Project to Submit – Project 1

DESCRIPTION

Reduce the time a Mercedes-Benz spends on the test bench.

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 a 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 is the leader 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, the company’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 Mercedes-Benz’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 Mercedes-Benz’s standards.

Following actions should be performed:

* If for any column(s), the variance is equal to zero, then you need to remove those variable(s).
* Check for null and unique values for test and train sets.
* Apply label encoder.
* Perform dimensionality reduction.
* Predict your test\_df values using XGBoost.

# Write Up:

**Analysis Tasks to be performed:**

1. Data was already divided into two files for training and test dataset.
2. Both had equal number of rows
3. Table

   Description automatically generated
4. Training dataset as usual had y column which is the target column which was missing in test data set
5. Calendar

   Description automatically generatedTraining data set had 369 binary features, 8 features which have datatype = ‘object’ is most probably categorical features and 1 remaining feature is our target variable i.e. ‘y’.
6. Performing univariate analysis on categorical features, to get the insight out of it. Any feature that has very low variance as compared to other categorical features, will be removed
7. Found 12 columns which had no variance. We removed them.
   * + 1. Name = X11
       2. Name = X93
       3. Name = X107
       4. Name = X233
       5. Name = X235
       6. Name = X268
       7. Name = X289
       8. Name = X290
       9. Name = X293
       10. Name = X297
       11. Name = X330
       12. Name = X347
       13. No of columns which has zero variance = 12
8. Also checked for null, duplicate rows and unique values
9. Got the details of categorical columns and integers columns separatel
10. Then applied encoder
11. Then Summarize outcome (testing time) in training dataset (created box-plot to see how the data is spread)
12. On y, we plotted histogram & violin plot to see if there are any outliers.

Chart

Description automatically generated

1. Perform dimensionality reduction.
2. The methods at our disposal using linear algebra are:
3. Principal Components Analysis Singular Value Decomposition Non-Negative Matrix Factorization. Identified 6 main components.
4. Created the dataset to include only these components for further analysis
5. Before using XGBoost, checked the model with the following methods

* Logistic Regression
* KNN
* SVM
* Random Forest

1. None of the model was good. We tried XGBoost.
2. For the first time we got good >95% accuracy.
3. We further tested with different booster='dart', ‘gbliner’, ‘gbtree’. Gbtree gave the best result, and
4. We enhanced it further with K-fold.
5. We got predicted y with accuracy of 99.9%

# Code:

Mercedes-Benz Greener Manufacturing

DESCRIPTION

Reduce the time a Mercedes-Benz spends on the test bench.

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 a 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 is the leader 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, the company’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 Mercedes-Benz’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 Mercedes-Benz’s standards.

Following actions should be performed:

If for any column(s), the variance is equal to zero, then you need to remove those variable(s).

Check for null and unique values for test and train sets.

Apply label encoder.

Perform dimensionality reduction.

Predict your test\_df values using XGBoost.

The data set is already divided into train and test

import numpy as np

import pandas as pd

from datetime import datetime as dt

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

from matplotlib.pylab import rcParams

rcParams['figure.figsize'] = 15, 6

import warnings

warnings.filterwarnings('ignore')

# importing csv module

import csv

# csv file name

train\_df = pd.read\_csv(r'D:\OneDrive\Studies\AI - ML\Python\Examples\ML Pracs\train.csv')

# importing csv module

import csv

# csv file name

test\_df = pd.read\_csv(r'D:\OneDrive\Studies\AI - ML\Python\Examples\ML Pracs\test.csv')

train\_df.info()

test\_df.info()

train\_df.describe()

test\_df.describe()

print("Number of datapoints: ", train\_df.shape[0])

print("Number of features: ", train\_df.shape[1])

train\_df.head()

print("Number of datapoints: ", test\_df.shape[0])

print("Number of features: ", test\_df.shape[1])

test\_df.head()

dtype\_df = train\_df.dtypes.reset\_index()

dtype\_df.columns = ["feature name","dtypes"]

dtype\_df.groupby("dtypes").agg("count").reset\_index()

there are 369 binary features,

8 features which have datatype = ‘object’ is most probably categorical features and

1 remaining feature is our target variable i.e. ‘y’.

Performing univariate analysis on categorical features, to get the insight out of it.

