```
#LAB1 Semiconductor Data Analysis-109AB8037 鄭美中
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

#create test set
from sklearn.model_selection import train_test_split

#logistic regression model
from sklearn.linear_model import LogisticRegression

#SVM model
from sklearn import svm

#KNN imputation
from sklearn.impute import KNNImputer

#normalizer
from sklearn.preprocessing import Normalizer

#variance threshold
from sklearn.feature_selection import VarianceThreshold

# import the metrics class
from sklearn import metrics

#load input CSV file
uci_secom = pd.read_csv('C:/uci-secom.csv')
```

```
#Result and Feature array
uni_target = uci_secom[['Pass/Fail']]
uni_data = uci_secom.drop(['Pass/Fail'], axis=1)

#data type and calculate its number
type_dct = {str(k): len(list(v)) for k, v in uni_data.groupby(uni_data.dtypes, axis=1)}
type_dct

#find object header
uni_data.select_dtypes(include=['object'])

#time feature not needed, numeric only
uni_data_numeric = uni_data.drop(['Time'], axis=1)



#make sure time stamp is removed
#print(uni_data_numeric)

#split data into training and testing using stratification
X_train, X_test, y_train, y_test = train_test_split(uni_data_numeric, uni_target, test_size=0.2,
random_state=33, stratify=uni_target)

# convert to pandas dataframe
X_train = pd.DataFrame(X_train, columns=uni_data_numeric.columns)
X_test = pd.DataFrame(X_test, columns=uni_data_numeric.columns)
y_train = pd.DataFrame(y_train, columns=uni_target.columns)
y_test = pd.DataFrame(y_test, columns=uni_target.columns)
```

```
#threshold setting to 0.3, applied to training data
def percentna(dataframe, threshold):
    columns = dataframe.columns[(dataframe.isna().sum()/dataframe.shape[1])>threshold]
    return columns.tolist()

#column drop
na_columns = percentna(X_train, 0.3)
X_train_dense = X_train.drop(na_columns, axis=1)
X_test_dense = X_test.drop(na_columns, axis=1)
n_features1 = X_train_dense.shape[1]
print(f'Missing value Thresholding: removing {len(na_columns)} features, there are {n_features1} features left.')


#fill up missing with knn
imputer = KNNImputer()
imputer.fit(X_train_dense)

X_train_imp = pd.DataFrame(imputer.transform(X_train_dense), columns = X_train_dense.columns)
X_test_imp = pd.DataFrame(imputer.transform(X_test_dense), columns = X_test_dense.columns)

# display the number feature to make sure if knn filling works
na_columns = percentna(X_train_imp, 0.01)
X_train_imp2 = X_train_imp.drop(na_columns, axis=1)
X_test_imp2 = X_test_imp.drop(na_columns, axis=1)
n_features1 = X_train_imp2.shape[1]
print(f'After imputing with KNN: removing {len(na_columns)} features, there are {n_features1} features left.')
```

```
#check pairwise correlation for 0.9
def correlation(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


corr_features = correlation(X_train_imp2, 0.9)
X_train_corr = X_train_imp2.drop(corr_features, axis=1)
X_test_corr = X_test_imp2.drop(corr_features, axis=1)
n_features2 = X_train_corr.shape[1]
print(f'Pairwise correlation check: removing {len(corr_features)} features, there are {n_features2}
features left.')


# feature output correlation
def corrwith_target(dataframe, target, threshold):
    cor = dataframe.corr()
    #Correlation with output variable
    cor_target = abs(cor[target])
    #Selecting non correlated features
    relevant_features = cor_target[cor_target<threshold]
    return relevant_features.index.tolist()[:-1]
# in order to find the correlation with target, I have to add target as a column to X_train_corr
dummy_train = X_train_corr.copy()
```

```
dummy_train['target'] = y_train

corrwith_cols = corrwith_target(dummy_train, 'target', 0.05)

X_train_corw = X_train_corr.drop(corrwith_cols, axis=1)

X_test_corw = X_test_corr.drop(corrwith_cols, axis=1)

n_features3 = X_train_corw.shape[1]

print(f'Feature and output correlation: After removing {len(corrwith_cols)} features, there are
{n_features3} features left.')


# remove low variance column

normalizer = Normalizer()

normalizer.fit(X_train_corw)


X_train_nrm = pd.DataFrame(normalizer.transform(X_train_corw), columns = X_train_corw.columns)

X_test_nrm = pd.DataFrame(normalizer.transform(X_test_corw), columns = X_test_corw.columns)


selector = VarianceThreshold()

selector.fit(X_train_nrm)


mask = selector.get_support()

columns = X_train_nrm.columns

selected_cols = columns[mask]

n_features4 = len(selected_cols)

print(f'Removing low variance column: number of remaining features: {n_features4}')



# LogisticRegression classifier


# instantiate the model (using the default parameters)

logreg = LogisticRegression()
```

```
# train the model
logreg.fit(X_train_nrm,y_train)

#predict Y with classifier
y_pred=logreg.predict(X_test_nrm)
print(y_pred)



#confusion matrix calculation
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
print(cnf_matrix)

#Accuracy, Preciion, and Recall (LogisticRegression)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))

#Rename X, Y training and testing for SVM
X_train2=X_train_nrm
y_train2=y_train
X_test2=X_test_nrm
y_pred2=y_pred

#Create a SVM classifier with linear kernal
linearC=svm.SVC(kernel='linear')

#train the model
linearC.fit(X_train2, y_train2)
```

```
#predict y with classifier
y_pred2=linearC.predict(X_test2)
print(y_pred2)

#confusion matrix calculation for SVM
cnf_matrix = metrics.confusion_matrix(y_test, y_pred2)
print(cnf_matrix)

#Accuracy, Preciion, and Recall (SVM)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred2))
print("Precision:",metrics.precision_score(y_test, y_pred2))
print("Recall:",metrics.recall_score(y_test, y_pred2))
```