# Machine Learning Final Project

# Final Project

M

## Members
Members of the team and contributions

P

## Progress
Why XGBoost

M

## Method
XGBoost Implementation

R

## Results
Confusion Matrix, recall, precision, F1-score

# Members


Phạm Ngọc Hải Dương
MSV: 23021513

Phạm Mai Anh
MSV: 23021469

Bùi Thế Kiệt
MSV: 23021469

# Progress

- From the beginning, our team decided to work on an algorithm using an ensemble method.
- Initially, we used libraries to test which algorithm would perform best on the given dataset.
- Using Lazy Classifier, we obtained the following results for various libraries...
- Based on these results, we decided to select **Random Forest**.
- We implemented Random Forest, but faced **challenges** such as long training time and suboptimal performance.
- As a result, we switched to a new algorithm, **XGBoost**, and decided to proceed with XGBoost for our implementation.

# Approaches

## Lazy Classifier Table

| | Model | Accuracy | Precision | Recall | F1_Score |
|---|---|---|---|---|---|
| 0 | XGBoost Classifier | 0.933333 | 0.932706 | 0.933333 | 0.932937 |
| 1 | XGBoost Regressor | 0.927583 | 0.927661 | 0.927583 | 0.927622 |
| 2 | LightGBM Classifier | 0.926333 | 0.925310 | 0.926333 | 0.925597 |
| 3 | RandomForestClassifier | 0.922833 | 0.924048 | 0.922833 | 0.923336 |
| 4 | LightGBM Regressor | 0.922417 | 0.923113 | 0.922417 | 0.922726 |
| 5 | RandomForestRegressor | 0.920417 | 0.922033 | 0.920417 | 0.921062 |
| 6 | DecisionTreeRegressor | 0.889667 | 0.896031 | 0.889667 | 0.891850 |
| 7 | DecisionTreeClassifier | 0.888667 | 0.894941 | 0.888667 | 0.890838 |
| 8 | LogisticRegression | 0.866583 | 0.900218 | 0.866583 | 0.873534 |
| 9 | SVM | 0.863667 | 0.899122 | 0.863667 | 0.870923 |
| 10 | KNN | 0.859583 | 0.883523 | 0.859583 | 0.865676 |
| 11 | KNN Regressor | 0.859583 | 0.883523 | 0.859583 | 0.865676 |
| 12 | Linear Regression | 0.826667 | 0.893391 | 0.826667 | 0.838157 |
| 13 | Naive Bayes | 0.749583 | 0.874951 | 0.749583 | 0.767672 |

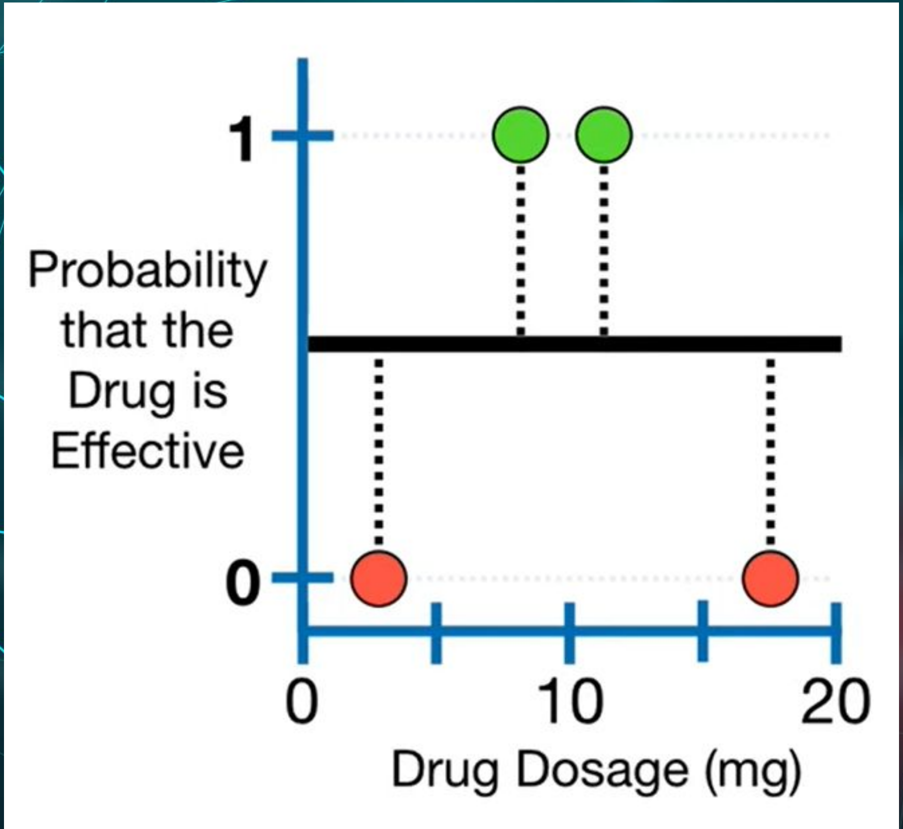
[5]:

| Model | Accuracy | Balanced Accuracy | ROC AUC | F1 Score | Time Taken |
|---|---|---|---|---|---|
| XGBClassifier | 0.93 | 0.89 | 0.89 | 0.93 | 0.50 |
| LGBMClassifier | 0.93 | 0.88 | 0.88 | 0.93 | 0.49 |
| RandomForestClassifier | 0.93 | 0.87 | 0.87 | 0.93 | 3.03 |
| AdaBoostClassifier | 0.91 | 0.86 | 0.86 | 0.91 | 1.26 |
| ExtraTreesClassifier | 0.92 | 0.86 | 0.86 | 0.92 | 1.92 |
| BaggingClassifier | 0.92 | 0.86 | 0.86 | 0.92 | 0.79 |
| DecisionTreeClassifier | 0.89 | 0.85 | 0.85 | 0.89 | 0.15 |
| SVC | 0.91 | 0.84 | 0.84 | 0.91 | 13.25 |
| NearestCentroid | 0.83 | 0.84 | 0.84 | 0.84 | 0.10 |
| LinearDiscriminantAnalysis | 0.89 | 0.83 | 0.83 | 0.89 | 0.27 |
| LogisticRegression | 0.89 | 0.83 | 0.83 | 0.89 | 0.33 |
| CalibratedClassifierCV | 0.89 | 0.83 | 0.83 | 0.89 | 6.59 |
| LinearSVC | 0.89 | 0.83 | 0.83 | 0.89 | 2.12 |
| QuadraticDiscriminantAnalysis | 0.73 | 0.82 | 0.82 | 0.75 | 0.16 |
| GaussianNB | 0.73 | 0.82 | 0.82 | 0.75 | 0.06 |

# Methodology: XGBoost

- **Definition**:
    - XGBoost (eXtreme Gradient Boosting) is a distributed, open-source machine learning library that uses **gradient boosted decision trees**, a **supervised learning boosting algorithm** that makes use of **gradient descent**.
    - It is known for its **speed**, **efficiency** and **ability to scale** well with large datasets.
- **Reasons** for choosing:
    - High in accuracy
    - Decision tree, Ensemble, Boosting technique (reduce overfitting)

