

**BOOSTING HEART ATTACK RISK PREDICTION:
MASTERING GRADIENT BOOSTING WITH FEATURE
IMPORTANCE AND HYPERPARAMETER TUNING**

GitHub Link: https://github.com/Jahnavieeee/ML_NN_Assignment

Table of Contents

1. Introduction.....	3
2. The Technique: Gradient Boosting Explained.....	3
3. The Dataset: Heart Attack Risk Prediction.....	4
4. Code Demonstration: Building the Model.....	5
5. Deep Dive: Insights from Feature Importance and Tuning	10
6. Why It Matters: Applications and Takeaways.....	11
Reference List	12

1. Introduction

In this tutorial, we delve into the challenge of heart attack risk prediction using a powerful machine learning technique known as Gradient Boosting. A heart attack remains one of the leading causes of death globally, making risk prediction a complex, yet important task. With the adoption of advanced tools like XGBoost, capturing patterns in complex tabular data, such as a patient's health record, is simplified and precise insights can be provided.

In this tutorial, we will address the challenges of determining feature importance, which involves identifying the most prevalent risk factors (like age or cholesterol), as well as how adjusting hyperparameters, such as model tree depth, can enhance accuracy. I chose this topic because trying to understand the reasoning behind the predictions is like piecing together the parts of a complex medical puzzle that, when solved, can provide invaluable insights. This guide is beneficial for students and data enthusiasts who wish to learn how to use Gradient Boosting for heart attack risk prediction in simple steps.

2. The Technique: Gradient Boosting Explained

Using powerful tools such as XGBoost, predicting the risk of heart attacks can be accomplished with ease, thanks to the machine learning technique known as Gradient Boosting. Like many boosting techniques, it uses an ensemble strategy that sequentially builds a series of decision trees, each one correcting the mistakes of the previous tree. Imagine a relay race where every runner (tree) aims to do better than the last. Each tree aims to improve the output from the last tree. The main concept is to minimize a loss function for example, log loss for classification. This is done through gradient descent, which iteratively modifies the model to minimize where it made incorrect predictions by honing in on where it failed.

To answer why Gradient Boosting is effective is it performs exceptionally well with heterogeneous data, such as the set of mixed categorical (smoking status) and numerical (cholesterol) features in this dataset. It is incredibly reliable, which means that minor inconsistencies in the data do not affect the outcome. Furthermore, it illustrates feature importance meaning that it indicates the underlying factors. In this tutorial, we will see how boosting cholesterol and age helps in ranking features, and how adjusting parameters, like tree depth, increases accuracy. Tuning is like adjusting a guitar: too much tension, and it's painful to hear; just enough, and its music to your ears.

