Institute for Advanced Studies
in Basic Sciences
GavaZang, Zanjan, Iran

Sentiment Analysis Model Loader

Prof. Majid Ramezani

Report for a Machine Learning Project <1>

Faeze Ahmadi

14044141

November 2025

تحلیل احساسات متن (Sentiment Analysis)

کد کامل این پروژه و فایل‌های مربوطه در ریپازیتوری گیت‌هاب قابل دسترسی هست:

https://github.com/Faeze-Ahmadi/sentiment-analysis-project

این پروژه یک سیستم ساده تحلیل احساسات (Sentiment Analysis) برای تشخیص مثبت یا منفی بودن متنی هست که از کاربر می‌گیریم. ایده اصلی این پروژه اینه که به کامپیوتر یاد بدیم محتوای یک متن رو بخونه و تشخیص بده که متن حس مثبت داره یا منفی؛ مثلا این متن می‌تونه نظری در مورد فیلم باشه. البته که دیتاست ما هم کامنت‌های مربوط به فیلم و استوری هست. یعنی ما با داشتن حجم زیادی از نظرهای مثبت و منفی و نشون دادن اون‌ها به مدلمون آموزش می‌دیم که چی مثبته و چی منفیه، تا وقتی کاربر بهش یه نظر داد، مدل ما خودش پیش‌بینی کنه که اون نظر مثبته یا منفی.

این پروژه دو بخش اصلی داره:

1. تبدیل متن خام به اعداد قابل فهم برای ماشین (به کمک TF-IDF): چون کامپیوتر نمی‌تونه متن رو بخونه و بخاطر همین ما باید متن‌ها رو به عدد تبدیل کنیم و اینطوری مدلمون رو آموزش بدیم. TF-IDF مخفف عبارت (Term Frequency – Inverse Document Frequency) هست که به عنوان یک روش معروف برای تبدیل متن به عدد شناخته می‌شه. Term Frequency (فراوانی اصطلاح) یعنی یه متن چندبار داخل متن تکرار شده و Inverse Document Frequency (معکوس فراوانی متن) یعنی این کلمه بین همه متن‌ها چقدر خاص یا کمیاب هست.

2. آموزش یک مدل Machine Learning برای تشخیص احساسات متن‌ها: بحث آموزش دادن هم بر اساس برچسب زدن به متن‌هاست. ما از مدل Logistic Regression برای این کار استفاده می‌کنیم. این مدل، برای classification هست. در واقع ما نیاز داریم که اینجا 0 و 1 ای فکر کنیم، یعنی اگر مدل مثبته پس 1 هست و اگر مدل منفیه پس 0 هست. اما بحث اینجاست که توی متن ما به تحلیل نیاز داریم، پس مدل رگرسیون لجستیک میاد می‌گه متن به دسته 1 تعلق داره یا 0؟.. اینجاست که ما متن یه عدد رو نسبت می‌دیم، مثلا می‌گیم متن به اندازه 0.87 مثبته؛ پس چون به متن > 0.5 هست، مثبته. حالا در ادامه بیشتر در مورد اینا صحبت می‌کنیم.

بعد از آموزش مدل، خود مدل و ابزار تبدیل متن رو ذخیره می‌کنیم تا بتونیم بعدها بدون نیاز به آموزش دوباره، متن‌های جدید رو پیش‌بینی کنیم. یعنی ما یک‌بار دیتاست رو به مدل می‌دیم، اون رو آموزش می‌دیم و بعدا دیگه فقط از مدل استفاده می‌کنیم. چون حجم دیتاها زیاده و آموزش هرباره، هم به زمان و هم به فضا نیاز داره و اصلا هم کار حرفه‌ای نیست.

ساختار پروژه

فولدر data/

این بخش شامل داده‌های اصلی پروژه هست. دو فولدر جداگونه برای نظرات مثبت و منفی وجود دارن. جمعا 25 هزار متن کوتاه درباره فیلم‌ها در این قسمت قرار داده شده (12,500 متن مثبت و 12,500 متن منفی) و مدل با استفاده از همین داده‌ها آموزش می‌بینه.

neg/ ← شامل متن‌های منفی

pos/ ← شامل متن‌های مثبت

فایل model.pkl

توی این فایل، مدل آموزش‌داده شده ذخیره شده.

این مدل همون چیزیه که بعد از آموزش می‌تونه تشخیص بده که متن مثبته یا منفی.

فایل vectorizer.pkl

توی این فایل، TF-IDF vectorizer ذخیره شده .

این ابزار وظیفه داره متن خام رو به یه بردار عددی تبدیل کنه تا مدل اون رو بفهمه. بدون این فایل، مدل نمی‌تونه متن جدید رو تحلیل کنه.

فولدر src/

توی این فولدر تمام فایل‌های مربوط به منطق برنامه وجود دارن. هر فایل یه وظیفه مشخص داره.

- فایل load_data.py

  وظیفه این فایل خوندن داده‌ها از فولدر `data` و برچسب‌گذاری آنهاست.

  به زبان ساده، این فایل متن‌ها رو می‌خونه و مشخص می‌کنه که کدومشون مثبت و کدومشون منفی هستن.

- فایل vectorize.py

  توی این فایل متون خام به مقادیر عددی تبدیل می‌شن.

این مرحله شامل ساخت TF-IDF vectorizer و تقسیم داده‌ها به train و test هست

یعنی این فایل متن رو به چیزی که مدل بتونه پردازش کنه تبدیل می‌کنه.

- فایل train_model.py

این فایل مدل ماشین لرنینگ رو آموزش می‌ده.

مدلی که استفاده شده Logistic Regression هست.

اینجا مدل با استفاده از داده‌ها یاد می‌گیره که چطور احساس یک متن رو تشخیص بده.

- save_model.py

توی این فایل مدل آموزش‌داده‌شده و vectorizer توی دو فایل جداگونه ذخیره می‌شن (`model.pkl` و `vectorizer.pkl`).

