정보처리 및 자연어 처리 : 기말 과제

전공: 컴퓨터공학 학년: 3학년 학번: 20201635 이름: 전찬

0. 목차

1. 구현 목표

이번 자연어 처리 기말 과제는 수업 시간에 배운 NLTK module, 그 중에서도 module 내에서 지도 기계 학습을 수행할 수 있는 NaiveBayesClassifier 을 활용하여 제공된 train_set, test_set을 바탕으로 feature extraction을 수행하며, 최대한 높은 정확도의 classifier을 만들어내는 형태이다. 따라서 전반적인 기계학습 절차를 이해하며, 제시된 data에서 분류에 영향을 미칠 수 있는 feature가 무엇인지 파악하며, 적절한 형태로 추출해내는 코드를 작성해야 한다.

2. 구현 절차 및 feature extraction

기계 학습은 크게 두 가지 관점이 존재한다. 하나의 관점은 컴퓨터공학에서 데이터를 통해 지도 학습을 바탕으로 분류, 예측을 수행하는 하나의 모델을 만들어내는 형태이다. 또 다른 관점은 데이터 마이닝 관점으로, 기존의 데이터를 통해 새로운 지식을 얻어내는 형태이다. 이번 기말 과제는 이 중 컴퓨터공학 관점으로, 데이터를 통해 주어진 test set의 각 어절의 개체명을 예측하는 형태이다. 따라서 아래와 같은 지도 기계학습 절차를 거쳐야 한다.

1. 데이터에 대해 원하는 형태로 전처리를 수행한다.
2. 데이터에 대해 적절한 feature extraction 방법을 구현한다.
3. 데이터를 train / test data 로 나눈다.
4. Training 과정을 수행한다. 각 train data에 feature extraction을 수행하며, 학습을 통한 classifier을 구현한다.
5. Prediction 과정을 수행한다. 각 test data에 feature extraction을 수행하여, 데이터를 예측해보며, 정확도 등을 측정한다.
6. 예측 값을 바탕으로 classifier을 개선한다.

따라서 위 절차에 따라서 구현한 과정은 아래와 같다.

- 데이터 전처리 과정

데이터 전처리 과정의 코드는 아래와 같다.

```
import nltk
import re
import Hangul

# data는 동일한 directory에 위치해 있다.

# file에서 data를 받아온다.
with open("train.txt", "r") as f1:
    train_data = f1.readlines()

with open("test.txt", "r") as f2:
    test_data = f2.readlines()

# 받은 data를 \t, \n (whitespace) 을 기준으로 나눈다.
for i in range(len(train_data)):
    train_data[i] = train_data[i].split()

for i in range(len(test_data)):
    test_data[i] = test_data[i].split()
```

<데이터 받아오기 및 전처리 과정>

동일한 directory에 존재하는 'train.txt', 'test.txt' 을 with open() 을 통해 각 line 단위로 읽어오며, 각각 train_data, test_data 변수에 할당했다. 또한 데이터 파일이 ₩t 로 구분되어 있기 때문에, 이를 나누기 위해서 .split() method을 통해서 데이터를 나누는 전처리를 수행해 주었다.

- feature extraction 방법

지도 기계 학습, 그 중에서 Naive Bayes와 같은 방법을 사용할 때에 가장 중요한 것은 feature extraction을 가장 높은 정확도를 가지며, 효율적으로 구현하는 것이다. 여기에서 효율적이라는 것은, 경제성의 원리에 따라, feature이 적어도 비슷한 결과가 나온다면 feature이 적은 것이 효율적인 classifier 이라고 할 수 있다. 따라서 이러한 원리를 바탕으로 구현한 feature extraction 함수는 아래와 같다.

```python
def feature_extraction(dataset, loc):
    features = {}

    #시간(DAT, TIM)을 위한 data
    for letter in "년월일시분초전후":
        features[f'has({letter})'] = letter in dataset[loc][1]

    features['word'] = dataset[loc][1]

    features['first_one'] = dataset[loc][1][0]
    features['first_two'] = dataset[loc][1][:2] if len(dataset[loc][1]) > 1 else dataset[loc][1]
    features['first_three'] = dataset[loc][1][:3] if len(dataset[loc][1]) > 2 else dataset[loc][1]

    features['last_one'] = dataset[loc][1][-1]
    features['last_two'] = dataset[loc][1][-2:]
    features['last_three'] = dataset[loc][1][-3:]

    features['delete_last_one'] = dataset[loc][1][:-1]
    features['delete_last_two'] = dataset[loc][1][:-2]
    features['delete_last_three'] = dataset[loc][1][:-3]

    features['before_word'] = '' if dataset[loc][0] == '1' else dataset[loc-1][1]
    features['after_word'] = '' if loc == len(dataset)-1 or dataset[loc+1][0] == '1' else dataset[loc+1][1]

    features['has_digit'] = any(letter.isdigit() for letter in dataset[loc][1])
    features['has_eng'] = True if re.search('[a-zA-Z]', dataset[loc][1]) else False

    return features
```

