Configuration Manual

MSc Research Project
Data Analytics

Anna Joy
Student ID:x23238241

School of Computing
National College of Ireland

Supervisor: Jorge Basilio

National College of Ireland

MSc Research Submission Sheet

School of Computing

Student Name:	…….Anna Joy…………..…………………………………………………………………………
Student ID:	…x23238241……………………………………………………………………………….……
Programme:	….MSc in Data Analytics….……………………… **Year:** ……………1……………..
Module:	…..MSc Research project……………………………………………………………….………
Lecturer:	……Jorge Basilio……………………………………………………………….………
Submission Due Date:	……12/12/2024…………………………………………………………………………….………
Research Title:	……Hybrid predictive model for Asthma diagnosis using environmental and life style factors ……………………………………….………
Word Count:	………………813………………… **Page Count:** …………………8………….…….………

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this research. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the research.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ……………………Anna Joy………………………………………………………………………

Date: ………………………11/12/2024…………………………………………………………………………………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each research (including multiple copies)	□
Attach a Moodle submission receipt of the online research submission, to each research (including multiple copies).	□
You must ensure that you retain a HARD COPY of the research, both for your own reference and in case a research is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Anna Joy
Student ID: x23238241

1 Introduction

The configuration manual explains the specific requirements related to the research study on the topic " Hybrid predictive model for Asthma diagnosis by using environmental and lifestyle factors". It contains the criteria and requirements for running the code such as hardware, software and the explanation of the code as well.

2 System Configuration

Hardware configuration

- Processor: minimum Intel Core i5 or equivalent
- RAM: 8 GB or 16 GB recommended for better performance
- Storage: 10 GB free space
- GPU (Optional): NVIDIA GPU with CUDA support for faster neural network training

Software configuration

- Operating System: Windows 11, macOS or Linux
- Python Version: 3.8 or higher
- Additional Software: Jupyter Notebook, for running and visualizing the code

Python libraries

Here, the provided libraries are used to implement and the run the code that used in asthma diagnosis using hybrid model and other traditional models. The python library contains various built-in modules that provide several access to different functions that can execute while running the programs.(Batchelder, 2024)

pandas	1.3.5
NumPy	1.21.5
seaborn	0.11.2
matplotlib	3.5.1
scikit-learn	1.0.2
imbalanced-learn	0.9.1
TensorFlow	2.7.0
keras	2.7.0
Xgboost	1.5.1

3 Research Development

Data preparation

The first step in execution of the code is importing libraries and fig.1 shows the code for importing the libraries from different packages.

```
In [3]: import pandas as pd
        import time
        import seaborn as sns
        import matplotlib.pyplot as plt

        from sklearn.model_selection import train_test_split, cross_val_score
        from sklearn.metrics import accuracy_score
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.metrics import mean_squared_error
        from sklearn.impute import SimpleImputer
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.linear_model import LogisticRegression
        from sklearn.preprocessing import StandardScaler
        from sklearn.naive_bayes import GaussianNB


        from sklearn.model_selection import train_test_split
        from imblearn.over_sampling import SMOTE
        from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, roc_curve
        import numpy as np

        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Dense, Dropout
        import xgboost as xgb
        from sklearn.metrics import classification_report
```

Fig.1

Then we have to load and explore the data using various code and fig.2 represents the code for loading, identifying summary and checking of missing values in the dataset.

```
In [4]: Asthma_data = pd.read_csv('C:\\Users\\Anna joy\\OneDrive\\Desktop\\RIC\\asthma_disease_data.csv')
        Asthma_data.head()
```

Out[4]:

