به نام خدا

تمرین کامپیوتری صفر درس آمار و احتمال مهندسی

مجید صادقی نژاد – ۸۱۰۱۰۱۴۵۹

- نکات

کد اصلی پروژه google colab نوشته شد و آدرس آن در زیر آمده است پیشنهاد می شود برای تجربه بهتر کد در همین محیط اجرا شود هر چند که فایل ها .ipynb , .py که ذخیره شده از همین پروژه در colab هستند هم در ضمیمه فایل های تحویل شده آمده اند **نکته قابل توجه است که در صورت اجرا کد در google colab در ابتدا باید دو فایل تست و تمرین را در آن محیط بارگذاری کنید ...**

آدرس :

https://colab.research.google.com/drive/۱pdyRxX۳fcYG۸_D۹qVSUn۳hGgEQTIHPT۸?usp=sharing

- بخش اول : پیش پردازش

در ابتدا اگر برنامه داخل google colab اجرا می شود بخش اول برای نصب کتابخانه ی هضم است در صورتی که برنامه جای دیگر اجرا شود نیاز به ران کردن این بخش از کد نیست و از قابل داخل فایل های آپلود شده پاک شده است.

در بخش بعد چند کتابخانه شامل «هضم برای پردازش متن» ، «کالکشن برای شمارش کلمات و تشکیل BoW » ، «پانداس برای کار با دیتا ها» و « numpy برای محاسبات» افزوده می شوند و همچنین چند تابع نیز تعریف می شوند توضیح هر کدام از این توابع در مراحل بعد هنگام استفاده از تابع توضیح داده می شود

```
import numpy as np
import pandas as pd
import re
import math
from hazm import *
from collections import Counter

normalizer = Normalizer()
stemmer = Stemmer()
lemmatizer = Lemmatizer()

dfTrain = pd.read_csv('books_train.csv')
dfTest = pd.read_csv('books_test.csv')
```

کتابخانه ها و خواندن فایل

در بخش بعدی توضیحات کتاب ها از دو فایل Train , Test استخراج شده و سپس به تابع cleaning داده می شود این تابع همان تابع اولیه برای پیش پردازش است که کاراکتر های بی اثر در پردازش را حذف می کند

این تابع در ابتدا کاراکتر های "»" ، "«" ، "(" ")" ، "،" ، "." ، ":" ، "-" ، "_" ، " " ، "؟" ، "!" ، "؛" ، "*" حذف می شوند این کاراکتر ها تاثیری در پردازش کلمات ندارند و فقط باعث تولید کلمات اضافه می شوند. سپس با تعریف یک ترجمه که از توابع داخلی پایتون است تمامی اعداد فارسی را به انگلیسی تبدیل کرده و سپس تمام اعداد را از متن حذف می کنیم ، حال با استفاده از تابع نرمالایز هضم اعراب از کلمات حذف شده و تا حد ممکن فاصله ی بین کلمات اصلاح می شود البته لازم به ذکر است که کتابخانه هضم بسیار قدرتمند نبوده و فاصله گذاری بسیاری از کلمات اصلاح نمی شود برای مثال کتابخانه ی هضم توانایی تبدیل دو کلمه ی "مردمنگارانه" و "مردم نگارانه" را به یک حالت واحد نداشته و همچین توانایی تبدیل کلمه ی "کتابخانهای" به "کتابخانه ای" را نیز ندارد.

```
def cleaning(SentenceList):

  removes = ['.', ':', '،', '(',')','«', '»','"','-','!','[',']', '؛','_','*','است','بود' ]
  signTrans = str.maketrans("", "", ''.join(removes))

  SentenceList = [item.translate(signTrans) for item in SentenceList]

  numberTrans = str.maketrans("۰۱۲۳۴۵۶۷۸۹", "0123456789")
  SentenceList = [item.translate(numberTrans) for item in SentenceList]
  SentenceList = [re.sub(r'\d', '', item) for item in SentenceList]

  SentenceList = [normalizer.normalize(item) for item in SentenceList]

  return SentenceList
```

تابع برای پیش پردازش جملات

حال که جملات توضیحات دو فایل تمرین و تست پیش پردازش شده اند هر کدام باید در لیستی جداگانه نگهداری شوند به نام لیست تمرین و لیست تست که هر لیست شامل جملات فایل توضیحات هر فایل است، حال از آنجا که برای تشکیل BoW نیاز به کلمات یکتای لیست تمرین داریم کلمات را ابتدا تمام جملات لیست تمرین را به هم متصل می کنیم سپس با استفاده از تابع word tokenize کتابخانه ی هضم کلمات را استخراج کرده حال با استفاده از تابع set پایتون موارد تکراری را از این کلمات را حذف می کنیم حال کلمات یکتا را در یک لیست داریم.

```
desTrainList = cleaning(dfTrain['description'].tolist())
desTestList = cleaning(dfTest['description'].tolist())

wholeText = ''.join(desTrainList)
wholeWords = word_tokenize(wholeText)
wholeWords = list(set(wholeWords))
```

