گزارش پروژه صفر امار و احتمال

البرز محمودیان : 810101514

در ابتدا شش تا دیکشنری ساختیم برای هر کتگوری که بتوانیم کلماتی که در توضیحات و عنوان کتاب های هر کتگوری وجود دارد را در آن نگه داریم که جلوتر توضیح می دهیم که این کلمات چگونه به دست می ایند و از چه فیلتر هایی رد شده اند . سپس یک دیکشنری به نام کل کلمات ساختیم تا تمام کلمات را در ان نگه داریم که در واقع از جمع کلید های شش دیکسنری اولیه ساخته شده است. که می دانیم خاصیت دیکشنری این است که کلید های تکراری نگه نمی دارد پس در واقع کلید های دیکشنری کل کلمات ما عملا یکتا می باشند.

ما ابتدا با خواندن دو فایل csv توسط توابع pandas که در اختیار داشتیم انها را در دو دیتا فریم جداگانه نگه می داریم تا بتوانیم عملیات مورد نظر را روی دیتا فریم ها انجام دهیم. ما با هدف ساختن bow ردیف به ردیف دیتافریم train را می خوانیم و کارهای لازمه را روی متن توضیحات و عنوان هر کتاب انجام می دهیم که عبارت اند از:

1. normalizing کردن متن : توسط متد کتابخانه هضم که فاصله های اضافی را حذف می کند و متن ما را بومی سازی می کند.
2. حذف کردن punctuation ها: لیستی از علایم نگارشی تشکیل دادیم و انها را از رشته خود حذف کردیم.
3. tokenizing کردن که رشته کلمات را در اخر به لیستی از کلمات تبدیل می کند
4. lemmatizing کردن که در بخش امتیازی به ان می پردازیم
5. حذف کردن stop wordها که به ان نیز در بخش امتیازی اشاره خواهیم کرد

سپس بعد ادیت کردن متن خود تشخیص می دهیم که این کتابی که ان را خواندیم به کدام کتگوری تعلق دارد که در همین حین به شمردن کلمه ها در هر کتگوری می پردازیم یعنی کلمه به کلمه متن ادیت شده را به عنوان کلید های دیکشنری خود اضافه می کنیم اگر این کلمه قبلا اضافه شده بود مقدار ان کلید را یکی اضافه می کنیم و اگر در کلید های ما وجود نداشت انرا به عنوان کلید جدید اضافه می کنیم و مقدار انرا 1 قرار می دهیم.

که در نهایت با جمع کردن تمام این 6 دیکشنری می توانیم دیکشنری کل کلمات و bow خود را کامل کنیم

اکنون دیتا های دیتافریم train را خوانده ایم و انها را پردازش کردیم و حالا نوبت فایل test است. که باید کتاب به کتاب بخوانیم و تشخیص دهیم که به کدام کتگوری تعلق دارد که با داشتن 7 دیکشنری اماده شده این کار بسیار راحت می باشد. ابتدا هر کتاب را می خوانیم و با نجام دادن 5 مرحله بالا متن توضیحات و عنوان هر کتاب را اماده می کنیم اکنون کلمه به کلمه در متن هر کتاب جلو می رویم و باید به دست بیاوریم که احتمال وجود این کلمه در کدام کتگوری بیشتر است حالا چگونه؟ ما تعداد تکرار هر کلمه را به عنوان مقدار هر کلید در 6 دیکشنری ذخیره کردیم که نسبت این مقدار به تعداد تکرار کل کلمات یک کتگوری این احتمال را به ما می دهد .

پس به ازای هر کلمه اگر کلمه در bow وجود داشت مقدار کلید ان را تقسیم بر تعداد کل کلمات ان کتگوری می کنیم و اگر وجود نداشت در پاسخ به پرسیش اول خواهیم گفت که از چه روشی استفاده خواهیم کرد و سپس احتمال هر کلمه را در هم ضرب کنیم زیرا که این احتمال ها از هم مساقل هستند. این کار را برای تمام 6 کتگوری حساب می کنیم و سپس هر کتگوری که بیشترین احتمال را داشت ان کتگوری را به عنوان کتگوری کتاب خوانده شده قرار می دهیم.

پاسخ به پرسش اول : اگر کلمه ای به bow تعلق نداشت چه کار کنیم؟

اگر تعداد تکرار های ان را در متن 0 در نظر بگیریم و تقسیم بر تعداد تکرار کل کلمات هر کتگوری کنیم به احتمال صفر خواهیم رسید که با ضرب کردن ان در احتمال های دیگر به عدد 0 خواهیم رسید که این دلخواه ما نیست پس با استفاده از روش additive smoothing می توانیم این مشکل را حل کنیم و با در نظر گرفتن یک مقدار ثابت برای الفا یک احتمال اولیه کوچک برای هر کلمه در نظر بگیریم

$$P(\text{"the"}|Spam) = \frac{N_{\text{"the"}|Spam} + \alpha}{N_{Spam} + \alpha \cdot N_{Vocabulary}}$$

$$P(\text{"secret"}|Spam) = \frac{N_{\text{"secret"}|Spam}}{N_{Spam}}$$

e" are thus given special treatment and their probability are increased

پاسخ به پرسی دوم : می دانیم که احتمال تعلق هر کلمه به هر کتگوری یک عدد دین صفر و یک و تقریبا به صفر نزدیک است و اگر تعداد زیادی از این احتمال ها در هم ضرب شوند به عدد بسیار بسیار کوچکی خواهیم رسید که برای حل این مشکل از احتمال کلی یک لگاریتم می گیریم که به صورت جمع لگاریتم تک تک احتمال هر در میاید که به عددی منفی خواهیم رسید ولی عددی قابل نمایش و قابل مقایسه است