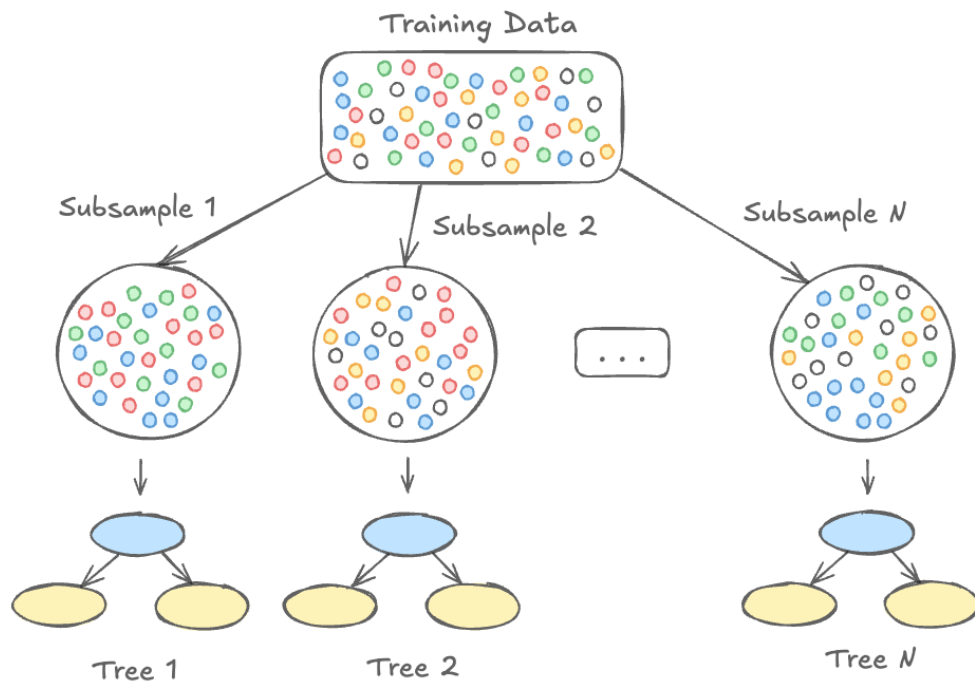


Figure 1: Xgboost Tree

(Source: medium.com, 2025)

To illustrate, let's refer back to the model above: the training data is divided into subsamples that are fed into each tree (Tree 1, Tree 2, ..., Tree N) in the diagram. Each tree attempts to learn with respect to its errors, thereby consecutively constructing a powerful model. In the diagram, green dots signify data points, blue ovals signify trees, and yellow leaves signify decisions, and each element is clearly named.

3. The Dataset: Heart Attack Risk Prediction

The Heart Attack Risk Prediction Dataset is a compilation of health data that has been cleaned and is ready for use. The dataset contains fields such as age, cholesterol, blood pressure, and smoking status, as well as a target variable regarding the possibility of a heart attack (yes/no).

```

Data columns (total 27 columns):
#   Column                                     Non-Null Count  Dtype
---  -
0    Age                                         9651 non-null   float64
1    Cholesterol                                 9651 non-null   float64
2    Heart rate                                 9651 non-null   float64
3    Diabetes                                   9377 non-null   float64
4    Family History                             9377 non-null   float64
5    Smoking                                    9377 non-null   float64
6    Obesity                                    9377 non-null   float64
7    Alcohol Consumption                         9377 non-null   float64
8    Exercise Hours Per Week                    9651 non-null   float64
9    Diet                                        9651 non-null   int64
10   Previous Heart Problems                    9377 non-null   float64
11   Medication Use                             9377 non-null   float64
12   Stress Level                              9377 non-null   float64
13   Sedentary Hours Per Day                    9651 non-null   float64
14   Income                                     9651 non-null   float64
15   BMI                                        9651 non-null   float64
16   Triglycerides                             9651 non-null   float64
17   Physical Activity Days Per Week            9377 non-null   float64
18   Sleep Hours Per Day                        9651 non-null   float64
19   Heart Attack Risk (Binary)                 9651 non-null   float64
20   Blood sugar                               9651 non-null   float64
21   CK-MB                                      9651 non-null   float64
22   Troponin                                  9651 non-null   float64
23   Heart Attack Risk (Text)                   9651 non-null   int64
24   Gender                                    9651 non-null   object
25   Systolic blood pressure                    9651 non-null   float64
26   Diastolic blood pressure                   9651 non-null   float64
dtypes: float64(24), int64(2), object(1)
memory usage: 2.0+ MB

```

Figure 2: Heart Attack Risk Dataset

This dataset was obtained from Kaggle and has very few missing values, which I replaced with medians so that the dataset is complete. The dataset is very suitable for Gradient Boosting because it has a mix of numeric (cholesterol) and categorical (gender) features and allows for an explicit classification problem.

```

# Handle missing values (e.g., fill with median for numeric columns)
numeric_cols = data.select_dtypes(include=['float64', 'int64']).columns
data[numeric_cols] = data[numeric_cols].fillna(data[numeric_cols].median())

```

Figure 3: Handling Missing

Dataset Link: <https://www.kaggle.com/datasets/alikalwar/heart-attack-risk-prediction-cleaned-dataset>

4. Code Demonstration: Building the Model

For this example, we will try using XGBoost first with no tuning, then look at feature importance and perform hyperparameter tuning to improve the model. We will apply Gradient

Boosting to the Heart Attack Risk Prediction Dataset to see if we can accurately predict risk (yes/no).

