Assignment2 보고서

영화 리뷰 긍정, 부정 분류

2016025496 서유림

1. 코드 설명

```
from konlpy.tag import Twitter

twitter = Twitter()

def tokenize(doc):
  # norm, stem은 optional
  return ['/'.join(t) for t in twitter.pos(doc)]
```

classify의 정확도를 높이기 위해 konlpy라는 모듈의 tokenizer를 사용해 보았다.
문서를 tokenize한 후에 해당 단어들의 품사를 / 뒤에 이어붙여 주었다. 같은 단어라도 품사에 따라 다르게 쓰이기 때문이다.

그리고 NaiveBayesClassifier 라는 클래스를 만들었다.
클래스에는 init, load_corpus, count_words, word_probabilities, compare_probability, train, classify 함수 들이 있다.

init 함수는 class로 객체를 만들 때 실행되는 함수인데, word_probs 리스트와 smoothing을 위한 k값을 세팅하도록 하였다.

```
def load_corpus(self, path):
    #train 할 doc들을 읽어오는 함수
    corpus = []
    f = open(path, "r")
    # 맨 첫줄은 id, documnet, label 이
    f.readline()

    while  True:
        line = f.readline()
        if not line : break;
        # tab으로 구분 되어 있으니 tab을
        temp = line.split('\t')
        val = temp[2][0]
        temp.pop(2)
        # 0번째는 다 그냥 id값이니 빼버린다
        temp.pop(0)
        # readline으로 읽어온 후 split을
        temp.append(float(val))
        corpus.append(temp)
    f.close()
    return corpus
```

load_corpus함수는 내가 train할 document를 읽어오는 함수이다.
traindata들은 tab으로 구분되어 있기 때문에 일단 '\t'를 기준으로 split해서 temp라는 임시 리스트에 담는다.

이때 split함수를 이용하면 끝에 doc의 label값이 string 형태로 들어가기 때문에 그 부분은 float형태로 바꿔서 다시 temp에 넣어주고
한 개의 리뷰와 label 값이 들어 있는 temp를 corpus list에 추가해준다.
모든 리뷰를 corpus에 추가할 때까지 이 과정을 반복하고, 끝나면 corpus를 리턴해준다.

```
def count_words(self, training_set):

    # 학습데이터는 영화리뷰 doccumnet , 긍정 부정 값(1, 0)으로 구성
    counts = defaultdict(lambda : [0, 0])
    cnt = 0
    for doc, point in training_set:

        # konlpy로 tokenize
        words = tokenize(doc)
        # 학습 데이터를 10000개의 배수로 읽을 때마다 몇개 읽었는지 출력
        cnt +=1
        if cnt%10000 == 0 :
            print(cnt)
        for word in words:
            counts[word][0 if point == 1 else 1] += 1
    return counts
```

리뷰 하나 하나를 tokenize해서 그 token들의 갯수를 label값이 1 인것과, 0인 것 따로 세어주는 함수이다.

```
def word_probabilities(self, counts, total_pos, total_neg, k):
    # 단어의 빈도수를 [단어, p(w|긍정), p(w|부정)] 형태로 반환
    return [(w,
             (pos + k) / (total_pos + 2*k),
             (neg + k) / (total_neg + 2*k))
            for w, (pos, neg) in counts.items()]
```

각 단어들이 나타나는 빈도 수 를 긍정 리스트에서 한번, 부정 리스트에서 한번 확률로 구한다.
그 값을[단어, p(w|긍정), p(w|부정)] 이런 형태의 데이터를 반환해준다.

확률은 왼쪽과 같은 식을 이용해서 구해준다.

$$P(w_i|positive) = \frac{k + count(w_i, positive)}{2k + \sum_{w \in V} count(w, positive)}$$

```
def train(self, trainfile_path):
    training_set = self.load_corpus(trainfile_path)

    # 범주0(긍정)과 범주1(부정) 문서 수를 세어 줌
    num_pos = len([1 for _, point in training_set if point == 1])
    num_neg = len(training_set) - num_pos

    # training!!!!
    word_counts = self.count_words(training_set)
    self.word_probs = self.word_probabilities(word_counts,
                                              num_pos,
                                              num_neg,
                                              self.k)
```

NaiveBayesClassifier클래서의 train함수는 위의 함수들을 이용해서 trainging data들을 학습 시켜준다.

여기까지가 label이 있는 값들을 train시키는데 필요한 함수들 이었다.
이제 classify를 위한 함수들을 보자.

```
def compare_probability(self, word_probs, doc):
    #konlpy 이용해서 tokenize
    docwords = tokenize(doc)

    # 모두 0으로 초기화 해준다.
    log_prob_if_pos = log_prob_if_neg = 0.0

    # 모든 단어에 대해 반복해준다.
    for word, prob_if_pos, prob_if_neg in word_probs:
        # 만약 리뷰에 word가 있다면
        # 해당 단어가 나올 log 확률을 더해 줌
        if word in docwords:
            #print(word)
            log_prob_if_pos += math.log(prob_if_pos)
            log_prob_if_neg += math.log(prob_if_neg)

        # 만약 리뷰에 word가 없다면
        # 해당 단어가 없을 log 확률을 더해 줌
        else:
            log_prob_if_pos += math.log(1.0 - prob_if_pos)
            log_prob_if_neg += math.log(1.0 - prob_if_neg)

    prob_if_pos = math.exp(log_prob_if_pos)
    prob_if_neg = math.exp(log_prob_if_neg)
    # 이렇게 긍정 확률과 부정 확률을 구한 후 둘 중 더 큰 확률을 채택해서,
    #내가 읽은 review가 긍정인지 부정인지 결정을 해준다.
    return 1 if (prob_if_pos / (prob_if_pos + prob_if_neg)) >
    (prob_if_neg / (prob_if_pos + prob_if_neg)) else 0
```

내가 분류할 doc을 먼저 tokenize한다.
그리고
doc에 있는 모든 단어가
긍정 리스트와 부정 리스트에 있다면 그 확률을 log취해서 더해준다.