استخراج کلمات یکتا

- بخش دوم : ساخت BoW

در بخش بعدی bow را می سازیم از آن جا که ۶ دسته بندی داریم پس باید ۶ لیست به اندازه تعداد کلمات یکتا داشته باشیم که هر کدام از این لیست به وسیله ی تابع تعریفی BowMaker ساخته می شود:

این تابع در ابتدا به وسیله ی کتابخانه ی پانداس تمام توضیحاتی که دارای «دسته بندی داده شده» (که در آرگومان تابع داده می شود) هستند را از فایل تمرین استخراج کرده و تابع cleaning را روی آن اعمال کرده تا موارد اضافی حذف شوند و نرمالایز

شود سپس تمام توضیحات به یکدیگر متصل می شوند و در نهایت با استفاده از کتابخانه ی هضم کلمات را از این متن ساخته شده استخراج می کنیم و لیستی از کلمات توضیحات مربوط به دسته بندی داده شده داریم حال با استفاده از تابع counter تعداد تکرار هر کلمه از کلمات یکتا در لیست کلمات دسته بندی شمرده می شود در لیستی ثبت می شود بدین معنی که اندیس صفرم لیست تعداد تکرار کلمه صفرم کلمات یکتا در دست بندی (برای مثال: کلیات اسلام) است و بدین ترتیب شیش لیست به طول تعداد کلمات یکتا ساخته می شود که در واقع همان BoW مسئله است.

```
def BowMaker(category, wholeWords):
    wordsList = word_tokenize(''.join(cleaning(dfTrain[dfTrain['categories'] == category]['description'].tolist())))
    wordCounts = Counter(wordsList)
    counts = [wordCounts[word] for word in wholeWords]
    return counts
```

```
islam = BowMaker('کلیات اسلام', wholeWords)
story = BowMaker('داستان کوتاه', wholeWords)
novel = BowMaker('رمان', wholeWords)
kids = BowMaker('داستان کودک و نوجوانان', wholeWords)
job = BowMaker('مدیریت و کسب و کار', wholeWords)
sociaty = BowMaker('جامعه‌شناسی', wholeWords)
```

تابع سازنده کیسه ی کلمات و تشکیل کیسه کلمات برای دسته های مختلف

- بخش سوم: محاسبه احتمال با قانون بیز

در مرحله بعد ابتدا جواب های مسئله که در واقع همان دسته بندی های کتاب در فایل تست هستند تبدیل به عدد می شوند بدین صورت که به هر دسته یک عدد داده شده و در نهایت در یک لیست به تعداد کتاب های فایل تست عدد از یک تا شش داریم حال به سراغ محاسبه بدون additive smoothing می رویم همونطور که در بخش ابتدایی فایل توضیحات پروژه پروژه آمده بود از آنجا که قصد ما مقایسه ی چند احتمال است پس لزومی به محاسبه مخرج کسر نیست و از آنجا که تعداد کتاب های هر دسته در فایل تست برابر است (برابر ۷۵ تاست) پس احتمال c نیز برای تمامی دسته ها برابر بوده و نیازی به وارد شدن در محاسبات ندارد.

$$P(c|X) \propto P(x_1|c) \times P(x_2|c) \times \ldots \times P(x_n|c) \times P(c)$$

- پرسش ۱ :

اگر احتمال کلماتی که در کیسه ی کلمات یک دست وجود ندارند را برابر صفر بگیریم به علت وجود ضرب میان احتمالات احتمال نهایی برابر صفر می شود که اشتباه است پس به ناچار آن کلمات را در نظر نمی گیریم که در نظر نگرفتن این کلمات هم سبب کم شدن شدید دقت محاسبات می شود ! زیرا دسته ای که کلمات توضیحات کتاب مورد بررسی به تعداد بیشتری در آن یافت می شود (فقط خود کلمات و نه تکرار آن ها) دارای تعداد بیشتری کسر می شود و بقیه دسته ها دارای تعداد کمتری کسر زیرا ما آن ها را در نظر نگرفتیم حال ضرب این تعداد زیاد کسر در هم سبب کم شدن احتمال دسته ی درست می شود ...

