Week7

Created By	DongGu Kim
Last Edited	@May 01, 2020 6:01 PM
Property	
Tags	

1. 스미싱 문자 분류기로 LGBM, SVM을 사용한 이유

2. LightGBM Classifier

3. Support Vector Machine Classifier

4. LGBM V.S. SVM

참고

- LGBM과 SVM 모델 성능을 비교하기 위해, Train/Test 및 random seed같은 변수는 그대로 두었음.

---

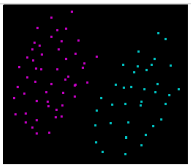
1. 스미싱 문자 분류기로 LGBM, SVM을 사용한 이유

- LGBM: 빠른 속도, 높은 정확성에 낮은 메모리 사용량
- SVM: 텍스트 데이터 분류에 선형모델 적합함

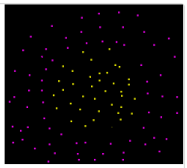
Linear Kernel: Why is it recommended for text classification ?

The Support Vector Machine can be viewed as a kernel machine. As a result, you can change its behavior by using a different kernel function.

https://www.svm-tutorial.com/2014/10/svm-linear-kernel-good-text-classification/



Linearly separable data



Non linearly separable data

---

2. LightGBM Classifier

1) 필요한 라이브러리 호출

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from lightgbm import LGBMClassifier
import lightgbm
import joblib
```

2) feature: 문자 메시지 내용, label: 스미싱 문자 여부 지정

```
X_origin = vec_train
Y_label = prepared_train['smishing']
```

3) train-test 분리

- 랜덤 추출을 하려고 했으나 인덱싱 관련 오류로, 그냥 ID가 빠른 순서대로 80% trainset, 하위 20%을 testset으로 줬다.
- 인덱싱 오류의 원인을 찾으면 랜덤추출로 전환할 예정이다.

```
%%time
X_train, X_test, Y_train, Y_test = train_test_split(X_origin,
                                                    Y_label,
                                                    test_size = 0.2,
                                                    shuffle = False)
```

4) K-Fold Cross Validation (교차검증)

- 전체 문자메시지(30만 건) 중 6%만(2만 건)이 스미싱 문자 데이터
- Imbalanced data를 보완하기 위해 교차 검증 실시
- 대신 별도의 Imbalanced data 개선하기 위한 작업은 하지 않음.
- 모델 정확도를 보고 추후 대책을 결정할 것임.
- seed 3개 * fold 4개 = 총 12개 모델

```
lucky_seed = [5, 8 , 10]

# enumerate: 인덱스와 값을 둘다 반복시킬 때 사용
for num,rs in tqdm(enumerate(lucky_seed)):


    kfold = KFold(n_splits=4, random_state = rs, shuffle = True)

    # numpy.zeros((row,col))
    # row*col size 영행렬 생성
    # train.shape[0],198 -> trainset 41400개, target값:198개
    cv=np.zeros((X_train.shape[0],2))

    for n, (train_idx, validation_idx) in tqdm(enumerate(kfold.split(X_train))):

        x_train, x_validation = X_train[train_idx], X_train[validation_idx]
        y_train, y_validation = Y_train.loc[train_idx], Y_train.loc[validation_idx]

        lgbm = LGBMClassifier(n_estimators=380,
                    learning_rate=0.035,
                    max_depth=7,
                    min_child_samples=50,
                    random_state=4321)

        lgbm.fit(x_train, y_train, eval_set=[(x_validation, y_validation)], early_stopping_rounds= 30, verbose=100)

        # 모델결과 저장 lib
        joblib.dump(lgbm, 'models/%s_fold_model_%s.pkl'%(n,rs))

        # numpy.zeros((row,col))로 만들어주었던 영행렬: cv
        # data object에 X_validation 예측 값을 넣어줌
        # CROSS-VALIDATION , EVALUATE CV
        cv[validation_idx,:] = lgbm.predict_proba(x_validation)
```

```
# MODEL LOAD & TEST PREDICT
# 12 MODELS 평균 사용
models = os.listdir('models/')
models_list = [x for x in models if x.endswith(".pkl")]

# 모델결과가 잘 나왔는지 check
# assert: 좌항과 우항의 값이 같으면 정상 작동, 다르면 오류 발생
assert len(models_list) ==12


temp_predictions = np.zeros((X_test.shape[0],2))

