KUBIG 머신러닝 분반 3조

김민혁 윤정현 윤지현 전보민

Index

1. Purpose of Analysis

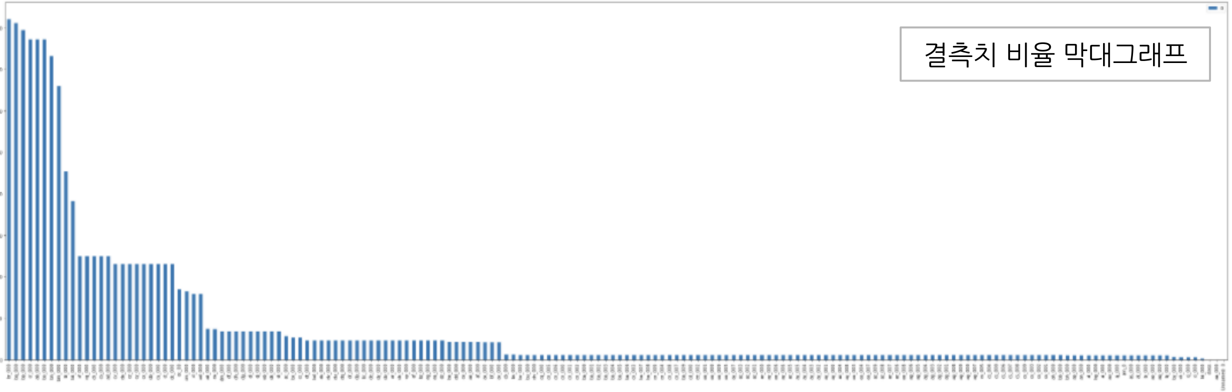
	Unnamed: 0	class	aa_000	ab_000	ac_000	ad_000	ae_000	af_000	ag_000	ag_001	...	ee_002	ee_003	ee_004	ee_005	ee_006	ee_007	
0	52803	neg	41386	NaN	508.0	488.0	0.0	0.0	0.0	0.0	...	438088.0	202172.0	383094.0	392838.0	228526.0	104226.0	1
1	38189	neg	29616	NaN	1616.0	1490.0	0.0	0.0	0.0	0.0	...	145524.0	72858.0	171332.0	308328.0	379466.0	213826.0	
2	23291	neg	241352	NaN	NaN	NaN	NaN	NaN	0.0	0.0	...	3617298.0	2477772.0	3631902.0	997462.0	436380.0	202002.0	1
3	16862	neg	8100	NaN	86.0	76.0	0.0	0.0	0.0	0.0	...	66980.0	36658.0	91898.0	86634.0	60276.0	23616.0	
4	14055	neg	2290	NaN	636.0	448.0	0.0	0.0	0.0	0.0	...	11542.0	7394.0	14206.0	69592.0	3108.0	108.0	

- 분석의 목적은 Scania Truck 부품별 데이터를 기반으로 정비 필요 여부를 예측하는 모형을 적합하는 것

- Loss = false positive * 10 + false negative * 500 를 최소화하는 것을 목적으로 한다.

2. Data Exploration

결측치 처리



	결측치 비율
br_000	82.059649
bq_000	81.121053
bp_000	79.456140
cr_000	77.171930
ab_000	77.171930
bo_000	77.161404
bn_000	73.222807
bm_000	65.884211

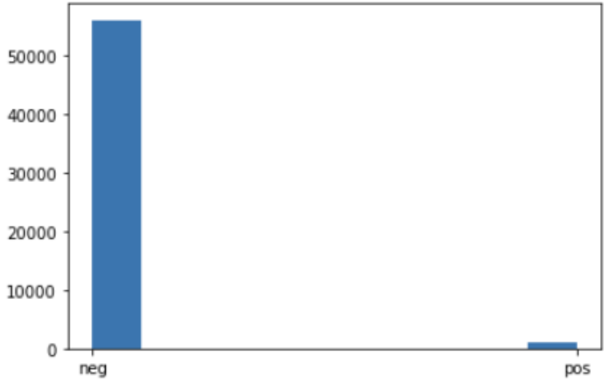
- 결측치 비율이 높은 열 존재
- 결측치 비율이 50%가 넘는 열은 불필요하다고 판단해 제거
- 나머지 결측치에 대해서는 평균값으로 대체

