Project 1 Report

2018312280
이상수
2023/4/12
Introduction to Artificial Intelligence_SWE3011_41(박호건)

For safe running, I copied the contents of original file to train_data.txt.
Then I pre-processed data moving it back and forth from train_data.txt to train_data2.txt
Therefore, after running the code, there will be two garbage files in the directory.

```
# copy from original
for line in train_ori:
    train_data.write(line)
train_ori.close()
```

Task 1

a. Convert all text to lowercase

```
# data to lower case
train_data.seek(0)
for line in train_data:
    b: str = line.lower()
    train_data2.write(b)
```

Just used line.lower()

b. Remove special characters.

```
# Remove special characters: ' ', !, ", #, $, %, &, ', (, ), *, +, , ,-, .
special_chars: list[str] = ['!', '"', '#', '$', '%', '&', "'",
                '(', ')', '*', '+', ',', '-', '.', '/', ':', ';', '<', '=
for line in train_data:
    if len(line)>1:
        if line[1] == ',':
            if line[0]=='5': #change start of review to noticeable value
                line: str = "GgOoDdSsTtAaRrTt" + line[1:]
            elif line[0]=='1':
                line: str = "BbAaDdSsTtAaRrTt" + line[1:]
        for w in special_chars:
            line: str=line.replace(w, ' ')
```

When doing this, I needed to change 1 and 5 stars to key words that could not be possible for person to use in review so even after processing I can know where the start is of the reviews.

c. Tokenize the text into words.

```
# tokenize words
comments: list = []
onereview: list = []
for line in train_data:
    line: list[str] = line.split()
    if line and len(line) > 0:
        word: str = line[0]
        if word in ("GgOoDdSsTtAaRrTt", "BbAaDdSsTtAaRrTt") and len(onereview) != 0:
            addinglist: list = []
            addinglist: list = onereview.copy()
            comments.append(addinglist)
            onereview.clear()
        onereview.extend(line)
    else:
        continue
```

Split each line and make a list for each review and add it to bigger list

d. Remove stop words.

```
# Remove stop words from stopwords.txt
stopfile: TextIOWrapper = open('stop words.txt', 'r')
stop_words: list[str] = stopfile.read().splitlines()
stopfile.close()

for word in stop_words:
    word: str = word.strip()

reviews: list = []
for line in comments:
    areview: list = []
    for word in line:
        word: Any = word.strip()
        if word not in stop_words:
            areview.append(word)
    reviews.append(areview)
return reviews
```

For each list of reviews, make new list without stop words and add it to new bigger list

e. Select 1000 most frequently appeared words for the final features

```
# select 1000 frequent words from final
words_dic: dict = {}
for line in reviews:
    for word in line:
        if word is line[0] and (word in ("GgOoDdSsTtAaRrTt", "BbAaDdSsTtAaRrTt")):
            continue
        else:
            if word in words_dic:
                words_dic[word] += 1
            else:
                words_dic[word] = words_dic.get(word, 0)+1
ranked_dic: list[tuple] = sorted(words_dic.items(), key=lambda x: x[1], reverse=True)
dictionary: list[tuple] = ranked_dic[:1000] #selected top 1000 words
```

Make new list of tuple (word, value) with top 1000 frequencies using dictionary(words_dic).
Last variable named 'dictionary' isn't actually dictionary right now but it changes later and it is used for checking 1000 words for rest of the code

Please print out the top 20-50 words from the selected features.

```
#print 50 words from top
i=1
print("top 50 words: ")
for word, val in dictionary[:50]:
    print(i, end='')
    print(': '+word)
    i+=1
```

Result:
top 50 words:
1: food
2: place
3: just
4: like
5: here
6: time
7: back
8: great
9: service
10: can
11: will

12: ve
13: only
14: don
15: went
16: know
17: people
18: best
19: who
20: well
21: order
22: told
23: didn
24: going
25: first
26: am
27: love
28: down
29: staff
30: minutes
31: ordered
32: now
33: way
34: day
35: chicken
36: restaurant
37: came
38: 2
39: nice
40: car
41: take
42: still
43: see
44: asked
45: little
46: store
47: made
48: try
49: want
50: experience

