دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)
دانشکده مهندسی برق

آنالیز احساسات متن

مقدمه ای بر هوش محاسباتی

نگارش
رومینا عالیشاه
عرفان پشته بان
معصومه محمدخانی

استاد راهنما
سرکار خانم دکتر فرزانه عبدالهی

ماه و سال
دی 1401

فهرست مطالب

صفحه

مقدمه

به فرایند پیدا کردن قوانین و الگوهای غیر بدیهی، جدید (از قبل نامشخص)، مخفی، احتمالاً مفید و کاربردی از انبوه داده‌های (پیکره) مستندات را متن کاوی (Text Mining) یا تحلیل متن (Text Analytics) می‌گویند. در تعریف دیگر، متن کاوی به فرایند تحلیل و اکتشاف انبوهی از متون غیرساخت‌یافته بوسیله نرم‌افزار به منظور شناسایی مفاهیم، الگوها، موضوعات، کلیدواژه‌ها و دیگر ویژگی‌های داده‌های متنی گفته می‌شود. به عبارت دیگر هدف متن کاوی، کشف معنا (مفهموم و هدف) و استخراج اطلاعات نهفته (برای مثال موجودیت‌ها و روابط) در داده‌های متنی است.

در اغلب زمینه‌های متن کاوی نیاز به پیش‌پردازش متن با استفاده از ابزارهای پردازش زبان طبیعی و سپس تبدیل داده‌های متنی به بردارهای عددی داریم. منظور از ابزارهای پردازش متن، کتابخانه‌هایی است که برای آماده‌سازی متن جهت متن کاوی و استخراج دانش از متن بکار می‌روند.

میتوان مراحل متن کاوی را به طور کلی به این شکل دسته بندی کرد:

1- انتخاب متن

2- پردازش متن

3- تبدیل متن به صفات خاصه

4- انتخاب صفات خاصه از متن

5- داده کاوی بر روی متن و کشف دانش از متن

6- تفسیر و ارزیابی خروجی متن کاوی

چکیده

آنالیز احساسات یا Sentiment Analysis نوعی متن کاوی است که در آن تمایلات در نظرات افراد (که در قالب متن در اختیار قرار گرفته اند) با استفاده ازNLP (پردازش زبان طبیعی)، تجزیه و تحلیل متن و زبانشناسی محاسباتی، مشخص میشوند.

هدف از انجام این پروژه، آنالیز احساسات روی داده های توییتر، کار با APIهای مختلف برای استخراج توییت و در نهایت پیاده سازی مدل آموزش داده شدهروی سخت افزار و تست آن در حالت بلادرنگ میباشد.

1-سطح مقدماتی

در این بخش سعی داریم ابتدا متن توئیت ها را تو سط روش های embedding پردازش کرده و یک مدل تحلیل احساسات را به کمک شبکه های عصبی طراحی و آموزش دهیم و در نهایت عملکرد و پیش بینی مدل را ارزیابی کنیم.

1-1-تجزیه و تحلیل داده(EDA)

ابتدا مجموعه داده را تجزیه و تحلیل میکنیم. دیتاست twitter_sentiment.csv شامل 13871 توئیت است که دارای 21 ویژگی نظیر ، چه کسی توئیت را نوشته، توئیت درباره چیست، اح سا سات و غیره، ا ست. این دیتا ست مربوط به دوره انتخابات آمریکا ا ست. به همین جهت پی شبینی میشود که توئیت های منفی بیشتر از توئیت های مثبت باشد. در واقع 61.2 درصد توئیت ها منفی، 16.1 در صد توئیت ها مثبت و 22.7 در صد توئیت ها خنثی ه ستند. با برر سی های فوق انتظار میرود دقت پی شبینی توئیت های منفی بی شتر از دقت پی شبینی توئیت های مثبت با شد.(چون داده بی شتری برای آموزش وجود دارد)

همچنین پس از اضافه کردن کتابخانه های مورد نیاز، دیتاست را به شکل زیر میخوانیم و با توجه به این که تنها دو ویژگی متن توئیت و احساسات آن را نیاز داریم فقط این بخش ها را نگه میداریم:

```
data = pd.read_csv('Sentiment.csv')
data = data[['text','sentiment']]
data
```

دیتا خوانده شده و مختصر شده به شکل زیر قابل مشاهده است:

	text	sentiment
0	RT @NancyLeeGrahn: How did everyone feel about...	Neutral
1	RT @ScottWalker: Didn't catch the full #GOPdeb...	Positive
2	RT @TJMShow: No mention of Tamir Rice and the ...	Neutral
3	RT @RobGeorge: That Carly Fiorina is trending ...	Positive
4	RT @DanScavino: #GOPDebate w/ @realDonaldTrump...	Positive
...	...	...
13866	RT @cappy_yarbrough: Love to see men who will ...	Negative
13867	RT @georgehenryw: Who thought Huckabee exceede...	Positive
13868	RT @Lrihendry: #TedCruz As President, I will a...	Positive
13869	RT @JRehling: #GOPDebate Donald Trump says tha...	Negative
13870	RT @Lrihendry: #TedCruz headed into the Presid...	Positive

1-2-پاکسازی داده(Cleaning data)

در این قسمت تلاش میکنیم داده ها را مرتب تر و ساده تر کنیم. برای مثال از لینک ها، علائم نگارشی، یا برخی کلمات مانند RT که به معنای Retweeting هست و تاثیری در تشخیص مدل ندارد، صرف نظر میکنیم.

