

گزارش کار مراحل انجام پروژه

بخش صفرم : شروع کار

مرحله اول :

در این مرحله شرایط لازم برای نوشتن پروژه را مهیا کردم، مثلاً **extension** های لازم را در **vscode** ریخته و در نصب هفتم روی ویندوز به مشکل فوردم که با اپدیت کردن **pip** حل شد.

به فایل آموزش پایتون رجوع کردم و **numpy** و **pandas** و **matplotlib** را با **pip** ریختم و برای شروع در با **pandas** فایل **train** را باز کردم و چک کردم که درست کار بکند. (البته بعداً متوجه شدم با **jupyter** به مراتب راحتتر میتوان چرول های **pandas** را مشاهده کرد)

```
import pandas as pd
data= pd.read_csv("books_test.csv")

for i in data.index:
    print(f" its title {data['title'][i]}")
    print(f" its describe {data['description'][i]}")
    print(f" its cat {data['categories'][i]}")
```

مرحله دوم:

بخش غیرامتیازی ویدیوی مربوط به پروژه را دیدم و پی دی اف پروژه را با دقت خواندم و اندکی روی روابط ریاضیاتی آن محاسبات انجام دادم که کلیات مسئله دستم بیاید

Handwritten mathematical derivations for the Naive Bayes classifier:

$$P(c, m) = P(m|c) \cdot P(c)$$

$$P(m) = \sum_c P(m|c) \cdot P(c)$$

$$P(c|m) = \frac{P(m|c) \cdot P(c)}{\sum_c P(m|c) \cdot P(c)}$$

The handwritten notes include Persian text explaining the steps and the meaning of the variables.

جواب های تقریبی ای برای پرسش های مطرح شده در فایل دارم که در ادامه پاسخ نهایی را خواهم داد.

بخش اول : پیش پردازش

مرحله سوم:

برای پیش پردازش ابتدا با هفتم، تمام رشته ها را به کلمه تبدیل کردم و سپس با **pandas** آنها را به یک **dandas dataframe** تبدیل کردم سپس یک لیست از علائم نگارشی ساختم تا بوسیله ی آنها علائم نگارشی اضافه را فیلتر کنم

```
punctuationMarks = [':', ',', '?', '!', '.', ';',  
                    '(', ')', '!', '+', '-', '*', '/', '=', '\\\\', '»', '«',  
                    '"', '_', '...']
```

در برقی موارد هفتم درست عملیات پردازش را انجام نمیداد و مثلا اگر عبارت (زکریای رازی (1970-2021) را با یکی دو تا فاصله ی اضافی بهش میدادیم، عبارت 2021-1970 را یک کلمه در نظر میگرفت که فب نه عدد بود و نه علامت نگارشی، پس نمیتوانستم این موارد را فیلتر کنم. پس از **regex** استفاده کردم و تلاش کردم کلا هر کلمه ای که هر حرفی بهز حروف فارسی دارد را حذف کنم.

این برای حروف انگلیسی بود ولی برای فارسی دردرس بیشتری داشت و باید با یونیکد حروف فارسی **regex** را مینوشتیم:

```
result = re.sub(r'^[A-Za-z]', '', s)
```

این برای فارسی نتیجه شد:

```
res = "".join(re.findall(r"[\u0600-\u06FF]+", s))
```

سپس درس زدم برقی کلمات انگلیسی هم معم اند پس این را نوشتم

```
res = "".join(re.findall(r"[\u0600-\u06FF]|[A-Z]|[a-z]+", s))
```

اما نهایتا زمانی که صرف میشد برای این تحلیل دقیق، بسیار زیاد بود پس فطای زیادتر را بیفیل شده و با فرض اینکه تمامی متن های توضیحات و عنوان کتاب ها کاملا از لحاظ ادبی اصلاح شده و درست اند و دارای فاصله های بیجا نیستند، به همان حذف علائم نگارشی و اعداد اکتفا کردم:

```
punctuationMarks = [':', ',', '?', '!', '.', ';',  
                    '(', ')', '!', '+', '-', '*', '/', '=', '\\\\',  
                    '»', '«', '"', '_', '...']
```

...

```
lambda w: w not in punctuationMarks and w.isdigit() == False
```

همچنین یکسری کلمات هستند که در بسیاری از متون وجود دارند و آمدن آنها در یک متن اطلاعات خاصی در باب موضوع آن متن به ما نمیدهد مانند برقی افعال پرکاربرد (است، بود، رفت) و برقی حرف اضافه ها (به، و، اما)، اینها را **StopWords** نامیده و مجموعه ای از آنها هم داخل خود کتابخانه ی هفتم موجود بود و هم در یک فایل برایمان در تلگرام قرار داده شد که من از فایل تلگرامی استفاده کردم و تمام **StopWord** ها را از بین کلماتم حذف کردم. (در ادامه هر تملیلی که میکنم، یکبار بدون حذف **stop words** و یکبار با حذف آن، نتیجه را نوشته ام)

نهایتاً به یک تابع رسیدیم که عملیات پیش پردازش را روی یک **cell** انجام میدهد و کافیسست آنرا روی **dataframe** اصلی **apply** کنیم (البته یک تابع خوب دیگر برای اینکار تابع **isalpha** بود که کد آنرا هم در فایل کد کامنت کردیم)

```
punctuationMarks = [':', '!', '?', '!', '.', '!', ';',
                    '(', ')', '!', '+', '-', '*', '/', '=', '\\', '»',
                    '«', '"', '_', '...']
stopWords = pd.read_csv("sw.csv")

def preprocess_a_cell(txt):
    wordsDf = pd.DataFrame(word_tokenize(txt), columns=["word"])
    wordsDf["word"] = wordsDf["word"].apply(
        lambda w: w not in punctuationMarks and w not in
stopWords["word"].values and w.isdigit() == False )
    wordsDf = wordsDf.dropna()

    return wordsDf
```

روند **apply** کردن بوسیله ی پانداس هم اینگونه نوشتیم:

```
df = pd.read_csv("bokbok.csv")
df["description"] = df["description"].apply(preprocess_a_cell)
df["title"] = df["title"].apply(preprocess_a_cell)
```