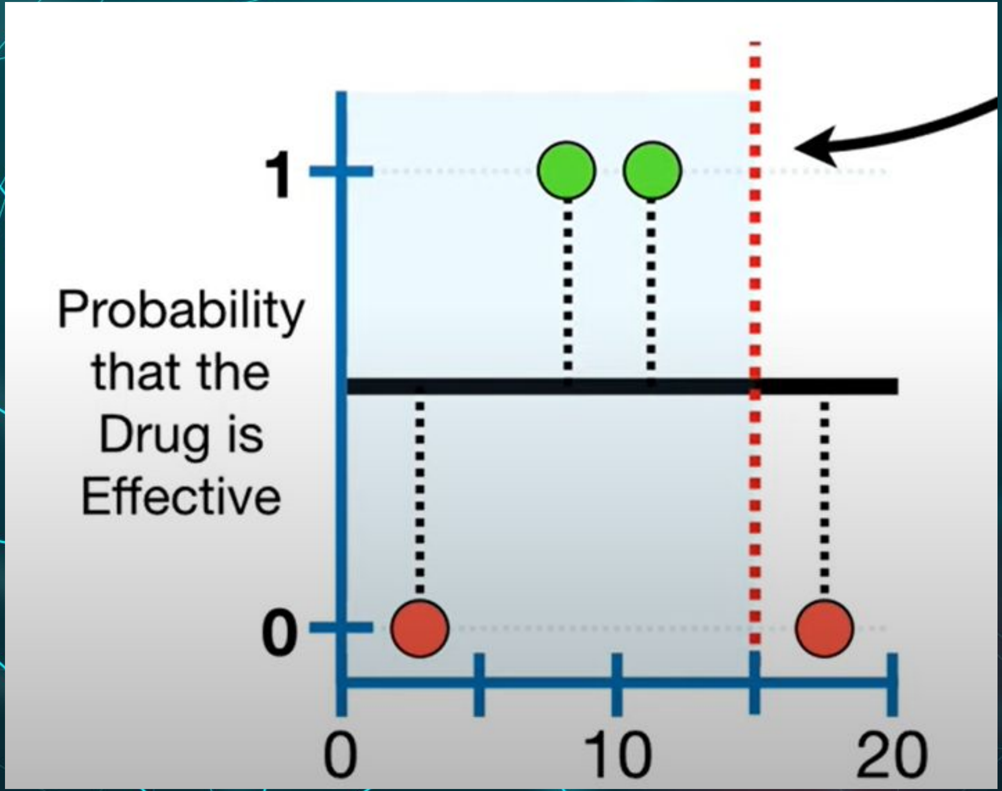
# XGBoost

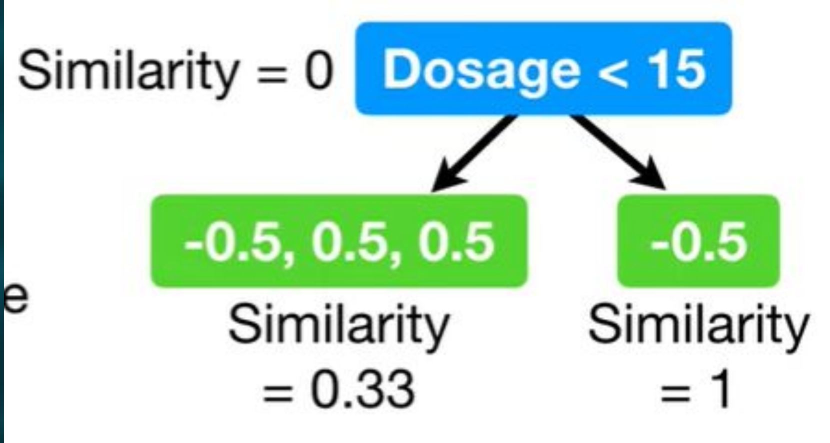



$$\frac{\left(\sum \text{Residual}_i\right)^2}{\sum \left[\text{Previous Probability}_i \times (1 - \text{Previous Probability}_i)\right] + \lambda}$$

**-0.5, 0.5, 0.5, -0.5**

# How to split a node

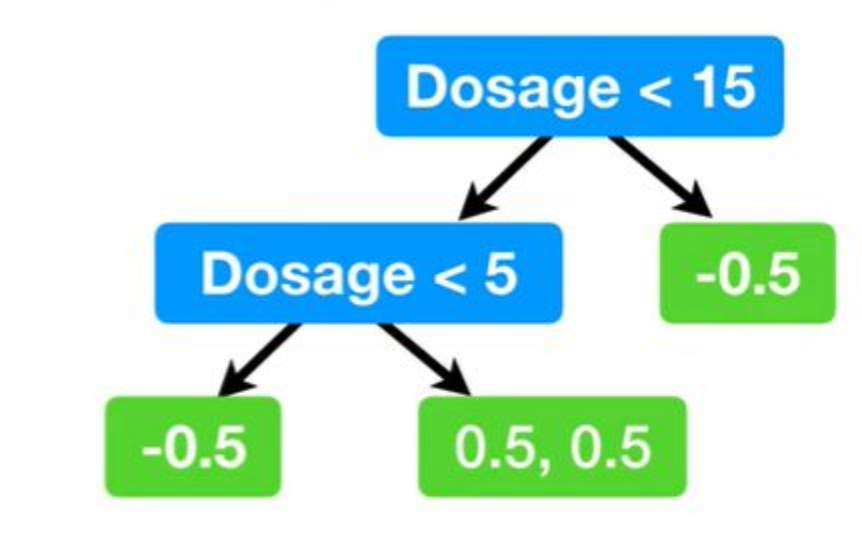



$$\text{Gain} = \text{Left}_{\text{Similarity}} + \text{Right}_{\text{Similarity}} - \text{Root}_{\text{Similarity}}$$

