DECISON TREE

TREE STRUCTURE IN MACHINE LERNING

QIANKANG WANG

Data Structures in Artificial Intelligence

Agenda

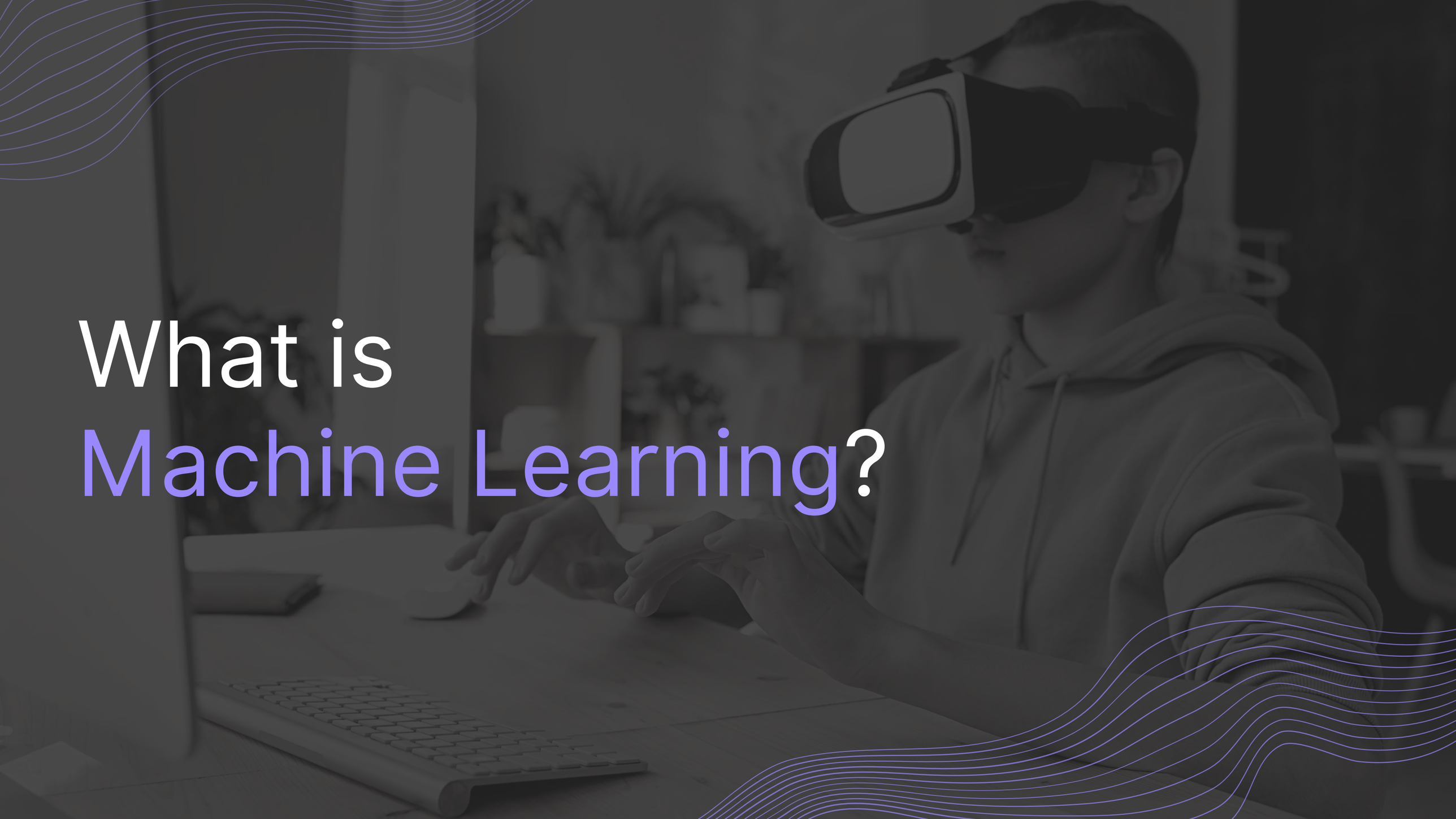
DECISION TREE - TREE STRUCTURE IN MACHINE LERNING

---

1. Why I chose decision trees?
2. What is Machine Learning?
3. What is a decision tree?
4. How do decision trees work?
5. How to build a decision tree?
6. One of the algorithms for decision tree purity calculation - Gini impurity
7. My Project - Decision Tree Prediction Classic Machine Learning Dataset: Titanic
8. Challenges in the project
9. My thoughts

Why I chose decision trees?

When I first learned about tree structures in this class, I thought about decision trees. Although machine learning is usually implemented in tools like Python, R, Jupyter, etc. rather than C++. However, I still want to use C++ because it helps me better understand data structures and also the nature of machine learning. Through this project, I was able to better understand the nature of data structures, and better integrate data structures into my machine learning knowledge.



What is Machine Learning?

Traditional programming

Programmers write detailed instructions (code) that specify how the computer should process specific inputs to produce a desired output.

Programmers are concerned with the correctness and robustness of their programs. They "materialize" the human mind into lines of code filled with if-conditional statements that respond to each "individual" situation.

Machine learning, also known as artificial intelligence.

The programmer writes the algorithmic framework and provides the dataset, and the algorithm automatically constructs the model by learning patterns and regularities in the data.

Data scientists (designers of machine learning algorithms) often work with uncertainty and variability. The core characteristic of machine learning is "learning from data, with performance gains, and without explicit programming".

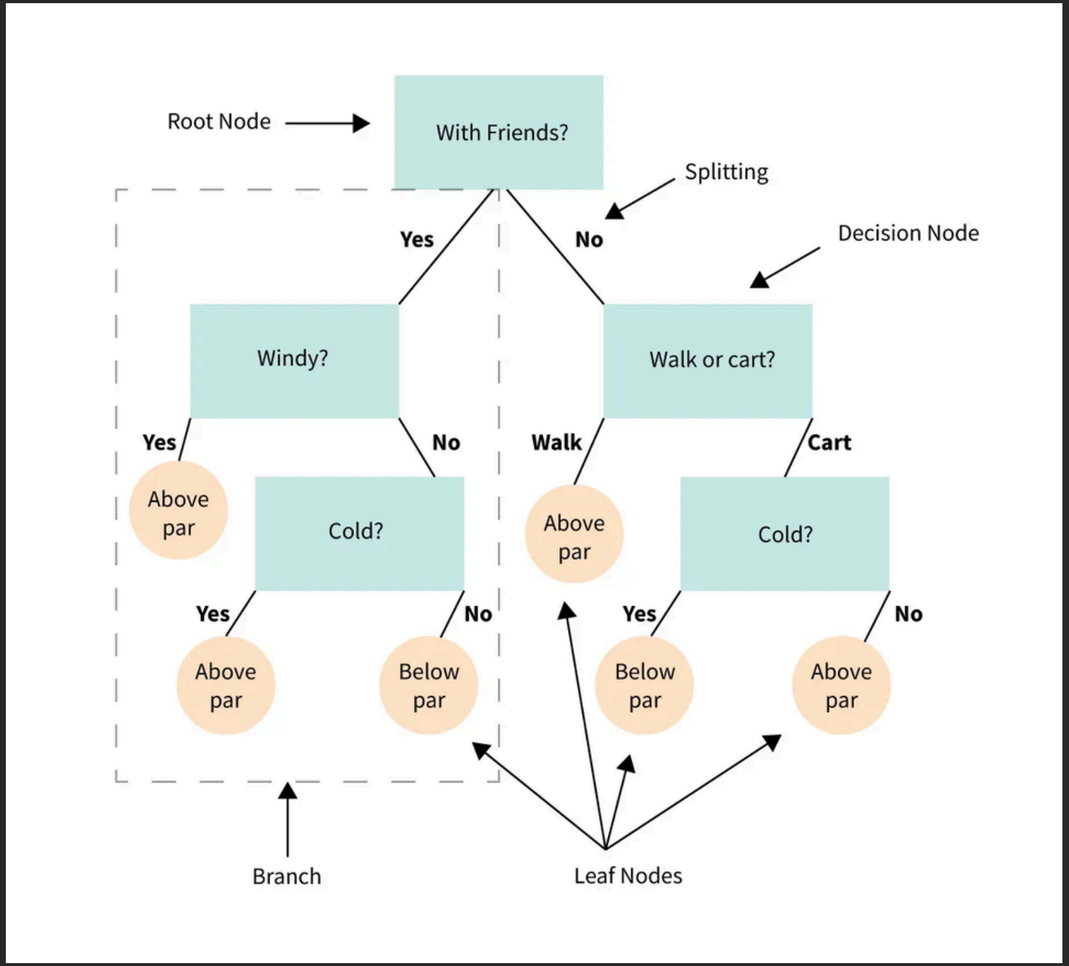
What is a decision tree?

