데이터 분석을 통한 트럭 수리 결정 판단 및 비용 최소화 작업

: Scania 데이터의 활용

조별 발표자료(2020.10.08)

머신러닝 4조
유재형 황연재 성유지 박시전 김세진 이나윤

목 차

1 데이터 탐색 및 전처리

2 전체 변수들로 모형 구축해보기

3 주요 변수들로만 모형 구축해보기

4 앙상블: 투표 분류기 활용

5 결론 및 분석

1 데이터 탐색 및 전처리

(1) 데이터셋 불러오기, 모양새 확인

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt

data = pd.read_csv("D:/DATA FROM SONY/Korea Univ/2018-1~/2020/2학기/강의 외/학회/조별 대회/Train_data
data.head()
```

	class	aa_000	ab_000	ac_000	ad_000	ae_000	af_000	ag_000	ag_001	ag_002	ag_003	ag_004	ag_0
52803	neg	41386	NaN	508.0	488.0	0.0	0.0	0.0	0.0	0.0	0.0	51396.0	88646
38189	neg	29616	NaN	1616.0	1490.0	0.0	0.0	0.0	0.0	0.0	0.0	452.0	4262(
23291	neg	241352	NaN	NaN	NaN	NaN	NaN	0.0	0.0	0.0	10140.0	639334.0	925933(
16862	neg	8100	NaN	86.0	76.0	0.0	0.0	0.0	0.0	0.0	0.0	112.0	6689(
14055	neg	2290	NaN	636.0	448.0	0.0	0.0	0.0	0.0	0.0	0.0	354.0	2732(

5 rows × 171 columns

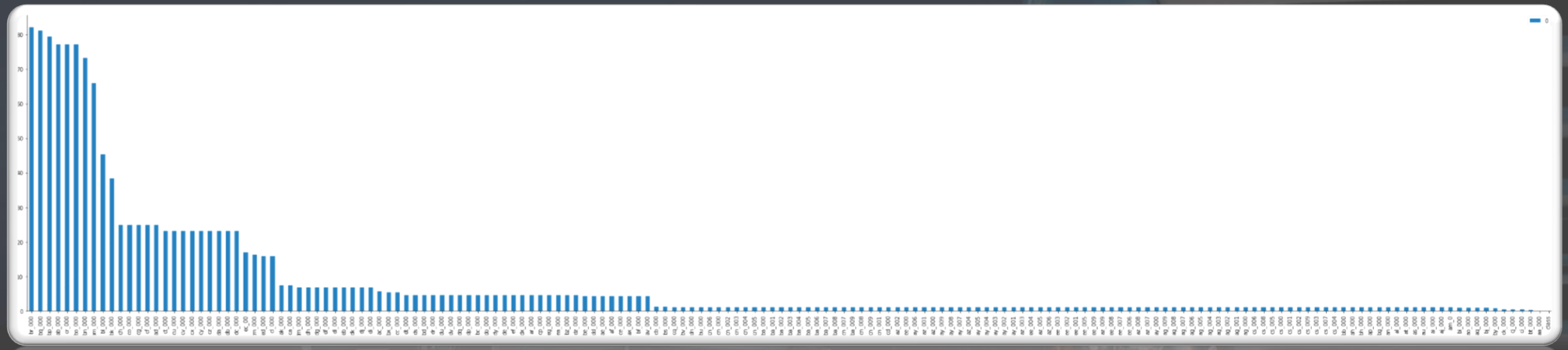
```
his_cols = [col for col in data if '001' in col]
his_cols = [i[:2] for i in his_cols]

for i in his_cols:
  i_k = [col for col in data if i in col]
  print(i_k)
```

```
['ag_000', 'ag_001', 'ag_002', 'ag_003', 'ag_004', 'ag_005', 'ag_006', 'ag_007', 'ag_008', 'ag_009']
['ay_000', 'ay_001', 'ay_002', 'ay_003', 'ay_004', 'ay_005', 'ay_006', 'ay_007', 'ay_008', 'ay_009']
['az_000', 'az_001', 'az_002', 'az_003', 'az_004', 'az_005', 'az_006', 'az_007', 'az_008', 'az_009']
['ba_000', 'ba_001', 'ba_002', 'ba_003', 'ba_004', 'ba_005', 'ba_006', 'ba_007', 'ba_008', 'ba_009']
['cn_000', 'cn_001', 'cn_002', 'cn_003', 'cn_004', 'cn_005', 'cn_006', 'cn_007', 'cn_008', 'cn_009']
['cs_000', 'cs_001', 'cs_002', 'cs_003', 'cs_004', 'cs_005', 'cs_006', 'cs_007', 'cs_008', 'cs_009']
['ee_000', 'ee_001', 'ee_002', 'ee_003', 'ee_004', 'ee_005', 'ee_006', 'ee_007', 'ee_008', 'ee_009']
```

- ✓ Size: 57,000(rows)*171(columns)
- ✓ **결측치의 양이 상당함**과 동시에, **Histogram Variable**이 무엇인지 드러나 있지 않아 **찾아낼 필요가 있었음.**
- ✓ 분석 결과 **ag, ay, az, ba, cn, cs, ee**가 포함된 column들이 **histogram variable**에 해당, 나머지 column은 numerical variable에 해당하는 데이터임을 확인

1 데이터 탐색 및 전처리

(2) 결측치 처리하기: 일부 column 제거하기



✓ 일부 column들의 경우 좌측 그림과 같이 결측치를 지나치게 많이 포함하고 있음을 확인

✓ 따라서 해당하는 column들의 경우 오히려 분석에 악영향을 줄 수 있겠다고 판단, 논의 끝 결측치의 비율이 23%가 넘어가는 column은 모두 삭제하였음.

✓ 총 24개의 column 삭제

1 데이터 탐색 및 전처리

(2) 결측치 처리하기: 잔여 결측치를 계급별 중앙값으로 대체