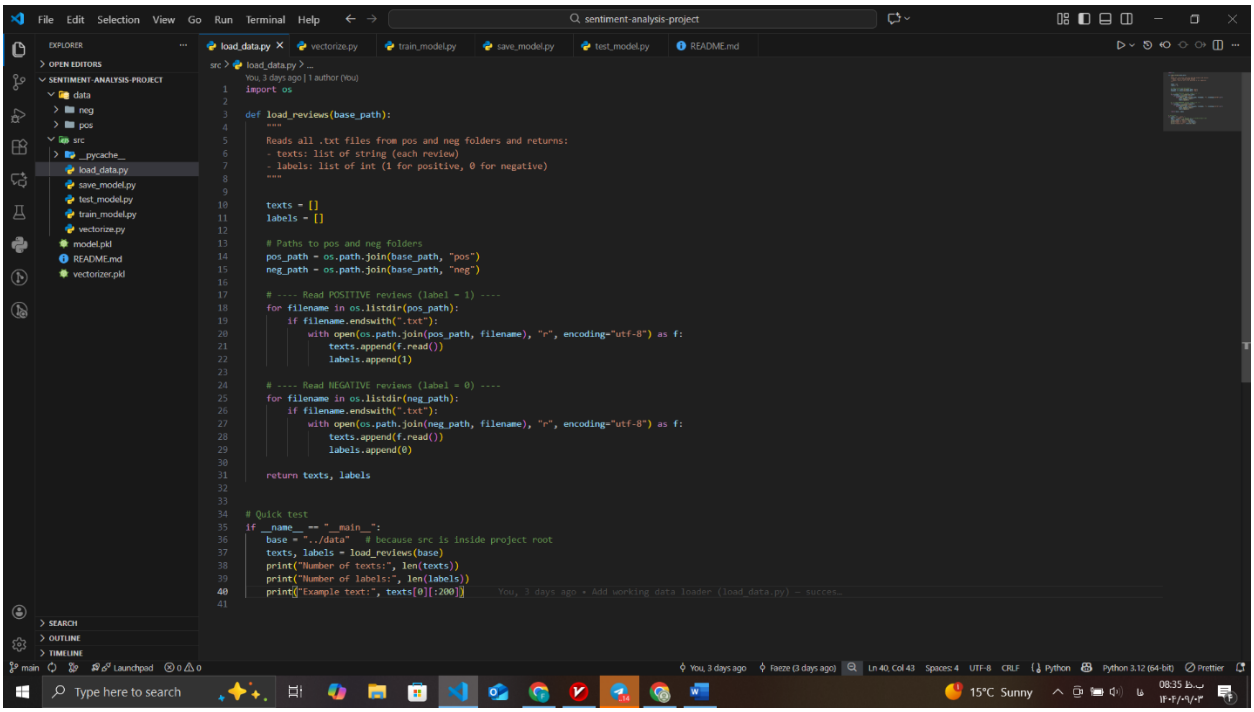
این کار باعث می‌شه که دفعه بعد نیاز نباشه مدل دوباره آموزش داده بشه.

- test_model.py

این فایل یه برنامه تعاملی ساده‌ست؛ برای اینکه ما مدلمون رو تست کنیم که درست کار می‌کنه یا نه. یعنی این فایل نتیجه نهایی پروژه رو به ما نشان می‌ده.

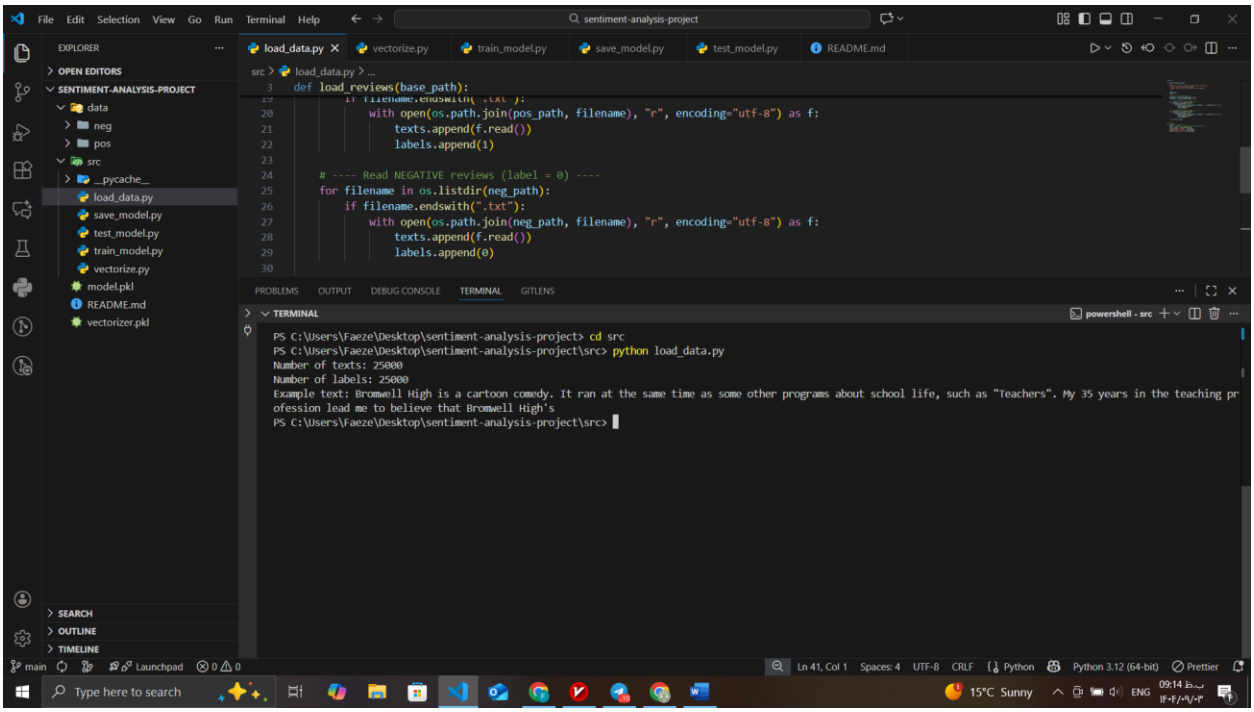
به این صورت که کاربر یه جمله وارد می‌کنه و مدل پیش‌بینی می‌کنه که جمله مثبت هست یا منفی.

