به نام خداوند جان و خرد

دانشگاه تهران

دانشکدگان فنی

دانشکده مهندسی برق و کامپیوتر

پردازش زبان طبیعی

تمرین شماره ۲ - بخش اول

نام و نام خانوادگی: علی خرم فر

شماره دانشجویی: ۸۱۰۱۰۲۱۲۹

اسفندماه ۱۴۰۲

فهرست مطالب

۱_ پاسخ سوال اول

برای تحلیل احساسات در این مسئله از سرویس Colab استفاده شده‌است. به این منظور ابتدا لازم است که یک API از Kaggle دریافت کنیم تا بتوانیم دیتاست را در Colab بارگذاری کنیم. پس از آن دیتاست را در Colab بارگذاری کرده و باتوجه به اینکه اسامی ستون‌ها در این دیتاست تعیین نشده، از اسامی که در یکی از پروژه‌های Kaggle به کار رفته را استفاده می‌کنیم:

```
column_names = ["target", "id", "date", "flag", "user", "text"]
```

```
import opendatasets as od
od.download(
    "https://www.kaggle.com/datasets/kazanova/sentiment140")
```

```
Please provide your Kaggle credentials to download this dataset. Learn more: http://bit.ly/kaggle-creds
Your Kaggle username: alikhoramfar
Your Kaggle Key: ·····
Downloading sentiment140.zip to ./sentiment140
100%|          | 80.9M/80.9M [00:01<00:00, 72.9MB/s]
```

Import Data to Colab

```
[19] import pandas as pd
     column_names = ["target", "id", "date", "flag", "user", "text"]
     file =('sentiment140/training.1600000.processed.noemoticon.csv')
     data = pd.read_csv(file, encoding='ISO-8859-1', names=column_names)
     data.head()
```

	target	id	date	flag	user	text
0	0	1467810369	Mon Apr 06 22:19:45 PDT 2009	NO_QUERY	_TheSpecialOne_	@switchfoot http://twitpic.com/2y1zl - Awww, t...
1	0	1467810672	Mon Apr 06 22:19:49 PDT 2009	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by ...
2	0	1467810917	Mon Apr 06 22:19:53 PDT 2009	NO_QUERY	mattycus	@Kenichan I dived many times for the ball. Man...

با توجه به بررسی‌های انجام شده ستون Target که همان متغیر کلاس است، دو مقدار ۰ و ۴ دارد. ۰ به معنی منفی و ۴ به معنی مثبت است.

۱-۱_ بخش اول

ابتدا داده‌ها را بر اساس متغیر targetکه ۰ یا ۴ باشه به دو قسمت تقسیم کرده و از هر کدام ۵۰۰۰ تا نمونه‌برداری می‌کنیم. در حال حاضر باتوجه به اینکه در ادامه مسئله دقت تخمین اهمیت دارد، randomstate را نیز مقداردهی می‌کنیم تا دقت را به خوبی ارزیابی کنیم.

Part 1: Preproccessing

```
negative_tweets = data[data['target'] == 0]
positive_tweets = data[data['target'] == 4]

n_sample = 5000
negative_samples = negative_tweets.sample(n=n_sample, random_state=42)
positive_samples = positive_tweets.sample(n=n_sample, random_state=42)
```

نرمال‌سازی با تبدیل حروف بزرگ به کوچک

در اولین مرحله حروف بزرگ را به کوچک تبدیل می‌کنیم. علت این کار یکسان‌شدن ارزیابی و شناسایی تمامی کلماتی است که ممکن است برخی از حروف آن‌ها بزرگ تایپ شده باشد. باتوجه به اینکه این دیتاست از توییت‌های تایپ شده توسط کاربران است علاوه بر اینکه برخی کلمات اول جمله بوده و حرف بزرگ دارند، برخی دیگر به علت خطای تایپی حروف بزرگ دارند. البته باید توجه داشت در بررسی تخصصی و تحلیل احساسات کوچک یا بزرگ بودن حروف اهمیت دارد ولی در این تمرین از این موضوع صرف نظر می‌شود تا بتوانیم دقت بالاتری در مدل داشته باشیم.