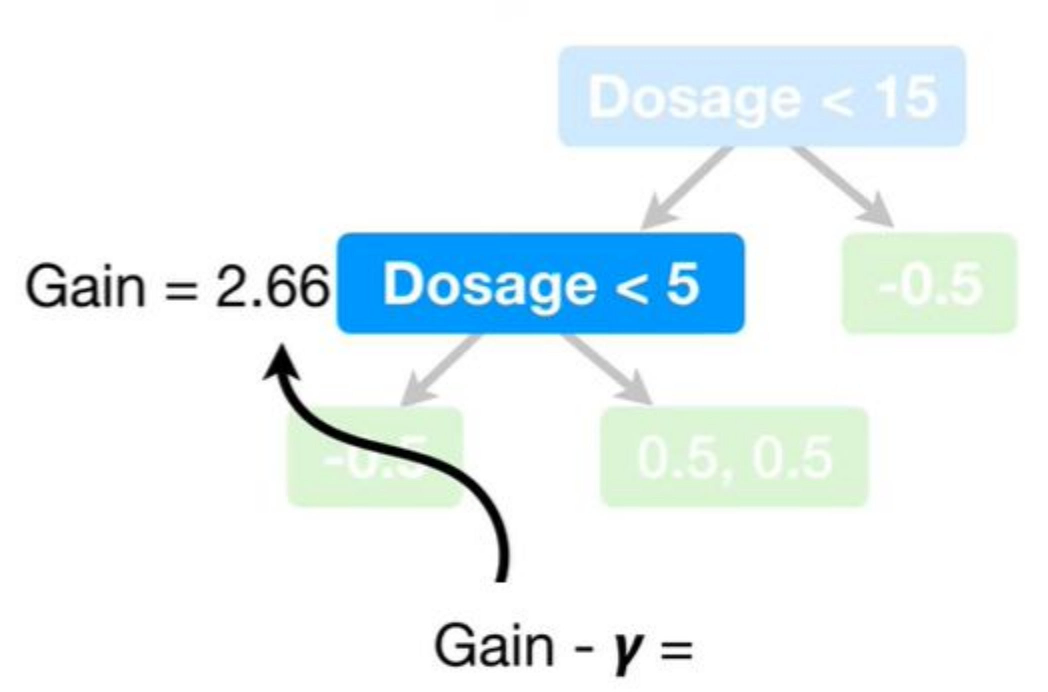
$$\text{Gain} = 0.33 + 1 - 0 = 1.33$$

# Cover (min_child_weight)



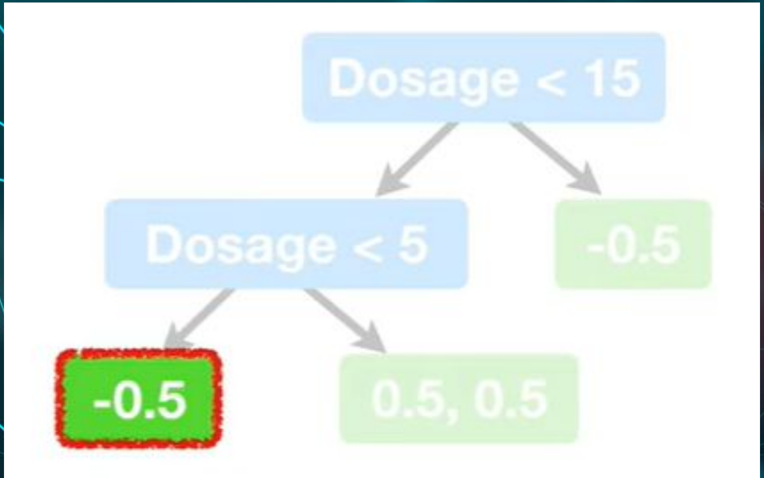

$$\text{Cover} = \sum \left[\text{Previous Probability}_i \times (1 - \text{Previous Probability}_i)\right]$$