مرحله اول: فایل load_data.py



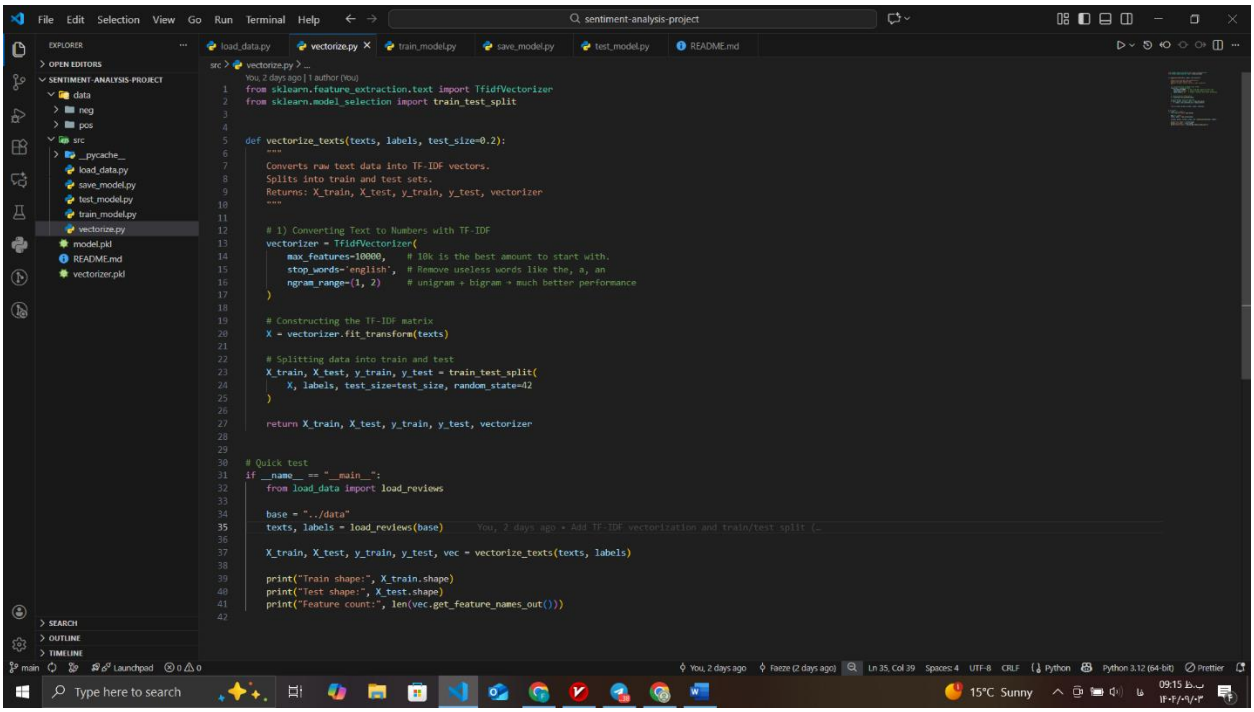
توی این کد، ابتدا کتابخونه استاندارد پایتون به نام os وارد شده که برای کار با مسیرها و فایل‌ها کاربرد داره. سپس تابعی به نام `load_reviews` تعریف شده که وظیفه‌اش خوندن همه فایل‌های متنی با پسوند `.txt` از دو فولدر `pos` و `neg` هست؛ این دو فولدر شامل نظرات مثبت و منفی هستن. ورودی این تابع مسیر پایه‌ای هست که این دو فولدر داخل اون قرار دارن. درون تابع، دو لیست خالی به نام‌های `texts` و `labels` ساخته شدن تا متن‌ها و برچسب‌هاشون (مثبت یا منفی بودن) توی اون‌ها ذخیره شدن. برای اینکه مسیر دقیق به فولدرهای `pos` و `neg` ساخته بشه، از تابع `os.path.join` استفاده شده تا کد مستقل از سیستم عامل و قابل اجرا روی ویندوز یا لینوکس باشه. بعد، با کمک تابع `os.listdir` لیست همه فایل‌های داخل پوشه مثبت خونده می‌شن و هر فایل با پسوند `.txt` باز می‌شه؛ متن داخل فایل با انکودینگ UTF-8 خونده شده و به لیست `texts` اضافه می‌شه و عدد 1 به عنوان برچسب مثبت به لیست `labels` افزوده می‌شه. همین روند برای پوشه نظرات منفی هم تکرار شده با این تفاوت که برچسب صفر (نمایانگر نظر منفی) به لیست برچسب‌ها اضافه می‌شه. در نهایت، دو لیست `texts` و `labels` که به ترتیب شامل تمام متن‌ها و برچسب‌های اون‌ها هستن، برگردونده می‌شن. در انتهای کد یک بخش تست سریع نوشته شده که در صورت اجرای مستقیم فایل، مسیر داده‌ها به صورت نسبی تنظیم می‌شه، سپس تابع `load_reviews` فراخوانی شده و تعداد کل متون و برچسب‌ها چاپ می‌شه، همچنین نمونه‌ای از متن اولین نظر هم برای اطمینان از صحت خوندن داده‌ها نمایش داده می‌شه. به طور کلی این کد وظیفه آماده‌سازی داده‌های اولیه پروژه رو بر عهده داره تا بعدا بتونیم اون‌ها رو برای آموزش مدل استفاده کنیم.

خروجی فایل load_data.py



خروجی به ما نشون می‌ده که تعداد متن‌ها و برچسب‌ها چقدره و این نتیجه لود شدن کامل دیتاست ماست. یه نمونه تکست هم به عنوان مثال نشون می‌ده که بارگذاری کامل رو تایید کنه.

مرحله دوم: vectorize.py



```python
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split


def vectorize_texts(texts, labels, test_size=0.2):
    """
    Converts raw text data into TF-IDF vectors.
    Splits into train and test sets.
    Returns: X_train, X_test, y_train, y_test, vectorizer
    """

    # 1) Converting Text to Numbers with TF-IDF
    vectorizer = TfidfVectorizer(
        max_features=10000,    # 10k is the best amount to start with.
        stop_words='english',  # Remove useless words like the, a, an
        ngram_range=(1, 2)     # unigram + bigram = much better performance
    )

    # Constructing the TF-IDF matrix
    X = vectorizer.fit_transform(texts)

    # Splitting data into train and test
    X_train, X_test, y_train, y_test = train_test_split(
        X, labels, test_size=test_size, random_state=42
    )

    return X_train, X_test, y_train, y_test, vectorizer


# Quick test
if __name__ == "__main__":
    from load_data import load_reviews

    base = "../data"
    texts, labels = load_reviews(base)

    X_train, X_test, y_train, y_test, vec = vectorize_texts(texts, labels)

    print("Train shape:", X_train.shape)
    print("Test shape:", X_test.shape)
    print("Feature count:", len(vec.get_feature_names_out()))
```

این کد وظیفه داره که متن‌های خام رو به شکل بردارهای عددی TF-IDF تبدیل کنه تا مدل‌های یادگیری ماشین بتونن اون‌ها رو پردازش کنن. برای این کار از کلاس `TfidfVectorizer` استفاده شده که ویژگی‌هایی مثل حذف کلمات اضافه مثل a و the ... (stop words)، استفاده از unigram و bigram، و محدود کردن تعداد ویژگی‌ها به 10 هزار تا رو داره تا کیفیت و سرعت کار بهینه باشه. بعد از ساخت ماتریس TF-IDF، داده‌ها به دو بخش آموزش و تست تقسیم میشن تا مدل بتونه روی آموزش یاد بگیره و روی تست ارزیابی بشه.