برای این کار کافیست تابع مورد نظر را روی ستون text هر دو نمونه اعمال کنیم.

```
# Normalization -> lower case
negative_samples['text'] = negative_samples['text'].str.lower()
positive_samples['text'] = positive_samplesD['text'].str.lower()
```

حذف StopWords انگلیسی از متن

حروفی مانند the معنی و تحلیل خاصی در احساسات ندارند و در اکثر جملات استفاده می‌شوند پس فرکانس آن‌ها بالاست و در روش این تمرین اطلاعات خاصی نمی‌دهند، پس حذف آن‌ها می‌تواند دقت مدل را افزایش دهد. به این منظور از پکیج nltk.corpus ، stopwords را import می‌کنیم. StopWods های انگلیسی در زیر خروجی گرفته شده اند که برخی از آن‌ها قابل مشاهده است.

```
# Removing Stop Words
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
stop_words
```

```
'all',
'am',
'an',
'and',
'any',
'are',
'aren',
"aren't",
'as',
'at',
'be',
```

کافی تمامی آن‌ها از مقدار ستون Text هر ردیف حذف شود:

```
[56] # Removing Stop Words
     import nltk
     from nltk.corpus import stopwords
     #nltk.download('stopwords')
     stop_words = set(stopwords.words('english'))

     def remove_stopwords(text):
         tokens = text.split()
         tokens = [word for word in tokens if word not in stop_words]
         text = ' '.join(tokens)
         return text
     negative_samples['text'] = negative_samples['text'].apply(lambda text: remove_stopwords(text))
     positive_samples['text'] = positive_samples['text'].apply(lambda text: remove_stopwords(text))
```

نرمال‌سازی با حذف علائم نگارشی

اگرچه علائم نگارشی مثل ! در تحلیل احساسات کاربرد دارند ولی در این روش احتمال می‌دهم که تاثیری نداشته باشند. پس تمام علائم نگارشی حذف می‌شوند. هرچند که در نتیجه گیری نهایی هر دو حالت بدون علامت و با علامت بررسی می‌شود تا میزان تاثیر آن مشخص شود. به این منظور تابع remove_punctuation تعریف شده که در این تابع علامات نگارشی با کمک string.punctuation با پوچ جایگذاری می‌شوند که بر روی تمامی string های text در هر دو نمونه برداری اعمال می‌شود :

```
# Normalization -> Remove Punctuation
import string
def remove_punctuation(text):
    translator = str.maketrans('', '', string.punctuation)
    return text.translate(translator)

negative_samples['text'] =negative_samples['text'].apply(lambda text: remove_punctuation(text))
positive_samples['text'] =positive_samples['text'].apply(lambda text: remove_punctuation(text))
```

اعمال توکنیایزر

برای توکنایز کردن متن هر توییت از پکیج NLTK تابع word_tokenize وارد می‌شود. سپس با کمک تابع تعریف شده متن هر توییت با توکن‌های آن جایگزین می‌شود:

```
# Tokenization
from nltk.tokenize import word_tokenize
nltk.download('punkt')
def tokenize_text(text):
    tokens = word_tokenize(text)
    return tokens
negative_samples['text'] =negative_samples['text'].apply(lambda text: tokenize_text(text))
positive_samples['text'] =positive_samples['text'].apply(lambda text: tokenize_text(text))
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