2. Data Exploration

class 열 확인 - 불균형 데이터 처리 필요성 인식

```
train['class'].value_counts()
```

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



3. Dimension Reduction (PCA)

Train / Validation set split

```
from sklearn.model_selection import train_test_split
```

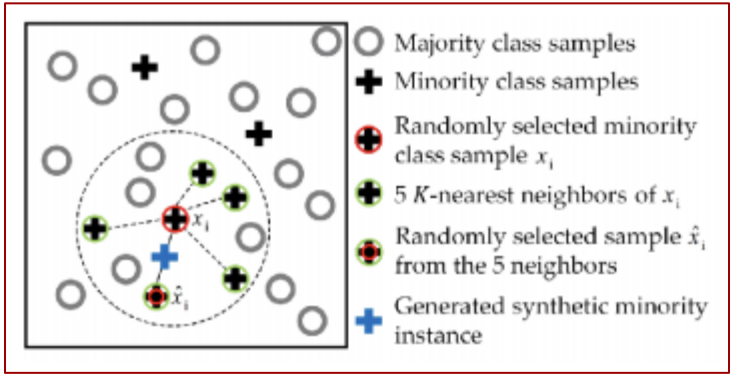
```
X_train, X_val, y_train, y_val = train_test_split(train_set, train_set['class'],
                                                   test_size = 0.2, random_state = 42,
                                                   stratify = train_set['class'])
```

- stratify parameter를 이용해 class열의 pos/neg 비율이 유지되도록 하면서 data split
- train data를 활용해 PCA, SMOTE 진행

3. Dimension Reduction (PCA)

Why?

불균형한 데이터 처리를 위해 사용할 SMOTE 기법



- KNN 알고리즘을 사용
- 차원의 저주 문제에 취약

PCA를 통해 차원을 축소한 뒤 SMOTE를 진행하기로 판단

3. Dimension Reduction (PCA)

Scaling

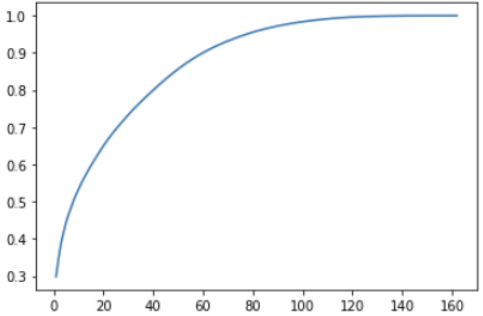
```
from sklearn.preprocessing import StandardScaler
# Fit on training set only.
scaler = StandardScaler()
scaler.fit(X_train)

# Apply transform to both the training set and the test set.
X_train_scaled = scaler.transform(X_train)
X_val_scaled = scaler.transform(X_val)
```

- PCA 진행 전, 변수에 따른 스케일로 인해 오류가 생기는 것 방지

3. Dimension Reduction (PCA)

PCA



> Explained_variance_ratio

```
pca = PCA(n_components=61)
pca.fit(X_train_scaled)
```

```
PCA(n_components=61)
```

```
print('explained variance ratio :', sum(pca.explained_variance_ratio_))
```

```
explained variance ratio : 0.903722412252041
```

```
pca_component = pca.fit_transform(X_train_scaled)
X_train_pca = pd.DataFrame(data=pca_component)
X_train_pca.head()
```

