به نام خدا

پروژه دوم

عاطفه نادری

مراحل کار به صورت زیر است:

1. وارد کردن کتابخانه ها
2. تمیزکردن داده
3. انتخاب یک روش برای تبدیل کلمات به اعداد
4. آموزش مدل
5. ارزیابی
6. پیش بینی

1 . وارد کردن کتابخانه ها:

همانطور که در تصویر مشخص است یه سری کتابخونه رو میایم import میکنیم بستگی به کارهایی که میخوایم انجام بدیم.

Imports Libraries

```
[5]:
```

```python
import pandas as pd
import numpy as np
import re
import seaborn as sns
from nltk.tokenize import TweetTokenizer
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize
from collections import Counter
import nltk
from nltk.tokenize import word_tokenize
from wordcloud import WordCloud
from sklearn.model_selection import train_test_split
from gensim.models import Word2Vec
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import matplotlib.pyplot as plt
from sklearn.ensemble import AdaBoostClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report
from nltk.tokenize import RegexpTokenizer
from nltk.stem.porter import PorterStemmer
from nltk.stem import WordNetLemmatizer
from tqdm import tqdm
nltk.download('punkt')
# nltk
import nltk
from nltk.corpus import stopwords
from  nltk.stem import SnowballStemmer
nltk.download('stopwords')
from sklearn.utils.class_weight import compute_class_weight
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import precision_score, recall_score, f1_score, confusion_matrix
from sklearn.tree import DecisionTreeClassifier
import gensim
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Atefe\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Atefe\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

قبل از هر چیز ابتدا ستون هایی که برای ما نیاز نیست را حذف کردیم و فقط ۳ تا ستون موندند.

['Tweet ID','sentiment','Tweet content']

برچسب ها یا همون احساسات رو هم به عددی بین ۰ تا ۳ تبدیل کردیم. و با استفاده از نمودار میله ای یک نمایشی از داده ها بر اساس برچسبشان نیز داشتیم:

convert categorical to numeric(Sentiment)

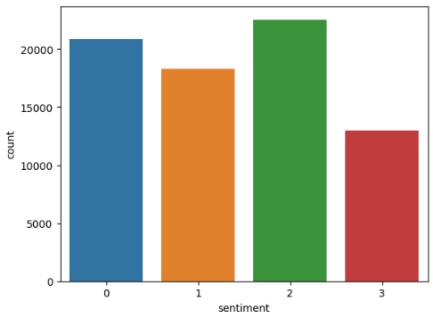
```
In [14]: training.sentiment.unique()
Out[14]: array(['Positive', 'Neutral', 'Negative', 'Irrelevant'], dtype=object)

In [15]: training['sentiment'].replace(['Positive', 'Neutral', 'Negative', 'Irrelevant'],
                                       [0, 1, 2, 3], inplace=True)
         test['sentiment'].replace(['Positive', 'Neutral', 'Negative', 'Irrelevant'],
                                   [0, 1, 2, 3], inplace=True)
         validation['sentiment'].replace(['Positive', 'Neutral', 'Negative', 'Irrelevant'],
                                         [0, 1, 2, 3], inplace=True)
```

Data visualization

Distribution of target class