	PatientID	Age	Gender	Ethnicity	EducationLevel	BMI	Smoking	PhysicalActivity	DietQuality	SleepQuality	...	LungFunctionFEV1	LungFunctionFVC
0	5034	63	0	1	0	15.848744	0	0.894448	5.488696	8.701003	...	1.369051	4.941206
1	5035	26	1	2	2	22.757042	0	5.897329	6.341014	5.153966	...	2.197767	1.702393
2	5036	57	0	2	1	18.395396	0	6.739367	9.196237	6.840647	...	1.698011	5.022553
3	5037	40	1	2	1	38.515278	0	1.404503	5.826532	4.253036	...	3.032037	2.300159
4	5038	61	0	0	3	19.283802	0	4.604493	3.127048	9.625799	...	3.470589	3.067944

5 rows × 29 columns

```
In [3]: Asthma_data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2392 entries, 0 to 2391
Data columns (total 29 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   PatientID         2392 non-null   int64
 1   Age               2392 non-null   int64
 2   Gender            2392 non-null   int64
 3   Ethnicity         2392 non-null   int64
 4   EducationLevel    2392 non-null   int64
 5   BMI               2392 non-null   float64
 6   Smoking           2392 non-null   int64
```

```
In [4]: Asthma_data.describe()
```

Out[4]:

	PatientID	Age	Gender	Ethnicity	EducationLevel	BMI	Smoking	PhysicalActivity	DietQuality	SleepQuality	...	Gastroes
count	2392.000000	2392.000000	2392.000000	2392.000000	2392.000000	2392.000000	2392.000000	2392.000000	2392.000000	2392.000000	...	
mean	6229.500000	42.137960	0.493311	0.669732	1.307274	27.244877	0.141722	5.051786	5.022867	7.019012	...	
std	690.655244	21.606655	0.500060	0.986120	0.898242	7.201628	0.348838	2.903574	2.909980	1.732475	...	
min	5034.000000	5.000000	0.000000	0.000000	0.000000	15.031803	0.000000	0.001740	0.003031	4.001437	...	
25%	5631.750000	23.000000	0.000000	0.000000	1.000000	20.968313	0.000000	2.578333	2.432043	5.498500	...	
50%	6229.500000	42.000000	0.000000	0.000000	1.000000	27.052202	0.000000	5.016881	5.115383	6.975839	...	
75%	6827.250000	61.000000	1.000000	1.000000	2.000000	33.555903	0.000000	7.540234	7.544216	8.526950	...	
max	7425.000000	79.000000	1.000000	3.000000	3.000000	39.985611	1.000000	9.995809	9.999904	9.996235	...	

8 rows × 28 columns

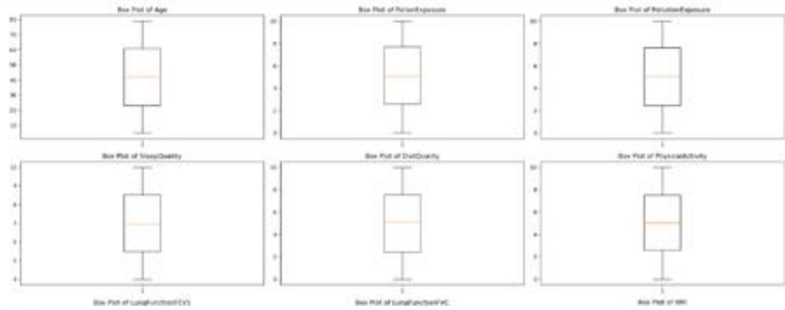
```
In [24]: Asthma_data.isnull().sum()

Out[24]: PatientID           0
         Age                 0
         Gender              0
         Ethnicity           0
         EducationLevel      0
         BMI                 0
         Smoking             0
         PhysicalActivity    0
```