	0	1	2	3	4	5	6	7	8
0	-0.575221	1.134071	-1.072540	0.193362	0.386716	0.641388	-0.673025	0.013854	-0.396641
1	-0.290070	-0.357221	0.126070	0.519604	-0.166828	-0.012768	-0.327135	-0.020440	0.531793
2	-0.641508	-0.435298	-0.259103	-0.237481	0.471393	-0.212538	-0.966813	0.030608	-1.738606
3	-3.036226	0.584424	0.498985	0.258336	-0.391732	0.249493	0.191922	-0.019848	0.735217
4	-0.668264	-0.293205	-0.152759	0.332539	-0.045693	0.339489	0.114415	-0.026084	0.447056

5 rows × 61 columns

- 61개의 변수로 약 90%의 설명력을 가짐 ➡ 이 61개의 변수만으로 분석 진행하기로 결정

4. Oversampling (SMOTE)

SMOTE를 활용한 불균형 데이터 처리

```
from imblearn.over_sampling import SMOTE
```

```
print("Before OverSampling, '1': {}".format(sum(y_train=='pos')))
print("Before OverSampling, '0': {}".format(sum(y_train=='neg')))
```

```
Before OverSampling, '1': 853
Before OverSampling, '0': 44747
```

```
smote = SMOTE(random_state = 42)
X_train_over, y_train_over = smote.fit_resample(X_train_pca, y_train)

print('After OverSampling, shape of X: {}'.format(X_train_over.shape))
print('After OverSampling, shape of y: {} \n'.format(y_train_over.shape))

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

```
After OverSampling, shape of X: (89494, 61)
After OverSampling, shape of y: (89494,)

After OverSampling, '1': 44747
After OverSampling, '0': 44747
```

pos의 개수가 neg와 같은 개수로

oversampling된 것을 확인

5. Modeling

Loss function 정의

“false positive * 10 + false negative * 500”

```
import numpy as np

def truckloss(y_true, y_pred):

    FP = np.logical_and(y_true == 'neg', y_pred == 'pos')
    FN = np.logical_and(y_true == 'pos', y_pred == 'neg')


    FPsum = sum(FP)
    FNsum = sum(FN)

    return FPsum*10 + FNsum*500
```

우리의 목표는 이 Loss값을 최소로 만드는 모델을 찾는 것!

5. Modeling

Softmax Regression

- Predict the class with the highest estimated probability

```
from sklearn.linear_model import LogisticRegression
softmax_reg = LogisticRegression(multi_class="multinomial", solver="lbfgs", C=10)
softmax_reg.fit(X_train_over, y_train_over)
```

...

```
y_pred_softmax = softmax_reg.predict(X_val_pca)
```

```
print('f1 score:', f1_score(y_val.replace({'neg': 0, 'pos' : 1}), pd.Series(y_pred_softmax).replace({'neg': 0, 'pos' : 1})))
print('accuracy score:', accuracy_score(y_val.replace({'neg': 0, 'pos' : 1}), pd.Series(y_pred_softmax).replace({'neg': 0, 'pos' : 1}))
```

```
f1 score: 0.5689404934687954
accuracy score: 0.9739473684210527
```

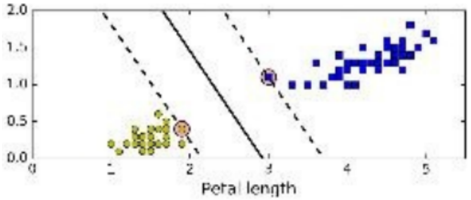
```
truckloss(y_val, y_pred_softmax)
```

```
11300
```

5. Modeling

Support Vector Machine

- capable of performing linear/ nonlinear classification, regression, even outlier detection
- Fitting the widest possible street between the classes




```
from sklearn.pipeline import Pipeline
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
```

```
Linear_svm_clf =Pipeline((
    ("linear_svc", LinearSVC(C=0.1, loss="hinge")),
))
Linear_svm_clf.fit(X_train_over, y_train_over)
```

...

