

## I/ Data preprocessing

### 1.1. Overview description of database and data cleaning(missing values, duplicates,...)

The initial phase involves data cleaning, which includes removing irrelevant data, fixing errors, imputing missing values and identifying duplicates. The details are shown in **Table 1**:

**Table 1: Data description and data cleaning actions**

	Instances	Features	Missing values	Other Accuracy	Solution
<b>DB1</b>	Total: 569 cases +Malignant: 212 (37%) + Benign: 357 (63%)	32 columns: + Target feature: Class column ( M = malignant, B = benign) + Predictor features (numerical): the standard error, mean, and worst values of ten cell nucleus.	None	None	
<b>DB2</b>	Total: 286 cases +Recurrence events: 85 (30%) + No recurrence events : 201 (70%)	10 columns: + Target feature: Class column(recurrence-events , no-recurrence-events) +Predictor features (categorical): age, menopause,tumor size, inv- nodes, node caps, degree of malignancy, breast position, breast quadrant and irradiation therapy.	+node_caps: 8 +breast_quad: 1(denoted by “?”)	All predictor features are categorical	+ Missing values: Replace them with Mode + Encoding Categorical features to numerical because they are relevant to analysis.
<b>DB3</b>	Total: 961 cases + Benign: 516 + Malignant: 445  <i>After cleaning:</i> Total: 830 cases + Benign: 427 (51%) + Malignant: 403 (49%)	6 columns: +Target feature: benign=0 or malignant=1 + Predictor features: .BI-RADS assessment(ordinal) . Age (integer) . Shape (nominal) . Margin(nominal) . Density (ordinal)	+BI-RADS assessment: 2 + Age: 5 + Shape: 31 +Margin: 48 + Density: 76	None	Drop missing values.
<b>DB4</b>	Total: 699 cases	11 columns:	+ Bare Nuclei:	None	Drop missing

	+ Benign: 458 + Malignant: 241  <i><b>After cleaning:</b></i> Total: 683 cases + Benign: 444 (65%) + Malignant: 239 (35%)	+ Target feature: benign=2 or malignant=4 + Predictor features (numerical): Uniformity of Cell Shape, Clump Thickness, Uniformity of Cell Size, Mitoses, Bare Nuclei, Bland Chromatin, Normal Nucleoli, Marginal Adhesion, and Single Epithelial Cell Size.	16		values.
<b>DB5</b>	Total: 24 cases +docetaxel-resistant tumors: 14 (58%) +docetaxel-sensitive tumors: 10 (42%)	9486 features: +Target feature: docetaxel-resistant tumors and docetaxel-sensitive tumors. +Predictor features (numerical): information about breast cancer core biopsy from patients  <b>After feature reduction: 8 components</b>	None	Number of predictor features is too large	Reduce dimensionality with Principal Component Analysis (PCA) - 8 components.
<b>DB6</b>	Total: 198 cases  <i><b>After cleaning:</b></i> Total: 194 cases + non-recurrent events: 148 (76%) + recurrent events: 46 (24%)	34 features: + Target feature: Class: R = recur, N = nonrecur + Predictor features (Numerical): radius, texture, perimeter, area, smoothness, compactness, concavity, concave points, symmetry, fractal dimension, tumor size, lymph node status	Lymph node status: 4	None	Drop the missing value
<b>DB7</b>	Total: 4024 cases + Alive: 3408 (85%) + Dead: 616 (15%)	16 features: + Target feature: Status: Alive and Dead + Predictor features: <b>Nominal (9):</b> Race, Marital Status, T Stage, N Stage, 6th Stage, Grade,	None	There are 8 nominal features which are irrelevant to analysis	Drop the irrelevant nominal features that might be the noises for the model

		A Stage, Estrogen Status, Progesterone Status. <b>Numerical (5):</b> 'Age', 'Tumor Size', 'Reginol Node Positive', 'Survival Months', 'Regional Node Examined'			training.
<b>DB8</b>	Total: 116 cases +healthy control: 52 (45%) + patient: 64 (55%)	10 features: + Target feature: Class: 1=Healthy controls, 2=Patients + Predictor feature (numerical): Age, BMI, Glucose, Insulin, HOMA, Leptin, Adiponectin, Resistin, MCP-1	None	None	

#### **Imbalance classification ?:**

The class distribution in all 8 datasets are considered to be not highly imbalanced. The ratio of class distribution in DB6 and DB7 is 76:24 and 85:15 respectively which might be a moderate class distribution. In this case, we can use resampling techniques to address the imbalance issue.

#### **Missing values/Duplicates/Error data?:**

The missing values in DB3, DB4, and DB6 (which are denoted by NaN or "?") were removed from the datasets to ensure data consistency and quality for calculations and modeling.

In DB2, where missing values are denoted by "?," they were replaced with the most frequent value for the respective categorical variables to preserve category distribution.

#### **Feature selection and reduction?:**

In DB5, the number of features is too large, the method called *PCA (Principal Component Analysis)* is used to reduce dimensionality from 9486 features to 8 features (uncorrelated variables) while preserving as much as variations in dataset as possible.

Code we used:

```
In [21]: from sklearn.decomposition import PCA

# Create a PCA instance with 8 components
pca = PCA(n_components=8)

# Extract the 'Disease_state' column for later
disease_state = norm_db5['Disease_state']

# Perform PCA on the numeric columns
numeric_data = norm_db5.drop(columns=['Disease_state']) # Remove 'Disease_state' for PCA
pca_result = pca.fit_transform(numeric_data)

# Create a new DataFrame with the PCA results and the 'Disease_state' column
pca_db5 = pd.DataFrame(data=pca_result, columns=[f'PCA_{i+1}' for i in range(8)])
pca_db5['Disease_state'] = disease_state
pca_db5
```

### Explain:

PCA module is imported from scikit-learn library. And then apply PCA to all the numeric predictor features, transforming it into a new set of 8 uncorrelated variables (principal components). Another important thing is that database is required to be standardized before applying PCA method. The use of PCA is basically to reducing noise, speeding up machine learning algorithms.

In DB7, irrelevant nominal features (Race, Marital Status,...) that don't contribute significantly to the target variable (the probability of outcomes: Alive or Dead) are removed. This reduces noise in the dataset and improves model performance by focusing on the most informative features.

### Code we used:

```
In [18]: # Select specific columns
db7mod = db7[['Age', 'Tumor Size', 'Reginol Node Positive', 'Survival Months', 'Regional Node Examined', 'Status']]
db7mod
```

In the above code, we used the code which created a new dataframe containing the specific columns.

## 1.2. Data transformation

After cleaning data, the next crucial step is data transformation which involves label-encoding techniques and feature scaling (normalization and standardization).

### **Label-encoding techniques:**

*Why do we need to encode the Categorical variables?*

For categorical features as in DB2, **label-encoding technique** was employed to convert categorical variables into numerical format because the machine learning algorithm just understand and work on numerical data.

