مهدی فقهی

401722136

هومورک چهارم nlp

Edit Distance And Spell Correction (30 points)

(a) (5 points)

State the types of spelling errors. Give a brief explanation and ways to recognize each of them.

Non-word Errors :

مثلا کلمه گیسو را گیصو بنویسم . تنها را شناخت این نوع از خطا این هست که ببینم در دایره لغات ما هستند یا نه هرچه دایره لغات ما بزرگتر شود توانایی ما برای شناخت این نوع خطا بهتر خواهد شد . برای تصحیح نیز تمام کلماتی که به کلمه موردنظر ما نزدیک هستند در دامنه لغات پیدا می کنیم از بین آن‌ها یا

آنکه Shortest weighted edit distance نسبت به کلمه اشتباه را دارد انتخاب می کنیم یا براساس Highest noisy channel probability کلمه موردنظر را پیدا می کنیم .

Real-word Errors :

Typographical errors

کلمه‌هایی که شباهت تایپی دارند مثل سنجاب و سنجاق .

بر

Cognitive Errors (homophones)

کلمه‌هایی که شباهت در گفتار دارند مثل خار و خوار .

هر دو آن‌ها از نوع نوشتار درست هستند ولی هیچ کدام نمی‌توانند به جای دیگری به کار بروند و در صورت که در جای درست خود به کار نروند باعث بوجود آمدن خطا نوشتاری می‌شوند .

برای در امان ماندن از این خطا

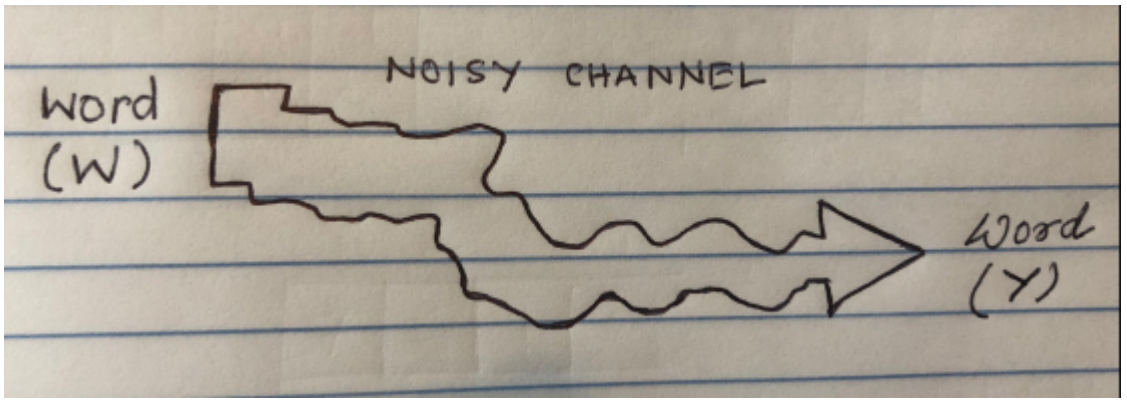
کلمات نامزد با تلفظ مشابه را پیدا کنید

کلمات نامزد با املای مشابه را پیدا کنید

از بین این موارد به کمک Noisy Channel Classifier بهترین کلمه را انتخاب می کنیم که بهتر است در آن جمله به کار رود .

(b) (5 points)

What is a noisy channel and explain how it works:



در این مدل ما یک کلمه ورودی W که فکر می‌کنیم دارای خطایی هست که در بالا ذکر کردیم را به عنوان ورودی وارد کانال می‌کنیم و سپس کلمه مثل y که بهترین تخمین از کلمه W است را به عنوان خروجی می‌گیریم .

حال سوالی که پیش می‌آید این است که چگونه به همچین خروجی دست می‌یابیم .

این مدل احتمالی است که در آن از قانون بیز برای تعیین احتمال کلمه W با توجه به کلمه Y استفاده می کنیم.

$$\hat{w} = \underset{w \in V}{\operatorname{argmax}} P(w \mid x)$$

$$= \underset{w \in V}{\operatorname{argmax}} \frac{P(x \mid w)P(w)}{P(x)}$$

$$= \underset{w \in V}{\operatorname{argmax}} P(x \mid w)P(w)$$

همانطور که در فرمول‌ها مشاهده می‌کنید در اینجا P(w) مدل زبانی است که احتمال w بودن کلمه را به ما می گوید.

P(x | w) در مدل noisy channel احتمال اینکه متن اصلی w با توجه به x باشد را به ما می گوید.

برای ساخت یک نویزی چنل ابتدا باید یک سری کاندیدا برای تغییر کلمه داده شده به کلمه معقول داشته باشیم .

۱ . کلمه‌هایی با نزدیکترین شکل تلفظ به کلمه موردنظر .

۲. کلمه با نزدیکترین شکل نوشتار به کلمه موردنظر

برای اینکه کلمه‌های ما به شکل نزدیک کلمه باشد فقط سعی می‌کنیم کلمه ها را انتخاب کنیم که از طریق شیوه زیر کلمه ما را حاصل کند .

۱. از طریق اضافه کردن یک حرف .

۲ .حذف یک حرف

۳ . تغییر یک حرف به یک حرف دیگر

۴ . جابه جا کردن دو حرف در کلمه

Insertion

Deletion

Substitution

Transposition of two adjacent letters

سپس بعد از پیدا کردن جدول مربوط به این قسمت که نشان می‌دهد از انجام هر کدام یک از این کارها چه کلماتی حاصل می‌شوند . احتمال هر کدام از این رخ دادها را محاسبه می‌کنیم سپس حاصل بدست آمده را در احتمال وجود همچین کلمه‌ای ضرب می‌کنیم و کلمه با بالاترین رخ داد انتخاب ما خواهد بود.

