

T 국가 세관 신고 데이터 기반 위법물 탐지 서비스 개발

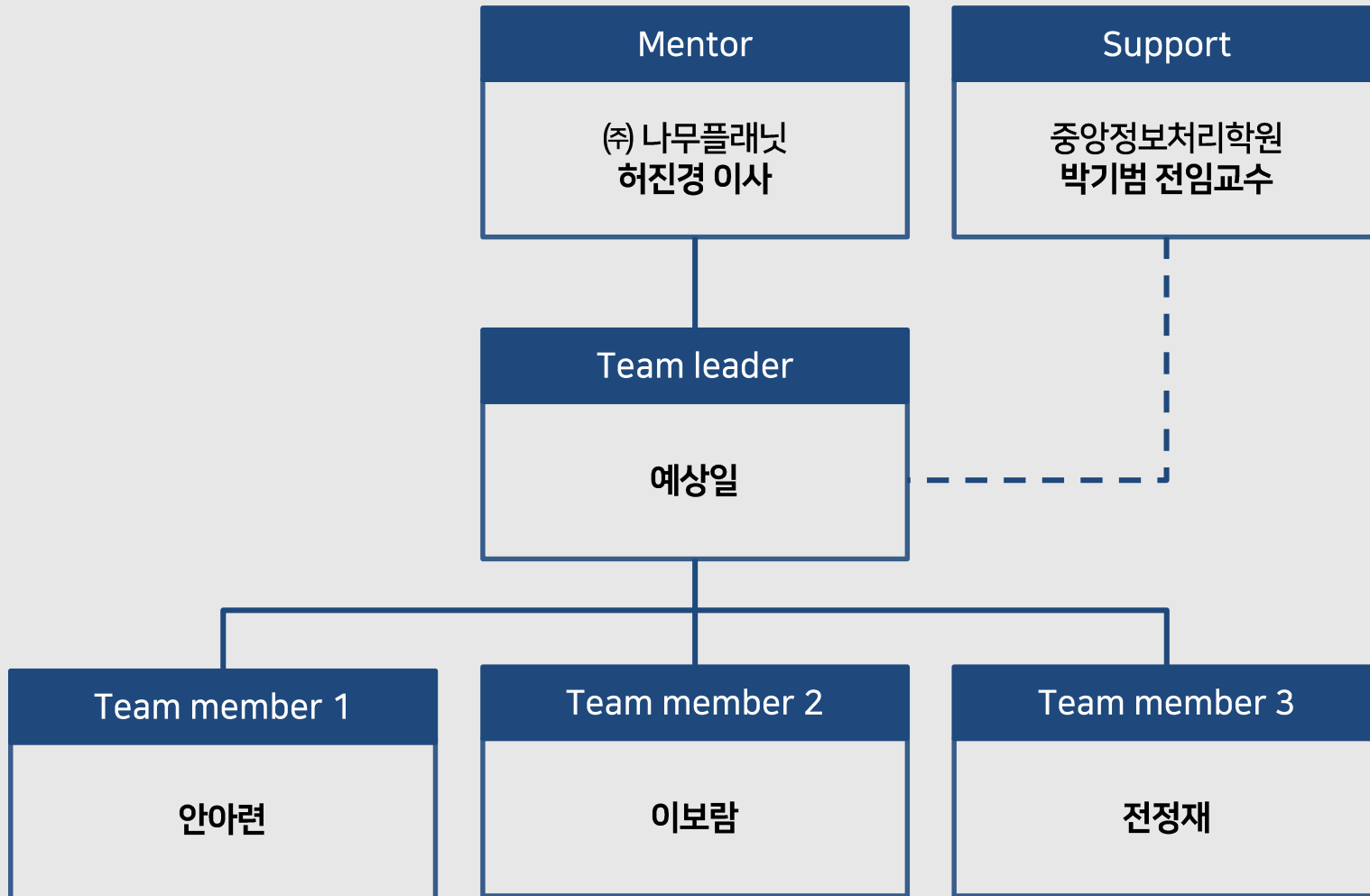
Team NA1

예상일 전정재 이보람 안아련

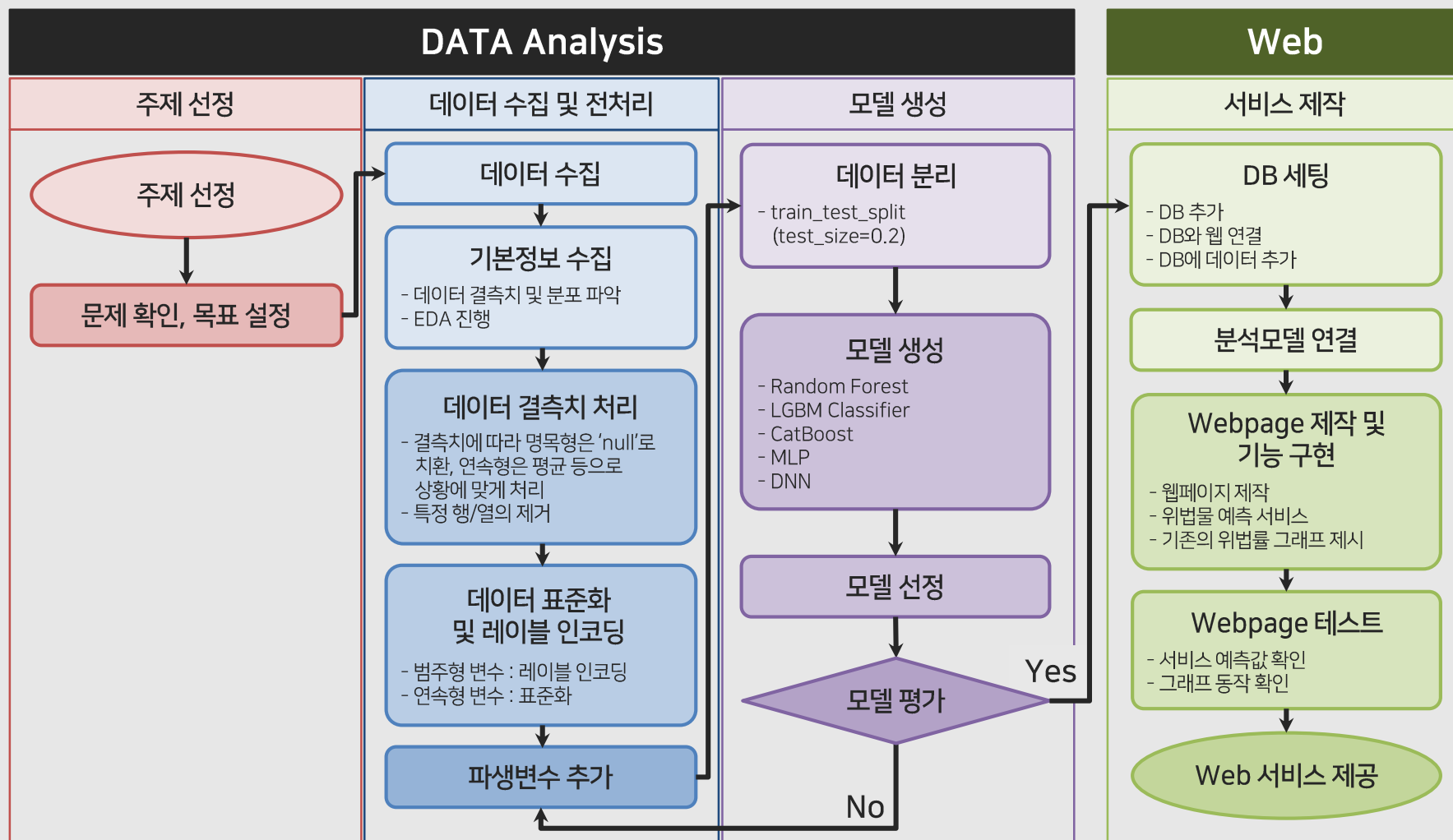
목 차

- 1 탐색적 데이터 분석
- 2 전처리
- 3 변수 선택 및 파생변수 선정
- 4 모델링
- 5 서비스 시연 및 목표

프로젝트 조직도



Project Flow Chart



프로젝트 사용 기술

사용언어



Python



SQL

개발 툴



Jupyter



PyCharm



Flask

운영체제



Windows 10

Windows10

라이브러리



NumPy

Numpy



pandas

Pandas



ScikitLearn



plotly

Plotly



LightGBM



CatBoost

CatBoost



TensorFlow

TensorFlow



Keras

Keras

데이터베이스



DataGrip



SQL Developer

1

탐색적 데이터 분석

분석에 활용하기 위하여 중복 행 제거 및 종속변수 설정

	IMP_TYPE_OF_DECLARATION_1	IMP_TYPE_OF_DECLARATION_2	TRD_TIN_2	TRD_NAME_2	TRD_COUNTRY_2	TRD_ADDR_2
0	ИМ	NaN	NaN	ООО ВИП-ТРАНС	RU	НИЖНЕВАРТОВСК КУЗОВАТКИНА ДОМ 5 СТРОЕНИЕ 3
1	ИМ	NaN	NaN	ООО ВИП-ТРАНС	RU	НИЖНЕВАРТОВСК КУЗОВАТКИНА ДОМ 5 СТРОЕНИЕ 3
2	ИМ	NaN	NaN	ООО ВИП-ТРАНС	RU	НИЖНЕВАРТОВСК КУЗОВАТКИНА ДОМ 5 СТРОЕНИЕ 3
3	ИМ	NaN	NaN	" М/С АКИДЖ ДУТЕ МИЛЛС ЛТД", Ч/З БАНДАР АБАС ПУ...	BD	АББАС
4	ИМ	NaN	NaN	" М/С АКИДЖ ДУТЕ МИЛЛС ЛТД", Ч/З БАНДАР АБАС ПУ...	BD	АББАС

■ 원본 데이터

- '17. 1월 ~ '20. 11월 까지의 T 국가 수입물
통관 데이터

■ 주요 내용

- 계약 관계자, 금액, 화물 내용, 운송 방법 등을
포함한 신고서 및 계약 내역
- 담당 관세사, 부과 세금의 종류, 납부 방법,
금액 등의 관세 관련 정보



데이터
과제 정의

적발건의 패턴을 학습하여
화물 위법 여부 예측 모형 개발

파일 목록 확인

```
['imp_2017_1-4.x|sx',  
 'imp_2017_5-8.x|sx',  
 'imp_2017_9-12.x|sx',  
 'imp_2018_1-4.x|sx',  
 'imp_2018_5-8.x|sx',  
 'imp_2018_9-12.x|sx',  
 'imp_2019_1-4.x|sx',  
 'imp_2019_5-8.x|sx',  
 'imp_2019_9-12.x|sx',  
 'imp_2020_1-4.x|sx',  
 'imp_2020_5-8.x|sx',  
 'imp_2020_9-11.x|sx']
```

중복 행 제거 및 데이터 병합

Labels

데이터 불러오기

```
1 illegal = pd.read_excel('drive/MyDrive/NAI/source_code/dataset/csv_data/적발내역.xlsx',
2                         sheet_name=1, engine="openpyxl")
3 illegal.drop(['Unnamed: 0', 'Unnamed: 1', 'Unnamed: 11'], axis=1, inplace=True)
4 illegal.rename(columns={'해당년도': 'YEAR', '업체': 'IMP_COMPANY', '적발건수': 'NUMB_OF_DETECTED',
5                         '세관코드': 'CUSTOMS_CODE', '신고서No.': 'CUST_NUMBER',
6                         'Unnamed: 7': 'HS_CODE', '품목': 'GOODS',
7                         '검사결과코드': 'INSPECTION_RESULT_CODE', '위반사항': 'VIOLATIONS'}, inplace=True)
```

데이터 불러오기 및 불필요 칼럼 삭제

한글 컬럼명을 영문으로 변경

```
8 illegal.drop(0, inplace=True)
```

칼럼 설명 칼럼(0번째 행) 삭제

```
1 ill_idx = []
2
3 for n, i in illegal[['CUST_NUMBER', 'HS_CODE']].iterrows():
4     try:
5         tup = (i.cust_number+' ', int(i.hs_code))
6         ill_idx.append(tup)
7     except:
8         pass
9 ill_idx[:5]
```

iterrows() 이용
CUST_NUM, HS_CODE 칼럼 값을

```
[('762118/200117/0000002 ', 6907100000),
 ('762118/040617/0000012 ', 6907100000),
 ('762118/260417/0100105 ', 3903909000),
 ('762118/280518/0000230 ', 3903909000),
 ('762118/300518/0000232 ', 3903909000)]
```

각각 (n, i)에 반환 받아
리스트(ill_idx)에 저장

F&L

독립변수, 종속변수 병합

```
1 df['y'] = 0 #종속변수 column
2 df.reset_index(drop=True, inplace=True)
```

칼럼 y 생성 (종속변수) 칼럼

```
1 df.iloc[:5, -3:]
```

	GEND_ISSUE_DATE_54	ACCEPTANCE_DATE	y
0	20130403.0	12.10.20 12:20:02,825000000	0
1	20130403.0	12.10.20 12:20:02,825000000	0
2	20130403.0	12.10.20 12:20:02,825000000	0
3	20130605.0	03.09.20 16:51:17,019000000	0
4	20130605.0	03.09.20 16:51:17,019000000	0

```
4 for i in range(len(ill_idx)):
5     idx = df[CUS_REF_NO_7] == ill_idx[i][0].index
6     if len(idx) != 0:
7         for i in range(len(idx)):
8             df['y'].iloc[idx[i]] = 1
9
10 print(df.y.value_counts())
```

Labels의 CUST_NUMBER가
Features의 CUS_REF_NO_7에 있는 경우
그 index를 리스트(idx)에 저장

적발 데이터가 존재할 때,
리스트(idx)에 포함된 모든 index만큼 반복문 실행
→ 리스트(idx)에 포함된 인덱스의 y 칼럼 값을 1로 변경

```
0    2173173
1     13958
Name: y, dtype: int64
```

y 칼럼 항목별
합계

for문, if문 이용
CUST_NUMBER 기준
Labels & Features를 매핑하여
적발 된 인덱스의 y 칼럼 값을 1로
변경

F&L

기본 데이터

```
10 print(df.y.value_counts())
```

```
0    2173173
1      13958
Name: y, dtype: int64
```

```
1 df.shape
```

```
(2148126, 74)
```

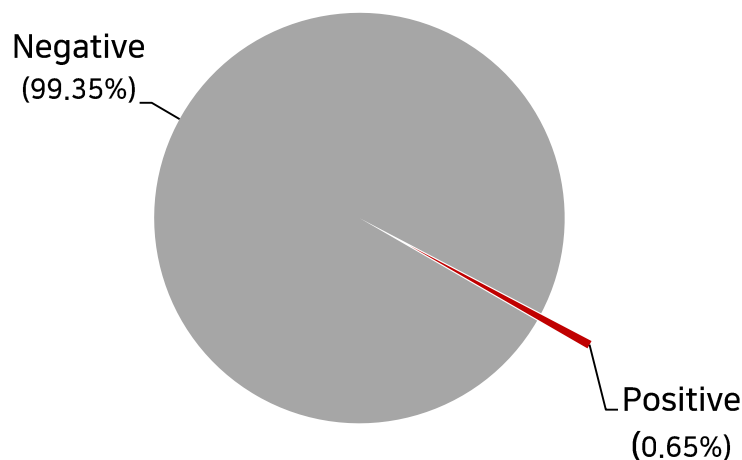
	IMP_TYPE_OF_DECLARATION_1	IMP_TYPE_OF_DECLARATION_2	TRD_TIN_2	TRD_NAME_2	TRD_COUNTRY_2	TRD_ADDR_2	CUS_SHIPMENT_SPEC_4	CUS_TOTAL_NUMBER_OF_ITEMS_5	CUS_TOTAL_NUMBER_OF_PACKAGES_6	CUS_REF_NO_7	ACCEPTANCE_DATE	LABEL
0	ИМ	NaN	NaN	ООО ВИП-ТРАНС	RU	НИЖНЕВАРТОВСК КУЗОВАТКИНА ДОМ 5 СТРОЕНИЕ 3	NaN	1	64.0	762235/140217/0000189	14.02.17 11:55:18,551000000	0
1	ИМ	NaN	NaN	ООО ВИП-ТРАНС	RU	НИЖНЕВАРТОВСК КУЗОВАТКИНА ДОМ 5 СТРОЕНИЕ 3	NaN	1	64.0	762235/140217/0000189	14.02.17 11:55:18,551000000	0
2	ИМ	NaN	NaN	ООО ВИП-ТРАНС	RU	НИЖНЕВАРТОВСК КУЗОВАТКИНА ДОМ 5 СТРОЕНИЕ 3	NaN	1	64.0	762235/140217/0000189	14.02.17 11:55:18,551000000	0
3	ИМ	NaN	NaN	" М/С АКИДЖ ДУТЕ МИЛЛС ЛТД", Ч/З БАНДАР АБАС ПУ...	BD	АББАС	NaN	1	1213.0	762241/140217/0100665	14.02.17 15:53:44,921000000	0
4	ИМ	NaN	NaN	" М/С АКИДЖ ДУТЕ МИЛЛС ЛТД", Ч/З БАНДАР АБАС ПУ...	BD	АББАС	NaN	1	1213.0	762241/140217/0100665	14.02.17 15:53:44,921000000	0

총 2,148,126건
74개 칼럼
적발건수 : 13,961건(0.65%)

...

프로젝트 진행에 사용할 인사이트 탐색

[정상/이상 데이터 비교]



```
fig, ax = plt.subplots(figsize=(8, 6), subplot_kw=dict(aspect="equal"))

negative_ratio = round(((df.LABEL.value_counts()[0]/df.LABEL.value_counts().sum())*100),2)
positive_ratio = round(((df.LABEL.value_counts()[1]/df.LABEL.value_counts().sum())*100),2)

label = ["Negative ({}%)".format(negative_ratio),
        "Positive ({}%)".format(positive_ratio)]

data = [df.LABEL.value_counts()[0], df.LABEL.value_counts()[1]]

color = sns.color_palette("pastel", len(label))

wedges, texts = ax.pie(data, wedgeprops=dict(width=0.5), startangle=-40, colors=color)

bbox_props = dict(boxstyle="square,pad=0.3", fc="w", ec="k", lw=0.72)
kw = dict(arrowprops=dict(arrowstyle="->"),
          bbox=bbox_props, zorder=0, va="center")

for i, p in enumerate(wedges):
    ang = (p.theta2 - p.theta1)/2. + p.theta1
    y = np.sin(np.deg2rad(ang))
    x = np.cos(np.deg2rad(ang))
    horizontalalignment = {-1: "right", 1: "left"}[int(np.sign(x))]
    connectionstyle = "angle,angleA=0,angleB={}".format(ang)
    kw["arrowprops"].update({"connectionstyle": connectionstyle})
    ax.annotate(label[i], xy=(x, y), xytext=(1.35*np.sign(x), 1.4*y),
                horizontalalignment=horizontalalignment, **kw)

ax.set_title("Comparison : Negative & Positive")

plt.show()
```

전체 데이터 : 2,148,126건

- 정상 데이터 : 2,134,168건 (99.35%)
- 이상 데이터 : 13,961건 (0.65%)



프로젝트 진행을 위한 핵심 task
: 클래스 불균형 해소

적발건수 그래프 출력 사용자 함수

- df : 시각화 하고자 하는 데이터가 포함된 데이터프레임
- col_name : df 안의 시각화하고자 하는 column 이름
- x_name : 그려지는 그래프의 x축 이름
- y_name : 그려지는 그래프의 y축 이름
- title : 그려지는 그래프의 제목
- show_num : 표시하고자 하는 값의 개수 (default = 40)

```

2 def show_detection_bar(df, col_name, x_name, y_name, title, show_num=40):
3     import plotly.express as px
4     idf = df.loc[df.LABEL==1] # 적발된 데이터만 추출한 데이터프레임
5     a_index = idf[col_name].value_counts().index # 적발 df에서 입력한 column에 대한 값들의 index
6     a = idf[col_name].value_counts() # 적발 df에서의 입력한 column에 대한 value들 전체
7
8     temp_case = [] # 적발 건수를 담기 위한 list
9     for anum in a:
10         temp_case.append(anum) # 단순 적발 건수를 list에 추가
11     temp_df = pd.Series(temp_case, index=a_index).T
12
13     res_df = temp_df.sort_values(ascending=False)[:show_num] # 그래프 오름차순 설정
14     fig = px.bar(x=res_df.index, y=list(res_df), text=list(res_df)) # 그래프에 x, y축 값 입력
15     fig.update_layout(title=title, xaxis_title = x_name, yaxis_title = y_name, # title,x/y축 이름을 입력값으로 지정
16                       autosize=False, width=500, height=300) # fig size 조정
17     fig.show()

```

적발건수 비율 그래프 출력 사용자 함수

- `df` : 시각화 하고자 하는 데이터가 포함된 데이터프레임
- `col_name` : `df` 안의 시각화하고자 하는 column 이름
- `title` : 그려지는 그래프의 제목
- `ratio_sort` : `True` 입력 시 비율 순으로 내림차순, `False` 입력 시 내림차순으로 그래프 정렬 (default = `True`)
- `show_num` : 표시하고자 하는 값의 개수 (default = 40)
- `x_name` : 그려지는 그래프의 x축 이름
- `y_name` : 그려지는 그래프의 y축 이름

```

3 def show_ratio_bar(df, col_name, x_name, y_name, title, ratio_sort=True, show_num=40):
4     idf = df.loc[df.LABEL==1] # 적발된 데이터만 추출한 데이터프레임
5     a_index = idf[col_name].value_counts().index # 적발 df에서 입력한 column에 대한 값들의 index
6     a = idf[col_name].value_counts() # 적발 df에서의 입력한 column에 대한 value들 전체
7     b = df[col_name].value_counts()[a_index] # 전체 df에서 비교를 위해 a_index를 기반으로 한 value들 전체
8
9     temp_ratio = [] # 적발 비율을 담기 위한 list
10    temp_case = [] # 적발 건수를 담기 위한 list
11    for anum, bnum in zip(a, b): # zip으로 적발건수, 전체 거래건수 출력
12        temp_ratio.append(round((anum/bnum)*100, 4)) # (적발건수/전체 거래건수) * 100 => 해당 항목에 대한 적발 비율, 계산 후 list에 추가
13        temp_case.append(anum) # 단순 적발 건수를 list에 추가
14
15    temp_df = pd.DataFrame([temp_ratio, temp_case], index=['ratio', 'casenum'], columns=a_index).T # 컬럼을 ratio(적발 비율), casenum(적발 건수)로 가지는 데이터프레임 생성
16
17    if ratio_sort==True: # default 설정. 데이터프레임에 포함된 ratio 기준(내림차순)으로 정해진 데이터 개수만큼 그래프 출력
18        res_df = temp_df.sort_values('ratio', ascending=False)[:show_num]
19        fig = px.bar(x=res_df.index, y=list(res_df.ratio), text=list(res_df.casenum)) # x축을 시리즈의 인덱스로, y축을 시리즈의 데이터(적발 비율)로 설정하고 각 barplot에 건수 데이터 입력
20        fig.update_layout(title=title, xaxis_title = x_name, yaxis_title = y_name,
21                            autosize=False, width=500, height=300) # 플롯 제목, x축 이름, y축 이름을 각각 입력받은 값으로 설정
22        fig.show()
23
24    elif ratio_sort==False: # default 외 별도 설정. 데이터프레임에 포함된 casenum 기준(내림차순)으로 정해진 데이터 개수만큼 그래프 출력
25        res_df = temp_df.sort_values('casenum', ascending=False)[:show_num]
26        fig = px.bar(x=res_df.index, y=list(res_df.ratio), text=list(res_df.casenum))
27        fig.update_layout(title=title, xaxis_title = x_name, yaxis_title = y_name,
28                            autosize=False, width=500, height=300)
29        fig.show()

```

물건 수량 CUS_TOTAL_NUMBER_OF_ITEMS_5

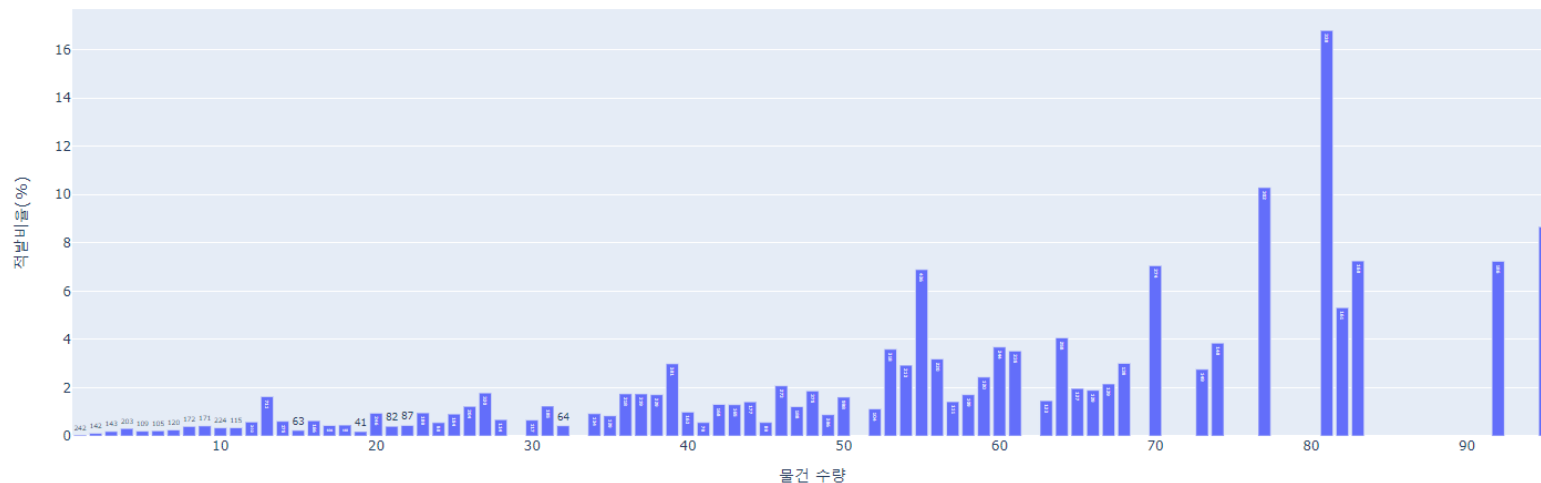
적발비율

- 수량이 많아질수록 적발율이 증가하는 경향을 보임
- 수량이 81개 일때 적발율 가장 높음(16.8%)



적발비율을 고려하여
변수 활용

물건 수량별 적발비율



수입업자명 CON_NAME_8

적발건수

- PER_NAME_9(대금지불업자명)와 비슷한 양상

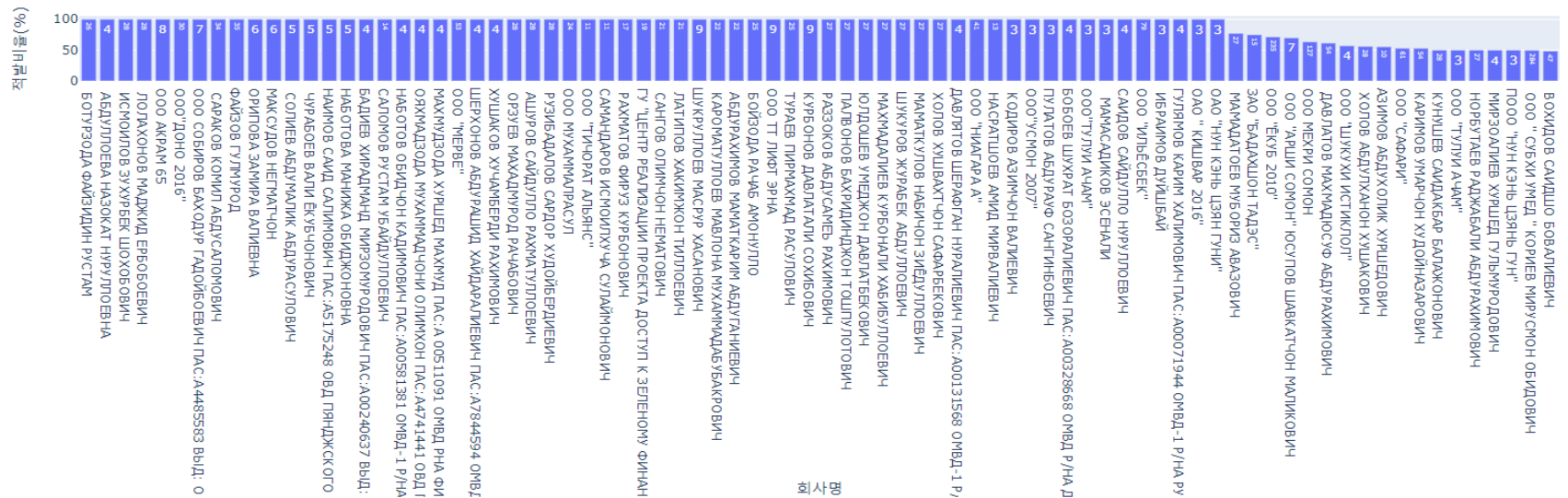
항목	합계
전체 수입업자 수	78,188
적발 수입업자 수	226

적발비율

- 62개의 수입업자명이 100% 위법물로 적발

적발비율을 고려하여
변수 활용

수입업자명에 따른 적발비율



HS CODE COM_COMBINED_NOMENCLATURE_33

적발건수

- Machinery, Nonmetals, ... 순으로 적발건수가 많음

항목	합계
전체 품목 수	20
적발 품목 수	19

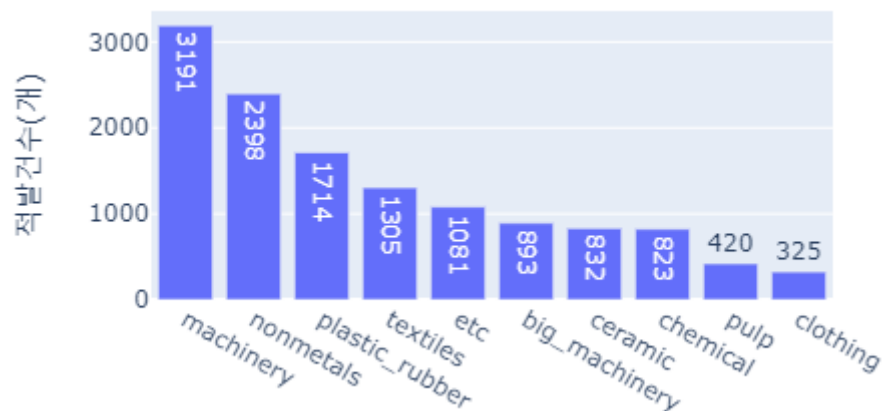
적발비율

- 건수를 고려한 비율로 출력 시,
art의 적발율이 가장 높음

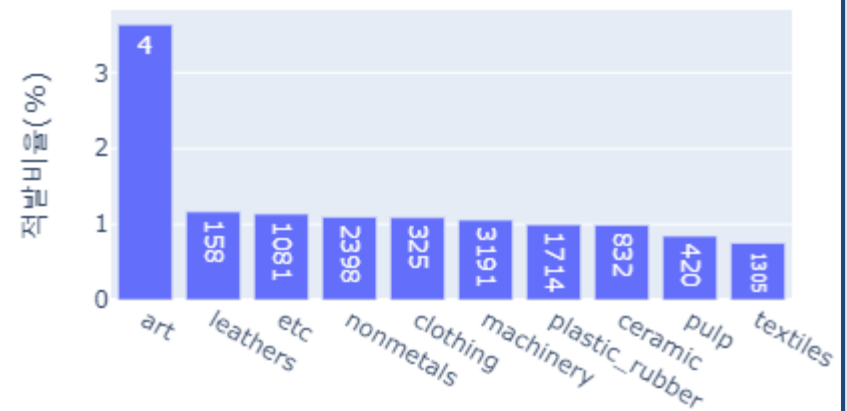


적발비율을 고려하여
변수 활용

품목별 적발건수



품목별 국가별 적발비율



원산지 IDG_COUNTRY_OF_ORIGIN_34

적발건수

- CN이 압도적으로 적발건수가 많음

항목	합계
전체 원산지 수	132
적발 원산지 수	37

적발비율

- 건수를 고려한 비율로 출력 시,
CN이 아닌 PT, NO, ... 순으로 국가별 적발 차이 발생

IMP_COUNTRY_..._16(품목 원산지) 및
적발비율을 고려하여 변수 활용

원산지별 적발건수



원산지별 적발비율



관세사 식별번호 REP_TIN_54

적발건수

- 'D5/02/0081'이 가장 거래량이 많음

항목	합계
전체 설명 수	930
적발 설명 수	86

적발비율

- 건수를 고려한 비율로 출력 시,
3개의 설명이 100% 위법물 적발되고
적발건수와 적발비율의 차이가 있음

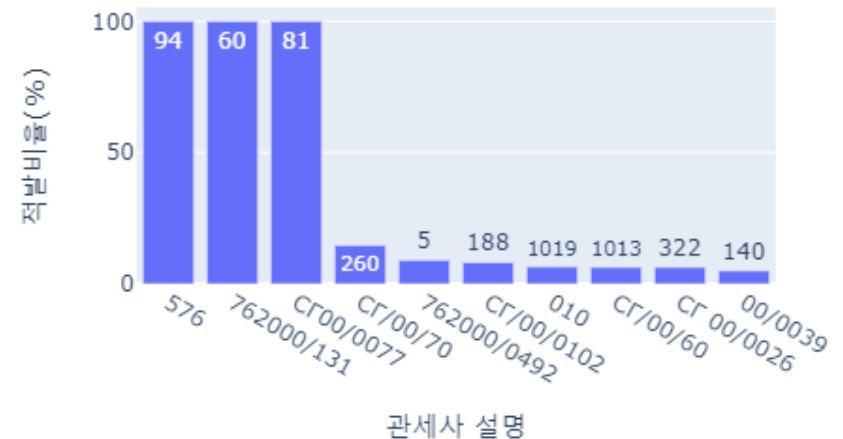


적발비율을 고려하여
독립변수 활용

관세사 설명별 적발건수



관세사 설명별 적발비율



세관 신고서 번호 CUS_REF_NO_7

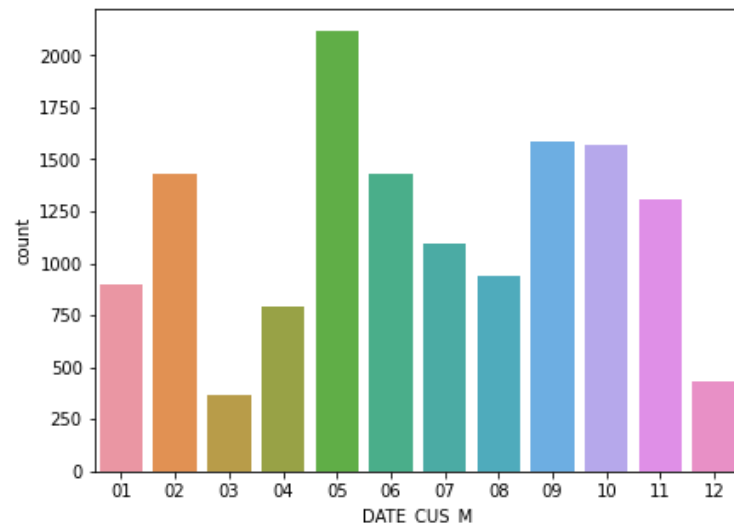
적발건수

- 월별 전체 수입 신고건수는 비슷한 양상을 보이거나 적발건수는 월별로 크게 달라짐
- 5월에 가장 적발건수가 많고, 3월에 가장 적음
- IMP_DATE_OF_DECLARATION 54와 비슷한 양상을 보임

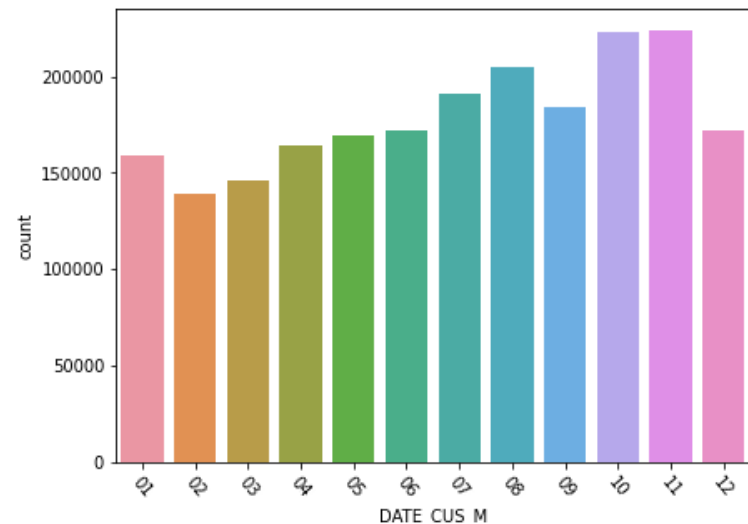


월 패턴을 고려하여 변수 활용

월별 적발건수



월별 수입신고건수



세관 신고서 접수 일자 ACCEPTANCE_DATE

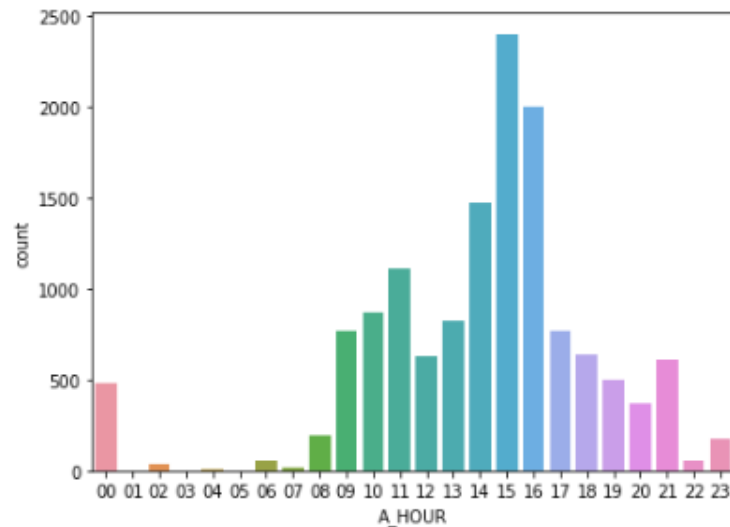
적발건수

- 시간별 전체 수입 신고건수와 시간별 적발 건수가 비슷한 양상을 보이거나 일부 시간에서 차이가 있음
- 00시, 09시, 14시, 15시, 21시 등 특정 시간에 적발 건수가 증가하는 것이 보임

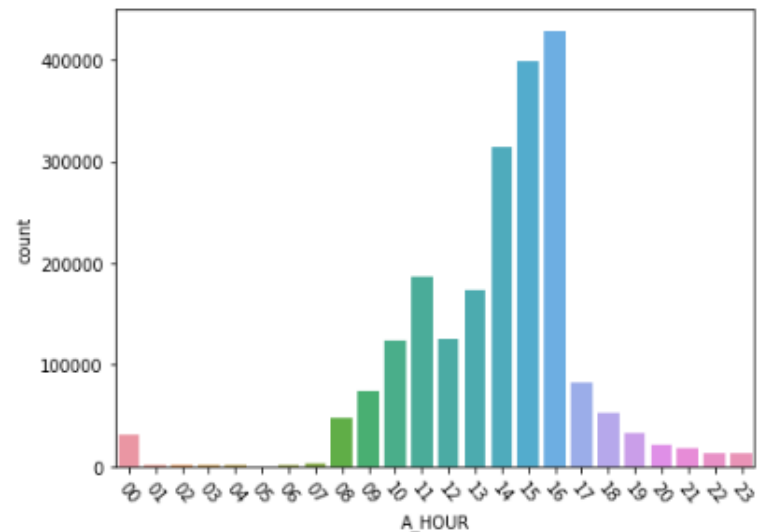


세관 신고 시간을 변수 활용

시간별 적발건수



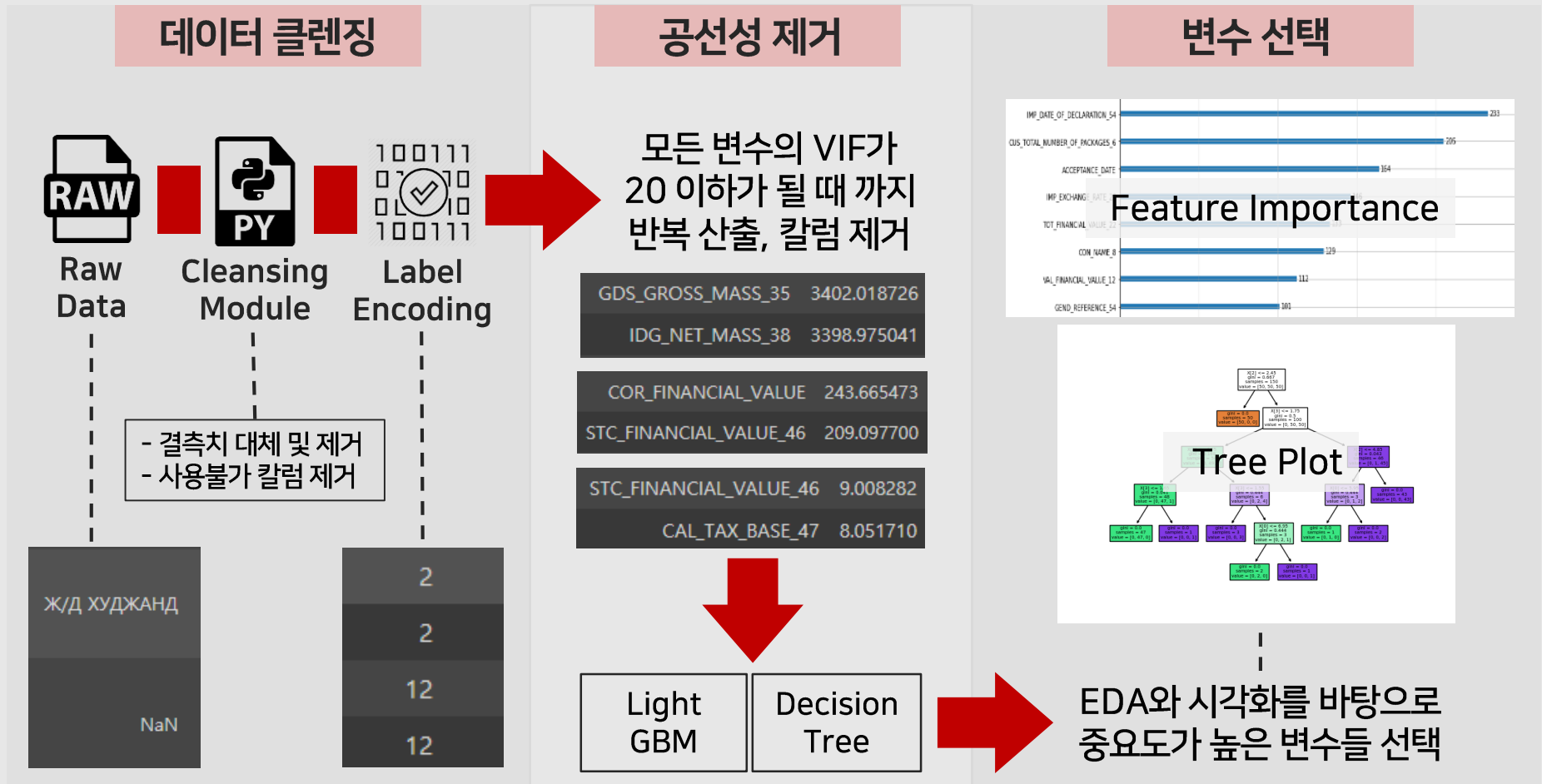
시간별 수입신고건수



2

전처리

패턴 탐색이 용이하도록 데이터 세트 정리



독립변수 전처리 모듈 설명

```
class Preprocess :

    def __init__(self, df) :
        self.df = df

    ##### Class Variables #####

    # 사용이 불가능하다고 판단, 데이터프레임에서 아예 드랍시킬 컬럼들(추후 변동 가능)

    drop_col_list = ['IMP_TYPE_OF_DECLARATION_1', 'IMP_TYPE_OF_DECLARATION_2',
                    'TRD_TIN_2', 'CUS_SHIPMENT_SPEC_4', 'CUS_TOTAL_NUMBER_OF_ITEMS_5',
                    'PER_TIN_9', 'PER_NAME_9', 'PER_COUNTRY_9', 'PER_ADDR_9',
                    'DNT_TIN_14', 'DNT_NAME_14', 'DNT_COUNTRY_14', 'DNT_ADDR_14',
                    'IMP_CONTAINER_FLAG_19', 'LOD_LOCATION_NAME_27', 'IDG_QUOTA',
                    'ZQNTY', 'ZAUXUOM', 'PERSON_POSITION_54', 'LOC_LOCATION_NAME_30',
                    'COV_CUST_VALUE_METHOD', 'IDG_STAT_VALUE_VAL_METH_46']

    # 사용은 가능하지만 중간에 null값이 있는 명목형 컬럼들. fillna로 처리(추후 변동 가능)

    fillna_col_list = ['TRD_COUNTRY_2', 'CUS_REF_NO_7', 'CON_COUNTRY_8',
                    'IMP_TRADING_COUNTRY_11', 'VAL_CURRENCY_12',
                    'IMP_CNT_OF_DISPATCH_EXP_CD_15', 'DEL_DELIVERY_TERM_CODE_20',
                    'TOT_CURRENCY_22',
                    'DEL_PLACE_OF_DELIVERY_20']

    # 사용은 가능하지만 중간에 null값이 있는 연속형 컬럼들. fillna로 처리(추후 변동 가능)

    num_col_list = ['CUS_TOTAL_NUMBER_OF_PACKAGES_6', 'VAL_FINANCIAL_VALUE_12',
                    'TOT_FINANCIAL_VALUE_22', 'IMP_EXCHANGE_RATE_23',
                    'IMP_INLAND_TRANSPORT_MODE_25', 'IMP_TRANSPORT_MODE_AT_BODR_26',
                    'GDS_GROSS_MASS_35', 'IDG_NET_MASS_38', 'FIN_FINANCIAL_VALUE_42',
                    'STC_FINANCIAL_VALUE_46']

    # Label 에 영향을 미치지 않아 단순히 null이 존재하는 인덱스만 드랍(추후 변동 가능)

    index_drop_list = ['PRF_PREFERENCE_CODE_1', 'COR_FINANCIAL_VALUE', 'GEND_REFERENCE_54',
                    'CAL_TYPE_OF_TAX_47', 'CAL_ADDITIONAL_RATE_OF_TAX_47']
```

```
##### Class Methods #####

# 각 컬럼별로 반복문이 돌아가면서
# fillna_col_list에 포함되었다면 null값을 'null'로
# num_col_list에 포함되었다면 컬럼의 평균값으로 null값을 대체

def null_solution(self):
    df = self.df.copy()
    for column_name in df.columns :

        if column_name in self.drop_col_list :
            df.drop(column_name, axis=1, inplace=True)

        elif column_name in self.fillna_col_list :
            df[column_name].fillna('null', inplace=True)

        elif column_name in self.num_col_list :
            mean = df[column_name].mean()
            df[column_name].fillna(mean, inplace=True)

        elif column_name in self.index_drop_list :
            drop_idx = df[df[column_name].isnull()].index
            df.drop(drop_idx, axis=0, inplace=True)

    return df
```

컬럼의 삭제, null값 처리 부분

독립변수 전처리 모듈 설명

```
# Label Encoder
def label(self, classes=None) :

    # 초기화 때 입력한 DataFrame의 사본을 사용
    df = self.df.copy()

    # LabelEncoder 객체 생성
    from sklearn.preprocessing import LabelEncoder
    import numpy as np
    import pandas as pd
    import pickle

    le = LabelEncoder()

    # for문을 사용, 각 칼럼별로 인코딩 적용
    for column in df.columns:
        # 칼럼의 데이터 타입이 str인 것만 인코딩 실시
        if type(df[column][0]) == str :
            # Label Encoding 실시
            column_encoded = le.fit_transform(df[column])
            df[column] = column_encoded
            # 인코딩한 칼럼 이름을 변수명으로 하는 dict 변수 생성
            # 칼럼_이름 = {원래 데이터 : 인코딩 된 번호}
            encoding_val = np.sort(df[column].unique())
            decoding_val = le.classes_
            val_dict = dict(zip(decoding_val, encoding_val))
            # dict를 pickle로 저장
            with open('{}{}.pickle'.format(column), 'wb') as f :
                pickle.dump(val_dict, f, protocol=pickle.HIGHEST_PROTOCOL)

        else : pass

    return df
```

```
# One-hot Encoder
def one_hot(self) :

    import numpy as np
    import pandas as pd

    # self.df의 복사본 생성
    df = self.df.copy()
    # 인코딩 대상 칼럼 넣는 DataFrame
    # 이렇게 하는 이유 : 안그러면 칼럼 이름에 값만 있음
    # 이렇게 해주면 칼럼 이름으로 원래 칼럼 이름 + 값이 있음
    oh_columns = pd.DataFrame()

    # for loop로 대상 칼럼 oh_columns에 넣고
    # df에서는 삭제
    for column in df.columns :
        if str(df[column].dtype) == 'object' :
            oh_columns[column] = df[column]
            df.drop(column, axis=1, inplace=True)
        else : pass

    # get_dummies 함수 사용하여 one_hot 진행
    oh_df = pd.get_dummies(oh_columns)
    for value in oh_df.columns :
        df[value] = oh_df[value]

    return df
```

모델 테스트 진행을 위한
Label Encoding, One-hot Encoding 처리 부분

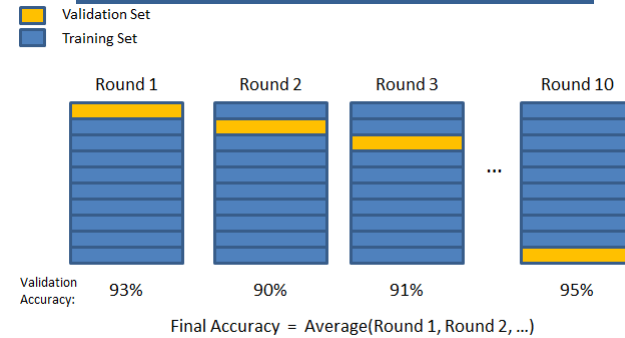
Ideas

클래스 불균형을 해소하기 위한 대안 탐색

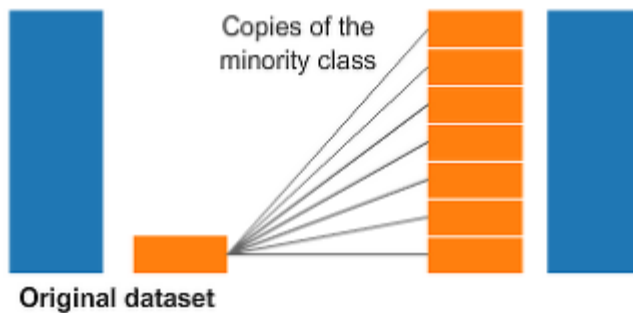
Under Sampling



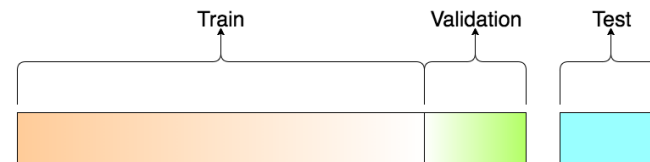
교차 검증



Over Sampling



계통 추출



[Total] Label Classes ratio
 = [Test set] Label classes ratio
 = [Validation set] Label classes ratio

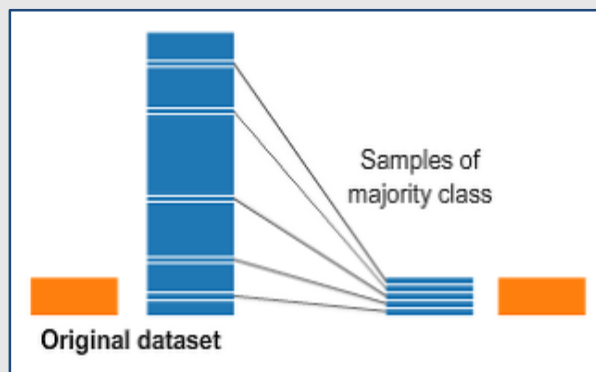
Under Sampling 불가

1 막대한 데이터 손실 발생

- 다수 클래스 개수 : 2,132,549건
 - 소수 클래스 개수 : 13,959건
- = 2,118,590건 **미반영**

2 샘플의 대표성 문제 발생

- 소수 클래스의 개수가 변수 분포를 대표할 만큼 충분하지 못함



소수 클래스의 개수와 일치하도록
다수 클래스를 다운 샘플링

Numb of Minor Class : 13959		
	COLUMN	UNIQUE
36	CAL_TAX_BASE_47	1104206
39	PAM_FINANCIAL_VALUE_47	822861
20	GRS_CODE_DESCRIPTION	545830
3	ACCEPTANCE_CODE	211665
46	VAL_FINANCIAL_VALUE_12	149970
9	FIN_FINANCIAL_VALUE_42	103902
32	TOT_FINANCIAL_VALUE_22	78066
16	CON_NAME_8	
5		

변수들의 Unique 값이
소수 클래스의 개수를
훨씬 초과

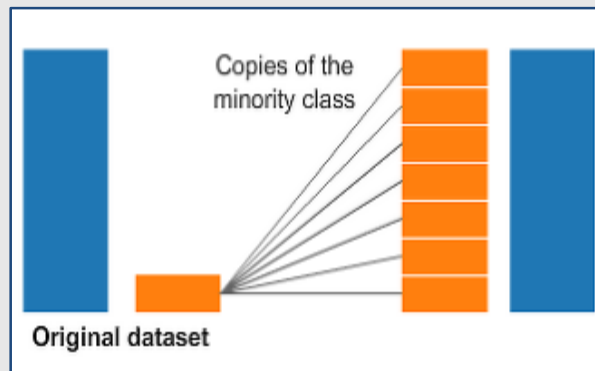
Over Sampling 불가

1 명목/ 범주형 변수 왜곡 발생

- 소수 클래스 복제 시 원본에는 없던 새로운 데이터 생성

2 분류 성능 하락

- 샘플링 된 데이터로 학습 및 검증 시 평가지표 급감 발생



다수 클래스의 개수와
일치하도록 **소수 클래스를 업 샘플링**

원본 데이터	오버 샘플링 데이터
78, 79, 80, 81, 82, 91, 92, 93, 94, 95, 106, 107, 108]	94, 95, 96, 97, 98 107, 108, 109, 110 120, 121, 122,
SMOTE	SVM SMOTE
오차행렬 [[420790 5751] [197 2564]] 정확도 : 0.9861, 정밀도 : 0.3084 재현율 : 0.9286,	오차행렬 [[422428 4113] [365 2396]] 정확도 : 0.9896, 정밀도 : 0.3681 재현율 : 0.8678,

교차검증 시행 불가

Label Distributions: Train/Test set의 Label=0/1 비율

```
[0.99349699 0.00650301]
[0.99349642 0.00650358]
```

- Early stopping, best iteration is:
[45] valid_0's binary_logloss: 0.0391765 valid_0's binary_logloss: 0.0391765
Confusion Matrix
[[710838 11]
 [1134 3519]]
Accuracy : 0.9984, Precision : 0.9969, Recall : 0.7563, f1 score : 0.8601
- Early stopping, best iteration is:
[51] valid_0's binary_logloss: 0.0453395 valid_0's binary_logloss: 0.0453395
Confusion Matrix
[[710832 17]
 [1223 3430]]
Accuracy : 0.9983, Precision : 0.9951, Recall : 0.7372, f1 score : 0.8469
- Early stopping, best iteration is:
[46] valid_0's binary_logloss: 0.276045 valid_0's binary_logloss: 0.276045
Confusion Matrix
[[710849 0]
 [1164 3489]]
Accuracy : 0.9984, Precision : 1.0000, Recall : 0.7498, f1 score : 0.8570
- Early stopping, best iteration is:
[42] valid_0's binary_logloss: 0.255218 valid_0's binary_logloss: 0.255218
Confusion Matrix
[[687664 23185]
 [1843 2810]]
Accuracy : 0.9650, Precision : 0.1081, Recall : 0.6039, f1 score : 0.1834
- Early stopping, best iteration is:
[50] valid_0's binary_logloss: 0.0346984 valid_0's binary_logloss: 0.0346984
Confusion Matrix
[[710844 5]
 [3601 1052]]
Accuracy : 0.9950, Precision : 0.9953, Recall : 0.2261, f1 score : 0.3685

```
## 교차 검증별 정밀도: [0.9969 0.9951 1. 0.1081 0.9953]
## 교차 검증별 재현율: [0.7563 0.7372 0.7498 0.6039 0.2261]
## 평균 검증 정확도: 0.8190638740091118
## 평균 검증 정확도: 0.6146572104018913
```

교차검증 결과

해당 데이터셋은 불균형한 분포도를 가졌기 때문에
단순한 Kfold 교차검증 시,
데이터가 매우 극단적으로 분할될 수 있음

교차검증 모델 중 StratifiedKFold 사용



평균 검증 정확도는 나쁘지 않지만,

- 교차검증별 오차행렬의 편향 발생
- 교차검증별 정밀도 등의 큰 차이 발생

= 교차검증별 결과가 균일하지 않음

계통 추출

Sklearn - Stratify 파라미터

- Train/Validation set Label 클래스 비율을 기준 Label(y_label)의 클래스(0/1) 비율과 동일하게 조정

➡ Test/Validation set 구성 시 클래스 편향 보완

Train/Test 데이터셋 분할 시 Stratify 파라미터 이용

```
# train_test_split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_features, y_label, test_size=0.2, random_state=0, stratify=y_label)
```

3

변수 선택 및 파생변수 선정

기본 파생변수

주요 변수들에 기반한 최근 6개월 기준 파생변수 19개 산출

변수 기준	기준 컬럼	항목	파생변수 컬럼명
업체명	CON_NAME_8	수입건수	R6M_CON_IMP_CNT
		수입금액	R6M_CON_IMP_AMT
		검사건수	R6M_CON_NUMB_OF_INSPECTION
		적발률	R6M_CON_NUMB_OF_DETECTED
HS Code	HS_CODE	수입건수	R6M_CODE_IMP_CNT
		수입금액	R6M_CODE_IMP_AMT
		검사건수	R6M_CODE_NUMB_OF_INSPECTION
		적발건수	R6M_CODE_NUMB_OF_DETECTED
원산지 국가	IDG_COUNTRY_OF_ORIGIN_34	수입건수	R6M_COUNTRY_IMP_CNT
		검사금액	R6M_COUNTRY_IMP_AMT
		검사건수	R6M_COUNTRY_NUMB_OF_INSPECTION
		적발건수	R6M_COUNTRY_NUMB_OF_DETECTED
		적발률	R6M_COUNTRY_RATIO_OF_DETECTED
관세사 번호	REP_TIN_54	수입건수	R6M_REP_IMP_CNT
		수입금액	R6M_REP_IMP_AMT
		검사건수	R6M_REP_NUMB_OF_INSPECTION
		적발건수	R6M_REP_NUMB_OF_DETECTED
		적발률	R6M_REP_RATIO_OF_DETECTED

세관 신고 월/시간 기준 파생변수

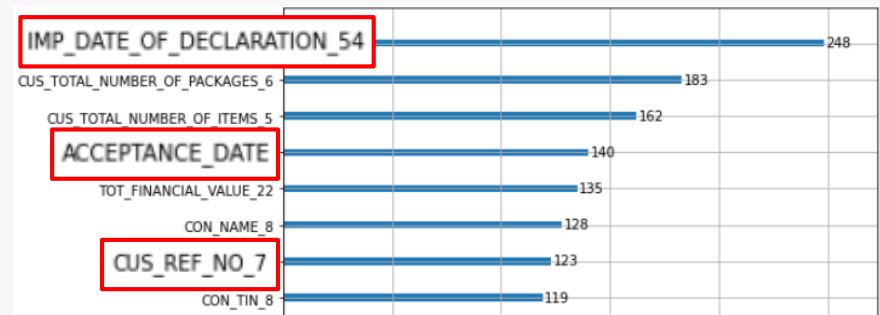
Feature Importnace 상위의 시간 관련 변수를 활용한 파생변수 산출

변수 기준	기준 컬럼	파생변수 컬럼명	변수 추가 사유
세관 신고 월	IMP_DATE_OF_DECLARATION_54 ACCEPTANCE_DATE CUS_REF_NO_7	C_MONTH	- 세관 신고 월별 적발 확률 변화 반영
세관 신고 시간		A_HOUR	- 세관 신고 시간별 적발 확률 변화 반영

```
# 시간 관련 함수들만 추출하여 DataFrame 생성
column_list = ['ACCEPTANCE_DATE', 'IMP_DATE_OF_DECLARATION_54', 'CUS_REF_NO_7']
edf = df[column_list]

# A_HOUR 칼럼 생성
edf['TIME'] = edf.apply(lambda x : x['ACCEPTANCE_DATE'][9:17], axis=1)
edf['D_TIME'] = pd.to_datetime(edf['TIME'])
edf['S_HOUR'] = edf['D_TIME'].dt.strftime('%H')
edf['A_HOUR'] = edf['S_HOUR'].astype(str)

# C_MONTH 칼럼 생성
edf['NEW_C_DATE'] = edf.apply(lambda x : '20' + x['CUS_REF_NO_7'][11:13] + x['CUS_REF_NO_7'][9:11] + x['CUS_REF_NO_7'][7:9], axis=1)
edf['C_MONTH'] = edf.apply(lambda x : x['NEW_C_DATE'][4:6], axis=1)
edf.C_MONTH = edf.C_MONTH.astype(str)
```



업체별 위법률 비율 기준 파생변수

EDA 결과 발견한 인사이트에 따른 파생변수 산출

변수 기준	기준 컬럼	파생변수 컬럼명	변수 추가 사유
업체별 위법률 비율	CON_NAME_8	CON_ILLEGAL_RATIO	- 일부 업체들의 높은 적발 확률 반영

```
# 사용할 변수들만 추출
edf = df[['CON_NAME_8', 'LABEL']]

# 위법률 비율을 담은 리스트 생성
result_list = []

# for문을 돌리기 위하여 CON_NAME_8의 유니크값만 추출
name_list = edf.CON_NAME_8.unique().tolist()

for value in name_list :
    # total = 해당 업체의 모든 샘플의 수
    mask = edf.CON_NAME_8 == value
    total = len(edf[mask])
    # illegal = 해당 업체의 모든 위법률 수
    mask2 = (edf.CON_NAME_8 == value) & (edf.LABEL == 1)
    illegal = len(edf[mask2])
    # 위법률 비율
    result = illegal / total
    result_list.append(result)

# 해당 업체 : 위법률 비율 dictionary 파일로 저장
mapping_dict = dict(zip(name_list, result_list))
with open('/content/drive/MyDrive/NA1/source_code/Feature_Engineering/dataset/Basic_Data_Illegal_dict/CON_ILLEGAL_RATIO_dict.pkl', 'wb') as f :
    pickle.dump(mapping_dict, f)

edf['CON_ILLEGAL_RATIO'] = edf.apply(lambda x : mapping_dict[x['CON_NAME_8']], axis=1)
edf.to_pickle('/content/drive/MyDrive/NA1/source_code/Feature_Engineering/dataset/CON_ILLEGAL_RATIO.pkl')
```

4

모델링

클래스 불균형을 극복하기 위한 최적합 지표 선택

모델 평가 기준

Accuracy

전체 대비
정확하게 예측한 개수의 비율

$$= \frac{TP+TN}{TP+TN+FP+FN}$$

소수 클래스와 다수 클래스의
예측과 실재를 모두 포함



과대한 다수 클래스에 의해 왜곡

Precision

Positive라고 예측한 비율 중
진짜 Positive의 비율

$$= \frac{TP}{TP+FP}$$

= Positive를
얼마나 잘 예측하였는가?

소수 클래스에 대한 예측과
실재를 바탕으로 산출



소수 클래스 분류 평가 시
가장 적합한 지표

Recall

실제 Positive 데이터 중
Positive라고 예측한 비율

$$= \frac{TP}{TP+FN}$$

= 실제 Positive한 것
중에서 얼마나 잘 예측하였는가?

실제 소수클래스에 대한 예측을
바탕으로 산출




소수 클래스 분류 평가에
적합해 보이지만
정확한 정보 전달 불가능

알고리즘별 비교

데이터셋 1


Train size = 0.8 / Test size = 0.2

 분석용 데이터셋	구분	Random Forest	LGBM Classifier	CatBoost	MLP
<ul style="list-style-type: none"> ■ 기본 파생변수 19개 	Accuracy	0.9978	0.9948	0.9961	0.9947
<ul style="list-style-type: none"> ■ EDA 기반 파생변수 - 시간 관련 파생변수 2개 	Precision	0.8633	0.8103	0.8807	0.7565
<p>: 총 21개 변수</p>	Recall	0.7779	0.2600	0.4549	0.2726
<ul style="list-style-type: none"> ■ 명목변수 인코딩 시 더 나은 성능을 보이는 One-hot Encoding 적용 	F1 Score	0.8184	0.3937	0.5999	0.4007
<ul style="list-style-type: none"> ■ MLP, DNN은 학습 이전 Standard Scaling 적용 	Time(초)	382	28	72	1508

알고리즘별 비교

데이터셋 2

Train size = 0.8 / Test size = 0.2


 분석용 데이터셋	구분	Random Forest	LGBM Classifier	CatBoost	MLP
<ul style="list-style-type: none"> ■ 기본 파생변수 19개 	Accuracy	0.9999	0.9968	0.9985	0.9956
<ul style="list-style-type: none"> ■ EDA 기반 파생변수 <ul style="list-style-type: none"> - 시간 관련 파생변수 2개 	Precision	0.9982	0.9365	0.9749	0.8032
<ul style="list-style-type: none"> ■ Feature Importance 기반 기존 중요변수 2개 : 총 23개 변수 	Recall	0.9921	0.5494	0.7944	0.4312
<ul style="list-style-type: none"> ■ 명목변수 인코딩 시 더 나은 성능을 보이는 One-hot Encoding 적용 	F1 Score	0.9951	0.6926	0.8755	0.5612
<ul style="list-style-type: none"> ■ MLP, DNN은 학습 이전 Standard Scaling 적용 	Time(초)	321	8	21	513

Random Forest는 과적합 양상을 보인다고 판단

알고리즘별 비교

데이터셋 3

Train size = 0.8 / Test size = 0.2


 분석용 데이터셋	구분	Random Forest	LGBM Classifier	CatBoost	MLP
	Accuracy	1.0	0.9987	0.9944	0.9962
	Precision	0.9993	0.9946	0.9980	0.8513
	Recall	0.9968	0.7973	0.9112	0.5107
	F1 Score	0.9980	0.8851	0.9526	0.6385
	Time(초)	290	5	22	353

MLP를 제외한 모든 알고리즘이 과적합 양상을 보인다고 판단

알고리즘별 비교

데이터셋 4

Train size = 0.8 / Test size = 0.2

 분석용 데이터셋	구분	Random Forest	LGBM Classifier	CatBoost	MLP
<ul style="list-style-type: none"> ■ 기본 파생변수 19개 ■ EDA 기반 파생변수 <ul style="list-style-type: none"> - 시간 관련 파생변수 2개 - 위법률 관련 파생변수 1개 : 총 22개 변수 ■ 명목변수 인코딩 시 더 나은 성능을 보이는 Label Encoding 적용 ■ MLP, DNN은 학습 이전 Standard Scaling 적용 	Accuracy	0.9990	0.9968	0.9982	0.9980
	Precision	0.9218	✓ 0.9764	0.9201	0.8766
	Recall	0.9165	0.5190	0.7962	0.8120
	F1 Score	0.9192	0.6777	0.8537	0.8431
	Time(초)	298	✓ 5	21	281

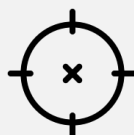
LightGBM 선정 근거

신속한 서비스 & 효과적이고 준수한 분류 성능을 지닌 알고리즘 선택

속도와 성능



가장 빠른 시간 (21초)

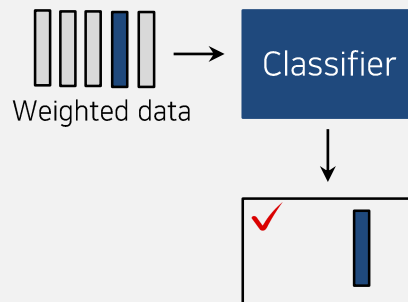


가장 높은 precision
(0.9757)

웹을 통하여 서비스를 구현

준수한 성능과 신속한 결과를
제공하는 LightGBM이 가장 적합

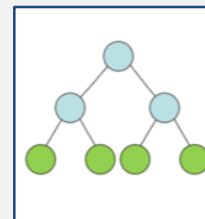
Boosting 방식



다수의 약한 학습기로
순차적 학습/예측

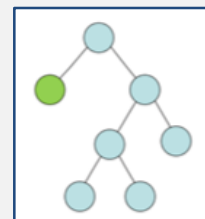
분류에 어려움을 겪었던 데이터는
다음 학습기에 가중치 부여

Leaf Wise 분기



Level Wise

- 균형잡힌 트리 유지
- Depth 최소화
- 과적합에 강함
- 연산 증가
- 많은 시간 소요

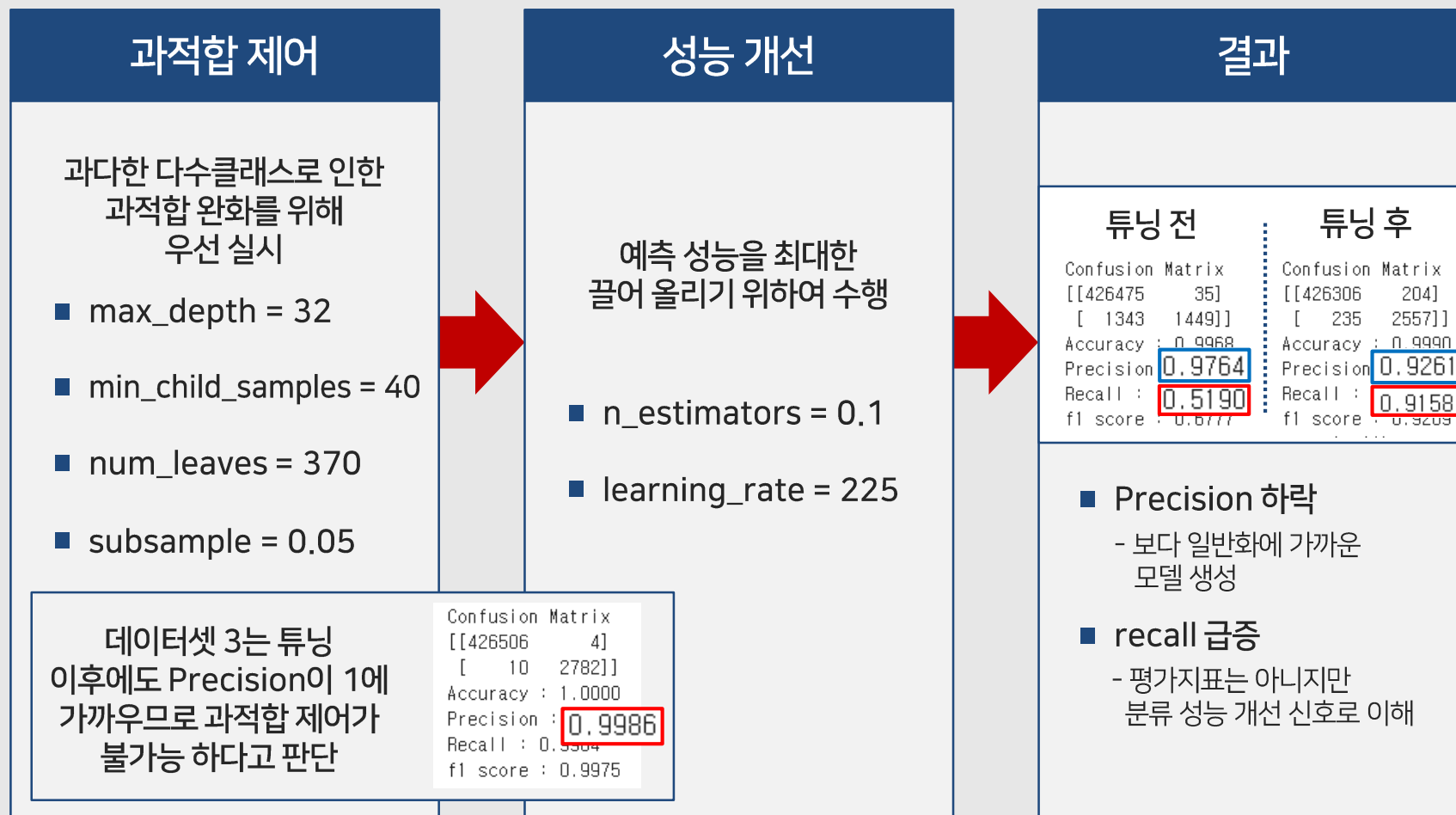


Leaf Wise

- 최대 손실 값 leaf node를 찾아 분기
- 과적합에 취약
- 예측 오류 손실 최소화
- 소수 클래스 분류에도
강점 보유

하이퍼 파라미터 튜닝 프로세스

일반화, 분류 역량 향상을 목표로 모델 개선

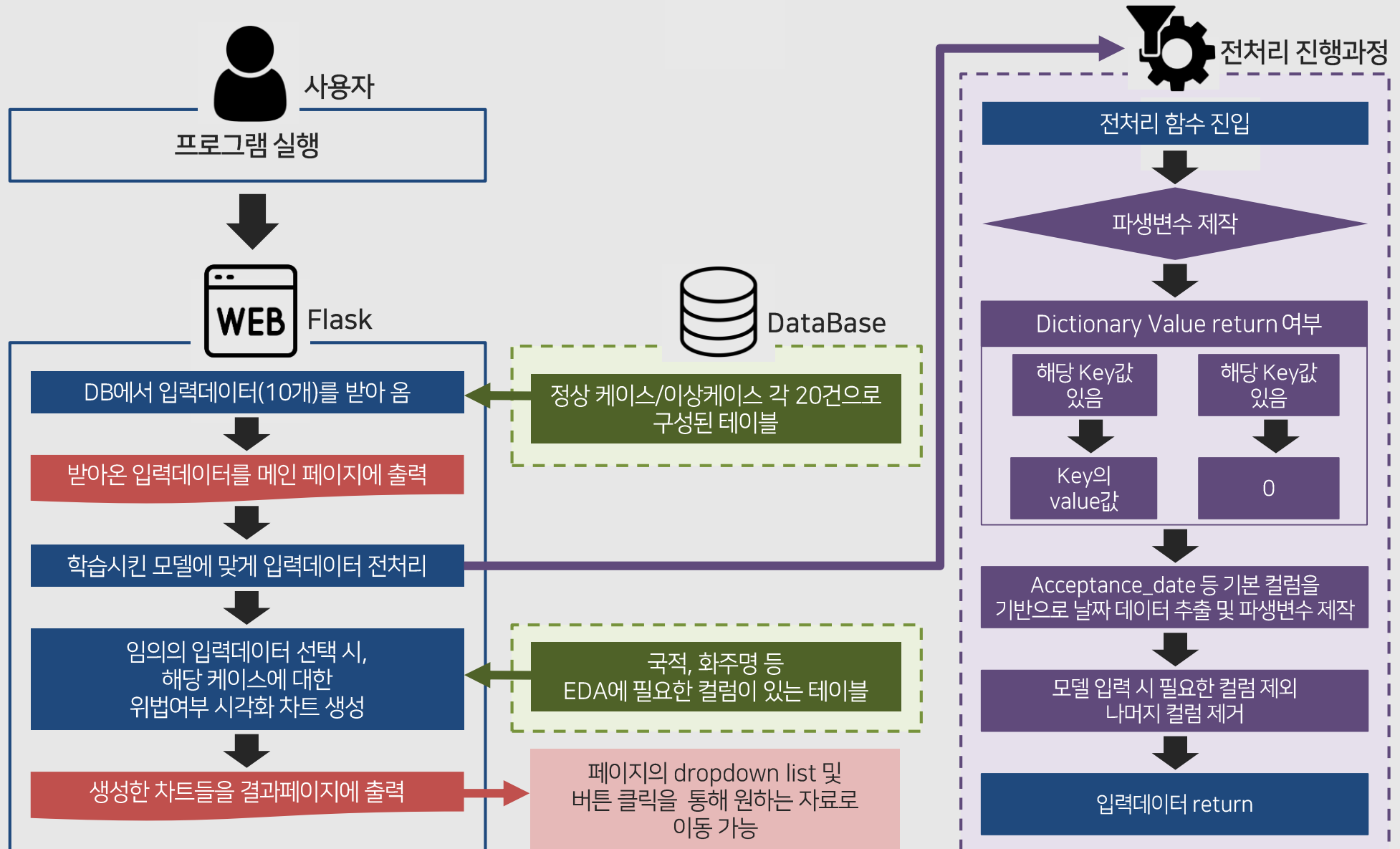


4

서비스 시연 및 목표

1 시스템 구현도

05 서비스 시연 및 목표



127.0.0.1:5000

NA1
팀원 구성

예상일
안아련
이브람
전정재

샘플케이스 선택
10개의 화물 중 하나를 골라 화물 검사하기 버튼을 누르세요

위법률 검사 대상 데이터

새 데이터 불러오기
화물 검사하기

신고번호: 762117/060317/0100606	신고번호: 762117/140217/0100475	신고번호: 762314/030217/0000064	신고번호: 762117/230317/0100747	신고번호: 762117/230317/0100747
신고번호: 762118/260417/0100105	신고번호: 762117/230317/0100747	신고번호: 762117/060317/0100606	신고번호: 762118/260417/0100105	신고번호: 762314/110117/0000017

오류 5:20
2021-05-06

클래스 불균형 극복 대안 요약 / 평가지표의 의의

소수 클래스에 대한 효과적 분류 & 프로젝트의 취지에 가장 적합한 정보 제공



계통 추출법 적용

- 클래스 비율 유지 & 왜곡 최소화
= 소수 클래스, 데이터의
본질적 의미 보존

평가지표로 Precision 사용



LightGBM 선택

- 가장 효과적으로 소수 클래스를
분류 가능한 알고리즘 활용

본 서비스에서 Precision이 지니는 의미

■ Precision

= 위법물이 존재할 것이라고 예측하여
개봉한 컨테이너에서 실제로 위법물이 적발될 확률

■ Recall

= 컨테이너를 열어보지는 않았지만
실제로 위법물이 포함된 컨테이너를
모델에서 위법물이라고 예측할 확률

모든 컨테이너를 개봉하여 위법물 확인은 불가능



■ Precision : 가장 적합한 지표

■ Recall : 정확한 정보 전달 불가 (평가 사용 불가)

서비스 최종 목표

예측 정확도 & 신뢰도 모두를 달성하는 서비스

Precision High	Recall High	: 위법물과 위법물이 아닌 화물을 정확히 예측하는 서비스
Precision High	Recall Low	: 위법물 예측이 잘 이뤄지지 않았지만 예측한 위법물에 대해선 정확도가 높은 서비스
Precision Low	Recall High	: 위법물 예측은 해냈지만 예측한 위법물에 위법물이 아닌 화물도 섞여 있는 서비스
Precision Low	Recall Low	: 위법물과 위법물이 아닌 화물 예측을 하지 못하는 서비스



위법물을 정확히 예측, 해당 화물 검사 시 반드시 위법물이 있는 서비스

- Recall을 사전 평가지표로 사용하지 않지만 사후 서비스 점검 보조 지표로 활용
- 향후 새로운 데이터 업데이트 시 지속적인 재학습을 통해 목표 달성
- 만약 재학습으로 목표 달성에 실패했을 시 Threshold 조정을 통한 개선 시도