Project Name: Drinking Water Potability Prediction using ML and H2O Auto ML

• Project begin with data analysis then used for logistic regression, SVM & random forest & using H20 ML. • H2O is a fully open source, distributed in-memory machine learning platform with linear scalability.

Context :

Access to safe drinking water is essential to health, a basic human right, and a component of effective policy for health protection. This is important as a health and development issue at a national, regional, and local level. In some regions, it has been shown that investments in water supply and sanitation can yield a net economic benefit, since the reductions in adverse health effects and health care costs outweigh the costs of undertaking the interventions.

The drinkingwaterpotability.csv file contains water quality metrics for 3276 different water bodies

We will use different ML models and H2O Auto ML library in this project

Time Line of the Project:

- Importing Libraries and DataSet
- Data Analysis and Preprocessing
- Feature Engineering
- Model Building using ML
- Model Building and Prediction using H2O Auto ML

Importing Libraries

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

```
from sklearn.model_selection import train_test_split
%matplotlib inline
```

Loading the Data Set

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
df= pd.read_csv("/content/drive/MyDrive/drinking_water_potability.csv")
```

```
df.head()
```

```
df.shape
```

```
(3276, 10)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3276 entries, 0 to 3275
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ph               2785 non-null   float64
 1   Hardness         3276 non-null   float64
 2   Solids           3276 non-null   float64
 3   Chloramines      3276 non-null   float64
 4   Sulfate          2495 non-null   float64
 5   Conductivity     3276 non-null   float64
 6   Organic_carbon   3276 non-null   float64
 7   Trihalomethanes  3114 non-null   float64
 8   Turbidity        3276 non-null   float64
```

```
 9   Potability       3276 non-null   int64
dtypes: float64(9), int64(1)
memory usage: 256.1 KB
```

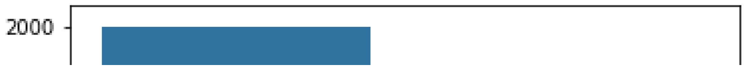
```
df.nunique()
```

```
ph                  2785
Hardness            3276
Solids              3276
Chloramines         3276
Sulfate             2495
Conductivity        3276
Organic_carbon      3276
Trihalomethanes     3114
Turbidity           3276
Potability             2
dtype: int64
```

Data Analysis

```
sns.countplot(data=df,x=df.Potability)
df.Potability.value_counts()
```

```
0    1998
1    1278
Name: Potability, dtype: int64
```




```
df.isnull().sum()
```

```
ph                 491
Hardness             0
Solids               0
Chloramines          0
Sulfate            781
Conductivity         0
Organic_carbon       0
Trihalomethanes    162
Turbidity            0
Potability           0
dtype: int64
```

Handling Null Values

[ ] ↳ *13 cells hidden*

Feature Engineering

[ ] ↳ *6 cells hidden*

Let us Standardize our data

[ ] ↳ *4 cells hidden*

Our data is ready for model building

[ ] ↳ *1 cell hidden*

Model Development

We will use the following models:

- Logistic Regression
- SVM
- Random Forest

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,classification_report
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
```

Logistic Regression

```
lr = LogisticRegression()
lr.fit(X_train, y_train)
y_train_hat = lr.predict(X_train)
y_test_hat = lr.predict(X_test)
```

```
print('Test performance')
print('--------------------------------------------------------')
print(classification_report(y_test, y_test_hat))

print('Roc_auc score')
print('--------------------------------------------------------')
print(roc_auc_score(y_test, y_test_hat))
```

```
print('')

print('Confusion matrix')
print('-------------------------------------------------------')
print(confusion_matrix(y_test, y_test_hat))
print('')

print('accuracy score')
print('-------------------------------------------------------')
print("test data accuracy score:",accuracy_score(y_test, y_test_hat)*100)
print("train data accuracy score:",accuracy_score(y_train, y_train_hat)*100)
```

