ESAA 제4회 컨퍼런스

DACON 뉴스 토픽 분류 AI 경진대회

ESAA OB 3조
김지민 장현정

대회 및 데이터 소개

✅ 목표: 한국어 뉴스 헤드라인을 이용하여, 뉴스의 주제를 분류하는 알고리즘 개발

🔍 데이터

- 종류: train_data / test_data / submission / topic_dict
- train.shape: (45654, 3)
- test.shape: (9131, 2)

	index	title	topic_idx
0	0	인천→핀란드 항공기 결항…휴가철 여행객 분통	4
1	1	실리콘밸리 넘어서겠다…구글 15조원 들여 美전역 거점화	4
2	2	이란 외무 긴장완화 해결책은 미국이 경제전쟁 멈추는 것	4
3	3	NYT 클린턴 측근韓기업 특수관계 조명…공과 사 맞물려종합	4
4	4	시진핑 트럼프에 중미 무역협상 조속 타결 희망	4

	topic	topic_idx
0	IT과학	0
1	경제	1
2	사회	2
3	생활문화	3
4	세계	4
5	스포츠	5
6	정치	6

EDA

- ✓ 결측치, 중복값 X
- ✓ topic별 문서 수, 문장 길이 차이 많이 나지 않음 ⇒ class 불균형으로 인한 문제 일어날 가능성 적음
- ✓ 최대 텍스트 길이 길지 않음
- ✓ train 데이터와 test 데이터의 텍스트 길이 차이 거의 안 남 ⇒ overfitting 가능성 낮음

- ✓ 각 topic별 자주 등장하는 단어 상위 10개

  (tf-idf로 추출)

	IT과학	경제	사회	생활문화	세계	스포츠	정치
0	출시	코스피	게시판	신간	트럼프	감독	朴대통령
1	개발	출시	코로나19	여행	이란	월드컵	대통령
2	5G	작년	지원	주말	터키	류현진	김정은
3	KT	영업익	KBS	출간	사망	아시안게임	북한
4	AI	특징주	사장	개막	중국	MLB	한국당
5	네이버	그래픽	개최	공연	대통령	시즌	문대통령
6	삼성	게시판	수능	한국	시리아	프로농구	정부
7	SKT	증가	선정	개최	미국	손흥민	여야
8	LG	1분기	모집	게시판	홍콩	꺾고	더민주
9	게시판	규모	MBC	서울	EU	SK	국회

전처리 전처리에서 고려할 것들

1. 텍스트 정제

- 숫자 제거
- 특수문자 제거

2. Tokenizer 선택

① 형태소분석 기반

- 사전 형태로 정의된 오픈소스 패키지 사용.
- 자주 쓰이는 단어가 많은 경우 좋은 결과

🧪 Mecab, Okt

② BPE 기반

- corpus(문서 전체 텍스트)를 주어주면 빈도를 계산하여 vocab 생성 후 tokenize
- 모르는 단어가 많은 경우 좋은 결과

🧪 WordPiece

전처리 전처리에서 고려할 것들

3. (Tokenizer를 기반으로) Vocabulary 생성

- vocab_size: 최대 크기 지정, 여기 없는 단어는 모두 <uknown> token으로 동일하게 처리됨
- min_freq: 최소 n번 등장해야 vocab에 들어가도록
- * vocab_size 미리 지정 안하고 min_freq=1이면, 데이터에서 나온 모든 단어가 vocab에 들어감

4. Encoding 방법 선택

① CountVectorizer: 각 문장에서 해당 token이 몇 번 등장했는지 count함

② TfidfVectorizer: 각 문장에서 해당 token의 tfidf 점수를 계산

③ 정수인코딩: sequential modeling에서 사용 (문장 내 token 순서 고려)

