Journal: The fundamental of Decision Tree

Option 1

Vi Nguyen
13592629

Table of Contents

Project link: https://colab.research.google.com/drive/1K8arNUKxYGyKayscMes-QbfJr-qmrmaM#scrollTo=FL3u4t7Kw1j9

1. Introduction

This journal entry documents my journey in exploring the benefits and drawbacks of decision trees, along with their application in a chosen dataset. A comprehensive knowledge of decision trees and the related models is unarguably an essential component to anyone who wants to learn in the field of machine learning and artificial intelligence.

1.1 Model introduction

The decision tree model is a fundamental machine-learning technique that being widely used around the world. Despite the simplicity, users worldwide have continuously utilised and achieved many successes in classification and regression in the supervised learning field.

In the famous book "Artificial Intelligence: A Modern Approach" (Russell and Norvig, 2022), the authors defined the decision tree as a function that matches the input attributes to output values, or the "decisions". The process starts from a root node, following a top-down structure where each level is split into two nodes. The process continues until the final layers, or leaf nodes, are reached. Each of the data is split according to a decision rule, based on the specific parameters. I want to focus on Boolean classification because of the motivation behind this study which I will further illustrate in section 1.2.

To sum up, a model for a decision tree includes the following components:

- **Training set of input and output**: $(x_1,y_1),(x_2,y_2)$,... (x_N, y_N)
- **Decision rule**, which determines the spitting criteria based on the value of the features. The decision rule will have two components:
    1. **Split feature**: Which feature or attributes to divide
    2. **The split threshold**: If the value is greater or less than the threshold, it will be allocated into two different nodes below
- **Stopping criteria**: When to stop growing a tree, whether it is a maximum depth or minimum simple leaf, ...
- **Entropy:** The main components that measure the importance of each attribute. The higher the entropy, the higher the node's location in the decision tree.
- **Impurity:** The lower, the more certain, the better split

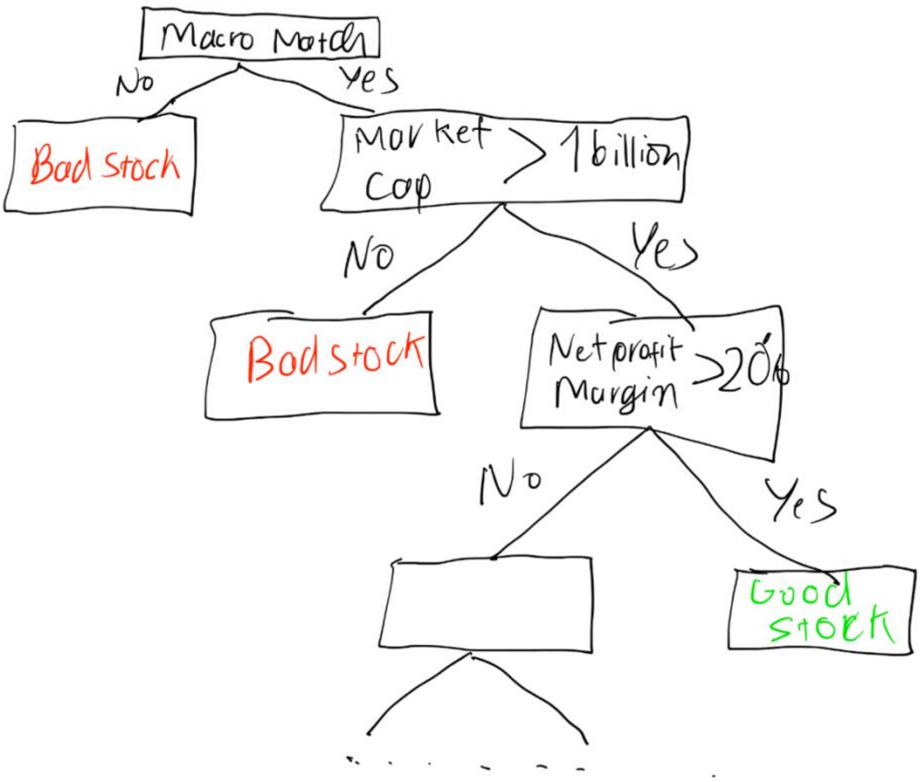
I will talk more about this, giving an example in section 1.2 in the classifying bad stock and good stock

1.2 Motivation for this study

I have always been interested in the decision tree-based model. I have a background in Financial Evaluation and Stock Analysis in the Vietnam stock market, and my main job is basically to classify whether a stock is a good stock or not based on a particular type of criteria such as:

- Is the company suitable for the current macroeconomic and monetary policy, e.g.
    - No -> bad stock
    - Yes -> Is the company's total market capitalisation over 1 billion dollars to provide enough liquidity for our investors?
        - No -> bad stock
        - Yes -> Is the net profit margin above 20%?
            - NO -> Bad stock
            - Yes -> … continue down

Here is my hand-written decision tree with the example above



I usually go with the macroeconomic condition first when I do this type of analysis. Regardless, I do not want to limit my opportunities because my set of criteria can be biased based on my personal experience and skill.

The process of choosing which criteria first, or what percentage of net profit margin,... can take a very substantial time if my set of criteria is more than 100. And I have to do the same thing for 1000 companies in the Vietnamese stock market. The process is challenging and time-consuming.

I see a very similar connection between my job and the process of the decision tree. Therefore, I decided to study deeper into the fundamental theory behind this module. My aim is to understand the construction of the tree in general and which data set is suitable for the model, along with the training process. Furthermore, I want to learn how to evaluate the model and measure the difference between the prediction output by the model and the actual result that I provided myself.

If this works out well, I can apply this module and a similar algorithm to assist my analysis. This could significantly reduce my working time, avoid missing out value and increase the efficiency overall.

Regardless, I do not have good coding skills, so I decided to start off with something straightforward that requires no data preprocessing or preparation. Therefore, I imported the breast_cancer data from sklearn’s datasets.

2. Study Points

There are four different study points in implementing my decision tree, which I will have an overview below. I also take extra focus on the implementation of the training process in 2.3 to enhance the result evaluated in 2.4. The main point I want to study further is in 2.5, where I want to reduce the False Negative prediction of the dataset.

2.1 Dataset introduction

This is the first part of the notebook

With some basic understanding of Python and the help of chatGPT, I have learned a few ways to discover the dataset using df modules, as shown in Day 2 Query 3 (Appendix).

```
import pandas as pd
from sklearn.datasets import load_breast_cancer

# Load the Breast Cancer dataset
cancer = load_breast_cancer()
```

The breast_cancer data contains 30 independent variables representing the 30 characteristics of the human breast. The aim of these variables is to predict whether the patient has breast cancer (1) or not (0). There are 569 rows and 31 attributes, in which the dependent variable is "target".

Data set overview

The function df.head() presented a summary of the data and attributes. The df.describe() function generates the statistical summary like min, max, standard deviation,... The df.types can output the data type of each attribute.

A comprehensive understanding of the data input is compulsory because not all the models can work with different data samples. For instance, a decision tree can not work with text data and requires an extra preprocessing step to convert this text to numerical data.

The breast_cancer dataset is very clean and requires no further exploration, but it is beneficial to acknowledge all the data preprocessing steps required for a smooth data analysis process. The typical examples of these data preprocessing steps are data cleaning, data transforming, encoding for text data or normalisation.