Any feature that has very low variance as compared to other categorical features, will be removed

dtype\_df = test\_df.dtypes.reset\_index()

dtype\_df.columns = ["feature name","dtypes"]

dtype\_df.groupby("dtypes").agg("count").reset\_index()

Question 1:

If for any column(s), the variance is equal to zero, then you need to remove those variable(s).

Starting with train and then with test data

variance = pow(train\_df.drop(columns={'ID','y'}).std(),2).to\_dict()

null\_cnt = 0

for key, value in variance.items():

if(value==0):

print('Name = ',key)

null\_cnt = null\_cnt+1

print('No of columns which has zero variance = ',null\_cnt)

train\_df = train\_df.drop(columns={'X11','X93','X107','X233','X235','X268','X289','X290','X293','X297','X330','X347'})

train\_df.shape

variance = pow(test\_df.drop(columns={'ID'}).std(),2).to\_dict()

null\_cnt = 0

for key, value in variance.items():

if(value==0):

print('Name = ',key)

null\_cnt = null\_cnt+1

print('No of columns which has zero variance = ',null\_cnt)

train\_df = train\_df.drop(columns={'X257','X258','X295','X296','X369'})

train\_df.shape

Question 2:

Check for null and unique values for test and train sets.

print(train\_df.nunique())

print(test\_df.nunique())

#Check for null value

print(train\_df.isnull().sum().any())

print(test\_df.isnull().sum().any())

train\_df.describe(include='object')

test\_df.describe(include='object')

dup\_ID = train\_df['ID'].duplicated().sum()

print(f"Here we have {dup\_ID} duplicate IDs")

No null data, all unique values across the file listed.

Henceforth working with Train data only as it is the data that we would use for our model.

Question 3:

Apply label encoder.

No null variable. All the variables are categorical applying encoder

from sklearn.preprocessing import LabelEncoder

le=LabelEncoder()

for i in train\_df.columns:

train\_df[i]=le.fit\_transform(train\_df[i])

train\_df.head()

train\_df.corr()

Summarize outcome (testing time) in training dataset

# Draw a vertical boxplot grouped

# by a categorical variable: X0

sns.set\_style("whitegrid")

object\_columns = test\_df.describe(include='object').columns

print('\nobject columns:\n',object\_columns)

cols = len(object\_columns)

sns.boxplot(x = 'X0', y = 'y', data = train\_df)

sns.boxplot(x = 'X1', y = 'y', data = train\_df)

sns.boxplot(x = 'X2', y = 'y', data = train\_df)

sns.boxplot(x = 'X3', y = 'y', data = train\_df)

sns.boxplot(x = 'X4', y = 'y', data = train\_df)

sns.boxplot(x = 'X5', y = 'y', data = train\_df)

sns.boxplot(x = 'X6', y = 'y', data = train\_df)

sns.boxplot(x = 'X8', y = 'y', data = train\_df)

#Now the target y

train\_df['y'].hist()

sns.violinplot(train\_df['y'].values)

The data seems optimized. The removal of few data points which had no variance had optimized the y. No reason to test for any more outliers.

Idealy, this test is done first, but if variance 0 is removed, it increases the chances of y being optimized, with no outliers.

Dimensionality reduction refers to techniques for reducing the number of input variables in training data.

Fewer input dimensions often means correspondingly fewer parameters or a simpler structure in the machine learning model, referred to as degrees of freedom. A model with too many degrees of freedom is likely to overfit the training dataset and may not perform well on new data.

It is desirable to have simple models that generalize well, and in turn, input data with few input variables. This is particularly true for linear models where the number of inputs and the degrees of freedom of the model are often closely related.

Dimensionality reduction is a data preparation technique performed on data prior to modeling. It might be performed after data cleaning and data scaling and before training a predictive model.

Question 4: Perform dimensionality reduction.

The methods at our disposal using linear algebra are:

Principal Components Analysis

Singular Value Decomposition

Non-Negative Matrix Factorization

# Draw a vertical boxplot grouped

# by a categorical variable: X0

train\_df.describe(include='int64')

bin\_columns = train\_df['ID']

print('\nobject columns:\n',bin\_columns)

