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

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

```
## !pip install lazypredict


# Python Imports
import math,time,random,datetime

# Data Manipulation
import numpy as np
import pandas as pd

# Maching Learning
from sklearn.model_selection import train_test_split
from sklearn import model_selection, tree, preprocessing, metrics, linear_model
from sklearn.svm import LinearSVC
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.tree import DecisionTreeClassifier

# Display all Columns
pd.set_option('display.max_columns', None)

# Ignore Warnings
import warnings
warnings.filterwarnings('ignore')

import six
import sys
sys.modules['sklearn.externals.six'] = six

# Imbalanced Data Handling
from imblearn.combine import SMOTETomek
from collections import Counter

# Install lazypredict
!pip install lazypredict

# Model Analysis
import lazypredict
from lazypredict.Supervised import LazyClassifier
```

```
Collecting lazypredict
  Downloading lazypredict-0.2.12-py2.py3-none-any.whl.metadata (12 kB)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from lazypredict) (8.1.7)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (from lazypredict) (1.3.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from lazypredict) (2.1.4)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from lazypredict) (4.66.5)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from lazypredict) (1.4.2)
Requirement already satisfied: lightgbm in /usr/local/lib/python3.10/dist-packages (from lazypredict) (4.4.0)
Requirement already satisfied: xgboost in /usr/local/lib/python3.10/dist-packages (from lazypredict) (2.1.1)
Requirement already satisfied: numpy>=1.17.0 in /usr/local/lib/python3.10/dist-packages (from lightgbm->lazypredict) (1.26.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from lightgbm->lazypredict) (1.13.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas->lazypredict) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->lazypredict) (2024.2)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas->lazypredict) (2024.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->lazypredict) (3.5.0)
Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.10/dist-packages (from xgboost->lazypredict) (2.23.4)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas->lazypredict) (1
Downloading lazypredict-0.2.12-py2.py3-none-any.whl (12 kB)
Installing collected packages: lazypredict
Successfully installed lazypredict-0.2.12
```

```
# Import the train and test data.
train = pd.read_csv('/content/drive/MyDrive/genesight data/Genetic-Disorder-Prediction-main/Genetic-Disorder-Prediction-main/Final_train.csv
test = pd.read_csv('/content/drive/MyDrive/genesight data/Genetic-Disorder-Prediction-main/Genetic-Disorder-Prediction-main/Final_test.csv')


## Capture the Independent Variables
X = train.drop(columns= ['Genetic Disorder', 'Disorder Subclass'], axis= 1)
```

```
## Capture the Dependent Variable
y1 = train['Genetic Disorder']
y2 = train['Disorder Subclass']


## For Genetic Disorder
os= SMOTETomek(random_state= 100)
X1_ns,y1_ns = os.fit_resample(X, y1)
print("The number of classes before fit {}".format(Counter(y1)))
print("The number of classes after fit {}".format(Counter(y1_ns)))
```

```
The number of classes before fit Counter({0: 11174, 2: 8371, 1: 2260})
The number of classes after fit Counter({1: 11168, 0: 10243, 2: 10243})
```

```
## Splitting the Data
X_train, X_test, y_train, y_test = train_test_split(X1_ns, y1_ns, test_size= 0.30, random_state= 42)


from sklearn.multiclass import OneVsRestClassifier
model = LogisticRegression()


## Step 1:
algo = OneVsRestClassifier(model)
model = algo.fit(X_train, y_train)                          # Creating the model. We will fit the algorithm to the training data.
log_acc = round(model.score(X_train, y_train)*100, 2)

## Step 2:  --> This code performs Cross Validation automatically.
log_train_pred = model_selection.cross_val_predict(algo, X_train, y_train, cv= 10, n_jobs= -1)

## Step 3:  --> Cross Validation accuracy metric.
log_acc_cv = round(metrics.accuracy_score(y_train, log_train_pred)*100, 2)
log_f1_cv = round(metrics.f1_score(y_train, log_train_pred, average='macro')*100, 2)


print('Accuracy of the model is: ', log_acc)
print('Accuracy of 10-Fold CV is: ', log_acc_cv)
print('F1 Score is: ', log_f1_cv)
```

```
Accuracy of the model is:  51.2
Accuracy of 10-Fold CV is:  50.81
F1 Score is:  48.92
```

```
# OvR Classifier
model = KNeighborsClassifier()

## Step 1:
algo = OneVsRestClassifier(model)
model1 = algo.fit(X_train, y_train)
knn_acc = round(model1.score(X_train, y_train)*100, 2)

## Step 2:
knn_train_pred = model_selection.cross_val_predict(algo, X_train, y_train, cv= 10, n_jobs= -1)

## Step 3:
knn_acc_cv = round(metrics.accuracy_score(y_train, knn_train_pred)*100, 2)
knn_f1_cv = round(metrics.f1_score(y_train, knn_train_pred, average='macro')*100, 2)