가벼운 모델인 **Logistic Regression**으로 여러 전처리 방법의 성능을 비교. train data의 30%를 test로 사용함

전처리

전체 성능 비교

tokenizer	텍스트 정제	vocab_size	vocab condition	encoding	최종 vocab size	model	Validation(30%) Accuracy
wordpiece	X	10000	min_freq=2	BOW - tfidf	9701	LogisticRegression	0.842301234
wordpiece	X	10000	min_freq=2	BOW - count	9701	LogisticRegression	0.830911879
wordpiece	X	10000	min_freq=3	BOW - tfidf	9701	LogisticRegression	0.842082208
wordpiece	X	10000	min_freq=3	BOW - count	9701	LogisticRegression	0.830984887
wordpiece	X	20000	min_freq=2	BOW - tfidf	19122	LogisticRegression	0.836825582
wordpiece	X	20000	min_freq=2	BOW - count	19122	LogisticRegression	0.830035774
wordpiece	X	20000	min_freq=3	BOW - tfidf	19121	LogisticRegression	0.83748266
wordpiece	X	20000	min_freq=3	BOW - count	19121	LogisticRegression	0.830400818
wordpiece	X	30000	min_freq=3	BOW - tfidf	28320	LogisticRegression	0.827407461
wordpiece	X	30000	min_freq=3	BOW - count	28320	LogisticRegression	0.829305687
mecab	X	-	-	BOW - tfidf	30459	LogisticRegression	0.847703877
mecab	X	-	-	BOW - count	30459	LogisticRegression	0.840110973
mecab	O	-	-	BOW - tfidf	30459	LogisticRegression	0.848287946
okt	X	-	-	BOW - tfidf	36110	LogisticRegression	0.842593269
okt	X	-	-	BOW - count	36110	LogisticRegression	0.84193619

- 숫자 및 특수문자 제거하는 것이 성능이 더 좋음
- vocab size가 작아도 성능이 좋을 수 있음
- Tokenizer는 mecab으로 사용
- Encoding은 count보다는 tfidf를 사용하는 것이 성능이 더 좋음
- LogisticRegression 모델로 성능 확인했을 때, tokenizer, vocabulary size, encoding방법 등의 전처리로 인한 성능 차이 크지는 않음

모델링 ML 기반 - 제출할 모델 추리기

📌 전처리 성능 비교 결과를 바탕으로 tokenizer와 encoding 방법 결정

- ✓ Tokenizer : Mecab 형태소 분석기
- ✓ Encoding 방법: TfidfVectorizer

📌 7가지 회귀 모델의 성능 비교

train data의 30%를 test로 사용함

LogisticRegression	DecisionTree	Random forest	GradientBoosting	XGBoost	LightGBM	SVM
0.8488	0.7057	0.7919	0.7517	0.708	0.8188	0.8506

➔ 가장 성능이 좋은 **LogisticRegression**, **LightGBM**, **SVM**을 사용하여 전체 train set에 대해 모델링 시도

모델링 ML 기반 - 기본 모델링

LogisticRegression, LightGBM, SVM을 사용하여 전체 train set에 대해 모델링 시도

```
tokenizer = Mecab()
train_text = [tokenizer.morphs(t) for t in train['title']]
test_text = [tokenizer.morphs(t) for t in test['title']]

vect = TfidfVectorizer(tokenizer=identity_tokenizer, lowercase=False)
vect = vect.fit(train_text)
matrix_train = vect.transform(train_text)
matrix_test = vect.transform(test_text)
```

LogisticRegression

```
lr = LogisticRegression(random_state = 42)
lr.fit(matrix_train,train['topic_idx'])
pred = lr.predict(matrix_test)
```

LGBM

```
lgbm = LGBMClassifier(random_state = 42)
lgbm.fit(matrix_train,train['topic_idx'])
pred = lgbm.predict(matrix_test)
```

SVM

```
svm = SVC(random_state = 42)
svm.fit(matrix_train,train['topic_idx'])
pred = svm.predict(matrix_test)
```