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaler.fit(train\_df)

sdata = scaler.transform(train\_df)

from sklearn.decomposition import PCA

sdata.shape

# lets take top 6 pca components

pca = PCA(n\_components=6)

pca.fit(sdata)

x\_pca = pca.transform(sdata)

x\_pca.shape

# number of components

n\_pcs= pca.components\_.shape[0]

n\_pcs

# get the index of the most important feature on EACH component i.e. largest␣,→absolute value

# using LIST COMPREHENSION HERE

most\_important = [np.abs(pca.components\_[i]).argmax() for i in range(n\_pcs)]

initial\_feature\_names = bin\_columns

most\_important

# using LIST COMPREHENSION HERE AGAIN

dic = {'PC{}'.format(i): most\_important[i] for i in range(n\_pcs)}

dic

pca.components\_

explained\_variance = pca.explained\_variance\_ratio\_

explained\_variance

#it is a measure of the variance of the data when projected onto that axis. The projection of each data point onto the

#principal axes are the “principal components” of the data. .4 is the var of PCA

#and .179 is the var of PCA2

#Creating training and test data with only these columns

selected\_columns = train\_df[['ID', 'X325', 'X27', 'X180', 'X161', 'X309', 'X85', 'y']]

X\_train = selected\_columns.copy()

X\_train.shape

X\_train

#Now creating a df with only these 5 components

selected\_columns = test\_df[['ID', 'X325', 'X27', 'X180', 'X161', 'X309', 'X85']]

X\_test = selected\_columns.copy()

X\_test.shape

X\_test

#now will perfrom XGBoost

Predict your test\_df values using XGBoost.

Model Selection

Logistic Regression

KNN

SVM

Random Forest

#Now splitting the data into train & test. Before that, identifying all input parameters as X, and output parameter as y

y\_train=train\_df['y']

y\_train

y\_train.shape

from sklearn.model\_selection import learning\_curve

from sklearn.linear\_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model\_selection import cross\_val\_score

from sklearn.metrics import classification\_report, confusion\_matrix

import xgboost as xgb

from sklearn.metrics import r2\_score

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

X\_train.shape

X\_test.shape

y\_train.shape

#We do not have X & y. Creating X

X = X\_train

#X = pd.concat([X\_train,X\_test])

print(X)

X.shape

#y = pd.concat([y\_train,y\_train])

y = y\_train

y.shape

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=72)

# Logistic Regression

logreg=LogisticRegression(solver='liblinear',multi\_class='ovr')

logreg.fit(X\_train,y\_train)

y\_pred=logreg.predict(X\_test)

y\_pred

#Accuracy Score

#print(metrics.accuracy\_score(y\_pred,y\_train))

accuracy = (logreg.score(X\_train,y\_train))

print(accuracy)

# Logistic Regression

logreg=LogisticRegression(solver='lbfgs',multi\_class='auto')

logreg.fit(X\_train,y\_train)

y\_pred=logreg.predict(X\_test)

y\_pred

#Accuracy Score

#print(metrics.accuracy\_score(y\_pred,y\_train))

accuracy = (logreg.score(X\_train,y\_train))

print(accuracy)

#SVM "Support Vector Classifier"

from sklearn.svm import SVC

svm = SVC(kernel='linear')

# fitting x samples and y classes

svm.fit(X\_train,y\_train)

y\_pred = svm.predict(X\_test)

from sklearn import metrics

accuracy = metrics.accuracy\_score(y\_test, y\_pred)

print(accuracy)

#KNN with 5 neighbours

knn = KNeighborsClassifier(n\_neighbors=5)

knn.fit(X\_train, y\_train)

y\_pred = knn.predict(X\_test)

print(metrics.accuracy\_score(y\_test, y\_pred))

print(classification\_report(y\_test,pred))

knn = KNeighborsClassifier(n\_neighbors=1)

knn.fit(X\_train, y\_train)

y\_pred = knn.predict(X\_test)

print(metrics.accuracy\_score(y\_test, y\_pred))

print(classification\_report(y\_test,pred))

avg\_score=[]

for k in range(2,30):

knn=KNeighborsClassifier(n\_jobs=-1,n\_neighbors=k)

score=cross\_val\_score(knn,X\_train,y\_train,cv=5,n\_jobs=-1,scoring='accuracy')

avg\_score.append(score.mean())

plt.figure(figsize=(12,8))

plt.plot(range(2,30),avg\_score)

plt.xlabel("n\_neighbours")

plt.ylabel("accuracy")

#plt.xticks(range(2,30,2))

#Random Forests Classifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import f1\_score

rfc=RandomForestClassifier(n\_jobs=-1,random\_state=51)

rfc.fit(X\_train,y\_train)

print(rfc.score(X\_test,y\_test))

print(f1\_score(y\_test,rfc.predict(X\_test),average='macro'))

