KUBIG
CONTEST

머신러닝 분반 3조

이영신 우유정 유정아

이가영 전지우 하은겸

프로젝트 개요

- *The dataset consists of data collected from heavy Scania trucks in everyday usage. The system in focus is the Air Pressure system (APS) which generates pressurized air that are utilized in various functions in a truck, such as braking and gear changes. The dataset's positive class consists of component failures for a specific component of the APS system. The negative class consists of trucks with failures for components not related to the APS. The data consists of a subset of all available data, selected by experts.*
- *Our goal is to minimize the cost associated with:*
- *1) Unnecessary checks done by mechanic. (10$)*
- *2) Missing a faulty truck, which may cause breakdown. (500$)*

Objective : Our main objective is to correctly predict if truck needed to be serviced or not and minimize the cost of service.

프로젝트 개요

<Train set>

	Unnamed: 0	class	aa_000	ab_000	ac_000	ad_000	ae_000	af_000	ag_000	ag_001	...	ee_002	ee_003	ee_004	ee_005	ee_006	ee_007	ee_008
0	52803	neg	41386	NaN	508	488	0	0	0	0	...	438088	202172	383094	392838	228526	104226	122526
1	38189	neg	29616	NaN	1616	1490	0	0	0	0	...	145524	72858	171332	308328	379466	213826	5764
2	23291	neg	241352	NaN	NaN	NaN	NaN	NaN	0	0	...	3617298	2477772	3631902	997462	436380	202002	173850
3	16862	neg	8100	NaN	86	76	0	0	0	0	...	66980	36658	91898	86634	60276	23616	7518
4	14055	neg	2290	NaN	636	448	0	0	0	0	...	11542	7394	14206	69592	3108	108	6

5 rows × 172 columns

<Test set from *Kaggle*>

	class	aa_000	ab_000	ac_000	ad_000	ae_000	af_000	ag_000	ag_001	ag_002	...	ee_002	ee_003	ee_004	ee_005	ee_006	ee_007	ee_008	ee_009
0	neg	60	0	20	12	0	0	0	0	0	...	1098	138	412	654	78	88	0	0
1	neg	82	0	68	40	0	0	0	0	0	...	1068	276	1620	116	86	462	0	0
2	neg	66002	2	212	112	0	0	0	0	0	...	495076	380368	440134	269556	1315022	153680	516	0
3	neg	59816	na	1010	936	0	0	0	0	0	...	540820	243270	483302	485332	431376	210074	281662	3232
4	neg	1814	na	156	140	0	0	0	0	0	...	7646	4144	18466	49782	3176	482	76	0

5 rows × 171 columns

Data Imbalance 확인

<Training set imbalance>

```
neg     55934
pos      1066
Name: class, dtype: int64
```

<Test set imbalance>

```
neg     15625
pos       375
Name: class, dtype: int64
```

Train data와 Test data는 imbalanced data

Train data → pos : neg=1:50

Test data → pos : neg=1:40

전처리

Train data를 pos:neg=1:50이 되도록 stratify를 이용했으며
80:20 비율로 training-validation set을 나누었다.

```
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val=train_test_split(train, train["class"], test_size=0.2, random_state=42, stratify=train["class"])
```

```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 45600 entries, 36627 to 50505
Columns: 172 entries, Unnamed: 0 to eg_000
dtypes: int64(2), object(170)
memory usage: 60.2+ MB
```

```
X_val.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11400 entries, 28199 to 6799
Columns: 172 entries, Unnamed: 0 to eg_000
dtypes: int64(2), object(170)
memory usage: 15.0+ MB
```

전처리 결과

<Train set>

	Unnamed: 0	class	aa_000	ab_000	ac_000	ad_000	ae_000	af_000	ag_000	ag_001	...	ee_002	ee_003	ee_004	ee_005	ee_006	ee_007	ee_008
36627	58126	neg	46976	0	128	124	0	0	0	0	...	176578	80224	141948	164404	1438208	32222	520
42898	72727	neg	39910	NaN	70	66	0	0	0	0	...	291494	110406	265298	254714	232414	182932	309914
23114	60535	neg	43614	NaN	152	144	0	0	0	0	...	314196	146948	297180	274392	247178	193972	320904
2962	58060	neg	60	NaN	0	NaN	0	0	0	0	...	578	190	468	732	138	0	0
45204	57687	neg	38938	NaN	460	150	0	0	0	0	...	459428	220256	413674	334330	196244	92842	57548