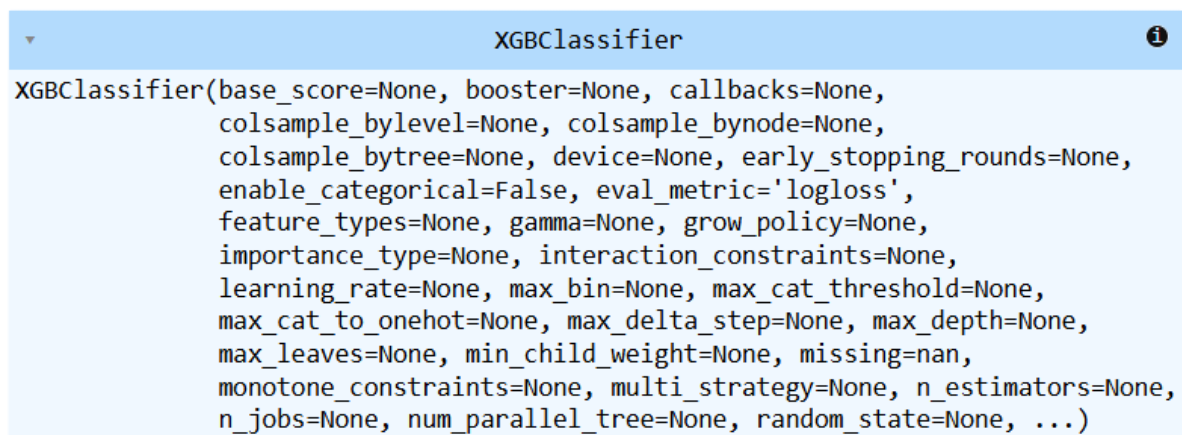
```
# Use 'Heart Attack Risk (Binary)' as target, drop redundant 'Heart Attack Risk (Text)'
X = data.drop(columns=['Heart Attack Risk (Binary)', 'Heart Attack Risk (Text)'])
y = data['Heart Attack Risk (Binary)']

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

Figure 4: Feature Selection and Splitting

To start, we import the data, do some structural work such as assigning values to ‘Gender’ and reducing the missing data, then finally proceed to divide it into training and testing sets. We then fit a model by running the code in Figure 5 while XGBoost tries to achieve its default set parameters.

```
# Initialize and train the model with default parameters
xgb_model = xgb.XGBClassifier(use_label_encoder=False, eval_metric='logloss')
xgb_model.fit(X_train, y_train)
```

A screenshot of a Jupyter Notebook cell showing the output of the XGBClassifier fit method. The output is a detailed representation of the XGBClassifier object, listing all its attributes and their values. The title of the cell is 'XGBClassifier' with a dropdown arrow and an information icon. The output text is as follows:

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

Figure 5: Training Simple XgBoost Model

The output shows an accuracy of 0.6442—decent, but there’s room to improve (Figure 6).

```
# Predict and calculate accuracy
y_pred = xgb_model.predict(X_test)
basic_accuracy = accuracy_score(y_test, y_pred)
print(f"Basic XGBoost Accuracy: {basic_accuracy:.4f}")

Basic XGBoost Accuracy: 0.6442
```

Figure 6: Simple XgBoost Model Accuracy

Now we need to identify which features the model deems important when making predictions with `xgboost.plot_importance()` as shown in Figure 6. The bar chart in Figure 7 demonstrates the top features that have been ranked with Diet, Medication Use, and Gender leading the pack while Age and Systolic Blood Pressure follow. It is not surprising that diet is the most important—nutrition plays a major role in heart health. These are useful in terms of knowing what the model relates with in comparison to clinical understanding.