Candidate Correction	Correct Letter	Error Letter	x|w	P(x|word)	P(word)	10^9 *P(x|w)P(w)
actress	t	-	c|ct	.000117	.0000231	2.7
cress	-	a	a|#	.00000144	.000000544	.00078
caress	ca	ac	ac|ca	.00000164	.00000170	.0028
access	c	r	r|c	.000000209	.0000916	.019
across	**o**	**e**	**e|o**	**.0000093**	**.000299**	**2.8**
acres	-	s	es|e	.0000321	.0000318	1.0
acres	-	s	ss|s	.0000342	.0000318	1.0

برای مثال در اینجا کلمه across به عنوان بهترین کاندیدا از بین بقیه کاندیدها انتخاب شده است .

همچنین برای گرفتن نتیجه بهتر می‌توانیم حاصل بدست آمده هر کلمه را در احتمال bigram همراه با کلمه بعد از آن و قبل از آن ضرب کنیم و ببینم احتمال آنکه در بین این کلمات قرار بگیرد چقدر است و از بین بالاترین کاندیدا آن که احتمال بیشتری در جمله دارد آن را انتخاب کنیم .

```
P(actress|versatile)=.000021 P(whose|actress) = .0010
P(across|versatile) =.000021 P(whose|across) = .000006

P("versatile actress whose") = .000021*.0010 = 210 x10^-10
P("versatile across whose")  = .000021*.000006 = 1 x10^-10
```

همانطور که می‌بینید با آنکه across در قسمت قبل بالاترین احتمال حضور بین کاندیدا ها را داشت ولی نسبت به کاندیدای نزدیک به آن یعنی actress بعد از بررسی bigram در جمله بالا به این نتیجه رسیدم که انتخاب actress انتخاب بهتری است .

(c) (20 points)

1) The first step is to implement a hamming distance algorithm. Hamming distance function gets two strings with equal lengths and checks the corresponding character in each string to find mismatches. input and output of your function should be like this. (2 points)

```python
def hamming_distance_algorithm(string_one:str, string_two:str):

    # If the size of two string aren't same it raise erre .
    if len(string_one) != len(string_one):
        raise "string must have same size"

    # inistion hamming dis at first zero.
    number_hamming_distance = 0

    # move on string one if in each place not same increase numbe of hamming dis

    for index in range(len(string_one)):

        if string_two[index] != string_one[index]:
            number_hamming_distance += 1

    return number_hamming_distance


hamming_distance_algorithm("GGGCCGTTGGT","GGACCGTTGAG")
```

```
3
```

برابر است با string میگیریم فاصله بین دوتا string است دوتا hamming distance algorithm به کمک این فانکشن که یکسان string تفاوت کارکترهایی که یکسان نیستند در دو .

2) In the next step, we give you a pattern, DNA string, and an integer as the maximum number of mismatches (provided in the assignment file as "prob1 c2.txt") and your function has to return all starting positions where Pattern appears as a substring of DNA string with at most d mismatches. (5 points)

```
def find_position_of_sub_string(pattern, text, number_of_mistake = 0):

    size_of_pattern = len(pattern)
    size_text = len(text)
    list_of_index = []

    for index in range(size_text):

        # if we position of index add by size of text and was bigger than this size of ma
        if size_of_pattern+index <= size_text:

            sample_sub = text[index:size_of_pattern+index]
            if hamming_distance_algorithm(pattern,sample_sub) <= number_of_mistake:
                list_of_index.append(index)

    return list_of_index
```

```
pattern_1 = "ATTCTGGA"
text_1 = "CGCCCGAATCCAGAACGCATTCCCATATTTCGGGACCACTGGCCTCCACGGTACGGACGTCAATCAAAT"
number_of_mistake_1 = 3
print(find_position_of_sub_string(pattern_1,text_1,number_of_mistake_1))
```

```
[6, 7, 26, 27]
```

توی این قسمت براساس الگویی که داریم بر روی text موردنظر حرکت می کنیم و جایگاه از text که از آن تا به اندازه سایز الگو یک substring با حداکثر تعداد number of mistake داشته باشیم را در لیست resualt نگه می‌داریم .

حاصل الگوریتم برروی فایل text برابر خواهد بود با یک لیست که آیتم ابتدایی آن برابر خواهد بود با :

```
Sample Output:
[5, 8, 11, 26, 31, 60, 76, 79, 82, 124, 144, 152, 167, 176, 179, 191, 193, 205, 219, 247, 250, 255, 258, 265, 272, 275, 296, 298, 299, 301,
304, 318, 327, 345, 354, 357, 380, 382, 383, 385, 392, 400, 417, 424, 454, 457, 484, 493, 496, 509, 516, 525, 552, 577, 592, 595, 607, 614,
625, 628, 643, 683, 694, 705, 735, 746, 752, 753, 772, 775, 809, 814, 836, 856, 858, 873, 881, 885, 894, 911, 922, 934, 941, 957, 966, 969,
983, 1004, 1026, 1033, 1063, 1066, 1070, 1077, 1087, 1095, 1116, 1119, 1139, 1140, 1153, 1189, 1194, 1207, 1224, 1230, 1250, 1252, 1282, 1293,
 1305, 1306, 1321, 1338, 1340, 1350, 1365, 1388, 1392, 1409, 1416, 1419, 1440, 1453, 1482, 1485, 1499, 1506, 1509, 1560, 1561, 1575, 1600,
1634, 1644, 1647, 1698, 1730, 1740, 1743, 1754, 1757, 1779, 1787, 1804, 1811, 1814, 1824, 1834, 1894, 1913, 1917, 1925, 1941, 1952, 1975,
1997, 2003, 2043, 2046, 2058, 2061, 2068, 2071, 2095, 2098, 2116, 2125, 2128, 2132, 2134, 2143, 2160, 2184, 2205, 2208, 2235, 2251, 2264,
2267, 2278, 2294, 2322, 2337, 2340, 2356, 2363, 2374, 2394, 2399, 2410, 2418, 2434, 2437, 2441, 2459, 2461, 2481, 2521, 2527, 2544, 2554,
2561, 2577, 2580, 2589, 2596, 2603, 2617, 2626, 2633, 2649, 2663, 2666, 2686, 2699, 2708, 2715, 2723, 2792, 2795, 2799, 2802, 2805, 2847,
2859, 2876, 2886, 2900, 2920, 2923, 2931, 2941, 2948, 2951, 2956, 2975, 3000, 3014, 3021, 3048, 3051, 3102, 3105, 3145, 3183, 3190, 3206,
3211, 3217, 3236, 3243, 3270, 3288, 3305, 3320, 3323, 3345, 3348, 3351, 3369, 3397, 3403, 3418, 3444, 3472, 3474, 3481, 3482, 3484, 3493,
```