<Validation set>

	Unnamed: 0	class	aa_000	ab_000	ac_000	ad_000	ae_000	af_000	ag_000	ag_001	...	ee_002	ee_003	ee_004	ee_005	ee_006	ee_007	ee_008
28199	40348	neg	39034	NaN	132	108	0	0	0	0	...	154734	69690	165178	133902	443552	691076	6636
49371	19970	pos	349286	NaN	NaN	NaN	NaN	NaN	0	26204	...	5012822	1532928	3381640	4543016	655000	207038	3480
56441	7008	neg	32400	NaN	714	626	0	0	0	0	...	320846	168344	408888	330388	153806	65222	52630
13775	69658	neg	378224	NaN	36	16	0	0	0	0	...	53412	24186	43278	37390	48242	293878	30980
38520	31183	neg	61174	NaN	0	NaN	0	0	0	0	...	417030	213806	466264	560570	527686	293296	224894

결측치 처리

Train set, Validation set과 Test set의 결측치를 -1로 대체

Column을 삭제할 경우 정보 손실이 우려

<Train set>

```
X_train = X_train.replace(np.nan, -1) #결측치에 -1 대입
X_train01=X_train.drop(['Unnamed: 0', 'class'], axis=1)
X_train01.reset_index(drop=True, inplace=True)
X_train01
```

<Validation set>

```
X_val = X_val.replace(np.nan, -1)
X_val01=X_val.drop(['Unnamed: 0', 'class'], axis=1)
X_val01.reset_index(drop=True, inplace=True)
X_val01
```

<Test set>

```
test = test.replace("na", -1)
test1 = test.drop(['class'], axis=1)
test1
```

결측치 처리 결과

<Train set>

	aa_000	ab_000	ac_000	ad_000	ae_000	af_000	ag_000	ag_001	ag_002	ag_003	...	ee_002	ee_003	ee_004	ee_005	ee_006	ee_007	ee_008
0	46976	0	128	124	0	0	0	0	0	0	...	176578	80224	141948	164404	1438208	32222	520
1	39910	-1	70	66	0	0	0	0	0	0	...	291494	110406	265298	254714	232414	182932	309914
2	43614	-1	152	144	0	0	0	0	0	0	...	314196	146948	297180	274392	247178	193972	320904
3	60	-1	0	-1	0	0	0	0	0	0	...	578	190	468	732	138	0	0
4	38938	-1	460	150	0	0	0	0	0	0	...	459428	220256	413674	334330	196244	92842	57548

<Test set from *Kaggle*>

	aa_000	ab_000	ac_000	ad_000	ae_000	af_000	ag_000	ag_001	ag_002	ag_003	...	ee_002	ee_003	ee_004	ee_005	ee_006	ee_007	ee_008
0	60	0	20	12	0	0	0	0	0	2682	...	1098	138	412	654	78	88	0
1	82	0	68	40	0	0	0	0	0	0	...	1068	276	1620	116	86	462	0
2	66002	2	212	112	0	0	0	0	0	199486	...	495076	380368	440134	269556	1315022	153680	516
3	59816	-1	1010	936	0	0	0	0	0	0	...	540820	243270	483302	485332	431376	210074	281662
4	1814	-1	156	140	0	0	0	0	0	0	...	7646	4144	18466	49782	3176	482	76

SMOTE를 이용한 OVERSAMPLING

minority class였던 pos가 neg와 같은 개수로 맞춰진 상태로 oversampling 되었다.

```
from imblearn.over_sampling import SMOTE

print("Before OverSampling, counts of label '1': {}".format(sum(y_train=="pos")))
print("Before OverSampling, counts of label '0': {} ₩n".format(sum(y_train=="neg")))

sm = SMOTE(random_state=2)
X_train_res, y_train_res = sm.fit_sample(X_train01, y_train)

print('After OverSampling, the shape of train_X: {}'.format(X_train_res.shape))
print('After OverSampling, the shape of train_y: {} ₩n'.format(y_train_res.shape))

print("After OverSampling, counts of label '1': {}".format(sum(y_train_res=="pos")))
print("After OverSampling, counts of label '0': {}".format(sum(y_train_res=="neg")))
```

```
Before OverSampling, counts of label '1': 853
Before OverSampling, counts of label '0': 44747

After OverSampling, the shape of train_X: (89494, 170)
After OverSampling, the shape of train_y: (89494,)

After OverSampling, counts of label '1': 44747
After OverSampling, counts of label '0': 44747
```

