گزارش تمرین تشخیص ایمیل اسپم

نیکتا گوهری صدر 9513433

در ابتدا کتابخانه های مورد نظر ایمپورت می شوند

سپس فایل ها خوانده میشوند که در اینجا به جای کلمه ham از okay استفاده شده است.

سپس داده ها پیش پردازش می‌شوند که این پیش پردازش شامل جداسازی کلمات یا tokenize، حذف stop word ها یا ایست واژه ها، ریشه یابی کلمات یا stemming و همچنین چک کردن اینکه بعد از ریشه یابی طول کلمه کمتر از ۲ نباشد و حذف کاراکترهای غیر فارسی مانند حروف انگلیسی، علامت گذاری ها یاpunctuation ، اعداد و غیره میباشد.

به دلیل این که داده ها جداگانه خوانده شده‌اند پیش پردازش داده های اسپم و غیر اسپم جداگانه صورت می‌گیرد

همچنین پیش پردازش از دو لیست استفاده می کند که لیست اول کلمات مناسب را در خود ذخیره می کند و سپس بعد از ریشه یابی و تمامی مراحل گفته شده، کلمات دوباره جوین شده و به صورت جمله در لیست نهایی ذخیر میشود.

این پیش پردازش هم روی داده های train و هم test انجام میشود که نمونه ی نتیجه ی آن در عکس زیر مشخص است.

عکس اول قبل از پیش پردازش است:

	0
0	با سلام و احترام!!!بدینوسیله پیرو نامه شماره ت...
1	«لا یحب الله جهربالسوء من القول الا من ظلم»!!!...
2	سلام. در مورد این که آموزش و پرورش مدرک آزفا...
3	سلام چه خوبه که شما احساس مسئولیت دارین و جوا...
4	سلام من مترجمی زبان انگلیسی خوندم البته نه سا...

عکس دوم بعد از پیش پردازش است:

	0
0	سلام احترام بدینوسیله پیرو نامه شماره تاریخ اه...
1	یحب الله جهربالسوء القول الا ظلم جناب آقا خرم ...
2	سلام آموزش پرورش مدرک آرفا پرسنل رسمی کنه نوی ...
3	سلام خوبه احساس مسئولیت جواب دین خوام زبانشناس...
4	سلام مترجمی زبان انگلیسی خوندم سال دلایل دست ت...

پس از پیش پردازش معیار خی دو را پیاده سازی می‌کنیم که برای این کار باید داده های اسپم و okرا ابتدا به هم بچسبانیم و لیست لیبل را هم درست کنیم.

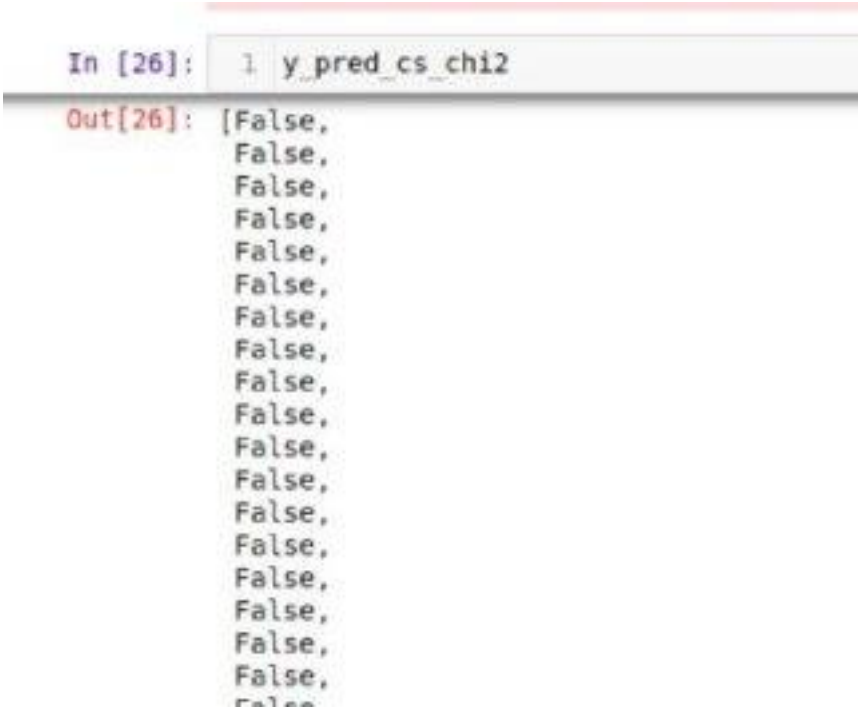
سپس با استفاده از تابع countvectorizer داده‌ها را وکتورایز می‌کنیم و بهترین ویژگی ها را انتخاب میکنیم که برابر است با k تا ویژگی ای که خی دو بیشترین امتیاز را میدهد.
که در این جا k=500 قرار داده شده است.
که نتیجه ی آن را در عکس زیر مشاهده میکنید.

```
In [15]: train_vector.shape
Out[15]: (500, 13416)

In [16]: type(train_vector)
Out[16]: numpy.ndarray

In [17]: test_vector.shape
Out[17]: (400, 13416)

In [18]: type(test_vector)
Out[18]: numpy.ndarray

In [19]: train_vector_chi2.shape
Out[19]: (500, 500)

In [20]: test_vector_chi2.shape
Out[20]: (400, 500)
```