# Pruning



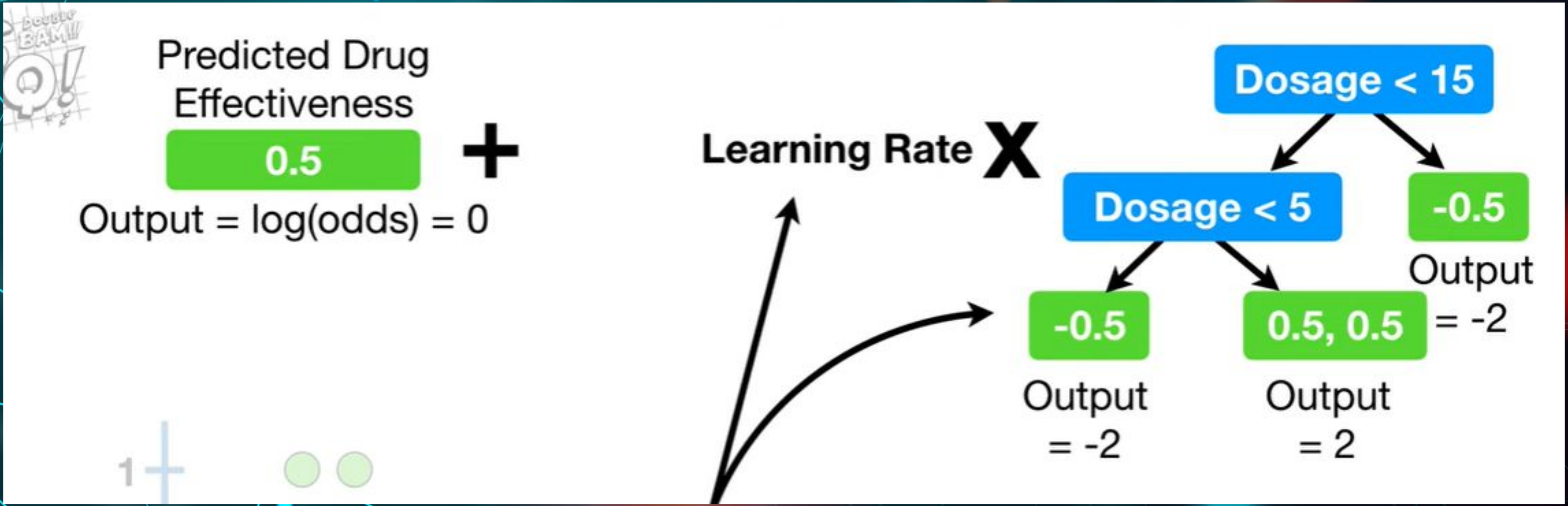
# Output value

$$\frac{\left(\sum \text{Residual}_i\right)}{\sum \left[\text{Previous Probability}_i \times (1 - \text{Previous Probability}_i)\right] + \lambda}$$



