Machine Learning Ensemble Strategy

그룹과제03-이준영-박상수-박지훈

DecisionTreeClassifier

DecisionTreeClassifier	Private	Public	Avg	Rank(Private)
JamesSteinEncoding	0.70694	0.70692	0.70693	1
M-estimatorEncoding	0.70687	0.70698	0.706925	2
WeightofEvidenceEncoding	0.70682	0.70867	0.707745	3
MeanEncoding	0.7061	0.70731	0.706705	4
CatBoostEncoding	0.70196	0.70283	0.702395	5
LabelEncoding	0.69345	0.69317	0.69331	6
OrdianlEncoding	0.69345	0.69317	0.69331	6
OneHotEncoding	0.6792	0.6777	0.67845	8
FrequencyEncoding	0.67878	0.67759	0.678185	9
BaseNEncoding	0.66681	0.66442	0.665615	10
HashingEncoding	0.5876	0.58689	0.587245	11
BinaryEncoding	0.50793	0.50443	0.50618	12
ProbabilityRatioEncoding	0.49995	0.4985	0.499225	13
LeaveOneOutEncoding	0.49995	0.49954	0.499745	13

DecisionTreeClassifier

Best Encoder

상위권에 있는 인코더는 JamesSteinEncoder와 M-estimator등 Target Encoding계열의 인코더였으며, 각 점수 차이가 유의미하게 크지 않았다

Comment:

성능이 잘 나온 이유는 데이터셋은 높은 Cardinality를 가졌는데 정보손실을 최소화하면서 타겟의 정보를 반영했기 때문이다

Worst Encoder

하위권에는 주로 OneHotEncoder계열의 인코더들(BaseN,Hashing,Binary)이 있었다

Comment:

성능이 잘 나오지 않은 이유는 Tree계열의 모델은 Feature에 0이 많을수록 성능이 떨어지기 때문이다 심지어 Cardinality를 축소하는 Binary, BaseN, Hashing 인코더는 Ohe보다 더 낮은 성능을 보였다

LogisticRegression

LogisticRegression	Private	Public	Avg	Rank(Private)
LeaveOneOutEncoding	0.786	0.78464	0.78532	1
OneHotEncoding	0.78318	0.78183	0.782505	2
WeightofEvidenceEncoding	0.78172	0.7808	0.78126	3
JamesSteinEncoding	0.78168	0.78088	0.78128	4
MeanEncoding	0.78161	0.78059	0.7811	5
M-estimatorEncoding	0.7813	0.78038	0.78084	6
CatBoostEncoding	0.78122	0.77993	0.780575	7
LabelEncoding	0.64005	0.63989	0.63997	8
FrequencyEncoding	0.63777	0.63638	0.637075	9
OrdianlEncoding	0.63508	0.6349	0.63499	10
BaseNEncoding	0.62669	0.62254	0.624615	11
HashingEncoding	0.60714	0.60358	0.60536	12
BinaryEncoding	0.51623	0.50999	0.51311	13
ProbabilityRatioEncoding	0.50006	0.49985	0.499955	14

LogisticRegression

Best Encoder

LeaveOneOutEncoding이 가장 좋은 성능을 보였고
그 다음으로는 Ohe였다. 1~7등에는 TargetEncoding
기반의 인코딩 기법이 대체적으로 해당 모델에서
좋은 성능을 보였다.

Comment:

성능이 월등히 높은 Looe와 3~7위에 위치한
타 인코더들과의 차이는 Overfitting과
Data leakage 고려 유무에 있다고 생각한다

Worst Encoder

Label, Ohe계열의 인코더들은 하위권에 분포하고 있었다
특히 8위 LabelEncoding과 CatBoostEncoding은
0.14로 큰 성능 차이를 보였다

Comment:

Regression의 모델 특성상 대소관계를 가지지 않는
Label값을 부여하는 것은 낮은 성능을 야기하기 때문이다
또한, Ohe는 좋은 성능을 보였으나, 정보를 축소시킨
HashingEncoder는 낮은 성능을 보였다 n_component를
늘릴수록 성능 개선이 짐작된다

MLPClassifier

MLPClassifier	Private	Public	Avg	Rank(Private)
LeaveOneOutEncoding	0.78606	0.78487	0.785465	1
CatBoostEncoding	0.78348	0.78223	0.782855	2
MeanEncoding	0.78216	0.78115	0.781655	3
JamesSteinEncoding	0.78173	0.78101	0.78137	4
M-estimatorEncoding	0.78144	0.78033	0.780885	5
WeightofEvidenceEncoding	0.78087	0.77999	0.78043	6
BaseNEncoding	0.71974	0.71827	0.719005	7
LabelEncoding	0.71207	0.71088	0.711475	8
OrdianlEncoding	0.71188	0.7111	0.71149	9
FrequencyEncoding	0.6777	0.67437	0.676035	10
HashingEncoding	0.61391	0.61168	0.612795	11
BinaryEncoding	0.51679	0.51449	0.51564	12
ProbabilityRatioEncoding	0.50023	0.49995	0.50009	13