برای استفاده از الگوریتم knn نیاز به یک معیار شباهت داریم که ابتدا معیار cosine similarityو سپس knn را پیاده سازی می کنیم.

نتیجه ی این الگوریتم بدون معیار خی دو در عکس زیر مشخص است.

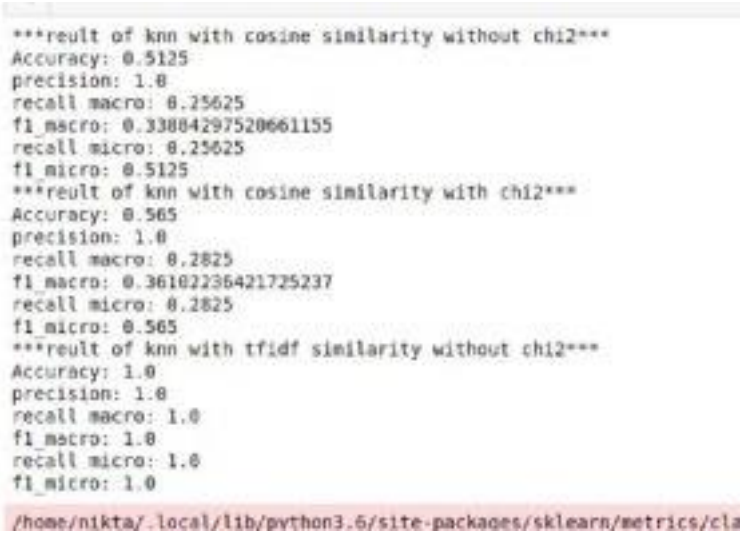
حال میخوایم همین الگوریتم را با معیار خی دو بررسی کنیم که نتیجه در عکس زیر معلوم است.



در مرحله ی بعد از معیار tfidf برای knn بدون خی دو استفاده میکنیم که اجرای این الگوریتم بسیار زمان بر است اما در نهایت خروجی داده شده و هم در عکس زیر هم در کد مشخص است.

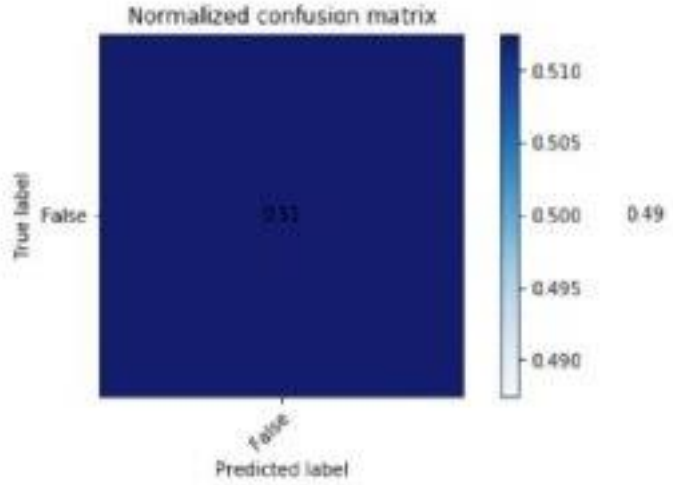
همچنین برای اجرای این الگوریتم به جای داده های به صورت جمله از داده های tokenize شده استفاده شده است.

در این مرحله میخواهیم با استفاده از معیار های مختلف مدل هایمان را ارزیابی کنیم.

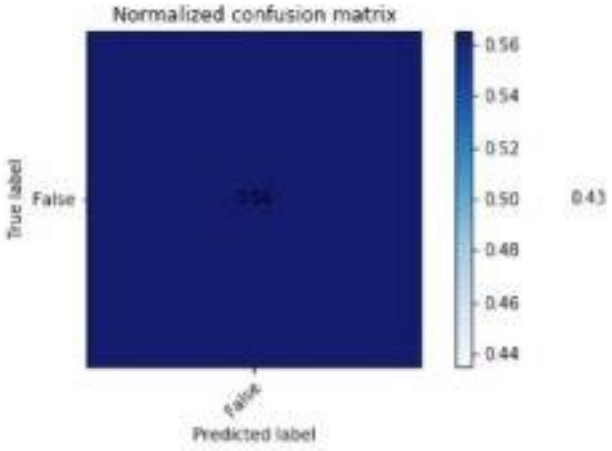


همانطور که در عکس مشخص است knn با tfidf بهترین دقت را داشته است و بعد از آن knnبا cosine similarity و با feature selection خی دو نتیجه ی بهتری تا بدون خی دو داشته است.