```
In [16]: sns.countplot(x='sentiment',data=training)
Out[16]: <Axes: xlabel='sentiment', ylabel='count'>
```



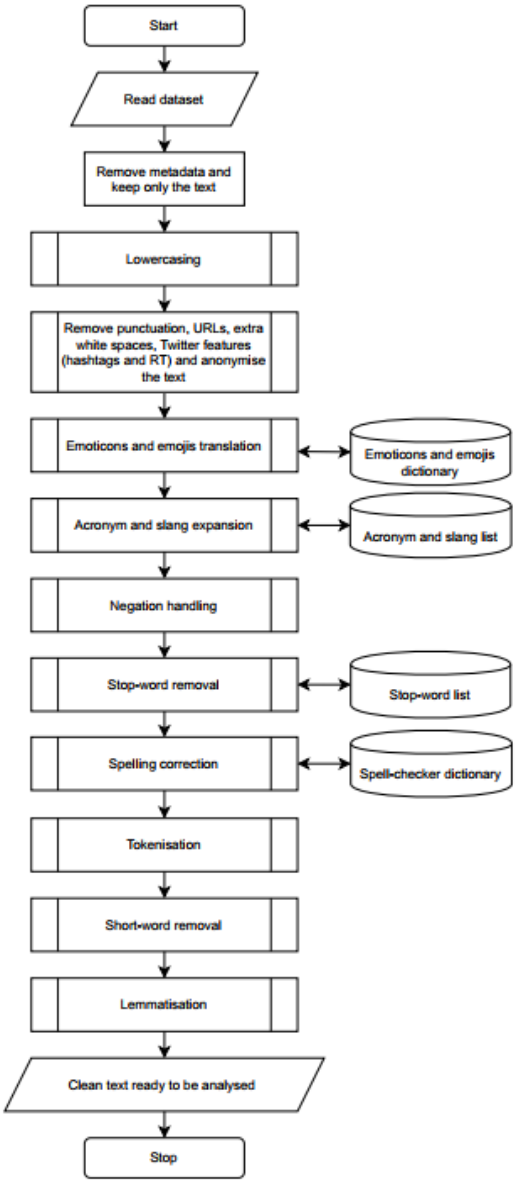
همانطور که در تصویر مشخص است، مقادیر داده ای که برچسب "نامرتبط" دارند کمتر است از بقیه و نوعی imbalance بودند داریم که برای اطمینان بیشتر اومدیم از class weight استفاده کردیم و وزن های هر کدوم رو به ما داد. تا در آموزش مدل ها از این خاصیت استفاده کنیم و به دقت بهتری برسیم.

*من یه دور با یه روش دیگه ای تست کردم و بنظرم حدود ۱۰ درصد دقت رو برد بالا. البته این ادعای قوی ای نیست چون باید چند بار تکرار بشه و هم چنین روش هم باید ثابت باشه.

2 . تمیز کردن داده ها:

در ادامه همانند شکل زیر که از مرجع [1] گرفته شده است، به ترتیب مراحل زیر را انجام دادیم:

تا جایی که میشد شبیه این فلوچارت رفتیم.


Start
Read dataset
Remove metadata and keep only the text
Lowercasing
Remove punctuation, URLs, extra white spaces, Twitter features (hashtags and RT) and anonymise the text
Emoticons and emojis translation
Emoticons and emojis dictionary
Acronym and slang expansion
Acronym and slang list
Negation handling
Stop-word removal
Stop-word list
Spelling correction
Spell-checker dictionary
Tokenisation
Short-word removal
Lemmatisation
Clean text ready to be analysed
Stop

طبق تصویر زیر، به حذف آدرس ها پرداختیم چون اثری روی تحلیل احساسات ندارند. هم چنین علایم نگارشی و اعداد و کاراکترها رو هم حذف کردیم.

Data Cleaning

```
]:
```

```python
def clean_tweets(tweet):
    # Check if the tweet is a non-null string
    if isinstance(tweet, str) and not pd.isnull(tweet):

        # remove URL
        tweet = re.sub(r"http\S+", "", tweet)
        # convert to lower case alpha
        tweet = tweet.lower()

        # Remove short links (e.g., buff.ly/2WmmiP5)
        tweet = re.sub(r'\b(?:buff\.ly|dlvr\.it)/\S+', "", tweet)

        # Remove usernames
        tweet = re.sub(r"@[^\s]+[\s]?", '', tweet)

        # remove special characters for example {!,?,...}
        tweet = re.sub('[^ a-zA-Z0-9]', '', tweet)

        # remove Numbers
        tweet = re.sub('[0-9]', '', tweet)

        return tweet
    else:
        # If the tweet is NaN or not a string, return it unchanged
        return tweet

