Scania trucks APS failure decision

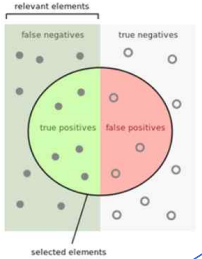
- binary classification -

딥러닝 1조

Goal

- **1차적으로 Recall rate**이 높으며, 2차적으로 Precision rate도 높일 수 있는 model
  - *Cost = FN * 500 + FP * 10*

    ∴ False negative(고장났으나 정상이라고 판단) cost가 500, False positive(고장나지 않았으나 고장이라고 판단) cost 10으로, FN을 줄이는 것이 전체적인 cost를 나누는 데 크게 기여하기 때문.



APS failure(positive; 1)으로 분류된 것 중 true failure의 비율
= 높을 수록 순도가 높은(잘못 예측된 값이 적게 섞인) 높음

실제 APS failure(positive; 1)인 것 중 positive로 predict된 것 비율
= 높을 수록 true class에서 놓치는 것이 적음

Summary

data_processing.py

1. na's replacement
2. Attributes reorganization
3. Standardization

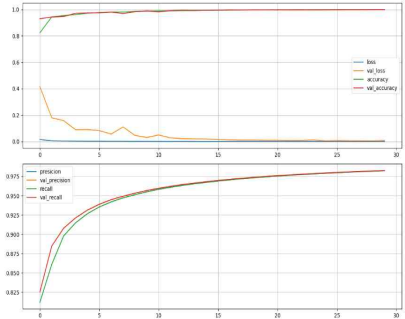
Classification.py

- Keras, 5 dense layers
- Configuration
  - CLASS_WEIGHT = {0: 0.015, 1: 0.985}
  - optimizer, LEARNING_RATE = Adam, 0.00001
  - BATCH SIZE = 200
  - EPOCHS = 30
  - layers = Dense layers, (100, 100, 50, 50, 50)

evaluation.py

Evaluate per model

- **Loss graph**



- **Confusion matrix and cost**

```
[[55872    62]
 [    9  1057]]
5120
```

Code Structure

Data_processing.py | Classification.py | Evaluation.py

1-1 na's replacement : replace na's depending on class(i.e. positive/negative)

- ✓ Why not replace with class-free mean values?
  : P(neg|NA) = over 90% (may be sensor is not working)
  If just replace with class-free mean values, looses na's effect on 'neg' class.

1-2 na's replacement : replace na's to negative's mean (Most of data is negative)

	aa_000	ab_000	ac_000	ad_000	ae_000	af_000	ag_000	ag_001	ag_002	ag_003	ag_004	ag_005	ag_006	ag_007	ag_008	ag_009	ah_000	ai_000	aj_000	ak_000
NA	0	43988	3237	14178	2420	2420	641	641	641	641	641	641	641	641	641	641	616	595	595	4251
TOTAL	57000	57000	57000	57000	57000	57000	57000	57000	57000	57000	57000	57000	57000	57000	57000	57000	57000	57000	57000	57000
NA / TOTAL	0.00%	77.17%	5.68%	24.87%	4.25%	4.25%	1.12%	1.12%	1.12%	1.12%	1.12%	1.12%	1.12%	1.12%	1.12%	1.12%	1.08%	1.04%	1.04%	7.46%
neg|NA	0	43167	2738	13466	2062	2062	638	638	638	638	638	638	638	638	638	638	573	554	554	3685
pos|NA	0	821	499	712	358	358	3	3	3	3	3	3	3	3	3	3	43	41	41	566
P(neg|NA)	0	98.13%	84.58%	94.98%	85.21%	85.21%	99.53%	99.53%	99.53%	99.53%	99.53%	99.53%	99.53%	99.53%	99.53%	99.53%	93.02%	93.11%	93.11%	86.69%
P(pos|NA)	0	1.87%	15.42%	5.02%	14.79%	14.79%	0.47%	0.47%	0.47%	0.47%	0.47%	0.47%	0.47%	0.47%	0.47%	0.47%	6.98%	6.89%	6.89%	13.31%

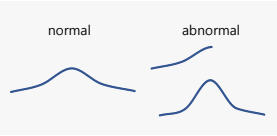
Code Structure

Data_processing.py | Classification.py | Evaluation.py

2. Attributes reorganization

1. non-histogram columns: no reorganizing
2. histogram columns: aggregate rowwise max, min, number of activated bins.

Why max and min values?
: most row wise value of histogram columns(e.g. ag_001 ~ ag_009) have similar shpae like a bell
=>abnormal data may have (1) an uncompletely drawn bell shape or (2) too high / low value.



Why activated bins?
: Considering data details, each bins may mean time axis. (e.g. ag_001 ~ ag_009 may mean pressure change upon time)
low activated bins value -> radical change (e.g. sudden acceleration)
high activated bins value -> retarded change (not working in time).

ag_001	ag_002	ag_003	ag_004	ag_005	ag_006	ag_007	ag_008	ag_009
0	0	0	51396	886464	1445974	463524	37460	288

ag_max	ag_min	ag_actv_bins
1445974	**288**	**7**

3. Standardization
: value range between columns varies a lot

Code Structure

Data_processing.py | **Classification.py** | Evaluation.py

Class weight application

1. Difference in loss per classes
   (loss = FN * 500 + FT * 10, have to increase Recall)
2. Imbalance between positive data and negative data
   neg data(about 99%) >> >pos data.

```
#model configuration setting
CLASS_WEIGHT = {0:0.015, 1:0.985}
LEARNING_RATE = 0.0001

class_weight = CLASS_WEIGHT
hist = model.fit(x_train, y_train, validation_data=(x_val, y_val), batch_size = 200, epochs=30, class_weight = class_weight)
loss, accuracy , recall, precision = model.evaluate(x_val, y_val)
print("Accuracy = {:.2f}".format(accuracy))
```

Code Structure

Data_processing.py | **Classification.py** | Evaluation.py

MLP model

- NO information about attributes
- Basic Deep learning model
- Dropout, Rnn model can be considered

```
model = Sequential()
model.add(Dense(100,input_shape=(x_train.shape[1],),activation='relu'))
model.add(Dense(100,activation='relu'))
model.add(Dense(100,activation='relu'))
model.add(Dense(50,activation='relu'))
model.add(Dense(50,activation='relu'))
model.add(Dense(50,activation='relu'))
# model.add(BatchNormalization(axis=1))
model.add(Dense(2,activation='softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(LEARNING_RATE),
              metrics=['accuracy', tf.keras.metrics.Recall(name='recall'),tf.keras.metrics.Precision(name='precision')])
model.summary()
```

Code Structure

Data_processing.py | Classification.py | Evaluation.py

- Replace Na to 'Train negative' mean
  - Accuraccy : 0.904 , Cost : 54060

```
[[16935  1756]
 [   73   236]]
54060
              precision    recall  f1-score   support

           0       1.00      0.91      0.95     18691
           1       0.12      0.76      0.21       309

    accuracy                           0.90     19000
   macro avg       0.56      0.83      0.58     19000
weighted avg       0.98      0.90      0.94     19000
```

Code Structure

Data_processing.py | Classification.py | Evaluation.py

- Replace Na depending on test class
  - Accuracy : 0.996

Confusion matrix

	Pred neg	Pred pos
Real neg	18655	36
Real pos	36	273

Total cost: 18360 dollar

	precision	recall	f1-score	support
0	1.00	1.00	1.00	18691
1	0.88	0.88	0.88	309
accuracy			1.00	19000
macro avg	0.94	0.94	0.94	19000
weighted avg	1.00	1.00	1.00	19000

End of slides
Thank you