Unigram چیست؟

Unigram یعنی وقتی متن رو به کلمات جداگونه تقسیم می‌کنیم، هر کلمه به تنهایی یک واحد (Token) محسوب می‌شه.

مثلا جمله "I love movies" رو وقتی با Unigram نگاه کنیم، سه کلمه داریم: "I"، "love" و "movies". پس هر کلمه به صورت جداگونه بررسی می‌شه و ویژگی‌ها براساس همین تک‌کلمه‌ها ساخته می‌شن.

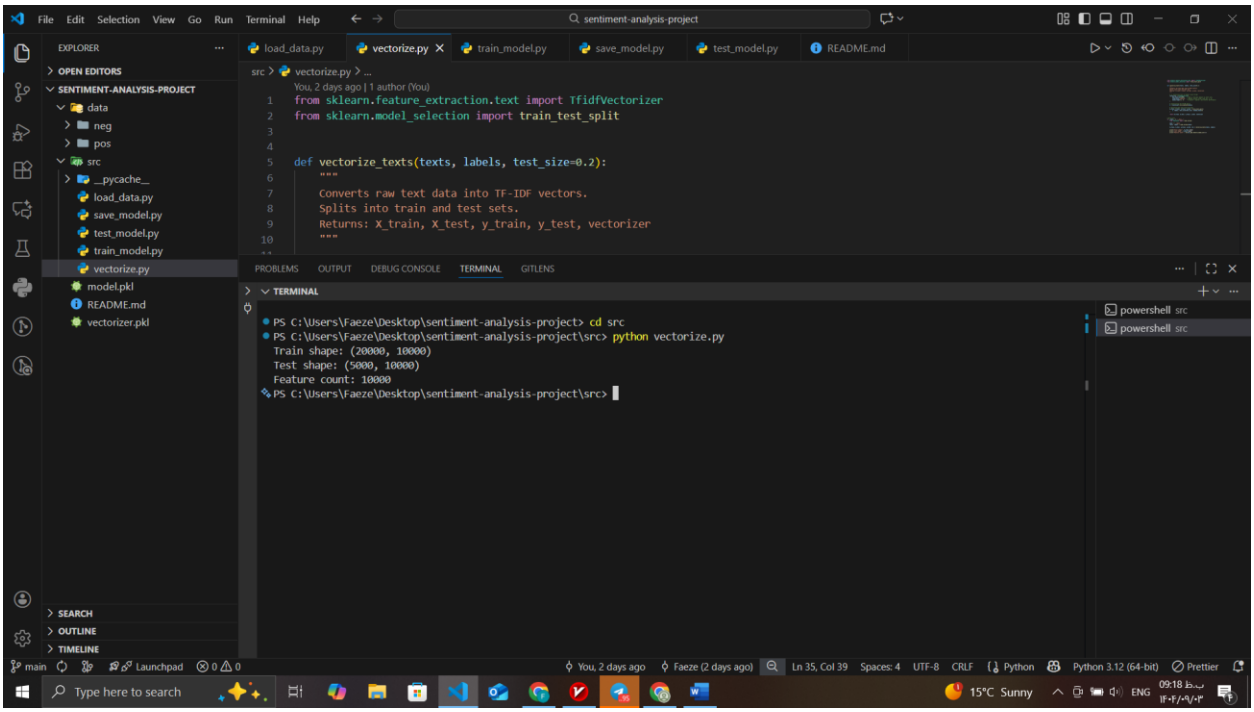
Bigram چیست؟

Bigram یعنی به جای نگاه کردن فقط به تک‌کلمه‌ها، به جفت‌های متوالی کلمات نگاه می‌کنیم.

مثلا همون جمله "I love movies" رو با Bigram به این صورت تقسیم می‌کنیم: "I love" و "love movies". در واقع، ترکیب دو کلمه پشت سر هم رو به عنوان یه واحد در نظر می‌گیریم.

حالا چرا از Bigram استفاده می‌شه؟ چون گاهی معنای یک کلمه به تنهایی کافی نیست و ترکیب دو کلمه مفهوم دقیق‌تری به ما میدن. مثلا توی جمله "not good"، اگه فقط به Unigram نگاه کنیم، کلمه "good" بار مثبت دارد. ولی "not good" ترکیبیه که معنی منفی می‌ده. Bigram کمک می‌کنه مدل این ترکیب‌ها رو بشناسه و نتیجه بهتری به ما بده.

خروجی فایل vectorize.py



خروجی تابع پنج تا متغیر هست: بردارهای آموزش و تست (X_train, X_test)، برچسب‌های مربوط به هر کدوم (y_train, y_test) و خود شیء vectorizer که برای تبدیل متن‌های جدید به بردارهای TF-IDF لازم هست. به طور خلاصه، این کد متون رو آماده و تقسیم‌بندی می‌کنه تا برای آموزش و ارزیابی مدل آماده باشن.

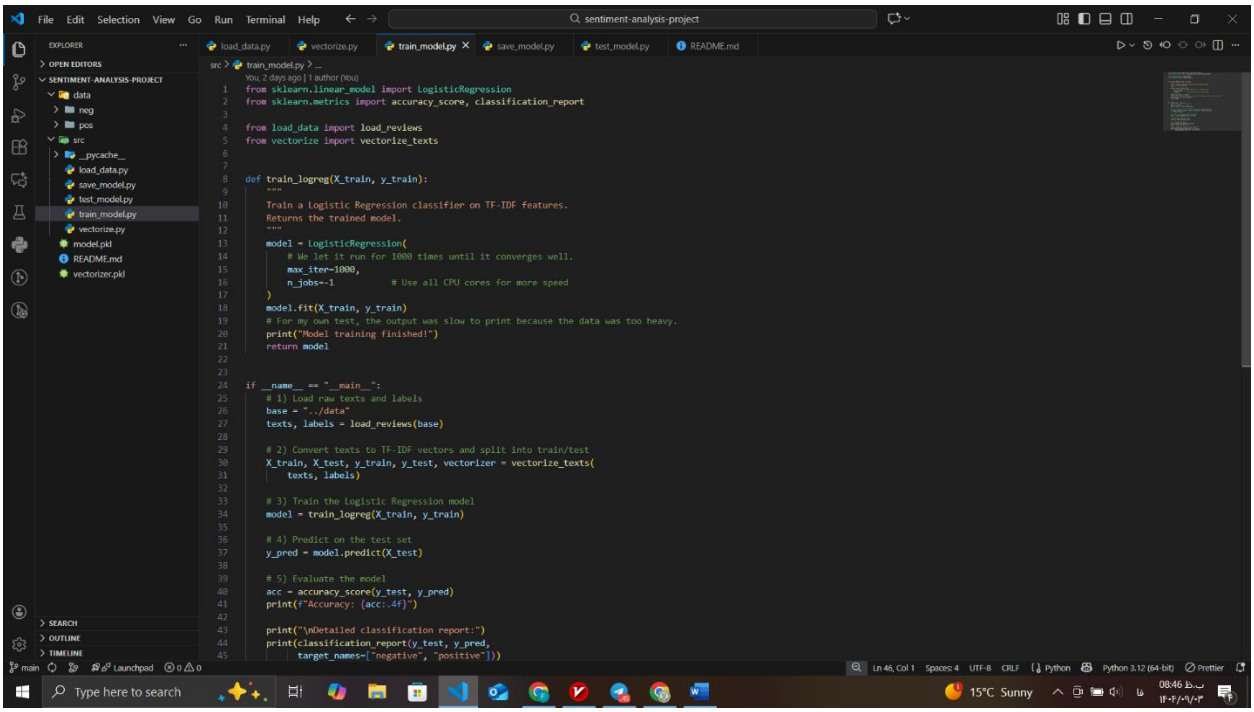
X_train : داده‌های آموزش.