```
truck_pos = train3[train3["class"] == "pos"]
truck_neg = train3[train3["class"] == "neg"]

truck_pos = truck_pos.fillna(truck_pos.median())
truck_neg = truck_neg.fillna(truck_neg.median())

train3 = pd.concat([truck_pos,truck_neg])
print(train3.shape)
train3.head()
#truck3: after filling Na using media
```

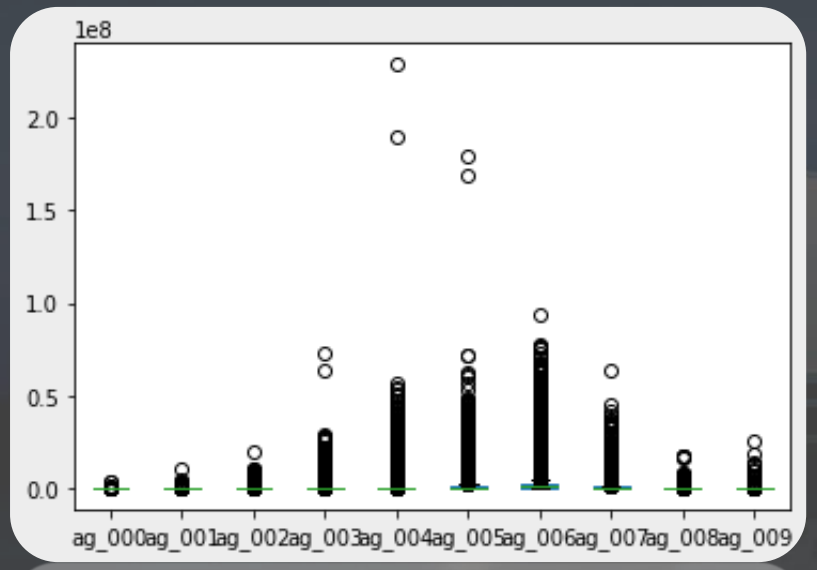
(56795, 147)

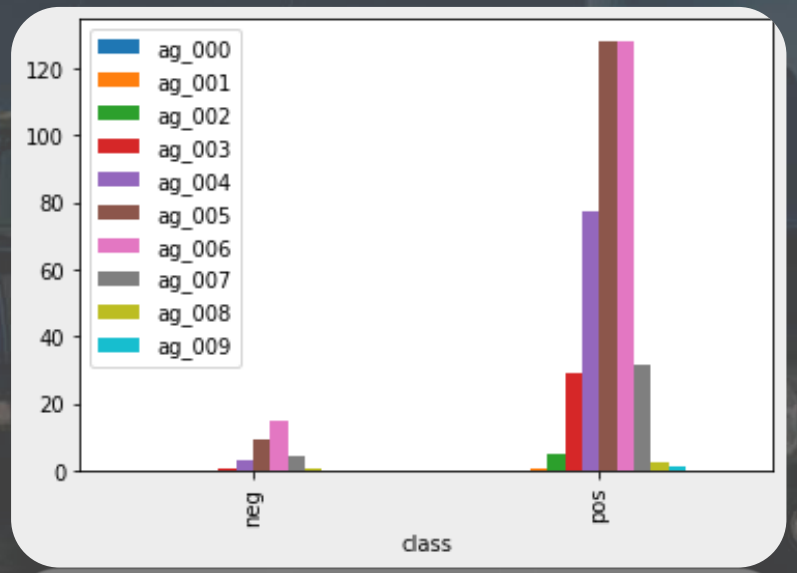
	class	aa_000	ac_000	ae_000	af_000	ah_000	ai_000	aj_000	ak_000
34	pos	495138	790.0	0.0	0.0	17925428.0	120834.0	0.0	0.0
146	pos	872812	0.0	0.0	0.0	29118480.0	26676.0	50.0	0.0
281	pos	372606	5142.0	0.0	0.0	10290410.0	0.0	0.0	0.0
292	pos	781020	790.0	0.0	0.0	17124994.0	17504.0	62.0	0.0
688	pos	124642	760.0	0.0	0.0	6508464.0	0.0	0.0	0.0

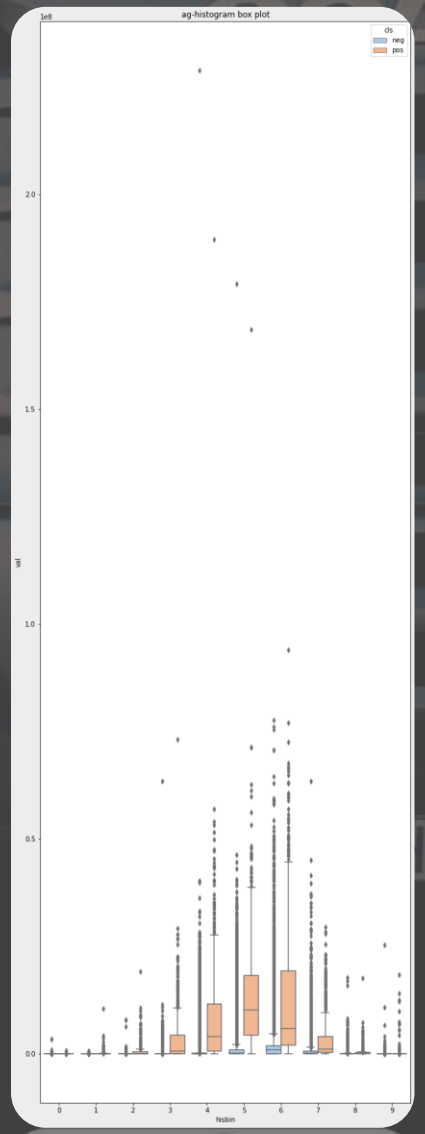
- ✓ 공지된 바와 같이 'negative' class와 'positive' class 간의 size 차이가 컸기 때문에, 우선 두 class를 분리시켰다.
- ✓ 이후 각 class의 column별로 median값을 찾은 후, 해당 값으로 column별 결측치를 메웠다.
- ✓ 평균, 최빈값, knn-imputing 등 여러 대안을 시도해보았으나, 최종적으로 모형을 돌렸을 때 결과가 median이 가장 좋아 이 것을 최종 대안으로 선택하였다.

1 데이터 탐색 및 전처리

(3) Histogram 변수에 대한 Scaling 실시 : 'ag' column을 예시로







분포가 고르지 못하고 Outlier의 값이 지나치게 많아 시각화가 어렵고, Scaling이 필요함을 확인: 이에 루트변환 등 여러가지 시도를 해본 결과, 로그변환이 가장 적절하다는 결론을 내림

1 데이터 탐색 및 전처리

(3) Histogram 변수에 대한 Scaling 실시 : 'ag' column을 예시로