```
[60] negative_samples
```

	target	id	date	flag	user	text
212188	0	1974671194	Sat May 30 13:36:31 PDT 2009	NO_QUERY	simba98	[xnausikaax, oh, u, order, thats, horrible]
299036	0	1997882236	Mon Jun 01 17:37:11 PDT 2009	NO_QUERY	Seve76	[great, hard, training, weekend, couple, days,...
475978	0	2177756662	Mon Jun 15 06:39:05 PDT 2009	NO_QUERY	x__claireyy__x	[right, work, 5, hours, go, im, free, xd]
588988	0	2216838047	Wed Jun 17 20:02:12 PDT 2009	NO_QUERY	Balasi	[craving, japanese, food]
138859	0	1880666283	Fri May 22 02:03:31 PDT 2009	NO_QUERY	djrickdawson	[jean, michel, jarre, concert, tomorrow, got, ...

در تصویر بالا مشخص است که متن توییت به توکن‌هایی که با کمک تابع مربوطه بدست آمده‌اند، تجزیه شده است.

جداسازی داده آزمون

باتوجه به اینکه بیان شده ۲۰ درصد داده‌ها باید داده آزمون باشند، پس ابتدا نمونه‌برداری‌هایی که از هرکلاس داشتیم را ترکیب کرده و ۲۰ درصد آن‌ها را به عنوان داده آزمون و ۸۰ درصد آن‌ها را به عنوان داده آموزش دسته‌بندی می‌کنیم:

```
from sklearn.model_selection import train_test_split
samples = pd.concat([negative_samples, positive_samples])
train_data, test_data = train_test_split(samples, test_size=0.2, random_state=42)
```

۲-۱_ بخش دوم

در این قسمت قصد داریم که جدول TF را برای توییت‌ها پیاده‌سازی کنیم. در هر سطر از این جدول یک نمونه است و ستون‌های آن نیز کلماتی هستند که در این توییت‌ها استفاده شده‌اند. در هر سطر مقادیر تکرارشدن کلمه مورد نظر در آن نمونه متناظر با ستون مربوط به آن کلمه قرار می‌گیرد.

ابتدا تمامی کلمات را در یک set اضافه می‌کنیم تا کلمات تکراری اضافه نشوند. بعد از بدست آوردن تمامی کلمات، متوجه شدیم که کلمات زیادی متشکل از اعداد هستند. پس در مرحله قبلی بهتر است اعداد هم حذف شوند زیرا که تاثیری بر تحلیل احساسات از این روش ندارند.

```
vocabulary = set()
for txt in train_data['text']:
  for word in txt:
    vocabulary.add(word)
vocabulary = sorted(list(vocabulary))
```

```
vocabulary
```

```
['0',
 '03',
 '041',
 '0448417513',
 '049',
 '06',
 '060',
 '0626',
 '07894539769',
 '09',
 '1',
 '10',
 '100',
 '1000',
 '1000kb',
```

سپس یک ماتریس به تعداد ردیف‌های تعداد نمونه‌های داده‌های آموزشی و تعداد ستون‌های کلمات می‌سازیم.

سپس در یک حلقه به تعداد کلماتی که در یک نمونه تعداد شده‌اند، اعداد ستون مربوط به آن کلمه را یکی اضافه می‌کنیم:

```
import numpy as np
TF_Matrix = np.zeros((len(train_data), len(vocabulary)),dtype=int)
counter = 0
for sample in train_data['text']:
  for word in sample:
    idx= vocabulary.index(word)
    TF_Matrix[counter,idx] +=1
  counter +=1
```

برای چک کردن اینکه ماتریس تولید شده به خوبی مقادیر را ثبت کرده ، جمله اول را بررسی می‌کنیم. همانطور که مشخص است برای نمونه کلمه need در این نمونه موجود است پس باید خانه مربوط به آن ۱ باشد. که خروجی درست است:

```
print(train_data.iloc[0].text)
TF_Matrix[0,vocabulary.index('need')]
```

```
['akynos', 'need', 'sit', 'one', 'day', 'drinks', 'convos', 'tweeter', 'bit', 'annoying']
1
```

برای اینکه این ماتریس شکل فرمال‌تری داشته باشد می‌توان آن را به یک دیتافریم تبدیل کرد که مقادیر ستون‌های آن کلمات هستند:

```
TF_Dataframe = pd.DataFrame(TF_Matrix,columns=vocabulary)
```

۳-۱_ بخش سوم

پارامتر Tf-Idf در اصل ترکیب ۲ پارامتر است:

Tf: که تعداد تکرار یک ترم در یک نمونه است. معمولا تبدیل لگاریتمی روی آن اعمال می‌شود.

$$\mathrm{tf}_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{count}(t,d) & \text{if } \mathrm{count}(t,d) > 0 \\ 0 & \text{otherwise} \end{cases}$$

Idf: معکوس فرکانس نمونه. یعنی نسبت کل نمونه‌ها به نمونه‌هایی که شامل کلمه i هستند:

$$\mathrm{idf}_i = \log\left(\frac{N}{\mathrm{df}_i}\right)$$

در نهایت برای بدست آوردن پارامتر Tf-Idf ضرب این دو پارامتر بدست می‌آید:

$$w_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t$$

حال به پیاده‌سازی این فرمول می‌پردازیم:

در قسمت قبلی تمرین ما مقدار Tf هر کلمه از هر نمونه را ذخیره کردیم. برای محاسبه idf هر کلمه کافی است مقادیر تمام ردیف‌های هر ستون که نماینده هر کلمه است را با هم جمع کنیم:

```
# Idf
N = len(TF_Matrix)
idf_values = {}
for word in vocabulary:
    df_i = np.sum(TF_Dataframe[word])
    idf_values[word] = np.log10(N / df_i) if df_i else 0
idf_values
```

```
'adventures': 3.425968732272281,
'advertise': 3.9030899869919438,
'advice': 3.204119982655925,
'advocate': 3.6020599913279625,
'aerliss': 3.9030899869919438,
'aerten': 3.9030899869919438,
'aest2am': 3.9030899869919438,
'aetelade': 3.9030899869919438,
'afew': 3.9030899869919438,
'affected': 3.9030899869919438,
'affiliate': 3.9030899869919438,
'afford': 3.1249387366083,
'afinefrenzy': 3.9030899869919438,
'afl': 3.9030899869919438,
'afraid': 3.3010299956639813,
'afraidquot': 3.9030899869919438,
'africa': 3.9030899869919438,
'aft': 3.9030899869919438,
```

✓ 0s completed at 7:58 AM

دلیل اینکه مقادیر زیادی ۳.۹۰ هستند این است که فقط ۱ بار تکرار شده‌اند.Log(8000/1) = 3.90

همچنین می‌توان از آرایه استفاده کرد تا سرعت ایندکس دهی بالاتر باشد.

```
[97] # Idf
     N = len(TF_Matrix)
     idf_Matrix = np.zeros(len(vocabulary),dtype=float)
     for i,word in enumerate(vocabulary):
       df_i = np.sum(TF_Dataframe[word])
       idf_Matrix[i] = np.log10(N / df_i) if df_i else 0
     idf_Matrix
```

```
array([3.90308999, 3.90308999, 3.90308999, ..., 3.90308999, 3.90308999,
       3.90308999])
```

حال کافیست برای بدست آوردن بردار Tf-Idf هر نمونه مقادیر متناظر هر پارامتر را در هم ضرب کرده تا حاصل آن بدست آید. برای اینکار کافی است از ماتریس tf یک کپی گرفته، به مقادیر آن تبدیل لگاریتمی اعمال کرده و سپس ضرب در مقدار idf کنیم.

```
# Calculate TF-IDF
Tf_Idf = np.where(TF_Matrix > 0, 1 + np.log10(TF_Matrix), 0)
for sample in Tf_Idf:
  for word_index in range(len(vocabulary)):
    sample[word_index] = sample[word_index] * Idf_Matrix[word_index]
```

حال مثلا برای نمونه اول،از بردار tf-idf مقدار کلمه need را خروجی میگیریم:

```
[155] Tf_Idf[0,vocabulary.index('need')]