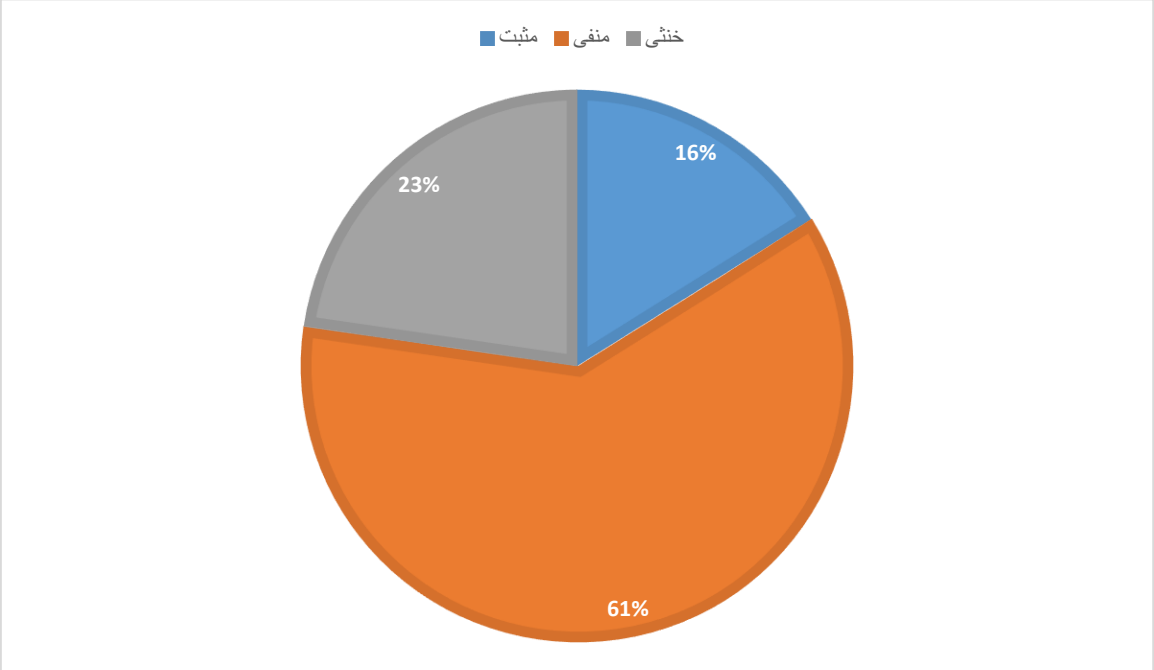
```
data = data[data.sentiment != "Neutral"]
data['text'] = data['text'].apply(lambda x: x.lower())
data['text'] = data['text'].apply((lambda x: re.sub('[^a-zA-z0-9\s]','',x)))

print(data[ data['sentiment'] == 'Positive'].size)
print(data[ data['sentiment'] == 'Negative'].size)

for idx,row in data.iterrows():
    row[0] = row[0].replace('rt',' ')
```

```
4472
16986
```

```
pos_cnt, neg_cnt, neu_cnt = 0, 0, 0
for i in data['sentiment']:
    if i == 'Positive':
        pos_cnt += 1
    elif i == 'Negative':
        neg_cnt += 1
    else:
        neu_cnt += 1
print("number of positive data:", '{:.3g}'.format(pos_cnt / len(data) * 100), "%")
print("number of negative data:", '{:.3g}'.format(neg_cnt / len(data) * 100), "%")
print("number of neutral data:", '{:.3g}'.format(neu_cnt / len(data) * 100), "%")
```

```
number of positive data: 16.1 %
number of negative data: 61.2 %
number of neutral data: 22.7 %
```

درصد توئیت های مثبت، منفی و خنثی در شکل بالا دیده میشود.



1-3-جداسازی واژگان (Tokenization)

فرایندی است که در آن یک داده متنی داده شده، به واحدهای زبانی کوچک‌تری به نام «توکن» (Token) تقسیم‌بندی می‌شود. کلمات، اعداد، علائم نقطه‌گذاری و سایر موارد، از جمله واحدهای زبانی هستند که به عنوان توکن(Token) شناخته می‌شوند. با استفاده از این روش میتوان لیستی از تکرار کلمات ایجاد کرد به شکلی که پرتکرارترین کلمه اولین عضو آن لیست باشد و به همین ترتیب آخرین

عضو کم تکرارترین کلمه باشد. در گام بعد به هر داده یک توالی عدد نسبت میدهد که در بخش Embedding استفاده میشود.

لازم به ذکر است که این مرحله شامل Stemming و POS tagging نمیشود. Stemming به معنای برگرداندن کلمات به ریشه آنهاست، برای مثال player و playing ، play در نظر گرفته شود، و POS tagging به دسته بندی کلمات بر حست جایگاه آنها در جمله (فعل، اسم، فاعل و...) اشاره دارد.

از دستورات زیر برای توضیحات بالا استفاده شده است:

```
max_fatures = 2000
tokenizer = Tokenizer(num_words=max_fatures, split=' ')
tokenizer.fit_on_texts(data['text'].values)
X = tokenizer.texts_to_sequences(data['text'].values)
X = pad_sequences(X)
```

1-4- انتخاب مدل LSTM

مفهوم اصلی word embedding این است که تمامی لغات استفاده شده در یک زبان را میتوان توسط مجموعه ای از اعداد اعشاری (در قالب یک بردار) بیان کرد. Word embedding ها بردارهای n-بعدی ای هستند که تلاش میکنند معنای لغات و محتوای آنها را با مقادیر عددی خود ثبت و ضبط کنند. ابتدا به بررسی روش کدگذاری one_hot میپردازیم. در روش one_hot هر المان در بردار به یک کلمه در لغت نامه اختصاص داده میشود یعنی هر درایه این بردار معرف یک کلمه از لغت نامه خواهد بود. در واقع اگر ما یک لغت نامه با 2000 کلمه داشته باشیم، برای هر کلمه یک بردار با 2000 جایگاه خواهیم داشت که تنها یکی از این خانه ها مقدار 1 و باقی 1999 خانه دیگر 0 است. به علاوه در این روش نمیتوانیم مقیاس معنا داری برای مقایسه بردار ها داسته باشیم و تنها میتوانیم مساوی بودن آنها را بررسی کنیم.