분석 모델 소개

1. random forest
2. Catboost
3. XGBoost
4. Logistic Regression
5. Softmax Regression
6. Linear SVC

분석 모델 1. random forest

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_jobs=4)
rf.fit(X_train_res, y_train_res)
```

```
RandomForestClassifier(n_jobs=4)
```

```
result = rf.predict(X_val01)
pd.Series(result).value_counts()
```

```
neg    11157
pos      243
dtype: int64
```

```
from sklearn.metrics import accuracy_score, f1_score
print('f1 score:', f1_score(y_val.replace({'neg': 0, 'pos' : 1}), pd.Series(result).replace({'neg': 0, 'pos' : 1})))
print('accuracy score:', accuracy_score(y_val.replace({'neg': 0, 'pos' : 1}), pd.Series(result).replace({'neg': 0, 'pos' : 1})))
```

```
f1 score: 0.7763157894736843
accuracy score: 0.9910526315789474
```

분석 모델 1. random forest

```
df1 = pd.DataFrame({'predicted':result, 'true':y_val})
df1.reset_index(drop=True,inplace=True)
```

```
i = 0
j= 0
false_neg = 0
false_pos = 0

for predicted, true in df1.values:
    if predicted != true: #예측이 틀렸을 때
        if true == 'neg':
            i = i+10
            false_pos = false_pos+1
        else :
            j = j+500
            false_neg = false_neg+1

print('RandomForest 총 비용 :','$',i+j)
print('positive를 negative로 분류 :', '$',false_neg*500)
print('negative를 positive로 분류 :', '$', false_pos*10)
```

분석 모델 2. Catboost

```
from catboost import CatBoostClassifier
cat = CatBoostClassifier()
cat.fit(X_train_res, y_train_res)
```

```
Learning rate set to 0.070202
0:	learn: 0.5707008	total: 82.7ms	remaining: 1m 22s
1:	learn: 0.4675095	total: 163ms	remaining: 1m 21s
2:	learn: 0.3939426	total: 239ms	remaining: 1m 19s
3:	learn: 0.3311132	total: 319ms	remaining: 1m 19s
4:	learn: 0.2797918	total: 393ms	remaining: 1m 18s
5:	learn: 0.2444688	total: 510ms	remaining: 1m 24s
6:	learn: 0.2152931	total: 594ms	remaining: 1m 24s
7:	learn: 0.1924672	total: 677ms	remaining: 1m 23s
8:	learn: 0.1743603	total: 747ms	remaining: 1m 22s
9:	learn: 0.1579811	total: 814ms	remaining: 1m 20s
10:	learn: 0.1451951	total: 890ms	remaining: 1m 20s
```

```
result2 = cat.predict(X_val01)
pd.Series(result2).value_counts()
```

```
neg    11159
pos      241
dtype: int64
```

분석 모델 3. XGBoost

```
import xgboost as xgb
xgb_model = xgb.XGBClassifier(objective="binary:logistic", random_state=42)
xgb_model.fit(X_train_res, y_train_res)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=42,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

```
result3 = xgb_model.predict(X_val01)
pd.Series(result3).value_counts()
```

```
neg    11175
pos      225
dtype: int64
```

분석 모델 4. Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score, log_loss, confusion_matrix,classification_report
logistic_clf= LogisticRegression(random_state=0, solver='lbfgs',C=1.0).fit(X_train_res, y_train_res)
```

```
result4= logistic_clf.predict(X_val01)
pd.Series(result4).value_counts()
```

```
neg    10730
pos      670
dtype: int64
```

분석 모델 5. Softmax Regression

```
softmax_reg = LogisticRegression(multi_class="multinomial", solver="lbfgs", C=10)
softmax_reg.fit(X_train_res, y_train_res)
```