What is a decision tree?

Decision tree is a typical tree model. In it, each internal node represents the judgment of an attribute, each branch represents the output of the judgment, and finally each leaf node represents the classification result.

CONSTRUCTING DECISION TREES:

Decision tree is a commonly used classification method to construct a decision tree from a given training dataset by supervised learning. In the classification phase, the classification results are obtained by dividing the decision tree layer by layer starting from the root node up to the leaf nodes according to the classification attributes of the decision tree.



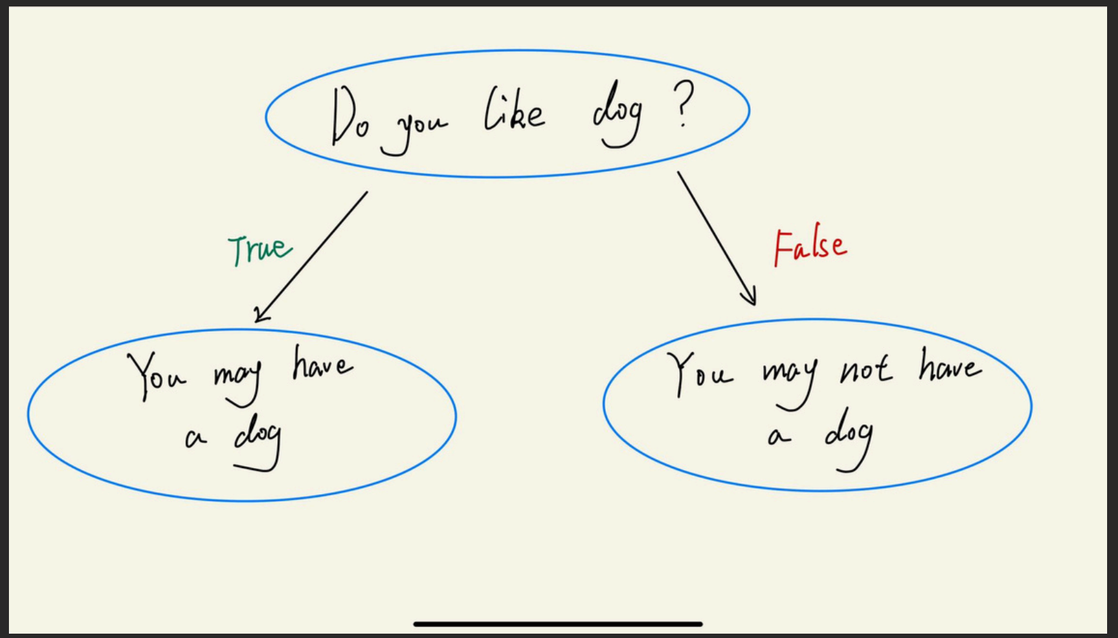
How do decision trees work?

How do decision trees work?

FOR EXAMPLE:

We constructed a simple model to predict whether a person owns a dog or not. Liking dogs is a feature of this model. If a person likes dogs, then the tree will be "true", returning "he may have a dog", and vice versa "he may not have a dog".

In this model, the root node is whether to like dogs or not, and the leaves are predictions.

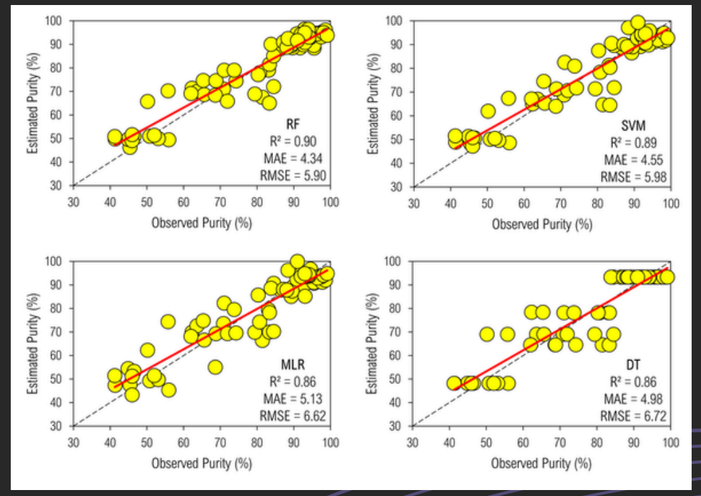


How to build a decision tree?

How to build a decision tree?

Briefly, the construction of a decision tree is divided into the following key steps:

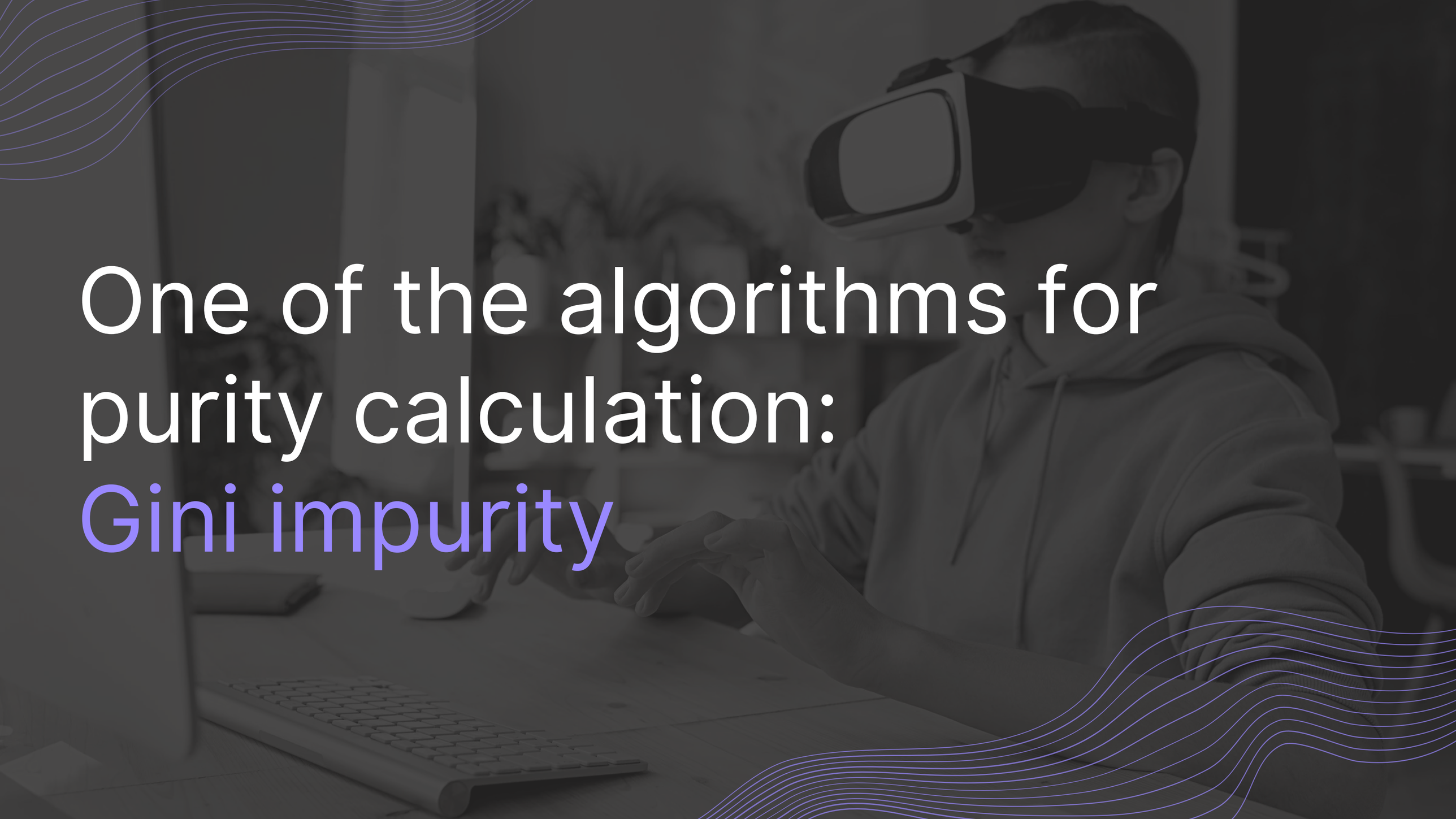
1. Select which features to use as the root node.
2. Which features is selected as the child node.
3. When to stop building, i.e. leaf node generation



Graph: Analyzing purity using linear regression

How to choose the right features?