	۱	۲	۳	٤	٥	٦	۷	۸	۹
پیرمرد	۱	۰	۰	۰	۰	۰	۰	۰	۰
پیرزن	۰	۱	۰	۰	۰	۰	۰	۰	۰
پسر	۰	۰	۱	۰	۰	۰	۰	۰	۰
دختر	۰	۰	۰	۱	۰	۰	۰	۰	۰
شاهزاده	۰	۰	۰	۰	۱	۰	۰	۰	۰
شاهدخت	۰	۰	۰	۰	۰	۱	۰	۰	۰
ملکه	۰	۰	۰	۰	۰	۰	۱	۰	۰
پادشاه	۰	۰	۰	۰	۰	۰	۰	۱	۰
حکمران	۰	۰	۰	۰	۰	۰	۰	۰	۱

مثالی از کدگذاری one_hot

حالا اگر بخواهیم ابعاد این شیوه کدگذاری را کاهش دهیم یعنی از تعداد عدد کمتری برای نمایش هر کلمه استفاده کنیم، میتوانیم بعد ها و ویژگی هایی را در نظر بگیریم که برای هر کلمه یک عدد اعشاری بین 0 تا 1 در هر بعد در نظر گرفته میشود که مقدار دارا بودن یا نبودن آن ویژگی را برای کلمه تعریف میکند. این مجموعه embedding ها بهینه تر از embedding هایی است که در ابتدا با کمک روش رمزگذاری one-hot بدست آورده شد. از طرفی کلمات مشابه embedding های مشابه دارد و همچنین ماتریس متراکم تری در انتها به دست می آید.

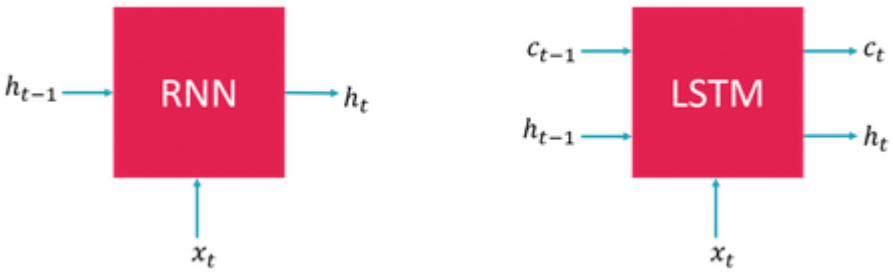
	مردانگی	جوانی	حکومت
پیرمرد	۱	۰٫۱	۰
پیرزن	۰	۰٫۱	۰
پسر	۱	۰٫۹۵	۰
دختر	۰	۰٫۹۵	۰
شاهزاده	۱	۰٫۸۵	۰
شاهدخت	۰	۰٫۸۵	۰
ملکه	۰	۰٫۴۵	۱
پادشاه	۱	۰٫۳۰	۱
حکمران	۰٫۵	۰٫۵	۱

مثالی از مجموعه embedding اصلاح شده

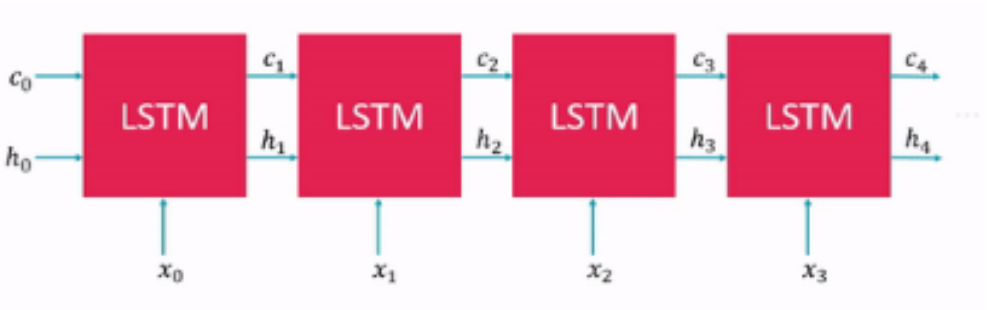
با توجه به توضیحات بالا لایه اولی که برای مدل در نظر گرفتیم لایه embedding از نوع دوم هست.

برای لایه دوم از spatial dropout استفاده کردیم. فرقی که این لایه با dropout عادی دارد این است که در این روش هر بار یک row صفر در نظر گرفته میشود و محاسبات با دیگر row ها انجام میشود. علت انتخاب این روش به دلیل correlated بودن دیتاست بود.

لایه سوم LSTM انتخاب شد. شبکه عصبی LSTM هم مانند شبکه RNN به صورت زنجیره‌ای پشت سرهم قرار می‌گیرد با این تفاوت که شبکه عصبی RNN مشکل وابستگی بلندمدت یا Long Term Dependency دارد. یعنی اینکه، نمی‌تواند در جمله‌ها، پاراگراف‌ها و تمامی دنباله‌های طولانی از داده‌ها عملکرد خوبی از خود نشان دهد. شبکه LSTM مخفف عبارت Long Short Term Memory است. یعنی حافظه بلندمدت دارد و دقیقا نقطه مقابل شبکه RNN است که از این مشکل رنج می‌برد. در واقع تنها یک مسیر بین ورودی و خروجی شبکه RNN شکل می‌گیرد. اما شبکه LSTM متفاوت است. این شبکه دو ورودی و خروجی دارد که بین این ورودی و خروجی‌ها، یکی از ورودی‌ها مستقیم به خروجی متصل شده است. در شکل میتوان مشاهده کرد که ورودی Ct-1 مستقیما به خروجی Ct متصل شده است. این اتصال همین‌طور ساده از اول تا آخر دنباله ادامه دارد. C مخفف Cell State هست و یک مولفه کلیدی در LSTM است. به Cell State، حافظه بلندمدت یا Long Term Memory هم گفته می‌شود.