X_test : داده‌های تست.

y_train : برچسب‌های آموزش.

y_test : برچسب‌های تست.

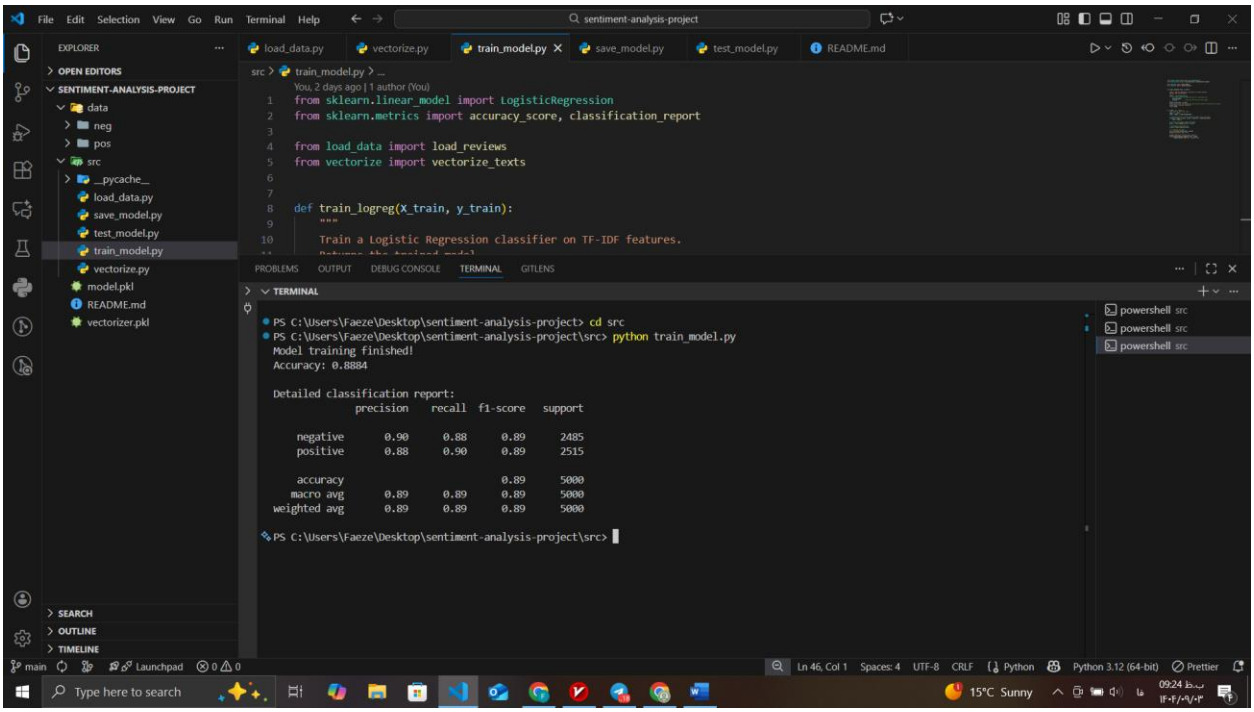
مرحله سوم: فایل train_model.py



```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

from load_data import load_reviews
from vectorize import vectorize_texts


def train_logreg(X_train, y_train):
    """
    Train a Logistic Regression classifier on TF-IDF features.
    Returns the trained model.
    """
    model = LogisticRegression(
        # We let it run for 1000 times until it converges well.
        max_iter=1000,
        n_jobs=-1          # Use all CPU cores for more speed
    )
    model.fit(X_train, y_train)
    # For my own test, the output was slow to print because the data was too heavy.
    print("Model training finished!")
    return model


if __name__ == "__main__":
    # 1) Load raw texts and labels
    base = "../data"
    texts, labels = load_reviews(base)

    # 2) Convert texts to TF-IDF vectors and split into train/test
    X_train, X_test, y_train, y_test, vectorizer = vectorize_texts(
        texts, labels)

    # 3) Train the Logistic Regression model
    model = train_logreg(X_train, y_train)

    # 4) Predict on the test set
    y_pred = model.predict(X_test)

    # 5) Evaluate the model
    acc = accuracy_score(y_test, y_pred)
    print(f"Accuracy: {acc:.4f}")

    print("\nDetailed classification report:")
    print(classification_report(y_test, y_pred,
          target_names=["negative", "positive"]))
```

این کد برای آموزش یک مدل Logistic Regression استفاده میشه که روی ویژگی‌های TF-IDF کار می‌کنه. اول با استفاده از تابع `load_reviews` داده‌های متنی و برچسب‌ها رو از پوشه داده‌ها می‌خونیم. بعدش متن‌ها رو با تابع `vectorize_texts` تبدیل به بردارهای عددی TF-IDF می‌کنیم و داده‌ها رو به مجموعه آموزش و تست تقسیم می‌کنیم. توی تابع `train_logreg` یک مدل Logistic Regression ساخته شده که اجازه می‌ده تا 1000 بار روی داده‌ها تکرار کنه تا به بهترین حالت برسه و از تمام هسته‌های پردازنده برای سرعت بیشتر استفاده می‌کنه. مدل روی داده‌های آموزشی آموزش داده میشه و پس از اتمام، پیام " Model training finished!" نمایش داده میشه. سپس مدل روی داده‌های تست پیش‌بینی انجام میده و دقت (accuracy) مدل محاسبه و چاپ می‌شه. در نهایت گزارش جزئی‌تری از عملکرد مدل با استفاده از `classification_report` نمایش داده میشه که نشون میده مدل چقدر در تشخیص درست نظرات مثبت و منفی موفق بوده. این کد به طور کلی مراحل اصلی آموزش، تست و ارزیابی مدل رو به صورت ساده و شفاف انجام میده.