Data scientists refer to the goodness of a feature dataset as purity. Simply put, the higher the purity, the cleaner the results of the feature dataset, which means more accurate feedback values for the features. As an example: Let's still take dog ownership as example, we feature whether or not we like dogs. When we build the model, if among people who like dogs, almost all of them own dogs, and among people who don't like dogs, almost all of them don't own dogs. Then the more correct the feedback will be for this feature . Then we say this feature has high purity.

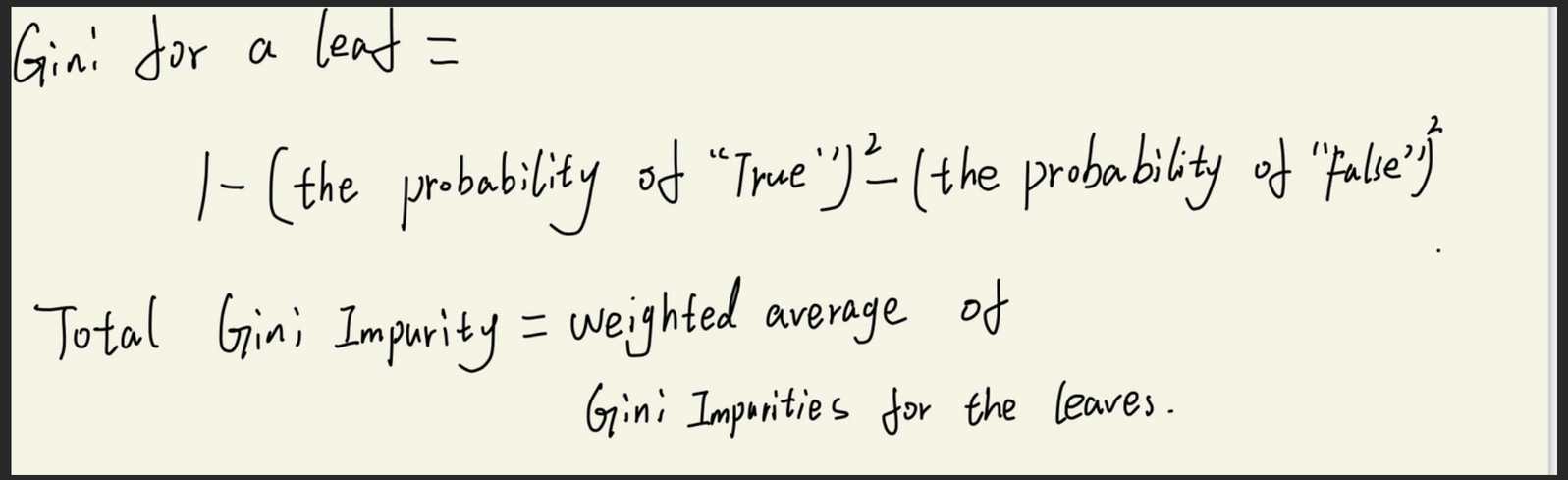


One of the algorithms for purity calculation: Gini impurity

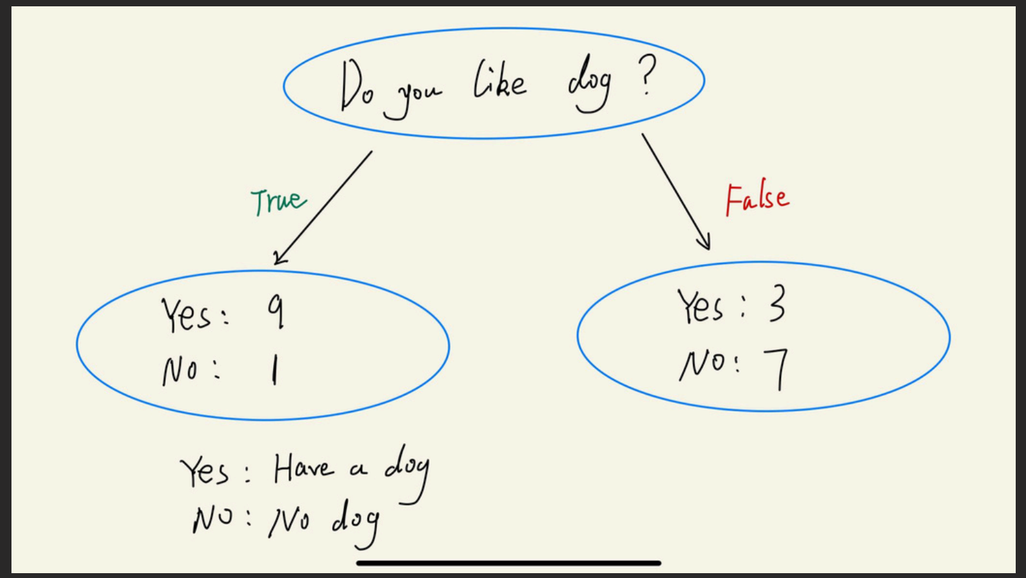
Gini impurity:

Gini impurity is an algorithm that describes the purity of a feature dataset. In short, when the Gini impurity is higher, it means the feature dataset is more chaotic. Therefore, feature datasets with lower Gini impurity are generally chosen.

Formula for calculating Gini impurity



For example:



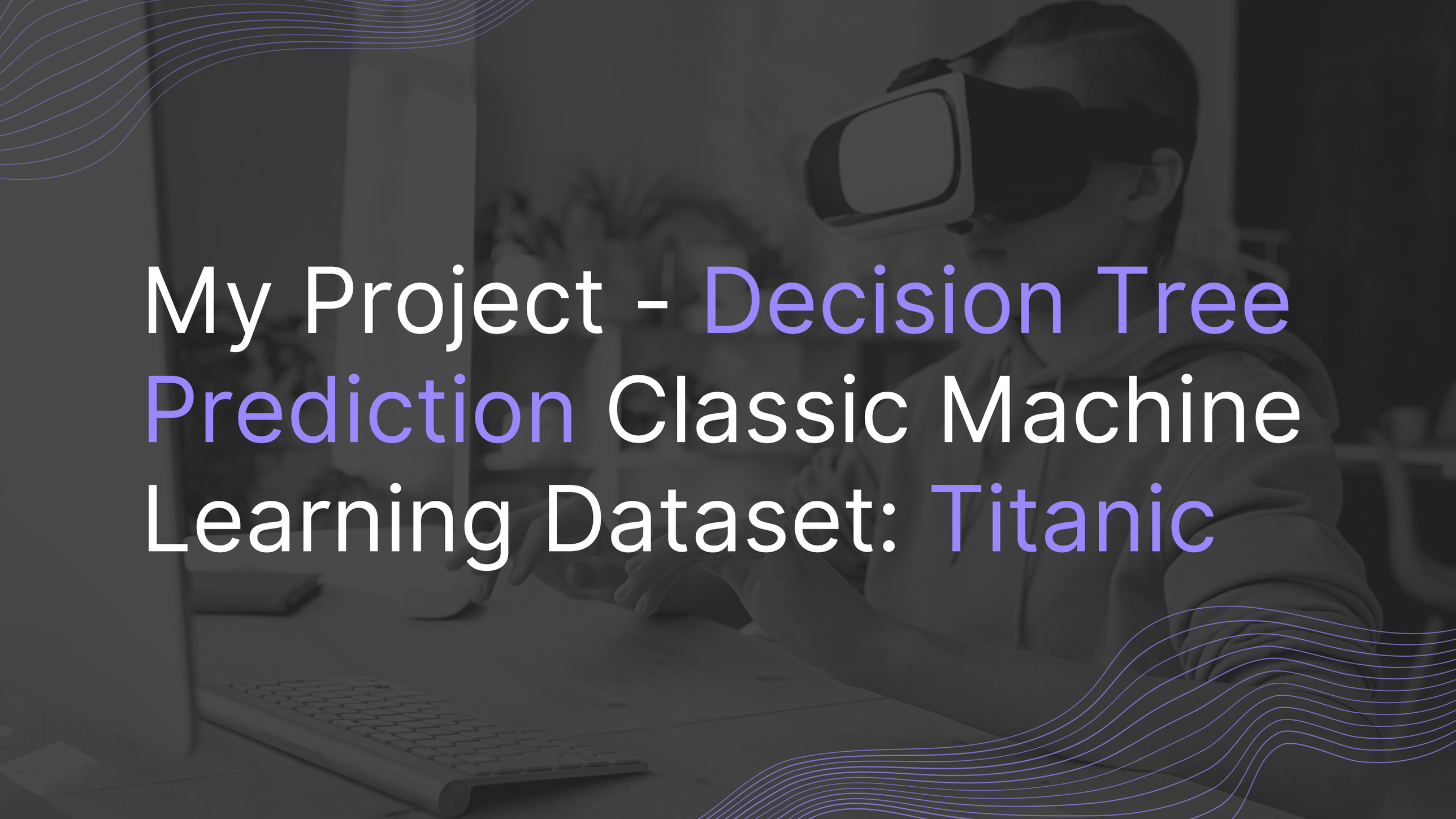
In the case of dog lovers. We recorded 20 people. Ten of them liked dogs, and of those who liked dogs, nine owned dogs and one did not. Of the other ten people who did not like dogs, three owned dogs and seven did not.