### Code we used:

```
In [62]: from sklearn.preprocessing import LabelEncoder

# Initialize the LabelEncoder
label_encoder = LabelEncoder()

# Iterate through categorical columns and apply label encoding
categorical_columns = ['age', 'menopause', 'tumor_size', 'inv_nodes', 'node_caps', 'breast', 'breast_quad', 'irradiat',
for col in categorical_columns:
    db2[col] = label_encoder.fit_transform(db2[col])
db2
```

### Explain:

We are using LabelEncoder from sklearn to convert the string categorical to a numerical value.

Each unique value in a column is mapped to a specific integer. For example, in the 'age' column, the values ['20-29', '30-39', '40-49', '50-59', '60-69', '70-79'] were encoded as [0, 1, 2, 3, 4, 5] after label encoding.

### **Feature scaling:**

*What is the feature scaling and Why we need rescaling feature?*

The next critical step in the process is feature scaling to address variations in units and ranges among numerical features.

Normalization and standardization are two commonly used techniques for feature scaling:

- Normalization: scaling the values of a feature to a range between 0 and 1

$$\text{Normalized Value} = (X - \min) / (\max - \min) * (b - a) + a$$

Where:

X: The original data point you want to normalize.

min: The minimum value in the original dataset for the feature.

max: The maximum value in the original dataset for the feature.

a: The minimum value of the new range you want to scale to (usually 0).

b: The maximum value of the new range you want to scale to (usually 1).

- Standardization: transforming the values of a feature to have a mean of 0 and a standard deviation of 1. The shape of the data distribution wouldn't be changed.

$$z = \frac{x - \mu}{\sigma}$$

Where:

z is the standardized value.

x is the original data point.

$\mu$  is the mean of the feature.

$\sigma$  is the standard deviation of the feature.

Since normalization has a restricted range and standardization does not, normalization is more likely to be affected by outliers, where standardization is less likely to be affected by outliers.

Normalization is useful for when a distribution is unknown or not normal (not bell curve), while standardization is useful for normal distributions.

### Proposed scaling method for each database:

To determine whether to choose normalization or standardization for each database, we initially transformed the datasets into both normalized and standardized versions. Afterward, we compared the Auto-ML accuracy scores achieved using both scaling techniques for each database and selected the scaling technique that yielded the higher accuracy score. Then, we have the proposed scaling methodology table as bellow:

Database	Proposed scaling method
DB1	Standardization to the population mean ( $\mu$ ) = 0, and variance (var) = 1
DB2	Standardization to the population mean ( $\mu$ ) = 0, and variance (var) = 1
DB3	Normalization in the interval [0,1]
DB4	Standardization to the population mean ( $\mu$ ) = 0, and variance (var) = 1
DB5	Standardization to the population mean ( $\mu$ ) = 0, and variance (var) = 1
DB6	Normalization in the interval [0,1]
DB7	Normalization in the interval [0,1]
DB8	Standardization to the population mean ( $\mu$ ) = 0, and variance (var) = 1

#### Code for Normalization:

```
In [74]: from sklearn.preprocessing import MinMaxScaler

# Extract the Class column
class_column = db2.pop('Class')

# Initialize the MinMaxScaler
scaler = MinMaxScaler()

# Normalize the numeric columns
norm_db2 = pd.DataFrame(scaler.fit_transform(db2), columns=db2.columns)

# Add back the Class column
norm_db2 = pd.concat([class_column, norm_db2], axis=1)
norm_db2
```

#### Explain:

In the above code, MinMaxScaler object is used for feature scaling to the range [0, 1]. After doing normalization, all numeric features in dataset have a range of [0,1], making it easier for machine learning algorithms to work effectively.

#### Code for Standardization:

```
In [68]: from sklearn.preprocessing import StandardScaler
# Extract the features from the DataFrame
features = db2.drop(columns=['Class'])

# Initialize the StandardScaler
scaler = StandardScaler()

# Perform data standardization on the features
features_standardized = scaler.fit_transform(features)

# Convert the standardized features back to a DataFrame
features_standardized_df = pd.DataFrame(features_standardized, columns=features.columns)

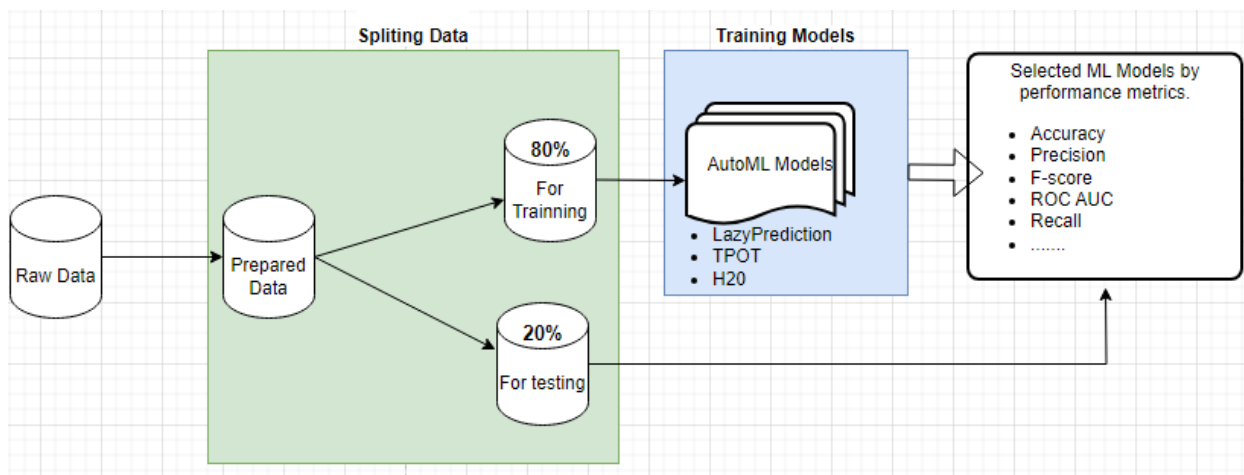
# Combine the standardized features with the 'Class' column
db2_standardized = pd.concat([db2['Class'], features_standardized_df], axis=1)
db2_standardized
```

### Explain:

To standardize data in Python, we applied fit\_transform method, which calculates the mean and standard deviation of each feature and scales the data accordingly.

## II/AUTO-ML

### 2.1/ Auto ML Process and Performance Metrics



There are a lot of metrics which help to understand the model performance. Based on AutoML evaluation, it would use different metrics to evaluate ML models. In the breast cancer research, the writer would evaluate machine learning models based on Accuracy results.

### Confusion Matrix

Confusion Matrix is a matrix used to determine the performance of the classification models. It is a table ( contingency table) to analyze the classification performance by showing the number of True Positive, True Negative, False Positive, and False Negative.

	Predicted Positive	Predicted Negative
Actual Positive	True Positive ( TP)	False Negative (FN)
Actual Negative	False Positive ( FP)	True Negative ( TN)

True Positive (TP): The patient has breast cancer, and the model predicts "breast cancer"

False Positive (FP): The patient is healthy, but the model predicts "breast cancer"

True Negative (TN): The patient is healthy, and the model predicts "healthy"

False Negative (FN): The patient has breast cancer, and the model predicts "healthy"

### Accuracy

It measures the proportion of correct predictions (TP + TN) made by the model out of all the predictions.