مقایسه ورودی ها و خروجی های شبکه عصبی RNN و LSTM



نمای کلی ساختار شبکه عصبی LSTM

در آخر هم برای classification هم از لایه dense با تابع فعالساز softmax استفاده میکنیم.

```
#Embedding
embed_dim = 128
lstm_out = 196

model = Sequential()
model.add(Embedding(max_fatures, embed_dim,input_length = X.shape[1]))
model.add(SpatialDropout1D(0.4))
model.add(LSTM(lstm_out, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(3,activation='softmax'))
model.compile(loss = 'categorical_crossentropy', optimizer='adam',metrics = ['accuracy'])
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 28, 128)           256000

 spatial_dropout1d (SpatialD  (None, 28, 128)          0
 ropout1D)

 lstm (LSTM)                 (None, 196)               254800

 dense (Dense)               (None, 3)                 591

=================================================================
Total params: 511,391
Trainable params: 511,391
Non-trainable params: 0
_________________________________________________________________
None
```

برای انتخاب بهتر و دقت بالاتر، مدل BERT هم تست شد و بر روی دیتاست train شد. نحوه کار مدل به این شکل است که در هر مرحله به صورت رندوم 15 درصد از کلمات را در نظر گرفته و پیشبینی را برای آنها انجام میدهد. نتایج این دو مدل را با هم مقایسه خواهیم کرد.

5-1- آموزش (train)

مدل را با 50 epock آموزش میدهیم تا به دقت و خطای مطلوب برسیم.

```
Y = pd.get_dummies(data['sentiment']).values
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.33, random_state = 42)
print(X_train.shape,Y_train.shape)
print(X_test.shape,Y_test.shape)
```

```
(7188, 28) (7188, 2)
(3541, 28) (3541, 2)
```

```
#train
batch_size = 32
model.fit(X_train, Y_train, epochs = 50, batch_size=batch_size, verbose = 2)
```

```
Epoch 45/50
225/225 - 10s - loss: 0.0615 - accuracy: 0.9736 - 10s/epoch - 45ms/step
Epoch 46/50
225/225 - 10s - loss: 0.0598 - accuracy: 0.9722 - 10s/epoch - 43ms/step
Epoch 47/50
225/225 - 10s - loss: 0.0606 - accuracy: 0.9733 - 10s/epoch - 43ms/step
Epoch 48/50
225/225 - 10s - loss: 0.0578 - accuracy: 0.9738 - 10s/epoch - 43ms/step
Epoch 49/50
225/225 - 10s - loss: 0.0593 - accuracy: 0.9748 - 10s/epoch - 43ms/step
Epoch 50/50
225/225 - 10s - loss: 0.0608 - accuracy: 0.9734 - 10s/epoch - 45ms/step
```

در انتها به خطای حدود 0.0608 و دقت 97.34 درصد رسیدیم.

```
model.save("model.h5")
```

مدل را ذخیره میکنیم.

6-1- تست مدل

حال با داده های تست، مدل آموزش دیده را تست میکنیم تا دقت و خطا آن را بررسی کنیم.

```
#Extracting a validation set, and measuring score and accuracy
validation_size = 1500

X_validate = X_test[-validation_size:]
Y_validate = Y_test[-validation_size:]
X_test = X_test[:-validation_size]
Y_test = Y_test[:-validation_size]
score,acc = model.evaluate(X_test, Y_test, verbose = 2, batch_size = batch_size)
print("score: %.2f" % (score))
print("acc: %.2f" % (acc))
```

```
64/64 - 1s - loss: 1.0172 - accuracy: 0.8182 - 867ms/epoch - 14ms/step
score: 1.02
acc: 0.82
```

همانطور که در شکل بالا مشخص است دقت داده های پیشبینی شده 81.82 درصد و خطای آن 1.0172 بوده است.

از آنجایی که دو دسته احساسات مثبت و منفی داشتیم، تعداد حدس ها درست برای هر کدام را محاسبه میکنیم.

```
#measuring the number of correct guesses
pos_cnt, neg_cnt, pos_correct, neg_correct = 0, 0, 0, 0
for x in range(len(X_validate)):

    result = model.predict(X_validate[x].reshape(1,X_test.shape[1]),batch_size=1,verbose = 2)[0]

    if np.argmax(result) == np.argmax(Y_validate[x]):
        if np.argmax(Y_validate[x]) == 0:
            neg_correct += 1
        else:
            pos_correct += 1

    if np.argmax(Y_validate[x]) == 0:
        neg_cnt += 1
    else:
        pos_cnt += 1
```

```
pos_acc 57.92880258899677 %
neg_acc 89.67254408060454 %
total_acc 83.13333333333334 %
```

مشاهده میشود 57.9 توئیت های مثبت و 89.6 توئیت های منفی درست حدس زده شده اند. این تفاوت از قبل هم پیشبینی شده بود و علت آن تعداد بیشتر توئیت های منفی بود که باعث آموزش دقیق تر مدل شده بود.

در انتها به مقایسه دقت دو مدل LSTM و BERT میپردازیم:

دقت	آموزش	تست
LSTM	97.3%	83.1%
BERT	93.8%	74.2%

مشاهده میشود که عملکرد دقت مدل LSTM هم در آموزش و هم در تست از مدل BERT بهتر بوده است.

2-سطح متوسط

در این بخش سعی داریم که با استفاده از Twitter API ، 20 توئیت آخر با هشتگ #FIFAWorldCup و #TedCruz را استخراج کرده و آنالیز احساسات را روی آنها انجام دهیم.