MLPClassifier

Best Encoder

LeaveOneOutEncoding이 가장 좋은 성능을 보였다
또한 타 모델과 마찬가지로 Target의 평균을 반영하는
인코더들이 좋은 성능을 보였음을 알 수 있었다

Comment:

Looe는 타겟에 대한 평균을 낼 때,
해당 샘플을 제외한 평균을 내는 것으로 알고있다.
이로써 이상치를 완만하게 제어할 수 있게 된다.
따라서 Test Target에 대한 Roc-auc성능이
높게 나온것을 짐작할 수 있었다

Worst Encoder

하위권의 인코더들은 Binary, Hashing, Label 등으로
변수가 팽창되거나 라벨을 부여하는 인코더들은
대체적으로 낮은 성능을 보였다

Comment:

MLP모델의 특성 상, Input layer와 Column의 수는
비례하기 때문에 Ohe계열의 인코더들이 시간 대비
모델의 효율성이 심각하게 떨어졌음을 포착했다

추가 학습 및 한계점

●BaseNEncoder의 발견

Binary Encoder는 Cardinality가 높을 때 유리하게 사용될 수 있다.
BaseNEncoder는 이와 비슷하게 지정한 N(자연수)값에 따라 N진수 형태의 인코딩을 한다는 것을 추측할 수 있었다
모든 모델에서 BaseNEncoder는 BinaryEncoder보다 0.1~0.2 높은 성능을 보였다

●MLP의 Input layer와 Columns

MLP 모델은 Input layer를 통해 hidden layer를 거쳐 output layer로 가는 신경망 구조의 모델이다
또한 Column이 많아질수록 Input layer가 많이 생성되어 높은 Cardinality를 가질수록 연산이 오래 걸린다는 것을 알았다

●High Cardinality를 다루는 법

높은 Cardinality를 가지는 변수는 TargetEncoder으로도 다룰 수 있지만
Threshold를 이용하여 나머지를 기타 값으로 처리하는 방법도 있을 수 있음을 깨달았다.
그 말인 즉, 전체 데이터의 90%에 해당하는 데이터까지 살리고 나머지 10%에 해당하는 범주는
모두 기타값으로 처리하는 방식이다. 이로써 High Cardinality를 조금은 해소할 수 있음을 알게 되었다

●DT_LeaveOneOutEncoder 문제

DT에서 Looe를 실행하고 predict_proba가 산출되지 않고 클래스를 받았다
해당 원인을 계속 찾고 분석했으나 이유를 알 수 없었다
만약 predict_proba값이 나왔다면 해당 모델에서 Roc_auc 점수가 높게 나왔으리라 예상한다

선택과제_Best Encoder

::Work Flow::

01 Cardinality에 따른 변수 분할

Binary, low_nom, high_nom, low_ord, high_ord, cyclic

02 데이터셋 전처리 -최빈값

SimpleInputer

03 범주별 인코딩 조합 시도

KFoldTargetEncoding
CyclicEncoding
LooEncoding

04 모델 성능 확인

DT : 0.70858
LR : 0.78339
MLP : 0.78229

05 문제 확인 및 오류 수정

Change Combination

선택과제_Best Encoder

●Cardinality에 따른 변수 분할

```
cat_all = ['bin_0', 'bin_1', 'bin_2', 'bin_3', 'bin_4', 'nom_0', 'nom_1',
       'nom_2', 'nom_3', 'nom_4', 'nom_5', 'nom_6', 'nom_7', 'nom_8',
       'nom_9','ord_0', 'ord_1', 'ord_2', 'ord_3', 'ord_4', 'ord_5','day','month']
binary = ['bin_0','bin_1','bin_2','bin_3', 'bin_4']
low_nom = ['nom_0', 'nom_1','nom_2', 'nom_3', 'nom_4']
high_nom = ['nom_5', 'nom_6', 'nom_7', 'nom_8', 'nom_9']
low_ord = ['ord_0', 'ord_1', 'ord_2']
high_ord = ['ord_3', 'ord_4', 'ord_5']
cyclic = ['day','month']
```

Cardinality가 높고 낮음에 따라서 15개가 넘으면 High-Cardinality 라고 간주하여 인코딩을 다르게 하려고 하였다

●Data Preprocessing

```
#문자형 -> binary 변환
X_train['bin_3']= np.where(X_train['bin_3']=='F',0,1)
X_train['bin_4']= np.where(X_train['bin_4']=='N',0,1)
X_test['bin_3']= np.where(X_test['bin_3']=='F',0,1)
X_test['bin_4']= np.where(X_test['bin_4']=='N',0,1)
```

```
#submission 제출을 위해 결측값 제거 대신 대체값(최빈값) 사용
from sklearn.impute import SimpleImputer

imp = SimpleImputer(strategy="most_frequent")

X_train[cat_all] = imp.fit_transform(X_train[cat_all])
X_test[cat_all] = imp.transform(X_test[cat_all])
X_train = X_train.drop(columns=['id'])
X_test = X_test.drop(columns=['id'])
```

