گزارش:

بخش 1: describing code

-در قسمت ابتدایی ما کتابخانه هایی در طی پروژه لازم داریم را import میکنیم

```
import the libiraries

from hazm import Normalizer, Lemmatizer, POSTagger #import the hazm library
from hazm import word_tokenize
import pandas as pd
import csv
import math
df=pd.read_csv('books_train.csv')#read the data
normalizer = Normalizer()
lemmatizer = Lemmatizer()
posTagger = POSTagger(model = 'pos_tagger.model',  universal_tag = True)
[1]
+ Code  + Markdown
```

-در قسمت بعد ما کار پیش پردازش داده را انجام میدهیم که برای این کا هم باید کارایتر های اضافی رو از متن حذف کرد و هم اعداد را که برای حذف کاراکتر های اضافی ابتدا لیستی از کاراکتر هایی که میخواهم حذف کنم را درست کردم و سپس بار استفاده از تابع (Replace.) در متن بجای انها جای خالی قرار دادم

سپس برای حذف اعداد اومدم طبق Ascii اعداد 0 تا 9 فارسی را به لیست کاراکتر های اضافی اضافه کردم

```
(replace_bad_characters) function is replace the bad character that defined with blank

def replace_bad_characters(s):
    bad_characters = [';', ':', '!', "*", ")", "(", "؛", "\n", ",", "-", ".", "،", "»", "«", "…","[", "]", "\'" ,"؟" ]
    persian_zero_unicode = ord('۰')
    bad_characters += [chr(code) for code in range(persian_zero_unicode, persian_zero_unicode + 10)]
    for bad_character in bad_characters:
        s = s.replace(bad_character, ' ')
    return s
2]
```

-در قسمت بعدی کد ما تابعی به نام (clean_csv_data) تعریف کردیم که این تابع یک فایل csv دریافت میکند و روی کل داده های قسمت (discription) و (title) ان هم عمل normalize و هم عمل پیش پردازش را انجام میدهد (کتگوری هر کتاب نباید تغییر داده شود)

نکته مهمی که وجود دارد این تابع در خروجی یک ارایه دوبعدی از کتاب ها میدهد که هر کتاب خود ارایه ای است مکه درایه اول ان category کتاب هست و درایه دوم ان تمام کلمات استفاده شده در ان کتاب هست

دو بخش امتیازی پروژه رو هم که postagger و lemmatization را هم به این تابع اضافه کردم تا با دادن true یا false به ورودی تابع بتوان از انها استفاده کرد یا استفاده نکرد

```python
def remove_extra_words(words):#a functio that remove all sentence components except nouns and verbs
    word_tags=posTagger.tag(tokens = words)
    result=[]
    for word_tag in word_tags:
        if word_tag[1]=='NOUN' or word_tag[1]=='VERB':
            result.append(word_tag[0])
    return result

def clean_csv_data(df, use_lemmatizer=True, remove_frequent_words=True):
    t=[]
    for i in range(len(df['title'])):
        title = normalizer.normalize(replace_bad_characters(df["title"][i]))
        description = normalizer.normalize(replace_bad_characters(df["description"][i]))
        t.append( [title + " " + description, df["categories"][i]] )

    result=[]
    for text, category in t:
        words=word_tokenize(text)
        clean_words=[]
        if remove_frequent_words==True:
            words=remove_extra_words(words)


        for word in words:
            if use_lemmatizer==True:
                word=lemmatizer.lemmatize(word)

            clean_words.append(word)

        result.append( [clean_words , category] )
    return result
```

Python

-در قسمت بعدی از انجا که برای ساختن ماتریس bow نیاز داریم که تمام کلمات استفاده شده در فایل train رو در دسترس داشته باشیم به همین این قسمت از کد را تعریف کردم که تمام کلمات موجود در فایل train را داخل یک ارایه نگه دارد

making a list that include all words in title and description that exist in train file

```python
all_words=set()
for i in range(len(train_data)):
    item=train_data[i]
    item_words=item[0]
    for j in range(len(item_words)):
        item_word=item_words[j]
        all_words.add(item_word)

all_words = list(all_words)
```

-در این قسمت باید ماتریس bow را بدست اورد اما در ابتدا از انجا که گشتن به دنبال یک کلمه در در ارایه بسیار وقت گیر است از دیکشنری استفاده میکنیم یعنی ارایه عی که تمام کلمات فایل را در خود ذخیره کرده بود به دیکشنری عی تبدیل میکنیم که شامل کلمات و ایندکس های ان است