برای استفاده از Twitter API مشکلاتی بود. از جمله این که برای ساختن developer اکانت در توئیتر باید شماره تلفن همراه verify میشد که شماره با IP ایران را نمیپذیرفت. پس استفاده از یک شماره با IP غیر از ایران هم باید اپلیکیشن فرم پر میشد که تایید شدن آن هم دو روز طول میکشید. برای جلوگیری از اتلاف وقت ابزار دیگری به اسم twint را جایگزین کردیم. ابزار twint به ما این امکان را میدهد که توئیت های مورد نظر را با فرمت دلخواه و با فیچر یا بدون فیچر استخراج کنیم. البته در نهایت اکانت developer تایید شد و کد را هم با استفاده از Twitter API آپدیت کردیم.

2-1- تعریف کلاس TwitterSentClass

ابتدا یک کلاس به اسم TwitterSentClass تعریف میکنیم. در این کلاس ابتدا توکن های مورد نیاز اکانت developer تعریف و مقداردهی میشود(API_key, API_secret, access_token, access_token_secret). این توکن ها مرتب باید آپدیت شوند که به دلیل امنیت اکانت و عدم سوء استفاده از آن است. تابع __init__ در واقع دسترسی و عدم دسترسی به اکانت developer و Twitter API را بررسی میکند.

```
class TwitterSentClass():
    def __init__(self):
        API_key = 'WZPFyMbeDLBqiTbXSl5WlqWxv'
        API_secret = 'YY7MBZk758fEoBjyvxPFDEuVBdmipBWiB7S52GJhtAgvQ0hvXx'
        access_token = '1613990303534354449-PHrmHWiBJ1GYozKZF6DNuJv1sTF8E3'
        access_token_secret = 'syzYvlQnBk6nMEM7xEXxfLBmZ5yybksOMweLct56XkQpO'
        try:
            self.auth = OAuthHandler(API_key,
                                     API_secret)
            self.auth.set_access_token(access_token,
                                       access_token_secret)
            self.api = tweepy.API(self.auth)
            print('Authenticated')
        except:
            print("Sorry! Error in authentication!")
```

تابع بعدی، تابع cleaning_process است که وظیفه پاکسازی کد و حذف عبارات، لینک ها و کاراکتر های بی تاثیر یا کم اهمیت را دارد.

```
def cleaning_process(self, tweet):
    text = tweet.lower()
    text = re.sub('\[.*?\]', '', tweet)
    text = re.sub('https?://\S+|www\.\S+', '', text)
    text = re.sub('<.*?>+', '', text)
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
    text = re.sub('\n', '', text)
    text = re.sub('\w*\d\w*', '', text)
    return text
```

تابع بعد get_sentiment است که عمل جداسازی واژگان را روی توئیت ها انجام میدهد و احساسات آنها را به سه دسته مثبت، منفی و خنثی تقسیم میکند.

```
def get_sentiment(self, tweet):
    lst = tweet.split()
    tweet = tokenizer.texts_to_sequences(lst)
    tweet = pad_sequences(tweet)
    tweet = [[int(i[0]) for i in tweet]]
    analysis = model.predict([tweet],batch_size=1,verbose = 3)[0]
    if np.argmax(analysis) > 0:
        return 'positive'
    elif np.argmax(analysis) == 0:
        return 'neutral'
    else:
        return 'negative'
```

تابع آخر هم get_tweets است که وظیفه فچ کردن توئیت و استخراج متن و احساسات آن را دارد.

```
def get_tweets(self, query, count=1000):
    tweets = []
    try:
        fetched_tweets = self.api.search_tweets(q = query, count = count)
        for tweet in fetched_tweets:
            parsed_tweet = {}
            parsed_tweet['text'] = tweet.text
            parsed_tweet['sentiment'] =self.get_sentiment(tweet.text)
            if tweet.retweet_count > 0:
                if parsed_tweet not in tweets:
                    tweets.append(parsed_tweet)
            else:
                tweets.append(parsed_tweet)
        return tweets
```

2-2- توابع لازم برای نمودار های world cloud

برای طراحی یک world cloud که کلماتی که بیشترین استفاده را در توئیت های مدنظر داشته اند را نشان دهد، توابع زیر را تعریف کردیم که شکل دهی و ویژگی های نمایش آن را مشخص میکنند.

```
def load_mask(mask_url):
    with urllib.request.urlopen(mask_url) as resp:
        mask = np.asarray(bytearray(resp.read()), dtype="uint8")
        mask = cv2.imdecode(mask, cv2.IMREAD_COLOR)
        mask = cv2.cvtColor(mask, cv2.COLOR_BGR2RGB)

    return mask
```

```
def plot_word_cloud(df_tweet, colormap, category):
    new_mask = cv2.imread('Twitter-featured.png')

    wordcloud = WordCloud(
        background_color="white",
        colormap=colormap,
        mask=new_mask,
        random_state=42,
        max_font_size=50,
        max_words=1000,
    )

    text = ''.join(df_tweet[1:])


    wordcloud.generate(text)

    image_colors = ImageColorGenerator(new_mask)

    plt.figure(figsize=(16, 8))
    plt.title('Word cloud for {} tweets'.format(category))
    plt.imshow(wordcloud, interpolation="bilinear")
    plt.axis("off")
    plt.show()
```

2-3- بررسی هشتگ #TedCruz

ابتدا هشتگ #TedCruz را بررسی میکنیم:

20 توئیت آخر این هشتگ را استخراج میکنیم، توئیت های مثبت و منفی را تشخیص و از هم جدا میکنیم و در نهایت 5 توئیت آخر مثبت و 5 توئیت آخر منفی را ذخیره میکنیم.