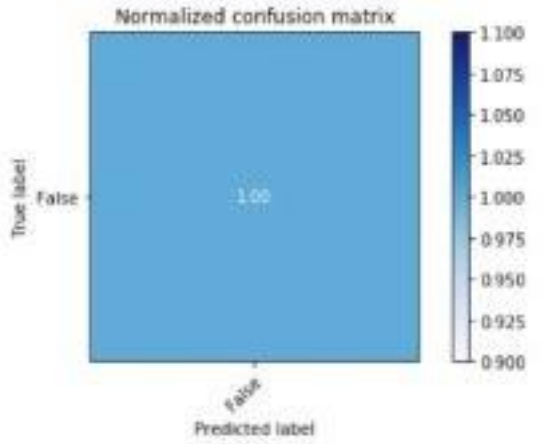
Confusion matrixمدل های نیز در عکس های زیر مشخص است



nan nan



nan nan



در مرحله ی بعد الگوریتم نایو بیز جند جمله ای پیاده سازی شده است که این بخش امتیازی بوده.

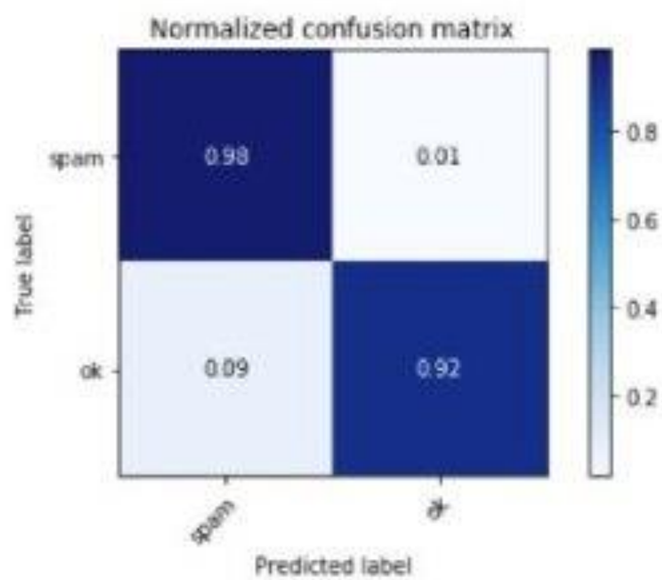
عکس اول الگوریتم نایو بیز بدون معیار خی دو است.

```
In [58]: 1 Y_pred_nb
Out[58]: ['ok',
          'ok',
          'ok',
          'ok',
          'ok',
          'ok',
          'ok',
          'ok',
          'ok',
          'ok',
          'ok',
          'ok',
          'ok',
          'ok',
          'ok',
          'ok',
          'ok',
          'ok',
          'ok',
          'ok',
```

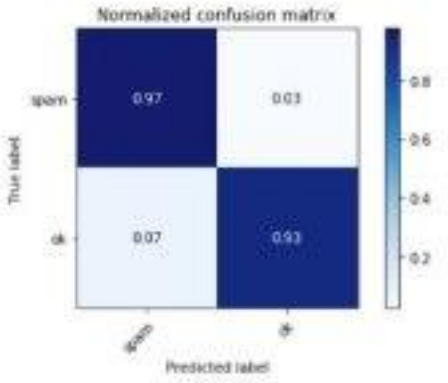
عکس دوم الگوریتم نایو بیزبا معیار خی دو است.

```
In [59]:  1 Y_pred_nb_chi2

Out[59]: ['ok',
          'ok',
          'ok',
          'ok',
          'ok',
          'ok',
          'ok',
          'ok',
          'ok',
          'ok',
          'ok',
          'ok',
          'ok',
          'ok',
          'ok',
          'ok',
          'ok',
          'ok',
          'ok',
```

معیار های ارزیابی و confusion matrix این دو مدل نیز در عکس زیر آمده است.



```
Out[63]: <matplotlib.axes._subplots.AxesSubplot at 0x7f97b5ca5e48>
```



همانطور که میبینیم به طور کلی الگوریتم نایو بیز از knn هم بهتر عمل میکند هم سریع تر. اما مقایسه ی نایو بیز با خی دو و بدون خی دو دشوار است زیرا در برخی معیار ها با خی دو بهتر است و در برخی بدون خی دو.