Gini impurities are calculated as shown in the figure at right:

$$\text{Gini for True} = 1 - \left(\frac{9}{9+1}\right)^2 - \left(\frac{1}{9+1}\right)^2 = 0.18$$

$$\text{Gini for False} = 1 - \left(\frac{3}{3+7}\right)^2 - \left(\frac{7}{3+7}\right)^2 = 0.42$$

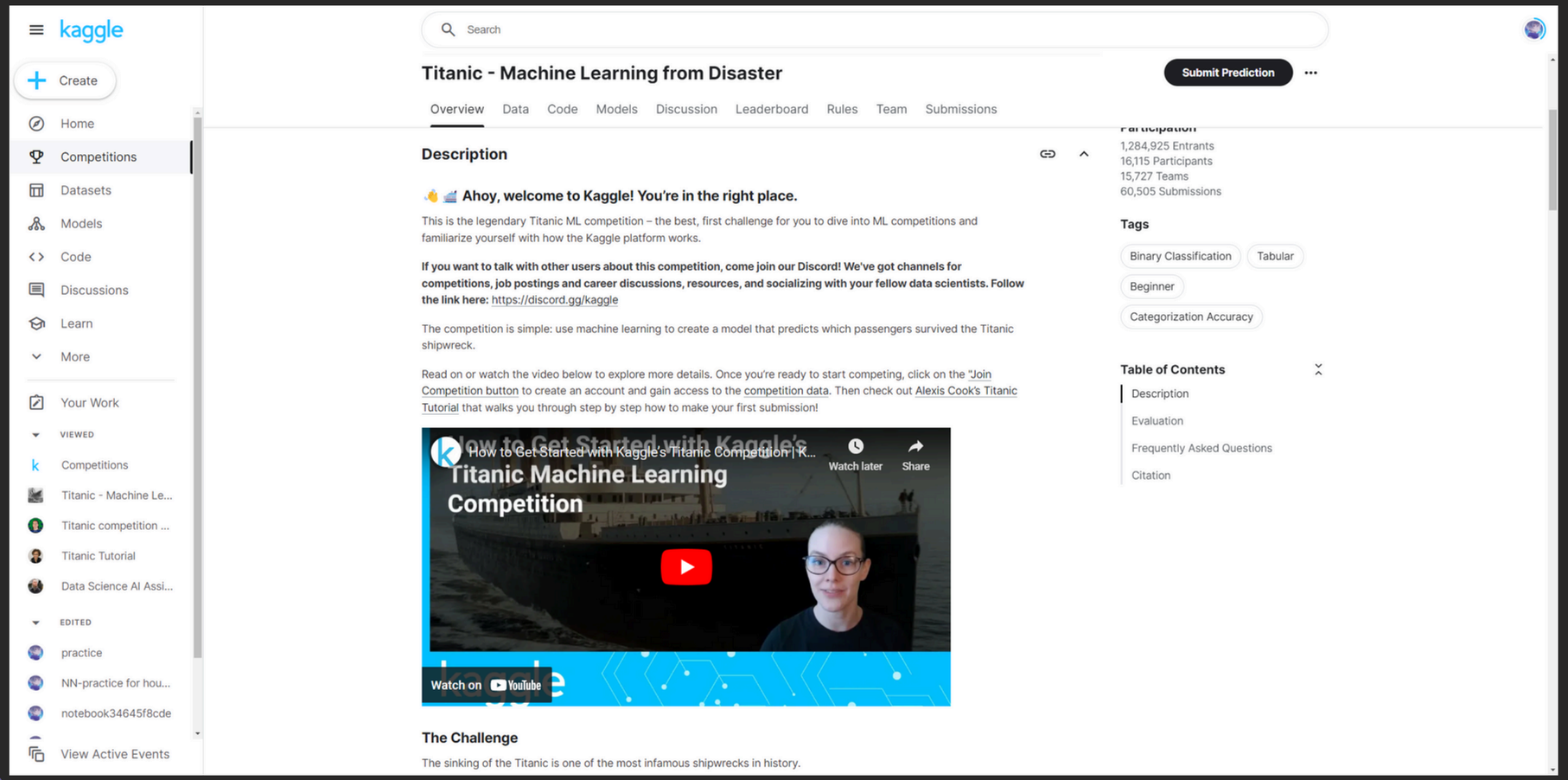
$$\text{Total Gini} = \left(\frac{10}{10+10}\right) \cdot 0.18 + \left(\frac{10}{10+10}\right) \cdot 0.42 = 0.3$$



My Project - Decision Tree Prediction Classic Machine Learning Dataset: Titanic

Titanic - Machine Learning from Disaster

Titanic is one of the most iconic machine learning datasets on Kaggle. It is also the first dataset of many machine learning engineers.



Titanic Data Set

The Titanic dataset has many features. Different features require different analysis. This is difficult to implement manually in C++. Therefore, I have chosen only a few of the most representative features to analyze. (Sex, Age, pclass)

Data Dictionary

Variable	Definition	Key
survival	Survival	0 = No, 1 = Yes
pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd
sex	Sex	
Age	Age in years	
sibsp	# of siblings / spouses aboard the Titanic	
parch	# of parents / children aboard the Titanic	
ticket	Ticket number	
fare	Passenger fare	
cabin	Cabin number	
embarked	Port of Embarkation	C = Cherbourg, Q = Queenstown, S = Southampton

Step 1 - Data Preprocessing

First I preprocessed the dataset in Python and filled in the missing values in the dataset and One-Hot Encoding some of the data stored as strings. The dataset is stored in CSV format. I didn't want to use a third-party library to read the CSV format, so I converted it to txt.