print('Accuracy of the model is: ', knn_acc)
print('Accuracy of 10-Fold CV is: ', knn_acc_cv)
print('F1 Score is: ', knn_f1_cv)
```

```
Accuracy of the model is:  75.04
Accuracy of 10-Fold CV is:  61.85
F1 Score is:  57.9
```

```
# OvR Classifier
model = GaussianNB()

## Step 1:
algo = OneVsRestClassifier(model)
model2 = algo.fit(X_train, y_train)
gnb_acc = round(model2.score(X_train, y_train)*100, 2)

## Step 2:
gnb_train_pred = model_selection.cross_val_predict(algo, X_train, y_train, cv= 10, n_jobs= -1)
```

```
## Step 3:
gnb_acc_cv = round(metrics.accuracy_score(y_train, gnb_train_pred)*100, 2)
gnb_f1_cv = round(metrics.f1_score(y_train, gnb_train_pred, average='macro')*100, 2)

print('Accuracy of the model is: ', gnb_acc)
print('Accuracy of 10-Fold CV is: ', gnb_acc_cv)
print('F1 Score is: ', gnb_f1_cv)
```

```
Accuracy of the model is:  52.16
Accuracy of 10-Fold CV is:  51.92
F1 Score is:  50.28
```

```
# OvR Classifier
model = LinearSVC()

## Step 1:
algo = OneVsRestClassifier(model)
model3 = algo.fit(X_train, y_train)
svc_acc = round(model3.score(X_train, y_train)*100, 2)

## Step 2:
svc_train_pred = model_selection.cross_val_predict(algo, X_train, y_train, cv= 10, n_jobs= -1)

## Step 3:
svc_acc_cv = round(metrics.accuracy_score(y_train, svc_train_pred)*100, 2)
svc_f1_cv = round(metrics.f1_score(y_train, svc_train_pred, average='macro')*100, 2)

print('Accuracy of the model is: ', svc_acc)
print('Accuracy of 10-Fold CV is: ', svc_acc_cv)
print('F1 Score is: ', svc_f1_cv)
```

```
Accuracy of the model is:  51.28
Accuracy of 10-Fold CV is:  50.99
F1 Score is:  48.72
```

```
# OvR Classifier
model = SGDClassifier()

## Step 1:
algo = OneVsRestClassifier(model)
model4 = algo.fit(X_train, y_train)
sgd_acc = round(model4.score(X_train, y_train)*100, 2)

## Step 2:
sgd_train_pred = model_selection.cross_val_predict(algo, X_train, y_train, cv= 10, n_jobs= -1)

## Step 3:
sgd_acc_cv = round(metrics.accuracy_score(y_train, sgd_train_pred)*100, 2)
sgd_f1_cv = round(metrics.f1_score(y_train, sgd_train_pred, average='macro')*100, 2)

print('Accuracy of the model is: ', sgd_acc)
print('Accuracy of 10-Fold CV is: ', sgd_acc_cv)
print('F1 Score is: ', sgd_f1_cv)
```

```
Accuracy of the model is:  48.36
Accuracy of 10-Fold CV is:  48.42
F1 Score is:  44.69
```

```
# OvR Classifier
model = DecisionTreeClassifier()

## Step 1:
algo = OneVsRestClassifier(model)
model5 = algo.fit(X_train, y_train)
dec_acc = round(model5.score(X_train, y_train)*100, 2)

## Step 2:
dec_train_pred = model_selection.cross_val_predict(algo, X_train, y_train, cv= 10, n_jobs= -1)

## Step 3:
dec_acc_cv = round(metrics.accuracy_score(y_train, dec_train_pred)*100, 2)
dec_f1_cv = round(metrics.f1_score(y_train, dec_train_pred, average='macro')*100, 2)

print('Accuracy of the model is: ', dec_acc)
```

```
print('Accuracy of 10-Fold CV is: ', dec_acc_cv)
print('F1 Score is: ', dec_f1_cv)
```

```
Accuracy of the model is:  100.0
Accuracy of 10-Fold CV is:  53.21
F1 Score is:  51.76
```

```
# OvR Classifier
model = GradientBoostingClassifier()

## Step 1:
algo = OneVsRestClassifier(model)
model6 = algo.fit(X_train, y_train)
grd_acc = round(model6.score(X_train, y_train)*100, 2)

## Step 2:
grd_train_pred = model_selection.cross_val_predict(algo, X_train, y_train, cv= 10, n_jobs= -1)

## Step 3:
grd_acc_cv = round(metrics.accuracy_score(y_train, grd_train_pred)*100, 2)
grd_f1_cv = round(metrics.f1_score(y_train, grd_train_pred, average='macro')*100, 2)

print('Accuracy of the model is: ', grd_acc)
print('Accuracy of 10-Fold CV is: ', grd_acc_cv)
print('F1 Score is: ', grd_f1_cv)
```

```
Accuracy of the model is:  61.88
Accuracy of 10-Fold CV is:  59.71
F1 Score is:  57.26
```

```
# OvR Classifier
model = RandomForestClassifier()

## Step 1:
algo = OneVsRestClassifier(model)
model7 = algo.fit(X_train, y_train)
rf_acc = round(model7.score(X_train, y_train)*100, 2)

## Step 2:
rf_train_pred = model_selection.cross_val_predict(algo, X_train, y_train, cv= 10, n_jobs= -1)

## Step 3:
rf_acc_cv = round(metrics.accuracy_score(y_train, rf_train_pred)*100, 2)
rf_f1_cv = round(metrics.f1_score(y_train, rf_train_pred, average='macro')*100, 2)

print('Accuracy of the model is: ', rf_acc)
print('Accuracy of 10-Fold CV is: ', rf_acc_cv)
print('F1 Score is: ', rf_f1_cv)
```

```
Accuracy of the model is:  100.0
Accuracy of 10-Fold CV is:  67.33
F1 Score is:  66.04
```

```
cv_models = pd.DataFrame({'Model':[' Logistic Regression', 'K-Nearest Neighbours', 'Gaussian Naive Bayes',
                                   'Linear Support Vector Machines (SVC)', 'Stochastic Gradient Descent',
                                   'Decision Tree Classifier', 'Gradient Boost Trees', 'Random Forest'],
                          'Score':[log_acc_cv, knn_acc_cv, gnb_acc_cv, svc_acc_cv, sgd_acc_cv, dec_acc_cv, grd_acc_cv, rf_acc_cv]})

print('-----Cross-Validation Accuracy Scores-----')
cv_models.nlargest(9,'Score')
```

```
-----Cross-Validation Accuracy Scores-----
```