حال با این توضیحات بدون در نظر گرفتن کلمات ناموجود در کیسه ی کلمات احتمال را حساب کرده :

```
category1 = []
sums = [sum(islam),sum(story),sum(novel),sum(kids),sum(job),sum(sociaty)]

for sentence in desTestList:
  chance = [[] for _ in range(6)]
  words = word_tokenize(sentence)

  for word in words:
    if word in wholeWords:
      indexInWhole = wholeWords.index(word)
      if islam[indexInWhole] != 0:
        chance[0].append(math.log((islam[indexInWhole])/(sums[0])))
      if story[indexInWhole] != 0:
        chance[1].append(math.log((story[indexInWhole])/(sums[1])))
      if novel[indexInWhole] != 0:
        chance[2].append(math.log((novel[indexInWhole])/(sums[2])))
      if kids[indexInWhole] != 0:
        chance[3].append(math.log((kids[indexInWhole])/(sums[3])))
      if job[indexInWhole] != 0:
        chance[4].append(math.log((job[indexInWhole])/(sums[4])))
      if sociaty[indexInWhole] != 0:
        chance[5].append(math.log((sociaty[indexInWhole])/(sums[5])))

  chances = [sum(chance_i) for chance_i in chance]
  category1.append(chances.index(max(chances))+1)

result = [a == b for a, b in zip(answer, category1)]

print(category1)
print(answer)
print((result.count(True)/450)*100)
```

الگوریتم محاسبه احتمال و تعیین دسته بندی بدون additive smoothing

- پرسش ۲:

از آنجا که احتمال ها اعداد بسیار کوچکی هستند و تعداد بالایی دارند در صورت ضرب شدن در یکدیگر منجر به جواب صفر می شوند که احتمال اشتباهی است پس با اعمال لگاریتم به دو طرف رابطه ضرب را به جمع اعمال کرده و در نهایت بزرگترین عدد را انتخاب می کنیم که همان جواب است (همه ی اعداد منفی هستند پس عددی بزرگتر است که به صفر نزدیک تر باشد)

حال با محاسبه احتمال ها و لگاریتم گیری از آن ها و در نهایت تولید جواب نهایی بدون additive smoothing به درصد دقت ۴ % می رسیم که درصد بسیار پایینی است که علت این اتفاق در بالا توضیح داده شد.

```
[4, 4, 4, 1, 5, 4, 4, 4, 4, 4, 4, 4, 4
[2, 2, 1, 3, 4, 1, 1, 2, 6, 1, 5, 5, 2
4.0
```

خط اول دسته بندی جواب ، خط دوم دست بندی به دست آمدی از بیز، خط سوم درصد تطابق

حال با استفاده از additive smoothing که فرمول آن در زیر آمده است محاسبات احتمال را انجام می دهیم

$$\hat{P}(x_i \mid \omega_j) = \frac{N_{x_i,\omega_j} + \alpha}{N_{\omega_j} + \alpha d} \quad (i = (1, \ldots, d))$$

where

- N_{x_i,ω_j}: Number of times feature x_i appears in samples from class ω_j.
- N_{ω_j}: Total count of all features in class ω_j.
- α: Parameter for additive smoothing.
- d: Dimensionality of the feature vector $\mathbf{x} = [x_1, \ldots, x_d]$.

برای مقدار a بهترین عدد ۰.۱ است که این عدد با آزمون و خطا یافت می شود و از آنجا که دقت برای اعداد بالاتر از این عدد به شدت افت می کند و برای اعداد کمتر از این عدد دقت با شیب کمی کم می شود پس بهترین انتخاب ۰.۱ است.

```
category = []
a= 0.1
sums = [sum(islam),sum(story),sum(novel),sum(kids),sum(job),sum(sociaty)]

for sentence in desTestList:
  chance = [[] for _ in range(6)]
  words = word_tokenize(sentence)
  wordsNum = len(words)


  for word in words:
    if word in wholeWords:
      indexInWhole = wholeWords.index(word)
      chance[0].append(math.log((islam[indexInWhole]+a)/(sums[0]+a*wordsNum)))
      chance[1].append(math.log((story[indexInWhole]+a)/(sums[1]+a*wordsNum)))
      chance[2].append(math.log((novel[indexInWhole]+a)/(sums[2]+a*wordsNum)))
      chance[3].append(math.log((kids[indexInWhole]+a)/(sums[3]+a*wordsNum)))
      chance[4].append(math.log((job[indexInWhole]+a)/(sums[4]+a*wordsNum)))
      chance[5].append(math.log((sociaty[indexInWhole]+a)/(sums[5]+a*wordsNum)))

  chances = [sum(chance_i) for chance_i in chance]
  category.append(chances.index(max(chances))+1)

result = [a == b for a, b in zip(answer, category)]

print(category)
print(answer)
print((result.count(True)/450)*100)
```