نتیجه گیری : در کل در این بخش تلاشیم بر این بود که چیزهایی که بیش از حد عام هستند (مانند **stopWords** و علائم نگارشی) و یا چیزهایی که بیش از حد اختصاصی هستند (مانند کلمات خاص انگلیسی یا اعداد) را از بین کلمات حذف کنیم تا احتمال هایی که در نهایت مناسبه می شوند، دقیق تر باشند و احتمال خطا کمتر شود. (در ادامه نیز برای دقیقتر شدن از ریشه یابی نیز استفاده خواهیم کرد چون کلماتی که از یک ریشه هستند، امروندان در جمله اطلاعات یکسانی از موضوع آن جمله به ما میدهد)

بخش دوم : حل مسئله

تا اینجای کار ما یک دیتافریم پانداس داریم که 3 ستون دارد و در ستون **title** و **description** در هر کدام از سطر ها یک دیتافریم دیگر موجود است (البته **pandas** برای دیتافریم هایی که یک ستون دارند پیازی بنام **series** هم دارد که چون استفاده از آن تاثیر فاصی در ذخیره ی زمان نداشت، همان دیتافریم را ترجیح دادیم) که یک ستون دارد و شامل کلمات بررد بفعوری است که نگه داشته ایم و در ستون **categories** هم یک رشته (**string**) بعنوان موضوع کتاب آمده است.

مرحله چهارم:

برای شروع سافت **BOW** باید تعداد سطرهای آنرا مناسبه کنیم، پس آمدم و تعداد کتاب های موجود در هر موضوع را شمردم تا احتمال وجود هر کتاب در هر موضوع هم درست بنیاید چون بعدا در محاسبات به آن نیاز داریم:

```
countOfCategories = {}
for i in df.index:
    if (df['categories'][i] in countOfCategories):
        countOfCategories[df['categories'][i]] += 1
    else:
        countOfCategories[df['categories'][i]] = 1
print(countOfCategories)
out:
{'جامعه شناسی': 425, 'کلیات اسلام': 425, 'داستان کودک و نوجوانان': 425, 'داستان کوتاه': 425, 'مدیریت و کسب و کار': 425, 'رمان': 425}
```

در نتیجه جمعا 6 موضوع داریم که یعنی **BOW** دارای 6 سطر است و همچنین احتمال اینکه یک کتاب رندوم دارای موضوع **X** باشد، دقیقا یک ششم است.

مرحله پنجم:

یک **BOW** فالی دارای 6 ستون میسازیم با نام های موضوعاتمان و سپس به وسیله ی کد زیر پیمایش کرده و پس از حدود 9 دقیقه و 30 ثانیه بدول **BOW** تکمیل میشود(و بعد **BOW.copy** این است که اگر نباشد در طول کد در اصل به تعداد بالایی ما ستون اضافه میکنیم به دیتافریم اصلی و این اضافه کردن ها **performance** را بشدت کاهش میدهند و هم زمان اجرای برنامه زیار میشود و هم به **performanceWarning** میخوریم. پس برای رفع این مشکلات یک کپی سافته و تغییرات را رویش اعمال می کنیم.

```
categories = ['کلیات اسلام', 'جامعه شناسی', 'رمان', 'مدیریت و کسب و کار', 'داستان کوتاه', 'داستان کودک و نوجوانان']
BOW = pd.DataFrame(index=categories)
for i in df.index:
    for word in df['description'][i]['word']:
        if word not in BOW.columns:
            BOW = BOW.copy()
            BOW[word] = 0
        BOW[word][df['categories'][i]] += 1
BOW
```

Out :

بیاور	عاد	خبرخواهی	ستمگر	واهمه ای	...	جامعه نظریه	دربدارنده	رمضانی	ابوالفضل	نوشته	ایران	جامعه شناسی	نظریه های	ساختار
0	0	0	0	0	...	206	52	9	1	3	132	182	26	32
0	0	0	0	0	...	61	13	6	0	2	97	34	1	2
1	1	1	1	1	...	4	0	5	6	2	122	21	0	0
0	0	0	0	0	...	24	1	22	0	1	156	46	0	3
0	0	0	0	0	...	26	8	5	0	0	159	23	6	7
0	0	0	0	0	...	39	1	1	0	0	147	39	0	5

6 rows × 41110 columns

مرحله ششم:

زمانی که برای ساختن BOW مصرف میشد بیش از حد انتظار بود در نتیجه تصمیم گرفتیم ساختار را به دو دیکشنری تو در تو تغییر دهیم تا با dynamic شدن تعداد اعضای هر سطر، از بهبود آمدن cell های اضافه با مقدار 0 جلوگیری کرده و در زمان صرفه جویی کنیم.

در نهایت یک دیکشنری تو در تو به شکل زیر داریم :

```
BOW = {'داستان کودک و نوجوانان': {}, 'کلیات اسلام': {}, 'جامعه‌شناسی': {},  
       'رمان': {}, 'مدیریت و کسب و کار': {}, 'داستان کوتاه': {}}
```

و کد نهایی اینگونه شد:

```
for bookIndex in df.index:  
    curCategory = df['categories'][bookIndex]  
    for word in df['description'][bookIndex]['word']:  
        if word not in BOW[curCategory].keys():  
            BOW[curCategory][word] = 0  
            BOW[curCategory][word] += 1  
    for word in df['title'][bookIndex]['word']:  
        if word not in BOW[curCategory].keys():  
            BOW[curCategory][word] = 0  
            BOW[curCategory][word] += 1
```