Original data set (top five)

	A	B	C	D	E	F	G	H	I	J	K	L
1	Passenge	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
2	1	0	3	Braund, M	male	22	1	0	A/5 21171	7.25		S
3	2	1	1	Cumings,	female	38	1	0	PC 17599	71.2833	C85	C
4	3	1	3	Heikkinen	female	26	0	0	STON/O2.	7.925		S
5	4	1	1	Futrelle, M	female	35	1	0	113803	53.1	C123	S
6	5	0	3	Allen, Mr.	male	35	0	0	373450	8.05		S

Processed data sets

	A	B	C	D	E	F	G	H
1	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
2	0	3	0	22	1	0	7.25	2
3	1	1	1	38	1	0	71.2833	0
4	1	3	1	26	0	0	7.925	2
5	1	1	1	35	1	0	53.1	2
6	0	3	0	35	0	0	8.05	2

Convert to txt format

```
0 3 0 22.0 1 0 7.25 2
1 1 1 38.0 1 0 71.2833 0
1 3 1 26.0 0 0 7.925 2
1 1 1 35.0 1 0 53.1 2
0 3 0 35.0 0 0 8.05 2
```

Step 2 - Read the dataset and create passengers in C++

I wrote a file called ReadPassenger.cpp to read the passenger. In this project, I am using Vector to stored passenger information. And I created a print function to print it out

```cpp
vector<Passenger> readPassengers(string& filename) {
    vector<Passenger> passengers;
    ifstream file(filename);
    string line;
    if (file.is_open()) {
        while (getline(file, line)) {
            std::istringstream iss(line);
            Passenger passenger;
            iss >> passenger.survived >> passenger.pclass >> passenger.sex
                >> passenger.age >> passenger.sibsp >> passenger.parch
                >> passenger.fare >> passenger.embarked;
            passengers.push_back(passenger);
        }
        file.close();
    }
    else {
        cerr << "Unable to open file: " << filename << endl;
    }
    return passengers;
}

void printPassengers(vector<Passenger>& passengers) {
    for (int i = 0; i < passengers.size(); ++i) {
        cout << "Survived: " << passengers[i].survived
            << ", Class: " << passengers[i].pclass
            << ", Sex: " << passengers[i].sex
            << ", Age: " << passengers[i].age
            << ", Siblings/Spouses: " << passengers[i].sibsp
            << ", Parents/Children: " << passengers[i].parch
            << ", Fare: " << passengers[i].fare
            << ", Embarked: " << passengers[i].embarked << endl;
    }
}
```

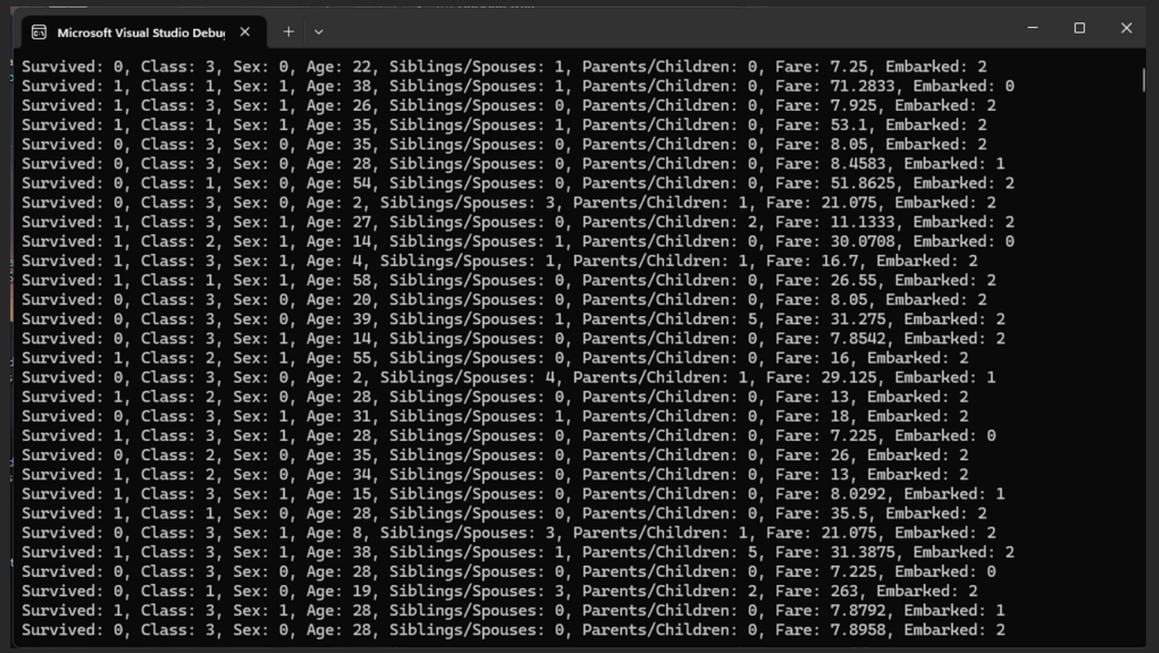
Step 2 - Read the dataset and create passengers in C++, continued.

I defined the passenger in the main function and printed it out

```cpp
int main() {
    string filename = "F:\processed_train_simple.txt";
    vector<Passenger> passengers = readPassengers(filename);

    printPassengers(passengers);
```

Command Window Results



Great!
It's work!

Step 3 - Create Gini function for each feature.

Next, we need to analyze the different features. This is because each feature is different. Therefore, when we want to model, we have to construct the formula for Gini impurities. Therefore, I created another file: gini.cpp.

Since the complexity of creating and analyzing all the features was too high, I chose only three features for simple creation: Sex, pclass, and Age.

Let's start with sex!

Step 3 continued - Gini impurity for sex

In the case of sex, since there are only two types: male and female, we can use the bool type to denote sex, with male denoting 0 and female denoting 1. Since there are only two types, we can use the Gini impurity formula directly.

```
3   double calculateGiniForSex(vector<Passenger>& passengers) {
4       double countFemale = 0, countMale = 0;
5       double survivedFemale = 0, survivedMale = 0;
6
7       for (int i = 0; i < passengers.size(); ++i) {
8           if (passengers[i].sex == 1) {  // Assuming 1 represents females
9               countFemale++;
10              if (passengers[i].survived)
11                  survivedFemale++;
12          }
13          else {
14              countMale++;
15              if (passengers[i].survived)
16                  survivedMale++;
17          }
18      }
19
20      double pSurvivedFemale = 0, pSurvivedMale = 0;
21
22      if (countFemale > 0) {
23          pSurvivedFemale = survivedFemale / countFemale;
24      }
25      if (countMale > 0) {
26          pSurvivedMale = survivedMale / countMale;
27      }
28
29      double giniFemale = 1 - (pSurvivedFemale * pSurvivedFemale) - (1 - pSurvivedFemale) * (1 - pSurvivedFemale);
30      double giniMale = 1 - (pSurvivedMale * pSurvivedMale) - (1 - pSurvivedMale) * (1 - pSurvivedMale);
31
32      double total = countFemale + countMale;
33      double totalGini = 0.0;
34
35      if (total > 0) {
36          totalGini = countFemale / total * giniFemale + countMale / total * giniMale;
37      }
38
39      return totalGini;
40  }
```

First, declare int for counting the number of male and female

Note: Since we'll need to calculate the percentage later, we'll declare these variables as double so we don't have to convert them later.

counting the number of survived male and survived female

Calculation of the percentage of male and female survivors

plug into the Gini formula, respectively

calculate total Gini (weighted Gini) and return value.

Step 3 continued - Gini impurity for pclass

The pclass is a bit different because it involves three values: 1, 2, and 3. Therefore, the value of the gini we are calculating also needs to change. I use an array to store them separately. The analysis is very similar to sex except for the three values.

```cpp
double calculateGiniForPclass(vector<Passenger>& passengers) {
    double count[3] = { 0, 0, 0 }; // Array to count the number of passengers in each category
    double survivedCount[3] = { 0, 0, 0 }; // Number of survivors in each category

    for (int i = 0; i < passengers.size(); ++i) {
        if (passengers[i].pclass >= 1 && passengers[i].pclass <= 3) {
            int index = passengers[i].pclass - 1;
            count[index]++;
            if (passengers[i].survived) {
                survivedCount[index]++;
            }
        }
    }

    double total = passengers.size();
    double totalGini = 0.0;

    for (int i = 0; i < 3; i++) {
        if (count[i] > 0) {
            double pSurvived = survivedCount[i] / count[i];
            double gini = 1 - (pSurvived * pSurvived) - ((1 - pSurvived) * (1 - pSurvived));
            totalGini += (count[i] / total) * gini;
        }
    }

    return totalGini;
}
```

This function counts the number of people in different pclasses and the number of survivors separately

Calculate Gini values separately

Calculate the weighted Gini.