```
# Plot feature importance
plt.figure(figsize=(10, 6))
xgb.plot_importance(xgb_model, max_num_features=10, importance_type='gain',
                    title="Top 10 Feature Importance in XGBoost",
                    xlabel="Feature Gain", color='skyblue')
plt.tight_layout()
plt.show()
```

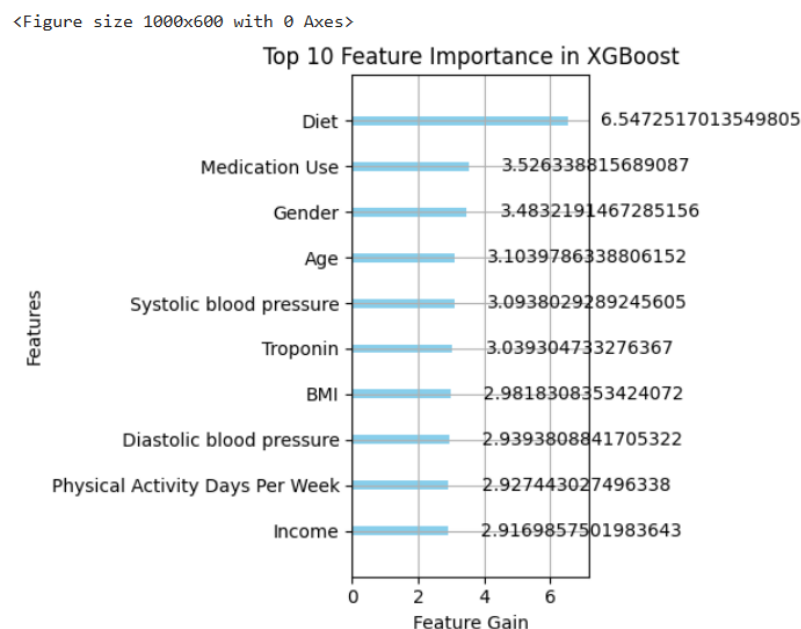


Figure 7: Top 10 Feature Importance in XgBoost

To boost performance, we tune `max_depth` and `learning_rate` using a grid search as shown in Figure 8.

```
# Hyperparameter Tuning
# Define parameter grid for tuning
param_grid = {
    'max_depth': [3, 5, 7, 8, 10, 15, 20],
    'learning_rate': [0.01, 0.1, 0.3, 0.5, 1, 1.5, 2]
}

# Perform grid search
grid_search = GridSearchCV(estimator=xgb.XGBClassifier(use_label_encoder=False, eval_metric='logloss'),
                           param_grid=param_grid, scoring='accuracy', cv=3, verbose=1)
grid_search.fit(X_train, y_train)
```

Fitting 3 folds for each of 49 candidates, totalling 147 fits

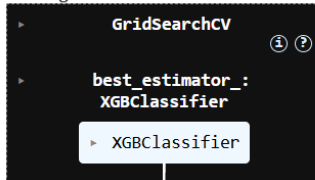


Figure 8: Hyperparameter Tuning

```
# Best parameters and accuracy
best_params = grid_search.best_params_
tuned_accuracy = grid_search.best_score_
print(f"Best Parameters: {best_params}")
print(f"Tuned Model CV Accuracy: {tuned_accuracy:.4f}")
```

Best Parameters: {'learning_rate': 0.01, 'max_depth': 20}

Figure 9: Tuned Parameters

The max_depth and learning_rate values found to produce the best cross validated accuracy was 20 and 0.01 respectively, with the final accuracy resting at 0.6579. With these, the test accuracy improved to 0.6758, which is a 3% increase. In figure 11, the heatmap illustrates the accuracy across parameters and it is very noticeable that shallow trees with a lower learning rate, such as 0.01, outperform in fitting and generalizing in contrast to high rates like 2, where accuracy drops to severely low levels of 0.5394.