<feature extraction 함수>

각 feature을 설명하면, 아래와 같다.

features['has(~)'] 는 DAT, TIM 을 파악하기 위한 feature로, 년, 월, 일, 시 등의 날짜, 시간과 관련된 어휘를 포함하는지를 파악할 수 있는 feature 이다.

features['word'] 는 어절 자체를 의미하는 feature 이다.

features['first_one'], features['first_two'], features['first_three'] 는 각각 어절의 앞의 한 문자, 두 문자, 세 문자를 의미한다. 이때 어절의 길이가 2, 3 보다 짧다면, 각각 어절 자체를 대입하는 형식으로 발생할 수 있는 에러를 피했다.

features['last_one'], features['last_two'], features['last_three'] 는 각각 어절의 뒤의 한 문자, 두 문자, 세 문자를 의미한다. 이를 통해서 은는이가, 에게, 에게서 등의 조사를 파악할 수 있다.

features['delete_last_one'], features['delete_last_two'], features['delete_last_three'] 는 어절의 마지막 한 문자, 두 문자, 세 문자를 제외한 나머지 문자를 의미한다. 이를 통해 '한국은', '한국에서', '한국에게서' 등의 어절에 대해, 모두 '한국' 으로 판단할 수 있는 정보를 제공할 수 있다.

features['before_word'], features['after_word'] 는 각각 어절의 앞 어절, 뒤 어절을 의미한다. 또한 어절이 문장의 맨 처음, 끝인 경우, ''(아무것도 포함하지 않는 문자열) 을 대입한다. 이를 통해 "한 단어는 문맥에 의해 의미가 결정된다." 라는 분포 가설을 바탕으로 문맥을 고려할 수 있다.

features['has_digit'] 과 features['has_eng'] 는 각각 해당 어절이 숫자, 혹은 문자를 포함하는지를 파악할 수 있는 feature 이다.

추가로 구현했지만, 정확도에 큰 차이가 존재하지 않아 제거한 feature도 존재하는데, cvl, org, trm 등 고유명사가 많이 존재하는 경우에는, 빈도수가 높은 50개의 단어를 추출하여 dataset으로 구현하며, feature extraction을 수행하는 어절이 해당 단어들을 포함하고 있는지를 파악할 수 있는 feature을 구현했었다. 한 예시인 'CVL' 에 대한 고빈도 list와 feature 추출은 아래와 같다.

```
# 빈도수를 바탕으로 cvl dataset 얻어내기
cvl_data_set = []
for data in train_data:
    if data[2] == 'CVL':
        if re.search("\w+", data[1]):
            word = re.search("\w+", data[1]).group()
            if word[-1] in "은는이가을던의를에도들":
                cvl_data_set.append(word[:-1])
            else:
                cvl_data_set.append(word)

CVL_freqdist = nltk.FreqDist(cvl_data_set)

cvl_list = [i for (i, _) in CVL_freqdist.most_common(50)]

# feature_extraction 함수 내부에서 아래와 같이 feature로 추가
features['in_cvl_list'] = False
for i in cvl_list:
    if i in dataset[loc][1]:
        features['in_cvl_list'] = True
```

<고빈도 cvl_data_set 구현 및 feature 추출>

이때 cvl_data_set은 '은는이가', '은는을던' 등의 조사를 제거한 형태이며, 따라서 feature extraction을 수행하는 어절 안에 이러한 단어가 존재하는지를 파악하기 위해 if i in data[1] 와 같이 코드를 작성했다. 또한 이와 동일하게 org, trm, loc 등의 고빈도 list 또한 구현했으나, 정확도에 큰 차이가 없어서, feature에서 제거해 주었다.