Fig.2

Following that we performed various visualizations such as histogram for feature distributions and box plots for checking outliers in the data and implement a Barplot for the target variable in order to identify any distributions or specific characters(fig.3).

```
In [25]: # List of variables in order to check distribution
variables = ['Age', 'PollenExposure', 'PollutionExposure', 'SleepQuality', 'DietQuality', 'PhysicalActivity', 'LungFunctionFEV1',
# To Plot histograms in order to identify various distrubutions
plt.figure(figsize=(20, 10))

for i, var in enumerate(variables, 1):
    plt.subplot(4, 3, i)
    plt.hist(Asthma_data[var].dropna(), bins=30, edgecolor='k')
    plt.title(f'Distribution of {var}')

plt.tight_layout()
plt.show()
```

```
plt.figure(figsize=(20, 15))# Ploting box plots for the variables

for i, var in enumerate(variables, 1):
    plt.subplot(4, 3, i)
    plt.boxplot(Asthma_data[var].dropna())
    plt.title(f'Box Plot of {var}')

plt.tight_layout()
plt.show()
```



```
In [29]:
# Plotting the distribution of target variable
diagnosis_counts = Asthma_data['Diagnosis'].value_counts()# various Count occurrences of different class
diagnosis_counts.plot(kind='bar', color=['green', 'red'])
plt.xlabel('Diagnosis')
plt.ylabel('Count')
plt.title('Distribution of Diagnosis')
plt.show()
print(diagnosis_counts) #In order to see the exact numbers,we print value counts
```

Fig.3

Feature Engineering

In this section, we dropped unnecessary columns such as “PatientID”, “Doctorincharge” for better implementation of modelling and then we create target and features as provided in fig.4, along with that we also perform a corelation heatmap to identify the corelation between various features.

```
[27]: cleaned = Asthma_data.drop(columns=['PatientID', 'DoctorInCharge'])# differentating target variable and response variable
w = cleaned.drop(columns=['Diagnosis'])#
z = cleaned['Diagnosis']
```

```
[28]:
correlation_matrix = cleaned.corr()#to perform heatmap,we have to create corelation matrix
plt.figure(figsize=[20, 18])
plt.title('Heatmap')# in order to visualize heatmap
sns.heatmap(correlation_matrix, vmin=-1, vmax=1, center=0, annot=True, annot_kws={"size": 8}, color= "red")
plt.show()
```

Fig.4

Data Transformation

Here, the first procedure is to handle missing values, but for this asthma diagnosis dataset, there is no missing values. Then we standardize the features for better performance for the modelling stage as shown in fig.5. Data is split into train and test at a test size of 0.3 and then we perform an oversampling technique called SMOTE in order to handle class imbalance for the target variable.

OVERSAMPLING TECHNIQUES: SMOTE

```
In [5]:
p = Asthma_data.drop(columns=['Diagnosis', 'PatientID', 'DoctorInCharge'])# Separating features and target
y = Asthma_data['Diagnosis']

p_train, p_test, y_train, y_test = train_test_split(p, y, test_size=0.3, random_state=42, stratify=y)#splitting the data into tra

scaler = StandardScaler()#scale the response variables
p_train_scaled = scaler.fit_transform(p_train)
p_test_scaled = scaler.transform(p_test)


smote = SMOTE(random_state=42)#Applying the oversampling technique smote for handling imbalance

p_train_balanced, y_train_balanced = smote.fit_resample(p_train_scaled, y_train)
```

Fig.5

4 Model Application and Evaluation

In this research study, we are designing a hybrid model which is a combination of gradient boosting and neural network model and comparing the hybrid model with other traditional models. So the model application part and evaluation part consists of various code for implementing different models and code for confusion matrix, classification report and ROC curve and AUC score.

Logistic regression

Logistic regression

```
In [36]:
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, roc_curve
import matplotlib.pyplot as plt


model1 = LogisticRegression(random_state=42, max_iter=1000)#intilizing
model1.fit(p_train_balanced, y_train_balanced)#fitting the model on the balanced training data
y_pred = model1.predict(p_test_scaled)# Predicting on the test data
y_pred_proba = model1.predict_proba(p_test_scaled)[:, 1]

# Evaluating the model
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
roc_auc = roc_auc_score(y_test, y_pred_proba)#to identify ROC-AUC score
print(f"\nROC-AUC Score: {roc_auc:.4f}")

fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)#to plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {roc_auc:.4f})")
plt.plot([0, 1], [0, 1], 'k--', label="Random Guess")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()
```

Fig.6

Here, the logistic regression library is utilized to perform this model and the above fig.6 explains the various steps procedures in implementing logistic regression model.

