گزارش کار تمرین دوم یادگیری ماشین

بیتا آذری جو ۸۳۰۵۹۸۰۰۷

بخش اول :

۱,۱ در این قسمت عملیات tokenization برحسب جداسازی فاصله صورت گرفته و سایر عملیات ممکن برای بهبود این کار در بخش ۵,۳ ارائه شده است در این بخش دو خط اول در کد اصلی کامنت شده تا حتی تگ های html که در بعضی از اسناد بود همانگونه بماند . تمام نتایج تا بخش ۵ نیز برحسب همین تابع بیان شده است (Tokenize text using NLTK in python, n.d.).

```
def tokenize_doc(doc):
    """
    IMPLEMENT ME!

    Tokenize a document and return its bag-of-words representation.
    doc - a string representing a document.
    returns a dictionary mapping each word to the number of times it appears in doc.
    """

    clean = re.compile('<.*?>')
    doc=re.sub(clean, '', doc)

    dataset = nltk.sent_tokenize(doc)

    stopwords=set(nltk.corpus.stopwords.words('english'))
    bow=defaultdict(float)

    for i in range(len(dataset)):
        dataset[i]=dataset[i].lower()
        dataset[i] = re.sub(r'\W', ' ', dataset[i])   # Remove all non-word characters
        dataset[i] = re.sub(r'\s+', ' ', dataset[i])  # Remove all punctuations.


        words=nltk.word_tokenize(dataset[i])
        # words = [w for w in words if w not in stopwords]

        for word in words:
            bow[word]+=1

    return bow
```

موقع تست در تابع produce_hw2_results() هم تمام assert ها درست کار کرده و خطایی نداد.

۱,۲

```
for word,count in bow.items():
    self.vocab.add(word)
    self.class_word_counts[label][word] = self.class_word_counts[label][word] + count

self.class_total_doc_counts[label] = self.class_total_doc_counts[label] + 1
self.class_total_word_counts[label] = self.class_total_word_counts[label] + sum(bow.values())
```

برای هر داکیومنت ابتدا BoW را به دست آورده و سپس کلیدهای آن را در مجموعه ی کل لغات اضافه کرده. سپس به تعداد کل داکیومنت های دارای آن label یک واحد اضافه می شود و کل تعداد کلمات آن کلاس دارای label مخصوص با جمع تعداد کل کلمات در BoW جمع می شود. در نهایت برای اینکه بدانیم در یک کلاس یک کلمه چند بار تکرار شده است در حلقه ی بالا تعداد تکرار کلمه را با تعداد تکرارش در BoW جمع می کنیم.

نتایج خواسته شده در سوال ۱,۲ به شرح ذیل است :

```
bita@bita-K401UQK:~/programming/tokenization/hw2$ python nb.py
Starting training with paths /home/bita/programming/tokenization/hw2/large_movie_review_dataset/train/pos
REPORTING CORPUS STATISTICS
NUMBER OF DOCUMENTS IN POSITIVE CLASS: 12500.0
NUMBER OF DOCUMENTS IN NEGATIVE CLASS: 12500.0
NUMBER OF TOKENS IN POSITIVE CLASS: 3061221.0
NUMBER OF TOKENS IN NEGATIVE CLASS: 2997290.0
VOCABULARY SIZE: NUMBER OF UNIQUE WORDTYPES IN TRAINING CORPUS: 74887

VOCABULARY SIZE: 74887
```

۱,۳ پیاده سازی تابع top_n به صورت زیر است.

```
def top_n(self, label, n):
    """
    Implement me!

    Returns the most frequent n tokens for documents with class 'label'.
    """
    counter=collections.Counter(self.class_word_counts[label])
    most_frequent=defaultdict(float)

    for word,count in counter.most_common(n):
        most_frequent[word]=count

    return most_frequent.items()
```