      1.6382721639824072
```

۴-۱_ بخش چهارم

در این قسمت پارامتر PPMI را محاسبه می‌کنیم. ابتدا این پارامتر برای تمامی جفت کلمه‌ها محاسبه می‌شود. به طور کلی در آمار و احتمالات شاخص PMI برابر است با :

$$\mathrm{PMI}(X,Y) = \log_2 \frac{P(x,y)}{P(x)P(y)}$$

یعنی نسبت اینکه دو رخداد باهم رخ دهند به نسبت اینکه مستقلا رخ دهند.

حال در پردازش زبان طبیعی این مفهوم به صورتی است که به این سوال پاسخ می‌دهد که آیا تعداد دفعاتی که یک جفت کلمه با هم می‌آید بیشتر از دفعاتی است که جدا از هم می‌آیند یا خیر.

$$\mathrm{PMI}(word_1, word_2) = \log_2 \frac{P(word_1, word_2)}{P(word_1)P(word_2)}$$

که از رابطه بالا این مورد محاسبه می‌شود.

مشکلی که وجود دارد این است که زمانی که لگاریتم بین ۰ و ۱ باشد مقدار آن منفی است. این حالت زمانی رخ می‌دهد که تعداد دفعاتی که جفت کلمه باهم می‌آیند کمتر از تعداد دفعاتی باشد که مستقل می‌آیند. پس در این شرایط PPMI استفاده می‌شود که در آن مقادیر منفی با صفر جایگزین می‌شوند:

$$\text{PPMI}(word_1, word_2) = \max\left(\log_2 \frac{P(word_1, word_2)}{P(word_1)P(word_2)}, 0\right)$$

پس ابتدا ماتریس PPMI را برای تمام کلمات محاسبه می‌کنیم. اندازه این ماتریس |V|در |V| که در اینجا اندازه ۸۰۰۰ V است.

برای محاسبه باهم آیی یا Co-occur ماتریس زیر را تشکیل داده و به صورت زیر عمل می‌کنیم:

```
[23] # Co-Occur Matrix
     co_occurrence_matrix = np.zeros((len(vocabulary), len(vocabulary)), dtype=int)
     vocab_index = {word: i for i, word in enumerate(vocabulary)}
     for sample in train_data['text']:
         for i, word1 in enumerate(sample):
             for j, word2 in enumerate(sample):
                 if i != j:
                     index_word1 = vocab_index[word1]
                     index_word2 = vocab_index[word2]
                     co_occurrence_matrix[index_word1][index_word2] += 1
```

در این روش دو حلقه تو در تو برای کلمه‌های پشت سر هم تشکیل داده و با توجه به ایندکس آنها یک واحد به ماتریس Co-Occur اضافه می‌شود.

مرحله بعدی، محاسبه احتمال است. احتمال هر کلمه و احتمال هر جفت محاسبه شده تا در مرحله بعدی PPMI محاسبه شود.

```
total_occurrences = co_occurrence_matrix.sum()
word_probabilities = co_occurrence_matrix.sum(axis=1) / total_occurrences
joint_probabilities = co_occurrence_matrix / total_occurrences
```

حال با توجه به احتمالات محاسبه شده، PPMI محاسبه می‌شود:

```
ppmi_matrix = np.zeros_like(joint_probabilities, dtype=float)
for i in range(len(vocabulary)):
    for j in range(len(vocabulary)):
        if co_occurrence_matrix[i][j] > 0:
            pmi = np.log2(joint_probabilities[i][j] / (word_probabilities[i] * word_probabilities[j]))
            ppmi_matrix[i][j] = max(pmi, 0)
```

تا اینجا PPMI به صورت کلمه به کلمه محاسبه شده است. هدف این تمرین محاسبه بردار PPMI برای هر نمونه است.

برای رسیدن به این هدف می‌توان Centroid هر Sample را محاسبه کرد.