Random forest

Random Forest

```
In [37]: from sklearn.ensemble import RandomForestClassifier

         model2 = RandomForestClassifier(random_state=42, n_estimators=100, max_depth=None)#intilizing
         model2.fit(p_train_balanced, y_train_balanced)#fitting the model
         ypred2 = model2.predict(p_test_scaled)#predicting on testdata
         ypredproba2 = model2.predict_proba(p_test_scaled)[:, 1]

         print("Confusion Matrix:")# evaluvating model
         print(confusion_matrix(y_test, ypred2))
         print("\nClassification Report:")
         print(classification_report(y_test, ypred2))
         roc_auc = roc_auc_score(y_test, ypredproba2)#to calcuate ROC score
         print(f"\nROC-AUC Score: {roc_auc:.4f}")

         fpr, tpr, thresholds = roc_curve(y_test, ypredproba2)#to plot ROC curve
         plt.figure(figsize=(8, 6))
         plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {roc_auc:.4f})")
         plt.plot([0, 1], [0, 1], 'k--', label="Random Guess")
         plt.xlabel("False Positive Rate")
         plt.ylabel("True Positive Rate")
         plt.title("ROC Curve")
         plt.legend()
         plt.show()
```

Fig.7

Here, fig.7 shows the random forest application with several settings that including initializing, fitting , predicting the test data and evaluating the model as well.

Gradient boosting

Gradient Boosting

```
In [38]: from sklearn.ensemble import GradientBoostingClassifier

         model3 = GradientBoostingClassifier(random_state=42, n_estimators=100, learning_rate=0.1, max_depth=3)#intilizing
         model3.fit(p_train_balanced, y_train_balanced)

         ypred3 = model3.predict(p_test_scaled)#predicting
         ypredproba3 = model3.predict_proba(p_test_scaled)[:, 1]

         print("Confusion Matrix:")#evaluvating
         print(confusion_matrix(y_test, ypred3))
         print("\nClassification Report:")
         print(classification_report(y_test, ypred3))
         roc_auc = roc_auc_score(y_test, ypredproba3)#calculate ROC score
         print(f"\nROC-AUC Score: {roc_auc:.4f}")

         fpr, tpr, thresholds = roc_curve(y_test, ypredproba3)
         plt.figure(figsize=(8, 6))
         plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {roc_auc:.4f})")
         plt.plot([0, 1], [0, 1], 'k--', label="Random Guess")
         plt.xlabel("False Positive Rate")
         plt.ylabel("True Positive Rate")
         plt.title("ROC Curve")
         plt.legend()
         plt.show()
```

Fig.8

In this section, Gradient boosting classifier is imported from the sklearn.ensemble library and the base settings is implemented.

Decision tree

Here, for executing the decision tree model, decision tree classifier is imported and following base code is performed as shown in fig.9.

Decision Tree

```
In [39]: from sklearn.tree import DecisionTreeClassifier

model4 = DecisionTreeClassifier(random_state=42, max_depth=None, min_samples_split=2)#initilizing
model4.fit(p_train_balanced, y_train_balanced)#fitting the model
ypred4 = model4.predict(p_test_scaled)#predicting on test data
ypredproba4 = model4.predict_proba(p_test_scaled)[:, 1]

print("Confusion Matrix:")#evaluvating the model
print(confusion_matrix(y_test, ypred4))
print("\nClassification Report:")
print(classification_report(y_test, ypred4))

roc_auc = roc_auc_score(y_test, ypredproba4)#calculating ROC score
print(f"\nROC-AUC Score: {roc_auc:.4f}")

fpr, tpr, thresholds = roc_curve(y_test, ypredproba4)#plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {roc_auc:.4f})")
plt.plot([0, 1], [0, 1], 'k--', label="Random Guess")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()
```

