Introduction

The Heart Attack Prediction Project aims to develop a machine learning model that can predict the likelihood of a person having a heart attack based on various health factors and medical data. By analyzing a dataset containing information about patients, we can build a predictive model to identify individuals at high risk of experiencing a heart attack.

The main goal of this project is to create an accurate and reliable prediction model that can assist healthcare professionals and individuals in taking proactive measures to prevent heart attacks. The model will be trained on a relatively small dataset, demonstrating the potential of machine learning even with limited data availability.

In summary, the Heart Attack Prediction Project seeks to develop a predictive model that can identify individuals at risk of experiencing a heart attack, ultimately contributing to the prevention of heart attacks and the improvement of patient outcomes.

Dataset

The dataset used in this project contains various attributes related to patients' health and medical information. Here is a detailed description of each attribute:

- *age*: The age of the patient.
- *sex*: The gender of the patient.
- *cp*: Chest Pain types
    - Value 1: Typical angina
    - Value 2: Atypical angina
    - Value 3: Non-anginal pain
    - Value 4: Asymptomatic
- *trtbps*: Resting blood pressure (in mm Hg).
- *chol*: Cholesterol level in mg/dl, obtained via BMI sensor.
- *fbs*: Fasting blood sugar > 120 mg/dl (1 = True; 0 = False).
- *restecg*: Resting electrocardiographic results
    - Value 0: Normal
    - Value 1: Having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)
    - Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria
- *thalachh*: Maximum heart rate achieved.
- *exng*: Exercise-induced angina (1 = Yes, 0 = No).
- *oldpeak*: ST depression induced by exercise relative to rest.
- *slp*: Slope of the peak exercise ST segment.
- *caa*: Number of major vessels (0-3).
- *thall*: Thalium Stress Test result.
- *output*: The predicted chance of heart attack
    - 0 = Less chance of heart attack
    - 1 = More chance of heart attack

Method

Data preprocessing steps were undertaken, including the removal of a column without correlation and standardization of the data. For both **Logistic Regression** and **Random Forest** models, hyperparameter optimization was performed. The **dataset was split into 5 folds** for cross-validation purposes. A **test size of 0.2** was used to separate a portion of the data for evaluating the final models.

In the model evaluation phase, **recall** and **F1** score were employed as the primary metrics. The choice of recall was crucial for this heart attack prediction model, as it measures the model's ability to correctly identify positive cases (individuals at risk of heart attack). A high recall ensures that fewer positive cases are missed, which is particularly important in healthcare applications where false negatives can have severe consequences.

After training and evaluating the models, the results of both Logistic Regression and Random Forest were analyzed and compared.

Development

Plan

This project aims to find the best machine learning model for predicting heart attacks. The dataset has 14 features, and the "output" column is used as the target variable. The project is done using Python and the scikit-learn library in Jupyter Notebook. A standard computer should be enough for this project, considering its size and the dataset used.

Analysis

To analyze the features in the dataset, I used exploratory data analysis (EDA) techniques. I made bar plots for categorical features and distribution plots and box plots for numerical features. These plots helped me understand how each feature relates to the target variable. I also created a heatmap to look at the correlations between features. After analyzing, I saw that the "fbs" feature had no effect on the target variable, so I removed it from the dataset.

Design

I tried two different models in this project: Logistic Regression and Random Forest. To tune the hyperparameters for each model, I used RandomizedSearchCV. For Logistic Regression, I chose the 'C', 'penalty', and 'solver' hyperparameters. For Random Forest, I selected the 'n_estimators', 'max_depth', 'min_samples_split', and 'min_samples_leaf' hyperparameters. To evaluate the models' performance, I used 5-fold cross-validation.

Implementation

In this section, I will showcase some code blocks and provide brief explanations. Additionally, all detailed explanations can be found in the b2200765031-Supervised.ipynb file.

In my implementation, I started by reading the "medical_heart.csv" file into a pandas DataFrame called 'df'. I used 'head()' to take a quick look at the data and 'describe()' to get summary statistics for the numerical columns.

This helped me understand the distribution and range of values in each feature.