Support Vector Machines

```
svm = SVC()
svm.fit(X_train, y_train)
y_train_hat = svm.predict(X_train)
y_test_hat = svm.predict(X_test)

print('Test performance')
print('-------------------------------------------------------')
print(classification_report(y_test, y_test_hat))

print('Roc_auc score')
print('-------------------------------------------------------')
print(roc_auc_score(y_test, y_test_hat))
print('')

print('Confusion matrix')
print('-------------------------------------------------------')
print(confusion_matrix(y_test, y_test_hat))
print('')

print('accuracy score')
```

```
print('---------------------------------------------------------')
print(accuracy_score(y_test, y_test_hat)*100)
print("test data accuracy score:",accuracy_score(y_test, y_test_hat)*100)
print("train data accuracy score:",accuracy_score(y_train, y_train_hat)*100)
```

Random Forest

```
rf = RandomForestClassifier(n_jobs=-1,random_state=123)
rf.fit(X_train, y_train)
y_train_hat = rf.predict(X_train)
y_test_hat = rf.predict(X_test)

print('Test performance')
print('---------------------------------------------------------')
print(classification_report(y_test, y_test_hat))

print('Roc_auc score')
print('---------------------------------------------------------')
print(roc_auc_score(y_test, y_test_hat))
print('')

print('Confusion matrix')
print('---------------------------------------------------------')
print(confusion_matrix(y_test, y_test_hat))
print('')

print('accuracy score')
print('---------------------------------------------------------')
print("test data accuracy score:",accuracy_score(y_test, y_test_hat)*100)
print("train data accuracy score:",accuracy_score(y_train, y_train_hat)*100)
```

Using Auto ML

- H2O Auto ML

H2O is a fully open-source, distributed in-memory machine learning platform with linear scalability. H2O supports the most widely used statistical & machine learning algorithms, including gradient boosted machines, generalized linear models, deep learning, and many more.

Installing H2O Auto ML

```
!pip install requests
!pip install tabulate
!pip install "colorama>=0.3.8"
!pip install future
```

```
!pip install h2o
```

Importing the h2o Python module and H2OAutoML class

```
import h2o
from h2o.automl import H2OAutoML
h2o.init(max_mem_size='16G')  ##  the h2o.init() makes sure that no prior instance of H2O is running.
```

Loading data

```
df = h2o.import_file("/content/drive/MyDrive/drinking_water_potability.csv")
```

```
    Parse progress: |██████████████████████████████████████████████████████████| (done) 100%
```

```
df.head()
```

H2O auto ml can do all the data preprocessing techniques

```
df_train,df_test= df.split_frame(ratios=[.8])
```

Splitting the data

```
y = "Potability"  ## dependent variable
x = df.columns  ## Independent variable
x.remove(y)
```

Defining the model

```
aml = H2OAutoML(max_runtime_secs=300,max_models = 10, seed = 10, verbosity="info", nfolds=2)
```

Fitting the model

```
aml.train(x=x,y=y, training_frame=df_train)
```

Seeing the Leaderboard

```
lb = aml.leaderboard
```

```
lb
```

Getting all the model ids

```
model_ids = list(aml.leaderboard['model_id'].as_data_frame().iloc[:,0])
```

```
model_ids
```

```
aml.leader.model_performance(df_test)
```

Getting the model details for best performing model

```
h2o.get_model([mid for mid in model_ids if "StackedEnsemble" in mid][0])
```

```
output= h2o.get_model([mid for mid in model_ids if "StackedEnsemble" in mid][0])
output.params
```

```
aml.leader
```

```
y_pred=aml.leader.predict(df_test)
```

```
    stackedensemble prediction progress: |██████████████████████████████████████████| (done) 100%
```

```
y_pred
```

predict
0.485927
0.398239
0.386352
0.350281
0.49567
0.386966

If probablity greater than 0.5 than it is a 1 else it is a 0

0.30379