#Till now SVM followed by Random forest is the leading model

from xgboost import XGBClassifier

from sklearn.metrics import mean\_squared\_error

from sklearn import svm

from xgboost import XGBClassifier

import xgboost as xgb

#Here, we are using XGBRegressor as a Machine Learning model to fit the data.

model = xgb.XGBRegressor(booster='dart', objective='reg:squarederror', num\_class = 1, eval\_metric = 'merror', n\_estimators = 10, seed = 123)

model.fit(X\_train, y\_train)

print(); print(model)

# Predict the model

pred = model.predict(X\_test)

# RMSE Computation

rmse = np.sqrt(mean\_squared\_error(y\_test, pred))

print("RMSE : % f" %(rmse))

expected\_y = y\_test

predicted\_y = model.predict(X\_test)

print(metrics.r2\_score(y\_test, predicted\_y))

predicted\_y

#Here, we are using XGBRegressor as a Machine Learning model to fit the data.

model = xgb.XGBRegressor(booster='gblinear', objective='reg:squarederror', num\_class = 1, eval\_metric = 'merror', n\_estimators = 10, seed = 123)

model.fit(X\_train, y\_train)

print(); print(model)

# Predict the model

pred = model.predict(X\_test)

# RMSE Computation

rmse = np.sqrt(mean\_squared\_error(y\_test, pred))

print("RMSE : % f" %(rmse))

expected\_y = y\_test

predicted\_y = model.predict(X\_test)

print(metrics.r2\_score(y\_test, predicted\_y))

#As we see gbliner is not the right model.

#Here, we are using XGBRegressor as a Machine Learning model to fit the data.

model = xgb.XGBRegressor(booster='gbtree', objective='reg:squarederror', num\_class = 1, eval\_metric = 'merror', n\_estimators = 10, seed = 123)

model.fit(X\_train, y\_train)

print(); print(model)

# Predict the model

pred = model.predict(X\_test)

# RMSE Computation

rmse = np.sqrt(mean\_squared\_error(y\_test, pred))

print("RMSE : % f" %(rmse))

expected\_y = y\_test

predicted\_y = model.predict(X\_test)

print(metrics.r2\_score(y\_test, predicted\_y))

model = xgb.XGBRegressor()

model.fit(X\_train, y\_train)

print(); print(model)

plt.figure(figsize=(10,10))

sns.regplot(expected\_y, predicted\_y, fit\_reg=True, scatter\_kws={"s": 100})

#e'll check the training accuracy with cross-validation and k-fold methods.

# Applying k-Fold Cross Validation

from sklearn.model\_selection import cross\_val\_score, KFold

kfold = KFold(n\_splits=10, shuffle=True)

kf\_cv\_scores = cross\_val\_score(model, X\_train, y\_train, cv=kfold )

print("K-fold CV average score: %.2f" % kf\_cv\_scores.mean())

#Now we have predicted the output by passing X\_test and also stored real target in expected\_y.

expected\_y = y\_test

predicted\_y = model.predict(X\_test)

print(metrics.r2\_score(expected\_y, predicted\_y))

#This is the best model so far

#Q 5. Predicting y with XGBoost

#Displaying predicted values

predicted\_y

# ScreenShots:

A picture containing text, screenshot, computer, computer

Description automatically generated

A computer screen capture

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated

A picture containing text, screenshot, computer, computer

Description automatically generated

A computer screen shot

Description automatically generated with low confidence

A picture containing text, screenshot, computer

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

A picture containing text, screenshot, computer, computer

Description automatically generated

A picture containing text, screenshot, computer, computer

Description automatically generated

A picture containing text, screenshot, computer

Description automatically generated

Graphical user interface, application, Excel

Description automatically generated

Graphical user interface

Description automatically generated

Chart, bar chart

Description automatically generated

Graphical user interface, chart

Description automatically generated

A picture containing text, screenshot, computer, computer

Description automatically generated

Graphical user interface, text, application

Description automatically generated

A picture containing text, screenshot, computer, computer

Description automatically generated

Graphical user interface, text, application

Description automatically generated

A picture containing text, screenshot, computer, computer

Description automatically generated

Graphical user interface, application

Description automatically generated

A screenshot of a computer

Description automatically generated

A picture containing text, screenshot, computer, computer

Description automatically generated

Chart

Description automatically generated

# Embedded File:

Python File:



# Code hosted on GitHub :

<https://github.com/ks-alokranjan/Simplilearn_ML>