I also checked for missing values using 'isnull().sum()' to ensure data quality before moving on with my analysis.

To visualize the correlations between features, I created a heatmap using seaborn's 'heatmap()' function. I customized the heatmap by setting the figure size, adding annotations, and adjusting the line widths to make it more readable. These initial steps allowed me to explore the dataset, handle missing values, and understand the relationships between features, setting the ,stage for further analysis and modeling.

[3]:
```
df = pd.read_csv("medical_heart.csv")
df.head()
```

[3]:

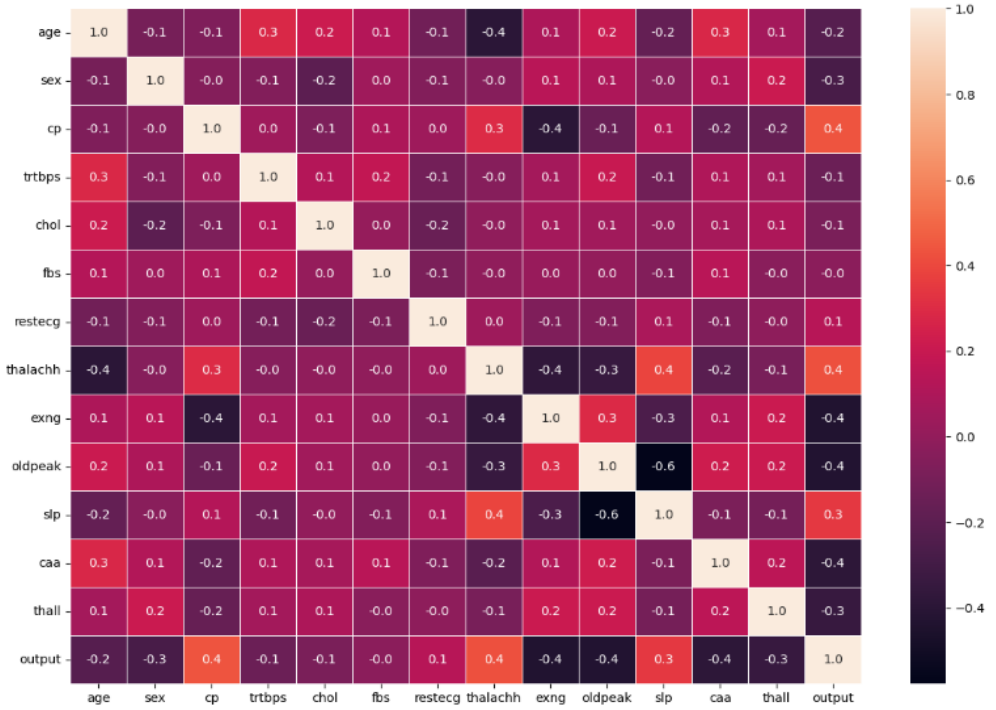
	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

[5]:
```
df.describe()
```

[5]:

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000

[17]:
```
plt.figure(figsize = (14,10))
sns.heatmap(df.corr(), annot = True, fmt = ".1f", linewidths = .7)
plt.show()
```



```python
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import recall_score, confusion_matrix, f1_score

def model_selection_and_evaluation(X, y, model_choice, test_size):
    # Train test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=42)

    # Model selection and hyperparameter optimization
    if model_choice == "Logistic Regression":
        model = LogisticRegression()
        param_dist = {
            'C': [0.1, 1, 10],
            'penalty': ['l1', 'l2'],
            'solver': ['liblinear', 'saga']
        }
    elif model_choice == "Random Forest":
        model = RandomForestClassifier()
        param_dist = {
            'n_estimators': [100, 200, 300],
            'max_depth': [None, 5, 10],
            'min_samples_split': [2, 5, 10],
            'min_samples_leaf': [1, 2, 4]
        }
    else:
        raise ValueError("Unvalid Model Selection !")

    # Hyperparameter optimization
    random_search = RandomizedSearchCV(estimator=model, param_distributions=param_dist, n_iter=10, cv=5, random_state=42)
    random_search.fit(X_train, y_train)
    best_model = random_search.best_estimator_

    # Modelling and predicting
    best_model.fit(X_train, y_train)

    y_pred = best_model.predict(X_test)

    recall = recall_score(y_test, y_pred)

    f1 = f1_score(y_test, y_pred)

    cm = confusion_matrix(y_test, y_pred)

    # Results
    print(f"Choosen Model: {model_choice}")
    print(f"Best Hyperparameters: {random_search.best_params_}")
    print(f"Test size : {test_size}")
    print(f"Test Set Recall Score: {recall}")
    print(f"Test Set F1 Score: {f1}")

    plt.figure(figsize=(8, 6))
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    plt.title('Confusion Matrix')
    plt.colorbar()
    tick_marks = np.arange(len(np.unique(y_test)))
    plt.xticks(tick_marks, np.unique(y_test), rotation=45)
    plt.yticks(tick_marks, np.unique(y_test))
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')

    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            plt.text(j, i, format(cm[i, j], 'd'),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")

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

I created a function called 'model_selection_and_evaluation' to make the process of selecting and evaluating models easier. This function splits the dataset into training and test sets, trains the chosen model with hyperparameter optimization, and evaluates the model's performance on the test set.

I gave the function parameters like the independent variables (X), the target variable (y), the type of model selected, and the size of the test set. Depending on the model chosen, I set the appropriate hyperparameters for either Logistic Regression or Random Forest. Then, I used RandomizedSearchCV to perform hyperparameter optimization, which helped me find the best hyperparameters.

After training the model with the best hyperparameters, I made predictions on the test set and evaluated the performance. I used recall and F1 score metrics because it was important to minimize false negatives and assess the overall performance of the model in a balanced way for this problem.

Finally, I printed the results by showing the selected model, the best hyperparameters, the test set recall and F1 scores, and visualizing the confusion matrix. This function allowed me to quickly and effectively evaluate and compare different models.

Programmer Catalog

The time I spent on different stages of the project is as follows:

- Analysis: 2 hours
- Design: 1 hour
- Implementation: 4 hours
- Testing: 1 hour
- Reporting: 3 hours

The code I wrote is highly reusable for other purposes or different datasets. I specifically designed the 'model_selection_and_evaluation' function to be easily adaptable for various datasets and models. The function takes the dataset, target variable, selected model, and test set size as parameters. By changing these parameters, the function can be applied to different problems.

Moreover, data preprocessing steps (checking for missing values, transforming categorical variables, etc.) and visualization techniques (heatmap, confusion matrix, etc.) can also be used in other projects.

Programmers can use the following unit/function explanations and usage examples to understand and adapt different parts of the code:

```
def model_selection_and_evaluation(X, y, model_choice, test_size):
    """
    Performs model selection and evaluation.
    Parameters:
    X (pd.DataFrame): Independent variables
    y (pd.Series): Target variable
    model_choice (str): Selected model type ('Logistic Regression' or 'Random Forest')
    test_size (float): Size of the test set (between 0 and 1)
    Output:
    Prints the performance metrics and confusion matrix of the selected model.
    """
    # Example usage
    model_selection_and_evaluation(X, y, 'Logistic Regression', 0.2)
```

User Catalog

The code I wrote cannot be directly used by a regular user without programming knowledge. However, some parts of the code (e.g., data visualization techniques) can be presented to the user through an interface like Jupyter Notebook.

Users can load different datasets and explore the data by running specific cells. For example, they can run the df.isnull().sum() cell to check for missing values, the df.describe() cell to see the distribution of features, or the heatmap cell to visualize the correlation matrix.

However, the process of training and evaluating the model is not suitable to be done directly by the user. These steps require programming knowledge and may not be user-friendly.

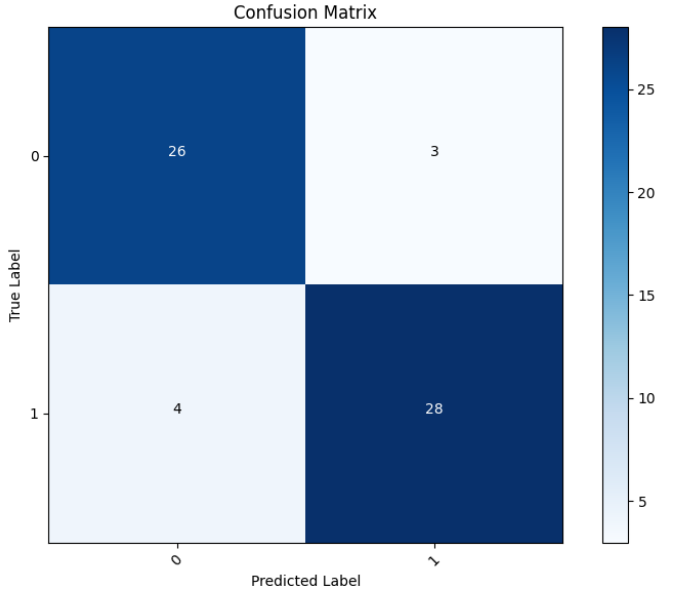
The limitations of the project are:

- The project is suitable only for binary classification problems.
- The dataset must be in a specific format (e.g., CSV) and contain the required columns.
- Users are not expected to perform data preprocessing and model training steps directly.

Results

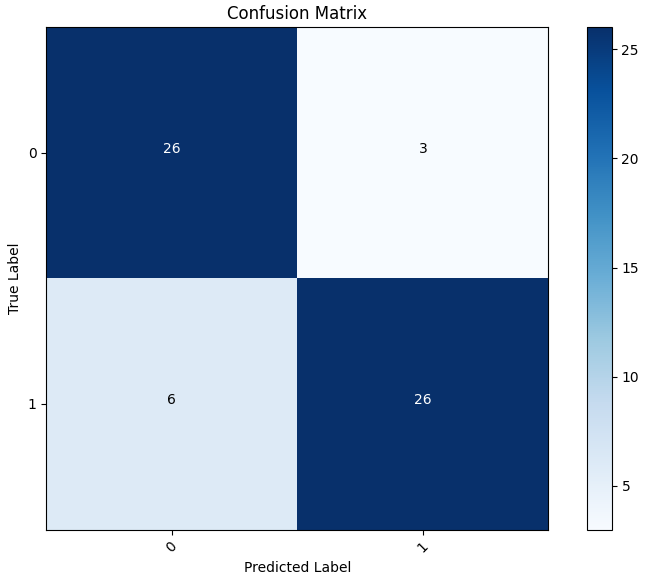
```
model_selection_and_evaluation(X,y,"Logistic Regression", 0.2)

Choosen Model: Logistic Regression
Best Hyperparameters: {'solver': 'saga', 'penalty': 'l1', 'C': 10}
Test size : 0.2
Test Set Recall Score: 0.875
Test Set F1 Score: 0.8888888888888888
```




```
model_selection_and_evaluation(X,y,"Random Forest", 0.2)