قسمت امتیازی فاز اول:

می دانیم که کلماتی مانند گفت و گو / گفتن / بگو / گفت و ... در متن ما وجود دارد و اگر تمام این هارا بتوانیم کلماتی یکتا در نظر بگیریم در قسمت احتمال گیری به ما می تواند کمک کند که ما با استفاده از تابع lemmatizer که در کتابخانه hazm وجود دارد این کار را انجام می دهیم در واقع کلمات مشابه را به ریشه ی انها تبدیل می کند که باعث می شود که این کلمات همگی یک کلمه ی یکتا حساب شوند و در دیکشنری به عنوان یک کلید حساب شود که در ادامه درصد خروجی را بدون این تابع و با کامنت کردن خط 31 می بینیم:

```
27  def simplify_text(string):
28      normalized_string = normalizer.normalize(string)
29      normalized_string = ''.join(char for char in normalized_string if char not in punctuation)
30      tokenized_list = hazm.word_tokenize(normalized_string)
31      #lemmatized_list = (lemmatizer.lemmatize(" ".join(tokenized_list))).split(" ")
32      final_list = [word for word in tokenized_list if word not in stopwords_list]
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

    list_all_words = make_bag_of_words()
  File "c:\Users\NanoCamp\Downloads\CA0_V1\aaa.py", line 45, in make_bag_of_words
    description_list_words = simplify_text(df_train.loc[row]['description'])
  File "c:\Users\NanoCamp\Downloads\CA0_V1\aaa.py", line 32, in simplify_text
    final_list = [word for word in lemmatized_list if word not in stopwords_list]
NameError: name 'lemmatized_list' is not defined
PS C:\Users\NanoCamp\Downloads\CA0_V1> python -u "c:\Users\NanoCamp\Downloads\CA0_V1\aaa.py"
0.74
PS C:\Users\NanoCamp\Downloads\CA0_V1>
```

قسمت امتیازی فاز دوم:

همانطور که در صورت پروژه گفته شد کلماتی مانند حروف اضافه ، فعل ها و.. وجود دارند که به درد ما نمی خورند و بهتر است که از متن حذف شوند که stopword.list() لیستی از این کلمات را در اختیار ما می دهد که می انهارا از متن خود حذف می کنیم که در ادامه درصد خود را بدون استفاده از این قابلیت خواهیم دید :

```
27  def simplify_text(string):
28      normalized_string = normalizer.normalize(string)
29      normalized_string = ''.join(char for char in normalized_string if char not in punctuation)
30      tokenized_list = hazm.word_tokenize(normalized_string)
31      lemmatized_list = (lemmatizer.lemmatize(" ".join(tokenized_list))).split(" ")
32      #final_list = [word for word in lemmatized_list if word not in stopwords_list]
33      return lemmatized_list
34
35  def get_each_cat_words(categorie, description_list_words, title_list_words, index_category, dict_cat):
36      if(categorie == cats[index_category]):
37          for word in (description_list_words + title_list_words):
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\NanoCamp\Downloads\CA0_V1> python -u "c:\Users\NanoCamp\Downloads\CA0_V1\aaa.py"
0.7533333333333333
PS C:\Users\NanoCamp\Downloads\CA0_V1>
```

قسمت امتیازی فاز سوم:

در بخش اول و دوم امتیازی این گزارش را به همراه عکس انجام دادم و در نهایت درصد کلی من 0.76 درصد بود

```
        count = count + value
    return count

def calculate_probability_this_cat(list_cat_words, test_words):
    final_probability = 0
    word_count_in_dict = sum_dict_values(list_cat_words)
    for word in test_words:
        if word in list_cat_words.keys():
            final_probability = final_probability + math.log(list_cat_words[word] / word_count_in_dict)
        else:
            final_probability = final_probability + math.log(0.01 / word_count_in_dict)
    return final_probability
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\NanoCamp\Downloads\CA0_V1> python -u "c:\Users\NanoCamp\Downloads\CA0_V1\aaa.py"
0.7555555555555555
PS C:\Users\NanoCamp\Downloads\CA0_V1>
```

کد دوم : کد دیگری نیز اماده کردم که زمان اجرای ان کمی بالاتر است ولی دقت بالاتری به همراه دارد که درصد ان 82 درصد می باشد اما چون انرا توسط لیست ها پیاده سازی کردم دسترسی به اعضای لیست و حلقه زدن های متعدد برای پیمایش لیست باعث ایجاد تاخیر در زمان اجرای ان شده است که دیکشنری این خاصیت را نداشت و دسترسی را برای ما راحت کرد و زمان اجرای کد اصلی را توانست به 10 seconds کاهش دهد.

```
list_all_words = []
cats = ["کلیات اسلام", "مدیریت و کسب و کار", "داستان کودک و نوجوانان", "رمان", "داستان کوتاه", "جامعه‌شناسی"]

def simplify_text(string):
    normalized_string = normalizer.normalize(string)
    normalized_string = ''.join(char for char in normalized_string if char not in punctuation)
    tokenized_list = hazm.word_tokenize(normalized_string)
    lemmatized_list = (lemmatizer.lemmatize(" ".join(tokenized_list))).split(" ")
    final_list = [word for word in lemmatized_list if word not in stopwords_list]
    return final_list

def uniqify_list(list):
    x = np.array(list)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\NanoCamp\Downloads\CA0_V1> python -u "c:\Users\NanoCamp\Downloads\CA0_V1\bbb.py"
0.8222222222222222
PS C:\Users\NanoCamp\Downloads\CA0_V1>
```