	Model	Score
7	Random Forest	67.33
1	K-Nearest Neighbours	61.85
6	Gradient Boost Trees	59.71
5	Decision Tree Classifier	53.21
2	Gaussian Naive Bayes	51.92
3	Linear Support Vector Machines (SVC)	50.99
0	Logistic Regression	50.81
4	Stochastic Gradient Descent	48.42

```
f1_models = pd.DataFrame({'Model':[' Logistic Regression', 'K-Nearest Neighbours', 'Gaussian Naive Bayes',
                                   'Linear Support Vector Machines (SVC)', 'Stochastic Gradient Descent',
                                   'Decision Tree Classifier', 'Gradient Boost Trees', 'Random Forest'],
                          'Score':[log_f1_cv, knn_f1_cv, gnb_f1_cv, svc_f1_cv, sgd_f1_cv, dec_f1_cv, grd_f1_cv, rf_f1_cv]})

print('-----F1 Scores-----')
f1_models.nlargest(9,'Score')
```

```
-----F1 Scores-----
```

	Model	Score
7	Random Forest	66.04
1	K-Nearest Neighbours	57.90
6	Gradient Boost Trees	57.26
5	Decision Tree Classifier	51.76
2	Gaussian Naive Bayes	50.28
0	Logistic Regression	48.92
3	Linear Support Vector Machines (SVC)	48.72
4	Stochastic Gradient Descent	44.69

```
## For Disorder Subclass
os= SMOTETomek(random_state= 100)
X2_ns,y2_ns = os.fit_resample(X, y2)
print("The number of classes before fit {}".format(Counter(y2)))
print("The number of classes after fit {}".format(Counter(y2_ns)))
```

```
The number of classes before fit Counter({6: 6121, 7: 4405, 2: 4183, 8: 2833, 3: 2011, 4: 1355, 5: 648, 0: 152, 1: 97})
The number of classes after fit Counter({5: 6121, 1: 6121, 0: 6121, 4: 6117, 3: 6089, 8: 6011, 2: 5844, 7: 5786, 6: 5625})
```

```
## Splitting the Data
X1_train, X1_test, y1_train, y1_test = train_test_split(X2_ns, y2_ns, test_size=0.30, random_state= 42)


from sklearn.multiclass import OneVsRestClassifier
model1 = LogisticRegression()

## Step 1:
algo = OneVsRestClassifier(model1)
model8 = algo.fit(X1_train, y1_train)                        # Creating the model1. We will fit the algorithm to the training data.
log_acc = round(model8.score(X1_train, y1_train)*100, 2)

## Step 2:  --> This code performs Cross Validation automatically.
log_train_pred = model_selection.cross_val_predict(algo, X1_train, y1_train, cv= 10, n_jobs= -1)

## Step 3:  --> Cross Validation accuracy metric.
log_acc_cv = round(metrics.accuracy_score(y1_train, log_train_pred)*100, 2)
log_f1_cv = round(metrics.f1_score(y1_train, log_train_pred, average='macro')*100, 2)

print('Accuracy of the model is: ', log_acc)
print('Accuracy of 10-Fold CV is: ', log_acc_cv)
print('F1 Score is: ', log_f1_cv)
```

```
Accuracy of the model is:  32.01
Accuracy of 10-Fold CV is:  31.64
F1 Score is:  28.4
```

```
# OvR Classifier
model1 = KNeighborsClassifier()

## Step 1:
algo = OneVsRestClassifier(model1)
model9 = algo.fit(X1_train, y1_train)
knn_acc = round(model9.score(X1_train, y1_train)*100, 2)

## Step 2:
knn_train_pred = model_selection.cross_val_predict(algo, X1_train, y1_train, cv= 10, n_jobs= -1)

## Step 3:
knn_acc_cv = round(metrics.accuracy_score(y1_train, knn_train_pred)*100, 2)
knn_f1_cv = round(metrics.f1_score(y1_train, knn_train_pred, average='macro')*100, 2)

print('Accuracy of the model1 is: ', knn_acc)
print('Accuracy of 10-Fold CV is: ', knn_acc_cv)
print('F1 Score is: ', knn_f1_cv)
```

```
Accuracy of the model1 is:  80.61
Accuracy of 10-Fold CV is:  67.43
F1 Score is:  62.6
```

```
# OvR Classifier
model2 = GaussianNB()

## Step 1:
algo = OneVsRestClassifier(model2)
model10 = algo.fit(X1_train, y1_train)
gnb_acc = round(model10.score(X1_train, y1_train)*100, 2)

## Step 2:
gnb_train_pred = model_selection.cross_val_predict(algo, X1_train, y1_train, cv= 10, n_jobs= -1)

## Step 3:
gnb_acc_cv = round(metrics.accuracy_score(y1_train, gnb_train_pred)*100, 2)
gnb_f1_cv = round(metrics.f1_score(y1_train, gnb_train_pred, average='macro')*100, 2)

print('Accuracy of the model2 is: ', gnb_acc)
print('Accuracy of 10-Fold CV is: ', gnb_acc_cv)
print('F1 Score is: ', gnb_f1_cv)
```

```
Accuracy of the model2 is:  31.54
Accuracy of 10-Fold CV is:  31.25
F1 Score is:  25.5
```

```
# OvR Classifier
model2 = LinearSVC()