```
y_pred_svc = Linear_svm_clf.predict(X_val_pca)
```

```
print('f1 score:', f1_score(y_val.replace({'neg': 0, 'pos' : 1}), pd.Series(y_pred_svc).replace({'neg': 0, 'pos' : 1})))
print('accuracy score:', accuracy_score(y_val.replace({'neg': 0, 'pos' : 1}), pd.Series(y_pred_svc).replace({'neg': 0, 'pos' : 1})))
```

```
f1 score: 0.5145888594164456
accuracy score: 0.9678947368421053
```

```
truckloss(y_val, y_pred_svc)
```

```
12970
```

5. Modeling

RandomForest

- Ensemble of Decision Trees

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_jobs=-1, random_state=42)
rf.fit(X_train_over, y_train_over)
```

```
RandomForestClassifier(n_jobs=-1, random_state=42)
```

```
y_pred_rf = rf.predict(X_val_pca)
```

```
from sklearn.metrics import accuracy_score, f1_score

print('f1 score:', f1_score(y_val.replace({'neg': 0, 'pos' : 1}), pd.Series(y_pred_rf).replace({'neg': 0, 'pos' : 1})))
print('accuracy score:', accuracy_score(y_val.replace({'neg': 0, 'pos' : 1}), pd.Series(y_pred_rf).replace({'neg': 0, 'pos' : 1})))
```

```
f1 score: 0.6691176470588235
accuracy score: 0.9842105263157894
```

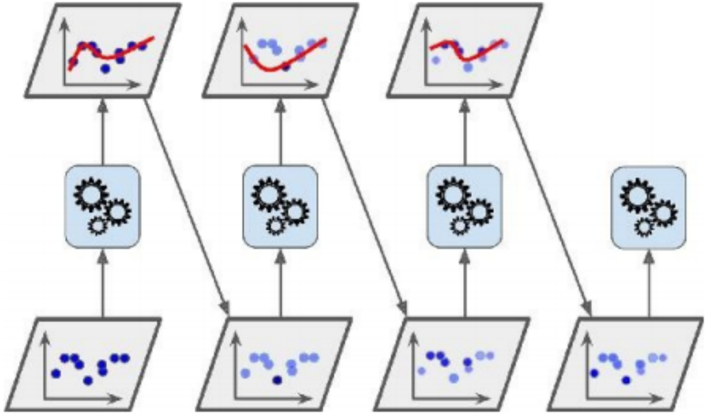
```
truckloss(y_val, y_pred_rf)
```

```
16990
```

5. Modeling

AdaBoost

- Boosting method (combine several weak learners into a strong learner)
- Train predictors sequentially, each trying to correct its predecessor



5. Modeling

AdaBoost

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier

ada_clf = AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=1), n_estimators=200,
    algorithm="SAMME.R", learning_rate=0.5, random_state=42)
ada_clf.fit(X_train_over, y_train_over)
```

```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=1),
                   learning_rate=0.5, n_estimators=200, random_state=42)
```

```
y_pred_ada = ada_clf.predict(X_val_pca)
```

```
print('f1 score:', f1_score(y_val.replace({'neg': 0, 'pos' : 1}), pd.Series(y_pred_ada).replace({'neg': 0, 'pos' : 1})))
print('accuracy score:', accuracy_score(y_val.replace({'neg': 0, 'pos' : 1}), pd.Series(y_pred_ada).replace({'neg': 0, 'pos' : 1})))
```

```
f1 score: 0.44866071428571425
accuracy score: 0.9566666666666667
```

```
truckloss(y_val, y_pred_ada)
```

```
10820
```

이 모델로 결정!

6. Result

Test data에 적용

```
class_pred_ada_df['class'].value_counts()
```

```
neg    17923
pos     1077
Name: class, dtype: int64
```

<Test Score>

loss	20940
accuracy	95.684

Thank you