همچنین از لحاظ زمانی بفاصله کم کردن تعداد ستون ها و تغییر فرمت از pandas DF به python dict بسیار صرفه جویی کردیم و زمان اجرای کد به کمتر از 30 ثانیه کاهش یافت:

```
> for i in BOW.keys():  
    print(f"{i} : \n {BOW[i]} \n تعداد کلمات : {len(BOW[i])}")  
[15] ✓ 25.1s Python  
...  
جامعه‌شناسی :  
{  
  'احمد': 8,  
  'اشرف': 4,  
  'پرویز': 2,  
  'پیران': 2,  
  'حسین': 8,  
  'بشپریه': 1,  
  'های': 28,  
  'جامعه‌شناسی': 35,  
  'نظریه': 12611  
}  
تعداد کلمات : 12611  
کلیات اسلام :  
{  
  'های': 6,  
  'مختلفی': 6,  
  'زندگی': 151,  
  'شخصیت': 21,  
  'امام': 345,  
  'مهدی': 56,  
  'عج': 56,  
  'اذعان': 6,  
  'پرسش': 10251  
}  
تعداد کلمات : 10251  
داستان کودک و نوجوانان :  
{  
  'بودکان': 234,  
  'مجموعه': 4,  
  'دریا': 11,  
  'قلم': 63,  
  'مهری': 3,  
  'ماهوتی': 3,  
  'تصویرگری': 104,  
  'عاطفه': 7,  
  'ملکی': 6950  
}  
تعداد کلمات : 6950  
داستان کوتاه :  
{  
  'داستان': 337,  
  'کوتاهی': 69,  
  'نوشته': 156,  
  'محمود': 17,  
  'راجی': 1,  
  'نویسنده': 317,  
  'معاصر': 86,  
  'ایرانی': 77,  
  'تعداد کلمات': 15125  
}  
مدیریت و کسب و کار :  
{  
  'فر': 8,  
  'زندگی': 228,  
  'می': 70,  
  'دنیا': 68,  
  'دوره': 10,  
  'خارق': 20,  
  'زیادی': 63,  
  'سرمایه': 9424  
}  
تعداد کلمات : 9424  
رمان :  
{  
  'بیتزر': 11,  
  'پن': 3,  
  'فاکتر': 4,  
  'اثر': 219,  
  'تحت': 7,  
  'سال': 373,  
  'منتشر': 57,  
  'برنده': 458,  
  'ساعت': 17019  
}  
تعداد کلمات : 17019
```

بخش سوم: تست کردن الگوریتم

مرحله هفتم:

حالا که مدل ما آماده شده است باید آنرا تست کنیم و بکمک فایل **books_test** در صد درستی آنرا چک کنیم

اگر فرض کنیم که :

اگر کتاب جدیدی باشد که کلمات $\{w_1, w_2, \dots, w_n\}$ داخل آن است، به سببی حسابی این احتمال و محاسبی آنها می توان موضوع آنرا حساب زد

$C = \text{بیشتر متعلق کتاب به موضوع } C$
 $w_i = \text{بیشتر متعلق یک کلمه ی خاص}$

$$P(w_1, w_2, \dots, w_n | C) \propto P(w_1 | C) \times P(w_2 | C) \times \dots \times P(w_n | C)$$

$$P(w_1 | C) \propto P(w_1 | \text{کلمات اسلام})$$

$$P(w_2 | C) \propto P(w_2 | \text{کلمات اسلام})$$

$$\vdots$$

$$P(w_n | C) \propto \dots$$

تا اینجا ما در مرحله ی چهارم احتمال **prior** یا همان p_C را که مساوی یک ششم بود یافتیم و بوسیله ی **BOW**، $p_{x_i|C}$ ها را داریم.

پس در این مرحله ابتدا **BOW** را به **PROBS BOW** تبدیل میکنیم یعنی همون دیکشنری قبل را داریم اما بعنوان مقدار جلوی هر کلمه، $p_{x_i|C}$ آنرا نوشته ایم.

```
def convert_BOW_to_probs_BOW(BOW):
    for category in BOW.keys():
        countOfAllWords = 0

        for word in BOW[category].keys():
            countOfAllWords += BOW[category][word]

        for word in BOW[category].keys():
            # change value of words to conditional prob of them
            BOW[category][word] = BOW[category][word]/countOfAllWords

    return BOW
```

نتیجه:

```
جامعه‌شناسی :
ای: '0.0005287508261731659', 'جامعه‌u200cساختر': '0.0006609385327164573', 'نظریه': '0.0006609385327164573'
تعداد کلمات: 12611
کلیات اسلام :
عج: '0.0014469162597214686', 'اذهان': '0.001550267421130145', 'پرسش‌u200cپرسش': '0.001550267421130145'
تعداد کلمات: 10251
داستان کودک و نوجوانان :
تصویرگری: '0.004340929960764671', 'عاطفه': '0.00029217797812839134', 'ملکی': '0.00029217797812839134'
تعداد کلمات: 6950
داستان کوتاه :
محمود: '0.0031097998564707', 'راجی': '1.9934614464556255'
تعداد کلمات: 15125
مدیریت و کسب و کار :
های: '0.00044880281848170005', 'زیادی': '0.0014137288782173551', 'سرمایه‌u200cراه': '0.0014137288782173551'
تعداد کلمات: 9424
رمان :
منتشر: '0.0008735766065380312', 'برنده‌u200cرمان': '0.0070192646630599705', 'ساعت': '0.0070192646630599705'
تعداد کلمات: 17019
```

مرحله ی هشتم:

به سراغ سافتن تابع اصلی تست میرویم

```
def test(BOW,testFileAddress):
    testDf = pd.read_csv(testFileAddress)
    testDf["description"] =
testDf["description"].apply(preprocess_a_test_cell)
    testDf["title"] = testDf["title"].apply(preprocess_a_test_cell)
    trueTests = 0
    for bookIndex in testDf.index:
        categoryProbs = {'جامعه‌شناسی': 0, 'کلیات اسلام': 0, 'داستان کودک و نوجوانان': 0,
        'رمان': 0, 'مدیریت و کسب و کار': 0, 'داستان کوتاه': 0}
        bookTitle = testDf['title'][bookIndex]['word']
        bookDescription = testDf['description'][bookIndex]['word']
        for category in categoryProbs.keys():
            categoryProbs[category] =
calculateCategorylikelihood(bookTitle,bookDescription,BOW[category])
            if returnModelGuess(categoryProbs) ==
testDf['categories'][bookIndex] :
                trueTests += 1
    return trueTests/len(testDf)
```

اگر در توضیحات موجود درباره ی یک کتاب، با کلمه ای مواجه شویم که در BoW وجود نداشته باشد چه باید کرد؟ احتمال صفر باید

در نظر گرفت یا باید آن کلمه را در نظر نگرفت؟ برای تابع calculateCategorylikelihood این حالات امتحان کرده

و نتیجه را گزارش و تفسیر می کنیم:

حالت اول :

احتمال 1 میگیریم برای کلمه ای که در BOW نیست: (یعنی آن کلمه را در نظر نمیگیریم)

```
def calculateCategorylikelihood (bookTitle,bookDescription,categoryDict):  
    likelihood = 1/6  
    # ایگنور کردن کلماتی که نیستن و دادن مقدار 1 بجاشون  
    for word in bookDescription :  
        if word in categoryDict.keys():  
            likelihood *= categoryDict[word]  
        else:  
            likelihood *= 1  
    for word in bookTitle :  
        if word in categoryDict.keys():  
            likelihood *= categoryDict[word]  
        else:  
            likelihood *= 1  
    return likelihood
```