# Apply the clean_tweets function to the 'Tweet content' column
training['Tweet content'] = training['Tweet content'].apply(clean_tweets)
validation['Tweet content'] = validation['Tweet content'].apply(clean_tweets)
test['Tweet content'] = test['Tweet content'].apply(clean_tweets)
```

توی دیتاست یه سری جاها بود که شامل ' ' بود، این ها رو با این تابع برداشتیم:

n [21]:
```
def remove_extra_spaces(df):
    df['Tweet content'] = df['Tweet content'].str.strip()
    for i in range(len(df)):
        if df['Tweet content'][i]== '':
            df['Tweet content'][i] = None
    training.dropna(subset=['Tweet content'], inplace=True)
    training.reset_index(drop=True,inplace=True)
    return df

training = remove_extra_spaces(training)
test = remove_extra_spaces(test)
validation = remove_extra_spaces(validation)
# training['Tweet content']=training['Tweet content'].str.s
```

```
C:\Users\Atefe\AppData\Local\Temp\ipykernel_14668\761347574.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['Tweet content'][i] = None
```

خب کلمات اضافی مثل حروف اضافه تاثیری ندارند و حذفشون کردیم:

6]:
```
# removing the stop words except "not"
stop_words = stopwords.words("english")
whitelist = ["n't", "not"]
filtered_list = [item for item in stop_words if item not in whitelist]
# Define a function to remove stop words
def remove_stop_words(word_list):
    tweet_without_stopwords = [word for word in
                               word_list.split()
                               if word.lower() not in filtered_list]
    joined_tokens = [" ".join(tweet_without_stopwords)]
#     return tweet_without_stopwords
    return joined_tokens
```

وقتی میخوایم مفهوم کلمات رو دربیاریم و یا در روش tf-idf کلمات رو به شکل ساده شدشون در میاریم:

Stemming

[30]:
```
#stem the words
ps = PorterStemmer()
# Define a function to stem the words
def stemming(word_list):
    tweet_with_stem = []
    split_list = [word for sentence in word_list for word in sentence.split()]
    for w in split_list:
        w = ps.stem(w)
        tweet_with_stem.append(w)
    return tweet_with_stem
```

[31]:
```
training['Tweet content'] = training['Tweet content'].apply(stemming)
test['Tweet content'] = test['Tweet content'].apply(stemming)
validation['Tweet content'] = validation['Tweet content'].apply(stemming)
```

3 . انتخاب روش:

در این پروژه از سه حالت:

- Tf-idf
- Glove
- Word2vec

استفاده کردم. که دو روش w2v و glove روش های word embedding هستند که به مفهوم و کلمات قبل و بعد نیز توجه میکند (بر خلاف tf-idf)

❖ روش glove

از حالت pretrain شده اش، استفاده کردم که از سایت دانلود میکنیم و یه فایل zip داره که توش باز glove در ابعاد مختلف ۵۰ و ۱۰۰ و ... داره که من ۱۰۰ رو انتخاب کردم.

Word embedding

Glove

In [103]:
```
import numpy as np

def load_glove_model(File):
    print("Loading Glove Model")
    glove_model = {}
    with open(File,'r',encoding="utf8") as f:
        for line in f:
            split_line = line.split()
            word = split_line[0]
            embedding = np.array(split_line[1:], dtype=np.float64)
            glove_model[word] = embedding
    print(f"{len(glove_model)} words loaded!")
    return glove_model
glove_model_path = 'glove.6B.100d.txt'  # path to your GLoVe file
glove_model = load_glove_model(glove_model_path)
```

```
Loading Glove Model
400000 words loaded!
```

In [212]:
```
def embedd(tweet):
    # If the tweet is a list, join its elements into a string
    if isinstance(tweet, list):
        tweet = ' '.join(map(str, tweet))

    # Tokenize the tweet
    tokens = word_tokenize(tweet.lower())

    # Filter valid words and get their embeddings
    embeddings = [glove_model[word] for word in tokens if word in glove_model]