## Step 1:
algo = OneVsRestClassifier(model2)
model11 = algo.fit(X1_train, y1_train)
svc_acc = round(model11.score(X1_train, y1_train)*100, 2)

## Step 2:
svc_train_pred = model_selection.cross_val_predict(algo, X1_train, y1_train, cv= 10, n_jobs= -1)

## Step 3:
svc_acc_cv = round(metrics.accuracy_score(y1_train, svc_train_pred)*100, 2)
svc_f1_cv = round(metrics.f1_score(y1_train, svc_train_pred, average='macro')*100, 2)

print('Accuracy of the model2 is: ', svc_acc)
print('Accuracy of 10-Fold CV is: ', svc_acc_cv)
print('F1 Score is: ', svc_f1_cv)
```

```
Accuracy of the model2 is:  31.09
Accuracy of 10-Fold CV is:  30.84
F1 Score is:  25.25
```

```
# OvR Classifier
model2 = SGDClassifier()
```

```
## Step 1:
algo = OneVsRestClassifier(model2)
model12 = algo.fit(X1_train, y1_train)
sgd_acc = round(model12.score(X1_train, y1_train)*100, 2)

## Step 2:
sgd_train_pred = model_selection.cross_val_predict(algo, X1_train, y1_train, cv= 10, n_jobs= -1)

## Step 3:
sgd_acc_cv = round(metrics.accuracy_score(y1_train, sgd_train_pred)*100, 2)
sgd_f1_cv = round(metrics.f1_score(y1_train, sgd_train_pred, average='macro')*100, 2)

print('Accuracy of the model2 is: ', sgd_acc)
print('Accuracy of 10-Fold CV is: ', sgd_acc_cv)
print('F1 Score is: ', sgd_f1_cv)
```

```
Accuracy of the model2 is:  25.97
Accuracy of 10-Fold CV is:  25.61
F1 Score is:  23.16
```

```
# OvR Classifier
model2 = DecisionTreeClassifier()

## Step 1:
algo = OneVsRestClassifier(model2)
model13 = algo.fit(X1_train, y1_train)
dec_acc = round(model13.score(X1_train, y1_train)*100, 2)

## Step 2:
dec_train_pred = model_selection.cross_val_predict(algo, X1_train, y1_train, cv= 10, n_jobs= -1)

## Step 3:
dec_acc_cv = round(metrics.accuracy_score(y1_train, dec_train_pred)*100, 2)
dec_f1_cv = round(metrics.f1_score(y1_train, dec_train_pred, average='macro')*100, 2)

print('Accuracy of the model2 is: ', dec_acc)
print('Accuracy of 10-Fold CV is: ', dec_acc_cv)
print('F1 Score is: ', dec_f1_cv)
```

```
Accuracy of the model2 is:  100.0
Accuracy of 10-Fold CV is:  49.13
F1 Score is:  49.26
```

```
# OvR Classifier
model2 = GradientBoostingClassifier()

## Step 1:
algo = OneVsRestClassifier(model2)
model14 = algo.fit(X1_train, y1_train)
grd_acc = round(model14.score(X1_train, y1_train)*100, 2)

## Step 2:
grd_train_pred = model_selection.cross_val_predict(algo, X1_train, y1_train, cv= 10, n_jobs= -1)

## Step 3:
grd_acc_cv = round(metrics.accuracy_score(y1_train, grd_train_pred)*100, 2)
grd_f1_cv = round(metrics.f1_score(y1_train, grd_train_pred, average='macro')*100, 2)

print('Accuracy of the model2 is: ', grd_acc)
print('Accuracy of 10-Fold CV is: ', grd_acc_cv)
print('F1 Score is: ', grd_f1_cv)
```

```
Accuracy of the model2 is:  47.19
Accuracy of 10-Fold CV is:  42.99
F1 Score is:  38.52
```

```
# OvR Classifier
model2 = RandomForestClassifier()

## Step 1:
algo = OneVsRestClassifier(model2)
model15 = algo.fit(X1_train, y1_train)
rf_acc = round(model15.score(X1_train, y1_train)*100, 2)

## Step 2:
```

```
rf_train_pred = model_selection.cross_val_predict(algo, X1_train, y1_train, cv= 10, n_jobs= -1)

## Step 3:
rf_acc_cv = round(metrics.accuracy_score(y1_train, rf_train_pred)*100, 2)
rf_f1_cv = round(metrics.f1_score(y1_train, rf_train_pred, average='macro')*100, 2)

print('Accuracy of the model2 is: ', rf_acc)
print('Accuracy of 10-Fold CV is: ', rf_acc_cv)
print('F1 Score is: ', rf_f1_cv)
```

```
Accuracy of the model2 is:  100.0
Accuracy of 10-Fold CV is:  70.92
F1 Score is:  69.09
```

```
f1_models = pd.DataFrame({'Model':[' Logistic Regression', 'K-Nearest Neighbours', 'Gaussian Naive Bayes',
                                   'Linear Support Vector Machines (SVC)', 'Stochastic Gradient Descent',
                                   'Decision Tree Classifier', 'Gradient Boost Trees', 'Random Forest'],
                          'Score':[log_f1_cv, knn_f1_cv, gnb_f1_cv, svc_f1_cv, sgd_f1_cv, dec_f1_cv, grd_f1_cv, rf_f1_cv]})

print('-----F1 Scores-----')
f1_models.nlargest(9,'Score')
```

```
-----F1 Scores-----
```