و در نهایت 6 عدد بعنوان احتمال مضمون در هر موضوع در دیکشنری categoryProbs خواهیم داشت اما نکته اینجاست که چون برای کلماتی که در BOW نیستند مقدار 1 را ذکر کرده ایم که بیشترین احتمال ممکن است، باید آن موضوعی را بعنوان نتیجه ی تحلیل ذکر کنیم که احتمالش از همه کمتر شده است (یعنی تعداد 1 های کمتری در هم ضرب شده اند و تعداد اعداد بین 0 و 1 زیاد بوده پس عدد نهایی یک عدد کوچک است):

```
def returnModelGuess(categoryProbs):  
    probs = list(categoryProbs.values())  
    categories = list(categoryProbs.keys())  
    return categories[probs.index(min(probs))]
```

نتیجه ی حالت اول با حذف stop words :

تعداد کل تست ها : 450
تعداد تست های درست پاسخ داده شده : 158
درصد بازدهی این مدل : 35.11111111111111
نتیجه ی حالت اول بدون حذف stop words :

تعداد کل تست ها : 450
تعداد تست های درست پاسخ داده شده : 132
درصد بازدهی این مدل : 29.333333333333332

حالت دوم:

احتمال 0 میگیریم برای کلمه ای که در BOW نیست:

```
def calculateCategoryLikelihood(bookTitle, bookDescription, categoryDict):
    likelihood = 1/6
    # ایگنور کردن کلماتی که نیستن و دادن مقدار 1 بجاشون
    for word in bookDescription :
        if word in categoryDict.keys():
            likelihood *= categoryDict[word]
        else:
            likelihood *= 0
    for word in bookTitle :
        if word in categoryDict.keys():
            likelihood *= categoryDict[word]
        else:
            likelihood *= 0
    return likelihood
```

در این حالت بر خلاف حالت 1، برای کلمه ای که در BOW نبوده کمترین احتمال ممکن (0) را در نظر گرفته ایم پس نهایتاً باید بین احتمال های مقبور در موضوعات مختلف، MAX را مناسبه کنیم و بعنوان نتیجه ذکر کنیم:

```
def returnModelGuess(categoryProbs):
    probs = list(categoryProbs.values())
    categories = list(categoryProbs.keys())
    return categories[probs.index(max(probs))]
```

نتیجه ی حالت دوم بدون حذف stop words:

تعداد کل تست ها : 450

تعداد تست های درست پاسخ داده شده : 75

درصد بازدهی این مدل : 16.666666666666664

نتیجه ی حالت دوم با حذف stop words:

تعداد کل تست ها : 450

تعداد تست های درست پاسخ داده شده : 75

درصد بازدهی این مدل : 16.666666666666664

تفسیر: بردیوتا بالافره در توضیحات و نام هر کتاب یک کلمه یافت میشود که در **BOW** نبوده باشد، در نتیجه همان یرونه **0** باعث میشود $p_{x|c}$ برای تمامی کتاب ها **0** شده و پایتون بطور رندوم یک کتاب را انتخاب کند که در نهایت هم به احتمال یک ششم یا همان **16%** انتخاب های درست را انجام داده ایم.

حالت سوم:

استفاده از لوگاریتم و در نظر نگرفتن کلماتی که در **BOW** نیستند

```
def calculateCategorylikelihood(bookTitle,bookDescription,categoryDict):
    likelihood = 0
    # ایگنور کردن کلماتی که نیستن و دادن مقدار 0 بجاشون
    for word in bookDescription :
        if word in categoryDict.keys():
            likelihood+= np.log(categoryDict[word])
        else:
            likelihood += 0
    for word in bookTitle :
        if word in categoryDict.keys():
            likelihood += np.log(categoryDict[word])
        else:
            likelihood += 0
    return likelihood
```

نتیجه ی حالت سوم بدون حذف stop words:

تعداد کل تست ها : 450
تعداد تست های درست پاسخ داده شده : 190
درصد بازدهی این مدل : 42.22222222222222
نتیجه ی حالت سوم با حذف stop words:

تعداد کل تست ها : 450
تعداد تست های درست پاسخ داده شده : 194
درصد بازدهی این مدل : 43.111111111111114

پاسخ پرسش 1: اگر کلمه ای باشد که در سطر مد نظر ما در **BOW** نباشد، کار بهتر این است که آنرا در نظر نگیریم، چون در این صورت اولاً که به احتمال بسیار زیاد در هر داده ی **TEST** بالافره کلمه ای است که **train** نشده باشد، پس تقریباً در تمام تست ها به حالتی می رسیم که احتمال تعلق داشتن به هر موضوع همان یک ششم است که در **prior** بود، و یا مثلاً

یک کلمه در هیچ موضوعی **train** نشده ولی بطور شانس در یک موضوع بی ربط مثلاً یک بار آمده است، و همین ضرب کردن باعث میشود که مدل ما فکر کند به احتمال **100** درصد آن کتاب مال آن موضوع است که کاملاً هم بی ربط بوده است با آن.

پاسخ پرسشی 2: در کل میدانیم که $p_{x_i|c}$ ها اعداد کوچکی هستند (همانطور که در عکس **PROBS BOW** دیدیم از اردر **0.0001** هستند) و در روش قبلی برای مناسبه، تعداد زیادی از این اعداد کوچک در هم ضرب میشدند که باعث میشد در برخی مواقع از میزان دقت پایتون فراتر رفته و مقدار آنها را تقریباً **0** محسوب کند که باعث فضا در محاسبات میشد، اما با لوگاریتم گرفتن، در اصل بجای ضرب کردن اعداد بسیار نزدیک به صفر، لوگاریتم آنها را که یکسری اعداد منفی هستند را با هم جمع میکنیم که فب عدد حاصل یک عدد منفی بزرگ ممکن است بشود اما انقدر بزرگ نمیشود که پایتون نتواند آنرا حساب کند پس ازین لحاظ همانند روش قبلی به مشکل نفواهیم خورد.