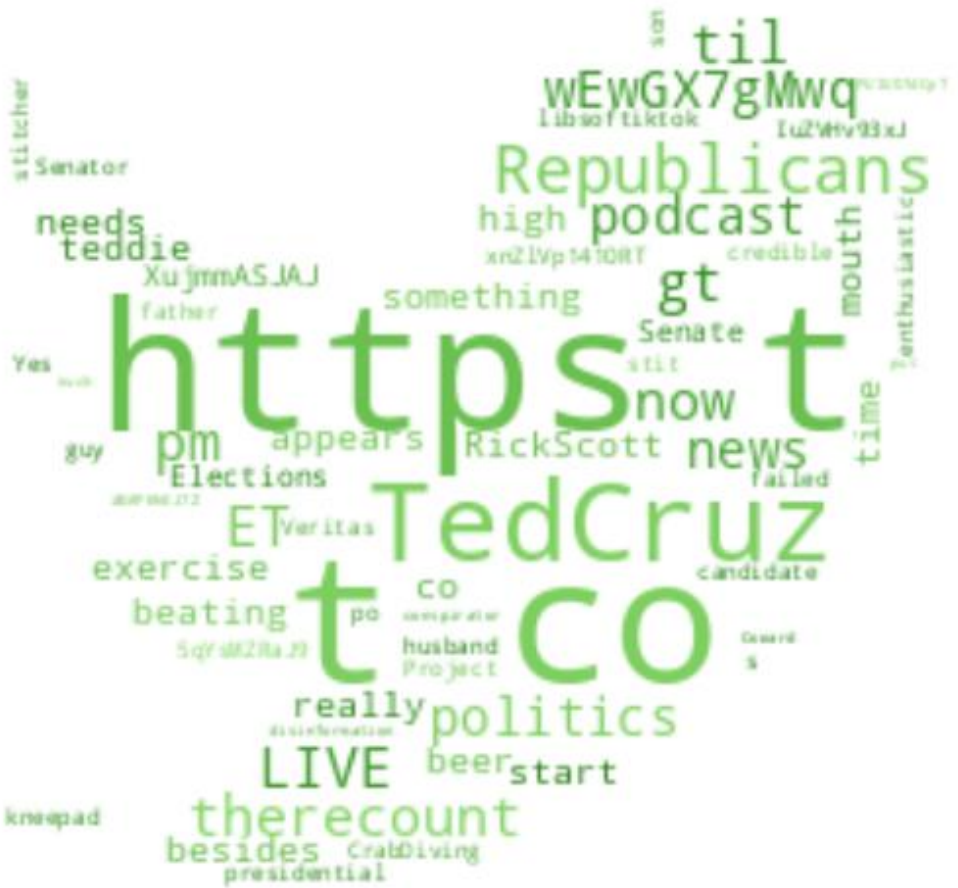
```
arr_pred = []
arr_pos_txt = []
arr_neg_txt = []
api = TwitterSentClass()
t = '#TedCruz'
tweets = api.get_tweets(query = t, count = 2000)

pos_tweets = [tweet for tweet in tweets if tweet['sentiment'] == 'positive']
pos = "Positive tweets percentage: {} %".format(100*len(pos_tweets)/len(tweets))

neg_tweets = [tweet for tweet in tweets if tweet['sentiment'] == 'negative']
neg="Negative tweets percentage: {}%".format(100*len(neg_tweets)/len(tweets))
arr_pred.append(pos)
arr_pred.append(neg)

# storing first 5 positive tweets
arr_pos_txt.append("Positive tweets:")
for tweet in pos_tweets[:5]:
    arr_pos_txt.append(tweet['text'])

# storing first 5 negative tweets
arr_neg_txt.append("Negative tweets:")
for tweet in neg_tweets[:5]:
    arr_neg_txt.append(tweet['text'])
```

تابع plot_most_common_terms تعریف شده تا بیشترین کلمات استفاده شده در توئیت ها را تشخیص و نمایش دهد.

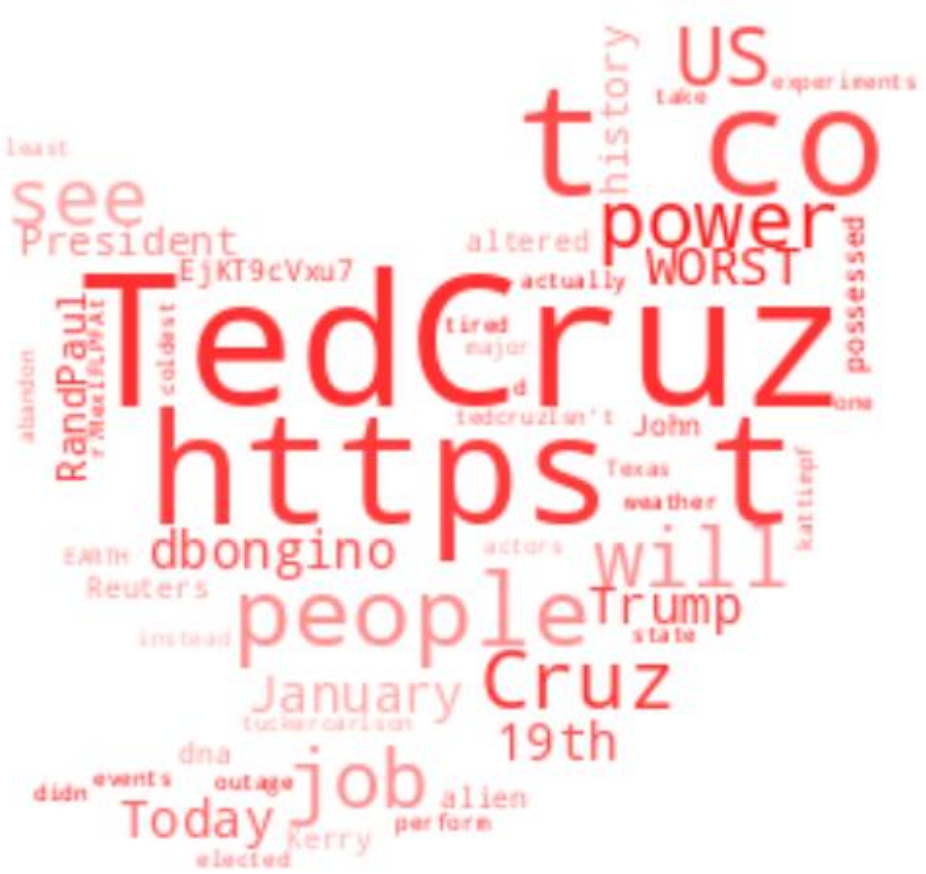
```
def plot_most_common_terms(df):
    word_list = []

    for i, j in pd.DataFrame(df).iterrows():
        for word in j['text'].split():
            if len(word) > 3 and 'ted' not in word.lower():
                word_list.append(word)

    count_dict = Counter(word_list)
    most_common_words_df = pd.DataFrame(count_dict.most_common(20), columns=['word', 'count'])

    fig = px.histogram(most_common_words_df,
                       x='word',
                       y='count',
                       title='Most common terms used in #TedCruz\'s tweets.',
                       color_discrete_sequence=['#D8E46B'] )
    fig.show()
plot_most_common_terms(tweets)
```