또한 features['length'] = len(dataset[i][1]) 와 같이 해당 어절의 길이를 의미하는 feature 또한 예측 값에 유의미한 차이가 존재하지 않아, feature에서 제거해 주었다.

- Training 과정과 Prediction 과정

training, prediction 과정은 아래와 같다. feature extraction을 수행한 train_set, test_set을 만들어 내며, 이를 바탕으로 train, predict 를 수행했다. 이는 아래와 같다.

```
# train, test_data의 모든 data에 feature_extraction을 적용하여 train, test_set 구현
train_set = [(feature_extraction(train_data, i), train_data[i][2])
             for i in range(len(train_data))]
test_set = [(feature_extraction(test_data, i), test_data[i][2])
             for i in range(len(test_data))]

# classifier을 만들고, test_set에 대해 accuracy를 측정한다.
classifier = nltk.NaiveBayesClassifier.train(train_set)
nltk.classify.accuracy(classifier, test_set)
```

<train과 test 과정>

또한 이를 통해 도출한 정확도는 아래와 같다.

0.8702424829349932

<nltk.classify.accuracy(classifier, test_set) 의 결과값>

- classifier 개선

classifier을 개선하기 위해서 첫 번째로 다음과 같이 분류 예측에서 잘못된 예측을 많이 수행하는 단어가 무엇인지 파악했다. 이 코드는 아래와 같다.

```
# error 가 자주 발생하는 어휘 판별
errors = []
for i in range(len(test_data)):
    guess = classifier.classify(feature_extraction(test_data, i))
    if guess != test_data[i][2]:
        errors.append((guess, test_data[i][2], test_data[i][1]))
```

```
errors
freq = nltk.FreqDist([data for data in errors])
freq
# len(errors)
```

```
FreqDist({('NUM', '-', '한'): 531, ('DAT', '-', '지난'): 130, ('DAT', '-', '전'): 118, ('NUM'
, 'DAT', '한'): 109, ('DAT', '-', '호시기'): 76, ('DAT', '-', '계절'): 70, ('DAT', '-', '유행기'
): 70, ('DAT', '-', '몇'): 69, ('DAT', '-', '월경기'): 68, ('DAT', '-', '올'): 65, ...})
```

<prediction error가 빈번한 어절 파악 과정>

따라서 이를 바탕으로 prediction error가 빈번한 어절을 파악할 수 있으며, 이를 통해서 개선 방법을 생각해볼 수도 있었다.

또한 두 번째로 test_data의 각 tag(개체명)의 빈도를 파악하고, 각 tag에서의 prediction 결과가 어떻게 되는지를 파악해서 classifier을 개선시킬 수 있었다. 각 개체명의 빈도를 파악하는 코드는 아래와 같다.

```
freqdist = nltk.FreqDist(data[2] for data in test_data)
```

```
freqdist.most_common()
```

```
[('-', 292164),
 ('NUM', 25966),
 ('CVL', 23972),
 ('PER', 19280),
 ('ORG', 18310),
 ('DAT', 13589),
 ('TRM', 8885),
 ('LOC', 8370),
 ('EVT', 7029),
 ('ANM', 2609),
 ('AFW', 2451),
 ('TIM', 1647),
 ('FLD', 949),
 ('MAT', 124),
 ('PLT', 87)]
```

<test set의 각 개체명의 빈도수 파악>

또한 아래와 같은 코드로 각 개체명에서 예측을 얼마나 잘 수행하는지를 파악할 수 있다.

```
for tag, _ in freqdist.most_common():
    data_div_set = []
    for data in test_set:
        if data[1] == tag:
            data_div_set.append(data)

    predict = nltk.classify.accuracy(classifier, data_div_set)
    print(tag, round(predict, 3))
```

```
- 0.878
NUM 0.916
CVL 0.881
PER 0.898
ORG 0.848
DAT 0.92
TRM 0.735
LOC 0.814
EVT 0.752
ANM 0.751
AFW 0.432
TIM 0.862
FLD 0.547
MAT 0.073
PLT 0.138
```

<각 개체명의 prediction 값>

이를 바탕으로 classifier을 개선할 때, 빈도가 많으며, prediction 값이 낮은 것들을 위주로 개선을 수행했으며, 개선과 prediction 과정을 반복하여 위와 같이 87%의 예측 정확도를 구현해낼 수 있었다.