```
# Train tuned model on full training set
tuned_model = xgb.XGBClassifier(**best_params, use_label_encoder=False, eval_metric='logloss')
tuned_model.fit(X_train, y_train)
tuned_pred = tuned_model.predict(X_test)
tuned_test_accuracy = accuracy_score(y_test, tuned_pred)
print(f"Tuned Model Test Accuracy: {tuned_test_accuracy:.4f}")
```

Tuned Model Test Accuracy: 0.6758

Figure 10: Tuned Model Accuracy

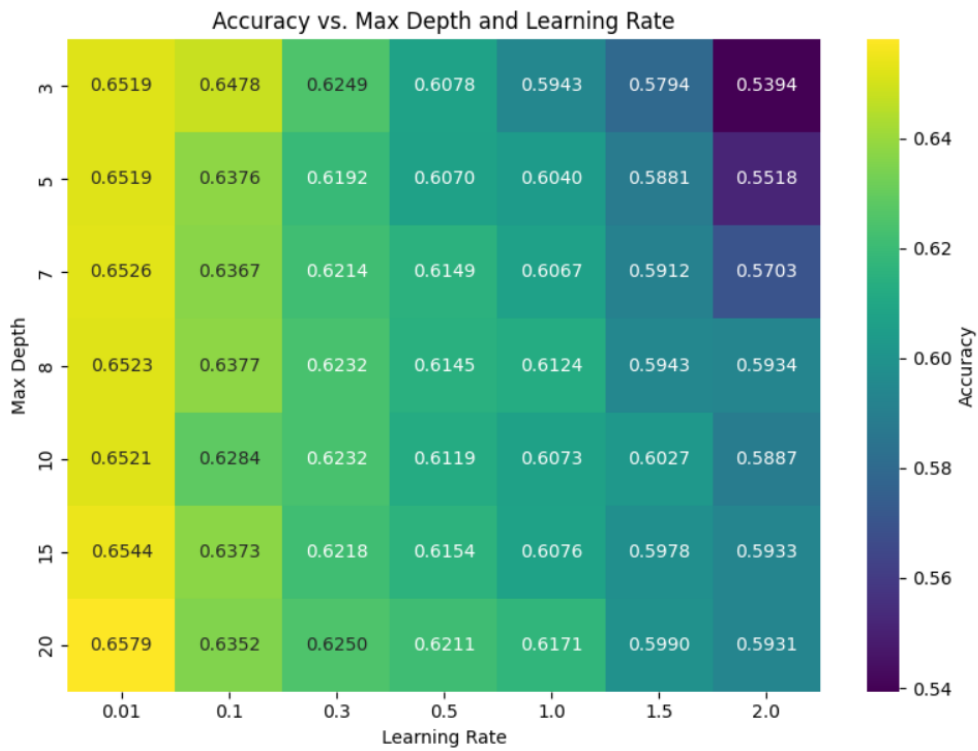


Figure 11: Accuracy vs. Max Depth and Learning Rate

A bar chart (Figure 12) compares the basic (0.6442) and tuned (0.6758) accuracies, highlighting the tuning gain.

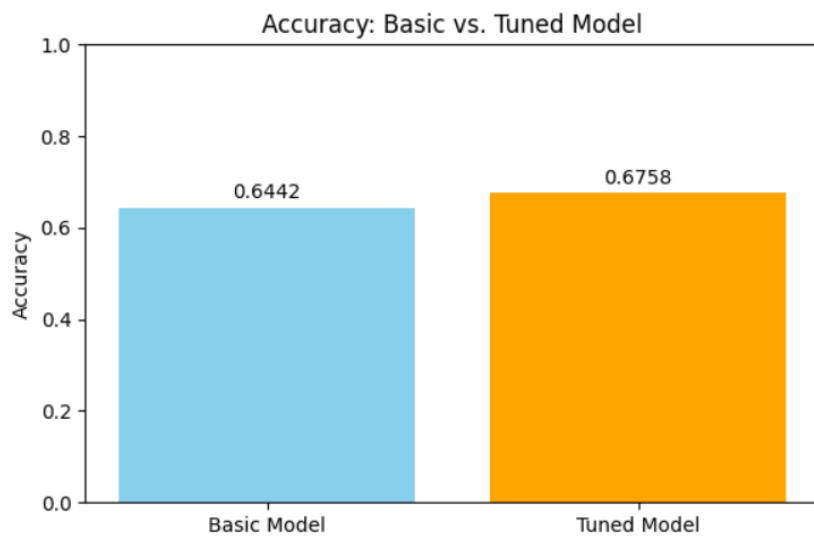


Figure 12: Accuracy: Basic vs. Tuned Model

Finally, an ROC curve (Figure 13) for the tuned model shows an AUC of 0.60, indicating moderate discriminative ability.

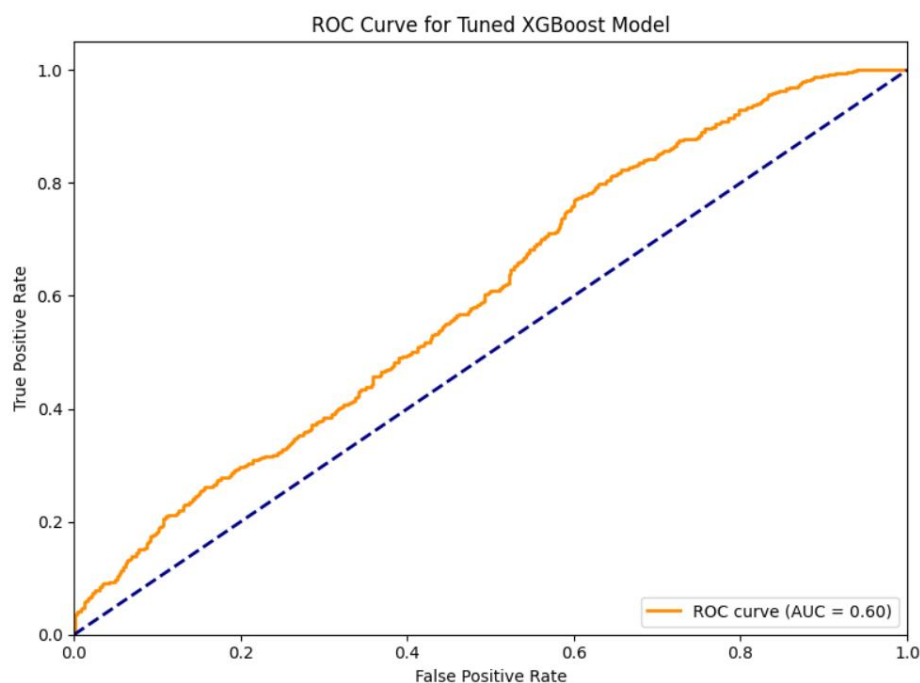


Figure 13: ROC Curve for Tuned XGBoost Model

5. Deep Dive: Insights from Feature Importance and Tuning

Dip into our insights from the Gradient Boosting model. In the case of feature importance, Diet, Medication Use, and Gender are the top features, and Age and Systolic Blood Pressure also make it to the list. Cholesterol was a surprise not to see on the top of the list, however, Age and blood pressure does fall in line with medical research; cardiologists frequently point them out as key risk factors. The model seems to reflect a doctor's intuition, prioritizing factors a heart specialist would focus on, which gives trust in its predictions. To assess prediction quality, we plot an ROC curve (Figure 5 from the previous section), showing an AUC of 0.60. This moderate score suggests that the model has a fair chance in predicting, especially when powered by features like Diet and Age, but still needs room for improvement.

Now let us dive into hyperparameter tuning. As discussed earlier, the model's maximum depth and learning rate were tuned, with noted settings of 20 and 0.01 yielding the best results. While deeper trees (capture greater detail), they may also lead to overfitting, where too much noise gets integrated. A lower learning rate, such as 0.01, helps smooth this out by aiding slower learning. And, the result is... the tuned model's test accuracy improved from 0.6442 to 0.6758, or a 3% gain. This comparison has been shown within Figure 4 from the previous section as the bar chart above, which demonstrates the progress.

