Decision trees

Nguyen Thanh Phat

B2005853

[1. **Build decision tree model** 1](#_Toc1)

[1.1. How to build the decision tree model for classifying the datasets 2](#_Toc2)

[To build the decision tree model for classifying the Golf dataset manually with entropy, we followed these steps: 2](#_Toc3)

[1.2. How many inductive rules are there in the decision tree model 3](#_Toc4)

[1.3. Use the decision tree model to classify 3 examples as follows: 4](#_Toc5)

[2. **Implement the decision tree program with scikit-learn** 4](#_Toc6)

[Implementation Details: 4](#_Toc7)

[Evaluation Procedure: 7](#_Toc8)

[Results: 7](#_Toc9)

[Dataset : Iris 7](#_Toc10)

[Dataset : Optics 8](#_Toc11)

[Dataset : Letter 8](#_Toc12)

[Dataset : Leukemia 9](#_Toc13)

[Dataset : Fp 9](#_Toc14)

[Dataset : Fp107 10](#_Toc15)

[3. **Why ensemble-based models improve the classification correctness of any single tree model?** 10](#_Toc16)

# **Build decision tree model**

**Assignment**: Given datasets Golf with 4 attributes Outlook, Temp, Humidity, Windy and an attribute Play (class).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Outlook** | **Temperature** | **Humidity** | **Windy** | **Class** |
| sunny | 85 | 85 | false | Don't Play |
| sunny | 80 | 90 | true | Don't Play |
| overcast | 83 | 78 | false | Play |
| rain | 70 | 96 | false | Play |
| rain | 68 | 80 | false | Play |
| rain | 65 | 70 | true | Don't Play |
| overcast | 64 | 65 | true | Play |
| sunny | 72 | 95 | false | Don't Play |
| sunny | 69 | 70 | false | Play |
| rain | 75 | 80 | false | Play |
| sunny | 75 | 70 | true | Play |
| overcast | 72 | 90 | true | Play |
| overcast | 81 | 75 | false | Play |
| rain | 71 | 80 | true | Don't Play |

* How to build the decision tree model for classifying the datasets
* How many inductive rules are there in the decision tree model
* Use the decision tree model to classify 3 examples as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Outlook** | **Temperature** | **Humidity** | **Windy** | **Class** |
| overcast | 63 | 70 | false | ? |
| rain | 73 | 90 | true | ? |
| sunny | 70 | 73 | true | ? |

## How to build the decision tree model for classifying the datasets

# To build the decision tree model for classifying the Golf dataset manually with entropy, we followed these steps:

**Data Examination:** We analyzed the Golf datasets, comprising Outlook, Temperature, Humidity, Windy, and the class attribute Play.

**Entropy Calculation:** We calculated the entropy of the target variable (class attribute) to measure the impurity in the data. Entropy is given by the formula:

Where is the set of instances, cc is the number of classes, and pipi​ is the proportion of instances in class ii.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Outlook** | **nYes** | **nNo** | **Entropy** | **W** | **W\*Entropy** | **IG** |
| sunny | 2 | 3 | 0.971 | 0.357142857 | 0.346768069 | 0.593517889 |
| overcast | 4 | 0 | 0 | 0.285714286 | 0 | 0.940285959 |
| rain | 3 | 2 | 0.971 | 0.357142857 | 0.346768069 | 0.593517889 |
|  |  |  |  | **Impurity** | 0.693536139 | 0.24674982 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Windy** | **nYes** | **nNo** | **Entropy** | **W** | **W\*Entropy** | **IG** |
| TRUE | 3 | 3 | 1 | 0.428571429 | 0.428571429 | 0.51171453 |
| FALSE | 6 | 2 | 0.811 | 0.571428571 | 0.4635875 | 0.476698459 |
|  |  |  |  | **Impurity** | 0.892158928 | 0.04812703 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Temperature** | 64 | 65 | 68 | 69 | 70 | 71 | 72 | 72 | 75 | 75 | 80 | 81 | 83 | 85 |
| **Class** | Play | Don't Play | Play | Play | Play | Don't Play | Don't Play | Play | Play | Play | Don't Play | Play | Play | Don't Play |
| **Split point** | 64.5 | 66.5 |  |  | 70.5 |  |  |  | 77.5 |  | 80.5 |  | 84 |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Temperature** | **nYes** | **nNo** | **Entropy** | **W** | **W\*Entropy** |  | **IG** |
| <64.5 | 1 | 0 | 0 | 0.071429 | 0 | **Impurity** |  |
| >=64.5 | 8 | 5 | 0.96124 | 0.928571 | 0.89257685 | 0.892576847 | 0.047709111 |
| <66.5 | 1 | 1 | 1 | 0.142857 | 0.14285714 | **Impurity** |  |
| >=66.5 | 8 | 4 | 0.9183 | 0.857143 | 0.78711072 | 0.929967858 | 0.010318101 |
| <70.5 | 4 | 1 | 0.72193 | 0.357143 | 0.25783146 | **Impurity** |  |
| >=70.5 | 5 | 4 | 0.99108 | 0.642857 | 0.63712032 | 0.894951787 | 0.045334172 |
| <77.5 | 7 | 3 | 0.88129 | 0.714286 | 0.6294935 | **Impurity** |  |
| >=77.5 | 2 | 2 | 1 | 0.285714 | 0.28571429 | 0.915207785 | 0.025078174 |
| <80.5 | 7 | 4 | 0.94566 | 0.785714 | 0.74301881 | **Impurity** |  |
| >=80.5 | 2 | 1 | 0.9183 | 0.214286 | 0.19677768 | 0.939796489 | 0.000489469 |
| <84 | 9 | 4 | 0.89049 | 0.928571 | 0.82688509 | **Impurity** |  |
| >=84 | 0 | 1 | 0 | 0.071429 | 0 | 0.826885094 |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Humidity** | 65 | 70 | 70 | 70 | 75 | 78 | 80 | 80 | 80 | 85 | 90 | 90 | 95 | 96 |
| **Class** | Play | Don't Play | Play | Play | Play | Play | Play | Play | Don't Play | Don't Play | Don't Play | Play | Don't Play | Play |
| **Split point** | 67.5 |  |  |  |  |  |  |  |  |  |  | 92.5 | 95.5 |  |

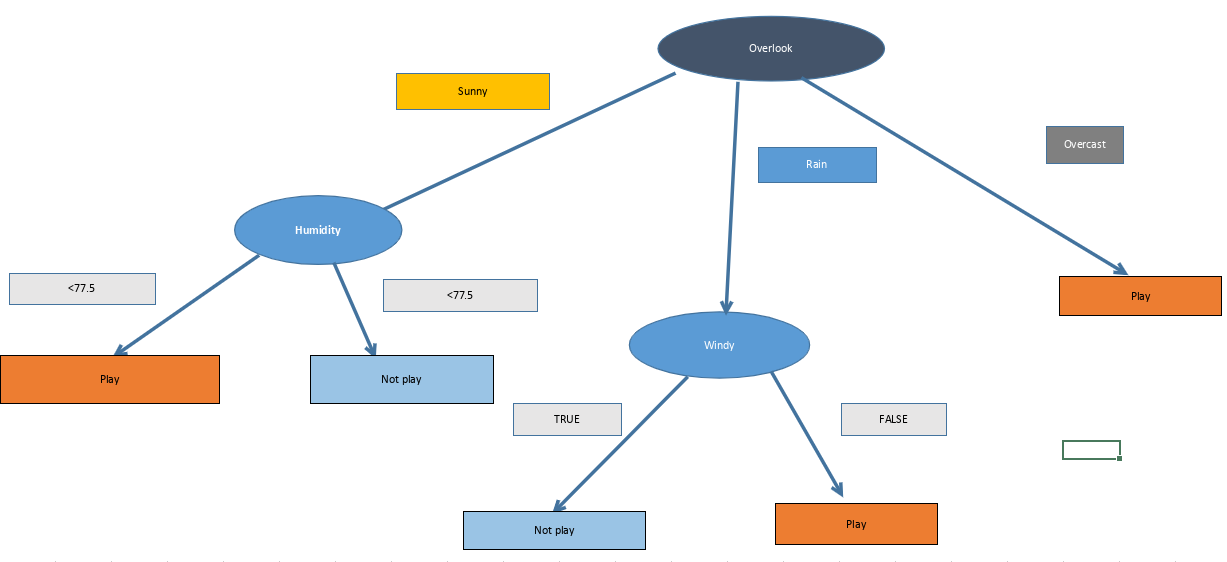
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Humidity** | **nYes** | **nNo** | **Entropy** | **W** | **W\*Entropy** |  | **IG** |
| <67.5 | 1 | 0 | 0 | 0.071429 | 0 | **Impurity** |  |
| >=67.5 | 8 | 5 | 0.96124 | 0.928571 | 0.89257685 | 0.892576847 | 0.047709111 |
| <92.5 | 8 | 4 | 1 | 0.857143 | 0.78711072 | **Impurity** |  |
| >=92.5 | 1 | 1 | 1.0000 | 0.142857 | 0.14285714 | 0.929967858 | 0.010318101 |
| <95.5 | 8 | 5 | 0.96124 | 0.928571 | 0.89257685 | **Impurity** |  |
| >=95.5 | 1 | 0 | 0.00000 | 0.071429 | 0.00000000 | 0.892576847 | 0.047709111 |