2.2 Data for training

This is the second session of the notebook, where I divide and select the dataset for training. A decision model uses training data to discover relationships and recognise patterns between a set of input attributes and the target variable.

```
X = cancer.data
y = cancer.target
```

I store all 30 features in X and the target variable in y.

```
X = df.drop('target', axis=1)
y = df['target']
```

After separating the input and target variable, the process of splitting the initial dataset into training, testing and validations is also essential. A correct implementation of data splitting can ensure that the model is trustworthy, accurate, generalised well and performs well on new unrecognised datasets. In particular, I have divided the data into three parts:

1. Training dataset (X_train, y_train): This consists of 70% of the dataset
2. Testing dataset (X_test
3. Testing dataset

After this, we split the data into three-part

1. 70% for training ~ 398 rows
2. 20% for testing ~ 114 rows
3. 10% for validating ~ 57 rows

```
X_train, X_test, y_train, y_test = train_test_split(
        X, y,
        test_size=0.3,  #30% of 569 is about 190
        random_state=42)
```

```
# Get the last 63 samples from the test to be the future data samples
futureSample_X = X_test[-63:]
futureSample_y = y_test[-63:]

# Remove the last 63 samples from the test dataset
X_test = X_test[:-63]
y_test = y_test[:-63]
```

2.3 The training process

The training process of the decision tree will be illustrated in Session 3: Decision Tree Classifier Construction of the notebook. There are five foremost steps in constructing the tree are illustrated below

Step 1: The __init__ function is the initialisation step for the decision tree in specifying the parameter

- max_depth=None
- min_samples_split=2
- min_samples_leaf = 1

These specific parameters will control how the tree grows and set up the structure of the tree

```
class DecisionTree:
    def __init__(self, max_depth=None, min_samples_split=2, min_samples_leaf=1):
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.min_samples_leaf = min_samples_leaf

    def fit(self, X, y):
        self.root = self._grow_tree(X, y)
```

Step 2: The fit function initialise the training with all the 30 variables X and label variable y, which is the 'target'

- Inside the fit function function, we initialise the root of the tree by calling the grow_tree function with the input of (X and y)

Step 3: The _grow_tree method will begin to build the tree architecture

```
def  (variable) n_samples: Any h=0):
    n_samples, n_features = X.shape
    n_classes = len(np.unique(y))
    majority_class = np.bincount(y).argmax()
    # Stopping conditions
    if (depth >= self.max_depth) or (n_classes == 1) or (n_samples < self.min_samples_split):
        return Node(class_label=majority_class)
    # Find the best split
    best_gini = 1.0
    best_feature, best_threshold = None, None
    for feature in range(n_features):
        thresholds = np.unique(X[:, feature])
        for threshold in thresholds:
            left_indices = X[:, feature] < threshold
            right_indices = X[:, feature] >= threshold
            if len(y[left_indices]) < self.min_samples_leaf or len(y[right_indices]) < self.min_samples_leaf:
                continue
            gini = self._gini_impurity(y[left_indices], y[right_indices])
            if gini < best_gini:
                best_gini = gini
                best_feature = feature
                best_threshold = threshold

    if best_gini == 1.0:
        return Node(class_label=majority_class)

    left_indices = X[:, best_feature] < best_threshold
    right_indices = X[:, best_feature] >= best_threshold
    left_subtree = self._grow_tree(X[left_indices], y[left_indices], depth + 1)
    right_subtree = self._grow_tree(X[right_indices], y[right_indices], depth + 1)

    return Node(feature=best_feature, threshold=best_threshold, left=left_subtree, right=right_subtree)
```

In this case, n is the number of

- n_samples: data samples
- n_features: features
- n_classes: unique classes in the target variable y.

Majority_class can be used to count the number of times the class label occurs using the np.bincount(y), then find the highest count with argmax() function. As a result, the majority_class stores all the class labels that have the most occurrences for the target variable "target".

Step 3.1 The tree will continue to grow until it meets its stopping condition.

```
# Stopping conditions
if (depth >= self.max_depth) or (n_classes == 1) or (n_samples < self.min_samples_split):
    return Node(class_label=majority_class)
```

In this case, the stopping conditions are max_depth or min_sample_split or n_classes = 1

If any of these conditions is met, the model will create a leaf node using the majority_class label defined above.

Step 3.2: If the stopping conditions in step 3.1 fail:

- The decision tree then finds the optimal split to partition the data
- It then iterates through the feature and threshold for each of the feature
    - For each threshold, it splits the data into left and right branches according to the value of the feature or attributes

```
# Find the best split
best_gini = 1.0
best_feature, best_threshold = None, None
for feature in range(n_features):
    thresholds = np.unique(X[:, feature])
    for threshold in thresholds:
        left_indices = X[:, feature] < threshold
        right_indices = X[:, feature] >= threshold
        if len(y[left_indices]) < self.min_samples_leaf or len(y[right_indices]) < self.min_samples_leaf:
            continue
        gini = self._gini_impurity(y[left_indices], y[right_indices])
        if gini < best_gini:
            best_gini = gini
            best_feature = feature
            best_threshold = threshold

if best_gini == 1.0:
    return Node(class_label=majority_class)

left_indices = X[:, best_feature] < best_threshold
right_indices = X[:, best_feature] >= best_threshold
left_subtree = self._grow_tree(X[left_indices], y[left_indices], depth + 1)
right_subtree = self._grow_tree(X[right_indices], y[right_indices], depth + 1)

return Node(feature=best_feature, threshold=best_threshold, left=left_subtree, right=right_subtree)
```

The Gini impurity is calculated for this split, measuring the impurity or uncertainty of the data. The split with the lowest Gini impurity is considered the best.

The Gini impurity is the measurement of impurity or the uncertainty of the model about the data. The Gini value can be utilised for the split where the lower the impurity of Gini means the model is certain about the split, resulting in better data partitioning. On the other hand, a higher Gini Impurity suggests that the model is uncertain about the classification. Hence, the goal of the decision tree is to minimize the Gini Impurity, which ultimately enhances the accuracy and reliability of the model

```
if gini < best_gini:
    best_gini = gini
    best_feature = feature
    best_threshold = threshold
```

If the gini is less than the current best_gini (initially 1), the decision tree will track the feature and threshold of the best split. The model will then create the current node with this split. Then, create a new node with the best feature and threshold, along with two child nodes.

```
return Node(feature=best_feature, threshold=best_threshold, left=left_subtree, right=right_subtree)
```

Step 3.3: The calculation of Gini impurity

```
def _gini_impurity(self, left, right):
    total_samples = len(left) + len(right)
    p_left = len(left) / total_samples
    p_right = len(right) / total_samples

    gini_left = 1.0 - sum([(np.sum(left == c) / len(left))**2 for c in np.unique(left)])
    gini_right = 1.0 - sum([(np.sum(right == c) / len(right))**2 for c in np.unique(right)])

    gini = p_left * gini_left + p_right * gini_right
    return gini
```

$$\text{Gini Impurity} = 1 - \left(\left(\frac{|L|}{|L+R|}\right) \cdot \text{Gini}(L) + \left(\frac{|R|}{|L+R|}\right) \cdot \text{Gini}(R)\right)$$

Where

|L| is the number of samples in the 'left' subset.

|R| is the number of samples in the 'right' subset.

|L+R| is the total number of samples in both 'left' and 'right' subsets.

Gini(L) is the Gini impurity of the 'left' subset.

Gini(R) is the Gini impurity of the 'right' subset.

On the other hand, entropy is another method for a decision tree to assess the performance of the classification. I would discuss this further in session 2.5

Step 4: Completing the tree

This process of creating a tree continues with more branches and nodes until the stopping conditions specified at the beginning of step 3, or the Gini Impurity can no longer be lowered, then the tree is completed.

```
def predict(self, X):
    return [self._predict_tree(x, self.root) for x in X]

def _predict_tree(self, x, node):
    if node.is_leaf():
        return node.class_label
    if x[node.feature] < node.threshold:
        return self._predict_tree(x, node.left)
    else:
        return self._predict_tree(x, node.right)
```

Step 5: Training Completion

After the completion training process and tree construction, the tree is held in the root attribute. Then we can create an instance of the class called DT_clf (short for decision tree classification) with the parameter input such as below.

```
DT_clf = DecisionTree(max_depth=10, min_samples_split=2, min_samples_leaf=1)
DT_clf.fit(X_train, y_train)
```

Then, we can start the training process using the 30 independent variables of the training dataset, X_train, to predict the label y_train.