حالت چهارم:

استفاده از لوگاریتم و **0** گذاشتن احتمال کلماتی که در **BOW** نیستند:

نتیجه ی حالت چهارم بدون حذف stop words:

تعداد کل تست ها : 450
تعداد تست های درست پاسخ داده شده : 75
درصد بازدهی این مدل : 16.666666666666664
نتیجه ی حالت چهارم با حذف stop words:

تعداد کل تست ها : 450
تعداد تست های درست پاسخ داده شده : 75
درصد بازدهی این مدل : 16.666666666666664

اینکار بدیهتاً بدلیل استفاده از $\log(0)$ در برخی مواقع، به **warning** هایی میفورد اما نهایتاً نتیجه دقیقاً مثل حالت **2** میشود چون بالاخره در توضیحات و نام هر کتاب یک کلمه یافت میشود که در **BOW** نبوده باشد، در نتیجه همان یرونه $\log(0)$ باعث میشود $p_{x|c}$ برای تمامی کتاب ها منفی بینهایت شده و پایتون بطور رندوم یک کتاب را انتخاب کند که در نهایت هم به احتمال یک ششم یا همان **16%** انتخاب های درست را انجام داده ایم

در باب additive smooth:

در روش **2** و **4** دیدیم که اگر حتی یکی از $p_{x_i|c}$ ها مساوی صفر شود، باعث میشود که کل $p_{x|c}$ هم صفر شود، پس در این روش می آیم و بجای صفر، یک مقدار کوچک بعنوان $p_{x_i|c}$ برای کلماتی که در **BOW** نبودند میگذاریم که به آن مشکل نفوریم.

حالا برای یافتن مقدار کوچک من ازین فرمول استفاده کردیم:

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

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 $\mathbf{x} = [x_1, \dots, x_d]$.

برای ثابت دلفواه الفا هم با اندکی تحلیل و همپنین آزمون مقدار مختلف، فهمیدیم عددی از اردر **1/100000** بهترین بازدهی را میدهد چون در کل از اردر **100000** کلمه داریم در **train** و اگر بطور کلی به احتمالات بنگریم برای کلمه ای که در **BOW** نیست هم بالاخره **1/100000** احتمال حضور وجود دارد. و با بررسی مقدار مختلف این اردر دیدیم **1/500000** بهترین نتیجه را میدهد.

حالت پنجم:

استفاده از **additive smooth** (بدون گرفتن لوگاریتم)

در این حالت باید تغییراتی در مرحله ای که **BOW** را اپریت میکردیم بوجود بیآوریم و احتمال هاییکه مینویسیم را با فرمول جدید بنویسیم:

```
def convert_BOW_to_additive_smooth_probs_BOW(BOW):
    for category in BOW.keys():
        countOfDifferentWords = len(BOW[category])
        countOfAllWords = countAllWords(BOW[category])
        for word in BOW[category].keys():
            # change value of words to conditional prob of them
            BOW[category][word] = (BOW[category][word] + ALPHA) /
            (countOfAllWords + (ALPHA * countOfDifferentWords))

    return BOW
```

و در مناسبه ی بخش **likelihood** نیز به گونه ای دیگر برای کلماتی که در **BOW** نیستند عمل میکنیم:

```
def calculateCategorylikelihood(bookTitle, bookDescription, categoryDict):
    likelihood = 1/6
    wordsCount = countAllWords(categoryDict)
    for word in bookDescription:
        if word in categoryDict.keys():
            likelihood *= categoryDict[word]
        else:
            likelihood *= ALPHA/(wordsCount +
                                   (ALPHA * len(categoryDict)))
    for word in bookTitle:
        if word in categoryDict.keys():
            likelihood *= categoryDict[word]
        else:
            likelihood *= ALPHA/(wordsCount +
                                   (ALPHA * len(categoryDict)))
    return likelihood
```

نتیجه ی حالت پنجم بدون حذف stop words:

تعداد کل تست ها : 450
تعداد تست های درست پاسخ داده شده : 140
درصد بازدهی این مدل : 31.11111111111111
نتیجه ی حالت پنجم با حذف stop words:

تعداد کل تست ها : 450
تعداد تست های درست پاسخ داده شده : 160
درصد بازدهی این مدل : 35.55555555555556

حالت ششم :

استفاده از **additive smooth** (با گرفتن لوگاریتم)

نهایتا این حالت از همه مدل ها بهتر عمل خواهد کرد زیرا لوگاریتم گرفتن فویش مشکلات مربوط به نزدیک صفر شدن احتمال ها را حل میکند و **smooth additive** هم مشکلات مربوط به اعدادی که در سطر مرد بررسی در **BOW** نیستند را حل می کند و بین روش های موجود، بهترین روش همین است. (که حالا با حذف **stop words** و یا ریشه یابی تا حدی هم میتوان آنرا بیشتر بهبود داد)

```
def calculateCategorylikelihood(bookTitle, bookDescription, categoryDict):
```

```

likelyhood = np.log(1)
wordsCount = countAllWords(categoryDict)
for word in bookDescription:
    if word in categoryDict.keys():
        likelyhood += np.log(categoryDict[word])
    else:
        likelyhood += np.log(ALPHA/(wordsCount +
                                   (ALPHA * len(categoryDict))))
for word in bookTitle:
    if word in categoryDict.keys():
        likelyhood += np.log(categoryDict[word])
    else:
        likelyhood += np.log(ALPHA/(wordsCount +
                                   (ALPHA * len(categoryDict))))
return likelyhood

```

نتیجه ی حالت ششم بدون حذف stop words:

```

[58] ✓ 21.1s
...
450 : تعداد کل تست ها
362 : تعداد تست های درست پاسخ داده شده
80.44444444444444 : درصد بازدهی این مدل

```

نتیجه ی حالت ششم با حذف stop words:

```

[57] ✓ 39.7s
...
450 : تعداد کل تست ها
367 : تعداد تست های درست پاسخ داده شده
81.55555555555556 : درصد بازدهی این مدل

```

با اینکه حالت ششم بدون حذف **stop words** از لحاظ تناسب کارایی و زمان اجرا در کل بهترین حالت است، اما با انجام ریشه یابی، از لحاظ صرفاً بازدهی، به بهترین حالت ممکن میرسیم

در باب lemmatization و stemming

برای ریشه‌یابی کلمات معمولاً از دو روش **Stemming** و **Lemmatization** استفاده می‌شود که هر دو روش قادر هستند ریشه‌ی یک کلمه را به دست بیاورند. برای مثال کلمه‌ای مانند “رفتن” را تصور کنید که ممکن است در جمله‌های مختلف به شکل‌های گوناگون ظاهر شود. برای مثال: رفتم / برویم / می‌رفتم / بروند. تمامی فعل‌های “رفتن” در این جملات را می‌توان به کلمه‌ی “رفت” نگاشت کرد. با این کار می‌توان ویژگی (رفتم، برویم، می‌رفتم، بروند) را به یک ویژگی (رفت) نگاشت کرد و با این عمل کاهش ویژگی (کاهش ابعاد) نیز انجام می‌شود. عمل **Lemmatization** نیز می‌تواند توسط روش‌هایی انجام شود. در این عمل نیاز است که از یک فرهنگ لغت یا پیزی شبیه به آن برای به دست آوردن ریشه‌ی لغات استفاده شود، چون عموماً روش‌های **Lemmatization** به صورت نا قاعده نیستند.