```
[40] num_samples = len(train_data['text'])
     num_words = len(vocabulary)
     PPMI_Vectors = np.zeros((num_samples, num_words), dtype=np.float64)

     for sample_idx, sample in enumerate(train_data['text']):
         if sample:
             for word in sample:
                 if word in vocab_index:
                     word_idx = vocab_index[word]
                     PPMI_Vectors[sample_idx] += ppmi_matrix[word_idx]

             PPMI_Vectors[sample_idx] /= len(sample)
```

```
PPMI_Vectors[0,vocabulary.index('need')]
```

```
2.355098565230598
```

در این راه حل Centroid هر Sample بر اساس PPMI_Matrix محاسبه شده‌است. خروجی وکتور سمپل اول هم گرفته شد. مقدار پارامتر کلمه need در این وکتور برابر مقدار بالاست.

۵-۱_ بخش پنجم

برای تنظیم مدل، ابتدا باید وکتورهایی که با آن‌ها قصد آموزش مدل را داریم، برای داده‌های تست نیز محاسبه شوند .طبق بررسی‌های بیشتر متوجه شدم که برای داده‌های تست نباید ماتریس PPMI و TF-IDF مجددا محاسبه شود بلکه باید از ویژگی‌های داده‌های آموزش استفاده کنیم. مشکلی که به وجود می‌آید کلماتی هستند که در داده آموزش هستند ولی در داده تست نیستند. پس در ماتریس‌های تولید شده در مراحل قبلی مقداری برای آن‌ها نداریم.برای بدست آوردن وکتور داده‌های تست باید از ماتریس هایی که با کمک داده های آموزشی تولید کرده ایم استفاده کنیم:

برای OOV یعنی کلماتی که در داده آموزش نداشته‌ایم می‌توانیم صفر در نظر بگیریم.

Vector of Test data based on Train Matrix

```
TF_Matrix_Test = np.zeros((len(test_data), len(vocabulary)), dtype=int)
counter = 0
for sample in test_data['text']:
    for word in sample:
        if word in vocabulary:D
            idx = vocabulary.index(word)
            TF_Matrix_Test[counter, idx] += 1
    counter += 1
TF_Idf_Test = np.where(TF_Matrix_Test > 0, 1 + np.log10(TF_Matrix_Test), 0)
for sample in TF_Idf_Test:
    for word_index in range(len(vocabulary)):
        sample[word_index] *= Idf_Matrix[word_index]

PPMI_Vectors_Test = np.zeros((len(test_data['text']), len(vocabulary)), dtype=np.float64)
for sample_idx, sample in enumerate(test_data['text']):
    if sample:
        for word in sample:
            if word in vocab_index:
                word_idx = vocab_index[word]
                PPMI_Vectors_Test[sample_idx] += ppmi_matrix[word_idx]

        PPMI_Vectors_Test[sample_idx] /= len(sample)
```

در این مسئله برای تحلیل احساسات از طبقه‌بند NB از نوع Multinomial استفاده می‌کنیم. باتوجه به تحقیقات انجام شده این نوع از طبقه بند برای داده‌های متنی استفاده می‌شود. ابتدا بهتر است مقادیر طبقه‌بندی را بر حسب منفی و مثبت به ۰ و ۱ تبدیل کنیم:

```
[45] from sklearn.naive_bayes import MultinomialNB
     train_targets = (train_data['target'] == 4).astype(int)
     test_targets = (test_data['target'] == 4).astype(int)
```

```
train_targets
```

```
861732     1
790821     0
81703      0
956378     1
1389784    1
          ..
1597563    1
1287472    1
885181     1
119197     0
986363     1
```

پس از آن مدل را آموزش می‌دهیم:

```
nb_ppmi = MultinomialNB()
nb_tfidf = MultinomialNB()

nb_ppmi.fit(PPMI_Vectors, train_targets)
nb_tfidf.fit(Tf_Idf, train_targets)