سپس در ابتدا یک ماتریس خالی که تمام درایه هایش صفر است میسازیم و سپس به اضای تکرار هر کلمه یک واحد به ارایه مربوط به ان کلمه اضافه میکنیم

```
def find_index(item, array):#a function that fine the index of a item in array
    for i in range(len(array)):
        if item == array[i]:
            return i
    return -1

def make_2d_array(n, m):#a function that make a n*m matrix with zero values
    result=[]
    for i in range(n):
        a=[]
        for j in range(m):
            a.append(0)
        result.append(a)
    return result
#making a dictionary that cotain word with its index
word_indexes = {}
for i in range(len(all_words)):
    word = all_words[i]
    word_indexes[word] = i
#making the bow matrix
bow=make_2d_array(len(all_categories), len(all_words))
for book in train_data:
    category=book[1]
    category_index=find_index(category, all_categories)
    for word in book[0]:
        word_index=word_indexes[word]
        bow[category_index][word_index]+=1
```

-حال از انجا که ماتریس bow ساخته شد برای ساده تر شدن مراحل اومدم ماتریس bow رو به ماتریس احتمال وجود هر کلمه در هر کدام از ان category ها تبدیل میکنیم که در بدست اوردن احتمال از قانون بیز راحت تر باشیم

همچنین با توجه به پرسش دوم برای در صورت ضرب شدن احتمال های کوچک جواب به صفر میل میشود به همین دلیل از لوگاریتم استاده میکنیم تا ضرب را به جمع تبدیل کنیم

اینجا ماتریس بوی را تبدیل به احتمال وجود هر کلمه در ان کتگوری میکنیم در ارایه جدید به ازای هر کتگوری یک احتمال برای عدم وجود هم اضافه میکنیم

```
probibility_bow = make_2d_array(len(all_categories) , len(all_words) + 1)
for i in range(len(probibility_bow)) :
    sum_row = 0

    #calculate sum_row
    for j in range(len(all_words)):
        sum_row += bow[i][j]

    for j in range(len(all_words)):
        probibility_bow[i][j]=math.log((bow[i][j]+1)/(len(all_words)+1+sum_row)) # Additive Smoothing algorithm
    probibility_bow[i].append(math.log(1/(len(all_words)+1+sum_row))) # Additive Smoothing for words that does not exist
```

[8]

-در این قسمت از انجا که در فرمول بیز به احتمال هر کدام از کتگوری ها در کل فایل نیاز داریم یه تابع برای بدست اوردن احتمال ان کتگوری تعریف میکنیم

پیدا کردن احتمال هر کتگوری

```
def category_probability(category):
    tekrar = 0
    for book in train_data:
        if book[1] == category:
            tekrar += 1
    return tekrar / len(train_data)
```

-در این قسمت از انجا که ما مقدار احتمال هرکدام از کلمه ها در هر کتگوری داریم پس:

در ابتدا تابعی تعریف میکنیم که یک کلمه و یک کتگوری بگیرد و در ماتریس probability-bow احتمال ان کلمه را اگر در کلمات وجود داشت بدهد و اگر وجود نداشت احتمالی که در ماتریس probability-bow برای کلماتی که وجود ندارند که بر اساس additive smoothing بدست اوردیم را بدهد

حال یک تابع دیگر تعریف میکنیم که یک کتاب و یک کتگوری دریافت کند و سپس در کلمات اون کتاب پیمایش کنه و سپس احتمال کلمات موجود در ان کتگوری را با توجه به تابع قبلی که تعریف کردیم بدست بیاره و با هم جمع کنه و و در اخر هم با لگاریتم احتمال اون کتگوری در فایل test جمع کنه

حال تابع دیگری را تعریف میکنیم که اطلاعات یک کتاب را بگیرد و سپس احتمال بودن ان کتاب در کتگوری های مختلف به وسیله تابعی که در قسمت قبل تعریف کردیم بدست اورده و بزرگترین احتمال را پیدا کرده و کتگوری ان را به عنوان خروجی بدهد

در قسمت اخر هم تابعی تعریف کردم که یک فایل دریافت کنه و سپس با استفاده از تابع بالا کتگوری که احتمال بشتری در ان کتاب داشته با کتگوری واقعی اون کتاب مقایسه میکنه و سپس دقت رو به عنوان خروجی بدهد

```python
def word_probability_in_category(category, word):#a functio that return the probability of word in given category from probability_bow
    category_index= all_categories.index(category)#find category index in all_category array
    if word in word_indexes:
        word_index=word_indexes[word]#find word index in all_words dictionary
        return probibility_bow[category_index][word_index]
    else:
        return probibility_bow[category_index][-1]#when the word does not exist in train file

def find_book_category_probability(book , category):#find the probability of each category in each book of test file
    probability=0
    for word in book[0]:
        probability+=word_probability_in_category(category , word )
    return probability+math.log(category_probability(category))

def find_best_category(book):#a function that find the category with highest probability
    best_category = ''
    best_probability=None
    for i in range(len(all_categories)):
        category_probability=find_book_category_probability(book , all_categories[i])
        if  best_probability==None or category_probability > best_probability:
            best_probability = category_probability
            best_category=all_categories[i]
    return best_category

def run_test(test_data):#this function show the probability of correctness
    true_valu=0
    for book in test_data:
        best_category=find_best_category(book)
        if best_category == book[1]:
            true_valu +=1

    print("the probability of correctness is:",true_valu/(len(test_data)))
```