```
LogisticRegression(C=10, multi_class='multinomial')
```

```
result5= softmax_reg.predict(X_val01)
pd.Series(result5).value_counts()
```

```
neg    10730
pos      670
dtype: int64
```

분석 모델 6. Linear SVC

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
```

```
Linear_svm_clf =Pipeline((
    ("scaler", StandardScaler()),
    ("linear_svc", LinearSVC(C=0.1, loss="hinge")),
))
Linear_svm_clf.fit(X_train_res, y_train_res)
```

```
Pipeline(steps=[('scaler', StandardScaler()),
                ('linear_svc', LinearSVC(C=0.1, loss='hinge'))])
```

```
result6=Linear_svm_clf.predict(X_val01)
pd.Series(result6).value_counts()
```

```
neg    10946
pos      454
dtype: int64
```

모델 결과 비교-Validation set

모델	비용	순위	False Positive	False Negative	F1 score
Random Forest	$19,170	6	67	37	0.772
CatBoost	$17,110	4	61	33	0.793
XGBoost	$17,460	5	46	34	0.817
Logistic Regression	$13,630	2	473	16	0.446
Softmax Regression	$13,240	1	474	17	0.444
LinearSVC	$13,630	2	473	16	0.59

모델 결과 비교-Test set

모델	비용	순위	False Positive	False Negative	F1 score
Random Forest	$12,810	3	31	25	0.926
CatBoost	$11,860	2	36	23	0.923
XGBoost	$10,680	1	18	21	0.948
Logistic Regression	$18,350	5	635	24	0.516
Softmax Regression	$22,980	6	648	33	0.501
LinearSVC	$15,310	4	331	24	0.664

XGBoost – Hyper parameter Tuning

Scale_pos_weight	1	10	20	50	52	52	70
Learning rate	0.25	0.2	0.2	0.2	0.2	0.07	0.2
Test set 비용	$11,180	$7,890	$7,370	$7,550	$7,050	$7,430	$7,610
False Positive	18	39	37	55	55	43	61
False Negative	22	15	14	14	13	14	14

XGBoost의 최종 파라미터

→ (learning_rate=0.2, subsample=0.8, objective="binary:logistic", scale_pos_weight=52, random_state=42)

Catboost – What is Catboost?

Catboost

-Yandex에 개발된 오픈 소스 Machine Learning

-Category와 Boosting을 합쳐서 만들어진 이름
Boost는 Gradient boosting machine learning algorithm에서 온 말이며, Gradient boosting을 기반으로 한다.

-구현하기가 쉬우며, 적은 데이터로도 좋은 결과를 얻을 수 있는 효율적인 방법이다.

Catboost – What is Catboost?

특징

1. Level-wise Tree

대칭 트리를 구현. 예측 시간을 줄이는 데 도움이 된다.

2. Ordered Boosting

기존의 부스팅 모델이 일괄적으로 모든 훈련 데이터를 대상으로 잔차 계산을 했다면,
Catboost는 일부만 가지고 잔차 계산을 한 뒤, 이걸로 모델을 만들고,
그 뒤에 데이터의 잔차는 이 모델로 예측한 값을 사용한다.

Catboost – What is Catboost?

3. Random Permutation

Ordered Boosting을 할 때, 데이터 순서를 섞어 주지 않으면 매번 같은 순서대로 잔차를 예측하는 모델을 만들 가능성 존재. Catboost는 이를 감안하며 데이터를 셔플링하여 뽑아낸다.

4. Categorical Feature Combinations

Information gain 이 동일한 두 특성 변수를 하나의 특성 변수로 묶어버림.
데이터 전처리에 있어 feature selection에 대한 부담을 조금 줄여준다.

Catboost – Hyper Parameter Tuning

Hyper Parameter:

-scale_pos_weight

Binary Classification에서 class1에 대한 weight

Imbalanced data에 대해, 보통 (sum_negative/sum_positive)

-learning rate

Catboost – Hyper Parameter Tuning

1. 대회의 목적은 총 비용을 줄이는 것

f1 score은 낮게 나오더라도 비용이 적게 나오는 hyperparameter 선택

2. false_negative를 하나라도 더 줄이는 것에 집중

false_negative의 페널티=500, false_positive의 페널티=10

false_positive 100개 더 많아지는 것 = false_negative 2개 많아지는 것

3. 단, validation set의 f1 score이 0.5 밑으로 떨어지지 않도록 방지

Catboost – Hyper Parameter Tuning

learning rate	0.1	0.1	0.15	0.2	0.2	0.3
scale_pos_weight	5	20	10	1	10	10
총 비용	8410	8090	6910	11810	7120	**5920**
false neg	12	5	7	23	7	6
false pos	241	559	341	31	362	292

scale_pos_weight: 10, learning_rate: 0.3 (기존 값: 0.07)

```
from catboost import CatBoostClassifier
cat2 = CatBoostClassifier(learning_rate=0.3, scale_pos_weight=10, verbose=True)
cat2.fit(X_train_res, y_train_res)
```