3) In the final step, we gave you a DNA string, the length of the pattern, and the maximum number of mismatches (provided in the assignment file as ”prob1 c3.txt”). Your function should return a pattern with the most frequent occurrence in the DNA string. (13 points)

```
import itertools

# this function give all possible sub string we can have with special chars.
def make_all_possible_string_by(set_char,length):
    l = ''.join(set_char)
    yield from itertools.product(*([l] * length))

# test all sub string we can have and see how much can we find in the text with this mismatches and at the end return list of
# the biggest sub string frequency can occur in text .
def find_frequent_sub(text, length, mismatches=0):

    set_chr_in_text = sorted(set(text))
    dic_of_sub = {}
    for item in make_all_possible_string_by(set_chr_in_text,length):
        dic_of_sub[''.join(item)] = len(find_position_of_sub_string(''.join(item),text,mismatches))
    bigger_item_in_dic = [key for key, value in dic_of_sub.items() if value == max(dic_of_sub.values())]
    return bigger_item_in_dic
```

```
print(find_frequent_sub("ACGTTGCATGTCGCATGATGCATGAGAGCT",4,1))
```

```
['ATGC', 'ATGT', 'GATG']
```

در قسمت اول یک تابع نوشتم که یک ورودی از کاراکترهای مجاز میگیره و براساس طول رشته تمام رشته‌های ممکن با آن کارکترها را می‌سازد .

در قسمت یک متن و یک طول و تعداد اشتباه می گیریم و براساس آن کلماتی که بیشترین شکل تکرار توی متن را داشتم به عنوان خروجی بیرون می‌دهم برای پیدا کردن این کلمات هم به کمک تابع بالا اول تمام جایگشت‌ها را با کارکترهای موجود در متن براساس سایز موردنظر می‌سازم وسپس به کمک تابع سوال دو تعداد آیتم‌های درون لیست که نشان دهنده تعداد حالت های مختلفی هست که متواند الگو موردنظر را در رشته ما پیدا کرد در نظر میگیرم و آنانکه بیشترین تعداد آیتم را دارد را به عنوان خروجی برمی گردانم .

```
with open("prob1_c3.txt", encoding = 'utf-8') as f:
    data = f.read()

list_of_input = data.split('\n')
```

```
print(list_of_input)
```

```
['ATACGACGACTTTCTCGACGACGAATACTTTTTTCTTATACGACTCGACTTCTTTTTTTATACTTCTTCGACGATTCTTCGAATACGACGACGACGACGACTTTTCTTCTCGACTATACTTCTTATACTTCTTATACGAAT
AATAATAATAATAATACGACTTTTCTCTTCTTTTTTCTTCTTCTCTTTTCTTCGAATACTCTTTATACTTTTTTCTT', '5 3', '']
```

```
numbers = list_of_input[1].split()
print(find_frequent_sub(list_of_input[0],int(numbers[0]),int(numbers[1])))
```

```
['ATTTA', 'ATTTC']
```

در اینجا resualt مربوط به فایل و نگاه می کنید که کلمات ATTTC , ATTTA هر سه بیشترین الگو تکراری باحداکثر سه اشتباه بوده اند .

Generative and Discriminative models (20 points)

What is the difference between a generative and discriminative model? Explain with a statistical view.

https://stackoverflow.com/questions/879432/what-is-the-difference-between-a-generative-and-a-discriminative-algorithm

برگرفته از توضیح بالا:

فرض کنید داده های ورودی x دارید و می‌خواهید داده ها را در برحسب های y طبقه بندی کنید.

یک مدل generative احتمال مشترک $p(x,y)$ را می آموزد و یک مدل discriminative توزیع احتمال شرطی $p(y|x)$ را می آموزد - که ما آن را به عنوان "احتمال x به شرط y" می‌خوانیم.

در اینجا با یک مثال برحسب (x, y) توضیح می‌دهیم .