- $\text{Accuracy} = (\text{True Positives} + \text{True Negatives}) / (\text{All Prediction})$

Accuracy is an important metric in evaluating the performance of a binary classification model. This metric measures the proportion of correct predictions.

### Precision

Precision is the proportion of true positives among all the predicted positive cases ( TP + FP).

This metric will indicate percentages of “True Positive of all “Predicted Positive” cases.

- $\text{Precision} = \text{True Positives} / (\text{All Predicted Positive})$

### Recall

Recall is the proportion of true positives among all actual positive cases( TP+FN).This metric will indicate the percentage of “True Positive” of all Actual Positive cases.

- $\text{Recall} = \text{True Positives} / (\text{Actual Positive})$

### F1 Score

The F1 score uses the Precision and the Recall metric to calculate and provide a more balanced measure of a model's performance.

- $\text{F1-score} = 2 * (\text{Precision} * \text{Recall}) / (\text{Precision} + \text{Recall})$

### ROC-AUC

ROC-AUC score provides a measure of how well the model can distinguish between the two classes (positive and negative) across different threshold values.

A higher ROC-AUC score indicates better discrimination between the classes, with a score of 1 indicating perfect separation and a score of 0.5 indicating no discrimination.



## 2.2/ Overview results

Why choosing “accuracy score” to evaluate:

+ It calculates the ratio of correctly predicted instances to the total number of instances. So, it, basically, answers the question: "Out of all the instances in the dataset, how many did the model classify correctly?"

+ The imbalance of class distribution on all databases is not considered too high.

This table below give information about the results of paper:

*Note: the red color represents the results of paper*

	Lazy Predict	H2O	TPOT	MLJar	Best Auto-ML in paper	Preproduced Best Auto-ML
<b>DB1</b>	99.12% (99.12%)	100% (98%)	99.12% (98.24%)	99.12% (99.12%)	Lazy & MLJar	H2O
<b>DB2</b>	83.92% (83.92%)	87.5% (79.37%)	74.13% (78.57%)	72.41% (75%)	Lazy	Lazy
<b>DB3</b>	78.31% (78.31%)	80.68% (75.3%)	78.91% (80.12%)	80.72% (79.52%)	MATLAB (87.3%)	H2O
<b>DB4</b>	99% (98.54%)	97.96% (97.57%)	97.81% (97.81%)	97.81% (97.81%)	Lazy	Lazy
<b>DB5</b>		**Error (data is too small)	60% (50%)	60% (50%)	Lazy -MATLAB	TPOT MLJAR -
<b>DB6</b>		81.39% (68.18%)	74.36% (75.50%)	64.10% (67.5%)	Orange (83.5%)	H2O
<b>DB7</b>		90.97% (90.92%)	91.18% (91.18%)	91.18% (91.05%)	TPOT	TPOT MLJAR -
<b>DB8</b>		72.72% (82.97%)	62.5% (50%)	58.3% (70.83%)	MATLAB (91.3%)	

### Summary:

- Lazy Predict shows strong performance on some datasets; however, Lazy Predict does not allow tuning hyperparameter.
- H2O consistently achieves high accuracy on most databases, outperforming other Auto ML like TPOT and MLJar. H2O's accuracy score is also closely matches or surpasses the paper's results on various datasets.
- When it comes to evaluate how Auto-ML perform well or not on each database and check overfitting, we assess based on:

- + Train Accuracy is close to Test Accuracy and both of them are higher than 95%: Indicates a well-performing model.
- + Train Accuracy and Test Accuracy are from 80% to 95%: Indicates moderate performance
- + Train Accuracy and Test Accuracy are lower than 80%: Indicates low performance
- + Train Accuracy > Test Accuracy by a large margin: Indicates overfitting.

## 2.3. Auto-ML tools

### 2.3.1. Lazy predict

a. Version

Report version: lazypredict==0.2.12

b. Code used and output

#### Sumarize the evaluated result from 4 Dataset

	DB1	DB2	DB3	DB4
<b>Pre produced</b>	99.12%	83.92%	78.31%	99%
<b>Paper</b>	99.12%	83.92%	78.31%	98.54%

Break-down the code we used:

```
df
pd.read_csv(r"D:\Data_Analytics\pythonProject1\code\breast-cancer-datasets-and-codes\DB-csv-
files\modDB1.csv")
# basic data preparation
X = np.array(df.drop(['class'],axis=1)) #input
X = X.astype('float32')
y = np.array(df['class']) #output
# integer encode
y = LabelEncoder().fit_transform(y)
```

We load each dataset by pandas and remove the categorical variable in the dataset. Then, we can use the LabelEncode function to convert categorical variables to numerical variables.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=123)
```

Split the dataset into 80% training and 20% test set:

```
clf = LazyClassifier(verbose=0, ignore_warnings=True, custom_metric=None)
models, predictions = clf.fit(X_train, X_test, y_train, y_test)