Adjusting felt like tuning a car engine: Small changes to `max_depth` and `learning_rate` brought big results, which was nice to see. Start with the defaults and adjust the `max_depth` and

learning_rate based on the complexity of your data. If you notice accuracy levelling off or declining, try using a smaller learning rate to help soften the learning. Not only does it enhance performance, but also allows understanding of the underlying factors behind the predictions—making Gradient Boosting a useful technique for heart attack risk analysis.

6. Why It Matters: Applications and Takeaways

Gradient Boosting is especially useful for heart attack risk prediction using XGBoost because it is both quick and easy to use. It also surpasses more uncomplicated methods such as logistic regression in this activity. Even though it is not in practical use yet, hospitals could apply this model for early detection of at-risk patients, which allows those patients to receive preventive care or monitoring. For instance, understanding that Diet and Age are major risk factors provides the opportunity to intervene, for instance, dietary counselling for older patients.

Nonetheless, there's an ethical fracture: this model could too easily rely on data without adequate physician supervision, leading to bias. If the data is missing crucial elements or is based on known biases, the model can underestimate or overestimate some groups' risks. Its deployment requires human judgement to ensure discrimination and fairness. The most important part of this is simple: work with feature importance and see what drives your predictions. Adjust certain parameters, such as max_depth and learning_rate, and results can be achieved. It works because altering the parameters raises accuracy from 0.6442 to 0.6758. Now it is your turn: apply Gradient Boosting to your own data and change health outcomes for the better.