$(1 ,2)$,$(2,0)$,$(1,0)$,$(1,0)$

$p(x,y)$ is

	y=0	y=1
x=1	1/2	0
x=2	1/4	1/4

$p(y|x)$ is

	y=0	y=1
x=1	1	0
x=2	1/2	1/2

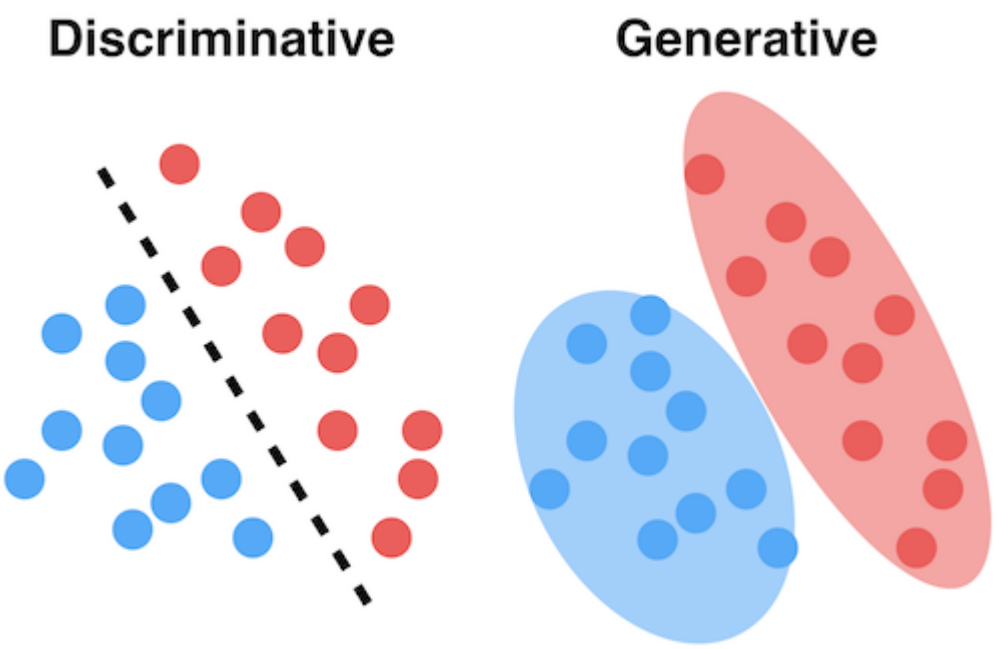
توزیع $p(y|x)$ توزیع طبیعی برای طبقه‌بندی مثال داده شده x به کلاس y است، به همین دلیل است که الگوریتم‌هایی که مستقیماً این را مدل می کنند، الگوریتم‌های discriminative نامیده می‌شوند.

الگوریتم‌های Generative به $p(x,y)$ را مدل می کنند که می‌توان آن را با اعمال قانون بیز به $p(y|x)$ تبدیل کرد و سپس برای طبقه‌بندی استفاده کرد. با این حال، توزیع $p(x,y)$ می تواند برای اهداف دیگر نیز استفاده شود. به عنوان مثال، می توانید از $p(x,y)$ برای ایجاد جفت های احتمالی (x,y) استفاده کنید.

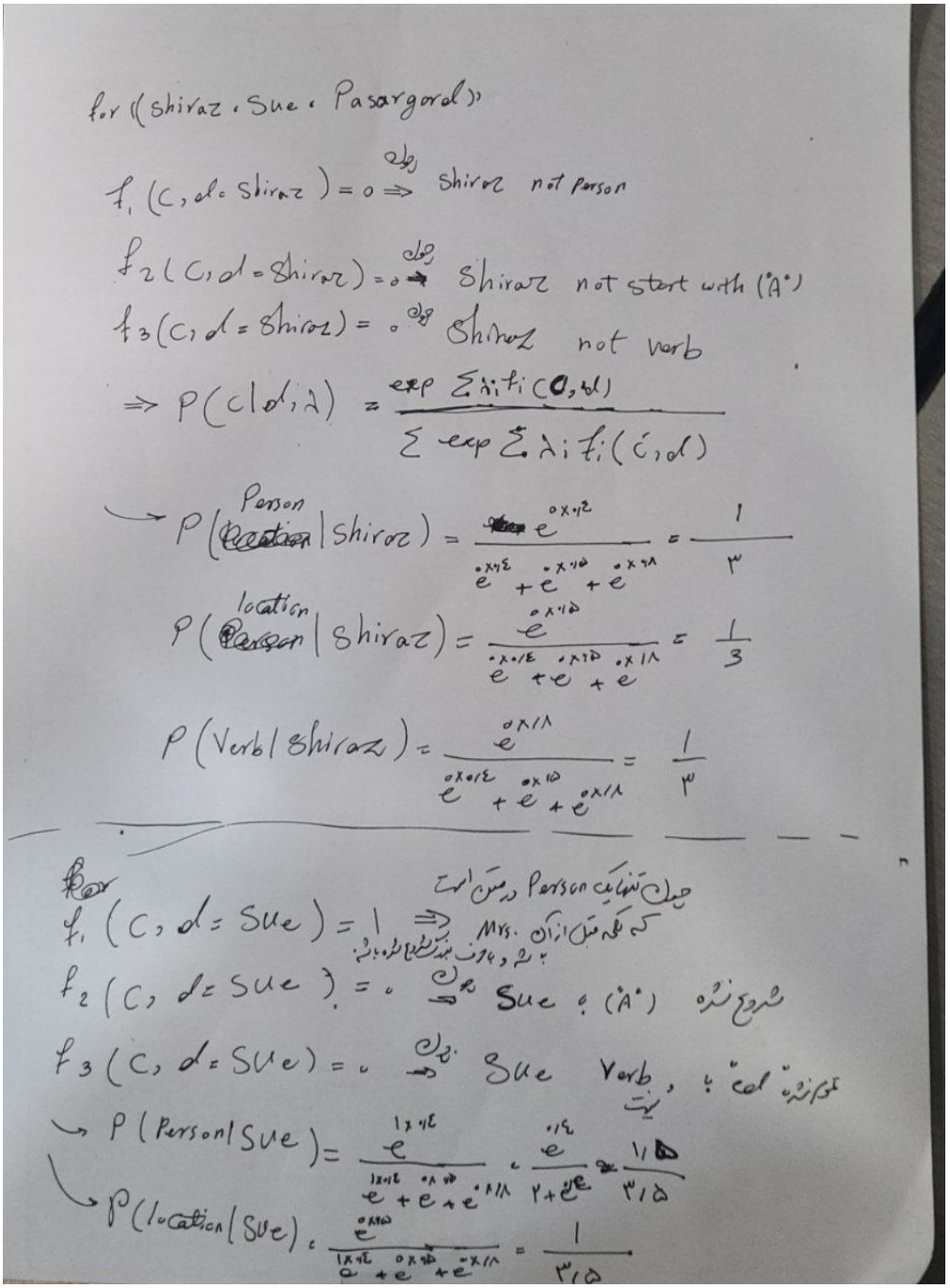
(b) (5 bonus points)

