



**DOKUZ EYLÜL UNIVERSITY**

**ENGINEERING FACULTY**

**DEPARTMENT OF COMPUTER ENGINEERING**

**DEUCENG**  
Dokuz Eylül University  
Dept. of Computer Engineering



**CME 3402 CONCENTPTS OF PROGRAMING LANGUAGES**

## **Decision Tree with GO Programming Language**

**By:**

**2022510013 – Yasamin Valishariatpanahi (GO)**

**2022510046 – Sara Mahyanbakhshayesh (JAVA)**

**2021510070 – Ege Yıldırım (PYTHON)**

**Lecturer: Doç. Dr. YUNUS DOĞAN**

**Spring 2025  
İzmir**

# INTRODUCTION

Decision trees offer a clear and intuitive approach to solving classification problems by guiding decisions through a series of yes/no questions. This assignment aimed to explore the inner workings of decision trees by implementing the ID3 algorithm entirely from scratch using the Go (Golang) programming language.

Key components such as entropy (used to measure the purity of data) and information gain (used to determine the most informative attribute for splitting) were calculated manually, without the help of external libraries. Based on these computations, the decision tree was also constructed manually, step by step, providing valuable insights into the logic and structure behind tree-based classifiers.

The final outcome is a command-line application that:

- Reads input data from CSV or TXT files
- Calculates and displays entropy and information gain at each decision point
- Builds the decision tree manually based on computed values
- Prints the tree in a clear, hierarchical text format
- Generates a visual representation of the tree as a PNG image
- Accepts new user input to make predictions using the constructed tree

This project deepened the understanding of how decision trees function while also providing valuable experience with the Go programming language.

## Dataset Used

To evaluate the implementation, the following datasets—composed of categorical attributes suited for ID3—were used:

- weather.csv/.txt – for the "Play Tennis" decision problem
- contact\_lenses.csv/.txt – for determining appropriate contact lenses
- breast\_cancer.csv/.txt – for classifying breast cancer cases

These datasets helped in validating the functionality and accuracy of the algorithm while reinforcing a deeper understanding of decision tree construction and classification.

# Building the Decision Tree Step by Step with ID3 Algorithm

## ENTROPY: UNDERSTANDING DATA IMPURITY IN DECISION TREES

Entropy is a fundamental concept from information theory that quantifies the amount of uncertainty or impurity in a dataset. In the context of decision trees, entropy is used to measure how mixed the class labels are within a given subset of data. It serves as a criterion to determine the quality of a split: the lower the entropy, the purer the subset.

A perfectly pure dataset—where all instances belong to the same class—has an entropy of 0, indicating no disorder or uncertainty. Conversely, if the dataset is evenly distributed across multiple classes, entropy is at its maximum, indicating the highest level of disorder and uncertainty in predicting the target class.

### Mathematical Definition

The entropy  $H(S)$  of a dataset  $S$  is calculated using the following formula:

$$H(S) = - \sum_{i=1}^c p_i \log_2 p_i$$

Where:

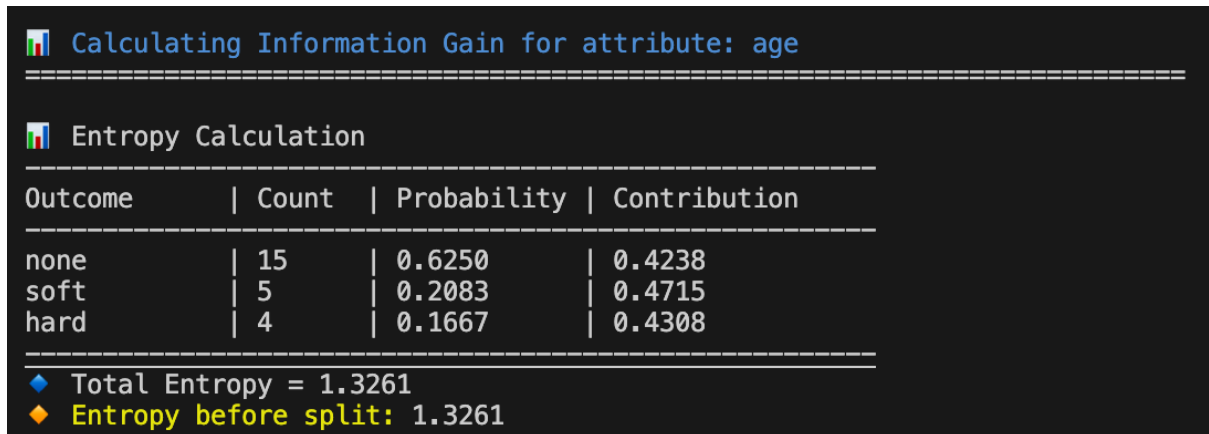
- $H(S)$  is the entropy of dataset  $S$
- $c$  is the total number of classes
- $p_i$  is the proportion of instances in class  $i$  relative to the total number of instances in the dataset