2.4 Evaluation method

We can compute the accuracy by dividing the correct classification (y_true == y_pred) by the total number.

```
def accuracy(y_true, y_pred):
        accuracy = np.sum(y_true == y_pred) / len(y_true)
        return accuracy

y_pred = DT_clf.predict(X_test)
outcome = accuracy(y_test, y_pred)

print("Accuracy:", outcome)
```

```
Accuracy: 0.9259259259259259
```

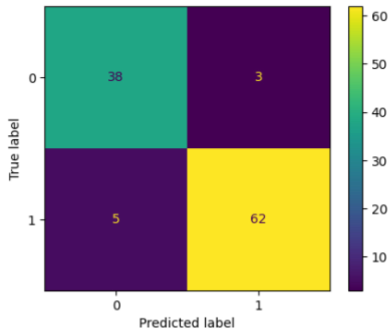
The accuracy is very high, indicating that it is an acceptable model. Regardless, there are other factors to consider in classification problems. This is the case for a dataset that is not balanced, where the occurrence of one instance/class takes up to 90% of the dataset. The accuracy will still be high, but the model can not correctly predict other classes.

In our case, 1 case of the patient being missed diagnosis is much more critical than 100 cases being false positive (or false alarm). The medical analysis will still consider the accuracy score, but the priority will be lower than precision and recall.

The solution to this problem is to employ other metrics for evaluation, such as precision, recall and f1 score. First, we can display a confusion matrix to have an overview of all the correct and wrong predictions.

```
from sklearn import metrics
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix

# Display the confusion matrix using y_test and y_pred
fig,ax = plt.subplots(figsize=(5,4),dpi = 100)
cm = confusion_matrix(y_test, y_pred)
cmp = ConfusionMatrixDisplay(cm, display_labels = ["0", "1"])
cmp.plot(ax = ax);
```



We can also import the model to display the confusion matrix of all 4 scores

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
# Evaluate the classifier's performance

imp_accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {imp_accuracy:.2f}")

classification_rep = classification_report(y_test, y_pred)
print("Classification Report:\n", classification_rep)

conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", conf_matrix)
```

```
Accuracy: 0.93
Classification Report:
               precision    recall  f1-score   support

           0       0.88      0.93      0.90        41
           1       0.95      0.93      0.94        67

    accuracy                           0.93       108
   macro avg       0.92      0.93      0.92       108
weighted avg       0.93      0.93      0.93       108
```

1. Accuracy = (TP + TN) / (TP + TN + FP + FN)
   - The total accuracy of correctly classifying both positive and negative class
2. Precision = TP / (TP + FP)
   - Indicate how many positives predicted by the model actually positive
3. Recall = TP / (TP + FN)
   - Indicate the actual positive was correctly predicted
4. F1 Score = 2 * (Precision * Recall) / (Precision + Recall)
   - The average between precision and recall, which assists in identifying an imbalanced dataset.

	Predicted	
		0 (Positive)	1 (Negative)
Actual	0 (Positive)	TP	FN
	1 (Negative)	FP	TN

Hence, higher recall is preferred because of the case where cancer is true, but the model fails to detect it. Enhancing the true positive, hence reducing the false negative in this case, is much more important than the overall accuracy. The recall score of 93% is extremely high, which is over-satisfied.

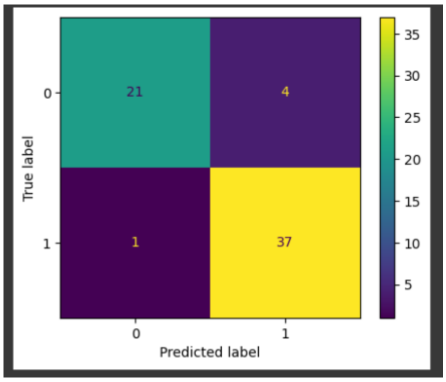
Repeat the same step but for the validation set

```
Accuracy: 0.92
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.84      0.89        25
           1       0.90      0.97      0.94        38

    accuracy                           0.92        63
   macro avg       0.93      0.91      0.92        63
weighted avg       0.92      0.92      0.92        63