حالت هفتم :

با **stemming**: در **stemming** ما عمل ریشه یابی را انجام می‌دهیم و در هضم هم این عمل فیلوی دقیق انجام نمی‌شود چون در این عمل فقط چند کاراکتر اثر کلمه را حذف می‌کند به خیال اینکه به ریشه برسد که اغلب منجر به املا و معنای نادرست می‌شود و حتی در افعالی از یک نوع هم این عمل به فوئی انجام نمی‌شود:

```
from hazm import *
print(Stemmer().stem("رفتم"),Stemmer().stem("رفتی"),Stemmer().stem("رفت"),
      Stemmer().stem("رفتیم"),Stemmer().stem("رفتید"),Stemmer().stem("رفتند"))
```

✓ 0.0s

رف رفت رفت رفتند

یا حتی کلماتی که ظاهری شبیه فعل دارند را خراب میکند:

```
from hazm import *
print(Stemmer().stem("تابستان"), Stemmer().stem("زمستان"), Stemmer().stem("نسیم"), )
✓ 0.0s
```

تابس زمس نس

به هر حال اگر بخواهیم از **stemming** استفاده کنیم بر اِعتی با **apply** اینکار انجام میشود و توابع پیش پردازشمان اینگونه می شود

```
def preprocess_a_cell(txt):
    punctuationMarks = [':', '!', '?', '.', ';',
                        '(', ')', '!', '+', '-', '*', '/',
                        '=', '\\', '»', «', '"', '_', '-', '...']
    stopWords = pd.read_csv("sw.csv")
```

```

wordsDf = pd.DataFrame(word_tokenize(txt), columns=["word"])
wordsDf["word"] = wordsDf["word"].apply(
    lambda w: w not in punctuationMarks and
    w not in stopWords["word"].values and
    w.isdigit() == False)
wordsDf = wordsDf.dropna()
wordsDf["word"] = wordsDf["word"].apply(Stemmer().stem)
return wordsDf
def preprocess_a_test_cell(txt):
wordsDf = pd.DataFrame(word_tokenize(txt), columns=["word"])
wordsDf["word"] = wordsDf["word"].apply(Stemmer().stem)
return wordsDf

```

نتیجه ی حالت هفتم بدون additive smooth و با حذف stop words :

تعداد کل تست ها : 450
تعداد تست های درست پاسخ داده شده : 139
درصد بازدهی این مدل : 30.88888888888889

نتیجه ی حالت هفتم با additive smooth و با حذف stop words :

تعداد کل تست ها : 450
تعداد تست های درست پاسخ داده شده : 313
درصد بازدهی این مدل : 69.55555555555556

نتیجه ی حالت هفتم بدون additive smooth و با stop words :

تعداد کل تست ها : 450
تعداد تست های درست پاسخ داده شده : 133
درصد بازدهی این مدل : 29.55555555555557

نتیجه ی حالت هفتم با additive smooth و با stop words :

تعداد کل تست ها : 450
تعداد تست های درست پاسخ داده شده : 361
درصد بازدهی این مدل : 80.22222222222222

تفسیر: حالت بدون additive smooth را اگر stemming هم نکنیم، همان حالت سوم میشود و حالت با additive smooth هم همان حالت ششم میشود که همانطور که گفتیم بدلیل ضعیف بودن عملیات stemming در ریشه یابی، نه تنها دقتمان زیاده نکرده بلکه مقداری کمتر هم شده است. و در بین این حالات بهترین کار این بود که stop words را حذف نکرده و stemming کنیم که احتمالا بفاطر این است که اگر هم stop words و هم stemming کنیم، تعداد زیادی از کلمات حذف شده و یا نصفه میشوند که در نهایت چون تعداد داده های train هم خیلی نیستند، باعث نتایج اشتباه میشوند و استفاده از stemming خالی بهترین کار است

حالت هشتم :

با **lemmatizing** در **lemmatizing** عمل بن واژه یابی را انجام می‌دهیم یعنی اینکار هر فرقه ای تر و پیچیده تر از **stemming** است و نیاز به پردازش‌هایی مثل **pos tagging** دارد و بر اساس نوع کلمه و جایی که بکار رفته است به یافتن ریشه می‌پردازیم.

این عملیات در هضم هم بقبوی انجام میشود اما بدلیل پیچیده بودن، سرعت اجرا را بسیار کند می‌کند و البته که این روش هم مشکلاتی دارد و مثلا در هضم کلماتی که واقعا شبیه به هم اند و از یک خانواده هستند را هم از هم جدا می‌گیرد و تعداد بسیاری از کلمات را تحلیل نمی‌کند

```
from hazm import *
print(Lemmatizer().lemmatize("رفتم"), Lemmatizer().lemmatize("رفتن"),
      Lemmatizer().lemmatize("روان"), Lemmatizer().lemmatize("رونده"),)
✓ 1.9s
رفت#زو رفتن روان رونده
```

در کل برای **lemmatizing** با هضم تابع‌های پیش پردازش را به این شکل تغییر می‌دهیم:

```
def preprocess_a_cell(txt):
    punctuationMarks = [':', ',', '?', '!', '.', ';',
                        '(', ')', '+', '-', '*', '/',
                        '=', '\\', '»', '«', '"', "'", '_', '-', '...']
    stopWords = pd.read_csv("sw.csv")

    wordsDf = pd.DataFrame(word_tokenize(txt), columns=["word"])
    wordsDf["word"] = wordsDf["word"].apply(
        lambda w: w not in punctuationMarks and
        w.isdigit() == False)
    wordsDf = wordsDf.dropna()
    wordsDf["word"] = wordsDf["word"].apply(Lemmatizer().lemmatize)
    return wordsDf

def preprocess_a_test_cell(txt):
    wordsDf = pd.DataFrame(word_tokenize(txt), columns=["word"])
    wordsDf["word"] = wordsDf["word"].apply(Lemmatizer().lemmatize)
    return wordsDf
```