برای محاسبه ی پرتکرارترین کلمات در هر دسته از یک شمارشگر استفاده شده است و یک دیکشنری جدید ساخته شده است که نتایج نهایی در آن ذخیره می شود. جواب حاصل از تست این تابع در صفحه ی بعد آمده است. از نتایج به دست آمده می بینیم که پرتکرارترین کلمات در دسته ی مثبت و منفی کلمات خنثی ای هستند که تاثیری بر روی بایاس متن ندارند. البته Naïve Bayes نسبت به این ویژگی ها حساس نیست و حذف آن ها تاثیر کمی در بهبود دقت خواهد داشت.

```
bita@bita-K401UQK:~/programming/tokenization/hw2$ python nb.py
Starting training with paths /home/bita/programming/tokenization/hw2/large_movie_review_dataset/train/pos a
REPORTING CORPUS STATISTICS
NUMBER OF DOCUMENTS IN POSITIVE CLASS: 12500.0
NUMBER OF DOCUMENTS IN NEGATIVE CLASS: 12500.0
NUMBER OF TOKENS IN POSITIVE CLASS: 3061221.0
NUMBER OF TOKENS IN NEGATIVE CLASS: 2997290.0
VOCABULARY SIZE: NUMBER OF UNIQUE WORDTYPES IN TRAINING CORPUS: 74887

VOCABULARY SIZE: 74887

TOP 10 WORDS FOR CLASS pos :
the  :  173344.0
and  :  89747.0
a  :  73.0
of  :  76855.0
to  :  66749.0
is  :  57247.0
in  :  50221.0
br  :  49235.0
it  :  48074.0
i  :  40774.0

TOP 10 WORDS FOR CLASS neg :
the  :  163405.0
a  :  79400.0
and  :  74393.0
of  :  69009.0
to  :  68975.0
br  :  52636.0
is  :  50085.0
it  :  48393.0
i  :  46916.0
in  :  43755.0

[done.]
```

۱,۴

خیر، همان طور که در شکل فوق مشاهده می شود کلمات بسیار شبیه هم می باشد حتی تگ br html حذف نشده است. اما می دانیم Naïve Bayes نسبت به این گونه ویژگی ها (غیر مرتبط) بایاس زیادی ندارد و در نتیجه ی نهایی تاثیر زیادی نخواهد گذاشت.

بخش دوم :

۲,۱ احتمال خواسته شده برابر تعداد تکرار آن کلمه در دسته ی مورد نظر تقسیم بر تعداد کل کلمات در آن دسته.

```
def p_word_given_label(self, word, label):

    """
    Implement me!

    Returns the probability of word given label (i.e., P(word|label))
    according to this NB model.
    """

    return self.class_word_counts[label][word]/self.class_total_word_counts[label]
```

۲,۲

```
fantastic probability given positive documents :  0.00021364024354987763
fantastic probability given negative documents :  4.804339920394757e-05

boring probability given negative documents :  0.0004934457459905447
boring probability given positive documents :  0.00010878012400934137
```

همان طور که مشاهده می شود احتمال وقوع کلمه ی fantastic در دسته ی مثبت ها بیشتر از منفی هاست و احتمال وقوع کلمه boring در دسته ی منفی ها بیشتر از مثبت هاست. نتایج مطابق شهود ما می باشد.

۲,۳

در این صورت احتمال برابر صفر خواهد بود چون صورت کسر در داده ها موجود نمی باشد. مانند کلمه ی boring_movie که احتمالش برابر صفر شد. راه حل رفع این مشکل در بخش بعد آمده است.

```
boring_movie probability given negative documents :  0.0
```

۲,۴

برای رفع این مشکل مقدار pseudocount اضافه می کنیم تا حتی اگر کلمه ی در داده ی تست ما نبود با مشکل مواجه نشویم و یک احتمال مینیمم برایش در نظر گرفته شود. این مقدار در این جا برابر ۱ که Laplace Smoothing Parameter است درنظر گرفته شده است. بنابراین فرمول نهایی به شکل زیر درخواهد آمد(Stanford University, n.d.) .

$$\hat{P}(w_i \mid c) = \frac{count(w_i, c) + 1}{\sum_{w \in V} \left(count(w, c) + 1 \right)}$$

$$= \frac{count(w_i, c) + 1}{\left(\sum_{w \in V} count(w, c) \right) + |V|}$$

در نتیجه :

- Calculate $P(c_j)$ terms
  - For each c_j in C do

    $docs_j \leftarrow$ all docs with class $= c_j$

$$P(c_j) \leftarrow \frac{|docs_j|}{|\text{total \# documents}|}$$

- Calculate $P(w_k \mid c_j)$ terms
  - $Text_j \leftarrow$ single doc containing all $docs_j$
  - For each word w_k in *Vocabulary*