Attribute Selection: Attribute Selection: Identified the attribute with the highest information gain to serve as the root node.

Decision Tree Construction: Constructed the decision tree recursively, selecting attributes at each node to maximize information gain.

Inductive Rules: Defined a set of rules representing conditions leading to assignment, observed by traversing paths from the root to leaf nodes

## How many inductive rules are there in the decision tree model



So, there are 4 inductive rules in this decision tree model.

* There are 4 leaf nodes in the tree, each representing a class.
* Each path from the root to a leaf node represents a rule.

## Use the decision tree model to classify 3 examples as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Outlook** | **Temperature** | **Humidity** | **Windy** | **Class** |
| overcast | 63 | 70 | false | Play |
| rain | 73 | 90 | true | Don’t play |
| sunny | 70 | 73 | true | Play |

Set 1:

Outlook -> overcast

Class: Play

Set 2:

Outlook -> rain

Windy -> true

So, Class: Don't Play

Set 3:

Outlook -> sunny

Humidity -> 70 -> 77.5

So, Class: Play

# Implement the decision tree program with scikit-learn

Implement the program using **DecisionTreeClassifier** in **scikit-learn** library. The program requires 2 parameters:

* + file name of trainset
  + file name of testset

The program reports the classification results (accuracy, confusion matrix) for 5 datasets:

* + Iris (.trn: trainset, .tst: testset)
  + Optics (.trn: trainset, .tst: testset)
  + Letter (.trn: trainset, .tst: testset)
  + Leukemia (.trn: trainset, .tst: testset)
  + Fp (.trn: trainset, .tst: testset)

## Implementation Details:

* The script is written in Python, leveraging the scikit-learn library for decision tree classification.
* It utilizes the DecisionTreeClassifier class for training decision tree models.
* Data loading, model training, evaluation, and result saving are all encapsulated in functions for modularity and clarity.