ppmi_predictions = nb_ppmi.predict(PPMI_Vectors_Test)
tfidf_predictions = nb_tfidf.predict(Tf_Idf_Test)
```

در این مسئله از دو وکتور PPMI و Tf-Idf برای آموزش استفاده شده‌است. نتایج به صورت زیر است:

PPMI:

Precision: 0.6835978835978836, Recall: 0.6538461538461539, F1-Score: 0.6683911019141232

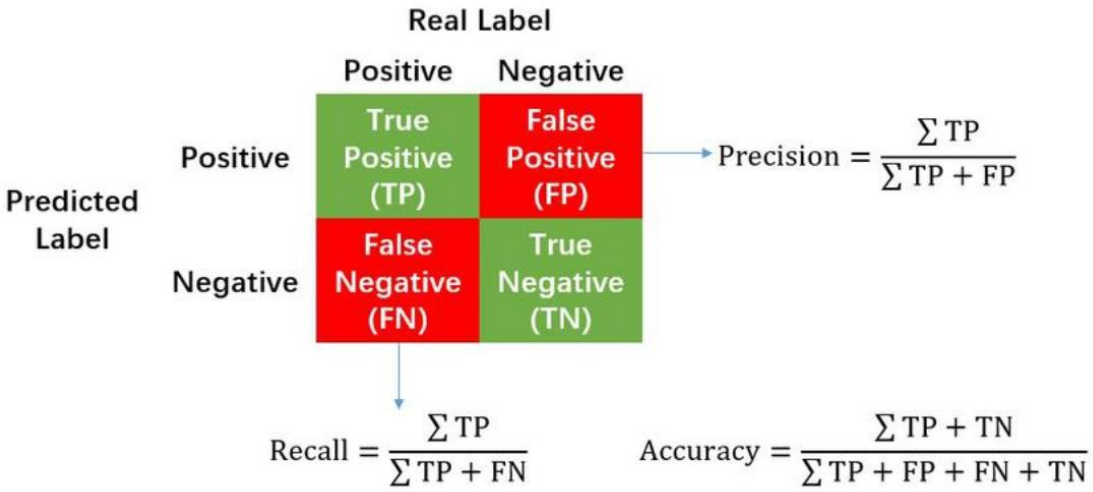
TF-IDF:

Precision: 0.6765714285714286, Recall: 0.5991902834008097, F1-Score: 0.6355340848094472

مشاهده می‌شود که استفاده از PPMI به عنوان بردار ویژگی دقت بالاتری می‌دهد.

حال به بررسی پارامترها می‌پردازیم:

از کتابخانه sklearn برای محاسبه F1-Score، precision و recall استفاده می‌کنیم.



Precision and recall

Precision: % of selected items that are correct
Recall: % of correct items that are selected

Precision همان نسبت تمام موارد مثبتی که درست تشخیص داده شده به موارد مثبت است که درست تشخیص داده شده و یا به اشتباه مثبت تشخیص داده شده‌اند.که در مخرج تمامی مواردی که مدل مثبت تشخیص داده مد نظر است. که این مقدار همان دقت مدل است که در اولین اجرا ۶۸ درصد بود. یعنی ۶۸ درصد مثبت‌هایی که تشخیص داده درست هستند.

Recall یا فراخوانی برابر است با نسبت تعداد موارد مثبتی که درست تشخیص شده به مواردی است که مثبت بوده و درست تشخیص داده و یا منفی بوده و به اشتباه مثبت تشخیص داده است. که در مخرج تمامی موارد مثبت واقعی مد نظر است. که مقدار ۶۵ درصد نشان می‌دهد ۶۵ درصد موارد مثبت واقعی تشخیص داده شده‌اند.

F1-Score یک معیار ارزیابی ترکیبی است که باتوجه به ضریب آن مقدار تاثیر هرکدام تعیین می‌شود.

The traditional F-measure or balanced F-score (**F_1 score**) is the harmonic mean of precision and recall:[2]

$$F_1 = \frac{2}{\mathrm{recall}^{-1} + \mathrm{precision}^{-1}} = 2\frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} = \frac{2\mathrm{tp}}{2\mathrm{tp} + \mathrm{fp} + \mathrm{fn}}.$$

که به نحوی میانگین هارمونیک precision و recall است. این معیار برای زمانی که تعداد نمونه های کلاس ها اختلاف دارد کاربرد دارد. هرچند که در اینجا از هر دسته به طور مساوری نمونه برداشتیم.

برای افزایش دقت میتوان موارد دیگر نیز بررسی شوند، از جمله افزایش اندازه پنجره در Co-Occur ماتریس و یا حذف اعداد در مرحله پیش پردازش. به دلیل کمبود زمان این موارد بررسی نشد.