print(models)
```

Next, we use the clf classifier to fit (train) the data using the various classification algorithms and predict the outcome by printing models variable.

**There are detailed results of lazy prediction for 4 Dataset:**

**Dataset 1: Top 5 ML Models get the highest scores**

Model	Accuracy	Balance Accuracy	ROC AUC	F1-Score
LinearSVC	0.99	0.99	0.99	0.99
ExtraTreeClassifier	0.99	0.99	0.99	0.99
RandomForestClassifier	0.99	0.99	0.99	0.99
QuadraticDiscriminantAnalysis	0.99	0.99	0.99	0.99
Perceptron	0.99	0.99	0.99	0.99

**Dataset 2: Top 5 ML Models get the highest scores:**

Model	Accuracy	Balance Accuracy	ROC AUC	F1-Score
Perceptron	0.84	0.8	0.8	0.84
XGBClassifier	0.75	0.67	0.67	0.74
AdaBoostClassifier	0.79	0.66	0.66	0.75
BernoulliNB	0.75	0.65	0.65	0.73
LGBMClassifier	0.77	0.65	0.65	0.74

**Dataset 3: Top 5 ML Models get the highest scores**

Model	Accuracy	Balance Accuracy	ROC AUC	F1-Score
-------	----------	------------------	---------	----------

AdaBoostClassifier	0.78	0.78	0.78	0.78
LGBMClassifier	0.78	0.78	0.78	0.78
QuadraticDiscriminantAnalysis	0.77	0.77	0.77	0.77
XGBClassifier	0.77	0.77	0.77	0.77
LinearSVC	0.77	0.77	0.77	0.77

**Dataset 4: Top 5 ML Models get the highest scores**

Model	Accuracy	Balance Accuracy	ROC AUC	F1-Score
KNeighborsClassifier	0.99	0.99	0.99	0.99
CalibratedClassifierCV	0.99	0.99	0.99	0.99
SGDClassifier	0.99	0.99	0.99	0.99
QuadraticDiscriminantAnalysis	0.98	0.98	0.98	0.98
PassiveAggressiveClassifier	0.98	0.98	0.98	0.98

In auto machine learning, Lazy\_Prediction, accuracy, balanced accuracy, ROC AUC (Receiver Operating Characteristic Area Under the Curve), and F1-score are common performance metrics used to evaluate the performance of classification models.

As Accuracy is used to measure the proportion of correct predictions, in this test, a higher accuracy indicates that the model is making more correct predictions, while a lower accuracy suggests that the model is making more errors in its predictions.

## 2.2. H20

Reporting version h2o-3.42.0.3

Article version h2o-3.36.1.1

Requirements:

Install: dependencies packages and JavaJDK 17

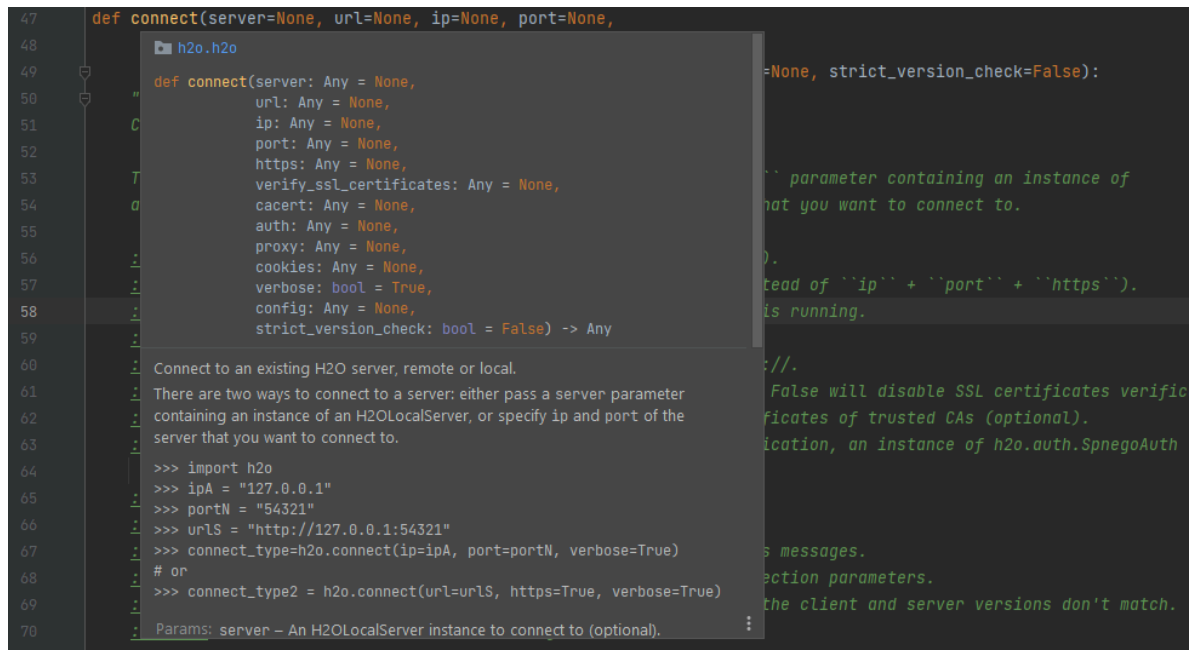
```
pip install requests
pip install tabulate
pip install future
pip install matplotlib
pip install -f http://h2o-release.s3.amazonaws.com/h2o/latest_stable_Py.html h2o
```

```
(pythonProject1) PS D:\Data_Analytics\pythonProject1> java --version
openjdk 17 2021-09-14
OpenJDK Runtime Environment (build 17+35-2724)
OpenJDK 64-Bit Server VM (build 17+35-2724, mixed mode, sharing)
```

### H2O interface support – H2O Flow

H2O supports a GUI web app ( H2OFlow), where we can trace the running jobs and work logs. After initial H2O, it would init H2OFlow as a web server and open connection on port 54321. As this is a default configuration, we can access the server site by this URL “<http://localhost:54321/flow/index.html>”.

If we need to define these parameters, we could edit them in the library h2o.py file.



```
47 def connect(server=None, url=None, ip=None, port=None,
48             h2o.h2o
49             def connect(server: Any = None,
50                         url: Any = None,
51                         ip: Any = None,
52                         port: Any = None,
53                         https: Any = None,
54                         verify_ssl_certificates: Any = None,
55                         cacert: Any = None,
56                         auth: Any = None,
57                         proxy: Any = None,
58                         cookies: Any = None,
59                         verbose: bool = True,
60                         config: Any = None,
61                         strict_version_check: bool = False) -> Any
62
63     """
64     Connect to an existing H2O server, remote or local.
65     There are two ways to connect to a server: either pass a server parameter
66     containing an instance of an H2OLocalServer, or specify ip and port of the
67     server that you want to connect to.
68
69     >>> import h2o
70     >>> ipA = "127.0.0.1"
71     >>> portN = "54321"
72     >>> urlS = "http://127.0.0.1:54321"
73     >>> connect_type=h2o.connect(ip=ipA, port=portN, verbose=True)
74     # or
75     >>> connect_type2 = h2o.connect(url=urlS, https=True, verbose=True)
76
77     Params: server – An H2OLocalServer instance to connect to (optional).
78
79     """
80
81     if server is not None:
82         # parameter containing an instance of
83         # that you want to connect to.
84         return server
85
86     # instead of ``ip`` + ``port`` + ``https``.
87     # is running.
88
89     # False will disable SSL certificates verification
90     # of trusted CAs (optional).
91     # If you want to use a custom certificate, an instance of h2o.auth.SpnegoAuth
92
93     # messages.
94     # connection parameters.
95     # the client and server versions don't match.
```



**Code:**

Running h2o on the local machine may miss an ML model XGBoost. When we run the H2O, it will return a warning message.

```
Loading file!!!  
Parse progress: |██████████████████████████████████████████████████████████████████████████| (done) 100%  
AutoML progress: |██████████████████████████████████████████████████████████████████████████|  
00:28:17.342: AutoML: XGBoost is not available; skipping it.
```

XGBoost models in AutoML use GPUs to run the algorithm. However, the current version does not support running on Windows & macOS, it only works on GPU cloud ( example: GoogleCollab).

H2O supports different ways to import datasets with 2 functions ( `upload_file` and `import_file`.

The import function is a parallelized reader and pulls information from the server from a location specified by the client. The path is a server-side path.

The upload function is a push from the client to the server. The specified path must be a client-side path. The client pushes the data from a local filesystem (for example, on your machine where R or Python is running) to the H2O server.

```
print("H2O Version:", h2o.__version__)
# Import a sample binary outcome train/test set into H2O
print("Loading file!!! ")
prostate = h2o.upload_file(r"D:\Data_Analytics\pythonProject1\code\breast-cancer-datasets-and-codes\DB-
csv-files\modDB1.csv")

prostate.head()

# split into train and testing sets
train, test = prostate.split_frame(ratios = [0.8], seed = 123)
x = train.columns
y = "class"
x.remove(y)
# For binary classification, response should be a factor
train[y] = train[y].asfactor()
test[y] = test[y].asfactor()
```

The dataset is split into 2 parts ( training & testing ), the ratio = 0.8, which means training = 80% and testing = 20%.