نتیجه ی حالت هشتم بدون additive smooth و بدون حذف stop words :

تعداد کل تست ها : 450

تعداد تست های درست پاسخ داده شده : 130

درصد بازدهی این مدل : 28.888888888888

نتیجه ی حالت هشتم با additive smooth و با حذف stop words :

تعداد کل تست ها : 450

تعداد تست های درست پاسخ داده شده : 368

درصد بازدهی این مدل : 81.777777777777

تفسیر: نهایتا درست است lemmatizing زمان ران شرن را بالا میبرد اما دقت را مقدار اندکی بیشتر میکند در حالت دارای additive smooth اما اگر additive smooth نباشد انقدر فضا زیاد است و مقدار train کم است که نتایج دارای منطق خاصی نیستند و استفاده از ریشه یابی هم کمک زیادی به ما بران نکرده و حتی بازدهی را کمتر میکند.

نتیجه گیری

در کل از لحاظ مجموعه ی بازدهی و زمان اجرا، حالت ششم از همه بهتر بود که جهت تحلیل دقیقتر، احتمال هر کدام از دسته بندی ها را حساب کردم و در فایل txt ای بصورت زیر گذاشتم

{خاسی: 4523, 862429371373- کلمات ایلام: 4525, 129978640578- 'داسان کوک و بوجوالی': 4554, 810796792539- 'داسان کوکاه': 4474, 150749069651- 'صیرت و کسب و کار': 4624, 055298921272- 'ران': 4366, 936593367656- 'جاسی: 4569, 87197840576- کلمات ایلام: 4601, 692090325234- 'داسان کوک و بوجوالی': 4605, 837091954772- 'داسان کوکاه': 4368, 364340233703- 'صیرت و کسب و کار': 4585, 542452805024- 'ران': 4433, 455915926718- 'جاسی: 1883, 6707811373615- 'داسان کوک و بوجوالی': 2041, 0712738656491- 'داسان کوکاه': 2033, 3019809524472- 'صیرت و کسب و کار': 2043, 5820458570122- 'ران': 2029, 7404421110075- 'جاسی: 4014, 1744446737618- 'داسان کوک و بوجوالی': 3951, 9696202541554- 'داسان کوکاه': 3931, 838838412442- 'صیرت و کسب و کار': 3940, 098837758456- 'ران': 3866, 389375441175- 'جاسی: 1213, 8091801113792- کلمات ایلام: 1259, 74409228095489- 'صیرت و کسب و کار': 1259, 74409228095489- 'ران': 1227, 9443539456684- 'جاسی: 3196, 4470961546385- کلمات ایلام: 3239, 9597934077724- 'داسان کوک و بوجوالی': 3534, 069485598679- 'داسان کوکاه': 3483, 60033272217- 'صیرت و کسب و کار': 3280, 69184732267- 'ران': 3456, 0416984133417- 'جاسی: 537, 7443349649199- کلمات ایلام: 509, 027784505001- 'داسان کوک و بوجوالی': 546, 628021945427- 'داسان کوکاه': 557, 2985528262149- 'صیرت و کسب و کار': 552, 2588975627949- 'ران': 555, 40674698696- 'جاسی: 4567, 30419666683- کلمات ایلام: 4639, 8827427548- 'داسان کوک و بوجوالی': 559751773291- 'داسان کوکاه': 4412, 076408180367- 'صیرت و کسب و کار': 4661, 125167445557- 'ران': 4426, 6218045521386- 'جاسی: 1670, 8317381408801- کلمات ایلام: 1734, 8370282525152- 'داسان کوک و بوجوالی': 1823, 9828911743614- 'داسان کوکاه': 1760, 1522230042904- 'صیرت و کسب و کار': 1681, 6808009243316- 'ران': 1747, 337044327076- 'جاسی: 854, 5143773784499- کلمات ایلام: 782, 9678547239145- 'داسان کوک و بوجوالی': 859, 6385952866657- 'داسان کوکاه': 863, 4538805657011- 'صیرت و کسب و کار': 867, 6533304229972- 'ران': 867, 6533304229972- 'جاسی: 287, 6831656896181- 'داسان کوک و بوجوالی': 304, 8321854485479- 'داسان کوکاه': 293, 1785153939169- 'صیرت و کسب و کار': 260, 0918647641666- 'ران': 2929, 278755252325- 'جاسی: 2641, 0243574272945- کلمات ایلام: 2789, 4865346225247- 'داسان کوک و بوجوالی': 2804, 0060089138054- 'داسان کوکاه': 2812, 087509353918- 'صیرت و کسب و کار': 2579, 300998946452- 'ران': 2286, 454265593414- 'جاسی: 2268, 832118878522- کلمات ایلام: 2276, 25111075663- 'داسان کوک و بوجوالی': 2237, 702950558967- 'داسان کوکاه': 2241, 126423625345- 'صیرت و کسب و کار': 2318, 010622965623- 'ران': 4020, 0648297788766- 'داسان کوک و بوجوالی': 4099, 334400817092- 'داسان کوکاه': 4025, 3288312119603- 'صیرت و کسب و کار': 4100, 064170059944- 'ران': 3945, 0232431346253- 'جاسی: 4099, 573595940963- کلمات ایلام: 4166, 131019305714- 'داسان کوک و بوجوالی': 3966, 3356075805245- 'داسان کوکاه': 3953, 0246584141983- 'صیرت و کسب و کار': 4110, 576501321622- 'ران': 4949, 558517206897- 'جاسی: 358, 80439070999137- کلمات ایلام: 359, 5138569121089- 'داسان کوک و بوجوالی': 321, 5202604302365- 'داسان کوکاه': 334, 06068446338617- 'صیرت و کسب و کار': 355, 18914941332673- 'ران': 325, 06572819455397- 'جاسی: 3160, 895911947156- کلمات ایلام: 3265, 790462814258- 'داسان کوک و بوجوالی': 3328, 49602449976- 'داسان کوکاه': 3364, 064126953755- 'صیرت و کسب و کار': 3080, 783526828613- 'ران': 3310, 224448053307- 'جاسی: 2226, 6862876726373- کلمات ایلام: 2293, 1732985556546- 'داسان کوک و بوجوالی': 2282, 169308863283- 'داسان کوکاه': 2264, 192898415276- 'صیرت و کسب و کار': 2122, 