# 12개 모델을 반복시켜서 결과산출 -> 12로 나눠서 평균값 계산
for model in models_list:
    model = joblib.load('models/'+model)
    predict_proba = model.predict_proba(X_test)
    temp_predictions += predict_proba/12
```

```
submission = pd.DataFrame(data=np.zeros((X_test.shape[0],2)))
submission.index = Y_test.index
submission.index.name = 'id'
submission+=temp_predictions

submission = submission.sort_index()
submission = submission.groupby('id').mean()
submission.to_csv('submission.csv', index=True)
```

```
submission['pred'] = 0
for idx in submission.index:
    if (submission[0][idx] < submission[1][idx]):
        submission['pred'][idx]= 1
```

5) 결과

- 모델링 시간: 약 45분
  - %%time을 빼먹어서 각각 모델링에 대한 시간을 더해줌

```
from sklearn.metrics import classification_report
result = classification_report(Y_test, submission['pred'], target_names=['normal','smishing'])
```

- Precision, Recall, F1-score
  - 결과

```
print(result)
```

	precision	recall	f1-score	support
normal	1.00	1.00	1.00	51205
smishing	1.00	0.97	0.99	7984
accuracy			1.00	59189
macro avg	1.00	0.99	0.99	59189
weighted avg	1.00	1.00	1.00	59189

- Precision: 모델이 True로 분류한 것 중 실제 True인 비율

$$(Precision) = \frac{TP}{TP + FP}$$

- Recall: 실제 True인 것 중에서 모델이 True라고 예측한 비율

$$(Recall) = \frac{TP}{TP + FN}$$

- F1-score: Precision과 Recall의 조화평균
  - 사용 이유
    - (직관적인 설명) 만약 label 데이터 비율이 A:B가 99:1이라면, 무조건 A라고 찍으면 Accuracy가 99%가 나온다. 하지만 이것을 정확한 모델이라고 보기는 힘들다.
    - 불균형한 label을 보완하기 위해 조화평균을 쓴다.

$$(F1\text{-}score) = 2\times\frac{1}{\frac{1}{Precision}+\frac{1}{Recall}} = 2\times\frac{Precision \times Recall}{Precision + Recall}$$

- 결과가 너무 정확해서 Overfitting이 된 것이 아닌지 불안감이 듦
  - 시드, train-testset 조정 후 다시 검토 예정

---

3. Support Vector Machine Classifier

1) 필요한 라이브러리 호출

```
from sklearn.calibration import CalibratedClassifierCV
from sklearn.svm import LinearSVC
```

2) feature: 문자 메시지 내용, label: 스미싱 문자 여부 지정

```
X_origin = vec_train
Y_label = prepared_train['smishing']
```

3) train-test 분리

- 랜덤 추출을 하려고 했으나 인덱싱 관련 오류로, 그냥 ID가 빠른 순서대로 80% trainset, 하위 20%을 testset으로 줬다.
- 인덱싱 오류의 원인을 찾으면 랜덤추출로 전환할 예정이다.

```
%%time
X_train, X_test, Y_train, Y_test = train_test_split(X_origin,
                                                    Y_label,
                                                    test_size = 0.2,
                                                    shuffle = False)
```

4) K-Fold Cross Validation (교차검증)

- 전체 문자메시지(30만 건) 중 6%만(2만 건)이 스미싱 문자 데이터
- Imbalanced data를 보완하기 위해 교차 검증 실시
- 대신 별도의 Imbalanced data 개선하기 위한 작업은 하지 않음.
- 모델 정확도를 보고 추후 대책을 결정할 것임.
- seed 3개 * fold 4개 = 총 12개 모델

```
%%time
# 4FOLD, 3SEED ENSEMBLE
# 총 12개의 모델을 평균내어 예측한다

#lucky_seed=[1996, 8, 25]
lucky_seed = [5, 8 , 10]

# enumerate: 인덱스와 값을 둘다 반복시킬 때 사용
for num,rs in tqdm(enumerate(lucky_seed)):


    kfold = KFold(n_splits=4, random_state = rs, shuffle = True)

    # numpy.zeros((row,col))
    # row*col size 영행렬 생성
    # train.shape[0],198 -> trainset 41400개, target값:198개
    cv=np.zeros((X_train.shape[0],2))

    for n, (train_idx, validation_idx) in tqdm(enumerate(kfold.split(X_train))):

        x_train, x_validation = X_train[train_idx], X_train[validation_idx]
        y_train, y_validation = Y_train.loc[train_idx], Y_train.loc[validation_idx]

        model_svc = LinearSVC(class_weight='balanced',
                              random_state=4321)

        svm_model = CalibratedClassifierCV(model_svc)

        svm_model.fit(x_train, y_train)