Confusion Matrix:
 [[21  4]
 [ 1 37]]
```



Even though the accuracy did not significantly reduce for the validation test set, the recall score significantly fluctuates in predicting classes 0 and 1. However, we only care about the classification of class 1, which means the model has correctly identified 97% of the true positive cases. 97% of the predicted being labelled “have breast cancer” is actually having breast cancer. Only 3% of the patients actually had breast cancer but was not detected, or False Negative. This is a very good model.

2.5 Further exploration

During my studies, I am motivated to study more about the applicaiton of Decision Trees in the medical field, particularly in diagnosis. Moreover, I want to investigate the factors that can be improved in order to enhance the recall score of the Decision Tree model. In other words, I want to minimise the number of false negative predictions. I decided to explore this field deeper. Lastly, I also want to investigate the Entropy as a splitting criteria, and if it differs greatly from Gini Impurity.

One of the first applications of the Decision Tree in a medical context
One of the earliest papers I could find was published in 1976, discussing using decision trees in the clinical context (Kassirer,1976). This concept was called decision analysis, but the basic structure is quite similar to a decision tree. Rather than the doctor utilising medical protocol to make medical decisions, decision analysis will construct a decision tree to summarise the list of choices, the possible outcomes, along with the associated probabilities and the utilities for each outcome. The utilisation of this decision model provides the doctor with a range of actions available based on different circumstances and accelerates the speed and accuracy of the diagnosis process.

Introduction of Random Forest
In 2001, Leo Breiman first introduced the concept of a random forest built based on combining multiple trees, which led to a significant improvement in accuracy while overcoming overfitting. This usually happens in training a decision tree. Overfitting can lead to poor generalisation, hence lower recall and accuracy when it predicting the wrong label. Moreover, the construction of a tree is entirely random, allowing the capture of a broader range of data hence enhancing the recall score. Random Forest is also well-known for its ability to work with noisy data. The different trees are combined and cancelled out the extreme cases, resulting in higher accuracy and recall.

Feature selection
A valid suggestion to reduce the false negative from ChatGPT is to perform a manual selection of the feature /attributes input before training the decision tree. Regardless, the biggest constraint of this method is the need for industry experts. While various methods exist to rank features, such as feature importance, consulting an expert will ensure a more precise selection of features.

Zhou et al. (2021) performed this preprocessing step on the Feature-Weight Decision Tree (FWDT). The feature weight is a calculator based on the improvement it makes to the model based on classification accuracy. Zhou and his colleagues concluded that FWDT model reduced the computing time, but it also enhanced the recall score on average from 35% to 38.09% (table 6, section 4.3.2).

Splitting condition: Information entropy vs Gini Impurity
Juneja et al. (2022) had an interesting review of the applicaiton of the C4.5 algorithm, discussing the limitations and suggestions for improving this algorithm. This algorithm was developed over a decade ago by Ross Quinlan, focusing on splitting the features using information entropy instead of Gini Impurity in my study. From this study, I understand that information entropy is a better choice for enhancing the recall score as it works better with imbalanced datasets. The Gini Impurity is more suitable for constructing a simple decision tree to deal with a larger dataset, hence providing a better generalisation that avoids overfitting.

Abdullahet al. (2021) also have a research paper on detecting the Case Fatality Rate of Dengue Hemorrhagic Fever (DHF) in Indonesia using the C4.5 algorithm. The classification of the study focuses on the severity of the patient, helping the doctors to recognise the individuals with higher risk quickly. The model can work as an early intervention by identifying the signal associated with the disease, allowing the patient to take further medication procedures. Even though this is not a direct approach from the decision tree to detect false negatives, it can still reduce the risk of misidentifying the patient with DHF

3. Reflection

For my implementation of the study, I have encountered a few problems. First, I want to use real-world finance/stock market data and use the decision tree to train the data. Regardless, I have spent 3-4 days trying to preprocess and clean the data. The model never works with too many errors sitting on top of each other. Therefore, I choose to do option 1 to study the mechanism behind the decision tree to apply it better in the future.

Secondly, I rarely work with dataframe structure, and my Python skill is not very strong. It took me a lot of time to do research (not only in ChatGPT) about the benefits and application of df. Regardless, chatGPT has helped me a lot with initiating this problem (Appendix, Day 2 Query 3). After this subject, I found this language and the dataframe concept extremely useful for my field. I think I would consider spending more time researching this topic and programming language.

The exciting thing I found during my study is the computation of the node Gini Impurity to split the node. I also find it interesting to use entropy as a criterion to split the tree. The computation of Gini Impurity requires a detailed statistical summary of the data set. For instance, the calculation required the probabilities of each distinct class, which does not work very well with a small dataset. However, the financial datasets like the problem I want to solve will overcome this problem easily.

One of the most challenging experiences for me is to understand the evaluation metrics with all the accuracy, recall or precision metrics. Especially the true, false, negative and positive calculations. I have to do extra research and watch many tutorials online to comprehend these metrics thoroughly. Regardless, this challenge turned out to be my most exciting topic, and I have decided to investigate deeper.

4. Reference

Abdullah, A. Z., Winarno, B., & Saputro, D. R. S. (2021). The decision tree classification with C4.5 and C5.0 algorithm based on R to detect case fatality rate of dengue hemorrhagic fever in Indonesia. Journal of Physics. Conference Series, 1776(1), 12040. https://doi.org/10.1088/1742-6596/1776/1/012040

Breiman, L. (2001). Random Forests. University of California. https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf

Juneja D, Sharma S, Jain A, Sharma S. (2022) A Novel Approach to Construct Decision Tree Using Quick C4.5 Algorithm. Orient. J. Comp. Sci. and Technol;3(2). http://www.computerscijournal.org/?p=2318

Kassirer, J. P. (1976). The principles of clinical decision making: an introduction to decision analysis. The Yale Journal of Biology & Medicine, 49(2), 149–164.
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2595272/pdf/yjbm00143-0047.pdf

Russell, S. J., & Norvig, P. (2022). Artificial intelligence : a modern approach (Fourth edition). Pearson.

Zhou, H., Zhang, J., Zhou, Y., Guo, X., & Ma, Y. (2021). A feature selection algorithm of decision tree based on feature weight. Expert Systems with Applications, 164, 113842. https://doi.org/10.1016/j.eswa.2020.113842

Appendix A

Note: Day 1 - 1 October

Day 1, Query 1: Can you give me an overview of the model for decision tree?

Vi

Can you give me an overview of the model for decision tree

Certainly! A decision tree is a fundamental machine learning model used for both classification and regression tasks. It's a simple and interpretable model that makes predictions by recursively splitting the dataset into subsets based on the values of input features. Here's an overview of how a decision tree model works:

1. **Node**:
   - A decision tree consists of nodes, which represent various decision points in the data.
   - The top node is called the "root node," and it represents the entire dataset.
2. **Splitting**:
   - At each node, the decision tree algorithm selects a feature and a threshold (for continuous features) or a set of categories (for categorical features) to split the data into subsets.
   - The goal is to create splits that maximize the purity or reduce the impurity of the data in the subsets.
   - Different impurity measures (e.g., entropy, Gini impurity, misclassification error) can be used to assess the quality of splits.
3. **Child Nodes**:
   - After a split, new child nodes are created, one for each possible outcome of the split.
   - The process of selecting the best feature and threshold (or categories) for the split is repeated independently for each child node.
4. **Leaf Nodes**:
   - The process of splitting continues until a stopping condition is met, such as a maximum depth, a minimum number of samples in a node, or when further splits don't significantly improve the impurity.
   - The final nodes in the tree, called "leaf nodes" or "terminal nodes," are used for making predictions.
5. **Prediction**:
   - To make a prediction, you start at the root node and follow the decision path down the tree, based on the values of the input features.
   - When you reach a leaf node, the prediction is made. For classification, the leaf node corresponds to the predicted class. For regression, it might be a predicted numerical value.

Day 1, Query 2: What is the difference between Entropy and Impurity?

Vi What is the difference between Entropu and Impurity

Entropy and impurity are both concepts used in the context of decision trees, particularly in the process of choosing the best split for a node in the tree. While they are related, they measure slightly different aspects of the data and have different formulas and interpretations.

1. **Entropy**:
   - Entropy is a measure of the disorder or randomness in a set of data. In the context of decision trees, it's used to measure the impurity of a node.
   - The entropy of a node is calculated using the formula:
     Entropy(S) = -p_1 * log2(p_1) - p_2 * log2(p_2) - ... - p_k * log2(p_k)
     where p_1, p_2, ..., p_k represent the proportions of each class in the node.
   - Entropy is at its maximum when all classes in the node are equally distributed (maximum disorder), and it's at its minimum (0) when all instances in the node belong to the same class (no disorder).
2. **Impurity**:
   - Impurity, in the context of decision trees, is a broader term that encompasses various measures, with entropy being one of them. Other common impurity measures include Gini impurity and misclassification error.
   - Gini impurity, for example, measures the probability of misclassifying a randomly chosen element if it was randomly classified according to the distribution of classes in the node.
   - Misclassification error directly calculates the proportion of the instances in a node that do not belong to the most frequent class.

In summary, while entropy is a specific measure of impurity based on the concept of entropy from information theory, impurity is a more general term used to describe the lack of purity or homogeneity in a node in a decision tree. Decision tree algorithms typically allow you to choose the impurity measure that best suits your problem, and the choice of measure can affect the way the tree is constructed and the splits are determined. Different impurity measures might be preferred depending on the specific characteristics of the dataset and the goals of the classification problem.

Day 2, query 1: Can you guide me a step by step to import data set and evaluate the decision tree

Certainly! Here's a step-by-step guide on how to import a dataset and evaluate a decision tree model in Python using popular libraries like scikit-learn:

1. **Import Required Libraries**:

python Copy code

```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusio
```

2. **Load Your Dataset**:
   - Make sure you have your dataset in a suitable format, such as a CSV file.
   - Use pandas to load the dataset:

python Copy code

```python
data = pd.read_csv('your_dataset.csv')
```

3. **Data Preprocessing**:
   - Check and clean your dataset as needed. This may involve handling missing values, encoding categorical variables, or feature scaling.
4. **Split the Data into Training and Testing Sets**:
   - Split your dataset into a training set and a testing set to evaluate the model's performance.

python Copy code

```python
X = data.drop('target_column', axis=1)  # Features
y = data['target_column']  # Target variable

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra
```

5. **Create and Train the Decision Tree Model**:
   - Initialize the DecisionTreeClassifier and fit it to your training data.

python Copy code

```python
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
```

6. **Make Predictions**:
   - Use the trained model to make predictions on the test data.

python Copy code