```
train_ag2 = train_ag.replace(0, 1)
ag_log2 = train_ag2.copy()
ag_log2 = ag_log2.iloc[:,:-1].apply(np.log)
ag_log2["class"] = train_ag["class"]
ag_log2
```

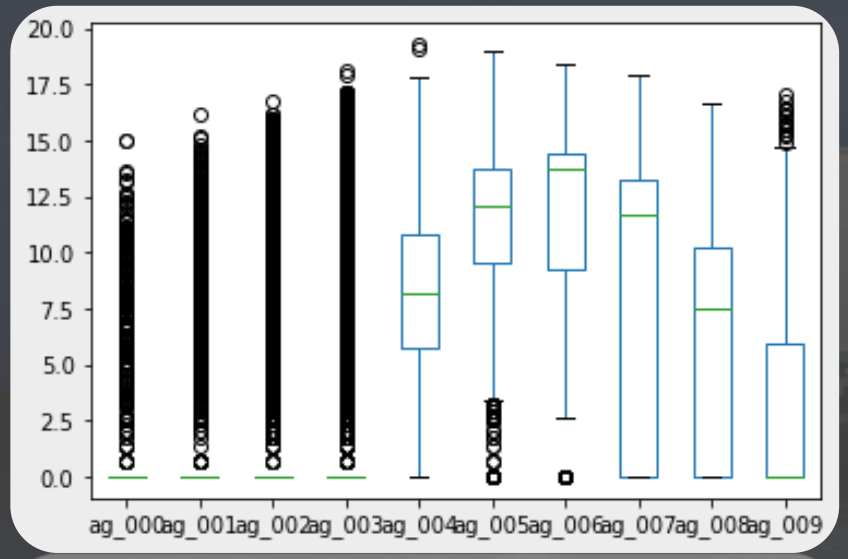
	ag_000	ag_001	ag_002	ag_003	ag_004	ag_005	ag_006	ag_007	ag_008	ag_009	class
0	0.0	0.0	0.0	0.000000	10.847316	13.694996	14.184294	13.046613	10.531029	5.662960	neg
1	0.0	0.0	0.0	0.000000	6.113682	10.660079	13.946497	13.295086	10.662469	7.212294	neg
2	0.0	0.0	0.0	9.224243	13.368182	16.041143	15.782481	13.425149	9.252633	4.736198	neg
3	0.0	0.0	0.0	0.000000	4.718499	11.110924	12.899600	11.105589	8.302018	0.000000	neg
4	0.0	0.0	0.0	0.000000	5.869297	10.215374	11.253533	10.360343	0.000000	0.000000	neg
...	...	...	...	...	...	...	...	...	...	...	...
56995	0.0	0.0	0.0	0.000000	5.472271	10.798391	11.249324	7.346010	0.000000	0.000000	neg
56996	0.0	0.0	0.0	0.000000	5.181784	5.327876	8.245384	0.000000	0.000000	0.000000	neg
56997	0.0	0.0	0.0	10.586130	13.926363	14.958908	14.091624	12.573566	11.246509	6.641182	pos
56998	0.0	0.0	0.0	0.000000	5.062595	11.857423	14.395178	13.256534	10.544815	6.830874	neg
56999	0.0	0.0	0.0	0.000000	7.615791	12.143207	14.780786	14.006653	11.319292	7.912789	neg

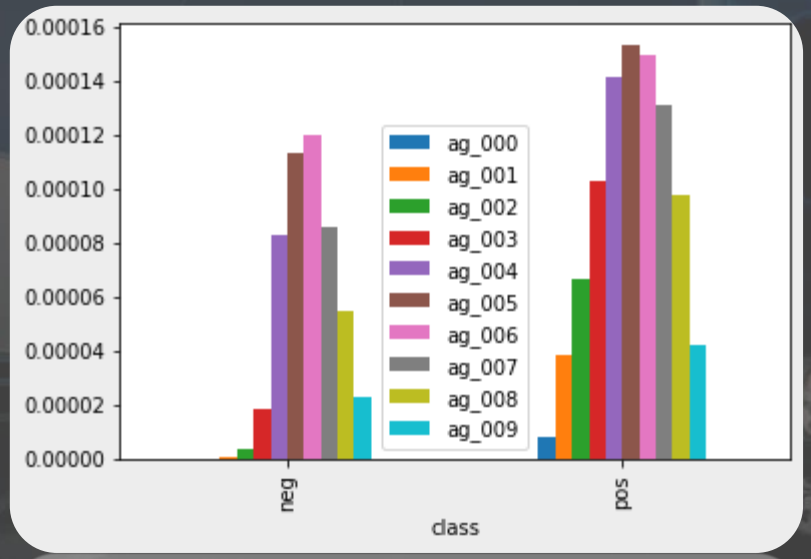
57000 rows × 11 columns

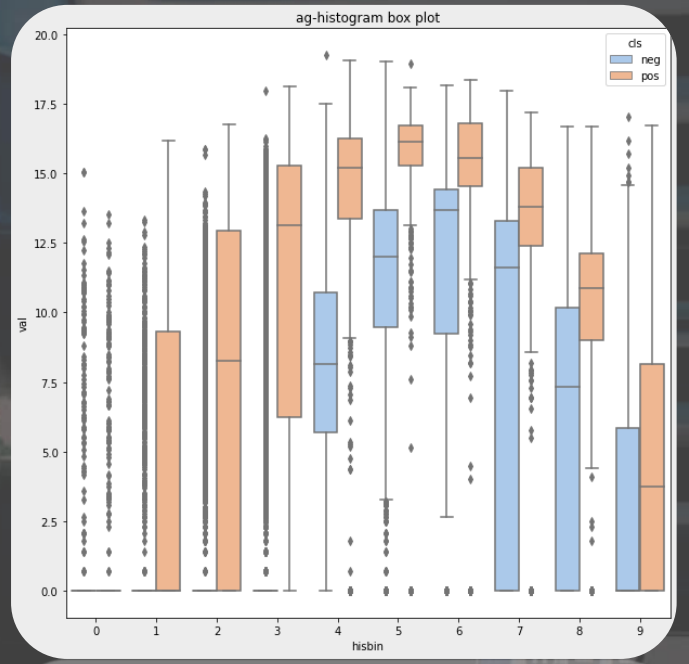
- ✓ 로그 변환을 위해 0에 해당되는 값에 모두 1을 대입함
  >> train_ag.replace(0,1)
- ✓ 로그변환 실시
  >> .apply(np.log)

1 데이터 탐색 및 전처리

(3) Histogram 변수에 대한 Scaling 실시 : 'ag' column을 예시로



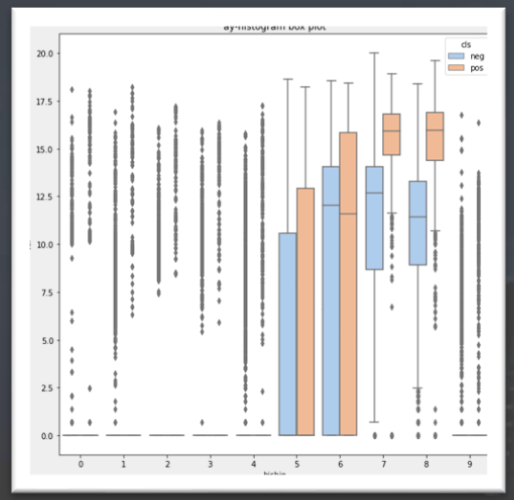
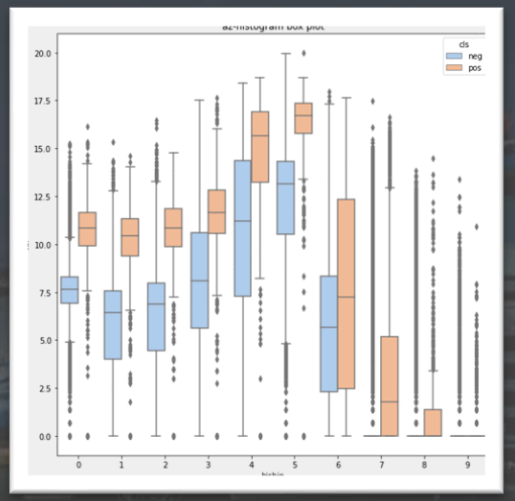
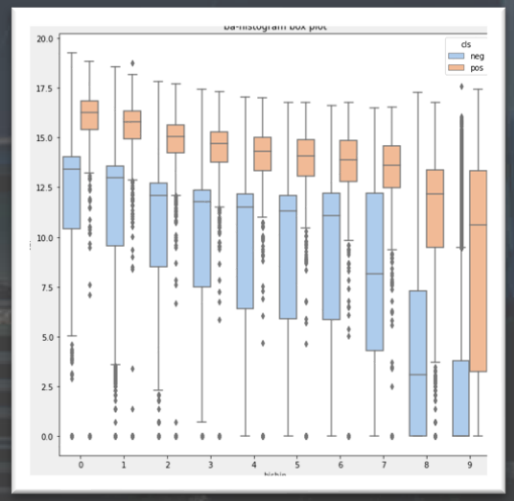
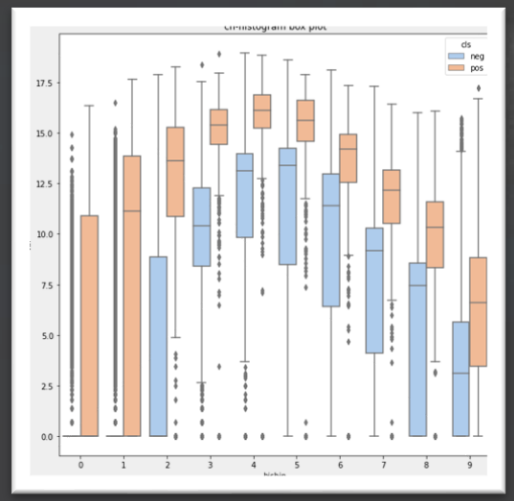
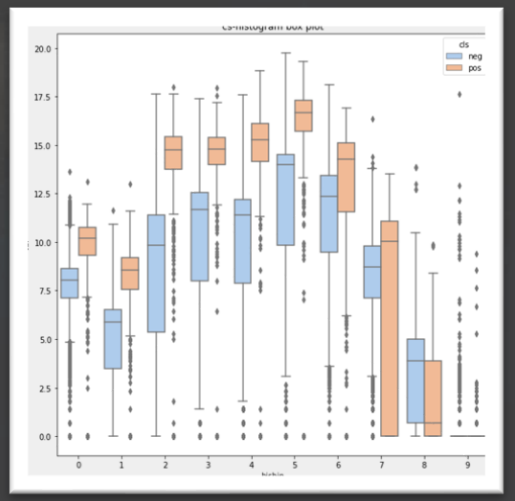
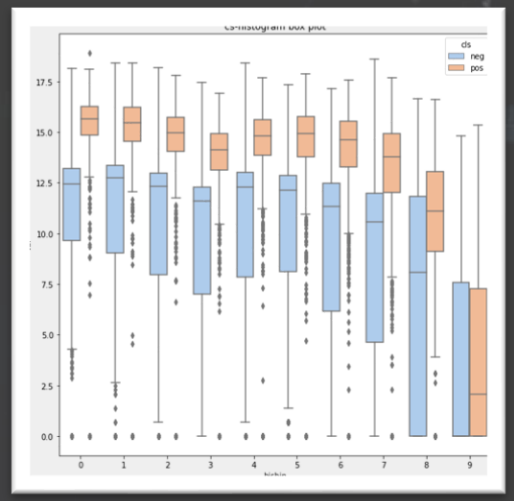




시각화 결과 보다 분포가 고르게 바뀌었음을 확인할 수 있음.

1 데이터 탐색 및 전처리

(3) Histogram 변수에 대한 Scaling 실시 : 나머지 column들

- ✓ 동일한 방법으로 나머지 histogram variable에도 로그변환 실시
- ✓ 1열 좌->우 순으로 각각 ay, az, ba column들에 대한 로그변환 실시 결과
- ✓ 2열 좌->우 순으로 각각 cn, cs, ee column들에 대한 로그변환 실시 결과

1 데이터 탐색 및 전처리

(4) 비용 측정 함수 만들기

```
from sklearn.metrics.scorer import make_scorer
from sklearn.metrics import confusion_matrix

def my_scorer(y_true,y_pred):
    tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()
    cost = 10*fp+500*fn
    return cost

my_func = make_scorer(my_scorer, greater_is_better=False)
```

- ✓ 대회 규정 상 false negative에는 $500의 페널티가, false positive에는 $10의 페널티가 붙음.
- ✓ 위양성과 위음성에 대해 같은 비용이 부과되지 않으므로, accuracy score를 측정하는 것만으로는 모형의 정확한 비용을 직관적으로 알기 어려움
- ✓ 이에 근거해, 자동으로 최종 비용을 산출해주는 함수를 만들었음: 직관적인 결과 파악 가능케 함.

2 전체 변수들로 모형 구축해보기

(1) Oversampling

```
[ ]  from sklearn.feature_selection import SelectKBest, chi2, f_classif, mutual_info_classif
     from sklearn.model_selection import train_test_split

     X = truck4.drop("class", axis = 1)
     y = truck4["class"]

     X_nd = truck4.drop("class", axis = 1).values
     y_nd = truck4["class"].values

     X_train, X_validTest, y_train, y_validTest = train_test_split(X_nd, y_nd, train_size=0.5,
     X_valid, X_test, y_valid, y_test = train_test_split(X_validTest, y_validTest, train_size =


[ ]  X_nd.shape

     (57000, 209)
```

Oversampling

```
[ ]  from collections import Counter
     from sklearn.datasets import make_classification
     from imblearn.over_sampling import SMOTE, ADASYN

     sm = ADASYN(random_state=42)
     X_res, y_res = sm.fit_resample(X_train, y_train)
     print('Resampled dataset shape %s' % Counter(y_res))
```

- ✓ 앞서 언급한 바와 같이 'positive' class와 'negative' class 간의 size 차이가 컸기에, 'positive' class에 대한 과대표집을 시도하였다.
- ✓ 먼저 sklearn의 train_test_split 함수를 통해 train/validation/test set을 각각 분할한 후(비율 5:3:2), train set을 ADASYN 패키지를 이용해 다시 한 번 resampling 하였다.
- ✓ 이후 모든 full-featured 모형은 이 과대표집된 train set으로 fit하였다.

2 전체 변수들로 모형 구축해보기

(2) CatBoost Using Full Features

```
from catboost import CatBoostClassifier, Pool
from sklearn.model_selection import train_test_split

cat_features = np.where(X.dtypes.astype("str").isin(["category","object"]))[0]

CatBoost = CatBoostClassifier(random_seed = 1234)

CatBoost.fit(X = X_res, y = y_res, cat_features = cat_features, eval_set = (X_valid, y_valid))
```

```
CatBoost.score(X_test, y_test)
```

```
0.9955263157894737
```

```
[ ]  y_pred_cat = CatBoost.predict(X_test)
```

```
print(my_scorer(y_test, y_pred_cat)/len(y_test)) # 0.68
pd.crosstab(y_test, y_pred_cat, rownames=['True'], colnames=['Predicted'], margins=True)
```

```
0.6894736842105263
```

Predicted	neg	pos	All
True			
neg	11142	36	11178
pos	15	207	222
All	11157	243	11400

- ✓ '통계적 머신러닝' 강의에서 강조되었던, 일종의 '블랙박스'형 머신러닝 툴
- ✓ 비교적 최근에 등장한, 범주형 변수 분류에 특히 강점을 보이는 모형
- ✓ "일단 이것부터 써라" – 박유성 교수님

=> 비용: 15*500 + 36*10 = $7,860

2 전체 변수들로 모형 구축해보기

(3) RandomForest Using Full Features

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators= 100)

model.fit(X = X_res, y = y_res)

print(model.score(X_res, y_res))
print(model.score(X_test, y_test))

1.0
0.9954385964912281


y_pred_rf = model.predict(X_test)
print(my_scorer(y_test, y_pred_rf)/len(y_test)) # 0.90
pd.crosstab(y_test, y_pred_rf, rownames=['True'], colnames=['Predicted'], margins=True)

1.0771929824561404
```

Predicted	neg	pos	All
True			
neg	11150	28	11178
pos	24	198	222
All	11174	226	11400

- ✓ 훈련을 통해 의사결정 나무들을 만들고, 그 나무들의 분류 결과를 취합하여 최종적으로 분류하는 머신러닝 모델
- ✓ 다수의 나무들로부터 분류를 집계하기 때문에 오버피팅이 나타나는 나무의 영향력을 줄일 수 있음.

=> 비용: 24*500 + 28*10 = $12,280

3 주요 변수들로만 모형 구축해보기

(1) Feature Selection: Establishing Selected Set

```
print(set(cat_features).intersection(set(rf_features)))
print(set(cat_features).intersection(set(chi2_features)))
print(set(rf_features).intersection(set(chi2_features)))
print(set(rf_features).intersection(set(chi2_features)).intersection(set(cat_features)))
print(set(rf_features).union(set(chi2_features)).union(set(cat_features)))

print("-----------------------------------------------------------------------------------------------------------")

feature1 = list(set(rf_features).union(set(chi2_features)).union(set(cat_features))) #세개의 합집합
feature2 = list(set(rf_features).union(set(cat_features))) # cat 과 rf 의 합집합
feature3 = list(set(rf_features).union(set(cat_features).intersection(set(chi2_features)))) # rf + cat & chi2 의 교집합
feature4 = list(set(cat_features).union(set(rf_features).intersection(set(chi2_features)))) # cat + rf & chi2 의 교집합

print(len(feature1))
print(len(feature2))
print(len(feature3))
print(len(feature4))

print("-----------------------------------------------------------------------------------------------------------")
print(feature1) #54
print(feature2) #46
print(feature3) #31
print(feature4) #35
```

- ✓ 개별 모형별로 뽑아낸 중요 feature들을 토대로, 몇 가지 경우의 수를 고려하여 새로운 feature set들을 만들었음.

- ✓ 총 4개의 feature set 생성

  - Set #1: rf&chi2&cat 합집합
  - Set #2: cat과 rf의 합집합
  - Set #3: rf + cat&chi2의 교집합
  - Set #4: cat + rf&chi2의 교집합

3 주요 변수들로만 모형 구축해보기

(2) Oversampling Revisited

```
from collections import Counter
from sklearn.datasets import make_classification
from imblearn.over_sampling import SMOTE, ADASYN

sm = ADASYN(random_state=42)
X_res_f1, y_res_f1 = sm.fit_resample(X_train_f1, y_train)
X_res_f2, y_res_f2 = sm.fit_resample(X_train_f2, y_train)
X_res_f3, y_res_f3 = sm.fit_resample(X_train_f3, y_train)
X_res_f4, y_res_f4 = sm.fit_resample(X_train_f4, y_train)


print('Resampled dataset feature1 %s' % Counter(y_res))
print('Resampled dataset feature2 %s' % Counter(y_res))
print('Resampled dataset feature3 %s' % Counter(y_res))
print('Resampled dataset feature4 %s' % Counter(y_res))
# Oversampled dataset
```

- ✓ 이후, 각 feature set들을 토대로 재차 resampling을 실시함
- ✓ 이후 모든 모형들에 대해, feature set을 기반으로 한 과대표집된 sample들로 fitting을 시도, 결과를 비교해보았음.

3 주요 변수들로만 모형 구축해보기

(3) CatBoost Using Selected Features

```
y_pred_f1 = CatBoost.predict(X_test_f1)
print(my_scorer(y_test, y_pred_f1)/len(y_test)) # 0.78
pd.crosstab(y_test, y_pred_f1, rownames=['True'], colnames=['Predicted'], margins=True)
```

```
y_pred_f2 = CatBoost.predict(X_test_f2)
print(my_scorer(y_test, y_pred_f2)/len(y_test)) # 0.69
pd.crosstab(y_test, y_pred_f2, rownames=['True'], colnames=['Predicted'], margins=True)
```

```
y_pred_f4 = CatBoost_f4.predict(X_test_f4)
print(my_scorer(y_test, y_pred_f4)/len(y_test)) # 0.56
pd.crosstab(y_test, y_pred_f4, rownames=['True'], colnames=['Predicted'], margins=True)
```

✓ feature set #1과 #2의 경우 full model을 사용할 때에 비해 cost가 좋지 않게 나옴.

✓ **그러나 Set #4 의 경우 full model보다 적은 feature 수로 cost를 더욱 낮출 수 있었음!**

3 주요 변수들로만 모형 구축해보기

(4) RandomForest Using Selected Features

Feature Set #1

Predicted / True	neg	pos	All
neg	11146	32	11178
pos	21	201	222
All	11167	233	11400

Feature Set #2

Predicted / True	neg	pos	All
neg	11142	36	11178
pos	20	202	222
All	11162	238	11400

Feature Set #3

Predicted / True	neg	pos	All
neg	11121	57	11178
pos	27	195	222
All	11148	252	11400

✓ feature set #3의 경우 full model을 사용할 때에 비해 cost가 좋지 않게 나옴.

✓ 그러나 Set #1, 2 의 경우 full model보다 적은 feature 수로 cost를 더욱 낮출 수 있었음!
>> 특히 Set #2가 20*500 + 36*10 = $10,360 으로 가장 적은 비용을 보였음.

3 주요 변수들로만 모형 구축해보기

(5) XGBoost Using Selected Features

Predicted neg pos All

Feature Set #1

True

neg	11062	116	11178
pos	7	215	222
All	11069	331	11400

Predicted neg pos All

Feature Set #2

True

neg	11058	120	11178
pos	8	214	222
All	11066	334	11400

- ✓ Gradient Boosting 기법에 기반
- ✓ 과적합 규제, 교차 검증, 피처 중요도 등의 기능을 가지고 있어 각광받고 있는 알고리즘
- ✓ Set #2보다는 #1에서 더욱 낮은 비용이 나왔음; 그러나 Set #2가 #1에 비해 원소의 개수가 훨씬 적기 때문에, "Simple is Best"에 입각해 Set #2 선택함.

=> 비용: 8*500 + 120*10 = $5,200

3 주요 변수들로만 모형 구축해보기

(6) K-Nearest Neighborhoods(KNN)

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 5)
knn.fit(X_res_f2_std, y_res_f2)

y_pred_knn = knn.predict(X_test_f2_std)
print(my_scorer(y_test, y_pred_knn)/len(y_test))
pd.crosstab(y_test, y_pred_knn, rownames = ["True"], coln
```

```
1.0578947368421052
```

Predicted	neg	pos	All
True			
neg	11022	156	11178
pos	21	201	222
All	11043	357	11400

- ✓ 특정 공간 내에서 입력과 제일 근접한 k 개의 요소를 찾아, 더 많이 일치하는 것으로 분류하는 알고리즘
  >> 우리 팀은 k=5를 사용함.

- ✓ KNN의 경우 앞의 세 모형에 비해 그 중요도가 떨어진다고 판단하여 한 개의 Feature Set에 대해서만 모형을 돌려보았음. (+ 시간 부족 ^^;)

=> 비용: 21*500+156*10 = $12,060

3 주요 변수들로만 모형 구축해보기

(7) Logistic Regression

```
from sklearn.linear_model import LogisticRegression, Ridge, Lasso
LogisticRegression=LogisticRegression(random_state=1234)
LogisticRegression.fit(X_res_f1,y_res_f1)

y_pred_l = LogisticRegression.predict(X_test_f1)
print(my_scorer(y_test, y_pred_l)/len(y_test))
pd.crosstab(y_test, y_pred_l, rownames = ["True"], colnames = ["Pr
```

Predicted	neg	pos	All
True			
neg	7634	3544	11178
pos	7	215	222
All	7641	3759	11400

- ✓ KNN에서와 마찬가지 이유로, 한 개의 Feature Set에 대해서만 모형을 fit함 >> 앞에서 Set #2가 가장 많이 다뤄졌으므로 여기서도 동일한 Set을 활용
- ✓ 타 모형에 비해 false negative의 값은 적으나 false positive의 값이 매우 크다는 특징이 있음.

=> 비용: 7*500+3544*10 = $38,940

4 앙상블: 투표 분류기 활용

서로 다른 feature set을 활용한 앙상블(hard/soft)

```
pipe1 = Pipeline([
    ('col_extract', ColumnExtractor( cols = feature4_num)),
    ('cat', CatBoost_f4)
    ])

pipe2 = Pipeline([
    ('col_extract', ColumnExtractor( cols= feature1_num)),
    ('rf', model_f1)
    ])


pipe3 = Pipeline([
    ('col_extract', ColumnExtractor( cols= feature2_num)),
    ('xgb', xgb_model_f2)
    ])

pipe4 = Pipeline([
    ('col_extract', ColumnExtractor( cols= feature1_num)),
    ('logistic', LogisticRegression)
    ])

pipe5 = Pipeline([
    ('col_extract', ColumnExtractor( cols= feature2_num)),
    ('scale', StandardScaler()),
    ('knn', knn)
    ])
```

- ✓ 모델별로 가장 성능이 좋았던 변수 조합을 각각 사용하는 pipeline 생성
- ✓ KNN, Logistic Regression Model 추가: 단 $10이라도 비용을 더 줄이기 위하여…
- ✓ 현 시점에서 동원할 수 있는 최선의 수!

그 결과는……

5 결론 및 분석

(1) 최초 수령한 데이터에 Voting Classifier를 적용한 결과

Predicted / True	neg	pos	All
neg	11125	53	11178
pos	11	211	222
All	11136	264	11400

✓비용: 11*500+53*10 = $6,030

✓비록 XGBoost같은 일부 단독 모형보다는 비용이 많이 나오는 것으로 결과가 도출되었으나, 보다 다양한 상황에서도(ex. random_state 값 변경) 안정적인 값이 나올 것이라는 판단 하에 이 모형을 4조의 최종 모형으로 선택함.

5 결론 및 분석

(2) 최종 Test 데이터에 Voting Classifier를 적용…

```
truck_test = pd.read_csv("Test_data_features.csv", index_col = "Unnamed: 0", na_values = "na")

truck2_test = truck_test.drop(cols_to_drop, axis=1)
print(truck2.shape)
print(truck2_test.shape)
print("-----------------------------------------------")

wighted_median = truck_pos.median().multiply(0.1) + truck_neg.median().multiply(0.9)
truck3_test = truck2_test.fillna(wighted_median)

print(truck3.shape)
print(truck3_test.shape)
print("-----------------------------------------------")

hist_list = [i for i in col if i[:2] in hist_var]
num_list_test = num_list.copy()
num_list_test.remove("class")

truck_hist_test = truck3_test[hist_list].replace(0,1).apply(np.log)
truck_num_test = truck3_test[num_list_test]
truck4_test = pd.concat([truck_hist_test,truck_num_test], axis = 1)

print(X.columns)
print(truck4_test.columns)
print("-----------------------------------------------")
```

```
6,030,,,,,,,6,030,,,,,,,,,,
6,030,,,,,,,6,030,,,,,,,,,,
6,030,,,,,,,6,030,,,,,,,,,,
```

5 결론 및 분석

(3) …한 결과 및 분석

머신러닝 4조	0.984	22150	array([[18426, 265], [39, 270]])

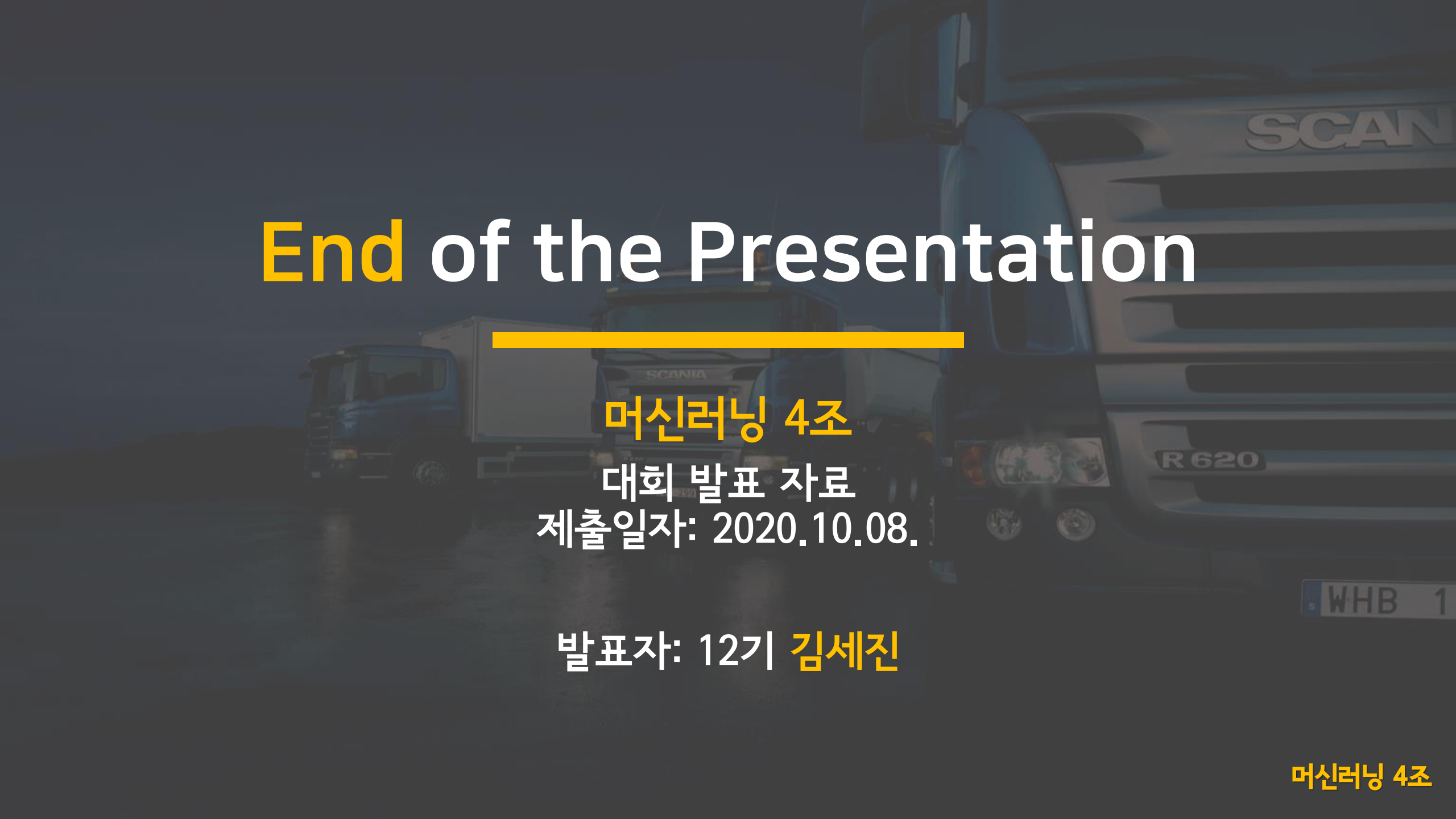
✓ (by 팀장님) 다시 돌아간다면?

=> 최초 Dataset을 전처리하는 과정에서 class 별로 데이터를 나눈 후 median 값으로 결측치를 메웠는데, 막상 Test Dataset에서 그렇게 할 수 없어 임의로 0.9*median(neg) + 0.1*median(pos)로 가중median값을 세팅해 대체하였다.

=> 그런데 비용이 500:10 이었음을 생각하면 **가중치를 훨씬 크게 했어야**…

✓ 실종되어버린 GRID SEARCH?

~~그래도 Accuracy Score는 1등인~~ 머신러닝 4조



End of the Presentation

머신러닝 4조

대회 발표 자료

제출일자: 2020.10.08.

발표자: 12기 김세진