Step 3 continued - Gini impurity for pclass

Of course, we can't just calculate the Gini value of pclass. Because the decision tree is making a decision, it is actually a bool type. It's impossible to feed back three scenarios to make a prediction. Therefore, in the decision tree algorithm, we will usually calculate the Gini values of two neighboring cases separately, find the one with the lowest Gini value, and then divide it into two different cases. For example, suppose the lowest Gini value is between 1 and 2. The result of the split would be 1 and 23

```cpp
double findBestSplitForPclass(vector<Passenger>& passengers) {
    int counts[3] = { 0, 0, 0 };
    int survivedCounts[3] = { 0, 0, 0 };

    for (int i = 0; i < passengers.size(); ++i) {
        if (passengers[i].pclass >= 1 && passengers[i].pclass <= 3) {
            int index = passengers[i].pclass - 1;
            counts[index]++;
            if (passengers[i].survived) {
                survivedCounts[index]++;
            }
        }
    }

    double bestGini = 1.0;
    double bestSplit = -1;
    double total = passengers.size();

    for (int i = 0; i < 3; i++) {
        if (counts[i] > 0) {
            double pSurvived = survivedCounts[i] / counts[i];
            double gini = 1 - (pSurvived * pSurvived) - (1 - pSurvived) * (1 - pSurvived);
            double weightedGini = (counts[i] / total) * gini;
            if (weightedGini < bestGini) {
                bestGini = weightedGini;
                bestSplit = i + 1;
            }
        }
    }

    return bestSplit;
}
```

This function counts the number of people in different pclasses and the number of survivors separately

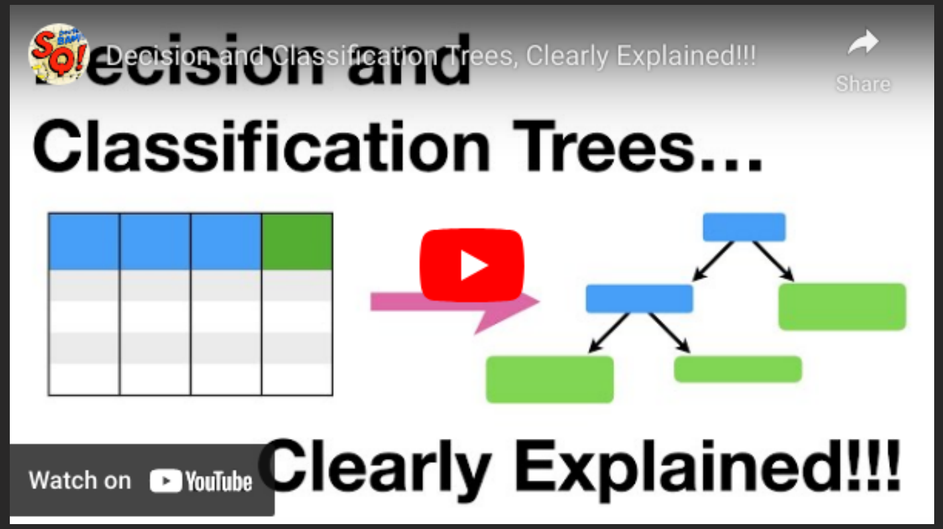
Calculate Gini value

Find the optimal split by finding the lowest Gini value.

Step 3 continued - Gini impurity for Age

Age is the hardest part. Because everyone's age is different. Some people may have the same age. Therefore, all our previous analysis methods are not applicable. In this case, we have to analyze the Gini values. Then, we need to sort the dataset first. (from smallest to largest). Here, I have used selection sorting algorithm to solve this problem. Then analyze the Gini value between two neighboring ages. Find the best Gini value Divide the dataset into two. This way we can execute the decision.

Below I've attached a video of how to calculate the Gini value for Age (from 9:55 to 11:45)



https://www.youtube.com/watch?v=_L39rN6gz7Y&t=765s

Step 3 continued - Gini impurity for Age

This is a selection sorting algorithm

```
void selectionSort(vector<Passenger>& passengers) {
    for (int i = 0; i < passengers.size() - 1; i++) {
        int minIndex = i;
        for (int j = i + 1; j < passengers.size(); j++) {
            if (passengers[j].age < passengers[minIndex].age) {
                minIndex = j;
            }
        }
        if (minIndex != i) {
            Passenger temp = passengers[i];
            passengers[i] = passengers[minIndex];
            passengers[minIndex] = temp;
        }
    }
}
```

Step 3 continued - Gini impurity for Age

I created a function that calculates the Gini value. Because it will be used many times later (By recursive function).

```
double calculateGini(int counts[2], double total) {
    if (total == 0) {
        return 0;
    }

    double sum = 0;

    // We are assuming counts is an array with two elements
    for (int i = 0; i < 2; ++i) {  // Since we know there are only two categories
        double probability = counts[i] / total;
        sum += probability * probability;
    }

    return 1 - sum;
}
```

Step 3 continued - Gini impurity for Age

The following function calculates the optimal splitting point of Age.

```
double findBestSplit(vector<Passenger>& passengers) {
    vector<Passenger> sortedPassengers = passengers;
    selectionSort(sortedPassengers);

    double bestGini = 1.0;
    double bestSplit = -1.0;
    int total = sortedPassengers.size();

    for (int i = 1; i < total; i++) {
        if (sortedPassengers[i].age != sortedPassengers[i - 1].age) {
            double currentSplit = (sortedPassengers[i].age + sortedPassengers[i - 1].age) / 2.0;

            int leftCounts[2] = { 0, 0 };  // [0] for not survived, [1] for survived
            int rightCounts[2] = { 0, 0 }; // [0] for not survived, [1] for survived

            for (int j = 0; j < i; j++) {
                leftCounts[sortedPassengers[j].survived]++;
            }
            for (int j = i; j < total; j++) {
                rightCounts[sortedPassengers[j].survived]++;
            }

            double leftGini = calculateGini(leftCounts, i);
            double rightGini = calculateGini(rightCounts, total - i);

            double weightedGini = (i * leftGini + (total - i) * rightGini) / total;
            if (weightedGini < bestGini) {
                bestGini = weightedGini;
                bestSplit = currentSplit;
            }
        }
    }

    return bestSplit;
}
```

Sort them first

I start in the middle and count the number of people of both ages and the number of survivors. Because Age is already sorted. So we can use a recursive algorithm to help us do this quickly (instead of a traditional loop, thank goodness I learned recursion!).

Step 3 continued - Gini impurity for Age

As with pclass, we need to calculate the Gini value of Age.

```cpp
double calculateGiniAtSplit(vector<Passenger>& passengers, double splitAge) {
    int leftCounts[2] = { 0, 0 };
    int rightCounts[2] = { 0, 0 };

    int leftTotal = 0, rightTotal = 0;

    for (int i = 0; i < passengers.size(); i++) {
        if (passengers[i].age <= splitAge) {
            leftCounts[passengers[i].survived]++;
            leftTotal++;
        }
        else {
            rightCounts[passengers[i].survived]++;
            rightTotal++;
        }
    }

    // Calculate Gini for each side
    double leftGini = calculateGini(leftCounts, leftTotal);
    double rightGini = calculateGini(rightCounts, rightTotal);

    // Calculate weighted Gini
    double total = leftTotal + rightTotal;
    double totalGini = (leftTotal * leftGini + rightTotal * rightGini) / total;

    return totalGini;
}
```

Much like finding the optimal splitting point, we utilize a recursive function here as well

