# گزارش پروژه یادگیری ماشین


## علیرضا امیری ۴۰۰۱۰۰۷۲۴


پکیج های استفاده شده:

```
import pandas as pd # for creating dataframes
import numpy as np # to handle arrays
import re # used in preprocessing for regex
import nltk # used for preprocessing text
from nltk.tokenize import word_tokenize #used for toekizing words
from nltk.corpus import stopwords # used to get english stopwords
from nltk.stem import WordNetLemmatizer # used to lemmatize words
from nltk.stem import PorterStemmer # used to stem words
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
from sklearn.model_selection import train_test_split, GridSearchCV #
used for splitting data and usuing gridsearch for hyperparameter
tuning
from sklearn.feature_extraction.text import CountVectorizer,
TfidfTransformer, TfidfVectorizer # used to generate vectors
from sklearn.metrics import classification_report,
ConfusionMatrixDisplay # used for reporting accuracy , f1score, ...
and displaying the cunfusion matrix
from sklearn.linear_model import LogisticRegression # using to
classify data
import matplotlib.pyplot as plt #used to plot the cunfusion matrix
!pip install fasttext
import fasttext #for using fasttext
```

مدل استفاده شده : Logistic Regression

مزایا:

تفسیرپذیری: رگرسیون لجستیک ضرایبی را تولید می کند که می تواند به عنوان اهمیت نسبی هر ویژگی در مدل تعبیر شود. این می تواند در درک کلمات یا ویژگی ها برای پیش بینی متغیر هدف مفید باشد.

سادگی: رگرسیون لجستیک یک الگوریتم ساده و سریع است که به راحتی قابل پیاده سازی و آموزش است. همچنین به منابع محاسباتی نسبتا کمی نیاز دارد، که آن را برای مجموعه داده های بزرگ مناسب می کند.

واریانس کم: رگرسیون لجستیک تمایل به واریانس کم دارد، به این معنی که کمتر مستعد برازش بیش از حد در مجموعه داده های کوچک است.

معایب:

مرز تصمیم خطی: رگرسیون لجستیک یک مرز تصمیم خطی بین کلاس ها را فرض می کند، که ممکن است همیشه در مسائل طبقه بندی متن که روابط بین کلمات می تواند پیچیده و غیرخطی باشد، صادق نباشد.

طبقات نامتعادل: رگرسیون لجستیک می تواند با طبقات نامتعادل مبارزه کند، جایی که یک طبقه بسیار رایج تر از دیگری است. این می تواند منجر به پیش بینی های مغرضانه نسبت به طبقه اکثریت شود.

# توضیح کد

در رویکرد اول ابتدا دیتا را میخوانیم . تعداد کلاس ها را بررسی میکنیم و میبینیم که دسته ها نامتعادل هستند و برای رفع این موضوع از over_sampling استفاده میکنیم و تعداد همه کلاس ها را برابر بزرگترین کلاس میکنیم. سپس به سراق پیش پردازش میرویم. در این مرحله ما تابع preprocess_test را تعریف میکنیم و ان را روس متن ها اعمال میکنیم.

```
def preprocess_text(text):
    text = re.sub(r'[^\w\s]', '', text)
    text = re.sub(r'\d+', 'NUM', text)
    text = re.sub(r'\s+', ' ', text)
    tokens = word_tokenize(text)
    stemmed_tokens =
[stemmer.stem(lemmatizer.lemmatize(token.lower())) for token in
tokens if token.lower() not in stop_words]
    preprocessed_text = ' '.join(stemmed_tokens)

    return preprocessed_text
```