خروجی فایل train_model.py



خروجی فایل `train_model.py` در اصل مدل آموزش‌داده شده Logistic Regression هست که آماده استفاده برای پیش‌بینی احساسات متون جدیده. علاوه بر این، هنگام اجرای فایل، دقت مدل روی داده‌های تست چاپ می‌شه و گزارشی کامل از عملکرد مدل شامل معیارهایی مثل دقت، بازیابی و F1-score برای هر کلاس (مثبت و منفی) نمایش داده می‌شه. این خروجی‌ها به ما کمک می‌کنه که بفهمیم مدل چقدر خوب کار می‌کنه و چقدر قابل اعتماد هست.

Precision (دقت):

می‌گه از بین تمام نمونه‌هایی که مدل گفته «مثبت» یا «یک» هستن، چند درصد واقعا درست پیش‌بینی شدن. یعنی وقتی مدل میگه «مثبت»، چقدر مطمئنیم که واقعا مثبت باشه.

Recall (بازخوانی یا حساسیت):

می‌گه از بین همه نمونه‌های واقعی مثبت، مدل چقدر تونسته درست تشخیص بده. یعنی مدل چقدر خوب نمونه‌های مثبت واقعی رو پیدا کرده.

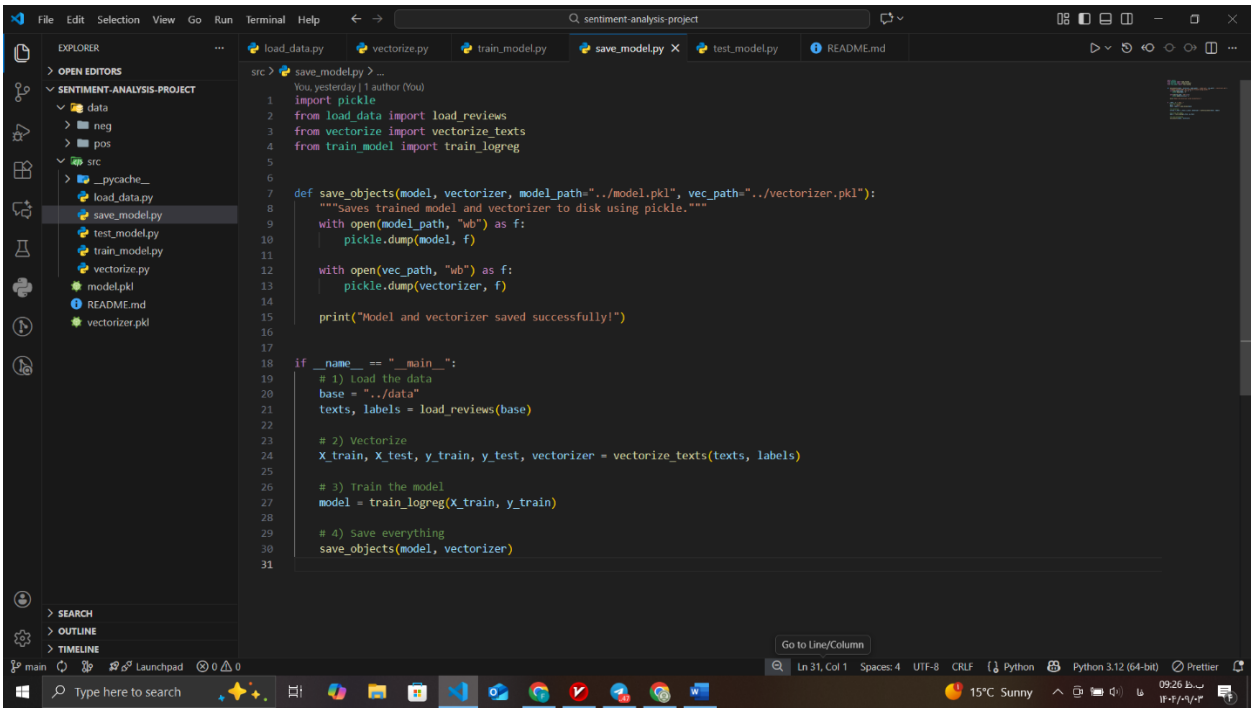
F1-score (امتیاز F1):

میانگینی بین Precision و Recall هست که تعادل بین این دو رو نشون می‌ده. وقتی هم دقت بالا باشه و هم بازخوانی بالا، F1 هم بالا میره.

Support (حمایت):

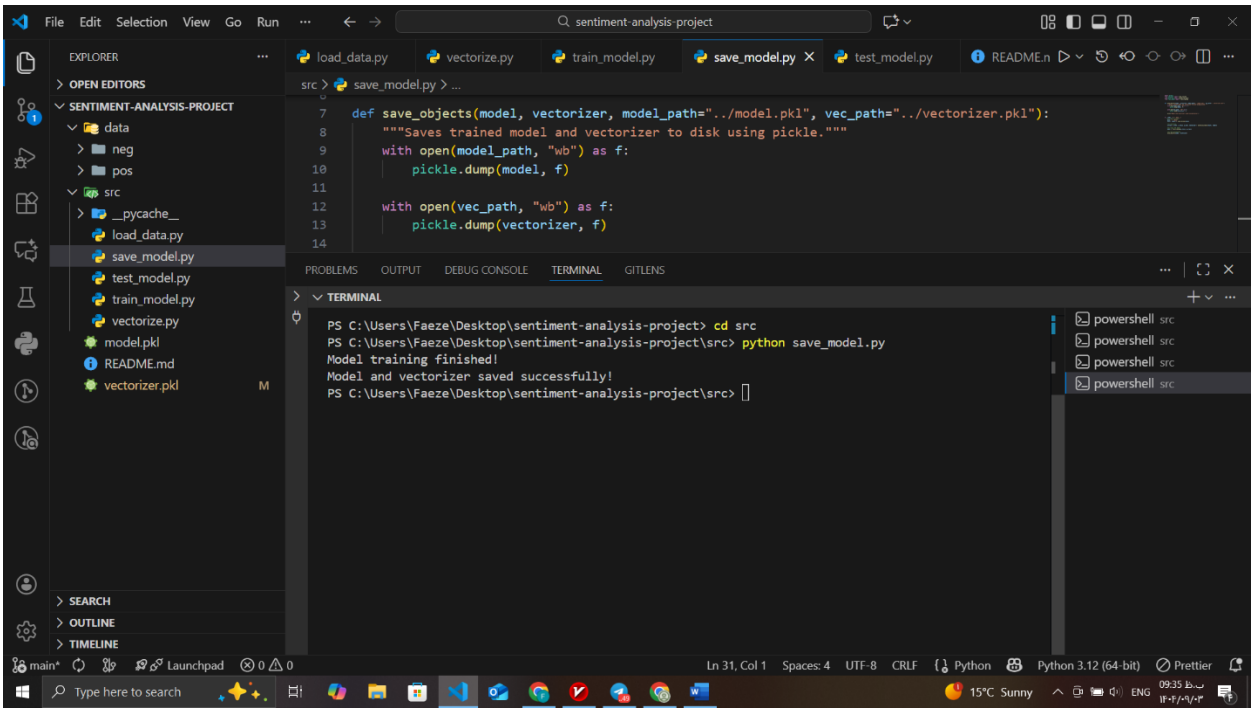
تعداد واقعی نمونه‌هایی که هر کلاس داره. یعنی مثلا چند داده واقعی مثبت داریم و چند داده واقعی منفی. این عدد فقط برای اطلاع هست و معیار عملکرد نیست.