```
aml = H2OAutoML(max_models=20, seed=1)
aml.train(x=x, y=y, training_frame=train)
```

```
# View the AutoML Leaderboard
lb = aml.leaderboard
lb.head(rows=lb.nrows)
```

Training 20 base models in H2O with 80% dataset, and then the performance is saved on the H2O leaderboard.

```
best_model = aml.get_best_model()
print(" ##### Best Models !!! #####")
print(best_model)
```

Print the best model, which has the highest accuracy metric in the leaderboard.

**Best Models - Model performance (test) !!!**

ModelMetricsBinomial: gbm

**\*\* Reported on test data. \*\***

MSE: 0.006151130859350847

RMSE: 0.07842914547125225

LogLoss: 0.02633047308446971

Mean Per-Class Error: 0.0

AUC: 1.0

AUCPR: 1.0

Gini: 1.0

```
best_model_accuracy = best_model.accuracy()

print(" Best Models - Model Accuracy !!! ")
print(best_model_accuracy)
```

**Best Models - Model Accuracy !!!**

[[0.7732624753502069, 1.0]]



```

Best Models - Model Accuracy !!!
[[0.7732624753502069, 1.0]]
#### Checking Leader Board ####
model_id      auc      logloss      aucpr      mean_per_class_error      rmse      mse
GBM_grid_1_AutoML_1_20230919_02457_model_1      0.991852      0.113645      0.988577      0.0469858      0.17943      0.0321951
DeepLearning_1_AutoML_1_20230919_02457      0.991831      0.104176      0.989785      0.0350177      0.165606      0.0274252
GLM_1_AutoML_1_20230919_02457      0.99181      0.0925777      0.991035      0.0303318      0.153058      0.0234267
StackedEnsemble_AllModels_1_AutoML_1_20230919_02457      0.991631      0.0967028      0.989917      0.0314716      0.160327      0.0257049
StackedEnsemble_BestOffFamily_1_AutoML_1_20230919_02457      0.991156      0.0985054      0.989528      0.0314716      0.160764      0.0258449
GBM_2_AutoML_1_20230919_02457      0.990924      0.124725      0.986938      0.0427432      0.189779      0.0360162
GBM_1_AutoML_1_20230919_02457      0.990079      0.11493      0.987112      0.0421099      0.180304      0.0325094
GBM_3_AutoML_1_20230919_02457      0.98837      0.132268      0.984691      0.0517351      0.190671      0.0363554
DRF_1_AutoML_1_20230919_02457      0.985984      0.201843      0.982661      0.0498354      0.197601      0.0390462
GBM_4_AutoML_1_20230919_02457      0.985383      0.137404      0.982278      0.0547112      0.195485      0.0382144
XRT_1_AutoML_1_20230919_02457      0.983853      0.267427      0.982932      0.0469225      0.191562      0.0366961
GBM_5_AutoML_1_20230919_02457      0.979262      0.175036      0.974709      0.0611702      0.218565      0.0477706
[12 rows x 7 columns]

```

## Output:

H2O exported 2 the training model results: training data and cross-validation data

Although 2 results use 80% raw datasets, in training data, models learns from 80% of the data to make prediction (error: zero) while in cross-validation data, models repeatedly splits the data into multiple subsets(folds), trains the model on each subset and evaluate performance on other subsets assess generalization (error existed). Cross-validation will generate different “samples” for training. For example, the confusion matrix of dataset DB1 in the article below

Training data	cross-validation data
Confusion Matrix (Act/Pred) for maxAccuracy @ threshold = 0.773262 <div> <div>B</div> <div>M</div> <div>Error</div> <div>Rate</div> </div> <div> <div>-----</div> <div>---</div> <div>---</div> <div>-----</div> <div>-----</div> </div> <div> <div>B</div> <div>282</div> <div>0</div> <div>0</div> <div>(0.0/282.0)</div> </div> <div> <div>M</div> <div>0</div> <div>168</div> <div>0</div> <div>(0.0/168.0)</div> </div> <div> <div>Total</div> <div>282</div> <div>168</div> <div>0</div> <div>(0.0/450.0)</div> </div>	Confusion Matrix (Act/Pred) for max Accuracy @ threshold = 0.718946 <div> <div>B</div> <div>M</div> <div>Error</div> <div>Rate</div> </div> <div> <div>-----</div> <div>---</div> <div>---</div> <div>-----</div> <div>-----</div> </div> <div> <div>B</div> <div>279</div> <div>3</div> <div>0.0106</div> <div>(3.0/282.0)</div> </div> <div> <div>M</div> <div>14</div> <div>154</div> <div>0.0833</div> <div>(14.0/168.0)</div> </div> <div> <div>Total</div> <div>293</div> <div>157</div> <div>0.0378</div> <div>(17.0/450.0)</div> </div>

Training Data: The model performed perfectly on the data it was trained on. No mistakes were made (0% error).

Cross-Validation Data: The model performed well on new, unseen data, but not as perfectly as on the training data. There were some mistakes (errors), but the overall performance was still good (around 3.78% error).

⇒ **The accuracy result in cross-validation is usually lower than the normal split data, which is shown below table:**

	Accuracy (split data)	Accuracy (Cross-validation)	Accuracy (Test)	PerformanceBest ML models	Evaluation
DB1	100%	96.22%	100%	Gradient Boosting Machine	+ High accuracy, model well performs
DB2	79.37%	75.39%	87.5%	Generalized Linear Modeling	+ Moderate accuracy, model moderately performs
DB3	87.76%	85.63%	80.68%	StackedEnsemble_ BestOfFamily	+ Moderate accuracy, probably overfitting
DB4	97.57%	97.38%	97.96%	Generalized Linear Modeling	+ High accuracy, model well performs
DB6	92.26%	83.87%	81.39%	Generalized Linear Modeling	+ Moderate accuracy, probably overfitting
DB7	90.61%	90.02%	90.97%	StackedEnsemble_ BestOfFamily	+ Model moderately performs
DB8	82.98	80.85	72.72%	Generalized Linear Modeling	+ Low accuracy, probably overfitting

When we use small dataset, H2O would have warned ( e.q DB2, DB6 DB8)

01:04:29.712: \_min\_rows param, The dataset size is too small to split for min\_rows=100.0: must have at least 200.0 (weighted) rows, but have only 94.0

the `_min_rows` parameter in H2O is used to specify the minimum number of observations (rows) that should contain during the model-building process.

The error message, is indicating that the specified value of `min_rows` (100.0) is too large for your dataset.

However, this dataset is relatively small, with only 94.0 (weighted) rows. In this case, we may edit the minimum value for the H2O parameter.

Checking the class distribution is balanced or imbalanced:

For each Dataset, we could briefly check the training/cross-validation/test data either balance or imbalance by confusion matrix result. For example, the confusion matrix DB1, DB2, DB3:

**Database1-train data**

Confusion Matrix (Act/Pred) for maxAccuracy @ threshold = 0.773262

	B	M	Error	Rate
----	---	---	-----	-----
B	282	0	0	(0.0/282.0)
M	0	168	0	(0.0/168.0)
Total	282	168	0	(0.0/450.0)

#### Database2-train data

ModelMetricsBinomialGLM: glm				
** Reported on train data. **				
Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.377724				
	no-recurrence-events	recurrence-events	Error	Rate
-----	-----	-----	-----	-----
no-recurrence-events	157	20	0.113	(20.0/177.0)
recurrence-events	32	43	0.4267	(32.0/75.0)
Total	189	63	0.2063	(52.0/252.0)

#### Database3-train data

ModelMetricsBinomial: gbm				
** Reported on train data. **				
Confusion Matrix (Act/Pred) for maxaccuracy @ threshold = 0.455367				
	0	1	Error	Rate
----	---	---	-----	-----
0	287	56	0.1633	(56.0/343.0)
1	31	280	0.0997	(31.0/311.0)
Total	318	336	0.133	(87.0/654.0)

## 2.3 TPOT

### a. Concept

TPOT, short for Tree-based Pipeline Optimization Tool, is an open-source software designed to automate machine learning processes. It leverages Scikit-learn and uses genetic programming to optimize machine learning pipelines, finding the best hyperparameters and features for maximum performance. The best-performing model is selected based on their performance on a validation dataset. TPOT works by using genetic programming to evolve a pipeline of data processing and machine learning algorithms.

### b. Version and Install

Version on this report: TPOT == 0.12.1

Version on the article: TPOT == 0.11.7

Install: pip install tpot

### c. Codes

## Step 1: Define X(Features) and Y(Labels) for Machine Learning model