문자형으로 이진 분류된 변수를 Integer 형태로 변환하였고,
Submission 제출을 위해 결측값 제거 대신 최빈값 대체를 사용하였다

선택과제_Best Encoder

●K_Fold_Target_Encoder

```
#high_nom -> Kfold_Te
#high_ord -> Kfold_Te
from sklearn.model_selection import KFold
X_train_imp, X_test_imp = X_train_en, X_test_en

X_train_imp_te, X_test_imp_te, = X_train_en.copy(), X_test_en.copy()

for c in high_ord+high_nom+low_ord:
    data_tmp = pd.DataFrame({c: X_train_imp[c], 'target': y_train})
    target_mean = data_tmp.groupby(c)['target'].mean()
    X_test_imp_te[c] = X_test_imp[c].map(target_mean)
    tmp = np.repeat(np.nan, X_train_imp.shape[0])

    kf = KFold(n_splits=4, shuffle=True, random_state=0)
    for idx_1, idx_2 in kf.split(X_train_imp):

        target_mean = data_tmp.iloc[idx_1].groupby(c)['target'].mean()

        tmp[idx_2] = X_train_imp[c].iloc[idx_2].map(target_mean)

    X_train_imp_te[c] = tmp
```

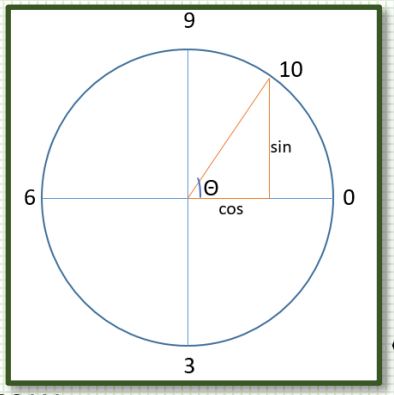
MeanEncoding 실행 시 , Train Set에 대한 OverFitting을
방지하기 위해 Kfold를 같이 시행하였다
적용 대상 변수 : High_nom , Low_ord, High_nom

●Cyclic_Encoding

```
def encode(data, col, max_val):
    data[col]=data[col].astype(int)
    data[col + '_sin'] = np.sin(2 * np.pi * data[col]/max_val)
    data[col + '_cos'] = np.cos(2 * np.pi * data[col]/max_val)
    data.drop(columns=[col],inplace=True)
    return data

encode(X_train_imp_te, 'day', 7)
encode(X_train_imp_te, 'month', 12)
encode(X_test_imp_te, 'day', 7)
encode(X_test_imp_te, 'month', 12)
```

시계열 데이터는 순환하는 데이터이기 때문에
타 범주형 데이터와는 다른 성격을 가졌다.
따라서 sin, cos 함수를 이용하여 위와 같이 인코딩 하였다



〈시계열 데이터의 논리구조〉

참고자료:https://www.kaggle.com/code/mansipriya/handling-cyclicdata/notebook?scriptVersionId=31108111

FINAL & CONCLUSION

●DecisionTree_Best_Encoder

Binary – X
Low_nom – JamesSteinEncoder
High_nom, Low_ord, High_ord – KfoldTargetEncoding
Cyclic – Sin/Cos Encoding
Roc-auc : 0.70858
Comment : 전체 데이터에 일괄적으로 시도한 모든 인코더들보다 높은 성능을 기록했다
HighCardinality와 Overfitting을 극복하기 위하여 KfoldTargetEncoding을 실시했고
시계열 데이터의 성격에 맞는 인코딩을 했다는 점에서 의미가 있다

●LogisticRegression_Best_Encoder

Binary – X
All variable expect for Binary – LeaveOneOutEncoder
Roc-auc : 0.78339
Comment : LR에서 가장 성능이 좋았던 Looe를 쓰되 Binary를 굳이 인코딩을 해야 하는지에 대한 질문을 던졌고,
실행 결과 해당 점수가 나왔다. 전체 데이터에 Looe를 적용한 결과보다 못한 이유는
Binary에 Target 평균의 정보가 빠져있기 때문이라고 추측했다

●MLPClassifier_Best_Encoder

Binary, Low_nom – OneHotEncoder
High_nom, Low_ord, High_ord, Cyclic – LeaveOneOutEncoder
Roc-auc : 0.78229
Comment : 기존의 Looe 기록보다 못한 결과이다. 이유는 Ohe가 MLPClassifier와 잘 맞지 않기 때문인 것 같다
Ohe 계열의 Encoder들이 전체적으로 하위권에 위치해 있었기 때문에, 가장 좋은 인코더와 나쁜 인코더를 결합했을 때
Ohe와 MLP 모델과의 시너지가 좋지 않다는 것을 검증했다는 데에 의미가 있다

감사합니다 ☺