مرحله چهارم: فایل save_model.py



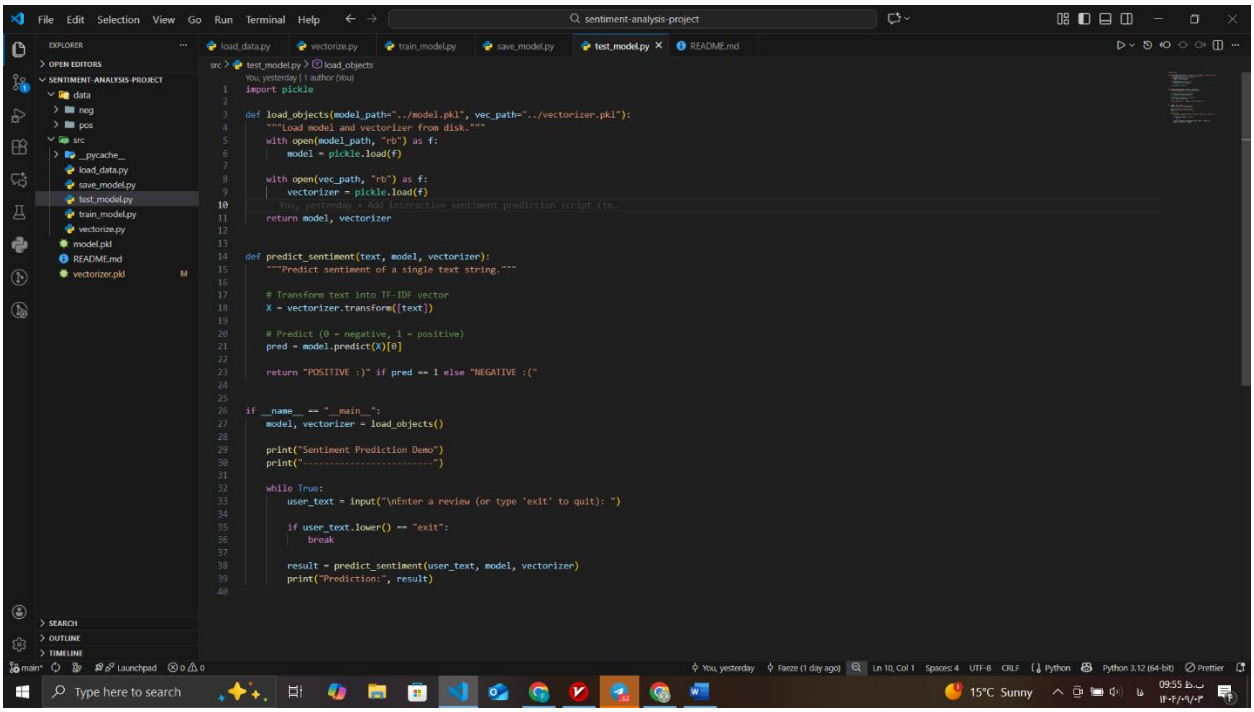
این کد یه پروژه ساده برای آموزش مدل تشخیص احساسات متون است که مراحل اصلی رو از بارگذاری داده‌ها تا ذخیره مدل نهایی پوشش می‌ده. ابتدا با استفاده از تابع `load_reviews`، متن‌های خام و برچسب‌های مربوط به اون‌ها (مثبت یا منفی) از پوشه داده‌ها خونده میشن. سپس این متن‌ها به کمک تابع `vectorize_texts` به بردارهای عددی TF-IDF تبدیل می‌شن و داده‌ها به دو بخش آموزش و تست تقسیم می‌شن تا مدل بتونه روی داده‌های آموزشی یاد بگیره و روی داده‌های تست ارزیابی بشه. در مرحله بعد، مدل Logistic Regression با تابع `train_logreg` روی داده‌های آموزش آموزش داده می‌شه تا بتونه الگوهای موجود در داده‌ها رو یاد بگیره. در نهایت، برای اینکه نیاز نباشه هر بار برنامه اجرا بشه و مدل دوباره آموزش ببینه، مدل آموزش‌دیده و همچنین بردارساز TF-IDF به کمک کتابخانه `pickle` به صورت باینری در فایل‌های جداگانه ذخیره می‌شن (`model.pkl` برای مدل و `vectorizer.pkl` برای بردارساز). این کار باعث می‌شه که در دفعات بعدی فقط با بارگذاری این فایل‌ها، بتوان مدل رو بدون آموزش مجدد به کار گرفت.

خروجی فایل save_model.py



خروجی نهایی این برنامه دو فایل ذخیره شده روی دیسک هست که شامل مدل آموزش‌دیده و ابزار تبدیل متن به بردار عددی هست و پایه و اساس مراحل بعدی پیش‌بینی احساسات متن‌های جدید هست. به طور خلاصه، این کد از داده‌های خام شروع می‌کنه، مدل یادگیری ماشین رو آموزش می‌ده و اون رو برای استفاده در آینده ذخیره می‌کنه.