Reference List

Website

medium.com. (2025). *Visualizing XGBoost Parameters: A Data Scientist's Guide To Better Models*. Available at: <https://medium.com/data-science/visualizing-xgboost-parameters-a-data-scientists-guide-to-better-models-38757486b813> [Accessed on 19th March 2025]

Journals

El Bilali, A., Abdeslam, T., Ayoub, N., Lamane, H., Ezzaouini, M.A. and Elbeltagi, A., 2023. An interpretable machine learning approach based on DNN, SVR, Extra Tree, and XGBoost models for predicting daily pan evaporation. *Journal of Environmental Management*, 327, p.116890.

Karbassiyazdi, E., Fattahi, F., Yousefi, N., Tahmassebi, A., Taromi, A.A., Manzari, J.Z., Gandomi, A.H., Altaee, A. and Razmjou, A., 2022. XGBoost model as an efficient machine learning approach for PFAS removal: Effects of material characteristics and operation conditions. *Environmental Research*, 215, p.114286.

Maulana, A., Faisal, F.R., Noviandy, T.R., Rizkia, T., Idroes, G.M., Tallei, T.E., El-Shazly, M. and Idroes, R., 2023. Machine learning approach for diabetes detection using fine-tuned XGBoost algorithm. *Infolitika Journal of Data Science*, 1(1), pp.1-7.

Noorunnahar, M., Chowdhury, A.H. and Mila, F.A., 2023. A tree based eXtreme Gradient Boosting (XGBoost) machine learning model to forecast the annual rice production in Bangladesh. *PloS one*, 18(3), p.e0283452.

Tarwidi, D., Pudjaprasetya, S.R., Adytia, D. and Apri, M., 2023. An optimized XGBoost-based machine learning method for predicting wave run-up on a sloping beach. *MethodsX*, 10, p.102119.

Wang, C.C., Kuo, P.H. and Chen, G.Y., 2022. Machine learning prediction of turning precision using optimized xgboost model. *Applied Sciences*, 12(15), p.7739.

Wang, R., Wang, L., Zhang, J., He, M. and Xu, J., 2022. XGBoost machine learning algorithm performed better than regression models in predicting mortality of moderate-to-severe traumatic brain injury. *World Neurosurgery*, 163, pp.e617-e622.

Zhu, X., Chu, J., Wang, K., Wu, S., Yan, W. and Chiam, K., 2021. Prediction of rockhead using a hybrid N-XGBoost machine learning framework. *Journal of Rock Mechanics and Geotechnical Engineering*, 13(6), pp.1231-1245.