و Accuracy را حساب میکند. Datacollector نیز قبلا تعریف شده و تنظیم داده‌ها و اضافه کردن padding به آن‌ها استفاده می‌شود.

```
trainer.train()
```

با این دستور مدل تعریف شده ترین میشود.

[Epoch 5/5 ,06:49 2095/2095]

Step	Training Loss
500	0.337700
1000	0.150600
1500	0.083500

0.040400	2000

```
TrainOutput(global_step=2095, training_loss=0.14742083492597702,
metrics={'train_runtime': 411.8827, 'train_samples_per_second': 81.358,
'train_steps_per_second': 5.086, 'total_flos': 692113180734144.0,
'train_loss': 0.14742083492597702, 'epoch': 5.0})
```

```
trainer.evaluate()
```

```
[114/114 00:03]
{'eval_loss': 0.9969585537910461,
 'eval_accuracy': 0.8978583196046128,
 'eval_f1': 0.8992416034669556,
 'eval_runtime': 5.5151,
 'eval_samples_per_second': 330.183,
 'eval_steps_per_second': 20.67,
 'epoch': 30.0}
```

1. eval_loss

0.46569469571113586

مدل نسبتاً به خوبی ترین شده است، اما هنوز مقداری خطا دارد.

2. eval_accuracy

0.9060955518945635

دقت مدل 90.61% است، که نشان می‌دهد مدل توانسته بیشتر نمونه‌های داده‌های evaluation را به درستی تشخیص دهد.

3. eval_f1

0.9075175770686857

نمایانگر تعادل خوب بین precision و recall مدل است.

4. eval_runtime

17.5595

زمان اجرای ارزیابی به ثانیه است.

5. eval_samples_per_second

103.705

تعداد نمونه‌هایی که مدل در هر ثانیه ارزیابی کرده.

eval_steps_per_second .6

6.492

تعداد گام‌های ارزیابی در هر ثانیه.

```
trainer.save_model(drive_output_dir)
tokenizer.save_pretrained(drive_output_dir)
```

توکنایزر مورد استفاده برای ترین کردن مدل را در همان دایرکتوری ذخیره می‌کنیم.

```
from transformers import AutoModelForSequenceClassification,
AutoTokenizer, pipeline

model_path = "/content/drive/My Drive/NLP
Challenge/finetuning-sentiment-model-sst2-samples"

model = AutoModelForSequenceClassification.from_pretrained(model_path)
tokenizer = AutoTokenizer.from_pretrained(model_path)
sentiment_model = pipeline("sentiment-analysis", model=model,
tokenizer=tokenizer)

results = sentiment_model(["I love this movie", "This movie sucks!"])
print(results)
```

مدلی که خود ترین کرده‌ایم را لود می‌کنیم. سپس دو جمله سمپل را به عنوان ورودی می‌دهیم و خروجی‌های دریافتی به صورت زیر می‌باشد:

```
Hardware accelerator e.g. GPU is available in the environment, but no
`device` argument is passed to the `Pipeline` object. Model will be on
CPU.
[{'label': 'LABEL_1', 'score': 0.999305248260498}, {'label': 'LABEL_0',
'score': 0.9988866448402405}]
```

که در مقایسه با distilbert که روی آن فاین تیون کردیم، عملکرد بسیاری بهتری در score دارد.