**Machine Learning**

**Project 1**

**Mercedes-Benz Greener Manufacturing**

**Made By:**

**Siddhartha Patra**

****

**Problem Statement Scenario:**

Since the first automobile, the Benz Patent Motor Car in 1886, Mercedes-Benz has stood for important automotive innovations. These include the passenger safety cell with the crumple zone, the airbag, and intelligent assistance systems. Mercedes-Benz applies for nearly 2000 patents per year, making the brand the European leader among premium carmakers. Mercedes-Benz cars are leaders in the premium car industry. With a huge selection of features and options, customers can choose the customized Mercedes-Benz of their dreams.

To ensure the safety and reliability of every unique car configuration before they hit the road, Daimler’s engineers have developed a robust testing system. As one of the world’s biggest manufacturers of premium cars, safety and efficiency are paramount on Daimler’s production lines. However, optimizing the speed of their testing system for many possible feature combinations is complex and time-consuming without a powerful algorithmic approach.

You are required to reduce the time that cars spend on the test bench. Others will work with a dataset representing different permutations of features in a Mercedes-Benz car to predict the time it takes to pass testing. Optimal algorithms will contribute to faster testing, resulting in lower carbon dioxide emissions without reducing Daimler’s standards.

**Source Code (Python):**

# Step1: Import the required libraries

# linear algebra

import numpy as np

# for data processing

import pandas as pd

# for dimensionality reduction

from sklearn.decomposition import PCA

# Step2: Read the data from train.csv

df\_train = pd.read\_csv('train.csv')

# let us understand the data

print('Size of training set: {} rows and {} columns'

.format(\*df\_train.shape))

# print few rows and see how the data looks like

df\_train.head()

# Step3: Collect the Y values into an arra

# seperate the y from the data as we will use this to learn as

# the prediction output

y\_train = df\_train['y'].values

# Step4: Understand the data types we have

# iterate through all the columns which has X in the name of the column

cols = [c for c in df\_train.columns if 'X' in c]

print('Number of features: {}'.format(len(cols)))

print('Feature types:')

df\_train[cols].dtypes.value\_counts()

# Step5: Count the data in each of the columns

counts = [[], [], []]

for c in cols:

typ = df\_train[c].dtype

uniq = len(np.unique(df\_train[c]))

if uniq == 1:

counts[0].append(c)

elif uniq == 2 and typ == np.int64:

counts[1].append(c)

else:

counts[2].append(c)

print('Constant features: {} Binary features: {} Categorical features: {}\n'

.format(\*[len(c) for c in counts]))

print('Constant features:', counts[0])

print('Categorical features:', counts[2])

# Step6: Read the test.csv data

df\_test = pd.read\_csv('test.csv')

# remove columns ID and Y from the data as they are not used for learning

usable\_columns = list(set(df\_train.columns) - set(['ID', 'y']))

y\_train = df\_train['y'].values

id\_test = df\_test['ID'].values

x\_train = df\_train[usable\_columns]

x\_test = df\_test[usable\_columns]

# Step7: Check for null and unique values for test and train sets

def check\_missing\_values(df):

if df.isnull().any().any():

print("There are missing values in the dataframe")

else:

print("There are no missing values in the dataframe")

check\_missing\_values(x\_train)

check\_missing\_values(x\_test)

# Step8: If for any column(s), the variance is equal to zero,

# then you need to remove those variable(s).

# Apply label encoder

for column in usable\_columns:

cardinality = len(np.unique(x\_train[column]))

if cardinality == 1:

x\_train.drop(column, axis=1) # Column with only one

# value is useless so we drop it

x\_test.drop(column, axis=1)

if cardinality > 2: # Column is categorical

mapper = lambda x: sum([ord(digit) for digit in x])

x\_train[column] = x\_train[column].apply(mapper)

x\_test[column] = x\_test[column].apply(mapper)

x\_train.head()

# Step9: Make sure the data is now changed into numericals

print('Feature types:')

x\_train[cols].dtypes.value\_counts()

# Step10: Perform dimensionality reduction

# Linear dimensionality reduction using Singular Value Decomposition of

# the data to project it to a lower dimensional space.

n\_comp = 12

pca = PCA(n\_components=n\_comp, random\_state=420)

pca2\_results\_train = pca.fit\_transform(x\_train)

pca2\_results\_test = pca.transform(x\_test)

# Step11: Training using xgboost

!pip install xgboost

import xgboost as xgb

from sklearn.metrics import r2\_score

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

x\_train, x\_valid, y\_train, y\_valid = train\_test\_split(

pca2\_results\_train,

y\_train, test\_size=0.2,

random\_state=4242)

d\_train = xgb.DMatrix(x\_train, label=y\_train)

d\_valid = xgb.DMatrix(x\_valid, label=y\_valid)

#d\_test = xgb.DMatrix(x\_test)

d\_test = xgb.DMatrix(pca2\_results\_test)

params = {}

params['objective'] = 'reg:linear'

params['eta'] = 0.02

params['max\_depth'] = 4

def xgb\_r2\_score(preds, dtrain):

labels = dtrain.get\_label()

return 'r2', r2\_score(labels, preds)

watchlist = [(d\_train, 'train'), (d\_valid, 'valid')]

clf = xgb.train(params, d\_train,

1000, watchlist, early\_stopping\_rounds=50,

feval=xgb\_r2\_score, maximize=True, verbose\_eval=10)

# Step12: Predict your test\_df values using xgboost

p\_test = clf.predict(d\_test)

sub = pd.DataFrame()

sub['ID'] = id\_test

sub['y'] = p\_test

sub.to\_csv('xgb.csv', index=False)

sub.head()

######################################################################

''' End '''

######################################################################