Compare generative and discriminative model performance based on the volume of a dataset and missing data.

discriminative مدل نسبت به missing data حساس‌تر هست اما با داشتن همان داده‌های مهم که در نتیجه‌میتوانند اثر زیادی داشته باشند به خوبی می‌تونیم خط تمایز بین داده‌ها را بکشیم و مدل را بدست آوریم پس به داده‌های زیادی الزاما نیاز نداریم ولی generative به صورت کلی نسبت به missing data زیاد حساس نیست ولی برای اینکه بتواند مدل با دقتی داشته باشیم باید تعداد داده‌های بیشتری داشته باشیم .



(c) (10 points)



پس sue یک شخص هست یا person .

$$P(\text{verb} \mid \text{SVe}) = \frac{e^{۰\times۰/۱۸}}{e^{۱\times۰/۴} + e^{۰\times۰/۱۵} + e^{۰\times۱۸}} = \frac{۱}{۳/۵}$$

$f_1(c, d = \text{Pasargard}) = 0 \Rightarrow$ نه اسم Person است
نه کلمه قبل از آن "Mrs." است.

$f_2(c, d = \text{Pasargad}) = 0$ location است ولی
با "A" شروع شده
و در ابتدای جمله هم نیست.

$f_3(c, d = \text{Pasergard}) = 0$ Verb نیست، و "کلمه" پایانی نیافته

$\hookrightarrow P(\text{Person} \mid \sout{\text{location}}\ \text{Pasargard}) = \frac{e^{۰\times۱۴}}{e^{۰\times۱۴} + e^{۰\times۰/۵} + e^{۰\times۱۸}} = \frac{1}{3}$

$\hookrightarrow P(\text{location} \mid \text{Passargard}) = \frac{e^{۰\times۰۱۵}}{e^{۰\times۱۴} + e^{۰\times۰/۵} + e^{۰\times۰/۱۸}} = \frac{1}{3}$

$\hookrightarrow P(\text{verb} \mid \text{Passargard}) = \frac{e^{۰\times۱۸}}{e^{۰\times۰/۴} + e^{۰\times۰/۵} + e^{۰\times۱۸}} = \frac{1}{3}$

Text Classification (60 points)

(a) (5 points)

Explain the reason for using logarithm in Naïve Bayes model calculation, and how it affects learning.

Naive Bayes calculations, like calculations for language modeling, are done in log space, to avoid underflow and increase speed.

طبق نوشته خود کتاب برای جلوگیری از underflow شدن چون در ضرب مقادیر بسیار کوچک احتمال آن هست که به صفر خیلی نزدیک شود و افزایش سرعت از logarithm استفاده می کنیم .

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} \log P(c) + \sum_{i \in positions} \log P(w_i|c)$$

(b) (5 points)

Consider the two following tables:

Compute micro and macro averaging precision, recall, and F1 score.

Class 1	Grand Truth		
		TRUE	FALSE
Predicted	TRUE	95	16
	FALSE	12	32

Class 2	Grand Truth		
		TRUE	FALSE
Predicted	TRUE	62	11
	FALSE	7	26

$$Micro - Precision = \frac{TruePositives1 + TruePositives2}{TruePositives1 + FalsePositives1 + TruePositives2 + FalsePositives2}$$

$$Micro - Recall = \frac{TruePositives1 + TruePositives2}{TruePositives1 + FalseNegatives1 + TruePositives2 + FalseNegatives2}$$

$$Micro - F - Score = 2.\frac{Micro - Precision\ .\ Micro - Recall}{Micro - Precision + Micro - Recall}$$

Micro precision = $(95 + 62)/\ ((95 + 16) + (\ 62 + 11))$ = 0.85326087

Micro_Recall = $(95 + 62)\ /((95+12)+(62+7))$ = 0.892045455

Micro_F_Score = $2*((0.85326087*0.892045455)/(\ 0.85326087 +$

$0.892045455))$=0.872222223

$$Macro - Precision = \frac{Pecision1 + Precision2}{2}$$

$$Macro - Recall = \frac{Recall1 + Recall2}{2}$$

$$Macro - F - Score = 2.\frac{Macro - Precision\ .\ Macro - Recall}{Macro - Precision + Macro - Recall}$$

persion1 = $95\ /(95+16)$ = 0.855855856

persion2 = $62/(62+11)$ = 0.849315068

Macro_Precision = $(0.855855856 + 0.849315068)/2$ = 0.852585462

Recall 1 = $95/(95+12)$ = 0.887850467

Recall 2 = $62/(62+7)$ = 0.898550725

Macro_Recall = $(0.887850467 + 0.898550725)/2$ = 0.893200596

Macro_F_Score = 2 * $(0.852585462 * 0.893200596)/(0.852585462 + 0.893200596)$

= 0.87242058

(c) (10 points)

Train a naïve Bayes Language model with add-1 smoothing on the following email title and corresponding

spam/non-spam as the label, then test the model on the test set. You should also add your own sample

in the empty rows (denoted by blue) of the train set.