    $n_k \leftarrow$ # of occurrences of w_k in $Text_j$

$$P(w_k \mid c_j) \leftarrow \frac{n_k + \alpha}{n + \alpha |Vocabulary|}$$

پیاده سازی تابع به صورت زیر خواهد بود.

```
def p_word_given_label_and_psuedocount(self, word, label,alpha):
    """
    Implement me!

    Returns the probability of word given label wrt psuedo counts.
    alpha - psuedocount parameter
    """
    return (self.class_word_counts[label][word]+alpha)/(self.class_total_word_counts[label]+(len(self.vocab)*alpha))
```

درواقع باید انتظار داشت که مقدار احتمال های محاسبه شده با پارامتر آلفا و بدون آن در شکل زیر مشاهده می شود.

```
fantastic probability given positive documents :  0.00021364024354987763
fantastic probability given negative documents :  4.804339920394757e-05

boring probability given negative documents :  0.0004934457459905447
boring probability given positive documents :  0.0001078012400934137

fantastic probability given pseudocount in positive documents :  0.00020885760311825996
fantastic probability given pseudocount in negative documents :  4.719780142875882e-05

boring probability given pseudocount in negative documents :  0.00048174307665215904
boring probability given pseudocount in positive documents :  0.00010650143426183027
```

ولی به هر حال احتمال وقوع کلمه ی fantastic در دسته ی مثبت و احتمال وقوع کلمه ی boring در دسته ی منفی بیشتر خواهد بود. احتمال وقوع boring_movie که در بخش قبل صفر بود به مینیمم احتمال ممکن ست شد.

```
boring_movie probability given pseudocount in negative documents :  3.255020788190264e-07
```

بخش سوم :

۳,۱

$$\log P(w_1, w_2, \dots, w_{d_n} \mid y_d) = \log \prod_{i=1}^{n} P(w_{d_i}|y_i) = \sum_{i \in positions} \log p(w_{d_i}|y_i)$$

y_i داکیومنت مربوطه و w_{d_i} کلمه در داکیومنت مربوطه می باشد.

۳,۲ این تابع دقیقا مجموع احتمالات که از فرمول بالا به دست آمده بود را برمی گرداند.

```
def log_likelihood(self, bow, label, alpha):

    """
    Computes the log likelihood of a set of words give a label and psuedocount.
    bow - a bag of words (i.e., a tokenized document)
    label - either the positive or negative label
    alpha - float; psuedocount parameter
    """
    ln_likelihood = 0.0

    for word in bow.keys():
        ln_likelihood += math.log(self.p_word_given_label_and_psuedocount(word, label, alpha))
```

۳,۳ این تابع تعداد لگاریتم کل داکیومنت ها در دسته ی مربوطه تقسیم بر کل تعداد داکیومنت های training را برمی گرداند.

```
def log_prior(self, label):

    """
    Implement me!

    Returns a float representing the fraction of training documents
    that are of class 'label'.
    """
    return math.log(self.class_total_doc_counts[label]/(self.class_total_doc_counts[POS_LABEL]+self.class_total_doc_counts[NEG_LABEL]))
```

بخش چهارم(Kumar, n.d.) :

۴,۱ مقدار احتمال توسط قضیه ی احتمال کل به دست می آید.

$$\boldsymbol{p(w_d) = P(w_d|POS\_LABEL)\, p(POS\_LABEL) + P(w_d|NEG\_LABEL)p(NEG\_LABEL)}$$

۴,۲

$$\boldsymbol{C_{NB} = log\ P(y_d|w_d) = arg\ max\ log\ p(y_d) + \sum_{i=1}^{n} log\ p(w_i|y_i)\ - log\ p(w_d)}$$

۴,۳ در واقع $\boldsymbol{log\ p(w_d)}$ یک عدد ثابت است و عدد ثابت برای هر داکیومنت در فرمول فوق کم شود تاثیری در بزرگی عدد به دست آمده و تصمیم گیری نهایی نخواهد داشت. بنابراین الزامی به نرمالیزه کردن نمی باشد. در واقع به این طریق نیز می توان $\boldsymbol{C_{NB}}$ را محاسبه کرد :

$$\boldsymbol{C_{NB} = log\ P(y_d|w_d) = arg\ max\ log\ p(y_d) + \sum_{i=1}^{n} log\ p(w_i|y_i)}$$