در دو نمودار word cloud زیر میتوانیم بیشترین کلمات استفاده شده در توئیت های مثبت و توئیت ها منفی را ببینیم:

```
colors_pos = ["#88D969", "#1D800E"]
cmap_pos = LinearSegmentedColormap.from_list("mycmapos", colors_pos)

colors_neg = ["#ffbaba", "#ff0000"]
cmap_neg = LinearSegmentedColormap.from_list("mycmapos", colors_neg)

colors_neutral = ["#ffecb3", "#ffca28"]
cmap_neutral = LinearSegmentedColormap.from_list("mycmapos", colors_neutral)

pos_cloud = []
for tweet in pos_tweets:
    pos_cloud.append(tweet['text'])

neg_cloud = []
for tweet in neg_tweets:
    neg_cloud.append(tweet['text'])

plot_word_cloud(pos_cloud, cmap_pos, 'positive')
plot_word_cloud(neg_cloud, cmap_neg, 'negative')
```



بیشترین کلمات استفاده شده در توئیت های مثبت برای هشتگ #TedCruz



بیشترین کلمات استفاده شده در توئیت های منفی برای هشتگ #TedCruz

درصد توئیت های مثبت و منفی این هشتگ را استخراج میکنیم:

```
print("percentage of positive tweets: ", '{0:.3g}'.format(len(pos_cloud) / (len(tweets)) * 100), "%")
print("percentage of negative tweets: ", '{0:.3g}'.format(len(neg_tweets) / (len(tweets)) * 100), "%")
```

```
percentage of positive tweets:  14.8 %
percentage of negative tweets:  27.8 %
```

دقت پیشبینی های احساسات را بررسی میکنیم:

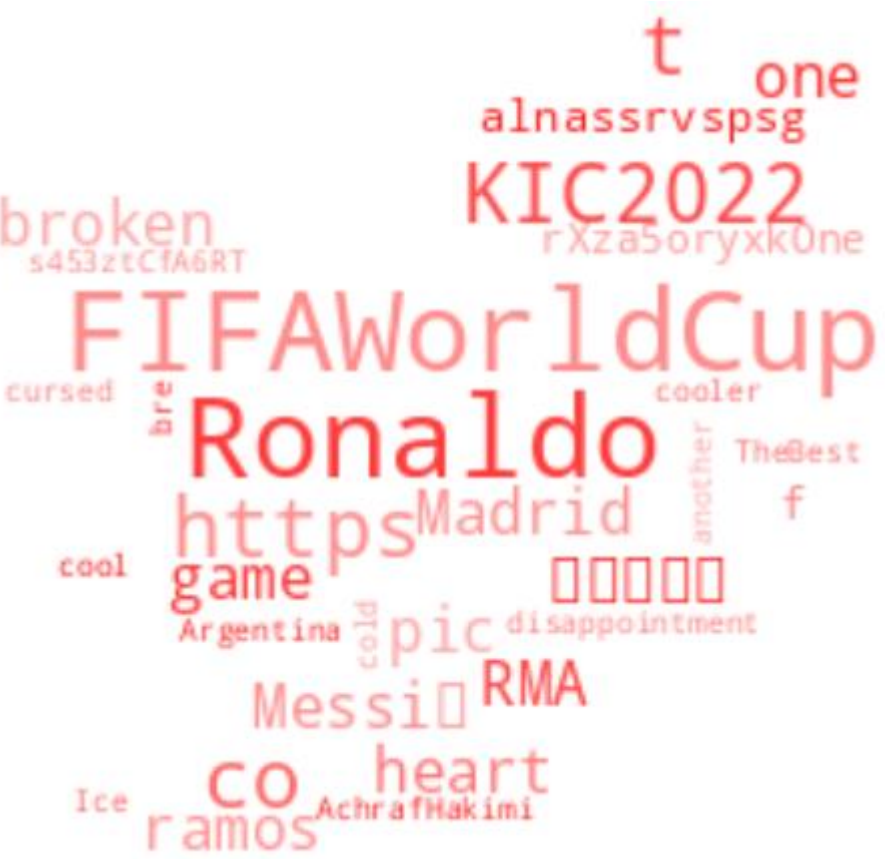
```
cnt = 0
for i in range(20):
    if tweets[i]['sentiment'] == lebeled_tweets[i]['sentiment']:
        cnt += 1
print("accuracy for the #TedCruz first 20 tweets:", cnt / 20 * 100, "%")
```

دقت مشاهده شده 70 درصد از صحت پیش بینی ها گزارش شده است.