	Email title	Label
Train	congrats you have achieved certificate	non-spam
	send us your google password	spam
	review your google password	non-spam
	send us your review	non-spam
	congrats you won lottery	spam
	review our website	spam
Test	review our changes send us your certificate	
	congrats your profile achieved our website	

جمله پیشنهادی:

send your certificate non-spam

درصد جمله های(-)non spam , smap(+) نسبت به کل جمله ها :

P(-) = 4/7 p(+) = 3/7

ما از smoothing در این شیوه استفاده می کنیم پس به اضافه یک در صورت و به اضافه تعداد کلمات vocabulary میکنیم.

vocabulary=

|{1 .Congrats, 2 .you, 3 .have, 4. achieved, 5 .certificate, 6. send, 7 .us, 8 .your, 9 .google, 10 .password,

11 .review, 12 .won, 13 .lottery, 14 .website, 15 .our}|= 15

P(“review”|-) = (1 + 1)/(16 + 15) = 2/31

P(“our”|-) = (0 + 1)/(16 + 15) = 1/31

P(“changes”|-) = در لغت نامه نیست پس در نظرش نمی گیریم

P(“send”|-) = (2+1)/(16 + 15)= 3/31

P(“us”|-) = (1 +1) / (16 + 15) = 2/31`

P(“your”|-) = (3+1)/(16 + 15) =4/31

P(“certificate”|-) = (2+1)/(16+15) = 3/31

P(“review”|+) = (1 + 1)/(12 + 15) = 2/27

P(“our”|+) = (1 + 1) / (12 + 15) = 2/27

P(“changes”|+) = در لغت نامه نیست پس در نظرش نمی گیریم

P(“send”|+) = (1+1) / (12 + 15) = 2/27

P(“us”|+) = (1+1) / (12 + 15) = 2/27

P(“your”|+) = (1 + 1) / (12 + 15) = 2/27

P(“certificate”|+) = (0+1) / (12 + 15) = 1/27

$P(-)P(S|-)$ = 4/7*((2*1*3*2*4*3)/(31**6)) = 0.000000093 بزرگتر از پایینی

$P(+)P(S|+)$ = 3/7*((2*2*2*2*2*1)/(27^6)) = 0.000000035

پس جمله اول تست اسپم نیست چون احتمال اسپم نبودنش بیشتر است .

P(“Congrats”|-) = (1+1)/(16+15) = 2/31

P(“your”|-) = (3+1)/(16 + 15) =4/31

P(“Profile”|-) = در لغت نامه نیست پس در نظرش نمی گیریم

P(“achieved”|-) = (1 +1) / (16 + 15) = 2/31`

P(“our”|-) = (0 + 1/(16 + 15) = 1/31

P("website"|-) = (0 + 1/(16 + 15) = 1/31

P("Congrats"|+) = (1+1)/(12+15) = 2/27

P("your"|+) = (1 + 1) / (12 + 15) = 2/27

P("Profile"|+) = در لغت نامه نیست پس در نظرش نمی گیریم

P("achieved"|+) = (0 +1) / (12 + 15) = 2/27

P("our"|+) = (1 + 1) / (12 + 15) = 2/27

P("website"|+) = (1 + 1/(12 + 15) = 2/27

$P(-)P(S|-)$ = (4/7)*((2*4*2*1*1)/(31^5)) = 0.000000319

$P(+)P(S|+)$ = (3/7)*((2*2*2*2*2)/(27^5)) = 0.000000956 بزرگتر است

پس جمله دوم تست اسپم تشخیص داده می‌شود .

(d) (35 points + 5 bonus)

In this part, we're going to train a Naïve Bayes Classifier for the task of sentiment analysis on the IMDB movie reviews dataset. Please complete the notebook provided in your assignment folder (35 points).

Criterion:

** You can't import any libraries in the notebook.

** You have to write comments in your code that makes it fully apprehensible.

** A code that meets the above criterion will result in a complete score for this section.

Bonus: Your model should have accuracy above 80 percent on the test set. (5 points)

برای این قسمت ابتدا به کمک pandas کلمات را در ساختار یک data fram قرار دادم .

```
######## Your Code Here ########
df =  pd.read_csv('IMDB-Dataset.csv')
df.head(10)
```

10 rows × 2 columns

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive
5	Probably my all-time favorite movie, a story o...	positive
6	I sure would like to see a resurrection of a u...	positive
7	This show was an amazing, fresh & innovative i...	negative
8	Encouraged by the positive comments about this...	negative
9	If you like original gut wrenching laughter yo...	positive

سپس برای انجام Preprocess براساس لایبرری‌های موجود یک کلاس ساختم که یک دیتافریم می‌گیرد و براساس نام ستونی که بهش داده شده توانایی این که فانکشن‌های موردنظر برای prepocess را روی آن انجام دهیم به ترتیب تبدیل کردن به کلمات uppercase به lower case پاک کردن tag ها، پاک کردن لینک و عدد و punctuation و space های اضافه و emoji و stopwords و در نهایت انجام عملیات lemmatization روی سطرهای آن ستون مشخص انجام می‌دهیم .

```python
# remove html tage
def remove_tags(string):
    result = re.sub('<.*?>','',string)
    return result


 # remove  links
def remove_links(string):
    result = re.sub('http[s]?:\/\/.*','',string)
    return result


class Preprocess:

    def __init__(self, data_frame,column_name):
        self.df = data_frame
        self.column_name = column_name

    def make_lower_case(self):
        self.df[self.column_name] = self.df[self.column_name].apply(lambda x: x.lower())

    def remove_tags(self):
        self.df[self.column_name] = self.df[self.column_name].apply(lambda cw : remove_tags(cw))

    def remove_link(self):
        self.df[self.column_name] = self.df[self.column_name].apply(lambda cw : remove_links(cw))
```

```python
def remove_link(self):
    self.df[self.column_name] = self.df[self.column_name].apply(lambda cw : remove_links(cw))

def remove_number(self):
    self.df[self.column_name] = self.df[self.column_name].apply(lambda x: re.sub(r'\d+', '', x))

def remove_punctuations(self):
    self.df[self.column_name] = self.df[self.column_name].apply(lambda x: x.translate(str.maketrans('', '', string.punctuation)))

def remove_double_spaces(self):
    self.df[self.column_name] = self.df[self.column_name].apply(lambda x: re.sub(' +', ' ', x))

def remove_emoji(self):
    self.df[self.column_name] = self.df[self.column_name].apply(lambda x: x.encode('ascii', 'ignore').decode('ascii'))
```

```
def remove_stopwords(self):

    stop_words = stopwords.words('english')
    # add new stopwords to stop words.
    new_stopwords = ['<*>']
    stop_words.extend(new_stopwords)
    # remove stopwords not .
    stop_words.remove('not')

    self.df[self.column_name] = self.df[self.column_name].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop_words)]))

def do_lemmatization(self):
    lemmatizer = WordNetLemmatizer()
    self.df[self.column_name] = self.df[self.column_name].apply(lambda x: lemmatizer.lemmatize(x))
```

```
model_pre = Preprocess(df,"review")
new_df = model_pre.do_all_preprocess()
new_df.to_csv("out.csv")
new_df.head(10)
```