    # Check if there are valid embeddings
    if embeddings:
        # Calculate the mean embedding (e.g., using np.mean)
        aggregated_embedding = np.mean(embeddings, axis=0)
        return aggregated_embedding
    else:
        # Return zeros if there are no valid embeddings
        return np.zeros(100)
```

بعد از اینکه مدلش رو load کردیم، باید هر tweet رو با این مدل embedd کنیم، جوری که میایم اول بررسی میکنیم کلمه موجود در tweet، آیا در مدل ما وجود دارد که وکتورش هم باشه یا خیر. اگر باشه میایم وکتورش رو از مدل بدست می‌آوریم و با توجه به ابعاد ۱۰۰ در آخر برای کل توییت یک فضای با طول ۱۰۰ ایجاد خواهد شد که شامل وکتور اون جمله است. اینجا از میانگین وکتورها برای اینکه در اخر همه ۱۰۰ باشند(تا جایی که فهمیدم) استفاده میشه.

❖ روش word2vec

به صورت کلی مشابه با glove است، ولی اینجا از مدل pretrain شده استفاده نکردم و با توجه به دیتاست خودم مدل رو ساختم.

Word embedding

W2V

```
[10]: 1 X_train = training['Tweet content']
      2 SIZE = 50
      3 model = gensim.models.Word2Vec(X_train
      4 , min_count=1
      5 , vector_size=SIZE
      6 , window=5
      7 , workers=4)
      8 model.wv.most_similar('hi', topn=3)

[10]: [('hello', 0.9060459733009338),
       ('hey', 0.8979060053825378),
       ('ghostrecon', 0.8853891491889954)]

[16]: 1 def compute_avg_w2v_vector(w2v_dict, tweet):
      2     list_of_word_vectors = [w2v_dict[w] for w in tweet if w in w2v_dict.key_to_index.keys()]
      3     if len(list_of_word_vectors) == 0:
      4         result = [0.0]*SIZE
      5     else:
      6         result = np.sum(list_of_word_vectors, axis=0) / len(list_of_word_vectors)
      7
      8     return result

[17]: 1 training['W2v'] = training['Tweet content'].apply(lambda x: compute_avg_w2v_vector(model.wv, x))
      2 test['W2v'] = test['Tweet content'].apply(lambda x: compute_avg_w2v_vector(model.wv, x))
      3 validation['W2v'] = validation['Tweet content'].apply(lambda x: compute_avg_w2v_vector(model.wv, x))
```

برای ساخت مدل علاوه بر کتابخونه، یه سری پارامتر هم باید ست کنیم مثل سایز پنجره. من این word2vec رو از راه دیگه ای هم رفتم. که خودم یک مدل از cbow ساختم و ایمبد کردم. که در انتهای فایل مشخص شده وجود دارد.
بعد از ساخت مدل به سراغ embed کردن رفتیم و اگر هم احیانا جایی بود که سایز صفر بود با مقدار ۰ ولی با توجه به ابعاد پرش میکنیم.(۵۰) شایان ذکر است که وقتی از word2vec استفاده میکینم، میانگین همه وکتورها رو در نظر میگیریم و به همون ابعاد مذکور که اینجا ۵۰ است میرسیم.

- روش TF-IDF