بخش2 :پاسخ به پرسش اول:

از انجا که با صفر درنظر گرفتن احتمال برای کلمه ای که وجود ندارد حاصل ضرب احتمال ها صفر میشود میایم از پاسخ پرسش دوم استفاده میکنیم تا ضرب را به جمع تبدیل کنیم اما در این صورت هم دقت به شدت کاهش پیدا میکند (دقت در بخش 4)

بنابر این برای افزایش دقت کافیست از تکنیک additive smoothing استفاده میکنیم در حالت کلی در این روش ما میایم به تعداد کل اعداد موجود در ماتریس 1 واحد اضافه میکنیم و اینجوری برای کلمه ای که در متن وجود ندارد احتمال $\frac{1}{\text{مجموع کل تکرار ها+ تعداد کل کلمات}}$ در نظر میگیریم و برای کلماتی که وجود دارند احتمال $\frac{\text{تعداد تکرار کلمه}+1}{\text{تعداد کل کلمات+مجموع کل تکرار ها}}$ را در نظر میگیریم با این کار دقت افزایش پیدا میکند

بخش 3 : پیاده سازی ساده(without additive smooth)

```
clean the test_file and show the result

df_test=pd.read_csv('books_test.csv')

test_data=clean_csv_data(df_test, False, False)
# show_guesses(test_data)
run_test(test_data)
[44] ✓ 1.4s
... the probability of correctness is: 0.006666666666666667
```

همونطور که در تصویر دیده میشود دقت در این حالت 0.006 است که خیلی کم است

بخش4 :پیاده سازی additive smooth

```
clean the test_file and show the result

df_test=pd.read_csv('books_test.csv')

test_data=clean_csv_data(df_test, False, False)
# show_guesses(test_data)
run_test(test_data)
[28] ✓ 4.2s
... the probability of correctness is: 0.7933333333333333
```

در این حالت همونطور که مشاهده میکنید دقت بسیار افزایش پیدا کرد و به عدد 0.79 رسید

بخش5 :پاسخ به پرسش دوم:

در شرایطی که متن توضیحات یک کتاب طولانی باشد در اینصورت با ضرب شدن احتمال های هر کلمه که مقدار کوچکی است جواب به صفر میل میکند به همین دلیل برای جلوگیری از این اتفاق میتوان با استفاده از لگاریتم ضرب را به جمع تبدیل کنیم و از انجایی که از احتمال فقط برای مقایسه استفده میکنیم همچین کاری امکان پذیر است

بخش 6 : فاز امتیازی

- قسمت اول:

در این قسمت من برای بدست اوردن ریشه کلمات از تابع lemmatization از کتابخانه هضم استفاده کردم

استفاده از lemmatization در حالت ساده:

```
clean the test_file and show the result

df_test=pd.read_csv('books_test.csv')

test_data=clean_csv_data(df_test, True , False)
# show_guesses(test_data)
run_test(test_data)
[56] ✓ 1.6s
... the probability of correctness is: 0.024444444444444446
```