10 rows | 10 rows × 2 columns

	review	sentiment
0	one reviewers mentioned watching oz episode youll hook…	positive
1	wonderful little production filming technique unassumi…	positive
2	thought wonderful way spend time hot summer weekend si…	positive
3	basically theres family little boy jake thinks theres …	negative
4	petter matteis love time money visually stunning film …	positive
5	probably alltime favorite movie story selflessness sac…	positive
6	sure would like see resurrection dated seahunt series …	positive
7	show amazing fresh innovative idea first aired first y…	negative
8	encouraged positive comments film looking forward watc…	negative
9	like original gut wrenching laughter like movie young …	positive

خوب سپس داده‌های train و test را به نسبت ۸۰ درصد train و ۲۰ درصد test تقسیم بندی کردم .
به کمک `from sklearn.model_selection import train_test_split as tts`

```
######## Your Code Here ########
X_train, X_test, y_train, y_test = tts(df['review'], df['sentiment'],random_state=12, train_size = .80)
```

سپس در انتها یک کلاس به اسم `MehNaiveBeyes` مبتنی بر الگوریتم Naive byes ساختم که در این الگوریتم یک فانکشن fit و predict داریم .

در فانکشن fit براساس داده‌های باینری train میزان که صورت یک بردار TF_IDF شده توسط کتاب خانه `TfidfVectorizer` ابتدا کلاس را به دو قسمت label مثبت و منفی تقسیم می‌کنیم .

و درصد داده‌های مثبت در train و test را به عنوان یک پارامتری از کلاس که باید بدست آوریم حساب می‌نمایم و سپس به ازای هر کلمه درون vocabulary یک احتمال برای حضور در جملات مثبت و منفی بدست می‌آوریم که این حاصل smooth شده هست .

```
######## Your Code Here ########
class MehNaiveBeyes:

    def __init__(self):
        self.p_pos = 0
        self.p_neg = 0
        self.list_of_p_pos = []
        self.list_of_p_neg = []

    def fit(self,vectors_feature,list_label,name_pos,name_neg):

        # At first, I find index of label pos and neg
        numbers_of_item_pos_label = [i for i in range(len(list_label)) if list_label[i] == name_pos]
        numbers_of_item_neg_label = [i for i in range(len(list_label)) if list_label[i] == name_neg]

        # Then I sum all vector of passive
        list_numbers_pos_item = np.zeros(vectors_feature.shape[1])
        for index in numbers_of_item_pos_label:
            item = vectors_feature[index].toarray()[0]

            list_numbers_pos_item = list_numbers_pos_item + item
```

```
    # And all vector of negative
    list_numbers_neg_item = np.zeros(vectors_feature.shape[1])
    for index in numbers_of_item_neg_label:
        item = vectors_feature[index].toarray()[0]
        list_numbers_neg_item = list_numbers_neg_item + item

    # Then Find Percentage of pas class and neg class
    self.p_pos = len(list_numbers_pos_item)/(len(list_numbers_pos_item)+len(list_numbers_neg_item))
    self.p_neg = len(list_numbers_neg_item)/(len(list_numbers_pos_item)+len(list_numbers_neg_item))


    # then for all element I find P(element | neg) P(element | pas)
    sum_list_numbers_pos_item = sum(list_numbers_pos_item)
    self.list_of_p_pos = np.array([(item+1)/(sum_list_numbers_pos_item+vectors_feature.shape[1]) for item in list_numbers_pos_item])

    sum_list_numbers_neg_item = sum(list_numbers_neg_item)
    self.list_of_p_neg = np.array([(item+1)/(sum_list_numbers_neg_item+vectors_feature.shape[1]) for item in list_numbers_neg_item])
```

بعد از بدست آوردن احتمالات حال به کمک این احتمالات می‌توانیم بردار TF IDF تست که براساس مدل fit شده train بدست آمده را به تابع predict بدهیم تا به کمک فرمول Naive Byes براساس جمله‌ای که به ما داده شده است یک احتمال برای مثبت بودن یا منفی بودن جمله بدست آوریم و درنهایت به کلاسی که بیشترین شباهت را دارد جمله مربوط را اطلاق دهیم .