|  |
| --- |
| import numpy as np  from sklearn.tree import DecisionTreeClassifier  from sklearn.metrics import accuracy\_score, confusion\_matrix  import matplotlib.pyplot as plt  import seaborn as sns    def load\_data(filename):      """Loads data from a CSV or space-delimited text file."""      try:          data = np.loadtxt(filename, delimiter=",", dtype=float)      except:          data = np.loadtxt(filename, delimiter=" ", dtype=float)      X = data[:, :-1]      y = data[:, -1].astype(int)      return X, y    def print\_confusion\_matrix(confusion):      """Prints the confusion matrix in text format."""      for row in confusion:          print(row)    def save\_results\_to\_file(accuracy, confusion, features, dataset\_name):      """Saves accuracy, confusion matrix, and tree features to a text file named "results.txt"."""      with open("results.txt", "a") as f:          f.write(f"Dataset: {dataset\_name}\n")          f.write(f"Accuracy: {accuracy:.4f}\n")          f.write("\nTree Features:\n")          f.write(features)          f.write("Confusion Matrix:\n")          np.savetxt(f, confusion, fmt="%d")    def export\_confusion\_matrix(confusion, accuracy, num\_nodes, dataset\_name):      """Generates and saves a confusion matrix heatmap image."""      sns.heatmap(confusion, annot=True, fmt="d", cmap="Blues", cbar=False)      plt.xlabel("Predicted Label")      plt.ylabel("True Label")      plt.title(          f"Confusion Matrix - {dataset\_name} (Accuracy: {accuracy:.4f}, Nodes: {num\_nodes})"      )        plt.tight\_layout()      plt.savefig(f"{dataset\_name}\_combined\_plot.png")      plt.close()    def test\_model(clf, X\_test, y\_test):      """Makes predictions using the classifier, calculates accuracy, and returns both."""      y\_pred = clf.predict(X\_test)      accuracy = accuracy\_score(y\_test, y\_pred)      return accuracy, y\_pred    def get\_tree\_features\_context(clf):      """Extracts relevant features from the trained decision tree."""      num\_nodes = clf.tree\_.node\_count      max\_depth = clf.tree\_.max\_depth      num\_features = clf.n\_classes\_      features = (          f"Number of Nodes: {num\_nodes}\n"          f"Maximum Depth: {max\_depth}\n"          f"Number of Features: {num\_features}\n"      )      return features    def decision\_tree\_classification(trainset\_filename, testset\_filename, dataset\_name=""):      """Performs decision tree classification for a single dataset."""      # Load train and test data      X\_train, y\_train = load\_data(trainset\_filename)      X\_test, y\_test = load\_data(testset\_filename)        # Initialize DecisionTreeClassifier      clf = DecisionTreeClassifier()        # Train classifier      clf.fit(X\_train, y\_train)        # Get tree features      features = get\_tree\_features\_context(clf)        # Test and evaluate      accuracy, y\_pred = test\_model(clf, X\_test, y\_test)      confusion = confusion\_matrix(y\_test, y\_pred)        # Print the result to console      print(f"\nDataset: {dataset\_name}")      print("Test Accuracy:", accuracy)      print(features)      print("\nConfusion Matrix:")      print\_confusion\_matrix(confusion)        # Save results      num\_nodes = clf.tree\_.node\_count      export\_confusion\_matrix(confusion, accuracy, num\_nodes, dataset\_name)      save\_results\_to\_file(accuracy, confusion, features, dataset\_name)    if \_\_name\_\_ == "\_\_main\_\_":      datasets = [          {              "name": "Iris",              "train\_file": "data//iris//iris.trn",              "test\_file": "data//iris//iris.tst",          },          {              "name": "Optics",              "train\_file": "data//optics//optics.trn",              "test\_file": "data//optics//optics.tst",          },          {              "name": "Letter",              "train\_file": "data//letter//letter.trn",              "test\_file": "data//letter//letter.tst",          },          {              "name": "Leukemia",              "train\_file": "data//leukemia//leukemia.trn",              "test\_file": "data//leukemia//leukemia.tst",          },          {              "name": "Fp",              "train\_file": "data//fp//fp.trn",              "test\_file": "data//fp//fp.tst",          },          {              "name": "Fp017",              "train\_file": "data//fp107//fp107.trn",              "test\_file": "data//fp107//fp107.tst",          },      ]        for dataset in datasets:          decision\_tree\_classification(              dataset["train\_file"], dataset["test\_file"], dataset["name"]          )          print("\n") |

