Assignment 3 -Semiconductor manufacturing process dataset

Student Name: Brendan Lai
Student Number: 19241173

Project Description

Source: https://www.kaggle.com/saurabhbagchi/fmst-semiconductor-manufacturing-project

A complex modern semiconductor manufacturing process is normally under constant surveillance via the monitoring of signals/variables collected from sensors and or process measurement points. However, not all of these signals are equally valuable in a specific monitoring system. The measured signals contain a combination of useful information, irrelevant information as well as noise. Engineers typically have a much larger number of signals than are actually required. If we consider each type of signal as a feature, then feature selection may be applied to identify the most relevant signals. The Process Engineers may then use these signals to determine key factors contributing to yield excursions downstream in the process. This will enable an increase in process throughput, decreased time to learning, and reduce per-unit production costs. These signals can be used as features to predict the yield type. And by analyzing and trying out different combinations of features, essential signals that are impacting the yield type can be identified.

Dataset: SemiconductorManufacturingProcessDataset.csv (on Canvas)

Later, we will learn how to apply PCA (Principal Component Analyses) for feature selection; then we will apply ANN to predict the Pass/Fail. in this exercise our objective is to repeat the same steps we did above for Supplier Data: Cleaning & Scaling Data, Encode Categorical Data, Split the Data to Training & Test Sets.

Importing the Libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Importing the Dataset

```
dataset = pd.read_csv('SemiconductorManufacturingProcessDataset.csv')
```

Showing the Dataset in a Table

```
dataset.head()
```

	Time	Sensor 1	Sensor 2	Sensor 3	Sensor 4	Sensor 5	Sensor 6	Sensor 7	Sensor 8	S
0	7/19/2008 11:55	3030.93	2564.00	2187.7333	1411.1265	1.3602	97.6133	0.1242	1.5005	0
1	7/19/2008 12:32	3095.78	2465.14	2230.4222	1463.6606	0.8294	102.3433	0.1247	1.4966	-0
2	7/19/2008 13:17	2932.61	2559.94	2186.4111	1698.0172	1.5102	95.4878	0.1241	1.4436	0
3	7/19/2008 14:43	2988.72	2479.90	2199.0333	909.7926	1.3204	104.2367	0.1217	1.4882	-0
4	7/19/2008 15:22	3032.24	2502.87	2233.3667	1326.5200	1.5334	100.3967	0.1235	1.5031	-0

5 rows × 439 columns

A Quick Review of the Data

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1567 entries, 0 to 1566
Columns: 439 entries, Time to Pass/Fail
dtypes: float64(437), object(2)
memory usage: 5.2+ MB
```

Seperate The Input and Output

Here, we put the independent variables in X and the dependent variable in y.

```
X = dataset.iloc[:, 1:438].values
y = dataset.iloc[:, -1].values
```

Showing the Input Data in a Table format

```
pd.DataFrame(X)
```

	0	1	2	3	4	5	6	7	8	
0	3030.93	2564.00	2187.7333	1411.1265	1.3602	97.6133	0.1242	1.5005	0.0162	-0.(
1	3095.78	2465.14	2230.4222	1463.6606	0.8294	102.3433	0.1247	1.4966	-0.0005	-0.(
2	2932.61	2559.94	2186.4111	1698.0172	1.5102	95.4878	0.1241	1.4436	0.0041	0.(
3	2988.72	2479.90	2199.0333	909.7926	1.3204	104.2367	0.1217	1.4882	-0.0124	-0.(
4	3032.24	2502.87	2233.3667	1326.5200	1.5334	100.3967	0.1235	1.5031	-0.0031	-0.(
...	...	...	...	...	...	...	...	...	...	
1562	2899.41	2464.36	2179.7333	3085.3781	1.4843	82.2467	0.1248	1.3424	-0.0045	-0.(
1563	3052.31	2522.55	2198.5667	1124.6595	0.8763	98.4689	0.1205	1.4333	-0.0061	-0.(
1564	2978.81	2379.78	2206.3000	1110.4967	0.8236	99.4122	0.1208	NaN	NaN	
1565	2894.92	2532.01	2177.0333	1183.7287	1.5726	98.7978	0.1213	1.4622	-0.0072	0.(
1566	2944.92	2450.76	2195.4444	2914.1792	1.5978	85.1011	0.1235	NaN	NaN	

1567 rows × 437 columns

A Quick Check of the Output Data

```
pd.DataFrame(y)
```

Taking care of missing data

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
imputer.fit(X)
X = imputer.transform(X)
```

Encoding Categorical Data

No Independent Variable Data to encode

1 Pass

Encoding the Dependent Variable

3 Pass

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)
```

1562 Pass

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X = sc.fit_transform(X)
```

Splitting the Dataset into the Training set and Test set

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 1)
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
LogitReg = LogisticRegression()
LogitReg.fit(X, y)
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:444: Converge
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

LogisticRegression

LogisticRegression()

Predicting a new Result

```
y_pred = LogitReg.predict(X_test)
```

Making the Confusion Matrix

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
print(f"Your Model Accuracy is = {accuracy_score(y_test, y_pred)*100}")
```

```
[[ 14   6]
 [  1 293]]
Your Model Accuracy is = 97.77070063694268
```

Concluding Analysis

We can see that given our prbability and model accuracy that our accuracy was quite high. Additionally with a false positive rate of 1/289 * 100% = 0.34% points to a significantly accurate model