☑ 제출 점수

- LogisticRegression: 0.82344
- LightGBM : 0.79079
- SVM : 0.82322

모델링 ML 기반 - 성능 향상을 위한 시도 1

1) 전처리 방법 변경: 추가적인 형태소 분석기 고려

- ✓ Mecab : 빠른 속도와 띄어쓰기에서 좋은 성능, 미등록어와 동음이의어 처리의 한계
- ✓ Okt : 정제되지 않은 데이터에 대해서 강점, 미등록어와 동음이의어 처리의 한계
- ✓ Komoran : 고유명사와 오탈자 구분에 강점, 띄어쓰기 없는 문장 분석에 취약
- ✓ Kkma : 띄어쓰기 오류에 덜 민감, 시간이 가장 오래 걸림

```
Mecab ['카니발', '축제', '보', '러', '가', '자', '브라질', '리우', '에', '대형', '유람선', '행렬']
Okt ['카니발', '축제', '보러', '가자', '브라질', '리우', '에', '대형', '유람선', '행렬']
Komoran ['카니발', '축제', '보', '러', '가자', '브라질', '리우', '에', '대형', '유람선', '행렬']
Kkma ['카니발', '축제', '보', '러', '가', '자', '브라질', '리', '우', '에', '대형', '유람선', '행렬']

Mecab ['美', '올랜도', '병원', '최악', '총기', '테러', '부상자', '치료비', '안', '발', '는다']
Okt ['美', '올랜도', '병원', '최악', '총기', '테러', '부상', '자', '치료', '비', '안', '받는다']
Komoran ['美', '올랜도', '병원', '최악', '총기', '테러', '부상자', '치료비', '안', '발', '는다']
Kkma ['美', '올', '랜', '도', '병원', '최악', '총기', '테러', '부상자', '치료비', '안', '발', '는', '다']
```

📌 정확도 측정 결과, 기존에 사용하였던 Mecab의 성능이 가장 좋음. Tokenizer 변경 X

모델링 ML 기반 - 성능 향상을 위한 시도 2

2) 하이퍼파라미터 튜닝 & 교차검증

✓ **GridSearchCV (그리드 탐색)** : 탐색하고자 하는 하이퍼파라미터와 시도해 볼 값을 지정한 후,
가능한 모든 파라미터 조합에 대해 교차 검증을 진행하여 평가

🧪 기본 모델링 성능이 좋았던 LogisticRegression, SVM을 사용하여 모델링 시도

```
# LogisticRegression
lr_clf = LogisticRegression(random_state=0)
params = {'C': [0.01, 0.1, 1, 10, 100], 'penalty': ['l1', 'l2']}
grid_cv = GridSearchCV(lr_clf, param_grid=params, cv=3, scoring='accuracy', verbose=1)
grid_cv.fit(matrix_train,train['topic_idx'])
```

✓ **제출 점수:** 0.8245

```
# SVM
sv_clf = svm.SVC(random_state=0)
params = [{'kernel':['linear'],'C':[1, 3.5, 5.5, 10]},
          {'kernel':['rbf'],'C':[1, 3.5, 5.5, 10]}]
grid_cv = GridSearchCV(sv_clf, param_grid=params, cv=3, scoring='accuracy', verbose=1)
grid_cv.fit(matrix_train,train['topic_idx'])
```

✓ **제출 점수:** 0.8324

➔ 하이퍼파라미터 튜닝 결과 성능이 향상됨

모델링 BERT - 전처리

- ☑ **Tokenizer:** Mecab 형태소 분석기
- ☑ **Vocabulary**
  - 전체 token 수: 30,903
  - Vocab 포함 조건: 3회 이상 등장
  - 최종 vocab 크기: 14,603
- ☑ **Encoding**
  - 순서를 고려한 정수 encoding
  - 최대 텍스트 길이인 26으로 max padding
  - Padding한 token은 계산하지 않기 위해 attention masking

모델링 BERT - 학습

✓ 학습 조건

- BERT pretrained model from: Huggingface `"bert-base-multilingual-cased"`
- Batch size: 32
- Optimizer: adam
- Learning reate: 2e-5
- Scheduler: linear scheduler
- Epochs: 5. 10
- Validation set X

✓ Epoch별 loss

epoch	1	2	3	4	5	6	7	8	9	10
loss	0.32	0.24	0.18	0.14	0.11	0.08	0.06	0.05	0.04	0.03

✓ 제출 점수: 0.7253

제출 결과 전체비교

Model	Model detail	제출 점수 (Accuracy)
LightGBM	base	0.7908
LinearRegression	base	0.8234
LinearRegression	vocab_size=20000	0.8223
LinearRegression	hyperparameter tuned	0.8245
LinearRegression	vocab w/ train+test	0.8217
SVM	base	0.8232
SVM	hyperparameter tuned	0.8324
BERT	epoch5	0.7279
BERT	epoch10	0.7253

📌 하이퍼파라미터 튜닝을 한 SupportVectorMachine의 성능이 가장 좋음

Q & A

감사합니다