الگوریتم محاسبه احتمال و تعیین دسته بندی با additive smoothing

حال بر اساس این فرمول محاسبات را انجام داده و لگاریتم گیری می کنیم و دقت ۸۰/۴ % را به دست می آوریم

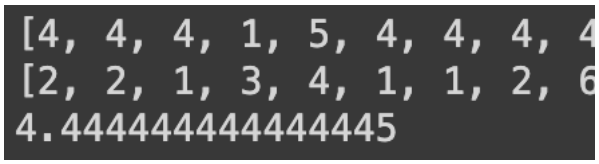
```
[2, 2, 1, 3, 4, 6, 1, 2, 6, 1, 5
[2, 2, 1, 3, 4, 1, 1, 2, 6, 1, 5
80.44444444444444
```

خط اول دسته بندی جواب ، خط دوم دست بندی به دست آمدی از بیز، خط سوم درصد تطابق

- امتیازی قسمت اول :

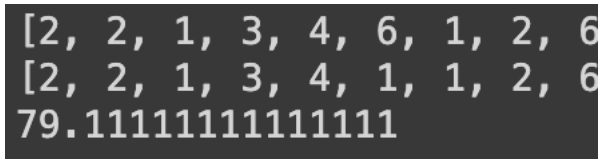
با توجه به مطالب خود کتابخانه ی هضم تابع lemmetizer قوی تر از تابع stemmer عمل می کند زیرا stemmer فقط بر اساس تعدادی کلمه ی از پیش تعیین شده اما lemmetizer ریشه هر کلمه را بررسی می کند به همین علت ما تابع lemmetizer را در تمامی قسمت های کد اعمال می کنیم و نتیجه را مشاهده می کنیم:

بدون روش additive :



که شاهد افزایش دقت هستیم

با روش additive :



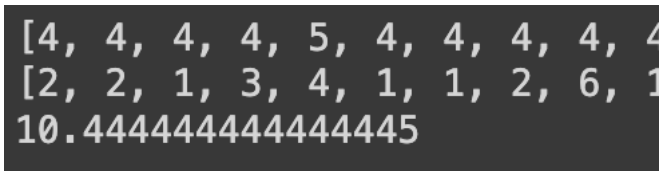
که شاهد کاهش بسیار کم دقت هستیم

اما نکته جالب این است که این روش به سبب کاهش کلمات تکرار سرعت الگوریتم ها را در هر دو حالت به شدت افزایش می دهد به طوری که زمان اجرای هر کدام نصف می شود !

- امتیازی قسمت دوم :

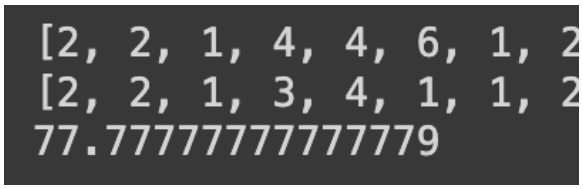
با افزودن تعداد کمی از کلمات و ترکیب ها مانند در ، از ، که ، شد به لیست حذفی ها می توان نتایج زیر را مشاهده کرد

بدون روش additive :



که شاهد افزایش دقت هستیم

با روش additive :



که شاهد کاهش دقت هستیم

با افزودن کلمات بیشتر به لیست حذفی ها می توان همین روند را مشاهد کرد البته کاهش دقت روش additive بسیار زیاده نبوده و دقت را در محدوده ۷۰ الی ۸۰ نگه می دارد اما مجددا مانند قسمت بالا اعمال این حذفی ها تاثیر بالایی در سرعت اجرا دارد

- **امتیازی قسمت سوم :**

می توان مشاهده کرد که با اعمال مورد اول تعداد کلمات کیسه ی کلمات کمتر شده و سرعت کد افزایش می یابد اما به سبب ضعف کتابخانه ی هضم تعدادی از کلمات از حالت اصلی خود خارج می شوند در نهایت با کاهش کلمات روش اول عملکرد بهتری نشان می دهد اما روش دوم از آنجایی که تقریبا به بالاترین دقت خود رسیده بود تاثیر زیادی از این عمل نمی برد و برای بخش دوم به علت حذف تعدادی کلمه و کاهش کلمات یکتای کل مجددا این توضیحات اتفاق می افتد با این تفاوت که در روش دوم تعداد کلمات یکتا به میزان بیشتری کاهش می یابد