2020-2R 기계학습 Term Project

2014210040 이수호

1. 코드에 대한 설명

코드는 크게 다음 세 파일로 분리되어 있습니다.

1. main.py
2. classifier.py
3. vectorizer.py

1-1. main.py

main.py에서는 크게 다음과 같은 역할을 수행합니다.

1. csv 파일을 읽어서 pandas.DataFrame으로 변경 (`load_data()` 함수가 수행)
2. DataFrame 내의 data(문장)를 전처리 (`preprocess()` 함수가 수행)
3. 모델 학습 (`train()` 함수가 수행)
4. 모델 테스트 (`test()` 함수가 수행)
5. 결과값을 `out.csv`에 저장

1-2. classifer와 vectorizer

classifier는 어떤 feature vector에 대한 classification 작업을 하는 모델입니다(과제에서는 *ham*과 *spam*을 구별). classifier의 input은 보통 feature vector이기 때문에, 자연어 처리 모델을 만들 때 보통 vectorizer를 함께 사용합니다. vectorizer는 자연어로 이루어진 문장에서 feature extraction을 하여 vector 포맷으로 바꿔줍니다. classifier와 vectorizer는 모델의 중요한 구성 요소이기 때문에, 어떤 classifier와 vectorizer를 사용하고 있는지를 쉽게 확인할 수 있도록 두 모델을 리턴하는 함수를 별도의 파이썬 파일로 분리하였습니다.

classifier와 vectorizer 파일에는 각각 `get_classifier()`와 `get_vectorizer()` 함수가 정의되어 있습니다. 각각 함수에서는 main.py에서 사용하는 classifier와 vectorizer를 리턴하여 줍니다.

classifier에는 모델 트레이닝을 위한 `classifier#fit()` 함수와 input에 대해 output을 예측하는 `classifier#predict()` 함수가 정의되어 있어야 합니다.

vectorizer에는 모델 트레이닝을 위한 `vectorizer#fit()` 함수와 input string에 대해 output vector를 반환하는 `vectorizer#transform()` 함수가 정의되어 있어야 합니다.

scikit-learn에서 제공하는 classifier와 vectorizer 구현체에는 공통적으로 해당 함수가 정의되어 있어, 해당 구현체를 사용한다면 추가적으로 코드를 수정할 부분은 없습니다. 다만, classifier 또는 vectorizer를 직접 구현하여 사용할 경우에, 각 함수에서 리턴하는 인스턴스에는 해당 메서드가 구현이 되어있어야 합니다.

2. 문제에 대한 접근법

2-1. Bag of Words (BOW)

어떤 문장이 스팸인지 아닌지를 분별하는 것은 classification 문제에 속합니다. classifier를 사용하기 위해서는, 자연어로 이루어진 문장을 feature vector로 변환하는 작업이 필요합니다. 문장을 feature vector로 변환하는 모델 중, 단어의 빈도수를 고려하는 Bag of Words (BOW) 모델을 먼저 사용해 보았습니다. scikit-learn의 CountVectorizer 클래스는 단어의 빈도수를 세어 BOW 모델을 만드는 모델을 구현하여, 먼저 이를 사용해 보았습니다.

2-2. TF-IDF

TF-IDF(Term Frequency - Inverse Document Frequency)는 단순히 단어의 occurance만을 고려하는 CountVectorizer 모델과 달리, 전체 문서(데이터셋)에서 자주 쓰이는 단어들에 대해 penalty를 주는 모델입니다. 즉, '자주 등장하는 단어'에 대해서는 CountVectorizer와 같이 가중치를 높이지만, '전체 문서에서 자주 등장하는 단어'는 classification에 큰 의미를 주지 못 한다고 보아 가중치를 줄입니다. scikit-learn에서는 TfIdfVectorizer 클래스를 통해 이 모델을 구현하였습니다. TfIdfVectorizer를 사용했을 때 조금 더 높은 accuracy를 얻을 수 있었는데, 이로서 전체적으로 자주 등장하는 단어는 classification에 큰 기여를 하지 않는다는것을 알 수 있었습니다. 또한, 전체적으로 너무 자주 등장하는 단어는 classification에 의미가 없다고 보아 vocabulary에서 제외할 수도 있습니다. TfIdfVectorizer에서는 max_df argument를 통해 특정 비율 이상 등장하는 단어를 제외할 수 있습니다. 이를 통해 Vectorizer의 모델을 조금 더 간단하게 할 수 있었습니다.