        # 모델결과 저장 lib
        joblib.dump(svm_model, 'svm_models/%s_fold_model_%s.pkl'%(n,rs))

        # numpy.zeros((row,col))로 만들어주었던 영행렬: cv
        # data object에 X_validation 예측 값을 넣어줌
        # CROSS-VALIDATION , EVALUATE CV
        cv[validation_idx,:] = svm_model.predict_proba(x_validation)
```

```
# MODEL LOAD & TEST PREDICT
# 12 MODELS 평균 사용
models = os.listdir('svm_models/')
models_list = [x for x in models if x.endswith(".pkl")]
```

```
# 모델결과가 잘 나왔는지 check
# assert: 좌항과 우항의 값이 같으면 정상 작동, 다르면 오류 발생
assert len(models_list) ==12


temp_predictions = np.zeros((X_test.shape[0],2))

# 12개 모델을 반복시켜서 결과산출 -> 12로 나눠서 평균값 계산
for model in models_list:
    model = joblib.load('svm_models/'+model)
    predict_proba = model.predict_proba(X_test)
    temp_predictions += predict_proba/12
```

```
# dacon code
submission = pd.DataFrame(data=np.zeros((X_test.shape[0],2)))
submission.index = Y_test.index
submission.index.name = 'id'
submission+=temp_predictions

submission = submission.sort_index()
submission = submission.groupby('id').mean()
submission.to_csv('svm_submission.csv', index=True)
```

```
submission['pred'] = 0
for idx in submission.index:
    if (submission[0][idx] < submission[1][idx]):
        submission['pred'][idx]= 1
```

5) 결과

- 모델링 시간: 약 2분
  - LGBM에 비해 더 속도가 빨랐다.
    - lgbm은 model.fit에서 'eval_set' 파라미터를 추가해 evaluation 결과를 반영하도록 추가되어 있음.
    - svm은 eval_set 같은 파라미터를 따로 설정하는 법을 몰라서, 바로 model.fit을 진행함.
  - 분석 속도의 차이가 eval_set 유무에 따른 차이인지 확인해볼 필요가 있음.

```
from sklearn.metrics import classification_report
result = classification_report(Y_test, submission['pred'], target_names=['normal','smishing'])
```

- Precision, Recall, F1-score
  - 결과

```
print(result)
```

	precision	recall	f1-score	support
normal	1.00	1.00	1.00	51205
smishing	1.00	0.98	0.99	7984
accuracy			1.00	59189
macro avg	1.00	0.99	0.99	59189
weighted avg	1.00	1.00	1.00	59189

- Precision: 모델이 True로 분류한 것 중 실제 True인 비율

$$(Precision) = \frac{TP}{TP + FP}$$

- Recall: 실제 True인 것 중에서 모델이 True라고 예측한 비율

$$(Recall) = \frac{TP}{TP + FN}$$

- F1-score: Precision과 Recall의 조화평균
  - 사용 이유
    - (직관적인 설명) 만약 label 데이터 비율이 A:B가 99:1이라면, 무조건 A라고 찍으면 Accuracy가 99%가 나온다. 하지만 이것을 정확한 모델이라고 보기는 힘들다.
    - 불균형한 label을 보완하기 위해 조화평균을 쓴다.

$$(F1\text{-}score) = 2\times\frac{1}{\frac{1}{Precision}+\frac{1}{Recall}} = 2\times\frac{Precision \times Recall}{Precision + Recall}$$

- 결과가 너무 정확해서 Overfitting이 된 것이 아닌지 불안감이 듦
  - 시드, train-testset 조정 후 다시 검토 예정

---

4. LGBM V.S. SVM

1) 단순 정확도(Accuracy) 비교

- LGBM: 정확도 0.9961 (58960 / 59189)
- SVM: 정확도 0.9974 (59037 / 59189)

2) classification report 비교

- Recall이 SVM 모델이 0.01 더 높음
- 나머지는 동일

3) 앞으로..

- Overfitting된 것이 아닌지 점검
- 모델링, 예측 시간 비교
  - SVM이 LGBM에 비해 월등히 모델링이 빨랐던 원인 탐색
- 딥러닝 LSTM 모델링 추가