```python
y_pred = clf.predict(X_test)
```

7. **Evaluate the Model**:
   - Calculate the accuracy and other performance metrics to evaluate the model.

python Copy code

```python
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# Print classification report for more detailed evaluation
print(classification_report(y_test, y_pred))

# Display the confusion matrix
cm = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(cm)
```

8. **Visualize the Decision Tree (Optional)**:
   - You can visualize the decision tree to better understand its structure and decision-making process.

python Copy code

```python
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plot_tree(clf, filled=True, feature_names=X.columns)
plt.show()
```

Remember to replace `'your_dataset.csv'` and `'target_column'` with the actual dataset filename and the name of the target variable in your dataset. This guide provides a basic framework for working with decision trees, and you can further fine-tune the model and evaluation based on your specific dataset and problem.

Day 2 Query 2: Can you suggest me any built in data set for decision tree binary classification

Vi

Can you suggest me any built in data set for decision tree binary classification

Certainly! scikit-learn, a popular machine learning library in Python, provides several built-in datasets that are suitable for binary classification tasks. Here are a few examples:

1. **Iris Dataset**:
   - While the Iris dataset is commonly used for multiclass classification, it can also be used for binary classification by considering just two of the three classes. For example, you can differentiate between Setosa (class 0) and Versicolor (class 1).

python Copy code

```python
from sklearn.datasets import load_iris
iris = load_iris()
X, y = iris.data, iris.target
X_binary = X[y != 2]
y_binary = y[y != 2]
```

2. **Breast Cancer Wisconsin (Diagnostic) Dataset**:
   - This dataset contains features computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. The task is to predict whether a breast mass is malignant (1) or benign (0).

python Copy code

```python
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
X, y = data.data, data.target
```

3. **Diabetes Dataset**:
   - The diabetes dataset is used for predicting the progression of diabetes one year after baseline. The target variable is a quantitative measure of disease progression, but you can turn it into a binary classification task by defining a threshold for progression.

python Copy code

```python
from sklearn.datasets import load_diabetes
data = load_diabetes()
X, y = data.data, (data.target > 140).astype(int)  # Example threshold: 140
```

4. **Wine Dataset**:
   - The Wine dataset is typically used for multiclass classification, but you can convert it into a binary classification problem by selecting two of the three classes.

python Copy code

```python
from sklearn.datasets import load_wine
wine = load_wine()
X, y = wine.data, wine.target
X_binary = X[y != 0]
y_binary = (y[y != 0] == 1).astype(int)
```

These datasets are readily available in scikit-learn, and you can use them to practice binary classification with decision trees or other machine learning algorithms. You can load them using the code snippets provided above.

Day 2 Query 3: Can you suggest to me a few ways to explore the dataset and the input in df for Python

Vi

Can you suggest me a few ways to explore the dataset and the input in df for python ?

Certainly! Exploring your dataset and understanding the input data (often stored in a Pandas DataFrame in Python) is a crucial step in any data analysis or machine learning project. Here are some common ways to explore the dataset and its inputs using Python and Pandas:

1. **View the First Few Rows:**
   - Use `df.head()` to display the first few rows of the DataFrame. This gives you a quick overview of the data's structure and the format of the input features.
2. **Summary Statistics:**
   - Utilize `df.describe()` to generate summary statistics for numerical columns, including mean, standard deviation, minimum, maximum, and quartiles. This helps you understand the distribution of your data.
3. **Data Types and Missing Values:**
   - Check data types of columns using `df.dtypes`. Ensure that the data types match your expectations.
   - Use `df.isnull().sum()` to identify missing values in each column. Decide how to handle or impute these missing values.
4. **Unique Values and Categories:**
   - Explore unique values in categorical columns with `df['column_name'].unique()`.
   - Count the number of occurrences of each unique value using `df['column_name'].value_counts()`.
5. **Data Distribution Plots:**
   - Create histograms, box plots, and density plots to visualize the distribution of numerical features. You can use libraries like Matplotlib or Seaborn for this.
6. **Correlation Analysis:**
   - Calculate and visualize the correlation between numerical features using methods like `df.corr()` and correlation matrices. This helps identify relationships between features.

Regenerate

8. **Cross-Tabulation and Pivot Tables:**
    - For categorical data, create cross-tabulations to understand relationships between features using `pd.crosstab()` or pivot tables using `df.pivot_table()`.
9. **Grouping and Aggregation:**
    - Use `df.groupby()` to group data by specific columns and calculate statistics or aggregations for each group.
10. **Feature Engineering:**
    - Create new features based on existing ones if needed. This can involve combining or transforming input features to make them more informative.
11. **Data Visualization:**
    - Visualize your data using various plots like scatter plots, bar charts, heatmaps, and more. Libraries like Matplotlib and Seaborn are valuable for this.
12. **Domain-Specific Analysis:**
    - Depending on the problem domain, perform specific data analyses or domain-related explorations to gain insights into the dataset.
13. **Outliers Detection:**
    - Identify and investigate potential outliers in your data. Box plots and scatter plots are useful for this purpose.
14. **Dimensionality Reduction:**
    - Apply dimensionality reduction techniques like PCA or t-SNE for visualizing high-dimensional input data in a lower-dimensional space.
15. **Data Preprocessing:**
    - If needed, perform data preprocessing tasks such as scaling, encoding categorical variables, and handling outliers.
16. **Time Series Analysis (if applicable):**
    - For time-series data, explore time-dependent trends, seasonality, and other patterns.
17. **Feature Importance (for machine learning):**
    - If you're building a machine learning model, analyze feature importance using algorithms like Random Forest or Gradient Boosting to understand the relevance of input features.

Regenera

Day 2 Query 4: What is the suitable portion for training, testing and validating ?

The correct portion for splitting your dataset into training, testing, and validation sets can vary depending on the size of your dataset, the nature of your problem, and your specific goals. However, a commonly used and recommended split is the following:

1. **Training Set**:
   - Typically, the majority of your data, around 60% to 80%, is allocated to the training set.
   - The training set is used to train your machine learning model. It's the portion of the data on which the model learns the underlying patterns and relationships.
2. **Testing Set**:
   - A smaller portion, often around 20% to 30% of the data, is allocated to the testing set.
   - The testing set is used to evaluate the model's performance after it has been trained. It helps you assess how well the model generalizes to new, unseen data.
3. **Validation Set**:
   - In addition to the training and testing sets, a separate validation set is sometimes used, especially when you need to fine-tune hyperparameters or make decisions about model selection.
   - The validation set is typically 10% to 20% of the data.
   - It is used to tune model hyperparameters (e.g., the learning rate, the depth of a decision tree) and assess the model's performance during development.
   - The validation set should not be used for training; it's strictly for model tuning and evaluation.

It's essential to maintain a clear separation between these sets to ensure that your evaluation accurately reflects how well your model will perform on new, unseen data. Cross-validation techniques, such as k-fold cross-validation, can also be used when dealing with limited data to obtain more robust estimates of model performance.

Remember that these split ratios are general guidelines and can be adjusted based on factors like the size of your dataset and the complexity of your model. Smaller datasets may require larger validation and testing sets, while larger datasets can still provide good training and testing with the suggested ratios. Additionally, the specific problem you're working on may also influence the split ratios. Always consider the characteristics of your data and the goals of your machine learning project when determining the appropriate split portions.

Day 2 Query 5: Can I extract the last 100 row , move into a new data set and deleted that 100 rows from the old data set for the purpose of splitting test set and validation set

Certainly, you can extract the last 100 rows from your dataset to create a validation set and keep the remaining data as your training set. This is a common practice when splitting data for machine learning purposes. Here's how you can do it using Python and Pandas:

Assuming you have your data loaded into a Pandas DataFrame (`df`):

python

Copy code