4-2- بررسی هشتگ FIFAWorldCup

همه این مراحل را برای هشتگ FIFAWorldCup# تکرار کردیم و به نتایج زیر رسیدیم:

بیشترین کلمات استفاده شده در توئیت های مثبت برای هشتگ FIFAWorldCup#



بیشترین کلمات استفاده شده در توئیت های منفی برای هشتگ #FIFAWorldCup

درصد توئیت های مثبت و منفی:

```
percentage of positive tweets:  23.3 %
percentage of negative tweets:  11.6 %
```

بررسی دقت صحت پیشبینی: 80 درصد

```
cnt = 0
for i in range(20):
    if tweets[i]['sentiment'] == labeled_tweets[i]['sentiment']:
        cnt += 1
print("accuracy for the #FIFAWorldCup first 20 tweets:", cnt / 20 * 100, "%")
```

```
accuracy for the #FIFAWorldCup first 20 tweets: 80.0 %
```

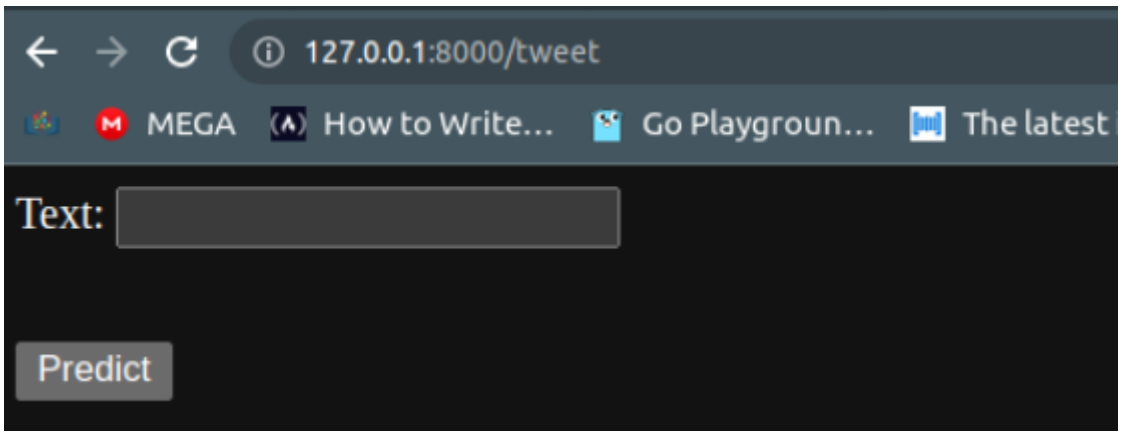
3-سطح پیشرفته

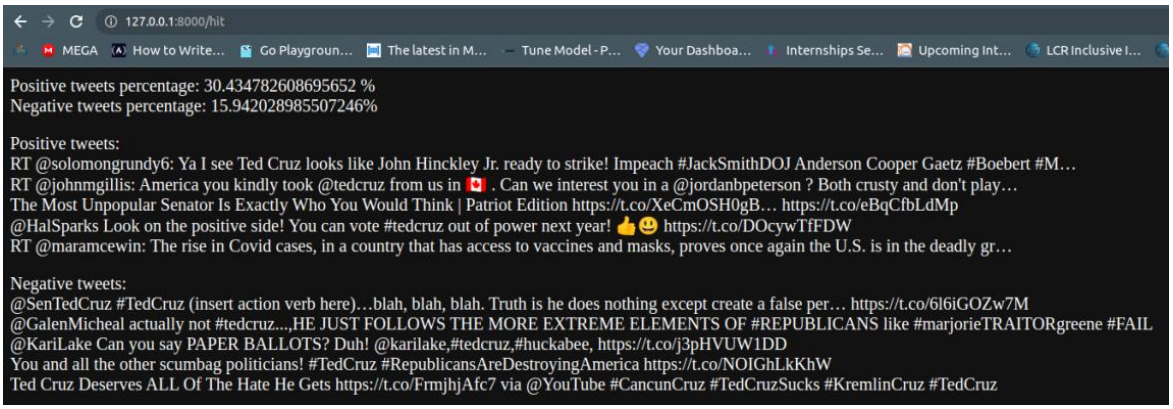
در این بخش سعی داریم آنالیز احساسات روی متن را به شکل آنی(real-time) و روی سخت افزار موبایل پیاده کنیم. مجدد از ابزار Twitter API استفاده کردیم منتها برای deploy کردن روش های مختلفی را امتحان کردیم.

1. کتابخانه deephaven : برای استفاده از این کتابخانه به مشکل deprecated بودن فانکشن ها برخوردیم و از آنجا که آپدیت هایی برای رفع مشکل وجود نداشت به سراغ روش های دیگر رفتیم.

2. روش heroku: مشکل این روش این بود که به IP ایران دسترسی نمیداد و حتی با VPNهم قابل دسترسی نبود.

3. روش jango: در آخر از این روش استفاده کردیم. برنامه به این صورت نوشته شده که کاربر عبارت مد نظر را وارد میکند و آخرین توئیت های مثبت و منفی مربوطه در خروجی نمایش داده میشود.





عملکرد برنامه در عکس های بالا مشاهده میشود.

منابع

1. https://www.kaggle.com/datasets/crowdflower/first-gop-debate-twitter-sentiment/
2. https://github.com/deephaven/deephaven-core
3. https://huggingface.co/bert-base-uncased
4. https://github.com/twintproject/twint
5. https://deephaven.io/