For next codes, I made 2 functions :

def process : need (reviews)

reviews = total reviews for evaluation (10%, 30%, 50%, 70%, 100% of train.csv reviews after pre-process)

```
    good_dict = {} #each frequency of words in good reviews (words in G)
    bad_dict = {} #each frequency of words in bad reviews (words in B)
    total_good_words = 0 # total freqeuncy of all words in good reviews (Wg)
    total_bad_words = 0 #total freqeuncy of all words in bad reviews (Wb)
    total_messages = 0 #total count of reviews (M)
    total_good_message = 0 #total count of good reivews (G)
    total_bad_message = 0 #total count of bad reviews (B)
```

Laplace smoothing

```
# Laplace smoothing
    for word in dictionary:
        total_good_words += 1
        total_bad_words += 1
        good_dict[word] = 1
        bad_dict[word] = 1
```

for each line(review) in reviews, check if line[0](first word) is that from above that represents 5, 1 starts. Then, add 1 to appropriate values for total words and each word. Also count total messages and good and bad messages

```
# get good and bad reviews
    for line in reviews:
        total_messages += 1 # total messages +1
        if line[0] == "GgOoDdSsTtAaRrTt":
            total_good_message += 1 # total good messages +1
            for word in line[1:]:
                if word in dictionary:
                    total_good_words += 1 # total frequency of words in good
review +1
                    good_dict[word] += 1 # frequency of certain word in good
review +1
        elif line[0] == "BbAaDdSsTtAaRrTt":
            total_bad_message += 1 # total bad messages +1
            for word in line[1:]:
                if word in dictionary:
                    total_bad_words += 1  # total frequency of words in bad
review +1
                    bad_dict[word] += 1 # frequency of certain word in bad
review +1
```

Get two probabilities of each message (good and bad)

```
#set ratio of two kinds of messages
    # p(B)=(B)/(M)
    ratio_of_good_messages = total_good_message/total_messages
    # p(G)=(G)/(M)
    ratio_of_bad_messages = total_bad_message/total_messages
```

get probabilities of each words to total words in good and bad reviews

```
#set ratio of each word
    for word in dictionary:
        good_ratio.setdefault(word, 0)
        bad_ratio.setdefault(word, 0)
        good_ratio[word] = good_dict[word]/total_good_words  # p(w | G)
        bad_ratio[word] = bad_dict[word]/total_bad_words  # p(w | B)
```

Call test_process for testing (explained below)
Return result from test

def test_process : need (ratio_of_good_messages, ratio_of_bad_messages, good_ratio, bad_ratio)
ratio_of_good_messages = good messages/total messages from train.csv p(G)
ratio_of_bad_messages = bad messages/total messages from train.csv p(B)
good_ratio = list that has each ratio of words given good messages from train.csv p(w | G)
bad_ratio = list that has each ratio of words given bad messages from train.csv p(w | B)

Pre-process test.csv (Same as before. Use copied file)
Delete all words from test.csv that are not included in top 1000 words for convenience.

Procedure of Evaluation :
1. Initially make two values good and bad
"good" is point of good review
"bad" is point of bad review
At the end of evaluation, if good >= bad program assume that answer is good review

3. multiply ratio for each messages possibility

```
for line in test_reviews:
           good = 1 # evaluation value for good
            bad = 1  # evaluation value for bad
            good *= ratio_of_good_messages
            bad *= ratio_of_bad_messages
```

4. If line[0] is 5star or 1star, set answer to 5 or 1

Then, for words in line, multiply ratio of good and bad words in good and bad

```
          if line[0] == "GgOoDdSsTtAaRrTt": # answer is good review
                answer = 5
            elif line[0] == "BbAaDdSsTtAaRrTt":  # answer is bad review
                answer = 1
            for word in line:
                if word is line[0] and (word in ("GgOoDdSsTtAaRrTt",
"BbAaDdSsTtAaRrTt")):
                    pass
                else:
                    good *= good_ratio[word]
                    bad *= bad_ratio[word]
```

5. If good>=bad evaluation is 5 and else 1

```
            if good >= bad:
                asumption = 5
            else:
                asumption = 1
```

6. Check if evaluation was right

```
            if answer == asumption:
                right += 1
            else:
                wrong += 1
```

7. Accuracy was :

$$Accuracy = \frac{TrueNegatives + TruePositive}{TruePositive + FalsePositive + TrueNegative + FalseNegative}$$

Which means it is just right/(right+wrong), return the result to process function

```
# for evaluation: accuracy:
    #(right good review + right bad review)/(all result=(right good + right
bad + wrong good + wrong bad))
    result = right/(right+wrong)
    return result
```