۴,۴

```
def unnormalized_log_posterior(self, bow, label, alpha):

    """
    Implement me!

    alpha - psuedocount parameter
    bow - a bag of words (i.e., a tokenized document)
    Computes the unnormalized log posterior (of doc being of class 'label').
    """
    return self.log_likelihood(bow,label,alpha)+self.log_prior(label)
```

۴,۵

```
def classify(self, bow, alpha):
    """
    Implement me!

    alpha - psuedocount parameter.
    bow - a bag of words (i.e., a tokenized document)

    Compares the unnormalized log posterior for doc for both the positive
    and negative classes and returns the either POS_LABEL or NEG_LABEL
    (depending on which resulted in the higher unnormalized log posterior).
    """
    pos_posteriori = self.unnormalized_log_posterior(bow,POS_LABEL,alpha)
    neg_posteriori = self.unnormalized_log_posterior(bow,NEG_LABEL,alpha)

    if pos_posteriori>neg_posteriori:
        return POS_LABEL
    elif pos_posteriori<neg_posteriori:
        return NEG_LABEL
    else:
        random.choice([POS_LABEL,NEG_LABEL])
```

همان طور که در تصویر فوق مشاهده می شود اگر احتمال پسین نظر مثبت بیشتر باشد، داکیومنت در دسته ی نظرهای مثبت قرار می گیرد. درصورتی احتمال پیسین نظر منفی بیشتر باشد، در دسته ی نظر منفی قرار می گیرد. در صورت تساوی این دو مقدار به صورت رندوم داکیومنت در دسته ی مثبت یا منفی قرار خواهد گرفت.

بخش پنجم:

۵,۱

```
def evaluate_classifier_accuracy(self, alpha):

    pos_classified={POS_LABEL:0,NEG_LABEL:0}
    neg_classified={POS_LABEL:0,NEG_LABEL:0}


    pos_path = os.path.join(TEST_DIR, POS_LABEL)
    neg_path = os.path.join(TEST_DIR, NEG_LABEL)
    pos_length=0
    neg_length=0
    print ("Starting testing with paths %s and %s" % (pos_path, neg_path))
    print()
    for (p, label) in [ (pos_path, POS_LABEL), (neg_path, NEG_LABEL) ]:
        filenames = os.listdir(p)
        counter = 5
        length=len(filenames)

        if label ==POS_LABEL:
            pos_length=length
        else:
            neg_length=length
        for f in filenames:
            with open(os.path.join(p,f),'r') as doc:
                content = doc.read()
                bow=tokenize_doc(content)
                classified_label=self.classify(bow,alpha)

                if label==POS_LABEL:
                    pos_classified[classified_label]+=1
                else:
                    neg_classified[classified_label]+=1

                if classified_label!=label and counter:
                    print(f," file in ",label,"class is worngly detected.")
                    counter=counter-1
        print()

    print("Number of positive missclassified examples :",pos_classified[NEG_LABEL])
    print("Number of negative missclassified examples :",neg_classified[POS_LABEL])
    print()

    return pos_classified[POS_LABEL]/self.class_total_doc_counts[POS_LABEL], neg_classified[NEG_LABEL]/self.class_total_doc_counts[NEG_LABEL],\
          (pos_classified[POS_LABEL]+neg_classified[NEG_LABEL])/sum(self.class_total_doc_counts.values())
```

