영화 리뷰 긍정/부정 분류하기

2013010926 강은석

1. 개요

Naïve Bayes Classificaion 기법을 이용하여 네이버 영화 리뷰 데이터셋 긍정/부정 분류하기

- 네이버 영화 리뷰 데이터 셋의 raw sentence를 가공하는 방법의 선택은 자유
- 분류기의 성능을 높이기 위한 최적화 기법 적용가능
- 긍정 / 부정 분류 정확도로 점수 산출

2. 형태소 분석기 사전 조사 및 결정

Konlpy의 tag package를 사용하기로 결정하고 tag Package인 "Hannanum", "Kkma", "Komoran", "Mecab", "Twitter"의 후보군 중 sample data를 이용해서 pos() method를 써본 결과 형태소 tokenize를 가장 큰 대분류로(ex Josa, Noun, Verb ...) 해주고 속도가 빠른 "Twitter"를 사용하기로 결정하였습니다.

3. 형태소 분석 및 전처리

Konlpy의 Twitter에도 morphs(), nouns(), phrases(), pos()등 다양한 메소드가 있고 그 중 형태소 분류를 해주는 pos()를 사용하기로 했습니다. Pos() method에서도 옵션으로 normalization과 stemming을 지원했기 때문에 각각에 대해 ratings_valid.txt을 기준으로 성능을 비교하는 작업을 진행하였습니다.

- norm=True, stem=False

```
preprocessing end
train tokenize end
use pos normalization result
accu : 0.8447409410174274

Process finished with exit code 0
```

- norm=False, stem = True

```
preprocessing end
train tokenize end
use pos normalization result
accu : 0.8185892668415078

Process finished with exit code 0
```

-norm=True, stem=True

```
preprocessing end
train tokenize end
use pos normalization result
accu : 0.8201330316121511
```

- norm=False, stem=False

```
preprocessing end
train tokenize end
use pos normalization result
accu : 0.86067

Process finished with exit code 0
```

이에 normalize와 stemming은 사용하지 않기로 결정을 하고 pos를 사용할 때 norm과 Stem flag를 False로 지정해주었습니다.

그 뒤 평가를 할 때 확률 테이블에 없는 token이 들어오면 무시할지 아니면 작은 값을 넣어서 처리를 할지에 대한 비교 작업을 진행했습니다.

```
for token in review:
    pos_exist = pos_dic.get(token)
    neg_exist = neg_dic.get(token)
    if pos_exist == None:
        predict_pos += math.log2(0.0000079)
    else:
        predict_pos += math.log2(pos_dic.get(token))
    if neg_exist == None:
        predict_neg += math.log2(0.0000079)
    else:
        predict_neg += math.log2(neg_dic.get(token))
```

```
preprocessing end
train tokenize end
use pos normalization result
accu : 0.86067

Process finished with exit code 0
```

ratings_valid.txt에서 review의 token이 pos_dic, neg_dic에 없는 경우 작은 값을 넣어 naïve bayes classifier을 계산한때 임의의 작은 값을 log취해서 더하는 방식을 진행한 결과 86%를 얻을 수 있었습니다.

```
for token in review:
    pos_exist = pos_dic.get(token)
    neg_exist = neg_dic.get(token)
    #if pos_exist == None:
        #predict_pos += math.log2(0.0000079)
    if pos_exist != None:
        predict_pos += math.log2(pos_dic.get(token))
    #if neg_exist == None:
        #predict_neg += math.log2(0.0000079)
    if neg_exist != None:
        predict_neg += math.log2(neg_dic.get(token))
```

```
preprocessing end
train tokenize end
use pos normalization result
accu : 0.70428

Process finished with exit code 0
```

이 후 pos_dic, neg_dic 에 없으면 무시하는 방식으로 진행한 결과 70%를 얻을 수 있었습니다. 따라서 token이 없는 경우 임의의 작은 값을 넣는 방식으로 진행했습니다.

4. 코드 설명

데이터 처리 모듈인 'nltk'와 'konlpy'가 설치 되어있는 상태여야 합니다.

코드의 진행은 아래의 그림 과 같이 진행됩니다.

```
if __name__ == '__main__':
    twitter = Twitter()
    no_data = []
    train_docs, train_labels = read_dataset("ratings_train.txt")
    test_ids, test_docs = read_test_dataset("ratings_test.txt")

    train_reviews = preprocessing_review(train_docs, train_labels, False)
    test_reviews = preprocessing_review(test_docs, no_data, True)
    print("preprocessing end")
    positive_tokens, negative_tokens = tokenize_review(train_reviews)
    re_pos = nltk.Text(positive_tokens, name="pos_train_data").vocab().most_common()
    re_neg = nltk.Text(negative_tokens, name="neg_train_data").vocab().most_common()
    print("train tokenize end")

    prob_pos, prob_neg, pos_dic, neg_dic = extract_prob(re_pos, re_neg, train_labels)
    t_result = naive_bayes_classifier(prob_pos, prob_neg, pos_dic, neg_dic, test_reviews)

    with open("result1.txt", "w", encoding="UTF-8") as f:
        f.write("id"+"₩t"+"document"+"₩t"+"label")
        f.write("₩n")
        for i in range(len(t_result)):
            f.write(test_ids[i]+"₩t"+test_docs[i+1]+"₩t"+t_result[i])
            f.write("₩n")
```