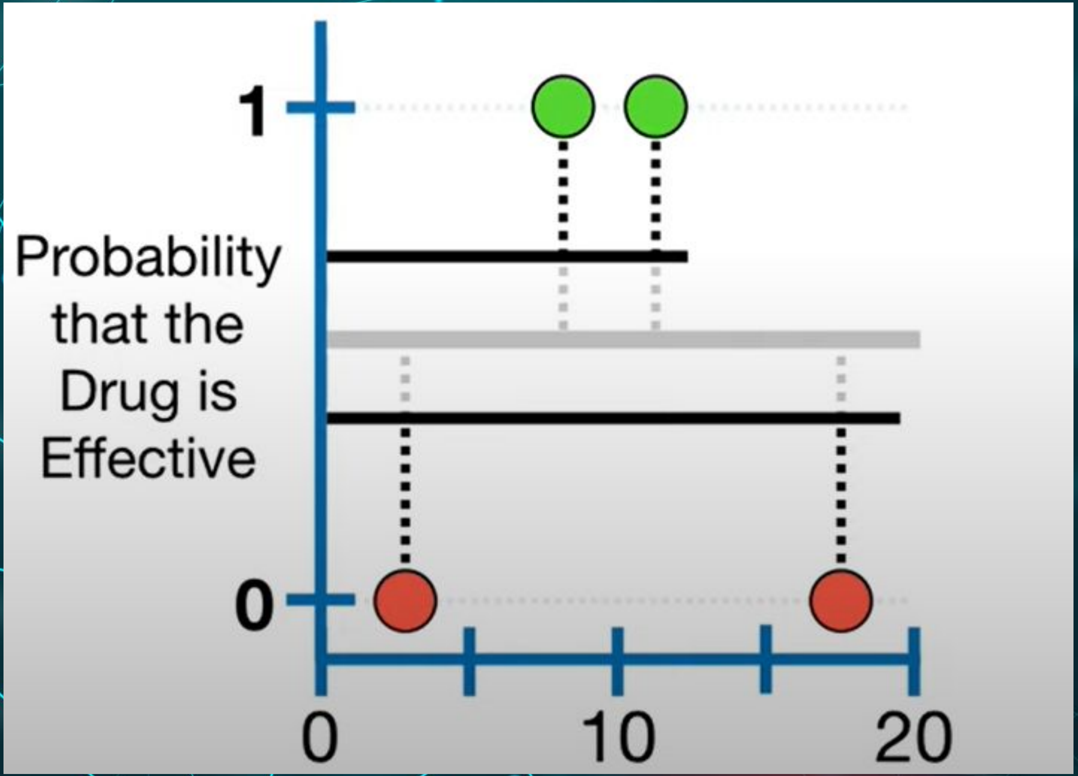

$$\frac{-0.5}{0.5 \times (1 - 0.5) + \lambda}$$

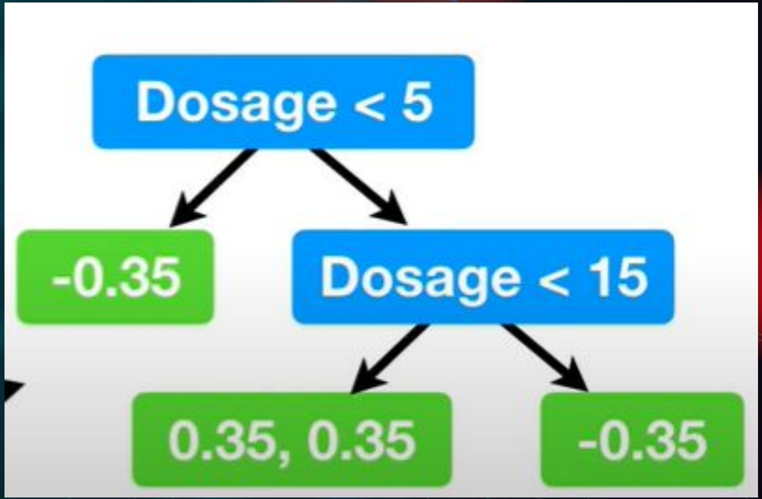
# Learning


Predicted Drug Effectiveness
0.5
Output = log(odds) = 0
+
Learning Rate X
Dosage < 15
Dosage < 5
-0.5
Output = -2
-0.5
Output = -2
0.5, 0.5
Output = 2

$$\text{Probability} = \frac{e^{0.6}}{1 + e^{0.6}} = 0.65$$

$$\log(\text{odds})\ \text{Prediction} = 0 + (0.3 \times 2) = 0.6$$


1
0
Probability that the Drug is Effective
0
10
20

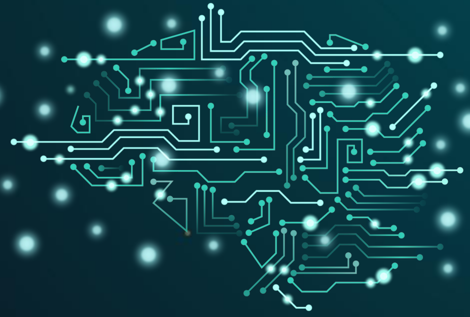

Dosage < 5
-0.35
Dosage < 15
0.35, 0.35
-0.35

# XGBoost Code Implementation

## Methods

### Regularization
Reduce overfitting by penalizing overly complex models

### Learning rate
Ensure the model doesn't become complex too quickly



### Tree Pruning
Post-pruning, stop growing the tree if performance no longer improves

### Cross-Validation
Ensure the model is generalizing well across all subsets of the data

### Subsampling
Reduce overfitting, effectively averages out the decision boundaries

### Ensemble Learning
Combine multiple weak learners into a strong learner, reduce the variance of the predictions

# Methodology: XGBoost

- **XGBClassifier class:**
  - Constructor
  - Fit
  - Predict
- **TreeBooster class:**
  - Constructor
  - _maybe_insert_child_nodes: Recursive Split
  - is_leaf(self): Leaf Node check
  - _find_better_split(self, feature_idx) : Finding the Best Split
  - predict(self, X) : Prediction for Multiple Rows
  - _predict_row(self, row) : Prediction for a Single Row

# Methodology: XGBoost

- **SquaredErrorObjective class:**
  - loss(self, y, pred): Mean Squared Error (MSE) loss
  - gradient(self, y, pred): gradient (first derivative) of the MSE loss with respect to predicted values
  - hessian(self, y, pred): Hessian (second derivative) of the MSE loss with respect to the predicted values
- **LogLossObjective class:**
  - loss(self, y, pred): binary cross-entropy (log) loss
  - gradient(self, y, pred): gradient (first derivative) of the log loss with respect to predicted values
  - hessian(self, y, pred): Hessian (second derivative) of the log loss with respect to the predicted values

# Methodology: XGBoost

- **Class XGBClassifier:**
  - Constructor:
    - subsample: The fraction of data used in each training iteration.
    - learning_rate: The step size for adjusting predictions at each iteration.
    - base_score: The initial prediction value before training.
    - max_depth: The maximum depth of each decision tree.
    - random_seed: A seed for random number generation to ensure reproducibility.