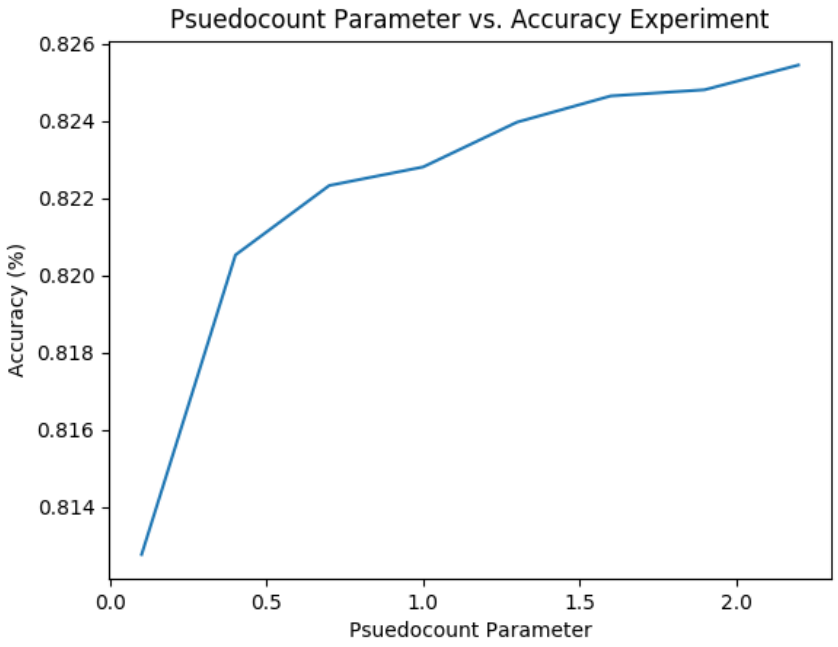
در این تابع داده های تست مورد ارزیابی قرار می گیرند و با توجه به روش احتمال پسین تعیین می شود که الگوریتم آن ها را در کدام دسته قرار می دهد. در اینجا برای دید بهتر میزان دقت و درست سنجی نظرهای مثبت و منفی جداگانه حساب شده و می توان دقت کلی الگوریتم را از روی مقادیر مربوط به هرکدام به دست آورد. مقدار آلفا در این مرحله برابر یک بوده است.

```
Positive class accuracy with alpha = 1 is :  0.74808
Negative class accuracy with alpha =  1 is :  0.89752
Total accuracy with alpha = 1 is :  0.8228
```

۵،۲ با افزایش مقدار آلفا دقت بهبود پیدا کرد اما پس از مقدار ۱ تقریبا پیشرفت دقت خیلی کمتر شد.

```
pseudocounts=np.arange(0.1,2.3,0.3)
accuracies=[]
for i in pseudocounts:
    _,_,accuracy=nb.evaluate_classifier_accuracy(i)
    accuracies.append(accuracy)

plot_psuedocount_vs_accuracy(pseudocounts.tolist(),accuracies)
```



۵،۳ من از روی دیتاست اصلی ۵ نظر در دسته ی مثبت و ۵ نظر در دسته ی منفی به ترتیب وقوع پیدا کردم که اشتباه classify شده بودند. نتایج اولیه به صورت زیر بود. در دسته ی مثبت ۳۱۴۹ نمونه اشتباه شناسایی شدند و در دسته ی منفی ۱۲۸۱ نمونه اشتباه شناسایی شدند. یعنی ۳۱۴۹ نمونه باید مثبت شناسایی می شدند ولی الگوریتم منفی تشخیص داد و ۱۲۸۱ نمونه باید منفی شناسایی می شدند ولی الگوریتم مثبت تشخیص داد.

```
7977_7.txt  file in  pos class is worngly detected.
3412_7.txt  file in  pos class is worngly detected.
9048_7.txt  file in  pos class is worngly detected.
11157_9.txt  file in  pos class is worngly detected.
1102_9.txt  file in  pos class is worngly detected.

6932_2.txt  file in  neg class is worngly detected.
10158_2.txt  file in  neg class is worngly detected.
2947_4.txt  file in  neg class is worngly detected.
9824_1.txt  file in  neg class is worngly detected.
721_4.txt  file in  neg class is worngly detected.

Number of positive missclassified examples : 3149
Number of negative missclassified examples : 1281
```