535126636689- 'ران': 3676, 108746048155- 'جاسی: 3606, 682867706135- کلمات ایلام: 3690, 5885445002406- 'داسان کوک و بوجوالی': 3753, 0685778678203- 'داسان کوکاه': 3710, 4282995860695- 'صیرت و کسب و کار': 3436, 9225564723324- 'ران': 3676, 108746048155- 'جاسی: 3857, 556771280093- کلمات ایلام: 3788, 6722379557073- 'داسان کوک و بوجوالی': 3737, 9999707234633- 'داسان کوکاه': 3878, 0761864435303- 'صیرت و کسب و کار': 3878, 0761864435303- 'ران': 3594, 9527106087025- 'جاسی: 3433, 821284518933- 'داسان کوک و بوجوالی': 3472, 170126147027- 'داسان کوکاه': 3472, 170126147027- 'صیرت و کسب و کار': 3250, 0606624463326- 'ران': 3463, 248660164033- 'جاسی: 3158, 4121153023766- کلمات ایلام: 3211, 2231120831125- 'داسان کوک و بوجوالی': 3462, 5177591543916- 'داسان کوکاه': 3372, 9744835103525- 'صیرت و کسب و کار': 3324, 8418632233393- 'ران': 3363, 4998859895745- 'جاسی: 1467, 7033699596015- کلمات ایلام: 1477, 3653086373768- 'داسان کوک و بوجوالی': 1346, 3854394200482- 'داسان کوکاه': 1445, 6745289295102- 'صیرت و کسب و کار': 1485, 7654938520377- 'ران': 1450, 146701245005- 'جاسی: 3746, 260804281964- 'داسان کوک و بوجوالی': 3571, 73890980162803- 'داسان کوکاه': 3516, 4922067619280- 'صیرت و کسب و کار': 3776, 9356999449374- 'ران': 503, 838658653045- 'جاسی: 2363, 1590580881647- 'داسان کوک و بوجوالی': 2329, 521586446466- 'داسان کوکاه': 2308, 584214197258- 'ران': 2275, 2362448207646- 'جاسی: 561, 1018105374742- کلمات ایلام: 557, 1369248485682- 'داسان کوک و بوجوالی': 549, 6449161684938- 'داسان کوکاه': 541, 5525025548832- 'صیرت و کسب و کار': 588, 9560996694149- 'ران': 556, 8852043554451- 'جاسی: 4096, 237869422224- کلمات ایلام: 4125, 638939957744- 'داسان کوک و بوجوالی': 3969, 3439754339106- 'داسان کوکاه': 3895, 6773298804485- 'صیرت و کسب و کار': 4113, 332715735014- 'ران': 3948, 5309171510003- 'جاسی: 1814, 967379931064- 'داسان کوک و بوجوالی': 1830, 8109062066555- 'داسان کوکاه': 1827, 9196301296727- 'صیرت و کسب و کار': 1719, 5109276185638- 'ران': 1888, 3201820051652- 'جاسی: 4103, 406727007288- کلمات ایلام: 4119, 8239342099505- 'داسان کوک و بوجوالی': 4098, 437451307715- 'داسان کوکاه': 3959, 321489683419- 'صیرت و کسب و کار': 4112, 8235082315205- 'ران': 3984, 4446945212844- 'جاسی: 3764, 580304668578- کلمات ایلام: 3439, 5352672595072- 'داسان کوک و بوجوالی': 3770, 1728628394603- 'داسان کوکاه': 3745, 4473632039953- 'صیرت و کسب و کار': 3846, 29389218982- 'ران': 3809, 3271152727834- 'جاسی: 1237, 0447689690292- کلمات ایلام: 1237, 0447689690292- 'داسان کوک و بوجوالی': 1167, 0689581283933- 'داسان کوکاه': 1235, 4097849708331- 'صیرت و کسب و کار': 1235, 4097849708331- 'ران': 1162, 0804462440453- 'جاسی: 3640, 221265089533- کلمات ایلام: 3773, 7379182318637- 'داسان کوک و بوجوالی': 3773, 7379182318637- 'داسان کوکاه': 3594, 3959224524504- 'صیرت و کسب و کار': 3792, 1152966177738- 'ران': 3895, 830847019904- 'جاسی: 1351, 2641736357834- کلمات ایلام: 1202, 5029402229036- 'داسان کوک و بوجوالی': 1430, 6418856166658- 'داسان کوکاه': 1451, 1304075314572- 'صیرت و کسب و کار': 1450, 9563533544344- 'ران': 1459, 097488950556- 'جاسی: 476, 2289700581729- کلمات ایلام: 469, 4121492649853- 'داسان کوک و بوجوالی': 550, 9881410346055- 'داسان کوکاه': 543, 8351757713406- 'صیرت و کسب و کار': 484, 56220838294325- 'ران': 527, 3431757754332- 'جاسی: 9432, 943240672849- کلمات ایلام: 2753, 553827005959- 'داسان کوک و بوجوالی': 2064, 7707001474573- 'داسان کوکاه': 2668, 2131713835847- 'صیرت و کسب و کار': 2713, 226457125373- 'ران': 2668, 5005759275114- 'جاسی: 2776, 2335683277865- کلمات ایلام: 2986, 4544164351732- 'داسان کوک و بوجوالی': 3004, 1414721418823- 'داسان کوکاه': 3030, 303584488223- 'صیرت و کسب و کار': 2919, 66924668066- 'ران': 2995, 0698221267467- 'جاسی: 781, 2787887157954- کلمات ایلام: 804, 0612326022566- 'داسان کوک و بوجوالی': 850, 4875244686593- 'داسان کوکاه': 835, 9219414669298- 'صیرت و کسب و کار': 766, 9680221526203- 'ران': 834, 1291448986431- 'جاسی: 3287, 294499991821- کلمات ایلام: 3287, 294499991821- 'داسان کوک و بوجوالی': 3289, 925682734356- 'داسان کوکاه': 3372, 4724623264063- 'صیرت و کسب و کار': 3395, 473327792517- 'ران': 3181, 090518123712- 'جاسی: 1580, 9631740956163- کلمات ایلام: 1578, 888688217136- 'داسان کوک و بوجوالی': 1578, 888688217136- 'داسان کوکاه': 1585, 2253261632075- 'صیرت و کسب و کار': 1612, 8630454170038- 'ران': 1583, 7266068992321- 'جاسی: