**Feature Selection Using Logistic Regression**

|  |
| --- |
| Semiconductor Manufacturing Process Data) |

**응용통계학과**

**주건재**

|  |
| --- |
|  |

<  목    차  >

1 서론 1

1.1분석개요 1

1.2데이터소개 1

2 본론 2

2.1 분석 플로우 2

2.2 EDA 및 전처리 4

2.3모델링 5

2.4평가 6

2.5 분석 및 토의

3 결론 7

3.1 연구의 결론 및 한계 7

4 REFERENCES 8

5 Appendix 9

5.1 CODE 9

1. 서론
2. 분석개요

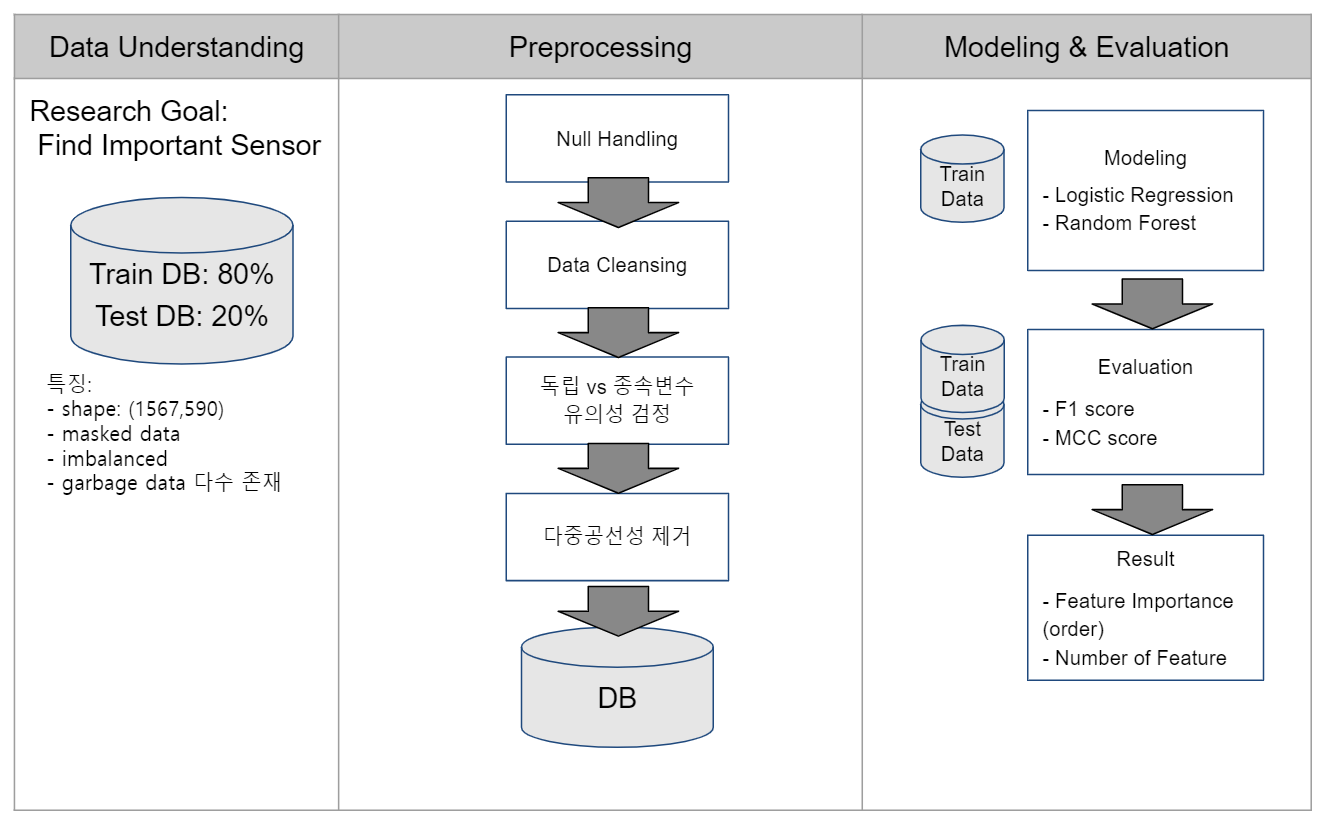
반도체의 두께는 수 나노미터의 단위로 만들어 진다. 이에 따라 반도체 웨이퍼를 만드는 공정 또한 아주 작은 단위의 작업이다. 이러한 작은 단위의 공정에서는 센서를 사용하여 데이터를 수집하는 것이 보통이다. 수집된 데이터는 가공 후 분석 및 머신러닝을 통해 통계적 해석과 새로운 데이터에 대한 예측을 하는데 주로 쓰여진다.

본 연구에서는 분석을 통해 어떤 센서의 데이터가 불량품을 판별하는데 중요한 역할을 하는지 볼 것이다. Logistic Regression 모델을 사용하여 Intrinsic 하고 Model-specific한 분석을 하고 Random Forest 모델을 사용하여 Post-hoc하고 Model-agnostic한 분석을 진행하겠다. 연구의 핵심은 센서 데이터와 불량여부의 인과관계를 밝히기 위함이 아닌 모델링을 통해 불량 여부를 예측하는 개별 센서 데이터의 중요도를 보는 것이다.

1. 데이터소개

SECOM(Semiconductor Manufacturing) dataset은 McCann et al이 re-experimentation용도로 공개한 데이터로 uci 저장소[[1]](#endnote-1)에서 제공하고 있다. 데이터는 590의 센서 데이터 변수, 1개의 이진분류 종속변수(양품:1/불량:0), 만들어진 시간변수 1개로 이루어져 있으며 총 1567개의 샘플을 포함하고 있다. 센서의 정보는 마스크 되어있으며 1부터590으로 번호로 명명 되어있다. 1567개 중 불량은 104개로 양품과 불량품의 비율은 14:1의 불균형을 보이고 있다. 데이터 중 80%를 train 데이터로 나머지를 test 데이터로 사용할 것이다.

1. 본론
   1. 분석 플로우



<표1> 전체 분석 플로우

분석은 <표1>과 같은 플로우로 진행된다. 분석의 목표는 로지스틱 회귀를 통해 intrinsic한 분석법으로 Feature Importance를 보는 것이다. 이 분석을 다른 모델을 사용한 분석과 비교하기 위해 랜덤포레스트를 이용한 post-hoc 분석법으로 진행할 것이다.

* 1. EDA 및 전처리

1. 결측치 처리

40~50%:4개, 50~60%: 4개, 60~70%:16개, 80~90%:4개, 90~100%:4개 결측 되어있다. 이중 20%이상 결측 된 32개의 변수는 사용하지 않는다. 20%이하의 결측치에 대해서는 각 변수의 평균값으로 채웠다.

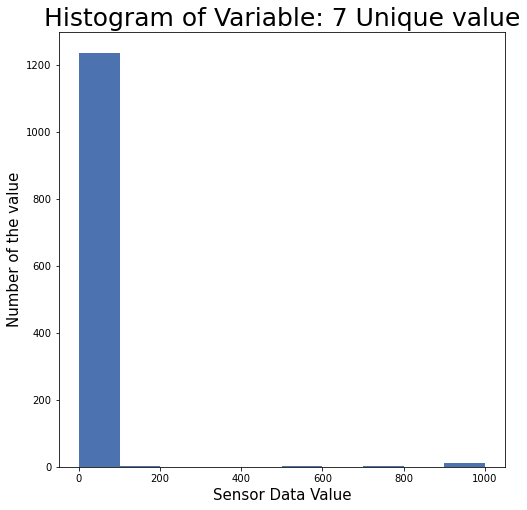
1. Unique Value

유일한 값으로만 이뤄진 변수가 충 16개, 유일한 값과 null 값으로 이뤄진 값이 총 100개이다. 마스크가 된 데이터라 도메인 지식으로 사용할 수 있는 방법이 없어 116개의 모든 변수에 대해 제거하기로 한다.

1. 변수 타입

Train 데이터 샘플 1253 중 40개의 변수가 100개 이하의 유니크 값을 갖는다. 이는 변수의 타입을 결정하는 과정 중 100개 이하의 값들 중에서 그림[1]과 같은 의미 없는 데이터가 다수 존재하여 제거한다.

100개 이상의 유니크 값을 갖는 변수는 연속형으로 유니크 값이 샘플에 비해 적은 이유는 대개 최대와 최소값의 범위가 작고 소수점 둘째자리 까지만 측정 되어있기 때문이다.

<그림1>

1. 독립 vs 종속 변수 유의성 검정

모수적 방법 (T-test)

T-test를 하기위해 정규성 검정을 샤피로-윌크 검정(Shapiro–Wilk test)과 콜모고로프-스미르노프 검정(Kolmogorov-Smirnov test)를 각각 해본 결과 p-value <<0.01로 정규성을 만족한다는 Null Hypothesis를 기각하지 못했다. 즉 정규성이 있다 보기 힘들어 T-test를 사용하지 않고 비모수적 방법을 사용하였다.

비모수적 방법 (Wilcoxon Rank sum 2 test)

Wilcoxon rank sum test 유의수준 0.05 이상의 변수 331개를 제거하였다. Test는 종속 변수 클래스(0,1)로 하나의 변수에서 두개의 집단으로 나누고 “두 집단의 평균이 같다.” 라는 null hypothesis에 대한 유의값을 계산해준다. 두 집단의 평균이 같다면 해당 독립변수는 종속변수에 대한 설명력이 없다고 판단하여 제거했다.

1. 다중 공선성 제거

다중 공선성은 아래와 같은 방법을 모든 변수의 VIF가 10이하가 될 때까지 반복하여 제거했다.

1. 모든 변수의 VIF 계산
2. 가장 VIF가 높은 변수 제거

이 과정을 총 49회 반복하여 변수 49개를 제거하였다.

* 1. 모델링

1. Logistic Regression

로지스틱 회귀는 종속변수가 이진분류일 때 주로 사용되며 각 독립변수()의 계수()의 크기가 해당 변수가 종속변수를 결정하는데 기여하는 중요도를 대변한다.

Logistic regression을 사용하여 모델링을 하면 계수의 추정치와 CI(Confidence Interval)을 계산할 수 있어 해당 모델에 국한되긴 하지만 Intrinsic한 분석이 가능한 장점이 있다. 머신러닝에서는 white-box 모델이라 부르며 분석이 용이하다.

1. Random Forest

Tree 모델의 앙상블로 이루어진 모델로 다중 공선성 및 독립변수가 많은 데이터에 유용하며 많은 상황에서 성능이 좋은 모델이다. 로지스틱 회귀와 다르게 black-box 모델로 불리며 어떠한 변수들이 종속변수를 결정하는데 결정적 기여를 했는지 알기가 직관적으로는 어렵다. 본 데이터는 독립변수가 많기 때문에 MDI(Mean Decrease in Impurity) 보다는 Permutation Importance를 사용해서 독립변수의 중요도를 볼 것이다.

* 1. 평가

1. Confusion-Matrix

종속변수의 클래스가 두개인 이진 분류 모델링이다. Confusion-Matrix는 분류의 결과를 한눈에 보기 쉬우며 FP-Rate과 MCC 평가 기준 모두 Confusion-Matrix를 기반으로 한다.

|  |  |  |  |
| --- | --- | --- | --- |
| CONFUSION-MATRIX | | PREDICTIVE VALUE | |
| Positive (1) | Negative (0) |
| ACTUAL  VALUE | Positive (1) | TP | FN |
| Negative (0) | FP | TN |
|  |  | 1: 양품 0: 불량품 | |

TP: 양품을 양품으로 예측  
FN: 양품을 불량품으로 예측  
FP: 불량품을 양품으로 예측  
TN: 불량품을 불량품으로 예측

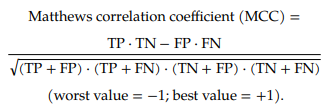
1. FP-Rate

본 데이터는 양품이 대다수인 데이터 셋이며 상황상 불량품을 양품으로 예측하는 것은 치명적인 비즈니스 로스로 돌아온다. 즉, 불량품을 잘 걸러내며 FP를 줄이는 것이 중요한 평가기준이 된다. FP-Rate가 이에 맞는 평가가 된다.

FPR 이 낮을수록 좋은 모델이며 값은 0과1사이에 존재한다.

1. Matthews correlation coefficient

MCC는 불균형 데이터에 주로 쓰이는 평가 방법으로 1에 가까울수록 완벽한 예측, 0에 가까울수록 랜덤한 예측, -1에 가까울수록 반대되는 예측을 하는 것이다.

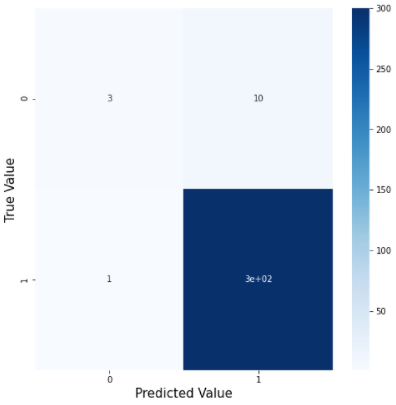
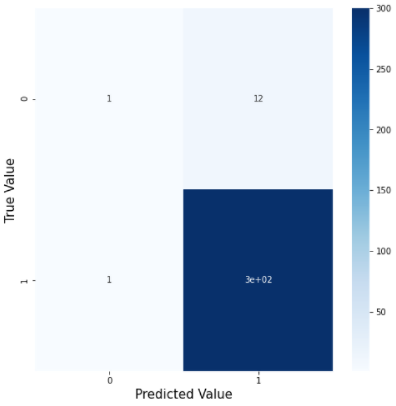


* 1. 분석결과 및 토의

1. 모델링 결과

|  |  |  |
| --- | --- | --- |
|  | F1 | mcc |
| Logistic Regression | 0.96 | 0.18 |
| Random Forest | 0.96 | 0.40 |

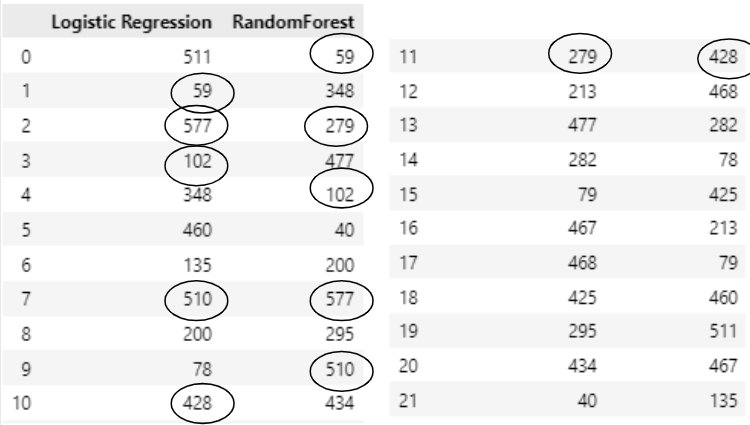
<표2> 모델링 결과표



<그림2> Logistic Regression Confusion-Matrix <그림3> Random Forest Confusion-Matrix

Mcc점수를 보면 두 모델 모두 Negative(불량품)을 분류하는 성능이 떨어진다. Confusion-Matrix를 보면 불량품의 샘플이 너무 적어서 mcc 점수가 민감하다. 두 모델의 결과 정답 차이는 2개밖에 안 나지만 mcc 점수는 크게 변한다. 이는 oversampling의 필요성을 시사한다.

1. Feature Importance



<표3> 모델에 따른 변수 중요도 순위

위 표는 두 모델의 결과를 바탕으로 변수 중요도를 중요한 순서부터 나열한 것이다. 동근 원 안의 변수번호들은 두 모델 모두의 경우 상위에 위치한 변수들이다. 인과 관계를 따지는 문제가 아니기에 저 번호의 센서들이 담당하는 공정이 웨이퍼 공정에 중요한 부분이라고 할 수 없다. 허나 저 센서들의 데이터 수집의 성능을 높인다면 보다 더 불량품을 걸러내는데 용이 할 것이다.

1. 결론

연구의 결론 및 한계

데이터가 마스크 되어 있어 도메인 지식을 적용하기 어렵고 센서 데이터끼리 공선성이 높고 그 수 많아 데이터 전처리에 어려움이 있다. Munirathinam와 Ramadoss의 “Predictive models for equipment fault detection in the semiconductor manufacturing process” (2016)를 보면 SME(Subject Matter Expert), 즉 도메인지식을 적용하여 모델링의 성능을 향상시켰다.

또 다른 한계점은 데이터를 Oversampling 하는 이론적 지식과 테크닉의 부족이다. 너무 적은 수의 negative데이터로 인해 좋은 모델링과 평가를 하기 힘들었다. 추후에는 조금 더 미시적인 전처리와 데이터에 맞는 oversampling을 사용하여 데이터를 처리하고 더욱 여러가지 평가지표와 모델을 사용하여 분석하면 더 좋은 결과가 도출 될것이다.

1. REFERENCES

[1] Cao, C., Chicco, D., & Hoffman, M. M. (2020). The MCC-F1 curve: a performance evaluation technique for binary classification. *arXiv preprint arXiv:2006.11278*.

[2] Kim, J., Han, Y., & Lee, J. (2016). Data imbalance problem solving for smote based oversampling: Study on fault detection prediction model in semiconductor manufacturing process. *Advanced Science and Technology Letters*, *133*, 79-84.

[3] Munirathinam, S., & Ramadoss, B. (2016). Predictive models for equipment fault detection in the semiconductor manufacturing process. *IACSIT International Journal of Engineering and Technology*, *8*(4), 273-285.

[4] Kerdprasop, K., & Kerdprasop, N. (2010, March). Feature selection and boosting techniques to improve fault detection accuracy in the semiconductor manufacturing process. In *World Congress on Engineering 2012. July 4-6, 2012. London, UK.* (Vol. 2188, pp. 398-403). International Association of Engineers.

1. APPENDIX
   1. CODE

#!/usr/bin/env python

# coding: utf-8

# for basic operations

import numpy as np

import pandas as pd

from scipy import stats

from sklearn.model\_selection import train\_test\_split

# for visualizations

import matplotlib.pyplot as plt

import seaborn as sns

# for preprocessing

from sklearn.impute import KNNImputer

from sklearn.preprocessing import RobustScaler

# model

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.svm import SVC

from sklearn.naive\_bayes import GaussianNB

# evaluation

from sklearn.metrics import f1\_score, matthews\_corrcoef,confusion\_matrix,plot\_confusion\_matrix

# feature importance

from sklearn.inspection import permutation\_importance

# to avoid warnings

import warnings

warnings.filterwarnings('ignore')

warnings.warn("this will not show")

# In[2]:

class FileController:

def readCsv(self,path):

return pd.read\_csv(path)

# In[3]:

class PreprocessController:

def nullKnnImputer(self,df\_train,df\_test):

imputer = KNNImputer()

imputer.fit(df\_train)

return pd.DataFrame(imputer.transform(df\_train), columns = df\_train.columns),pd.DataFrame(imputer.transform(df\_test), columns = df\_test.columns)

def dropCol(self,df\_train,df\_test,col):

return df\_train.drop(col,axis=1),df\_test.drop(col,axis=1)

# In[4]:

class EdaController:

'''

데이터 프레임을 설명에 쓰이는 클래스.(does not modify df)

'''

def summary(self,df, pred=None):

obs = df.shape[0]

Types = df.dtypes

Counts = df.apply(lambda x: x.count())

Min = df.min()

Max = df.max()

Uniques = df.apply(lambda x: x.unique().shape[0])

Nulls = df.apply(lambda x: x.isnull().sum())

print('Data shape:', df.shape)

if pred is None:

cols = ['Types', 'Counts', 'Uniques', 'Nulls', 'Min', 'Max']

st = pd.concat([Types, Counts, Uniques, Nulls, Min, Max], axis = 1, sort=True)

st.columns = cols

print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\nData Types:')

print(st.Types.value\_counts())

print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

return st

def imbalancedVisual(self,ser):

labels = ['Pass', 'Fail']

size = ser.value\_counts()

colors = ['blue', 'green']

explode = [0, 0.1]

plt.style.use('seaborn-deep')

plt.rcParams['figure.figsize'] = (8, 8)

plt.pie(size, labels =labels, colors = colors, explode = explode, autopct = "%.2f%%", shadow = True)

plt.axis('off')

plt.title('Target: Pass or Fail', fontsize = 20)

plt.legend()

plt.show()

# plt.bar(2,ser.value\_counts())

ser.value\_counts().plot(kind="bar", color=['b','g'])

plt.show()

def checkUniqueValueColWithRawData(self,df,fillNull=True):

temp = df.apply(lambda x: x.unique().shape[0])

temp2 = df.fillna(method='ffill').apply(lambda x: x.unique().shape[0])

print('1 unique vaule:',len(temp[temp.isin([1])]),'개')

print('1 unique vaule + null :',len(temp2[temp2.isin([1])]),'개')

if fillNull:

return list(temp2[temp2.isin([1])].index)

else:

return list(temp[temp.isin([1])].index)

def checkNull(self,df,percent=None):

'''

return list of index which has greater missing percentage if percent value is passed.

'''

missPercentTable = df.isnull().sum()/df.shape[0]\*100

for n in range(10,100,10):

print(f'missing {n}~{n+10}%: {len([i for i,p in enumerate(missPercentTable) if p>=n and p<n+10 ])}개')

if percent:

return [i for i,p in missPercentTable.iteritems() if p<percent and p>=10 ],[i for i,p in missPercentTable.iteritems() if p>=percent ]

def correlation(self,dataset, threshold):

col\_corr = set() # Set of all the names of correlated columns

corr\_matrix = dataset.corr()

for i in range(len(corr\_matrix.columns)):

for j in range(i):

if abs(corr\_matrix.iloc[i, j]) > threshold: # we are interested in absolute coeff value

colname = corr\_matrix.columns[i] # getting the name of column

col\_corr.add(colname)

return col\_corr

# In[5]:

class ModelingController:

def logisticEvaluate(self,train\_df, test\_df, train\_target, test\_target):

model\_order = ['Logistic Regression','RandomForest']

lr\_score = []

rf\_score = []

lr\_coef = []

rf\_coef = []

scaler = RobustScaler()

scaler.fit(train\_df)

train\_std = pd.DataFrame(scaler.transform(train\_df), columns=train\_df.columns)

test\_std = pd.DataFrame(scaler.transform(test\_df), columns=test\_df.columns)

y\_train\_true = train\_target.values.ravel()

y\_test\_true = test\_target.values.ravel()

logreg = LogisticRegression()

rf = RandomForestClassifier(n\_estimators=5,random\_state=0,max\_depth=12,n\_jobs=-1)

for i,clf in enumerate([logreg,rf]):

clf.fit(train\_std, train\_target.values.ravel())

y\_train\_pred = clf.predict(train\_std)

y\_test\_pred = clf.predict(test\_std)

f1score\_train = f1\_score(y\_train\_true, y\_train\_pred, average='micro')

f1score\_test = f1\_score(y\_test\_true, y\_test\_pred, average='micro')

mccscore\_train=matthews\_corrcoef(y\_train\_true, y\_train\_pred)

mccscore\_test=matthews\_corrcoef(y\_test\_true, y\_test\_pred)

cm =confusion\_matrix(y\_test\_true,y\_test\_pred)

tn,fp,fn,tp = cm.ravel()

fpr = fp/(fp+tn)

sns.heatmap(cm,annot=True,cmap='Blues')

plt.show()

if i ==0:

lr\_score.extend([f1score\_train,f1score\_test,mccscore\_train,mccscore\_test,fpr])

t1\_df = pd.DataFrame([list(map(abs,clf.coef\_))[0],clf.coef\_[0]],columns=list(train\_std.columns))

t1\_df = t1\_df.T.sort\_values(by=0,ascending=False).drop([0],axis=1)

else:

rf\_score.extend([f1score\_train,f1score\_test,mccscore\_train,mccscore\_test,fpr])

imp = clf.feature\_importances\_

t3\_df = pd.DataFrame([list(map(abs,imp)),imp],columns=list(train\_std.columns))

t3\_df = t3\_df.T.sort\_values(by=0,ascending=False).drop([0],axis=1)

del y\_train\_pred, y\_test\_pred

imp\_result = pd.DataFrame([t1\_df.index,t3\_df.index],index=model\_order).transpose()

# display(imp\_result)

result = pd.DataFrame([lr\_score,rf\_score],index=model\_order,columns=['f1\_train','f1\_test','mcc\_train','mcc\_test','fpr'])

display(result)

return imp\_result ,result

# # Initailize Class

# In[6]:

#class initialize

fController = FileController()

eController = EdaController()

pController = PreprocessController()

mController = ModelingController()

# # Load Data Frame & Split Data

# In[41]:

# load csv file

rawData = fController.readCsv('./uci-secom.csv')

# In[42]:

X = rawData.iloc[:,:-1]

y = rawData.iloc[:,-1]

# 종속변수 1,0으로 변환

y=y.replace(1,0)

y=y.replace(-1,1)

#train test split

X\_train,X\_test,y\_train,y\_test = train\_test\_split(X,y,test\_size = 0.3,random\_state=0)

# # EDA

# # 1. Summary

# In[43]:

summaryTable = eController.summary(df=rawData)

display(summaryTable)

# In[44]:

# drop time

X\_train,X\_test = pController.dropCol(X\_train,X\_test,['Time'])

X\_train.shape,X\_test.shape

# ## 1.1 Imbalaced Target

# In[45]:

eController.imbalancedVisual(y)

# In[ ]:

# # 2. Null Handling

# ## 2.1 Null Check

# In[ ]:

# In[46]:

# missing value percentage

missingCol,dropCol = eController.checkNull(df=X\_train,percent = 20)

# 20% 이상 NUll 값이 존재하면 drop

# In[47]:

# drop

X\_train = X\_train.drop(dropCol,axis=1)

X\_test = X\_test.drop(dropCol,axis=1)

X\_train.shape,X\_test.shape

# ## 2.2 Fill Null value

# ### 2.2.1 KNN Imputer (App.1)

# ### 2.2.2 mean value

# In[48]:

X\_train = X\_train.fillna(X\_train.mean())

X\_test = X\_test.fillna(X\_test.mean())

# mController.logisticEvaluate(X\_train,X\_test,y\_train,y\_test)

# In[49]:

full\_train = X\_train

full\_test = X\_test

# In[50]:

full\_train.shape

# ## 1.2 Number of Unique Value

# In[51]:

uniqueColWithNull = eController.checkUniqueValueColWithRawData(X\_train,True)

len(uniqueColWithNull)

# In[52]:

# unique vaule + null drop

X\_train = X\_train.drop(uniqueColWithNull,axis=1)

X\_test = X\_test.drop(uniqueColWithNull,axis=1)

X\_train.shape,X\_test.shape

cleansed\_train = X\_train

cleansed\_test = X\_test

# In[53]:

# unique value histogram

temp = pd.Series(X\_train.nunique())

print(temp)

print()

plt.hist(temp,bins = 12,label='Num of Unique values')

plt.show()

# ### Drop Unique Val under 100

#

# In[57]:

thCol = [i for i,val in X\_train.nunique().iteritems() if val <=100]

X\_train,X\_test = pController.dropCol(X\_train,X\_test,thCol)

print(X\_train.shape, X\_test.shape)

a,b =mController.logisticEvaluate(X\_train,X\_test,y\_train,y\_test)

reduced\_train = X\_train

reduced\_test = X\_test

# ### Ranksum 2 test

# In[58]:

ranksum\_col = []

tempData = X\_train.merge(y\_train,left\_index=True,right\_index=True)

passData = tempData[tempData['Pass/Fail']==1]

failData = tempData[tempData['Pass/Fail']==0]

colList = list(X\_train.columns)

for i in colList:

pValue = stats.ranksums(passData[i],failData[i]).pvalue

if pValue>0.05:

ranksum\_col. append(i)

print(len(ranksum\_col))

X\_train,X\_test = pController.dropCol(X\_train,X\_test,ranksum\_col)

t\_train = X\_train

t\_test = X\_test

X\_train.shape,X\_test.shape

# In[60]:

mController.logisticEvaluate(X\_train,X\_test,y\_train,y\_test)

# ## 3. Multicollinearity (VIF)

# In[61]:

#calculate Variance Inflation Factor

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

condition = 1

temp\_train = X\_train

vifDropCol = []

while condition !=0:

vif\_scores = pd.DataFrame()

vif\_scores["Attribute"] = temp\_train.columns

# calculating VIF for each feature

vif\_scores["VIF Scores"] = [variance\_inflation\_factor(temp\_train.values, i) for i in range(len(temp\_train.columns))]

condition = (vif\_scores['VIF Scores']>10).sum()

droppingCol = vif\_scores['Attribute'][vif\_scores['VIF Scores'].idxmax()]

vifDropCol.append(droppingCol)

temp\_train =temp\_train.drop([droppingCol],axis=1)

display(vif\_scores)

# In[62]:

X\_train ,X\_test= pController.dropCol(X\_train,X\_test,vifDropCol)

f\_train,f\_test = X\_train,X\_test

# In[63]:

mController.logisticEvaluate(X\_train,X\_test,y\_train,y\_test)

# In[64]:

sns.pairplot(temp\_train)

1. Data source: https://archive.ics.uci.edu/ml/datasets/SECOM [↑](#endnote-ref-1)