```
12 X = np.array(df.drop(['Class'], axis=1))
13 #label_encoder = LabelEncoder()
14 #X = label_encoder.fit_transform(X1)
15 y1 = np.array(df['Class'])
16 label_encoder = LabelEncoder()
17 y = label_encoder.fit_transform(y1)
```

## Step 2: Model optimization and scoring

### \*Code using Cross-Validation

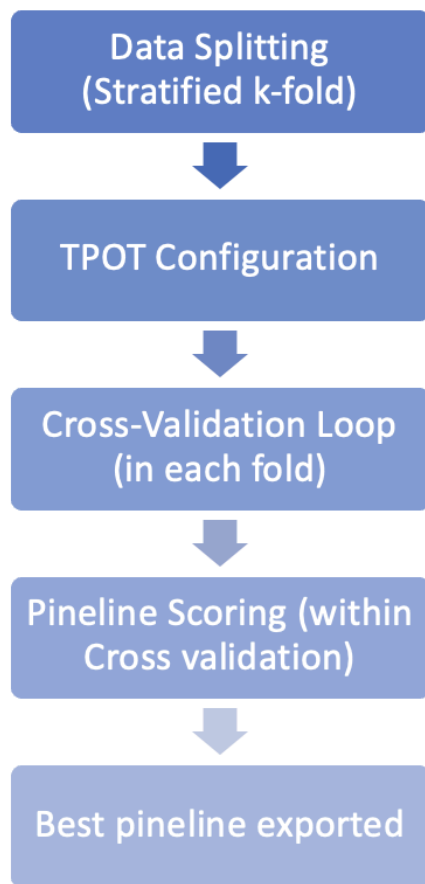
```
# define model evaluation
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=123)
tpot = TPOTClassifier(generations=10, population_size=50, cv=cv, scoring='accuracy', verbosity=2, random_state=1, n_jobs=-1)
tpot.fit(X, y)
tpot.export('bestmodel2.py')
```

### \*Code using Train-Test

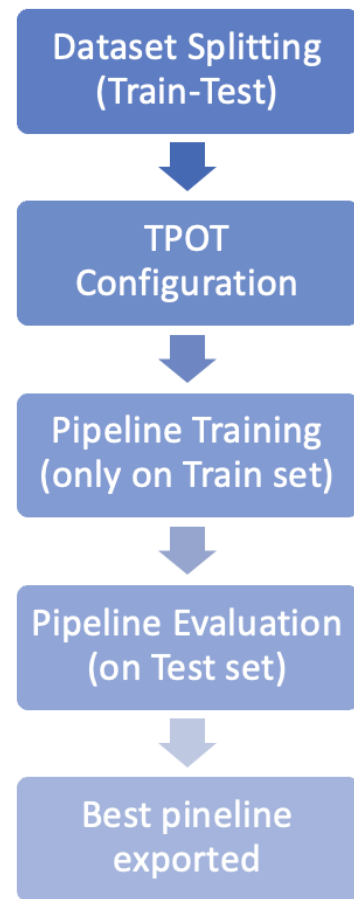
```
X_train, X_test, y_train, y_test = train_test_split(X,y, train_size=0.80, test_size=0.20 ,random_state=123)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
tpot = TPOTClassifier(generations=10,population_size=50,scoring='accuracy', verbosity=2, random_state=1, n_jobs=-1)
tpot.fit(X_train, y_train)
preds = tpot.predict(X_test)
print(accuracy_score(y_test, preds))
```

Workflows of codes using Stratified K-fold cross-validation and Train-Test split is shown in chart below:

### Stratified k-folds Cross Validation method



### Train-Test data split method



Summary some aspects of two methods: cross-validation and Train-test split:

	Cross-validation	Train-Test
<b>Data splitting</b>	Repeated Stratified k-fold: Data is divided into 5 folds. In each fold, class distribution is balanced and stratified data is repeated 3 times.	Data divided into Training (80%) and Test(20%) without considering class distribution.
<b>Model Optimization</b>	10 generations for each fold	10 generations for only the training set.
<b>Scoring metric</b>	Accuracy	Accuracy
<b>Additional Evaluation</b>	None (within CV)	Test set performance

<b>Data Utilization</b>	Maximizes data utilization	Splits data for training
<b>Performance Estimation</b>	More reliable	Slightly variable
<b>Recommended for</b>	Limited data, reliable CV	Large datasets, quick check

In summary, the key difference between the two workflows is how they evaluate and select the best pipeline. Cross-validation provides a more robust estimate of performance by using multiple folds of the data, while the train-test split approach evaluates performance on an independent test set. Cross-validation helps to reduce the risk of overfitting and provides a better assessment of how well the model generalizes to unseen data.

#### d. Output

1/ When comparing the best internal CV score (accuracy score on the train dataset), the results of Cross-validation techniques are mostly lower than those of the Train-Test(80:20) method on all databases.

	<b>Cross-validation accuracy score</b>	<b>Training accuracy score</b>	<b>Test accuracy score</b>	<b>Model well/moderate/low perform ?</b>
<b>DB1</b>	98.18	98.46	99.12	Well perform
<b>DB2</b>	77.14	78.08	74.13	Low perform
<b>DB3</b>	84.98	86.44	78.91	Low perform, probably overfitting
<b>DB4</b>	97.75	97.98	97.81	Well perform
<b>DB5</b>	92.77	90.00	60.00	Low perform, overfitting
<b>DB6</b>	82.44	84.51	74.36	Low perform
<b>DB7</b>	90.32	90.24	91.18	Well perform
<b>DB8</b>	78.88	81.52	62.5	Low perform, probably overfitting

2/ The capture of results:

*Note: **the red color** represents the results of paper*

	<b>Test Accuracy Score</b>	<b>Best ML</b>	<b>Interpretation</b>
DB1	99.12% (98.24%)	LogisticRegression (GradientBoostingClassifier)	+ High accuracy score; + 99.12% of the cases in the dataset are predicted correctly including Benign and Malignant by LogisticRegression model.
DB2	74.13% (78.57%)	RandomForestClassifier (GradientBoostingClassifier)	+ Moderate accuracy score; + 74.13% of the cases in the dataset are predicted correctly including Recurrence events and No Recurrence events by RandomForestClassifier model.
DB3	78.91% (80.12%)	RandomForestClassifier (RandomForestClassifier)	+ Moderate accuracy score; + 78.91% of the cases in the dataset are predicted correctly including Benign and Malignant by RandomForestClassifier model.
DB4	97.81% (97.81%)	RandomForestClassifier (ExtraTreesClassifier)	+ High accuracy score; + 97.81% of the cases in the dataset are predicted correctly including Benign and Malignant by RandomForestClassifier model.
DB5	60% (50%)	DecisionTreeClassifier (RandomForestClassifier)	+ Low accuracy score; + 60% of the cases in the dataset are predicted correctly including docetaxel-resistant tumors and docetaxel-sensitive tumors by Decision Tree Classifier model.
DB6	74.36% (75.50%)	SGDClassifier (GradientBoostingClassifier)	+ Moderate accuracy score; + 74.36% of the cases in the dataset are predicted correctly including non-recurrent events and recurrent events by SGDClassifier model.
DB7	91.18% (91.18%)	RandomForestClassifier (XGBClassifier)	+ High accuracy score; + 91.18% of the cases in the dataset are predicted correctly including Alive and Dead cases by RandomForestClassifier model.
DB8	62.5% (50%)	SGDClassifier (SGDClassifier)	+ Low accuracy score; + 62.5% of the cases in the dataset are predicted correctly including Health control and Patient cases by SGDClassifier model.

### **Summary:**

- DB1 has highest accuracy score, while DB5 has the lowest. Low performance in machine learning models can be attributed to various factors. Here are some reasons related to Data quality the we can assume:
  - + Data size is too small so that models cannot generalize well to new data (DB2, DB5, DB6, DB8)
  - + Poor data quality: noise, redundant features
  - + Imbalanced class distribution when splitting data for Auto-ML process.
- Differences in accuracy and best pipelines between our study and the paper are attributed to version differences
- Logistic Regression, Gradient Boosting, Random Forest, and Stochastic Gradient Descent (SGD) are the best ML model founded by TPOT tools:

#### **Logistic Regression:**

- + *Type*: Supervised Learning (Classification).
- + *Strengths*: Simple, interpretable, works well for linearly separable data, efficient.
- + *Weaknesses*: Limited to binary classification, may underperform on complex data.
- + *Use Cases*: Binary classification problems with linear decision boundaries.

#### **Gradient Boosting:**

- + *Type*: Ensemble Learning (Boosting).
- + *Strengths*: High predictive accuracy, handles complex relationships, robust to outliers.
- + *Weaknesses*: Prone to overfitting (requires tuning), computationally intensive.
- + *Use Cases*: Classification and regression tasks where high accuracy is crucial.

#### **Random Forest:**

- + *Type*: Ensemble Learning (Bagging).
- + *Strengths*: Robust, handles high-dimensional data, reduces overfitting, easy to use.
- + *Weaknesses*: May not capture complex relationships as well as boosting methods.
- + *Use Cases*: Versatile for classification and regression, especially with large datasets.

#### **Stochastic Gradient Descent (SGD):**

- + *Type*: Optimization Algorithm.
- + *Strengths*: Efficient, scalable to large datasets, suitable for deep learning.
- + *Weaknesses*: Sensitive to hyperparameters, may require careful tuning.
- + *Use Cases*: Training neural networks and other large-scale machine learning models.

**Recommendation:** In our study, separate codes for cross-validation and train-test splitting were employed. For future research, we recommend combining Stratified k-fold cross-validation with train-test splitting to ensure robust evaluation and maintain class distribution.

## **2.4. MLJar**

### **a. Concept**



The MLJar is an Automated Machine Learning Python package that works with tabular data. It abstracts the common way to preprocess the data, construct the machine learning models, and perform hyper-parameters tuning to find the best model. It is no black-box as we can see exactly how the ML pipeline is constructed (with a detailed Markdown report for each ML model).

The MLJar will help:

- Explain and understand the data (Automatic Exploratory Data Analysis),
- Try different machine learning models (Algorithm Selection and Hyper-Parameters tuning),
- Create Markdown reports from analysis with details about all models,
- Save, re-run and load the analysis and ML models.

It has four built-in modes of work:

- **Explain** mode, which is ideal for explaining and understanding the data, with many data explanations, like decision trees visualization, linear models coefficients display, permutation importances and SHAP explanations of data,
- **Perform** (which the article used) for building ML pipelines to use in production,
- **Compete** mode that trains highly-tuned ML models with ensembling and stacking, with a purpose to use in ML competitions.
- **Optuna** mode that can be used to search for highly-tuned ML models, should be used when the performance is the most important, and computation time is not limited (it is available from version 0.10.0)

## b. Version and Install

Version on this report: mljar-supervised == 1.0.2

Version on the article: mljar-supervised == 0.11.2

Install:

```
pip install lightwood
```

```
pip install mljar-supervised
```

## c. Codes

```
# Step 1: Data preparation
X = np.array(df.drop(['class'], axis='columns')) #input
#X = X.astype('float32')
y = np.array(df['class']) #output
# integer encode
y = LabelEncoder().fit_transform(y)
```

The independent variables are stored in x by dropping the 'class' column from the DataFrame.

The dependent variable (class) is stored in y.

Label encoding is applied to convert the target variable into numerical values using LabelEncoder.

This is typically done when dealing with classification tasks.

```
# Step 2: Split data
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=123)
```

The test\_size=0.2 argument specifies that 20% of data is allocated to the test set. In other words, 80% of the data will be used for training the machine learning model.

The random\_state=123 argument is used to seed the random number generator. This ensures that if we run this code multiple times, we'll get the same split each time, making results reproducible.

```
# Step 3: Train models with AutoML
automl = AutoML (mode="Perform", eval_metric="accuracy")
#explain_level=0, train_ensemble=True, golden_features=False, features_selection=False
automl.fit(X_train, y_train)
```

The AutoML class is instantiated with specific configurations:

- mode="Perform": Indicates that the AutoML should focus on model performance.
- eval\_metric="accuracy": Specifies the evaluation metric for the models. In this case, it's accuracy.

Note: There are commented-out parameters related to explainability, ensemble learning, golden features, and feature selection, which can be enabled if needed.