برای بهبود دقت، من ابتدا کلمات ..a,the,this, که جزء stopword به حساب می آیند و تاثیر چندانی روی نظر کاربر ندارند با دستور مربوط به خودش حذف کردم. می توان از تکنیک های stemming و lemmatization استفاده کرد تا لغاتی که ریشه ی یکسان دارند به عنوان یک کلمه حساب شوند مانند like, liking,….این دو روش خیلی شبیه هم اند ولی خروجی lemmatizer در دیکشنری یک کلمه ی معتبر است ولی stemmer لغت را تا بیشترین حد ممکن مختصر می کند و ممکن است نتیجه در دیکشنری یافت نشود مثلا compute و computing را comput در نظر می گیرد. من شخصا از lemmatizer استفاده کردم(Gupta, 2018) . تابع tokenize_doc به صورت زیر تغییر پیدا کرد. اما در کد تحویلی این خطوط کامنت شده اند چون موقع assert کلماتی مانند the,this با خطا مواجه می شدند چون این کلمات جزء stopword ها هستند.

```
dataset = nltk.sent_tokenize(doc)

stopwords=set(nltk.corpus.stopwords.words('english'))
bow=defaultdict(float)

for i in range(len(dataset)):
    dataset[i]=dataset[i].lower()
    dataset[i] = re.sub(r'\W', ' ', dataset[i])    # Remove all non-word characters
    dataset[i] = re.sub(r'\s+', ' ', dataset[i])   # Remove all punctuations.

    #swap commenting these two lines for seeing results with lemmatization and without any filtering

    lemmatized_sentence=lemmatize_sentence(dataset[i])
    words=nltk.word_tokenize(lemmatized_sentence)
    # words=nltk.word_tokenize(dataset[i])
    words = [w for w in words if w not in stopwords]

    for word in words:
        bow[word]+=1

return bow
```

دقت به میزان کمی بهبود پیدا کرد اما نباید انتظار تغییر چشمگیر در دقت را داشت چون روش Naïve Bayes نسبت به ویژگی های بی ربط بایاسی ندارد. نتایج به صورت زیر تغییر یافت:

```
bita@bita-K401UQK:~/programming/NaiveBayes/hw2$ python nb.py
Starting training with paths /home/bita/programming/NaiveBayes/hw2/large_movie_r
REPORTING CORPUS STATISTICS
NUMBER OF DOCUMENTS IN POSITIVE CLASS: 12500.0
NUMBER OF DOCUMENTS IN NEGATIVE CLASS: 12500.0
NUMBER OF TOKENS IN POSITIVE CLASS: 1590817.0
NUMBER OF TOKENS IN NEGATIVE CLASS: 1532789.0
VOCABULARY SIZE: NUMBER OF UNIQUE WORDTYPES IN TRAINING CORPUS: 64474

VOCABULARY SIZE: 64474
```

نتیجه می گیریم تعداد لغات یکتا به ۶۴۴۷۴ عدد کاهش یافت. اما اجرا کردن lemmatization با یافتن ریشه ی کلمات در جمله زمان زیادی برد. پیاده سازی این روش هم در کد مربوطه آمده است.

```
TOP 10 WORDS FOR CLASS pos :
br   :  49235.0
film   :  25169.0
movie   :  22665.0
one   :  14173.0
see   :  10888.0
make   :  10526.0
like   :  10215.0
good   :  9289.0
time   :  8378.0
get   :  8264.0

TOP 10 WORDS FOR CLASS neg :
br   :  52636.0
movie   :  29050.0
film   :  22762.0
one   :  13575.0
make   :  12480.0
like   :  12083.0
bad   :  10952.0
see   :  9878.0
get   :  9815.0
good   :  9631.0
```

۱۰ پرتکرارترین لغات در هر دسته به شکل روبرو تغییر یافت. حالااین لغات جدید حس

بهتری نسبت به نتایج قبلی به ما می دهد. اما باز هم نتایج شبیه هم است.

سایر مقادیر مشابه قسمت های قبل با روش زیر این نتایج را داد:

```
fantastic probability given positive documents :  0.000411109511653446
fantastic probability given negative documents :  9.3946394448290001e-05

boring probability given negative documents :  0.0006628440052740462
boring probability given positive documents :  0.00013137903354062724

fantastic probability given pseudocount in positive documents :  0.00039570081635192846
fantastic probability given pseudocount in negative documents :  9.078029103535235e-05

boring probability given pseudocount in negative documents :  0.0006367141791927816
boring probability given pseudocount in positive documents :  0.00012686591058611447
```

```
7977_7.txt  file in  pos class is worngly detected.
3412_7.txt  file in  pos class is worngly detected.
9048_7.txt  file in  pos class is worngly detected.
11157_9.txt  file in  pos class is worngly detected.
1102_9.txt  file in  pos class is worngly detected.

12301_4.txt  file in  neg class is worngly detected.
10158_2.txt  file in  neg class is worngly detected.
2947_4.txt  file in  neg class is worngly detected.
1752_3.txt  file in  neg class is worngly detected.
11637_4.txt  file in  neg class is worngly detected.

Number of positive missclassified examples : 2737
Number of negative missclassified examples : 1492
```

