### IMPORT LIBRARIES

##### Built in libraries

import os

###### Use pip or anaconda to install

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

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

from sklearn.preprocessing import KBinsDiscretizer, MinMaxScaler, normalize

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, confusion\_matrix, classification\_report

from sklearn.decomposition import PCA

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear\_model import LogisticRegression, LinearRegression, Lasso

from sklearn.neighbors import KNeighborsClassifier

from sklearn.neural\_network import MLPClassifier

from sklearn.svm import SVC

from sklearn.naive\_bayes import GaussianNB

### READ DATA

##### Needs to be ran from the project directory

csv\_measurements = os.path.join(os.getcwd(), 'datasets', 'measurements.csv')

df\_measurements = pd.read\_csv(csv\_measurements, *parse\_dates*=['measurement\_time'])

df\_measurements = df\_measurements.sort\_values(*by*=['measurement\_time'], *ascending*=[True])

csv\_failures = os.path.join(os.getcwd(), 'datasets', 'failures.csv')

df\_failures = pd.read\_csv(csv\_failures, *parse\_dates*=['failure\_time'])

df\_failures = df\_failures.sort\_values(*by*=['failure\_time'], *ascending*=[True])

### MERGE NEXT FAILURE TO MEASUREMENTS

df\_combined = pd.merge\_asof(

    df\_measurements,

    df\_failures,

*left\_on*='measurement\_time',

*right\_on*='failure\_time',

*by*='gadget\_id',

*direction*='forward',

)

### TRANSFORM COLUMNS

df\_combined['time\_to\_fail'] = df\_combined['failure\_time']-df\_combined['measurement\_time']

df\_combined['fail\_in\_1h'] = np.where(df\_combined['time\_to\_fail']<pd.Timedelta(*hours*=1), 1, 0)

### CALCULATE RUNNING MEASURES

df\_combined = df\_combined.reset\_index(*drop*=True)

df\_combined = df\_combined.sort\_values(*by*=['gadget\_id', 'measurement\_time'], *ascending*=[True, True])

df\_combined['temperature\_6h\_std'] = df\_combined.groupby('gadget\_id')['temperature'].rolling(6).std(*ddof*=0).reset\_index(*drop*=True)

df\_combined['pressure\_6h\_mean'] = df\_combined.groupby('gadget\_id')['pressure'].rolling(6).mean().reset\_index(*drop*=True)

### SPLIT TO TRAIN AND TEST

X = ['vibration\_y', 'pressure\_6h\_mean', 'temperature\_6h\_std']

y = 'fail\_in\_1h'

cols = X + [y]

df\_to\_split = df\_combined.copy()

df\_to\_split = df\_to\_split.dropna(*subset*=cols)

df\_to\_split = df\_to\_split.reset\_index(*drop*=True)

##### Create binary bins to

binner = KBinsDiscretizer(*n\_bins*=10, *encode*='onehot-dense', *strategy*='kmeans')

binner.fit(df\_to\_split[X])

arr\_bins= binner.transform(df\_to\_split[X])

df\_bins = pd.DataFrame(arr\_bins)

X = list(df\_bins.columns)

cols = X + [y]

df\_to\_split = pd.concat([df\_to\_split, df\_bins], *axis*=1)

df\_train = df\_to\_split[df\_to\_split['gadget\_id'].isin([1,2,3,4])].reset\_index(*drop*=True).copy()

df\_test = df\_to\_split[df\_to\_split['gadget\_id'].isin([5,6])].reset\_index(*drop*=True).copy()

print(*f*"Training data: {df\_train.shape}")

print(*f*"Test data: {df\_test.shape}")

### PREDICTION PARAMETERS

w0 = 1

w1 = 8

pos\_label = 1

### NEURAL NETWORK

nn = MLPClassifier(

*solver*='lbfgs',

*alpha*=1e-5,

*hidden\_layer\_sizes*=(10),

*random\_state*=1,

*max\_iter*=10000,

*activation*='relu',

*tol*=0.00001,

)

nn.fit(df\_train[X], df\_train[y])

df\_test['nn'] = nn.predict(df\_test[X])

### RANDOM FOREST MODEL

random\_forest = RandomForestClassifier(

*min\_samples\_leaf*=7,

*random\_state*=45,

*n\_estimators*=50,

*class\_weight*={0:w0, 1:w1}

)

random\_forest.fit(df\_train[X], df\_train[y])

df\_test['random\_forest'] = random\_forest.predict(df\_test[X])

### LOGISTIC REGRESSION MODEL

log\_regr = LogisticRegression(*class\_weight*={0:w0, 1:w1})

log\_regr.fit(df\_train[X], df\_train[y])

df\_test['log\_regr'] = log\_regr.predict(df\_test[X])

### LINEAR REGRESSION MODEL

lin\_regr = Lasso(*alpha*=0.1, *positive*=True)

lin\_regr.fit(df\_train[X], df\_train[y])

df\_test['lin\_regr'] = lin\_regr.predict(df\_test[X])

df\_test['lin\_regr'] = np.where(df\_test['lin\_regr']>=0.5,1,0)

### KNN MODEL

*def* knn\_weights(*knn\_y*):

    return np.where(*knn\_y*==1, w1, w0)

knn = KNeighborsClassifier(*weights*=knn\_weights)

knn.fit(df\_train[X], df\_train[y])

df\_test['knn'] = knn.predict(df\_test[X])

### SVM

svm = SVC(

*class\_weight*={0:w0, 1:w1},

*C*=1,

*random\_state*=42,

*kernel*='linear'

)

svm.fit(df\_train[X], df\_train[y])

df\_test['svm'] = svm.predict(df\_test[X])

### NAIVE BAYES

bayes = GaussianNB()

bayes.fit(df\_train[X], df\_train[y])

df\_test['bayes'] = bayes.predict(df\_test[X])

### PRINT RESULTS

model\_summary = []

models = ['random\_forest', 'log\_regr', 'lin\_regr', 'knn', 'nn', 'svm', 'bayes']

for m in models:

    print(*f*"\n-----------\n{m}")

    try:

        precision = precision\_score(df\_test['fail\_in\_1h'], df\_test[m], *zero\_division*=0, *pos\_label*=pos\_label)

        recall = recall\_score(df\_test['fail\_in\_1h'], df\_test[m], *pos\_label*=pos\_label)

        print(*f*"Precision: {precision}")

        print(*f*"Recall: {recall}")

        print(df\_test[m].value\_counts())

        model\_summary.append({

            'model': m,

            'precision': precision,

            'recall': recall

        })

    except:

        print("Can't calculate score")

#PRINT RESULT DATAFRAME

#print(df\_test[['gadget\_id', 'measurement\_time'] + cols + models].head(5))

#CREATE IMAGE FOR MODEL COMPARISON

df\_summary = pd.DataFrame(model\_summary)

x = np.arange(len(df\_summary['model']))

width = 0.35

fig, ax = plt.subplots()

rects1 = ax.bar(x - width/2, df\_summary['precision'], width, *label*='Precision')

rects2 = ax.bar(x + width/2, df\_summary['recall'], width, *label*='Recall')

# Add some text for labels, title and custom x-axis tick labels, etc.

ax.set\_ylabel('Result')

ax.set\_title('Precision and Recall by machine learning model')

ax.set\_xticks(x)

ax.set\_xticklabels(df\_summary['model'])

ax.legend()

fig.tight\_layout()

plt.savefig('img/results.png')