그림 1 code flow

우선 "ratings_train.txt" 와 "ratings_test.txt"을 읽어와서 id, document, label column을 읽어서 train data에서는 id를 제외한 document, label column을 읽고 test data에서는 label을 제외한 id, document column을 읽어와서 각각 저장했습니다. 이후 doc에 대해서 preprocessing_review 함수를 호출해서 형태소 분석기를 이용해 필요 없는 형태소를 제거한 뒤, tokenize_review함수를 이용해서 긍정, 부정에 해당하는 문장에 있는 단어들을 추출해서 positive_tokens과 negative_tokens에 저장을 했습니다. 그 뒤, nltk를 이용해서 단어들의 빈도수를 계산하고 extract_prob 함수를 이용해서 naïve bayes classifier에 쓸 확률 테이블을 만들고 naïve_bayes_classifier를 이용해서 리뷰에 대한 긍정/부정 여부를 판별하고 결과를 저장했습니다.

```
def preprocessing_review(reviews, labels, is_test):  # get train data and preprocess reviews
    preprocessed_review = []
    for i in range(1, len(reviews)):
        tokenized_review = twitter.pos(reviews[i])
        tokens = []
        for token in tokenized_review:       # get rid of useless morpheme
            if token[1] == "Josa" or token[1] == "Punctuation":
                continue
            if token[1] == "Eomi" or token[1] == "Determiner":
                continue
            if token[0] == "영화":
                continue
            tokens.append(token[0])
        if is_test == True:
            preprocessed_review.append(tokens)
        else:
            preprocessed_review.append((tokens, labels[i]))

    return preprocessed_review
```

그림 2 preprocess using twitter.pos()

Twitter의 pos() method를 이용해 형태소 분석을 통해서 긍정/부정에 영향을 적게 준다고 판단한 조사(Josa), 기호(Puncation), 어미(Emoi)와 한정사(Determiner)를 제외를 시켜주었습니다.

```
def tokenize_review(reviews):  # collect token by pos, collect token by neg
    pos_tokens = []
    neg_tokens = []
    for i in range(len(reviews)):
        if reviews[i][1] == "1":
            pos_tokens.extend(reviews[i][0])
        elif reviews[i][1] == "0":
            neg_tokens.extend(reviews[i][0])

    return pos_tokens, neg_tokens
```

그림 3 tokenize 분류 함수

위에서 여러 필요 없는 형태소를 제거해서 전처리된 review들에 대해서 나이브 베이지안 분류기의 확률표를 만들어서 사용하기 위해 긍정은 pos_tokens에 부정은 neg_tokens에 나눠서 저장했습니다.

```python
def extract_prob(pos_tokens, neg_tokens, labels):  # make probability table
    prob_pos = 0
    prob_neg = 0
    pos_dic = {}
    neg_dic = {}
    for label in labels:
        if label == "1":
            prob_pos += 1
        else:
            prob_neg += 1
    for token in pos_tokens:
        pos_dic[token[0]] = token[1]/len(pos_tokens)
    for token in neg_tokens:
        neg_dic[token[0]] = token[1]/len(neg_tokens)

    return prob_pos/len(labels), prob_neg/len(labels), pos_dic, neg_dic
```

그림 4 확률 테이블 추출

위에서 만든 pos_tokens 배열과 neg_tokens 배열을 가져와서 전체 리뷰 데이터 중 긍정 리뷰의 확률, 부정 리뷰의 확률, 각 token에 대해서 긍정일 때 부정일 때의 확률들을 계산해서 긍정은 pos_dic에 부정은 neg_dic에 저장했습니다. 이후 naïve bayes classifier에서 사용하기 쉽게 하기 위해 딕셔러니 형태로 만들었습니다.요

```python
def naive_bayes_classifier(prob_pos, prob_neg, pos_dic, neg_dic, val_reviews):
    results = []
    for i in range(len(val_reviews)):
        review = val_reviews[i]
        predict_pos = 0
        predict_neg = 0
        for token in review:
            pos_exist = pos_dic.get(token)
            neg_exist = neg_dic.get(token)
            if pos_exist == None:
                predict_pos += math.log2(0.0000078)
            else:
                predict_pos += math.log2(pos_dic.get(token))
            if neg_exist == None:
                predict_neg += math.log2(0.0000078)
            else:
                predict_neg += math.log2(neg_dic.get(token))

        predict_neg += math.log2(prob_neg)
        predict_pos += math.log2(prob_pos)
        if predict_pos > predict_neg:
            results.append("1")
        if predict_pos <= predict_neg:
            results.append("0")

    return results
```

그림 5 분류기

분류기에서는 math함수의 log2를 이용해서 확률을 변환시켜 더해주는 방식으로 계산을 하였습니다. 이어 token이 train data로 만들어진 dic에 없는 경우 임의의 작은 값을 추가해주어 계산해주었습니다.