Step 3 continued - Check Gini impurity functions

Check if it works

```cpp
int main() {
    string filename = "F:\processed_train_simple.txt";
    vector<Passenger> passengers = readPassengers(filename);

    printPassengers(passengers);

    double bestSplit = findBestSplit(passengers);
    cout << endl << "Gini impurity for Best split Age: " << bestSplit << endl;

    double giniImpurity = calculateGiniAtSplit(passengers, bestSplit);
    cout << "Gini Impurity at age " << bestSplit << " is " << giniImpurity << endl;

    // Calculate Gini impurity for sex split
    double giniSex = calculateGiniForSex(passengers);
    cout << "Gini impurity for sex: " << giniSex << endl;

    // Calculate Gini impurity for passenger class
    double giniPclass = calculateGiniForPclass(passengers);
    cout << "Gini impurity for passenger class: " << giniPclass << endl;

    double bestSplitPclass = findBestSplitForPclass(passengers);
        cout << "Gini impurity for Best split Age: " << bestSplitPclass << endl;
```

```
Survived: 0, Class: 3, Sex: 0, Age: 19, Siblings/Spouses: 0, Parents/Children: 0, Fare: 7.8958, Embarked: 2
Survived: 0, Class: 3, Sex: 0, Age: 28, Siblings/Spouses: 0, Parents/Children: 0, Fare: 7.8958, Embarked: 2
Survived: 1, Class: 1, Sex: 1, Age: 56, Siblings/Spouses: 0, Parents/Children: 1, Fare: 83.1583, Embarked: 0
Survived: 1, Class: 2, Sex: 1, Age: 25, Siblings/Spouses: 0, Parents/Children: 1, Fare: 26, Embarked: 2
Survived: 0, Class: 3, Sex: 0, Age: 33, Siblings/Spouses: 0, Parents/Children: 0, Fare: 7.8958, Embarked: 2
Survived: 0, Class: 3, Sex: 1, Age: 22, Siblings/Spouses: 0, Parents/Children: 0, Fare: 10.5167, Embarked: 2
Survived: 0, Class: 2, Sex: 0, Age: 28, Siblings/Spouses: 0, Parents/Children: 0, Fare: 10.5, Embarked: 2
Survived: 0, Class: 3, Sex: 0, Age: 25, Siblings/Spouses: 0, Parents/Children: 0, Fare: 7.05, Embarked: 2
Survived: 0, Class: 3, Sex: 1, Age: 39, Siblings/Spouses: 0, Parents/Children: 5, Fare: 29.125, Embarked: 1
Survived: 0, Class: 2, Sex: 0, Age: 27, Siblings/Spouses: 0, Parents/Children: 0, Fare: 13, Embarked: 2
Survived: 1, Class: 1, Sex: 1, Age: 19, Siblings/Spouses: 0, Parents/Children: 0, Fare: 30, Embarked: 2
Survived: 0, Class: 3, Sex: 1, Age: 28, Siblings/Spouses: 1, Parents/Children: 2, Fare: 23.45, Embarked: 2
Survived: 1, Class: 1, Sex: 0, Age: 26, Siblings/Spouses: 0, Parents/Children: 0, Fare: 30, Embarked: 0
Survived: 0, Class: 3, Sex: 0, Age: 32, Siblings/Spouses: 0, Parents/Children: 0, Fare: 7.75, Embarked: 1

Gini impurity for Best split Age: 6.5
Gini Impurity at age 6.5 is 0.46173
Gini impurity for sex: 0.333365
Gini impurity for passenger class: 0.418391
Gini impurity for Best split Age: 1
```

Thank goodness! Finally finished building the Gini impurity function!

The next step is to start building the decision tree model

Step 4 - Constructing decision tree models

Since the complexity of modeling is more difficult to describe than the construction of Gini impurity. Therefore, I will only briefly describe what each function does and how the model selects them.

These are the two most important parameters of the decision tree model. They control the size and the maximum depth of the tree.

```
const int MIN_SIZE = 2;
const int MAX_DEPTH = 5;
```

Step 4 continued - Constructing decision tree

These three functions are optimization functions for the decision tree algorithm. They are used to solve the problem of too few features. These functions use existing features to generate new features (called Category x).

```
>bool isPure(vector<Passenger>& passengers) { ... }

>int findMostCommonCategory(vector<Passenger>& passengers) { ... }

>void partitionPassengers(vector<Passenger>& passengers, string& bestFeature, double bestSplit, vector<Passenger>& leftSubSet, vector<Passenger>& rightSubSet) { ... }
```

Step 4 continued - Constructing decision tree

In order to store used features, we need to use the set datatype

```
TreeNode* buildDecisionTree(vector<Passenger>& passengers, int depth, set<string>& usedFeatures) {
    if (passengers.empty() || depth > MAX_DEPTH) {
        return nullptr;
    }

    if (passengers.size() < MIN_SIZE || isPure(passengers)) {
        int commonCategory = findMostCommonCategory(passengers);
        return new TreeNode(commonCategory);
    }

    string bestFeature;
    double bestSplit = -1;
    double minGini = 999999999.99;
    bool improved = false;

    if (usedFeatures.find("Sex") == usedFeatures.end()) { ... }

    if (usedFeatures.find("Pclass") == usedFeatures.end()) { ... }

    if (!improved) { ... }

    usedFeatures.insert(bestFeature);

    vector<Passenger> leftSubSet, rightSubSet;
    partitionPassengers(passengers, bestFeature, bestSplit, leftSubSet, rightSubSet);

    TreeNode* node = new TreeNode(bestFeature, bestSplit);
    node->leftChild = buildDecisionTree(leftSubSet, depth + 1, usedFeatures);
    node->rightChild = buildDecisionTree(rightSubSet, depth + 1, usedFeatures);

    return node;
}
```

If the input passenger dataset is empty or has reached its maximum depth (MAX_DEPTH), nullptr is returned, indicating that this node is a leaf node.

If the size of the passenger dataset is smaller than a predefined minimum size (MIN_SIZE) or if the dataset is pure (i.e., all passengers belong to the same category), the most common category is identified and a leaf node representing that category is created.

bestFeature is used to store the name of the best feature, bestSplit stores the best split point for that feature, and minGini stores the minimum Gini index currently found.

Examine each unused feature (e.g., sex and pclass category), compute the Gini index based on those features, and update the best feature and the best segmentation point.

Create the current node using the best features and split points found, and then recursively call the buildDecisionTree function on the left and right subsets to build the left and right subtrees, respectively.

Step 4 continued - Constructing decision tree

By learning about tree structures. I used what I learned and printed the tree by preorder to check the functions we used.

```
void printTreePreorder(TreeNode* node) {
    if (node != nullptr) {
        if (node->category != -1) {
            cout << "Category: " << node->category << " ";
        }
        else {
            cout << node->feature << ": " << node->splitValue << " ";
        }
        printTreePreorder(node->leftChild);
        printTreePreorder(node->rightChild);
    }
}
```

Step 5 - test the model

In testing the model, the first thing we need to test is the performance of the model trained with this dataset on the training dataset. It is a good indicator of whether our training is effective or not. Then we calculate the accuracy. Of course, this accuracy is only the performance in the existing training set. When new data becomes available, data that is not in the dataset, we need to evaluate the accuracy as well. Therefore, there is another dataset that we call the test dataset. In this project, however, because of the tedious process of implementation, I only implemented the evaluation of the training dataset. Of course, there's also the possibility of overfitting and so on. I wouldn't discuss this in this project.

Step 5 continued - test the model

prediction function

```
int predict(TreeNode* node, Passenger& passenger) {
    // Base case: If the node has no children, return its category.
    if (node->leftChild == nullptr && node->rightChild == nullptr) {
        return node->category;
    }

    // Recursive case: Check the feature and decide the next node based on the feature's value.
    if (node->feature == "Sex") {
        if (passenger.sex <= node->splitValue) {
            return predict(node->leftChild, passenger);
        }
        else {
            return predict(node->rightChild, passenger);
        }
    }
    else if (node->feature == "Pclass") {
        if (passenger.pclass == int(node->splitValue)) {
            return predict(node->leftChild, passenger);
        }
        else {
            return predict(node->rightChild, passenger);
        }
    }
    else if (node->feature == "Age") {
        if (passenger.age <= node->splitValue) {
            return predict(node->leftChild, passenger);
        }
        else {
            return predict(node->rightChild, passenger);
        }
    }
    else {
        return -1;  // Return an invalid category or handle error appropriately
    }
}
```

Calculate the accuracy function

```
double calculateAccuracy(vector<Passenger>& passengers, TreeNode* root) {
    double correctPredictions = 0;
    for (int i = 0; i < passengers.size(); ++i) {
        int predicted = predict(root, passengers[i]);
        if (predicted == passengers[i].survived) {
            correctPredictions++;
        }
    }
    double Accuracy = correctPredictions / passengers.size() * 100.0;

    return Accuracy;  // Convert to percentage
}
```

Step 6 - Main function and output results

I created a new file: main.cpp. to perform the main operation.

```cpp
int main() {
    string filename = "F:\processed_train_simple.txt";
    vector<Passenger> passengers = readPassengers(filename);

    printPassengers(passengers);

    double bestSplit = findBestSplit(passengers);
    cout << endl << "Gini impurity for Best split Age: " << bestSplit << endl;

    double giniImpurity = calculateGiniAtSplit(passengers, bestSplit);
    cout << "Gini Impurity at age " << bestSplit << " is " << giniImpurity << endl;

    // Calculate Gini impurity for sex split
    double giniSex = calculateGiniForSex(passengers);
    cout << "Gini impurity for sex: " << giniSex << endl;

    // Calculate Gini impurity for passenger class
    double giniPclass = calculateGiniForPclass(passengers);
    cout << "Gini impurity for passenger class: " << giniPclass << endl;

    double bestSplitPclass = findBestSplitForPclass(passengers);
        cout << "Gini impurity for Best split Age: " << bestSplitPclass << endl;

    set<string> usedFeatures;
    TreeNode* root = buildDecisionTree(passengers, 0, usedFeatures);
    printTreePreorder(root);
    cout << endl;
    double accuracy = calculateAccuracy(passengers, root);
    cout << "Accuracy of the decision tree: " << accuracy << "%" << endl;


    return 0;
}
```

I printed out all the passenger information first, to check if it was read correctly.