```python
import pandas as pd

# Assuming your data is already loaded into a DataFrame called 'df'
# Extract the last 100 rows for the validation set
validation_set = df.tail(100)

# Update the original DataFrame to exclude the last 100 rows (your training
df = df.iloc[:-100]

# Now, 'df' contains your training set, and 'validation_set' contains your v
```

In this code:

1. We use the `tail(100)` method to extract the last 100 rows of the DataFrame and assign them to the `validation_set` DataFrame.
2. We use the `iloc[:-100]` to remove the last 100 rows from the original DataFrame, effectively creating the training set.

Now, you have two separate DataFrames: `df` contains your training data, and `validation_set` contains your validation data. You can use these sets for training and evaluating machine learning models.

Day 3 Query 1: Can I have a decision tree from scratch please ?

python Copy code

```python
import numpy as np

class DecisionTree:
    def __init__(self, max_depth=None, min_samples_split=2, min_samples_leaf
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.min_samples_leaf = min_samples_leaf

    def fit(self, X, y):
        self.root = self._grow_tree(X, y)

    def _grow_tree(self, X, y, depth=0):
        n_samples, n_features = X.shape
        n_classes = len(np.unique(y))
        majority_class = np.bincount(y).argmax()

        # Stopping conditions
        if (depth >= self.max_depth) or (n_classes == 1) or (n_samples < sel
            return Node(class_label=majority_class)

        # Find the best split
        best_gini = 1.0
        best_feature, best_threshold = None, None
        for feature in range(n_features):
            thresholds = np.unique(X[:, feature])
            for threshold in thresholds:
                left_indices = X[:, feature] < threshold
                right_indices = X[:, feature] >= threshold
                if len(y[left_indices]) < self.min_samples_leaf or len(y[rig
                    continue
                gini = self._gini_impurity(y[left_indices], y[right_indices]
                if gini < best_gini:
                    best_gini = gini
                    best_feature = feature
                    best_threshold = threshold

        if best_gini == 1.0:
            return Node(class_label=majority_class)

        left_indices = X[:, best_feature] < best_threshold
        right_indices = X[:, best_feature] >= best_threshold
        left_subtree = self._grow_tree(X[left_indices], y[left_indices], dep
        right_subtree = self._grow_tree(X[right_indices], y[right_indices],

        return Node(feature=best_feature, threshold=best_threshold, left=lef

    def predict(self, X):
        return [self._predict_tree(x, self.root) for x in X]

    def _predict_tree(self, x, node):
        if node.is_leaf():
            return node.class_label
        if x[node.feature] < node.threshold:
            return self._predict_tree(x, node.left)
        else:
            return self._predict_tree(x, node.right)

    def _gini_impurity(self, left, right):
        total_samples = len(left) + len(right)
        p_left = len(left) / total_samples
        p_right = len(right) / total_samples

        gini_left = 1.0 - sum([(np.sum(left == c) / len(left))**2 for c in r
        gini_right = 1.0 - sum([(np.sum(right == c) / len(right))**2 for c i

        gini = p_left * gini_left + p_right * gini_right
        return gini

class Node:
    def __init__(self, feature=None, threshold=None, class_label=None, left=
        self.feature = feature
        self.threshold = threshold
        self.class_label = class_label
        self.left = left
        self.right = right

    def is_leaf(self):
        return self.class_label is not None
```

Regen

Day 3 Query 2: Can you help me to explain each section please?

python Copy code

```python
import numpy as np

class DecisionTree:
    def __init__(self, max_depth=None, min_samples_split=2, min_samples_leaf
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.min_samples_leaf = min_samples_leaf

    def fit(self, X, y):
        self.root = self._grow_tree(X, y)

    def _grow_tree(self, X, y, depth=0):
        n_samples, n_features = X.shape
        n_classes = len(np.unique(y))
        majority_class = np.bincount(y).argmax()
```

- The code starts with importing the `numpy` library and defining a class called `DecisionTree`, representing the decision tree classifier.
- In the constructor (`__init__`), you can specify parameters such as `max_depth` (maximum depth of the tree), `min_samples_split` (minimum number of samples required to split a node), and `min_samples_leaf` (minimum number of samples required in a leaf node).
- The `fit` method is used to fit the decision tree to the training data `X` and corresponding target values `y`.
- The `_grow_tree` method is a recursive function that's responsible for growing the decision tree. It takes the training data, target values, and the current depth as inputs.

python Copy code

```python
        # Stopping conditions
        if (depth >= self.max_depth) or (n_classes == 1) or (n_samples < sel
            return Node(class_label=majority_class)
```

- These are the stopping conditions for growing the tree. If the depth exceeds the `max_depth`, if all samples in the node belong to the same class, or if the number of samples in the node is less than the `min_samples_split`, a leaf node with the majority class label is created.

python Copy code

```python
        # Find the best split
        best_gini = 1.0
        best_feature, best_threshold = None, None
        for feature in range(n_features):
            thresholds = np.unique(X[:, feature])
            for threshold in thresholds:
                # Code for finding the best split based on Gini impurity is
```

- In this section, the code is responsible for finding the best split for the current node. It iterates over features and thresholds to calculate the Gini impurity and find the best split based on the feature and threshold that minimize the impurity.

python

```python
        if best_gini == 1.0:
            return Node(class_label=majority_class)
```

- If the best Gini impurity is 1.0, it means that the node cannot be split further, and a leaf node with the majority class label is created.

python

```python
        # Code for creating left and right subtrees based on the best split
```

- This section creates the left and right subtrees based on the best split found in the previous section.

python

```python
    def predict(self, X):
        return [self._predict_tree(x, self.root) for x in X]
```

- The `predict` method is used to predict class labels for a list of input samples `X`. It calls the `_predict_tree` method for each input sample.

python

```python
    def _predict_tree(self, x, node):
        if node.is_leaf():
            return node.class_label
        if x[node.feature] < node.threshold:
            return self._predict_tree(x, node.left)
        else:
            return self._predict_tree(x, node.right)
```

- The `_predict_tree` method recursively traverses the decision tree to make predictions for an input sample `x`.

python

```python
    def _gini_impurity(self, left, right):
        total_samples = len(left) + len(right)
        p_left = len(left) / total_samples
        p_right = len(right) / total_samples

        gini_left = 1.0 - sum([(np.sum(left == c) / len(left))**2 for c in r
        gini_right = 1.0 - sum([(np.sum(right == c) / len(right))**2 for c i

        gini = p_left * gini_left + p_right * gini_right
        return gini

class Node:
    def __init__(self, feature=None, threshold=None, class_label=None, left=
        self.feature = feature
        self.threshold = threshold
        self.class_label = class_label
        self.left = left
        self.right = right

    def is_leaf(self):
        return self.class_label is not None
```

- The `Node` class represents a node in the decision tree. It can be a split node or a leaf node.
- The `is_leaf` method checks if the node is a leaf node based on the presence of a class label.

This code provides a simplified implementation of a decision tree classifier. For practical use, you may want to consider various optimizations, advanced splitting criteria (e.g., entropy, information gain), and handling various data types. Real-world implementations of decision trees, such as those found in scikit-learn, provide more advanced and optimized solutions.

Day 3 Query 3: Why do i have no computation of entropy in this code ? My teacher mentioned about this component in class

I apologize for any confusion. The code provided in the previous response uses the Gini impurity as the criterion for splitting the decision tree, not entropy. Gini impurity and entropy are two common criteria used to measure the impurity or disorder in a dataset for decision tree splits.

If you'd like to implement entropy as the criterion instead, you can make the following modifications to the code:

1. Modify the `_grow_tree` method to calculate entropy instead of Gini impurity. You'll need to define a new method for calculating entropy. Here's an example of an entropy calculation method:

python Copy code