	Model	Score
7	Random Forest	69.09
1	K-Nearest Neighbours	62.60
5	Decision Tree Classifier	49.26
6	Gradient Boost Trees	38.52
0	Logistic Regression	28.40
2	Gaussian Naive Bayes	25.50
3	Linear Support Vector Machines (SVC)	25.25
4	Stochastic Gradient Descent	23.16

```
pred = model7.predict(X_test)
print(round(metrics.f1_score(y_test, pred, average='macro')*100, 2))
```

```
66.08
```

```
pred2 = model15.predict(X1_test)
print(round(metrics.f1_score(y1_test, pred2, average='macro')*100, 2))
```

```
70.38
```

```
test_id = pd.read_csv('/content/drive/MyDrive/genesight data/Genetic-Disorder-Prediction-main/Genetic-Disorder-Prediction-main/dataset/test.
test_id.head(1)
```

	White Blood cell count (thousand per microliter)	Blood cell count (mcL)	Patient Age	Father's age	Mother's age	No. of previous abortion	Blood test result	Gender	Birth asphyxia	Symptom 5	Heart Rate (rates/min	Respiratory Rate (breaths/min)	Folic acid details (peri-conceptional)

Next steps: Generate code with test_id | View recommended plots | New interactive sheet

```
test.shape
```

```
(9465, 19)
```

```
final_pred1 = model7.predict(test)
final_pred2 = model15.predict(test)
```

```
final_pred1[:20]
```

```
array([1, 0, 0, 0, 0, 1, 0, 1, 2, 1, 0, 0, 2, 1, 1, 2, 2, 0, 2, 1])
```

```
final_pred2[:20]
```

```
array([5, 8, 7, 7, 2, 5, 7, 5, 8, 5, 7, 2, 8, 3, 3, 2, 6, 6, 4, 2])
```

```
# Create a dataframe and append the relevant colimns.
submission = pd.DataFrame()
# Check for the correct column name in 'test_id' - it's likely called 'Patient_ID'

# Print the available columns in the test_id DataFrame to verify the correct name
print(test_id.columns)

# Use the correct column name from the printed output
submission['Patient Age'] = test_id['Patient Age']
# Replace '<Correct Column Name>' with the actual column name from test_id
submission['Genetic Disorder'] = final_pred1
submission['Disorder Subclass'] = final_pred2
submission.head()
```

```
Index(['White Blood cell count (thousand per microliter)',
       'Blood cell count (mcL)', 'Patient Age', 'Father's age', 'Mother's age',
       'No. of previous abortion', 'Blood test result', 'Gender',
       'Birth asphyxia', 'Symptom 5', 'Heart Rate (rates/min',
       'Respiratory Rate (breaths/min)',
       'Folic acid details (peri-conceptional)',
       'History of anomalies in previous pregnancies',
       'Autopsy shows birth defect (if applicable)',
       'Assisted conception IVF/ART', 'Symptom 4', 'Follow-up',
       'Birth defects'],
      dtype='object')
```

	Patient Age	Genetic Disorder	Disorder Subclass
0	-0.12	1	5
1	0.38	0	8
2	-0.25	0	7
3	0.75	0	7
4	0.25	0	2

Next steps: Generate code with submission | View recommended plots | New interactive sheet

```
submission['Genetic Disorder'].replace(0, 'Mitochondrial genetic inheritance disorders', inplace= True)
submission['Genetic Disorder'].replace(2, 'Single-gene inheritance diseases', inplace= True)
submission['Genetic Disorder'].replace(1, 'Multifactorial genetic inheritance disorders', inplace= True)
submission.head()
```

	Patient Age	Genetic Disorder	Disorder Subclass
0	-0.12	Multifactorial genetic inheritance disorders	5
1	0.38	Mitochondrial genetic inheritance disorders	8
2	-0.25	Mitochondrial genetic inheritance disorders	7
3	0.75	Mitochondrial genetic inheritance disorders	7
4	0.25	Mitochondrial genetic inheritance disorders	2

Next steps: Generate code with submission | View recommended plots | New interactive sheet

Generate | randomly select 5 items from a list | Close

```
submission['Disorder Subclass'].replace(0, "Alzheimer's", inplace= True)
submission['Disorder Subclass'].replace(1, 'Cancer', inplace= True)
submission['Disorder Subclass'].replace(2, 'Cystic fibrosis', inplace= True)
submission['Disorder Subclass'].replace(3, 'Diabetes', inplace= True)
submission['Disorder Subclass'].replace(4, 'Hemochromatosis', inplace= True)
submission['Disorder Subclass'].replace(5, "Leber's hereditary optic neuropathy", inplace= True)
submission['Disorder Subclass'].replace(6, 'Leigh syndrome', inplace= True)
submission['Disorder Subclass'].replace(7, 'Mitochondrial myopathy', inplace= True)
submission['Disorder Subclass'].replace(8, 'Tay-Sachs', inplace= True)
submission.head()
```