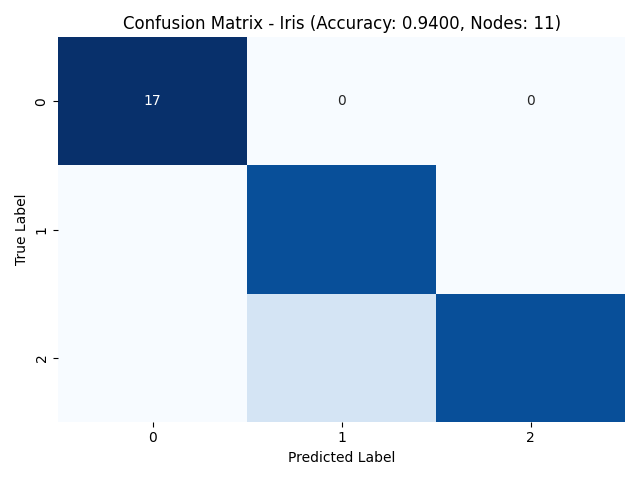
## Evaluation Procedure:

* The script loads the training and testing datasets for each dataset.
* It trains a decision tree classifier on the training data.
* The trained model is then evaluated on the testing data, and performance metrics such as accuracy and confusion matrix are computed.
* Results are both printed to the console and saved to files for further analysis.

## Results:

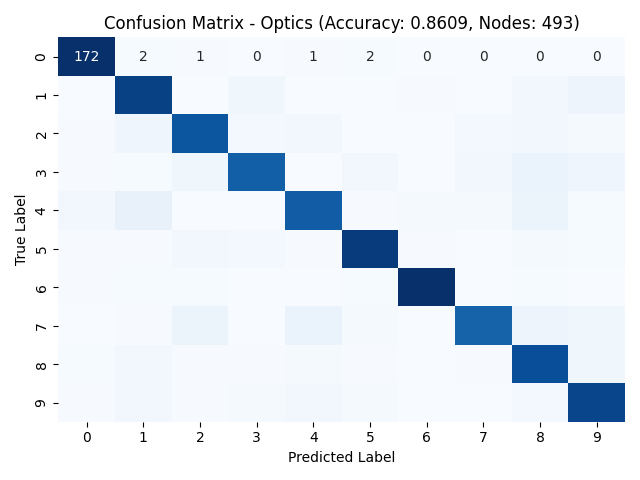
### **Dataset : Iris**

* Accuracy:
* Accuracy: 0.94
* Confusion Matrix



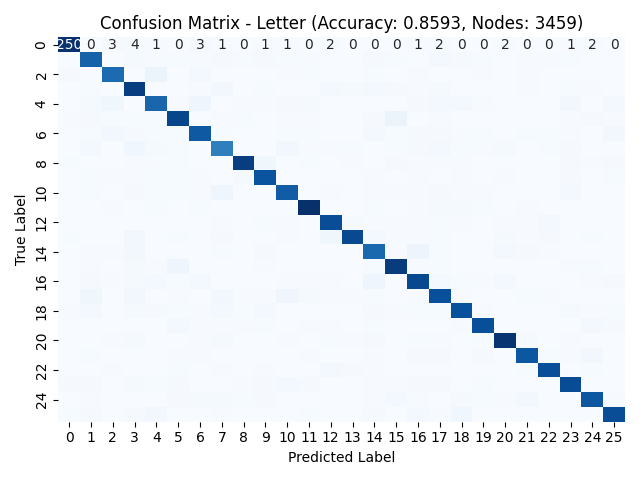
### Dataset : Optics

* Accuracy: 0.8609
* Confusion Matrix:



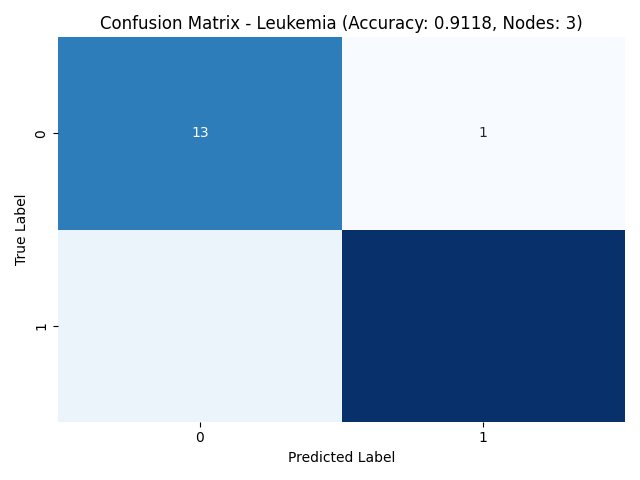
### Dataset : Letter

* Accuracy: 0.8593
* Confusion Matrix:



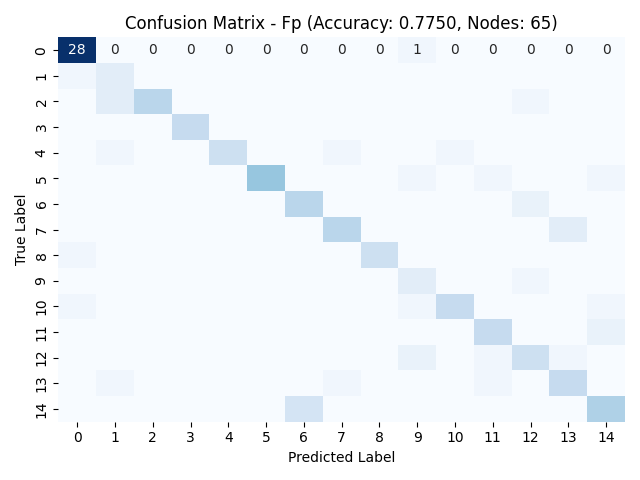
### Dataset : Leukemia

* Accuracy: 0.9118
* Confusion Matrix:



### Dataset : Fp

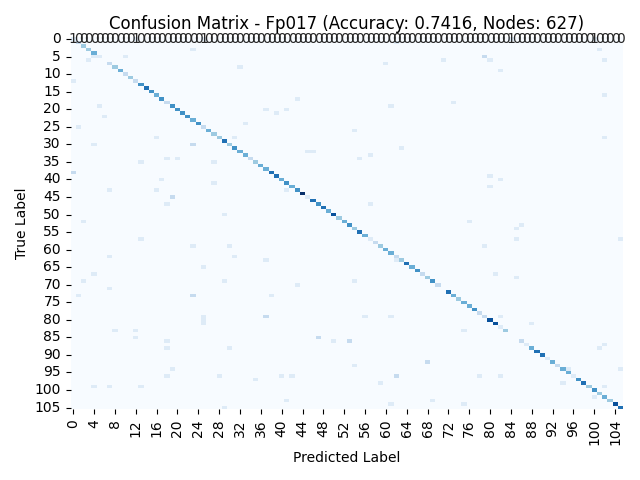
* Accuracy: 0.7750
* Confusion Matrix:



### Dataset : Fp107

* Accuracy: 0.7416

Confusion Matrix



# Why ensemble-based models improve the classification correctness of any single tree model?

Ensemble-based models boost the accuracy of classification compared to individual tree models through:

**Variance Reduction:** Ensemble methods, like Random Forests, amalgamate predictions from multiple trees, minimizing the impact of individual tree variability. This results in a more stable and reliable model, enhancing correctness.

**Complementary Learning:** Each tree in an ensemble might capture unique patterns or rectify errors made by others. By combining diverse models, ensembles capitalize on complementary learning, resulting in a more comprehensive understanding of the data and improved correctness.

**Overfitting Mitigation:** Single trees can overfit, particularly with depth and complexity. Ensembles mitigate overfitting by aggregating predictions from multiple trees, thus averting the memorization of noise and enhancing the model's ability to generalize.

**Enhanced Generalization:** Ensembles leverage the collective knowledge of multiple models, making them more adept at generalizing to unseen data. This collective intelligence contributes to superior correctness across diverse datasets and scenarios.

In summary, ensemble-based models elevate classification correctness by amalgamating diverse insights, minimizing variance, addressing overfitting, and bolstering generalization capabilities. This results in more accurate and robust models for classification tasks.