```python
def _entropy(y):
    _, counts = np.unique(y, return_counts=True)
    probabilities = counts / len(y)
    entropy = -np.sum(probabilities * np.log2(probabilities))
    return entropy
```

1. Update the `_grow_tree` method to use entropy as the impurity measure:

python Copy code

```python
def _grow_tree(self, X, y, depth=0):
    # ...
    # Replace Gini impurity calculation with entropy calculation

    if best_entropy == 0.0:
        return Node(class_label=majority_class)

    # ...
```

1. Make sure to update the splitting criteria and decision-making based on entropy, similar to how it was done with Gini impurity in the original code.

Keep in mind that implementing entropy as the impurity measure involves a few changes throughout the code. Make sure to adapt all relevant parts to use entropy as the criterion.

Day 4 Query 1: My teacher allowed me to import an external package to illustrate the confusion matrix. Please help me to give some recommendations to visually display it

python Copy code

```python
# Import necessary libraries from scikit-learn for evaluating the model
from sklearn import metrics
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix

# Create a figure and axis for displaying the confusion matrix
fig, ax = plt.subplots(figsize=(5, 4), dpi=100)

# Calculate the confusion matrix using true labels (y_test) and predicted la
cm = confusion_matrix(y_test, y_pred)

# Create a ConfusionMatrixDisplay object, providing the confusion matrix anc
cmp = ConfusionMatrixDisplay(cm, display_labels=["0", "1"])

# Plot the confusion matrix on the specified axis
cmp.plot(ax=ax)
```

Explanation:

1. The code starts with importing the necessary libraries from scikit-learn. This includes `metrics` for general metrics and `ConfusionMatrixDisplay` and `confusion_matrix` for working with the confusion matrix.
2. A figure and axis are created to prepare the space for displaying the confusion matrix. The `figsize` parameter sets the size of the figure, and `dpi` controls the dots per inch for resolution.
3. The `confusion_matrix` function is used to calculate the confusion matrix based on the true labels (`y_test`) and the predicted labels (`y_pred`).
4. A `ConfusionMatrixDisplay` object (`cmp`) is created. It takes the confusion matrix (`cm`) as well as the display labels for classes, in this case, "0" and "1." The display labels help annotate the confusion matrix.
5. Finally, the `plot` method of the `ConfusionMatrixDisplay` object is used to visualize the confusion matrix on the specified axis (`ax`). This creates a visual representation of the true positives, true negatives, false positives, and false negatives in the classification results.

The code provides a clear and concise way to visualize the performance of a binary classification model through the confusion matrix. The resulting plot can help you assess the model's accuracy and its ability to correctly classify instances into their respective classes.

Day 4 Query 2: Can you help me to do an evaluation of a classifier's performance using various metrics

Certainly! Evaluating a classifier's performance involves using various metrics to assess how well the model is doing. The choice of metrics depends on the specific problem you're addressing (e.g., binary classification, multi-class classification, regression). Here, I'll provide a general framework for evaluating a classifier's performance using common metrics:

1. **Confusion Matrix**: The confusion matrix provides a summary of the classifier's predictions. It helps you understand the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN).

python Copy code

```python
from sklearn.metrics import confusion_matrix

# Assuming you have y_true (true labels) and y_pred (predicted labels)
cm = confusion_matrix(y_true, y_pred)
print("Confusion Matrix:\n", cm)
```

2. **Accuracy**: Accuracy measures the overall correctness of the classifier's predictions. It's the ratio of correctly predicted instances to the total number of instances.

python Copy code

```python
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(y_true, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

3. **Precision, Recall, and F1-Score**: Precision, recall, and the F1-score are essential for binary and multi-class classification tasks. They are especially useful when class imbalance is present.

python Copy code

```python
from sklearn.metrics import precision_score, recall_score, f1_score

precision = precision_score(y_true, y_pred)
recall = recall_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred)

print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1:.2f}")
```

4. **Classification Report**: The classification report provides a summary of precision, recall, and F1-score for each class (in the case of multi-class classification). It also includes the support, which is the number of occurrences of each class.

python Copy code

```python
from sklearn.metrics import classification_report

classification_rep = classification_report(y_true, y_pred)
print("Classification Report:\n", classification_rep)
```

Day 4 Query 3: Can you help me to do the computation of accuracy, precision, recall and f1

Vi

the computation of accuracy, precision recall and f1

Accuracy, precision, recall, and F1 score are commonly used metrics in machine learning and statistics to evaluate the performance of classification models. They are used to assess how well a model classifies instances into different classes. Here's how you calculate each of these metrics:

1. Accuracy:
   Accuracy is a measure of the overall correctness of a model's predictions. It is calculated as the ratio of correctly predicted instances to the total number of instances in the dataset.
   Accuracy = (TP + TN) / (TP + TN + FP + FN)
   - TP (True Positives) are the number of correctly predicted positive instances.
   - TN (True Negatives) are the number of correctly predicted negative instances.
   - FP (False Positives) are the number of negative instances incorrectly classified as positive.
   - FN (False Negatives) are the number of positive instances incorrectly classified as negative.
2. Precision:
   Precision measures how many of the instances predicted as positive are actually positive. It is the ratio of true positives to the total number of instances predicted as positive.
   Precision = TP / (TP + FP)
3. Recall (Sensitivity or True Positive Rate):
   Recall measures how many of the actual positive instances were correctly predicted as positive by the model. It is the ratio of true positives to the total number of actual positive instances.
   Recall = TP / (TP + FN)
4. F1 Score:
   The F1 score is the harmonic mean of precision and recall. It combines both precision and recall into a single metric, which can be useful when you want to balance the trade-off between false positives and false negatives.
   F1 Score = 2 * (Precision * Recall) / (Precision + Recall)

The F1 score is particularly useful when there is an imbalance between the classes in your dataset. It gives you a single metric that considers both false positives and false negatives.

These metrics provide valuable insights into the performance of a classification model. Depending on your specific problem and priorities, you may want to focus on accuracy, precision, recall, or F1 score to assess how well your model is doing.

Day 4 Query 4 : for medical purpose, which score is the most important

In medical applications, the importance of evaluation metrics can vary depending on the specific goals of the model and the nature of the medical problem. Here are some common evaluation metrics used in medical contexts and their significance:

1. **Sensitivity (True Positive Rate or Recall)**: Sensitivity measures the ability of a model to correctly identify positive cases (e.g., detecting a disease). It is crucial in medical diagnosis because it focuses on minimizing false negatives. In medical scenarios, missing a true positive (i.e., failing to diagnose a disease when it's present) can have serious consequences.
2. **Specificity (True Negative Rate)**: Specificity measures the ability of a model to correctly identify negative cases (e.g., correctly identifying a healthy individual). It is essential to reduce false positives, which can lead to unnecessary follow-up tests or treatments.
3. **Accuracy**: Accuracy is the overall correctness of the model's predictions. While accuracy is a commonly used metric, it may not be the most important in medical applications, especially when class imbalances exist. A highly accurate model might still perform poorly if it has high false negatives or false positives.
4. **Precision**: Precision focuses on the proportion of true positive predictions among all positive predictions. In medical applications, precision is important when false positives can lead to unnecessary treatments, and it's essential to minimize such occurrences.
5. **F1-Score**: The F1-score is the harmonic mean of precision and recall. It provides a balanced evaluation of the model's performance by considering both false positives and false negatives. A high F1-score indicates a good balance between precision and recall.
6. **Area Under the ROC Curve (AUC-ROC)**: AUC-ROC measures the ability of the model to distinguish between the two classes (e.g., disease vs. no disease). It is particularly relevant when you need to assess the classifier's performance across various probability thresholds.
7. **Positive Predictive Value (PPV)**: PPV is another term for precision. It quantifies the probability that a positive prediction is correct.
8. **Negative Predictive Value (NPV)**: NPV quantifies the probability that a negative prediction is correct.