# Methodology: XGBoost

- **Class XGBClassifier:**
  - fit
    - Input: X_train, y_train, Loss function, n_estimators
    - Subsampling X_train and use that data to generate new decision tree
    - Use the new decision tree to update current_prediction base on learning rate

# Methodology: XGBoost

- **Class XGBClassifier:**
  - Predict:
    - The predict method computes the prediction for input X based on the trained trees.
    - It calculates the predictions for each tree in Tree Boosters and sums them up.
    - The final prediction is obtained by adding the base score and adjusting with the learning rate.
    - It then converts the continuous output into binary predictions (0 or 1)

# Methodology: XGBoost

- **TreeBooster class:**
  - __init__(self, X, g, h, params, max_depth, idxs=None):
    - Initializes the TreeBooster object by accepting the dataset (X), gradients (g), Hessians (h), and other hyperparameters such as max_depth, min_child_weight, reg_lambda, and gamma.
    - The constructor sets up the initial values, including the root node's prediction (value), and if the max_depth is greater than zero, it proceeds to insert child nodes (i.e., recursively splits the data into sub-trees).
  - _maybe_insert_child_nodes(self):
    - This method attempts to find the best feature and threshold to split the data.
    - It iterates over all features (feature_idx) and calls the _find_better_split() method to evaluate potential split points for each feature.
    - If a good split is found, the data is split into two groups, and child nodes (left and right subtrees) are recursively created.

# Methodology: XGBoost

- **TreeBooster class:**
  - _find_better_split(self, feature_idx):
    - This method evaluates all potential split points for a given feature (feature_idx) and calculates the gain for each split.
    - The method uses the gradient and Hessian values (g and h) to determine the best point to split the data, based on a formula from the XGBoost paper (equation 7).
    - It sorts the data based on the feature values and tests each consecutive pair of values to compute the gain for the split.
    - If the gain improves, it updates the best split found so far and sets the threshold for the split.

# Methodology: XGBoost

- **Treebooster class:**
  - predict(self, X):
    - This method takes the dataset X and predicts the output by iterating over each row and passing it through the tree recursively.
    - It calls the _predict_row() method for each individual row to get predictions from the tree.
  - _predict_row(self, row):
    - This recursive method predicts the value for a single row of data.
    - If the current node is a leaf, it returns the value stored at that node.
    - If not, it checks the value of the feature corresponding to the split and chooses the left or right child node to continue the prediction.

# Methodology: XGBoost

- **SquaredErrorObjective class:**
  - loss(self, y, pred):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$(y_i:\ actual\ value,\ \hat{y}_i:\ predicted\ value)$$

  - gradient(self, y, pred):

$$\frac{\delta MSE}{\delta \hat{y}_i} = \hat{y}_i - y_i$$

  - hessian(self, y, pred):

$$\frac{\delta^2 MSE}{\delta \hat{y}_i^2} = 1$$

- **LogLossObjective class:**
  - loss(self, y, pred):

$$Log\ Loss = -(y\ log(\hat{y})) + (1 - y)log(1 - \hat{y})$$

$$(y:\ true\ label,\ \hat{y}:\ predicted\ probability)$$

  - gradient(self, y, pred):

$$\frac{\delta LogLoss}{\delta \hat{y}} = \hat{y} - y$$

  - hessian(self, y, pred):

$$\frac{\delta^2 LogLoss}{\delta^2 \hat{y}} = \hat{y}(1 - \hat{y})$$

# Results

## Adjust parameters

| Times | Accuracy | F1 Score | Precision | Recall |
|---|---|---|---|---|
| 1 | 93.38% | 0.8514 | 0.9282 | 0.7863 |
| 2 | 93.39% | 0.8527 | 0.9213 | 0.7936 |
| 3 | 93,31% | 0.8482 | 0.9354 | 0.7759 |
| 4 | 93.16% | 0.8444 | 0.9342 | 0.7704 |
| 5 | 93.18% | 0.8473 | 0.9205 | 0.7849 |