	Patient Age	Genetic Disorder	Disorder Subclass
0	-0.12	Multifactorial genetic inheritance disorders	Leber's hereditary optic neuropathy
1	0.38	Mitochondrial genetic inheritance disorders	Tay-Sachs
2	-0.25	Mitochondrial genetic inheritance disorders	Mitochondrial myopathy
3	0.75	Mitochondrial genetic inheritance disorders	Mitochondrial myopathy
4	0.25	Mitochondrial genetic inheritance disorders	Cystic fibrosis

```
# convert submission dataframe to csv.
submission.to_csv('/content/drive/MyDrive/genesight data/Genetic-Disorder-Prediction-main/Genetic-Disorder-Prediction-main/final_submission_
print('Submission csv is ready')
```

```
Submission csv is ready
```

```
pip install xgboost
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.10/dist-packages (2.1.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.26.4)
Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.10/dist-packages (from xgboost) (2.23.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.13.1)
```

```
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import pandas as pd



# Load dataset (replace 'data.csv' with your dataset file)
data = pd.read_csv('/content/drive/MyDrive/genesight data/Genetic-Disorder-Prediction-main/Genetic-Disorder-Prediction-main/Final_train.csv'

# Example: Assuming the last column is the target (genetic disorder label)
X = data.iloc[:, :-1]  # Features (all columns except the last one)
y = data.iloc[:, -1]   # Target (the last column)



# Splitting data into train and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)



# Create an XGBoost classifier
model = xgb.XGBClassifier(use_label_encoder=False, eval_metric='mlogloss')

# Train the model on training data
model.fit(X_train, y_train)
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='mlogloss',
              feature_types=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimators=None,
              n_jobs=None, num_parallel_tree=None, objective='multi:softprob', ...)
```

```
# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Accuracy: 57.56%
```

```
model = xgb.XGBClassifier(
    n_estimators=100,        # Number of boosting rounds
    learning_rate=0.1,       # Learning rate (step size)
    max_depth=6,             # Maximum depth of trees
    subsample=0.8,           # Subsample ratio of the training instances
    colsample_bytree=0.8,    # Subsample ratio of columns when constructing each tree
    use_label_encoder=False, # Disables warning for label encoding
    eval_metric='mlogloss'   # Multiclass log-loss for evaluation
)
```

```
!pip install xgboost
```

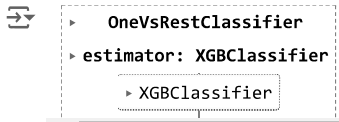
```
Requirement already satisfied: xgboost in /usr/local/lib/python3.10/dist-packages (2.1.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.26.4)
Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.10/dist-packages (from xgboost) (2.23.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.13.1)
```

```
from xgboost import XGBClassifier
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import accuracy_score
```

```
# Importing necessary libraries
from xgboost import XGBClassifier
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

# Assuming your data is loaded into X and y variables

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the XGBoost classifier
xgb_model = XGBClassifier()

# Wrap it in the OneVsRestClassifier for multi-label classification
one_vs_rest_xgb = OneVsRestClassifier(xgb_model)

# Train the model
one_vs_rest_xgb.fit(X_train, y_train)

# Make predictions
y_pred = one_vs_rest_xgb.predict(X_test)

# Evaluate the model accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"XGBoost Accuracy: {accuracy * 100:.2f}%")
```