در ادامه ما دیتا خود را به ۳ دسته train, test, val تقسیم میکنیم. سپس با استقاده از TfidfVectorizer ما دیتا خود را به بردار تبدیل میکنیم.
حال با استفاده از grid_search ابرپارامتر ها را بدست میاوریم و مدل خود را بر اساس ان میسازیم. در ادامه پس از اموزش روی داده train شروع به بررسی میکنیم. خروجی به صورت زیر میشود:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.76 | 0.85 | 0.80 | 7804 |
| 1 | 0.63 | 0.57 | 0.60 | 7818 |
| 2 | 0.66 | 0.66 | 0.66 | 7930 |
| 3 | 0.61 | 0.52 | 0.56 | 8075 |
| 4 | 0.73 | 0.83 | 0.78 | 7905 |
| accuracy | | | 0.68 | 39532 |
| macro avg | 0.68 | 0.69 | 0.68 | 39532 |
| weighted avg | 0.68 | 0.68 | 0.68 | 39532 |

ماتریس کانفیوژن هم در نوتبوک امده است.
در رویکرد دوم از fasttext استفاده میکنیم.بدین منظور اول داده اموزش را به فایل train.txt تبدیل میکنیم و مدل را روی ان اموزش میدهیم. سپس A را اینگونه میسازیم که هر متن را با استقاده از مدلمان به یک وکتور تبدیل کرده و در A میریزیم. b نیر همان sentiment ها است. سپس Aو b را به یک مدا لاجسیتیک دیگر که مانند قبل با استفاده از دادههای validation بدست امده میدهیم و این مدل را ارزیابی میکنیم. نتیجه به صورت زیر میشود:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.70 | 0.53 | 0.60 | 680 |
| 1 | 0.72 | 0.73 | 0.72 | 2680 |
| 2 | 0.83 | 0.88 | 0.85 | 7917 |
| 3 | 0.74 | 0.71 | 0.73 | 3295 |
| 4 | 0.76 | 0.59 | 0.67 | 933 |

| | | | | |
|---|---|---|---|---|
| accuracy | | | 0.78 | 15505 |
| macro avg | 0.75 | 0.69 | 0.71 | 15505 |
| weighted avg | 0.78 | 0.78 | 0.78 | 15505 |

در اینجا میبینم که نتیجه بهتری از رویکرد اول میگیریم. دلیل ان به شرح زیر است:
اول، FastText یک مدل مبتنی بر شبکه عصبی برای جاسازی کلمات است که می تواند روابط پیچیده تری بین کلمات و زمینه آنها در مقایسه با رویکردهای سنتی مانند TF-IDF ثبت کند.

دوم، پیش پردازش داده های متنی با دست می تواند زمان بر و مستعد خطا باشد. شناسایی همه ویژگی‌های مرتبط در داده‌های متنی ممکن است دشوار باشد و از دست دادن اطلاعات مهم آسان است. از سوی دیگر، FastText می‌تواند به‌طور خودکار ویژگی‌های مرتبط را از داده‌های متنی بیاموزد و جاسازی‌هایی ایجاد کند که الگوها و روابط زیربنایی در داده‌ها را ثبت کند.

سوم، FastText می تواند کلمات خارج از داده را بهتر از روش های سنتی مانند TF-IDF مدیریت کند.

## مدل های شکست خورده:

مدل های دیگری که برای این پروژه استفاده شد . نتیجه مطلوبی نگرفت شامل Naive Bayes و SVM است. در بیز دقت حدود ۵۰٪ میشد که زیاد مطلوب نبود و SVM باعث کرش کردن میشد و استفاده از ان غیر ممکن شد. دلایل نامطلوب بودن بیز میتواند این ها باشد:

کلمات نادر: Naive Bayes همه کلمات را به یک اندازه مهم می داند و فراوانی یا نادر بودن هر کلمه را در نظر نمی گیرد. این می تواند در مواردی که کلمات خاص کمیاب هستند اما بسیار آموزنده هستند و می تواند به طور قابل توجهی بر تصمیم طبقه بندی تأثیر بگذارد، مشکل ساز باشد.

برازش بیش از حد: Naive Bayes می تواند مستعد overfitting باشد.

کلمات خارج از داده اموزش: Naive Bayes نمی تواند کلمات خارج از واژگان (OOV) را مدیریت کند، زیرا به واژگان ثابتی از کلمات که در داده های آموزشی وجود دارد متکی است.

دلیل شکست svm نیز میتواند این باشد که از لحاظ زمانی بسیار پر هزینه و وقت گیر است.