در هر دو روش ۵ فایل اولی که اشتباه تشخیص داده شده اند، آمده است. می بینیم فایل 6932_2.txt در دسته ی منفی ها این بار توسط الگوریتم درست تشخیص داده شده است. به طور کلی تعداد خطاهای دسته ی مثبت به ۲۷۳۷ کاهش یافته که خوب است. خطای دسته ی منفی حدود ۱۰۰ مقدار زیاد شده و به ۱۴۹۲ رسیده. در کل میزان دقت به ۰,۸۳۰۸۴ افزایش یافته.

```
Positive class accuracy with alpha = 1 is :  0.78104
Negative class accuracy with alpha =  1 is :  0.88064
Total accuracy with alpha = 1 is :  0.83084
```

بخش ششم :

۶,۱

کمترین مقدار ممکن برای تابع وقتی است که صورت کمترین مقدار ممکن باشد و مخرج بیشترین مقدار ممکن. مخرج را اگر با یک تقریب بزنیم(از p_given_label_and_pseudocount استفاده کنیم و فرض کنیم تمام کلمات اسناد در آن دسته را آن کلمه تشکیل داده باشد). مقدار مینیمم برابر احتمال وقوع boring_movie که در بخش ۲,۴ محاسبه شد، می باشد که ۰,۰۰۰۰۰۰۰۳۲۲۵. ماکزیمم مقدار هم عکس مقدار مینیمم می باشد(با فرض اینکه صورت بیشترین مقدار باشد و با یک تقریب زده شود) حاصل برابر ۳۱،۰۰۷،۷۵۱،۹۳۷۹ است. البته این اعداد حدودی هستند.

۶,۲

```
def likelihood_ratio(self, word, alpha):
    """
    Implement me!

    alpha - psuedocount parameter.
    Returns the ratio of P(word|pos) to P(word|neg).
    """
    LR = self.p_word_given_label_and_psuedocount(word,POS_LABEL,alpha)/\
         self.p_word_given_label_and_psuedocount(word,NEG_LABEL,alpha)
    return LR
```

۶,۳ همان طور که به طریق شهودی می توان گفت LR کلمه ی fantastic ، 4 است بدین معنی که محتمل تر است که داکیومنت مربوط به آن لغت یک نظر مثبت باشد. برعکس LR کلمه ی boring کمتر از یک است بدین معنی که این کلمه در داکیومنت های با نظر منفی بیشتر محتمل است بیاید.

برای کلمه های the و to یافتن این نسبت بی معنی است چون مقدار پارامترشان به ۱ خیلی نزدیک است و این یعنی کلمه ی مورد نظر بایاس ندارد و نمی توان تعیین کرد در نظر مثبت رخ داده یا منفی. بنابراین پارامتر LR برای چنین کلماتی بی معنی است.

```
likelihood ratio of 'fantastic':  4.425155341896873
likelihood ratio of 'boring':  0.2210751735177074

likelihood ratio of 'the':  1.039198576967064
likelihood ratio of 'to':  0.9480003281456533

[done.]
```

۶,۴ همان طور که در صفحه ی پیش استدلال شد، لغت با LR=1 یعنی کلمه روی مثبت یا منفی بودن نظر کاربر تاثیری ندارد. اما کلمه با LR=100 بدین معنی نیست که ۱۰۰ برابر محتمل تر است این کلمه در دسته ی مثبت باشد. گفته می شود این نسبت هر قدر هم بالاتر برود، افزایش احتمال رویداد آن بالاتر از ۴۵ درصد نمی شود. اما خیلی محتمل تر است که لغت در نظر مثبت رخ دهد. به همین دلیل LR=0.01 هم دلیلی نمی شود که احتمال آمدن کلمه در دسته ی منفی ۱۰۰ برابر دسته ی مثبت شود.