```
# Step 4: Compute the accuracy on test data
predictions = automl.predict_all(X_test)
print(predictions.head())
print("Test accuracy:", accuracy_score(y_test, predictions["label"].astype(int)))

print(predictions.head())
print(predictions.tail())
print(X_test.shape, predictions.shape)
print(predictions)
```

The trained AutoML model is used to make predictions on the test data using the predict\_all method. The predictions are stored in the predictions variable. The code then calculates and prints the test accuracy using scikit-learn's accuracy\_score function, which compares the predicted labels to the actual labels (y\_test).

### Cross-Validation tuning:

To tune the cross-validation parameter, we can modify the 'autofe' parameter when instantiating the AutoML class. The 'autofe' parameter controls the number of cross-validation folds used during model training. By default, it's set to 5. (MLJar already uses 5-fold cross-validation technique to enhance the performance of single ML model)

Code to tune the CV set up, such as 10-fold cross-validation:

```
automl = AutoML(  
    mode="Perform",  
    eval_metric="accuracy",  
    autofe={"cv": 10} # Set the number of folds to 10 for cross-validation  
)
```

#### d. Output

Note: *the red color represents the results of article*

	Accuracy	Best ML model	Interpretation
<b>DB1</b>	Train: 98,68% Test: 99,12% <i>(Train: –% Test: 99,12%)</i>	Ensemble	<b>- Model:</b> + <b>High model performance:</b> on both the training and test data, which implies that this model is a strong candidate for detecting malignant and benign tumors. + <b>Good generalization:</b> the fact that the test accuracy is slightly higher than the training accuracy suggests that the model generalizes well to unseen data. <b>- Data:</b> + <b>Low overfitting:</b> low accuracy gap indicates that there is limited evidence of significant overfitting. + <b>Data quality:</b> The small gap also states that the test data is consistent with the training data and does not contain significant outliers or noise.
<b>DB2</b>	Train: 80% Test: 73,21% <i>(Train: 79,09% Test: 75%)</i>	CatBoost Golden Features (17)	<b>- Model:</b> + <b>Low model performance:</b> trained accuracy and test accuracy of both databases are low, which implies that this model is a not a good candidate for detecting recurrence or no recurrence in breast cancer diagnosis. + <b>Limited generalization:</b> The significantly lower Test Accuracy compared to Trained indicates that the model's

	Train self-processed: 79,82% Test self-processed: 68,97%	CatBoost (29)	<p>performance drops when making predictions on data it hasn't seen during training.</p> <p><b>- Data:</b></p> <p>+ <b>High overfitting:</b> the substantial gap between Train Accuracy and Test Accuracy suggests that the machine learning model has overfit the training data.</p> <p>+ <b>Data quality:</b> The big gap also states that the test data is not consistent with the training data or insufficient or too imbalanced</p> <p><i>(*) the difference between 2 data sets is not enough to draw any conclusion in comparing them.</i></p>
<b>DB3</b>	Train: 87,20% Test: 78,92% <i>(Train: 87,20%)</i> <i>(Test: 79,52%)</i>	Ensemble	<p><b>- Model:</b></p> <p>+ <b>Model performance:</b> trained accuracy of the database is moderate while the test accuracy is significantly lower, which implies the problems in the dataset and the lack of evidence to conclude the performance of the model.</p> <p>+ <b>Limited Generalization:</b> The significantly lower Test Accuracy indicates that the model's performance drops when making predictions on data it hasn't seen during training.</p> <p><b>- Data:</b></p> <p>+ <b>High overfitting:</b> the substantial gap between Train Accuracy and Test Accuracy suggests that the machine learning model has overfit the training data.</p> <p>+ <b>Data quality:</b> The big gap also states that the test data is not consistent with the training data or insufficient or too imbalanced</p>
<b>DB4</b>	Train: 97,98% Test: 97,81% <i>(Train: -%)</i> <i>(Test: -%)</i>	CatBoost (27)	<p><b>- Model:</b></p> <p>+ <b>High model performance:</b> on both the training and test data, which implies that this model is a strong candidate for detecting malignant and benign cases.</p> <p>+ <b>Good generalization:</b> the fact that the test accuracy is slightly lower than the training accuracy suggests that the model generalizes well to unseen data.</p> <p><b>- Data:</b></p> <p>+ <b>Low overfitting:</b> low accuracy gap indicates that there is limited evidence of significant overfitting.</p> <p>+ <b>Data quality:</b> The small gap also states that the test data is consistent with the training data and does not contain significant outliers or noise.</p>

<b>DB5</b>	Train: 100% Test: 80% <i>(Train: 100% Test: 60%)</i>	Default CatBoost	<b>- Data:</b> The training accuracy is significantly higher than the test accuracy (e.g., 100% vs. 80%) suggests a potential of overfitting. + <b>High Overfitting:</b> The model has likely overfit the training data. As a result, the model performs exceptionally well on the training data but generalizes poorly to unseen data. + <b>Potential Data Issues:</b> The lower test accuracy (80%) indicates that the model is not generalizing well to new, unseen examples. It suggests that the model has learned specific patterns or noise in the training data that do not apply to other data points. Potential issues with the dataset could be outliers, noise, or insufficient diversity, making it challenging for the model to generalize.
	Train self-processed: 100% Test self-processed: 60%	Default CatBoost	
<b>DB6</b>	Train: 84,18% Test: 65% <i>(Train: 85,44% Test: 67,5%)</i>	Random Forest (33)	<b>- Model:</b> + Low model performance + Poor generalization <b>- Data:</b> + High Overfitting + Potential Data Issues
<b>DB7</b>	Train: 90,62% Test: 91,18% <i>(Train: 90,46% Test: 91,06%)</i>	Ensemble	<b>- Model:</b> + High model performance. + Good generalization. <b>- Data:</b> + Low overfitting. + Good data quality.
	Train self-processed: 90,59% Test self-processed: 91,18%	Ensemble	
<b>DB8</b>	Train: 85,87%	CatBoost Golden	<b>- Model:</b> + <b>Model performance:</b> lack of evidence to evaluate the performance of the model.

	Test: 62,5% <i>(Train:  85,87%  Test:  70,83%)</i>	Features (41)	+ <b>Limited Generalization:</b> the model's performance drops when making predictions on data it hasn't seen during training. - <b>Data:</b> + <b>High overfitting:</b> the substantial gap between Train Accuracy and Test Accuracy suggests that the machine learning model has overfit the training data. + <b>Data quality:</b> The big gap also states that the test data is not consistent with the training data or insufficient or too imbalanced
--	--	------------------	---

From the above interpretation, the model performs quite well with DB1, DB4 and DB7. The other datasets seem to have issues that need to be investigated more to improve the model performance and find out the best fit model.

From experience of applying MLJar, we can conclude these following evaluation about this technique:

- It's available and easily for tuning,
- It has good package of visualization the result and provides accessible results of each single ML model (no black box), which can benefit us in double-checking the results,
- It performs well on 3 databases out of 8 provided ones (DB1, DB4 and DB7) similar to other auto ML techniques,
- However, MLJar tends to have lower accuracy in comparison to other techniques.

GIT Repo: [https://github.com/danhtranhong/evaluate\\_ml\\_models](https://github.com/danhtranhong/evaluate_ml_models)