2-3. Word Embedding vs Character Embedding

2-3에서 사용한 TF-IDF Vectorizer는 단어 단위로 feature vectorizer를 생성합니다. 단어 단위 이외에도 character 단위로 feature vector를 생성할 수 있는데, 이를 Character Embedding이라고 합니다. 또한, 여러 단어 / character에 대해서도 feature vector를 *따로* 생성하는 n-gram을 적용해볼 수도 있었는데, 이를 정리하자면, Vectorizer에대해서 다음과 같은 4가지 조합을 실험해볼 수 있었습니다.

1. Word Embedding
2. Word Embedding + n-gram
3. Character Embedding
4. Character Embedding + n-gram

결론적으로, n-gram을 적용한 Character Embedding의 성능이 제일 좋았습니다. 이에 대한 이유를 대략적으로 파악해 보았습니다.

1. Spam classifier의 경우, 단어 이외의 occurance를 통해 스팸 여부를 파악하는 빈도가 많았습니다. 대표적인 예가 전화번호입니다. 전화번호와 같은 경우 classifier의 입장에서는 서로 비슷한 의미를 가질텐데 번호값이 하나만 달라지더라도 OOV(Out of Vocabulary)문제가 발생합니다. 따라서, word embedding 대신 character embedding을 사용하여 variance를 줄이는게 가능하다는 생각이 들었습니다.
2. train set에 알파벳 / 영어단어가 아닌 숫자(전화번호 등), 특수기호, 오타 등의 case가 많았습니다. character embedding이 OOV Word / 오타에 대해 더 핸들링을 잘 하기 때문에, 더 좋은 결과값이 나왔던 것입니다.
3. character embedding에 n-gram을 적용하였기에 word 단위의 데이터도 어느정도 고려할 수 있었습니다.

scikit-learn에서는 character embedding을 두 가지 다른 방식으로 구현하고 있는데(`analyzer` argument로 지정), 첫 번째는 문장 전체 단위를 스캔하는 `char` 이고, 두 번째는 문장 속 단어 내에서만 n-gram을 만드는 `char_wb` 가 있었습니다. `char_wb` 의 성능이 더 좋았는데, 서로 다른 맥락의 단어가 같은 vector 내로 섞이지 않아 noise가 줄어들기 때문이라고 판단하였습니다.

2-4. Support Vector Machine & SGD Learning

Classifier로는 Support Vector Machine을 사용하였습니다. 데이터의 특성을 보았을 때 linear하게 decision boundary가 정해져도 classification할 수 있다고 생각하였기 때문입니다. scikit-learn에서는 SVC, LinearSVC 클래스로 SVM을 구현하였습니다. SVM이 사용하는 loss function으로는 기본값인 hinge loss를 사용하였을 때 결과가 가장 좋았습니다. decision boundary의 smoothness를 조절하여 약간의 오차를 허용함으로써 overfitting을 줄이는 soft margin SVM 또한 사용할 수 있었는데, 이는 LinearSVC 클래스의 C parameter로 조절할 수 있었습니다. SVM을 Stochastic Gradient Descent를 사용하여 학습시키는 방법이 있어 함께 적용해 보았습니다. scikit-learn에서는 SGDClassifier 클래스에 구현되어있고, 이를 사용함으로서 별도의 test dataset이 주어졌을 때 모델이 더 잘 대응될 수 있다는 생각이 들어 LinearSVC 대신 적용하였습니다.

3. 코드 실행 및 환경 설정법

3-1. 환경 설정하기

- Python 런타임으로는 Anaconda 2020-07(Python 3.8)을 사용하였습니다. OS에 따른 종속성은 없습니다.
- 혹시 몰라

```
pip freeze > requirements.txt
```

로 현재 사용중인 디펜던시를 pinning 해놓았습니다. 이 경우

```
pip install -r requirements.txt
```

를 실행하여 디펜던시를 설치하면 됩니다.

3-2. 코드 실행하기

1. data 폴더 내에 train data와 test data가 있어야 합니다.

- train data와 test data의 csv 형식은 Kaggle competition에서 주어진 csv 파일의 형식과 같습니다.
- train data의 이름은 `train.csv`고, test data의 이름은 `leaderboard_test_file.csv`입니다.

2. train data와 test data를 폴더 내에 두었다면,

```
python main.py
```

를 호출하여 코드를 실행할 수 있습니다.