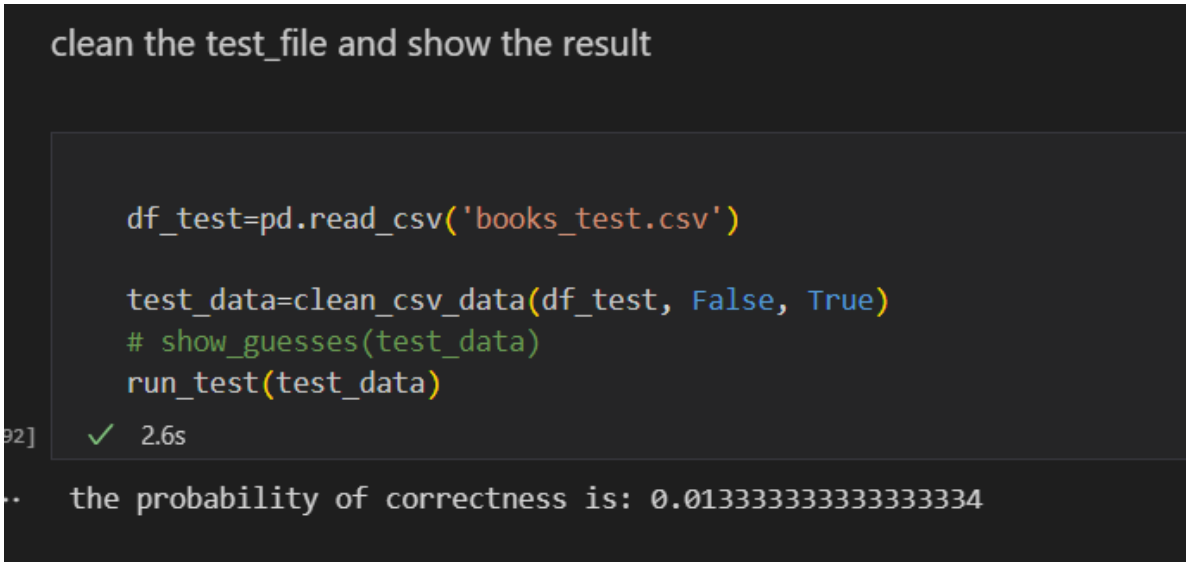
استفاده از lemmatization به همراه additive smoothing:

```
clean the test_file and show the result

df_test=pd.read_csv('books_test.csv')

test_data=clean_csv_data(df_test, True , False)
# show_guesses(test_data)
run_test(test_data)
8] ✓ 1.7s
. the probability of correctness is: 0.7955555555555556
```

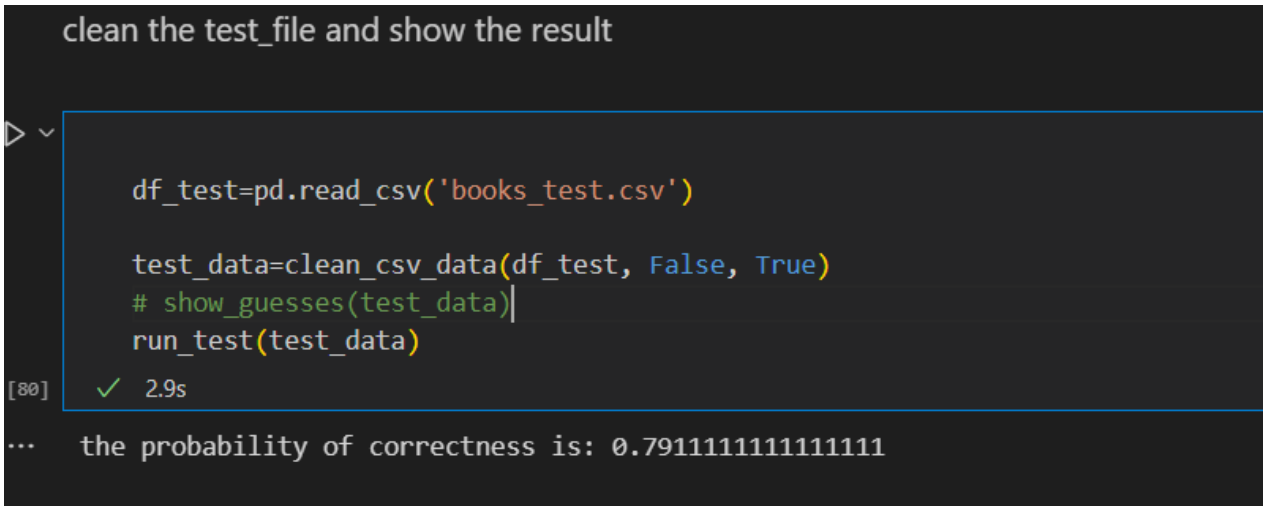
- قسمت دوم:

برای این قسمت هم من اومدم از تابع POSTagger از کتابخانه هضم استفاده کردم تا این حروف را حذف کند

استفاده از POSTagger در حالت ساده:



استفاده از POSTagger به همراه additive smoothing:



- قسمت سوم:

استفاده از هر دو در حالت ساده:

clean the test_file and show the result

```
df_test=pd.read_csv('books_test.csv')

test_data=clean_csv_data(df_test, True , True)
# show_guesses(test_data)
run_test(test_data)
```

04] ✓ 2.7s

```
the probability of correctness is: 0.04
```

استفاده از هر دو به همراه additive smoothing:

clean the test_file and show the result

```
df_test=pd.read_csv('books_test.csv')

test_data=clean_csv_data(df_test, True , True)
# show_guesses(test_data)
run_test(test_data)
```

116] ✓ 2.9s

```
the probability of correctness is: 0.8044444444444444
```

همونطور که مشاهده میکنید در این حالت بهترین دقت را داشتیم

با تشکر از توجه شما ☺