برای بدست آوردن این بزرگی به دلیل اینکه درحالت معمول مقادیر بسیار کوچک می‌شدند از log استفاده کردم.

```
    def predict(self,vectors_feature,name_pos,name_neg):



        label_predict = []
        # We have percentage of pas and neg and P(element | neg) P(element | pa
        # I use np.log because my score became vary small if use original formu
        for item in vectors_feature:
            item_arr = item.toarray()[0]

            pos_res = np.log(self.p_pos)
            neg_res = np.log(self.p_neg)

            pos_res = pos_res + np.log(self.list_of_p_pos.dot(item_arr))
            neg_res = neg_res + np.log(self.list_of_p_neg.dot(item_arr))

            # Each part have more probability I label with it .
            if pos_res >= neg_res:
                label_predict.append(name_pos)
            else :
                label_predict.append(name_neg)

        return label_predict




tfidfvectorizer = TfidfVectorizer(analyzer='word' , stop_words='english',)
tfidfvectorizer.fit(X_train)
tfidf_train = tfidfvectorizer.transform(X_train)
print("n_samples: %d, n_features: %d" % tfidf_train.shape)
```

```
 n_samples: 40000, n_features: 185662
```

```
naive_beyes = MehNaiveBeyes()
naive_beyes.fit(tfidf_train, np.array(y_train),"positive","negative")
```

```
tfidf_test = tfidfvectorizer.transform(X_test)
y_pred = naive_beyes.predict(tfidf_test,"positive","negative")
```

در ادامه برای بدست آوردن دقت نیز یک کلاس به نام Ma calcu Accuracy ساختم که در این کلاس به کمک توابع دقت ، confusion matrix , f1 measure ، recall ، precision دقت هر یک را بدست می‌آوریم . تمامی تابع ها همان فرمولیشن‌های ریاضی را پیاده سازی می کنند .

برای دقت : تعداد جواب‌های درست را به کل داده‌ها تقسیم می کنیم.

برای percision : تعداد جواب‌های مثبتی که مثبت پیش بینی شده اند را به کل تعداد داده‌هایی که مثبت اصلی تقسیم می کنیم .

برای recall : تعداد جواب‌های مثبت که مثبت پیش بینی کرده ایم را به کل تعداد داده‌هایی که مثبت پیش بینی کرده ایم تقسیم می کنیم .برای confision ماتریس نیز یک ماتریس که در سمت چپ ترین ستون بالا FP و سمت راست ترین ستون بالا FP و پایین چپ FN و پایین راست TN هست قرار دارد را ساخته ایم .

برای بدست آوردن F1 هم از ضرب percison در recall تقسیم بر حاصل جمع این دوتا ضرب در ۲ استفاده کردم .

```python
class Me_Calcu_Accuracy:
    @staticmethod
    def accuracy_score(y_main, y_pred):
        y_main = list(y_main)
        y_pred = list(y_pred)
        # find all true Item model find
        upper = sum([1 for i in range(len(y_main)) if y_main[i]==y_pred[i]])
        # find accuracy
        return upper/len(y_main)
    @staticmethod
    def precision(y_main, y_pred,name):
        y_main = list(y_main)
        y_pred = list(y_pred)
        # find true possetive
        true_pos = sum([1 for i in range(len(y_main)) if y_main[i]==y_pred[i] and y_main[i] == name])
        # Find all item was pos in our prediction list
        down = sum([1 for i in range(len(y_main)) if  y_pred[i] == name])
        return true_pos/down

    @staticmethod
    def recall(y_main, y_pred,name):
        y_main = list(y_main)
        y_pred = list(y_pred)

        # find true possetive
        true_pos = sum([1 for i in range(len(y_main)) if y_main[i]==y_pred[i] and y_main[i] == name])
        # Find all item was pos in our main list
        pos =  sum([1 for i in range(len(y_main)) if  y_main[i] == name])

        return true_pos/pos
```

```python
@staticmethod
def f1_measure(y_main, y_pred,name):
    precision = Me_Calcu_Accuracy.precision(y_main, y_pred,name)
    recall = Me_Calcu_Accuracy.recall(y_main, y_pred,name)
    return 2 * (precision * recall)/(precision+recall)

@staticmethod
def confustion_matrix_2D(y_main, y_pred):
    y_main = list(y_main)
    y_pred = list(y_pred)
    set_item = set(y_main)
    list_set = list(set_item)
    set_item = (list_set[1],list_set[0])
    if len(set_item) != 2:
        raise "this item for binary"

    confusion_matrix = []
    k = 0
    for item in set_item :
        list_make = []
        if k == 0:
            list_make.append(sum([1 for i in range(len(y_main)) if y_main[i]==y_pred[i] and y_main[i] == item]))
            list_make.append(sum([1 for i in range(len(y_main)) if y_main[i]!=y_pred[i] and y_pred[i] == item]))
            k +=1
        else:
            list_make.append(sum([1 for i in range(len(y_main)) if y_main[i]!=y_pred[i] and y_pred[i] == item]))
            list_make.append(sum([1 for i in range(len(y_main)) if y_main[i]==y_pred[i] and y_main[i] == item]))


        confusion_matrix.append(list_make)

    print(set_item[0],set_item[1])
    for item in confusion_matrix:
        print(item)
```

دقت ما برابر است با :

```python
######## Your Code Here ########
y_pred = naive_beyes.predict(tfidf_test,"positive","negative")
score1 = Me_Calcu_Accuracy.accuracy_score(y_test,y_pred)
```

```python
score1
```

```
0.7778
```

Precision

```
######## Your Code Here ########
precision = Me_Calcu_Accuracy.precision(y_test,y_pred,"positive")
print("precision : ",precision)
```

```
precision :  0.8925193465176269
```

Recall

```
######## Your Code Here ########
recall = Me_Calcu_Accuracy.recall(y_test,y_pred,"positive")
print("recall : ",recall)
```

```
recall :  0.6249247441300422
```

F-measure

```
######## Your Code Here ########
f_measure = Me_Calcu_Accuracy.f1_measure(y_test,y_pred,"positive")
print("f1_measure L ",f_measure)
```

```
f1_measure L  0.735127478753541
```

Confustion matrix

```
######## Your Code Here ########
Me_Calcu_Accuracy.confustion_matrix_2D(y_test,y_pred)
```

```
 positive negative
[3114, 375]
[1869, 4642]
```