Fig.9

Support vector machine(SVM)

While applying the support vector machine model , the SVC library is utilized , along with the base code for implementing this model is performed such as initialization, fitting and predicting as shown in fig.10.

```
In [40]: from sklearn.svm import SVC

model5 = SVC(kernel='rbf',C=1.0,gamma='scale',probability=True,random_state=42)#intilizing the svm model
model5.fit(p_train_balanced, y_train_balanced)#fitting yhe model
ypred5 = model5.predict(X_test_scaled)
ypredproba5 = model5.predict_proba(X_test_scaled)[:, 1]

cm = confusion_matrix(y_test, ypred5)#calculating confusion matrix
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

print("Classification Report:\n")
print(classification_report(y_test, ypred5))
roc_auc = roc_auc_score(y_test, ypredproba5)#identify ROC score
print(f"ROC-AUC Score: {roc_auc:.4f}")

fpr, tpr, thresholds = roc_curve(y_test, ypredproba5)
plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {roc_auc:.4f})", color='darkorange')
plt.plot([0, 1], [0, 1], 'k--', label="Random Guess", color='navy')
plt.fill_between(fpr, tpr, alpha=0.2, color='lightblue')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver Operating Characteristic (ROC) Curve")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```

Fig.10

Neural network model

```
In [6]: smote = SMOTE(random_state=42)#Applying the oversampling technique smote for handling imbalance

        p_train_balanced, y_train_balanced = smote.fit_resample(p_train_scaled, y_train)
        # neural netwok model
        neuralmodel = Sequential([Dense(128, activation='relu', input_shape=(p_train_balanced.shape[1],)),Dropout(0.3),Dense(64, activati
        neuralmodel.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])#compling the model
        neuralmodel.fit(p_train_balanced, y_train_balanced, validation_split=0.2, epochs=50, batch_size=32, verbose=1)#training the model
        neuralproba = neuralmodel.predict(p_test_scaled).flatten()#predicting the model
```

```
Epoch 42/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9873 - loss: 0.0390 - val_accuracy: 1.0000 - val_loss: 0.0058
Epoch 43/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9906 - loss: 0.0330 - val_accuracy: 1.0000 - val_loss: 0.0055
Epoch 44/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9881 - loss: 0.0323 - val_accuracy: 1.0000 - val_loss: 0.0086
Epoch 45/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9893 - loss: 0.0294 - val_accuracy: 1.0000 - val_loss: 0.0057
Epoch 46/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.9883 - loss: 0.0385 - val_accuracy: 1.0000 - val_loss: 0.0078
Epoch 47/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9909 - loss: 0.0327 - val_accuracy: 1.0000 - val_loss: 0.0053
Epoch 48/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.9961 - loss: 0.0183 - val_accuracy: 1.0000 - val_loss: 0.0099
Epoch 49/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9904 - loss: 0.0236 - val_accuracy: 1.0000 - val_loss: 0.0077
Epoch 50/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9932 - loss: 0.0247 - val_accuracy: 1.0000 - val_loss: 0.0051
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
```

Fig.11

Here we excueted the neural network model by applying the SMOTE technique , followed by intilizing, compling the model, training and predicting the model as shown in fig.11

Hybrid model using SMOTE