```
XGBoost Accuracy: 57.83%
```

```
xgb_model = XGBClassifier(
    n_estimators=200,        # Number of trees (boosting rounds)
    learning_rate=0.05,      # Lower learning rate
    max_depth=6,             # Depth of the tree
    subsample=0.8,           # Use 80% of the data to grow trees
    colsample_bytree=0.8,    # Use 80% of the features to build each tree
    gamma=1,                 # Minimum loss reduction
    reg_lambda=1,            # L2 regularization
    reg_alpha=0.1,           # L1 regularization
    use_label_encoder=False, # Disable label encoding warning
    eval_metric='mlogloss'   # Multiclass log loss
)

# Train the model
one_vs_rest_xgb = OneVsRestClassifier(xgb_model)
one_vs_rest_xgb.fit(X_train, y_train)
```




```
# Calculate the ratio of the minority class to the majority class
ratio_of_minority_class = 1 # Replace 1 with the actual calculated value

xgb_model = XGBClassifier(scale_pos_weight=ratio_of_minority_class)


from sklearn.model_selection import cross_val_score

# Perform cross-validation
scores = cross_val_score(one_vs_rest_xgb, X, y, cv=5, scoring='accuracy')

# Print average accuracy
print(f"Cross-validated accuracy: {scores.mean() * 100:.2f}%")
```

```
Cross-validated accuracy: 60.19%
```

```
!pip install --upgrade xgboost
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.10/dist-packages (2.1.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.26.4)
Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.10/dist-packages (from xgboost) (2.23.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.13.1)
```

```
# Train model with early stopping using a callback
from xgboost import XGBClassifier

# If you want to use early stopping you need to use the xgboost API not the scikit-learn API
xgb_model = XGBClassifier(
    n_estimators=200,
    learning_rate=0.05,
    max_depth=6,
    subsample=0.8,
    colsample_bytree=0.8,
    gamma=1,
    reg_lambda=1,
    reg_alpha=0.1,
    #use_label_encoder=False,
    eval_metric='mlogloss'
)


# Create DMatrices
from xgboost import DMatrix
dtrain = DMatrix(X_train, label=y_train)
dtest = DMatrix(X_test, label=y_test)

# Specify parameters for training
params = {
    'objective': 'multi:softmax',  # For multi-class classification
    'num_class': len(set(y_train)) # Number of classes
}

# Train the model with early stopping
num_rounds = 200
early_stopping_rounds = 10

xgb_model = xgb.train(
    params,
    dtrain,
    num_boost_round=num_rounds,
    evals=[(dtest, 'test')],
    early_stopping_rounds=early_stopping_rounds
)
```

```
[0]     test-mlogloss:1.62672
[1]     test-mlogloss:1.38247
[2]     test-mlogloss:1.22664
[3]     test-mlogloss:1.11885
[4]     test-mlogloss:1.04032
```

```
[5]	test-mlogloss:0.98323
[6]	test-mlogloss:0.93943
[7]	test-mlogloss:0.90575
[8]	test-mlogloss:0.87931
[9]	test-mlogloss:0.85911
[10]	test-mlogloss:0.84233
[11]	test-mlogloss:0.82942
[12]	test-mlogloss:0.81922
[13]	test-mlogloss:0.81051
[14]	test-mlogloss:0.80415
[15]	test-mlogloss:0.79814
[16]	test-mlogloss:0.79360
[17]	test-mlogloss:0.78989
[18]	test-mlogloss:0.78613
[19]	test-mlogloss:0.78369
[20]	test-mlogloss:0.78185
[21]	test-mlogloss:0.78052
[22]	test-mlogloss:0.77944
[23]	test-mlogloss:0.77844
[24]	test-mlogloss:0.77698
[25]	test-mlogloss:0.77629
[26]	test-mlogloss:0.77635
[27]	test-mlogloss:0.77575
[28]	test-mlogloss:0.77597
[29]	test-mlogloss:0.77603
[30]	test-mlogloss:0.77615
[31]	test-mlogloss:0.77674
[32]	test-mlogloss:0.77702
[33]	test-mlogloss:0.77684
[34]	test-mlogloss:0.77633
[35]	test-mlogloss:0.77738
[36]	test-mlogloss:0.77791
```