Rest of the code is just testing using function above and plotting using matplot

```
#print("400")
result_10=process(reviews[:int(len(reviews)*10/100)])
#print("1200")
result_30=process(reviews[:int(len(reviews)*30/100)])
#print("2000")
result_50=process(reviews[:int(len(reviews)*50/100)])
#print("2800")
result_70=process(reviews[:int(len(reviews)*70/100)])
#print("4000")
result_100=process(reviews[:int(len(reviews))])

print(result_10*100,end="%\n")
print(result_30*100, end="%\n")
print(result_50*100, end="%\n")
print(result_70*100, end="%\n")
print(result_100*100,end="%\n")

ypoints = np.array([result_10*100, result_30*100,
                    result_50*100, result_70*100, result_100*100])
xpoints=np.array([400, 1200, 2000, 2800, 4000])

plt.plot(xpoints, ypoints, linestyle='dotted')

plt.show()
```

Result output :

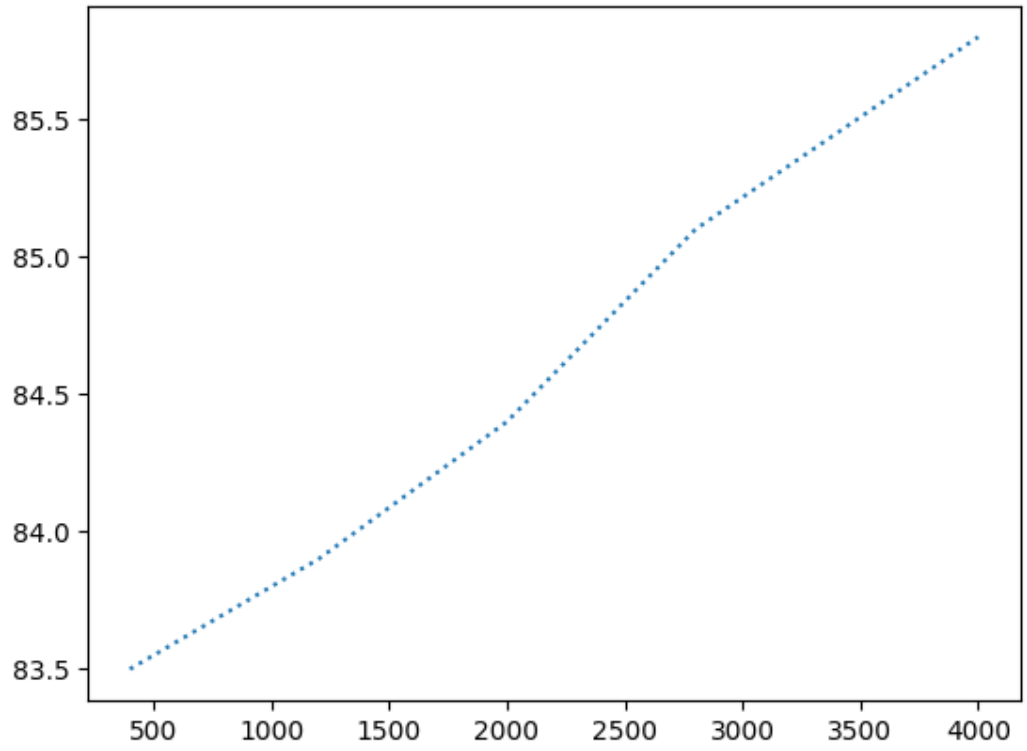
83.5%

83.89999999999999%

84.39999999999999%

85.1%

85.8%



We can see that learning curve goes higher as the train data increases from 400 reviews to 4000 reviews.

The reason why the accuracy doesn't change much even when data has increased almost 10 times is that initial possibility for good and bad messages (p(G) and p(B)) has much higher value (almost 50% most of the time) than possibility of each words given the message type (p(w | G) and p(w | B)) (less than 1% most of the time because there are 1000 kinds of words)

And we used points for evaluation like "good=1*p(G) * p(w1 | G) *p(w2 | G) ..."

Therefore, even when data increases, if ratio of good and bad messages are like before, result is very similar to before.

Reference

[1] Accuracy, Recall & Precision. Erika D. Dec 9, 2019

https://medium.com/@erika.dauria/accuracy-recall-precision-80a5b6cbd28d

(Just a same site from project 1 pdf from icampus that explain accuracy for using the image file from it in report)