Check that each Gini calculation function works

Constructing decision trees

Printing Features by Preorder

Calculate accuracy (training dataset)

Step 6 continued - Main function and output results

Print results

```
Survived: 0, Class: 3, Sex: 0, Age: 19, Siblings/Spouses: 0, Parents/Children: 0, Fare: 7.8958, Embarked: 2
Survived: 0, Class: 3, Sex: 0, Age: 28, Siblings/Spouses: 0, Parents/Children: 0, Fare: 7.8958, Embarked: 2
Survived: 1, Class: 1, Sex: 1, Age: 56, Siblings/Spouses: 0, Parents/Children: 1, Fare: 83.1583, Embarked: 0
Survived: 1, Class: 2, Sex: 1, Age: 25, Siblings/Spouses: 0, Parents/Children: 1, Fare: 26, Embarked: 2
Survived: 0, Class: 3, Sex: 0, Age: 33, Siblings/Spouses: 0, Parents/Children: 0, Fare: 7.8958, Embarked: 2
Survived: 0, Class: 3, Sex: 1, Age: 22, Siblings/Spouses: 0, Parents/Children: 0, Fare: 10.5167, Embarked: 2
Survived: 0, Class: 2, Sex: 0, Age: 28, Siblings/Spouses: 0, Parents/Children: 0, Fare: 10.5, Embarked: 2
Survived: 0, Class: 3, Sex: 0, Age: 25, Siblings/Spouses: 0, Parents/Children: 0, Fare: 7.05, Embarked: 2
Survived: 0, Class: 3, Sex: 1, Age: 39, Siblings/Spouses: 0, Parents/Children: 5, Fare: 29.125, Embarked: 1
Survived: 0, Class: 2, Sex: 0, Age: 27, Siblings/Spouses: 0, Parents/Children: 0, Fare: 13, Embarked: 2
Survived: 1, Class: 1, Sex: 1, Age: 19, Siblings/Spouses: 0, Parents/Children: 0, Fare: 30, Embarked: 2
Survived: 0, Class: 3, Sex: 1, Age: 28, Siblings/Spouses: 1, Parents/Children: 2, Fare: 23.45, Embarked: 2
Survived: 1, Class: 1, Sex: 0, Age: 26, Siblings/Spouses: 0, Parents/Children: 0, Fare: 30, Embarked: 0
Survived: 0, Class: 3, Sex: 0, Age: 32, Siblings/Spouses: 0, Parents/Children: 0, Fare: 7.75, Embarked: 1

Gini impurity for Best split Age: 6.5
Gini Impurity at age 6.5 is 0.46173
Gini impurity for sex: 0.333365
Gini impurity for passenger class: 0.418391
Gini impurity for Best split Age: 1
Sex: 0.5 Pclass: 1 Category: 0 Category: 0 Category: 1
Accuracy of the decision tree: 78.6756%
```

It's about 80 percent accurate, which is pretty good.

That's what this project is all about.

Challenges in the project

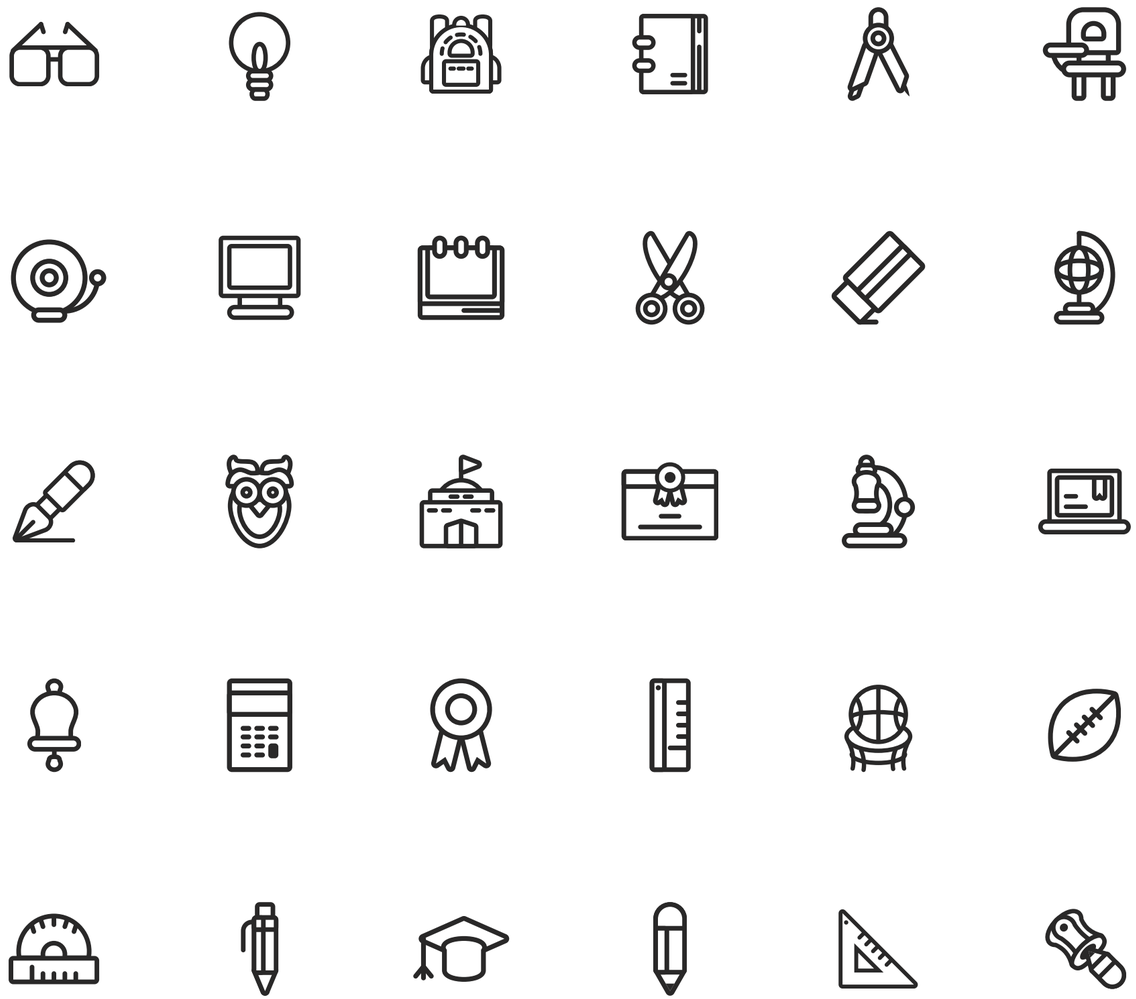
Challenges in the project

My biggest regret is that I was not able to include age as a feature when constructing the decision tree. there is no proper analysis environment in C++. So I couldn't analyze the specifics of age. When I added age as a feature to the decision tree, there were many subsets that did not have data to make predictions based on age, resulting in bugs in the predictions. I've spent a lot of time debugging. but it's still not working. so I had to remove it from the construction of the decision tree.

Also, because the project needs to call the data, sometimes the data may be unreadable. So I also spend a lot of time to deal with and solve these problems

My thoughts

This is a most meaningful program.



While studying machine learning, I never understood how data is correlated and analyzed. As a data science major, I needed to work with data. Therefore, learned data structures helped me a lot. Code that I couldn't understand before is now understandable.

This project is one of my favorite projects I've ever done. If I had implemented it in Python, I wouldn't have needed more than dozens of lines of code because there are rich machine learning libraries to help me analyze the data and build the model. But I never really understood the mechanics behind it. As a result, when I call these functions, I often don't understand them, I don't know how the data are stored, and I don't know how the algorithms work. This project has inspired me a lot. Implementing this project in C++ was a challenge in itself; no one had ever implemented machine learning in C++ before! I am interested in such challenging problems, and that is why I did this project.

This project not only helped me understand the class better, but also made me more determined to choose machine learning as my career path. I am very grateful for this opportunity.

Thanks!