To combine XGBoost and Random Forest algorithms in a stacked or ensemble method, you can use two approaches: Stacking or Blending. Below is an example of combining XGBoost and Random Forest using Stacking.

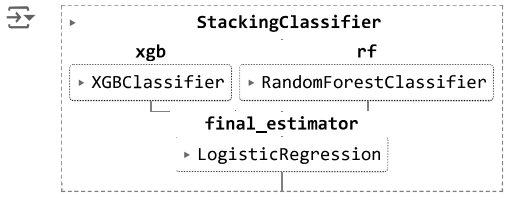
Code for Combining XGBoost and Random Forest: In this approach, we use Stacking where both models (XGBoost and Random Forest) are trained, and their predictions are combined in a meta-classifier to make the final prediction.

Step-by-Step Stacked Model:

```
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.ensemble import StackingClassifier
from sklearn.metrics import accuracy_score



# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)



# Initialize the base models
xgb_model = XGBClassifier(
    n_estimators=200,        # Number of trees
    learning_rate=0.05,      # Learning rate
    max_depth=6,             # Maximum depth of trees
    subsample=0.8,           # Fraction of samples for each tree
    colsample_bytree=0.8,    # Fraction of features for each tree
    use_label_encoder=False, # Disable warning
    eval_metric='mlogloss'   # Evaluation metric
)

rf_model = RandomForestClassifier(
    n_estimators=200,        # Number of trees
    max_depth=8,             # Maximum depth
    random_state=42
)



from sklearn.linear_model import LogisticRegression

# Create the stacking classifier
stacking_model = StackingClassifier(
    estimators=[
```

```
    estimators=[
        ('xgb', xgb_model),
        ('rf', rf_model)
    ],
    final_estimator=LogisticRegression(),
    cv=5  # Number of cross-validation folds
)
```

```
# Train the stacking model
stacking_model.fit(X_train, y_train)
```




```
# Make predictions on the test data
y_pred = stacking_model.predict(X_test)

# Evaluate the model accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Stacking Model Accuracy: {accuracy * 100:.2f}%")
```

```
Stacking Model Accuracy: 61.96%
```