مرحله پنجم: فایل test-model.py



توی این فایل عملا مرحله «استفاده از مدل» انجام می‌شه؛ یعنی جایی که دیگر قرار نیست مدل رو آموزش بدیم، بلکه می‌خوایم مدل ذخیره‌شده رو بالا بیاریم و روی متن‌های جدید پیش‌بینی انجام بدیم. همه‌چیز از دو فایل `model.pkl` و `vectorizer.pkl` شروع می‌شه؛ این دو فایل همون مدل آموزش‌داده‌شده و بردارساز TF-IDF هستن که قبلا ذخیره‌شون کرده بودیم.

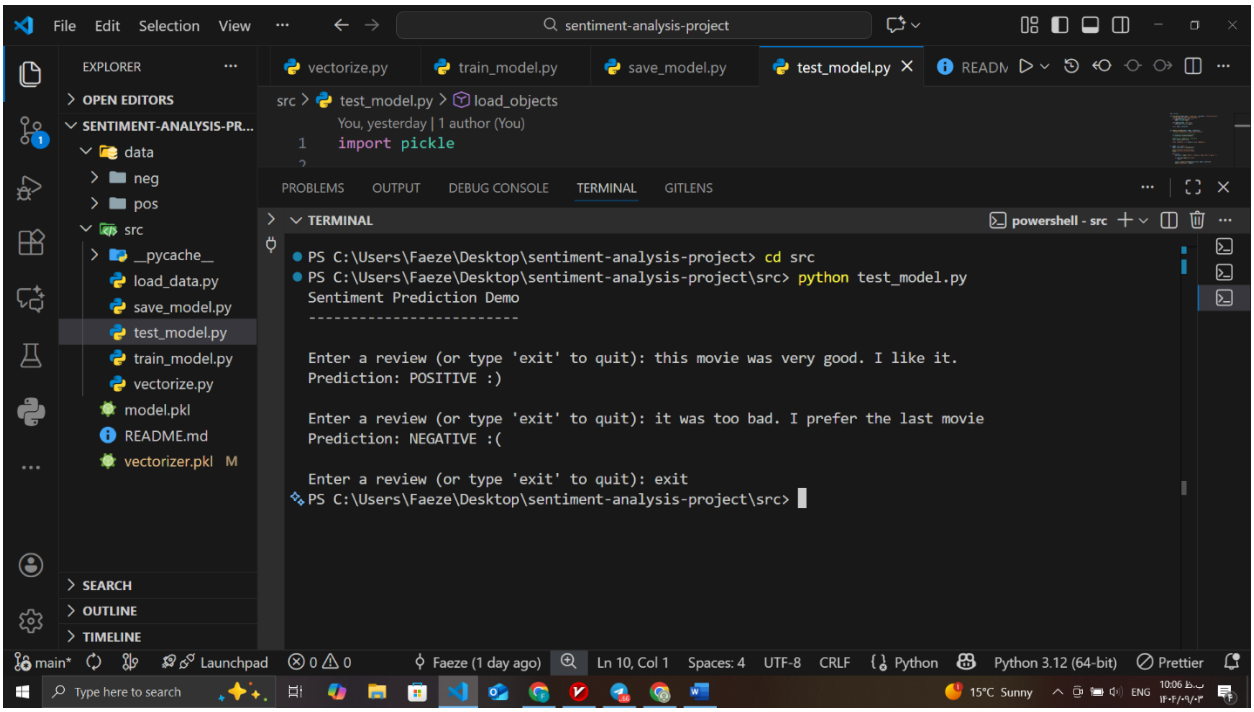
در ابتدای کد، تابع `load_objects` با استفاده از کتابخانه pickle کارش اینه که این دو فایل رو از دیسک باز کنه و دوباره تبدیلشون کنه به دو شیء قابل استفاده توی پایتون؛ یکی مدل و یکی vectorizer. بدون این مرحله، هر بار باید مدل رو از صفر آموزش می‌دادیم، اما با لود کردن، مستقیم می‌تونیم از اون استفاده کنیم.

بعد از این، تابع `predict_sentiment` نوشته شده که وظیفه‌اش گرفتن یه متن خام و تبدیل اون به یه بردار TF-IDF (با استفاده از همان vectorizer ذخیره‌شده) هست. سپس مدل روی این بردار پیش‌بینی انجام می‌ده و مشخص می‌کنه که این متن مثبته یا منفی. خروجی تابع به صورت یه رشته ساده نمایش داده می‌شه:

"POSITIVE :)" یا "NEGATIVE :("

توی بخش اصلی برنامه (`if __name__ == "__main__"`)، ابتدا مدل و بردارساز با `load_objects` لود می‌شن. سپس یک حلقه تعاملی شروع می‌شه که کاربر می‌تونه هر جمله‌ای تایپ کنه و برنامه فورا احساس متن رو پیش‌بینی و چاپ می‌کنه. اگر کاربر کلمه `exit` رو وارد کنه، برنامه متوقف می‌شه.

خروجی فایل test_model.py



خروجی اصلی این فایل برای متنی هست که کاربر وارد می‌کنه. یعنی هر بار کاربر یه جمله تایپ میکنه و برنامه بهش میگه مثبته یا منفی:

```

Prediction: POSITIVE(:

```

یا

```

Prediction: NEGATIVE):

```

یعنی خروجی نهایی این فایل، یک سیستم تعاملی تشخیص احساسات هست که از مدل آموزش‌داده‌شده استفاده می‌کنه و بدون نیاز به آموزش مجدد، روی هر متن دلخواه پیش‌بینی انجام می‌ده.

توضیحات کلی من راجع‌به این پروژه تموم شد، برای اطلاع بیشتر از روند پروژه و کدها، می‌تونین فایل Readme داخل ریپازیتوری گیت هابم رو بخونین.

من برای این پروژه از ویدئوهای مربوط به ماشین لرنینگ و پردازش زبان طبیعی یوتیوب، سایت w3schools و مدل‌های زبانی کمک گرفتم. شروع پروژه از روز جمعه 30 آبان ساعت 3 و نیم آغاز شد و الان که دارم گزارشم رو تکمیل می‌کنم، ساعت 10 و نیم شب روز 3 آذر هست. این فقط فرآیند برنامه نویسی بود و من از قبل این زمان هم برای درک دقیق پروژه، چرایی انجام اون و مواردی که لازم بود آموزش ببینم هم کلی تحقیق کردم. واقعیتش درک این پروژه و یاد گرفتن مسیر انجام اون، خیلی لذت‌بخش بود، چون حالا دیگه می‌دونم مبنای ماشین لرنینگ چیه و اصلا چجوری می‌شه یه ماشین رو آموزش داد.

بدون شک نتیجه نهایی قابل گسترش هست و من تمام سعیم رو به کار می‌گیرم که به محض یادگیری بیشتر، این پروژه رو هم توسعه بدم.