Catboost – Result

```
test_pred8 = cat2.predict(test1)
pd.Series(test_pred8).value_counts()
```

```
neg    15337
pos      663
dtype: int64
```

```
print('f1 score:', f1_score(test['class'].replace({'neg': 0, 'pos' : 1}), pd.Series(test_pred8).replace({'neg': 0, 'pos' : 1})))
print('accuracy score:', accuracy_score(test['class'].replace({'neg': 0, 'pos' : 1}), pd.Series(test_pred8).replace({'neg': 0, 'pos'
```

```
f1 score: 0.7090558766859344
accuracy score: 0.981125
```

```
t_df8 = pd.DataFrame({'predicted':test_pred8, 'true':test['class']})
t_df8.reset_index(drop=True,inplace=True)
```

Catboost – Result

```
i = 0
j= 0
false_neg = 0
false_pos = 0

for predicted, true in t_df8.values:
    if predicted != true: #예측이 틀렸을 때
        if true == 'neg':
            i = i+10
            false_pos = false_pos+1
        else :
            j = j+500
            false_neg = false_neg+1

print('CatBoost 총 비용 :','$',i+j)
print('positive를 negative로 분류 :',false_neg, '개', '$',false_neg*500)
print('negative를 positive로 분류 :',false_pos,'개', '$', false_pos*10)
```

```
CatBoost 총 비용 : $ 6450
positive를 negative로 분류 : 7 개 $ 3500
negative를 positive로 분류 : 295 개 $ 2950
```

Result – Comparison

Validation

Model	비용	순위	false p	false n	f1
RF	19640	5	38	64	0.7743
Cat	17100	3	33	60	0.7947
XGB	17460	4	34	46	0.8173
L Reg	13630	2	18	463	0.4477
SM Reg	13340	1	18	434	0.4631
SVC	13630	2	18	463	0.5898

Test

Model	비용	순위	false p	false n	f1
RF	11760	2	23	26	0.9349
Cat	11850	3	23	35	0.9238
XGB	10680	1	21	18	0.947
L Reg	20940	5	29	644	0.5069
SM Reg	23190	6	35	569	0.5295
SVC	15280	4	24	328	0.666

After Params Tuning

Validation

Model	비용	순위	false p	false n	f1
RF	19640	5	38	64	0.7743
Cat	9570	1	14	257	0.5949
XGB	12660	2	23	116	0.7321
L Reg	13630	4	18	463	0.4477
SM Reg	13340	3	18	434	0.4631
SVC	13630	4	18	463	0.5898

Test

Model	비용	순위	false p	false n	f1
RF	11760	3	23	26	0.9349
Cat	5920	1	6	292	0.7123
XGB	7050	2	13	55	0.9141
L Reg	20940	5	29	644	0.5069
SM Reg	23190	6	35	569	0.5295
SVC	15280	4	24	328	0.666

Final Result

```
print('accuracy score:', accuracy_score(final_test['class'].replace({'neg': 0, 'pos' : 1}), pd.Series(final_pred).replace({'neg': 0,
```

```
accuracy score: 0.9777894736842105
```

```
print(confusion_matrix(y_test, final_pred))
```

```
[[18301   390]
 [   32   277]]
```

```
i = 0
j= 0
false_neg = 0
false_pos = 0

for predicted, true in df.values:
    if predicted != true: #예측이 틀렸을 때
        if true == 'neg':
            i = i+10
            false_pos = false_pos+1
        else :
            j = j+500
            false_neg = false_neg+1

print('Catboost 총 비용 :','$',i+j)
print('positive를 negative로 분류 :', '$',false_neg*500)
print('negative를 positive로 분류 :', '$', false_pos*10)
```

```
Catboost 총 비용 : $ 19900
positive를 negative로 분류 : $ 16000
negative를 positive로 분류 : $ 3900
```

머신러닝 3조 최종 결과	
Accuracy Score	0.97779
Cost	$ 19900
Confusion Matrix	[[18301, 390] [32, 277]]

THANK YOU