없으면 없을 확률을 log취해서 더해준다.

그리고 그 값에 exp를 취해준다.

이렇게 하는 이유는 컴퓨터가 0에 가까운 floating point number를 제대로 처리하지 못하기 때문이다.
그래서 우도의 곱들을 log우도의 합으로 처리해

준다.

이렇게 해당 doc에 대한 긍정 리스트에 대한 확률과 부정 리스트에 대한 확률을 구한 후, 둘 중 확률이 더 큰 것을 채택해서 리턴해준다.

그래서 긍정 확률이 더 크면 1을 리턴해서 classify 함수에게 전해주고, 부정 확률이 더 크면 0을 리턴해준다.

```
def classify(self, doc):
    return self.compare_probability(self.word_probs, doc)
```

classify함수는 전해 받은 doc의 결정된 label값을 그대로 리턴만 해준다.

```
def resWrite(path):
    model = NaiveBayesClassifier()

    model.train(trainfile_path='ratings_train.txt')

    fi = open(path, "r")
    fo = open("ratings_result.txt", "w")

    first = fi.readline()
    fo.write(first)

    #true postive, true negative, false postive, false negative 계산
    # tp = 0
    # tn = 0
    # fp = 0
    # fn = 0

    while True :
        line = fi.readline()
        if not line : break;
        temp = line.split('\t')
        #valid data classify후 결과 비교 하기 위한 부분
        # label = int(temp.pop(2))
        res = model.classify(temp[1])

        #valid data classify후 결과 비교 하기 위한 부분
        # if(res == 1 and label == 1) :
        #   tp += 1.0
        # elif(res == 0 and label == 1) :
        #   fn += 1.0
        # elif(res == 1 and label == 0) :
        #   fp += 1.0
        # elif(res == 0 and label == 0):
        #   tn += 1.0
        fo.write(temp[0] + '\t' + temp[1] + '\t' + str(res) + '\n')


    # classify한 결과 출력
    # print("precision : ", tp/(tp+fp))
    # print("recall : ", tp/(tp+fn))
    # print("false : ", int(fn+fp))

    fi.close()
    fo.close()
```

resWrite함수는 내가 전달 받은 path의 데이터를 classify 해서 result파일에 결과를 써주는 함수다.

path의 파일을 한 줄 씩 읽어와서,
doc 부분만 추출해서 classify 함수를 통해 label 값을 알아낸후,
result파일에 리뷰의 id, doc, label값을 차례로 써준다.

주석 처리 되어 있는 부분은,
ratings_valid.txt 파일을 classify하고 그 결과 값에 대한
precision, recall, false 값을 구하기 위해 써둔 것이다.

2. 실험 결과

우리는 classify하면서 문서의 확률을 예측할때, 확률이 0이 되지 않게 하기 위해 smoothing이란 것을 한다. 그래서 임의로 k라는 값을 더해준다.
이 k 값을 1과 0.5 두가지 값으로 정한 뒤 같은 조건에서 ratings_valid.txt의 데이터의 분류 결과를 구해 보았다.

```
precision :  0.82653258557339
recall :  0.8510553564317005
false :  1645
```

```
precision :  0.8260027126525867
recall :  0.8488649940262843
false :  1657
```

왼쪽이 k값을 1로 했을 때의 결과이고 오른쪽이 k값을 0으로 했을 때의 결과이다.

precision 값은 true positive / (true positive + false positive) 이므로 내가 true라고 결정한 것 중 정말 true인 퍼센트를 말하는 것이다.

recall 값은 true positive / (true positive + false negative) 이므로 실제 true 중 내가 얼마나 많이 true라고 판단을 했는지에 대한 퍼센트이다.

false 값은 false negative 와 false postive를 더한 값으로 내가 분류에 실패한 데이터의 수이다.

그러므로 precision값과, recall값이 높을 수록 좋고 false값이 낮을 수록 좋다.

또 정확한 분류를 위해 konlpy 모듈의 tokenizer를 사용하는데, 내가 token을 normalize하거나 stem할것인지 옵션을 정할 수 있다.

두 가지 옵션을 줘서 나오는 모든 4가지 경우로 다 ratings_valid.txt를 분류해보았다.

norm = True, stem = True	norm = True,
precision : 0.82653258557339 recall : 0.8510553564317005 false : 1645	precision : 0.8360054085377632 recall : 0.8618080446037435 false : 1543
stem = True	no option
precision : 0.8291925465838509 recall : 0.8506571087216248 false : 1630	precision : 0.8370743034055728 recall : 0.8614097968936678 false : 1538

token들을 normalize 해주거나, 아무 옵션도 안 주는 경우가 나머지 두 경우에 비해 정확도가 높아지는 것으로 보인다.
정확도가 높아진 두 가지 경우의 precision과 recall은 근소한 차이가 나므로 false가 더 적은 no option을 선택하기로 했다.
위 처럼 k값을 1로 정하고 tokenize option은 아무것도 주지 않은 채로 ratings_test.txt를 classify해 보았다.
결과는 다음 처럼 나왔다.