The choice of the most important metric in a medical context depends on the specific goals of the model and the associated risks. Sensitivity and specificity are often crucial, as they relate to the ability to correctly diagnose or rule out a condition. Precision is also significant when minimizing false positives is essential, such as in cancer screening. The choice of metric should align with the clinical significance of different types of errors for the specific medical problem you're addressing. Additionally, it's common to consider multiple metrics to provide a comprehensive evaluation of the model's performance.

Day 5: Query 5: Can you help me identify any scholars in the field who have attempted to reduce the recall point? In other word, trying to minimise the false negative prediction.

There are several researchers and scholars in the field of machine learning and healthcare who have worked on reducing false negative predictions in medical diagnostic tasks, thus aiming to increase recall. While I cannot provide a comprehensive list, I can mention a few notable researchers and areas of study in this field:

1. **Andrew Ng:** Andrew Ng, a prominent figure in machine learning and co-founder of Google Brain, has worked on various healthcare-related projects. He and his team have explored the use of deep learning algorithms in medical imaging, particularly in detecting diseases such as diabetic retinopathy.
2. **Dr. Lily Peng:** Dr. Lily Peng, a researcher at Google Health, has conducted extensive research in medical imaging and diagnosis using deep learning techniques. Her work includes developing models for detecting diabetic retinopathy and lung cancer from medical images.
3. **Dr. Daby Sow:** Dr. Daby Sow and his team have worked on using machine learning and data mining techniques for medical diagnosis, particularly in the context of detecting diseases like malaria from blood smear images. Their focus is on achieving high sensitivity (recall) to ensure accurate disease detection.
4. **Dr. Susan Murphy:** Dr. Susan Murphy, a professor at Harvard University, is known for her work on optimizing personalized treatment plans for medical conditions. Her research often involves reinforcement learning and other machine learning techniques to improve treatment efficacy and reduce false negatives.
5. **Dr. Saeed Anwar:** Dr. Saeed Anwar's research in computer vision and deep learning has applications in medical image analysis. He has explored the use of convolutional neural networks (CNNs) to enhance the detection and diagnosis of diseases such as skin cancer.
6. **Deep Learning in Medical Imaging:** Various scholars and researchers are working on using deep learning techniques, such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), for medical image analysis. These approaches aim to improve recall and accuracy in the diagnosis of diseases from X-rays, MRIs, CT scans, and other medical images.
7. **Telemedicine and Remote Monitoring:** Scholars are increasingly investigating remote monitoring and telemedicine applications, especially in the context of chronic disease management. These approaches aim to increase the recall of detecting potential health issues at an early stage.

Please note that the field of machine learning in healthcare is rapidly evolving, and many researchers are actively working on improving diagnostic accuracy and reducing false negatives in various medical domains. The names and areas mentioned here represent just a small portion of the broader research efforts in this field. It's always a good practice to review the latest literature and publications to stay updated on the most recent developments in healthcare and machine learning.

Day 5 Query 2: Specifically for decision tree training (same problem as above)

While specific scholars may not be commonly associated with "reducing the recall point" in decision tree training, many researchers and practitioners in the field of machine learning and healthcare have worked on improving the performance of decision trees and other classifiers to minimize false negative predictions in medical applications. Here are some areas and scholars who have contributed to this research:

1. **Imbalanced Data Research:** Many researchers have explored techniques to address imbalanced datasets, which are common in medical applications. Some approaches involve modifying decision trees to handle imbalanced data, such as cost-sensitive learning or modifying the splitting criteria to reduce false negatives. While not tied to a specific scholar, this area of study is essential for improving recall in decision trees.
2. **Medical Decision Trees:** Researchers in medical informatics and healthcare have examined decision tree models to enhance disease diagnosis and treatment decisions. These studies often emphasize the importance of achieving high sensitivity (recall) to detect medical conditions accurately. Specific scholars may vary, but this research is a collective effort in the field.
3. **Dr. Chris Drummond:** Dr. Chris Drummond, a researcher in machine learning and health informatics, has investigated decision tree models in healthcare applications. His work includes exploring the use of decision trees for predicting clinical outcomes and enhancing disease diagnosis.
4. **Dr. Alex Freitas:** Dr. Alex Freitas is known for his contributions to the field of machine learning, including decision tree algorithms. While his work extends beyond healthcare, his research may be relevant in the context of optimizing decision trees for high sensitivity in medical applications.
5. **Medical Imaging and Diagnosis:** Researchers and scholars working on medical imaging and diagnosis, such as mammography, lung cancer detection, or diabetic retinopathy, often aim to minimize false negatives to ensure early disease detection. The names of specific scholars may vary depending on the specific study or domain.

It's important to note that the field of machine learning in healthcare and decision tree training for medical applications is collaborative and interdisciplinary. Many researchers and teams work on these topics, and the focus on minimizing false negatives is a shared goal. To find specific papers or scholars in this area, consider using academic databases, research journals, and conferences in the field of medical informatics, healthcare, and machine learning.

Day 5 Query 3: Can you help me suggest potential methods to reduce the false negative prediction and increase the recall score

Certainly, increasing the recall score and reducing false negative predictions in a classification model, such as a decision tree, can be crucial in many applications, particularly in healthcare and other fields where missing positive cases can have serious consequences. Here are some methods and strategies to consider:

1. **Threshold Adjustment:**
   - By default, classification models use a threshold of 0.5 for binary classification. Increasing the threshold can reduce the number of false negatives. However, this might result in more false positives. Adjust the threshold to find a balance that suits your specific application.
2. **Cost-Sensitive Learning:**
   - Cost-sensitive learning involves modifying the decision tree to consider the varying costs of false positives and false negatives. By assigning higher costs to false negatives, you encourage the model to minimize these errors.
3. **Resampling Techniques:**
   - Resampling methods, such as oversampling the minority class or undersampling the majority class, can help balance the class distribution in imbalanced datasets, thereby reducing false negatives.
4. **Ensemble Methods:**
   - Ensemble methods, like Random Forest or Gradient Boosting, combine multiple decision trees to improve predictive accuracy. They can help reduce false negatives by combining the predictions of multiple models.
5. **Feature Engineering:**
   - Carefully selecting and engineering features can improve the performance of decision trees. Feature engineering should be tailored to the specific domain and problem to ensure that the relevant information is captured.
6. **Data Augmentation:**
   - In medical imaging and similar applications, data augmentation techniques can be used to create additional training examples by applying transformations to existing data, thus increasing the diversity of the training dataset.
7. **Cost-Sensitive Splitting:**
   - Modify the splitting criteria of the decision tree to favor splits that reduce false negatives. For example, choose splits that maximize the reduction in false negatives rather than Gini impurity.
8. **Feature Selection:**
   - Feature selection techniques can help identify the most informative features for reducing false negatives. Reducing the number of irrelevant or redundant features can improve model performance.
9. **Model Selection:**
   - Consider using different classifiers, such as Random Forest, Support Vector Machines, or Neural Networks, which may perform better on specific datasets and reduce false negatives.
10. **Hybrid Models:**
    - Combine multiple models, such as a decision tree with a neural network, to benefit from the strengths of each model. Hybrid models can be effective in reducing false negatives.
11. **Cross-Validation:**
    - Implement cross-validation techniques to evaluate model performance and tune hyperparameters. This helps ensure that the model generalizes well to unseen data and minimizes false negatives.