Choosen Model: Random Forest
Best Hyperparameters: {'n_estimators': 200, 'min_samples_split': 5, 'min_samples_leaf': 1, 'max_depth':
5}
Test size : 0.2
Test Set Recall Score: 0.8125
Test Set F1 Score: 0.8524590163934426
```



For this project, two different classification models were tried: Logistic Regression and Random Forest. Our goal was to find the best performing model for the given dataset.

First, a Logistic Regression model was trained with specific hyperparameters. The 'solver' parameter was set to 'saga', which is suitable for L1 regularization. The 'penalty' parameter was set to 'l1', indicating the use of L1 regularization, which can help with feature selection. The 'C' parameter, which controls the inverse of the regularization strength, was set to 10. Using a test set size of 0.2, the model achieved a recall score of 0.875 and an F1 score of 0.889.

Next, a Random Forest model was trained with its own set of hyperparameters. The 'n_estimators' parameter, which determines the number of trees in the forest, was set to 200. The 'min_samples_split' parameter, which controls the minimum number of samples required to split a node, was set to 5. The 'min_samples_leaf' parameter, which determines the minimum number of samples required in a leaf node, was set to 1. The 'max_depth' parameter, which limits the maximum depth of the trees, was set to 5, restricting the depth of the trees. With the same test set size of 0.2, the model achieved a recall score of 0.813 and an F1 score of 0.852.

When the two models were compared, the Logistic Regression model performed slightly better in terms of both recall and F1 scores. However, the difference in performance was not significant, and both models demonstrated satisfactory results.

Overall, both models showed promising results, but I would still prefer the Logistic Regression model. Although the difference is small, Logistic Regression achieved better scores. The slight advantage in recall and F1 scores indicates that Logistic Regression performed a bit better for this dataset. Of course, this is just my opinion. Since the results are very close, some might prefer Random Forest. But for me, Logistic Regression gave slightly better results, so I would choose that one.

To put it simply, without getting too technical: there is no big difference between the two models, but Logistic Regression performed slightly better, so I would prefer it. However, since the results are very close, other factors should also be considered. But overall, Logistic Regression seems to be a bit more suitable for this dataset.

Models and Evaluation Metrics

In the heart attack prediction model, two different machine learning algorithms were used: Logistic Regression and Random Forest. These algorithms are commonly preferred in classification problems and offer different approaches. The aim of the study is to find the most suitable model by comparing the performance of these two algorithms.

Logistic Regression is a statistical machine learning algorithm that is frequently used in binary classification problems. This algorithm predicts class probabilities based on given inputs by modeling the relationship between independent variables and the dependent variable. In the heart attack prediction model, the patient's risk factors and clinical data were taken as inputs, and the probability of a heart attack was obtained as the output. One of the advantages of Logistic Regression is that the results are easily interpretable. The coefficients of the model show the effect of each independent variable on the probability of a heart attack. Additionally, Logistic Regression can help simplify the model and prevent overfitting by performing feature selection.