# Results

## Adjust parameters

| Times | Accuracy | F1 Score | Precision | Recall |
|---|---|---|---|---|
| 6 | 93.39% | 0.8526 | 0.9220 | 0.7929 |
| 7 | 93.38% | 0.8518 | 0.9257 | 0.7887 |
| 8 | 93.38% | 0.8499 | 0.9360 | 0.7784 |
| 9 | 93.47% | 0.8538 | 0.9264 | 0.7918 |
| 10 | 93.38% | 0.8508 | 0.9302 | 0.7839 |

# Results

## Adjust parameters

| Times | Accuracy | F1 Score | Precision | Recall |
|---|---|---|---|---|
| 11 | 93.39% | 0.8522 | 0.9244 | 0.7905 |
| 12 | 93.21% | 0.8481 | 0.9199 | 0.7867 |
| 13 | 93.33% | 0.8502 | 0.9269 | 0.7853 |
| 14 | 93.29% | 0.8501 | 0.9209 | 0.7894 |
| 15 | 93.21% | 0.8490 | 0.9146 | 0.7922 |

# RESULTS

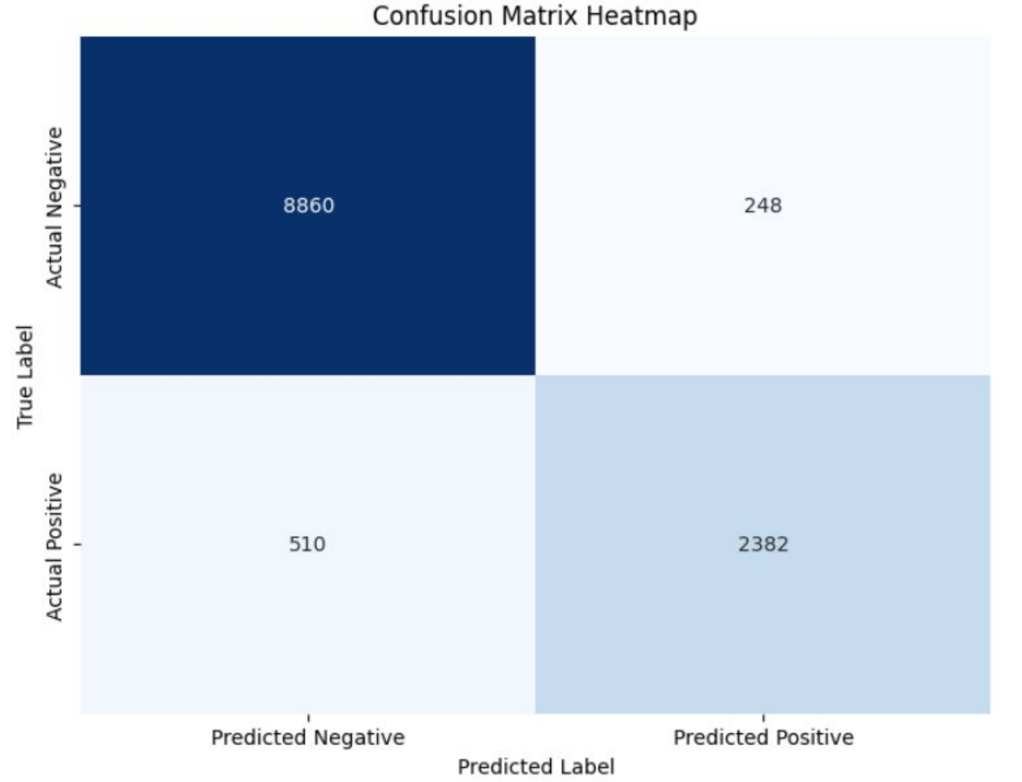
## Highest Score

| | |
|---|---|
| Accuracy | 93.68% |
| F1 Score | 0.8627 |
| Precision | 0.9057 |
| Recall | 0.8237 |

**Training time** 851.016900 s

**Testing time** 112.311276 s

# CONFUSION MATRIX



# Final Project

## STAGES

### Approaches
Try different training models

25%

### Code
Code XGBoost implementation

50%

### Parameters
Adjust the parameters

75%

### Report and Slide
Prepare the Report and Slide

100%

# Members and Contributions

## OUR MEMBERS

| B. The Kiet | Ph. Ng. Hai Duong | Ph. Mai Anh |
|---|---|---|
| • Data collection and preprocessing<br>• exploratory data analysis | • Model development<br>• hyperparameter tuning | • Model evaluation<br>• Slides preparing |
| 33.33% | 33.33% | 33.33% |

# Thank you for listening