```
In [50]:
gbmodel = xgb.XGBClassifier(n_estimators=100,max_depth=4,learning_rate=0.1,subsample=0.8,colsample_bytree=0.8,random_state=42,use
gbmodel.fit(p_train_balanced, y_train_balanced)
gbproba = gbmodel.predict_proba(p_test_scaled)[:, 1]#predicting with graident boosting


stacked_p = np.column_stack((neuralproba, gbproba))#stacking the both base model predictions
model6 = LogisticRegression(random_state=42)#train the metamodel,here its logistic regression
model6.fit(stacked_p, y_test)
stacked_pred = model6.predict(stacked_p)#prediciting the hybrid model
stacked_proba = model6.predict_proba(stacked_p)[:, 1]
print("Confusion Matrix:")#evaluvating the hybrid model
print(confusion_matrix(y_test, stacked_pred))
print("\nClassification Report:")
print(classification_report(y_test, stacked_pred))

roc_stacked = roc_auc_score(y_test, stacked_proba)#identify ROC score
print(f"Stacked Model ROC-AUC: {roc_auc_stacked:.4f}")

fpr, tpr, thresholds = roc_curve(y_test, stacked_proba)#plot roc curve
plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, label=f"Stacked Model ROC Curve (AUC = {roc_auc_stacked:.4f})", color='darkorange')
plt.plot([0, 1], [0, 1], 'k--', label="Random Guess", color='navy')
plt.fill_between(fpr, tpr, alpha=0.2, color='lightblue')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver Operating Characteristic (ROC) Curve")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```

Fig.12

In this case, the neural network model for the hybrid model is already excuted in the above section(fig.11), then we predicting the gradient boosting model and stacked together the predictions of both neural network model and gradient bosting, which is already applied the technique SMOTE. Then we train a logistic regression model as a metamodel to perform the hybrid model and we perform the base setting such as predicting the hybrid model, evaluting and plot the ROC curve.

Hybrid model using ADASYN

Here , we apply another oversampling technique called ADASYN and import the libararies for running the code as shown in fig.13. then slipt the data and apply the ADASYN technique.

Applying another over sampling tecnique ADASYN

```
In [51]: import pandas as pd
from imblearn.over_sampling import ADASYN
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score


W = Asthma_data.drop(columns=['Diagnosis', 'PatientID', 'DoctorInCharge'])
y = Asthma_data['Diagnosis']
w_train, w_test, y_train, y_test = train_test_split(w, y, test_size=0.3, random_state=42, stratify=y)#splitting the dataset

# Scale features
scaler = StandardScaler()
w_train_scaled = scaler.fit_transform(w_train)
w_test_scaled = scaler.transform(w_test)

adasyn = ADASYN(sampling_strategy='minority', random_state=42, n_neighbors=5)#applying adasyn
w_train_adasyn, y_train_adasyn = adasyn.fit_resample(w_train_scaled, y_train)

print(f"Original dataset shape: {X_train.shape}")
print(f"Resampled dataset shape: {X_train_adasyn.shape}")

Original dataset shape: (1674, 26)
Resampled dataset shape: (3157, 26)
```

Fig.13

Applying on Hybrid model

```
In [53]: from scikeras.wrappers import KerasClassifier
from sklearn.ensemble import StackingClassifier
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from xgboost import XGBClassifier
from sklearn.pipeline import Pipeline

#Neural Network model
def build_nn_model():
    model = Sequential()
    model.add(Dense(128, input_dim=w_train_adasyn.shape[1], activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model

model7 = KerasClassifier(build_fn=build_nn_model, epochs=50, batch_size=32, verbose=0)

# Define the Gradient Boosting model
gb_model = XGBClassifier(scale_pos_weight=1, random_state=42)

# Combine models using Stacking
stacking_model2 = StackingClassifier(estimators=[('nn', model7),('gb', gb_model)],final_estimator=LogisticRegression(),cv=5,n_jot

stacking_model2.fit(w_train_adasyn, y_train_adasyn)
```

Fig.14

Here we apply the adasyn technique to the two base models such as neural networks and gradient boosting, then we combain them by using stacking method and the following base setting is performed as shown in fig.14.

References

Batchelder, N., 2024. The Python Standard Library [WWW Document]. Python Doc. URL https://docs.python.org/3/library/index.html (accessed 12.10.24).