همانطور که در درس داشتیم و تمرین هم ازش حل کردیم بر اساس دو تا فرمول میره tf و idf رو حساب میکنه و نتیجه ضرب این دو تا رو با نام tf-idf ذخیره میکنه و اینطوری فضای هر tweet با توجه به بقیه tweet ها و کلماتی که دارند به اعداد برده میشه.

Vecotorization

TF-IDF

[78]:
```
def convert_TF_IDF(df):
    df['Tweet content'] = df['Tweet content'].astype(str)
    df['tf_idf'] = TfidfVectorizer(max_features=1000, dtype=np.float64).fit_transform(df['Tweet content']).todense().tolist(
    return df
```

[79]:
```
training = convert_TF_IDF(training)
test = convert_TF_IDF(test)
validation = convert_TF_IDF(test)
```

[80]:
```
validation
```

t[80]:

	Tweet ID	sentiment	Tweet content	tf_idf
0	3364	3	['mention', 'facebook', 'struggl', 'motiv', 'g...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
1	352	1	['bbc', 'news', 'amazon', 'boss', 'jeff', 'bez...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.3861389225206503, ...
2	8312	2	['pay', 'word', 'function', 'poorli', 'chromeb...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
3	4371	2	['csgo', 'matchmak', 'full', 'closet', 'hack',...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
4	4433	1	['presid', 'slap', 'american', 'face', 'realli...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.43097889205893375,...
...	...	...	...	...
495	8055	0	['special', 'shoutout', 'microsoft', 'excel']	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
496	6787	3	['dumb', 'lucki', 'fortnit', 'montag', 'youtub...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
497	3838	0	['dang', 'goe', 'birthday', 'present', 'mayb',...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
498	2008	3	['ab', 'fab', 'see', 'bungalow', 'built', 'wal...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
499	4096	1	['million', 'viewer', 'wtf', 'csgo', 'minecraf...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...

500 rows × 4 columns

4 . آموزش مدل:

از ۴ مدل:
Random Forest و لاجیستیک رگرشن و svm و AdaBoost استفاده شده است.

روی دو مدل word embedding اول random forest دقت خیلی خوبی داشته و نزدیک به ۹۰ درصد در برخی از معیارها و در باقی معیارها هم، وضعیت خوبی داشته است. و بعد از اون هم مدل لاجیستیک و svm تقریبا مثل هم عمل کردند.

روی TF-idf، کلا دقت خیلی پایین بود با وجود اینکه پروسس روی داده ها یکسان است. که این قدرت روش های word embd رو نشون میدهد. ولی در کل دقت اعدادی میان ۲۳ تا ۵۰ بوده است. و مدل‌های لاجیستیک رگرشن و svm بهتر عمل کردند.

5 . معیارهای ارزیابی:

precision، recall، Confusion Matrix و score f1 وaccuracy انتخاب شده است. چون ۵*۴*۳ حالت مختلف وجود دارد پس همه رو نمیتوان تحلیل کرد اما معیارهای مختلف از random forest رو word2vec را تحلیل مینمایم:

```
RF Precision: 0.8883871126353703
RF Recall: 0.8788929492206585
RF F1-Score: 0.8822629942684953
RF Confusion Matrix:
[[133   6   5   4]
 [  9 117   8   0]
 [  1   6 117   2]
 [  7   2   8  75]]
```

```
RF Accuracy: 0.884
RF Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.90      0.89       148
           1       0.89      0.87      0.88       134
           2       0.85      0.93      0.89       126
           3       0.93      0.82      0.87        92

    accuracy                           0.88       500
   macro avg       0.89      0.88      0.88       500
weighted avg       0.89      0.88      0.88       500
```

Precision:

- Precision measures the accuracy of the positive predictions made by the model.
- Class 0: 88.8%
- Class 1: 88.9%
- Class 2: 85%
- Class 3: 93%

Recall:

- Recall measures the ability of the model to capture all the relevant instances for each class.
- Class 0: 90%
- Class 1: 87%
- Class 2: 93%
- Class 3: 82%

F1-Score:

- F1-score is the harmonic mean of precision and recall. It provides a balance between precision and recall.
- The overall F1-score is 88.2%.

Confusion Matrix:

- The confusion matrix provides a detailed breakdown of correct and incorrect predictions for each class.
- The diagonal elements represent correct predictions, and off-diagonal elements represent misclassifications.

Accuracy:

- Overall accuracy is 88.4%, indicating the percentage of correctly classified instances.

به طور خلاصه، به نظر می رسد مدل random forest در چندین معیار عملکرد خوبی دارد. دقت، فراخوانی و امتیاز F1 خوبی را برای هر کلاس نشان می‌دهد که منجر به دقت کلی ۸۸.۴٪ می‌شود. به نظر می رسد این مدل به طور موثر به مجموعه داده داده شده generalized میشود.

و هم چنین مدل radnome forest تونست با استفاده از validation dataset به تعداد ۴۵۳ از ۵۰۰ تا را درست تخمین بزنه در صورتی که بقیه تقریبا تصف رو میتونند درست تخمین بزنند.

دلایل خوب بودن random forest:

توی این قسمت بنظرم ویژگی هایی از این مدا که توی دیتاست ودر مدل ها دیدم رو آوردم.

- اهمیت ویژگی (Feature Importanc):

Random Forest اهمیت هر ویژگی را در پیش بینی ها محاسبه می کند. ویژگی هایی که بیشتر به کاهش ناخالصی کمک می کنند (مثلاً ناخالصی Gini) اهمیت بیشتری دارند. این به مدل اجازه می دهد تا بر روی مرتبط ترین ویژگی ها تمرکز کند و به طور بالقوه تعمیم به داده های جدید و نادیده را بهبود بخشد

- robust نسبت به overfitting:

Random Forest تمایل دارد تا بیش از حد robust باشد، به خصوص زمانی که تعداد درختان در جنگل کنترل شود. تجمیع پیش‌بینی‌ها از چندین درخت به کاهش واریانس کمک می‌کند و مدل را کمتر مستعد تطبیق بیش از حد داده‌های آموزشی می‌کند.

- Hyperparameter Tuning:

Random Forest چندین هایپرپارامتر دارد که می توان آنها را برای بهینه سازی عملکرد تنظیم کرد. تنظیم صحیح هایپرپارامترها می تواند توانایی مدل را برای گرفتن الگوها در داده ها افزایش دهد.

- مقاومت در برابر داده های noisy:

Random Forestحتی در صورت وجود ویژگی های noisy یا irrelevant می تواند عملکرد خوبی داشته باشد. ماهیت مجموعه ای مدل به آن اجازه می دهد تا ویژگی های کمتر مهم را فیلتر کند.

دلایل خوب بودن Logistic Regression:

توی این قسمت بنظرم ویژگی هایی از این مدا که توی دیتاست ودر مدل ها دیدم رو آوردم.

- تفسیر پذیری:

رگرسیون لجستیک نتایج واضح و قابل تفسیری را ارائه می دهد. ضرایب مدل نشان دهنده تأثیر هر متغیر مستقل بر روی شانس نتایج است و تفسیر تأثیر ویژگی ها را آسان می کند.

- بدون فرض Feature Distribution:

رگرسیون لجستیک مفروضات قوی در مورد توزیع ویژگی ها ایجاد نمی کند. می‌تواند ترکیبی از ویژگی‌های عددی و دسته‌بندی را بدون نیاز به پیش پردازش گسترده مدیریت کند.

- Regularization:

رگرسیون لجستیک را می توان برای جلوگیری از برازش بیش از حد تنظیم کرد. روش های Regularization ، مانند منظم سازی L1 یا L2، به جلوگیری از حساسیت بیش از حد به داده های آموزشی کمک می کند و تعمیم به داده های جدید را بهبود می بخشد.

ب.

برای دو روش glove و w2v، بهترین مدلها بیشترین تخمین درست رو زدند شاید اونی که دقت بالاتری رو روی داده‌های تست نشون دادند، میتونیم بگیم روی داده های validation هم تونستد تخمین خوبی رو بزنند. در داخل کد همه اعداد رو چاپ کردم.لطفا ببینید از اونجا.ممنونم 😊 ولی برای روشtf-idf لزوما این نیست و اونی که بهترین جوابا رو داده بهترین دقت رو از تمام معیارها نداشته.

	RF	SVM	LR	AdaBoost	
Tf-idf	129	112	112	112	
Glove	457	208	208	208	
W2v	453	255	254	453	

ولی بصورت کلی RF برای glove و w2v بهترین نتایج رو داشته که به ترتیب: ۴۵۳ و ۴۵۷ تا از ۵۰۰ تا رو تونسته خوب تخمین بزنه. برای tf-idf هم بهترین نتیجه برابر با ۱۲۹ برای random forest بوده است که البته دقتی که نتیجه داده است ۲۳ درصد بوده که بهترین دقت نیست اما روی داده های validation بهترین تخمین را داده است.

[1] Palomino, M.A. and Aider, F., 2022. Evaluating the Effectiveness of Text Pre-Processing in Sentiment Analysis. *Applied Sciences*, *12*(17), p.8765.