بخش هفتم :

۷,۱

برنامه طوری که برای چندکلاسه ها هم بتواند کار کند در multiclassnb.py پیاده سازی شده است. اعداد به دست آمده به شرح زیر بود(جداسازی مانند بخش ۱ تا ۵ برحسب فاصله بوده است)پارامتر آلفا یک در نظر گرفته شده و دقت به دست آمده ۰,۸۲۲۸ بوده است(Kovachi, 2018) .

```
bita@bita-K401UQK:~/programming/NaiveBayes/hw2$ python multiclassnb.py
REPORTING CORPUS STATISTICS
NUMBER OF DOCUMENTS IN  NEG  CLASS: 12500
NUMBER OF DOCUMENTS IN  POS  CLASS: 12500
NUMBER OF TOKENS IN  NEG  CLASS: 2997290.0
NUMBER OF TOKENS IN  POS  CLASS: 3061221.0
VOCABULARY SIZE: NUMBER OF UNIQUE WORDTYPES IN TRAINING CORPUS: 74887
VOCABULARY SIZE: 74887

TOP 10 WORDS FOR CLASS NEG :
the  :  163405.0
a  :  79400.0
and  :  74393.0
of  :  69009.0
to  :  68975.0
br  :  52636.0
is  :  50085.0
it  :  48393.0
i  :  46916.0
in  :  43755.0

TOP 10 WORDS FOR CLASS POS :
the  :  173344.0
and  :  89747.0
a  :  83723.0
of  :  76855.0
to  :  66749.0
is  :  57247.0
in  :  50221.0
br  :  49235.0
it  :  48074.0
i  :  40774.0


Total accuracy with alpha = 1 is :  0.8228

[done.]
```

۷,۲ خیر در فرمول نرمالیزر تغییری ایجاد نمی شود وعدد نهایی به دست آمده که با احتمال کل به دست می آید ثابت خواهد بود و مانند حالت دو کلاسه (مثبت و منفی) است. اما چون در تمام دسته ها یک عدد ثابت است، محاسبه ی آن در تصمیم گیری نهایی تاثیری ندارد. (مثل اینکه یک عدد ثابت را با تمام نمونه ها جمع کنیم یا از آن کم کنیم یا در صورت در نظر نگرفتن لگاریتم یک عدد مثبت را ضرب کنیم در مقایسه بزرگی اعداد تاثیری نخواهد داشت.)

$$p(w_d) = P(w_d|LABEL_1)\,p(LABEL_1) + P(w_d|LABEL_2)p(LABEL_2) + \cdots + P(w_d|LABEL_n)P(LABEL_n)$$

۷,۳ برحسب تعداد کلاس ها، هر کلاس که احتمال پسین بیشتری داشت محتمل تر است. بنابراین باید بین احتمال پسین کلاس ها ماکزیمم بگیریم.

```
def classify(self, bow, alpha):

    labels_posteriori=defaultdict(int)
    for label in self.class_word_counts.keys():
        labels_posteriori[label]=self.unnormalized_log_posterior(bow,label,alpha)

    max_posteriori = max(labels_posteriori.items(), key=lambda x: x[1])          #return label with maximum value

    return max_posteriori[0]
```

مراجع :

(n.d.). Retrieved from https://www.geeksforgeeks.org/tokenize-text-using-nltk-python/

Gupta, G. (2018, September 27). Retrieved from https://medium.com/: https://medium.com/@gaurav5430/using-nltk-for-lemmatizing-sentences-c1bfff963258

Kovachi, J. M. (2018, January 21). *Implementing a Multinomial Naive Bayes Classifier from Scratch with Python*. Retrieved from https://medium.com/: https://medium.com/@johnm.kovachi/implementing-a-multinomial-naive-bayes-classifier-from-scratch-with-python-e70de6a3b92e

Kumar, A. (n.d.). *Naive Bayes Classifier: Calculation of Prior, Likelihood, Evidence & Posterior*. Retrieved from https://medium.com: https://medium.com/@abhishek.km23/naive-bayes-classifier-calculation-of-prior-likelihood-evidence-posterior-74d7d27eec24

Stanford University. (n.d.). *The Task of Text Classification*. Retrieved from web.stanford.edu: https://web.stanford.edu/class/cs124/lec/naivebayes.pdf

Tokenize text using NLTK in python. (n.d.). Retrieved from geeksforgeeks.org: https://www.geeksforgeeks.org/tokenize-text-using-nltk-python/