Random Forest, on the other hand, is an ensemble learning method that combines multiple decision trees. Each decision tree is trained on a random subset of the data, and the final prediction is obtained by combining the predictions of the trees. Random Forest can capture complex relationships between variables and handle high-dimensional datasets. One of the advantages of this algorithm is that it reduces the risk of overfitting. By using multiple decision trees, the generalization ability of the model is increased. Moreover, by calculating feature importances, it can be determined which risk factors are more important for heart attack prediction.

In the development of the heart attack prediction model, the different approaches and advantages of Logistic Regression and Random Forest algorithms were considered. The performance of the model was thoroughly evaluated using these two algorithms.

Recall measures how well the model identifies the actual positive cases. In the heart attack prediction model, recall shows how many people who actually had a heart attack were correctly predicted by the model. It aims to minimize false negatives.

The main reason for using recall in this model is that false negatives can have serious consequences. If the model incorrectly classifies a person who actually had a heart attack as healthy (false negative), it can prevent the patient from getting necessary treatment and potentially put their life at risk. Therefore, it's crucial for the model to detect heart attacks with high accuracy and minimize false negatives.

Recall helps us evaluate the model's performance in this regard. A high recall value indicates that the model successfully captures the actual heart attack cases, helping patients receive early diagnosis and treatment.

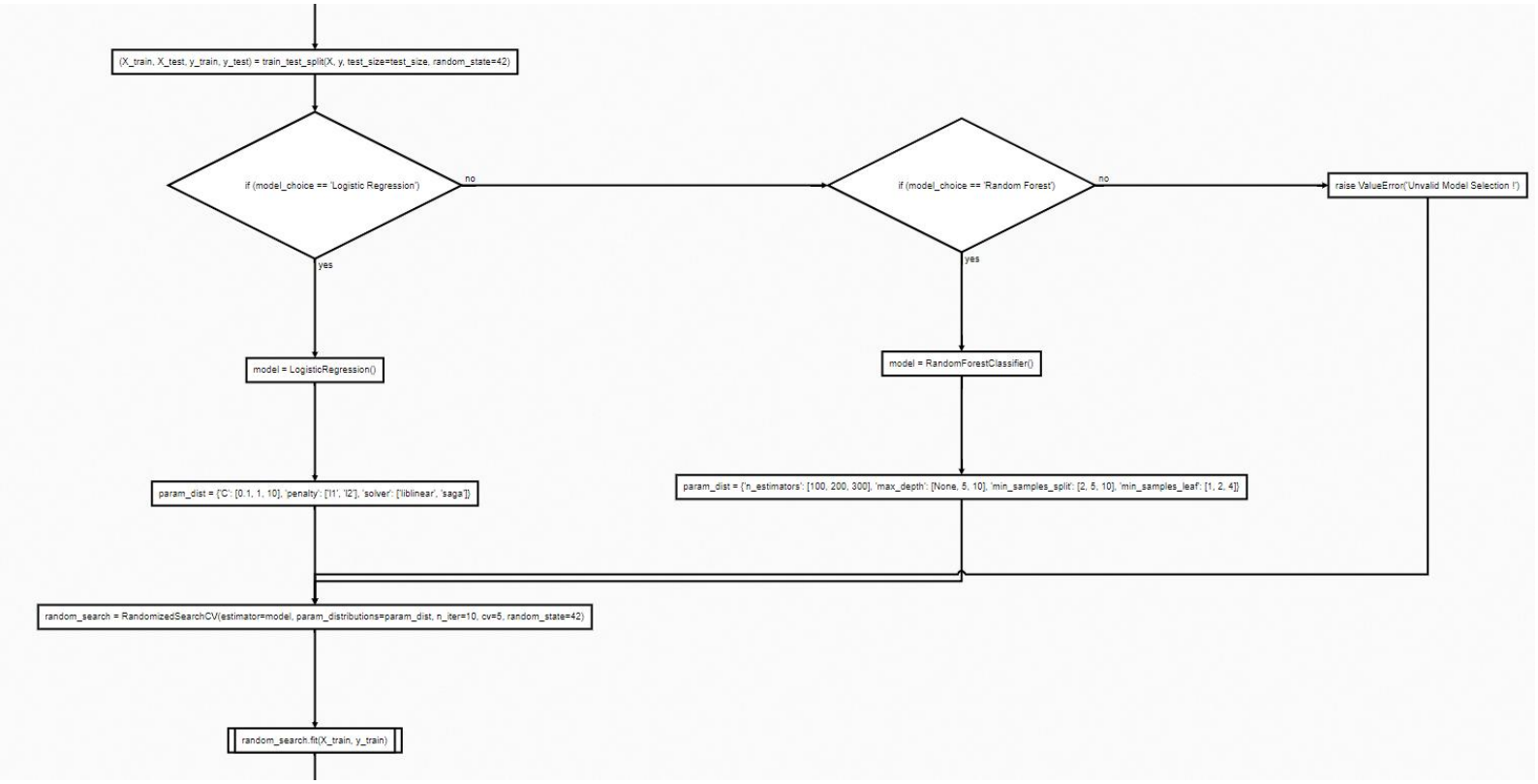
The F1 score is the harmonic mean of precision and recall. Precision measures how many of the positively classified cases are actually positive, while recall measures how many of the actual positives are correctly predicted. The F1 score combines these two metrics into a single value and provides a balanced evaluation of the model's overall performance.

Using the F1 score in the heart attack prediction model ensures that the model balances both false positives (classifying healthy individuals as having a heart attack) and false negatives (classifying individuals who actually had a heart attack as healthy). A high F1 score indicates that the model achieves a good balance between precision and recall.

The F1 score summarizes the model's overall performance in a single value, making it useful for comparing different models and selecting the most suitable one. It also provides a robust evaluation when the class distribution in the dataset is imbalanced.

In summary, using recall and F1 score metrics in the heart attack prediction model allows for a comprehensive evaluation of the model's performance. Recall focuses on minimizing false negatives, ensuring early diagnosis and treatment for patients, while the F1 score provides a balanced assessment of the model's overall performance. Using these metrics plays a critical role in evaluating the effectiveness and reliability of the heart attack prediction model.

Flowchart



References

- https://www.kaggle.com/datasets/rashikrahmanpritom/heart-attack-analysis-prediction-dataset
- https://en.wikipedia.org/wiki/Precision_and_recall
- https://en.wikipedia.org/wiki/Logistic_regression
- https://en.wikipedia.org/wiki/Random_forest
- https://en.wikipedia.org/wiki/Hyperparameter_optimization