### Interpreting Entropy Values

- **Entropy = 0:** The dataset is completely pure (all instances belong to a single class).
- **Entropy > 0 and < 1:** The dataset contains a mix of classes, but one class may dominate.
- **Entropy = 1 (for binary classification):** The dataset is equally split between two classes, representing maximum impurity.

## Example from the Project

The following screenshot from the project illustrates the calculation of entropy for a dataset with a balanced distribution of classes:



```
Calculating Information Gain for attribute: age
=====
Entropy Calculation
-----
Outcome      | Count | Probability | Contribution
-----
none         | 15    | 0.6250      | 0.4238
soft         | 5     | 0.2083      | 0.4715
hard         | 4     | 0.1667      | 0.4308
-----
◆ Total Entropy = 1.3261
◆ Entropy before split: 1.3261
```

Outcome	Count	Probability	Contribution
none	15	0.6250	0.4238
soft	5	0.2083	0.4715
hard	4	0.1667	0.4308

◆ Total Entropy = 1.3261  
◆ Entropy before split: 1.3261

Figure 1: Program output showing entropy calculation for the age attribute. The entropy value of 1.3261 indicates a high level of impurity in the dataset prior to splitting.

This high entropy suggests that no single class dominates, so the dataset is highly uncertain and not ideal for classification without further splitting.

## Why Entropy Matters

Entropy is used during the decision tree construction process to evaluate how informative an attribute is in classifying the dataset. Attributes that result in lower entropy after splitting the data are considered more effective, as they produce purer subsets. This forms the basis for calculating Information Gain, which guides the ID3 algorithm in choosing the best attribute at each step of the tree-building process.

## INFORMATION GAIN: SELECTING THE MOST INFORMATIVE ATTRIBUTE

Information Gain is a key metric used in decision tree algorithms to determine which attribute best separates the data into classes. It quantifies the reduction in uncertainty or impurity that results from splitting the dataset based on a particular attribute.

In essence, Information Gain measures how much knowing the value of an attribute improves the prediction of the target class. The attribute with the highest Information Gain is chosen at each step of the tree construction because it most effectively organizes the data into purer subsets.

## Mathematical Definition

Information Gain  $IG(S, A)$  for an attribute  $A$  on dataset  $S$  is calculated as:

$$IG(S, A) = H(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} H(S_v)$$

Where:

- $IG(S, A)$  is the Information Gain of attribute  $A$  on dataset  $S$
- $H(S)$  is the entropy of the entire dataset before the split
- $\text{Values}(A)$  is the set of all possible values for attribute  $A$
- $S_v$  is the subset of  $S$  where attribute  $A$  takes the value  $v$
- $|S_v|$  and  $|S|$  are the number of samples in  $S_v$  and  $S$  respectively
- $H(S_v)$  is the entropy of subset  $S_v$

## How It Works

- First, calculate the entropy of the full dataset,  $H(S)$ , which represents the current level of impurity.
- Then, calculate the weighted entropy of each subset  $S_v$  created by splitting the dataset on attribute  $A$ .
- The weighted sum of these subset entropies represents the expected entropy after splitting.
- Subtracting this from the original entropy gives the Information Gain.
- The attribute with the largest Information Gain is selected for splitting, as it yields the greatest reduction in uncertainty.

## Example from the Project

The screenshot below demonstrates the Information Gain calculation for the attribute age within the project. The output shows how the total entropy is reduced after splitting based on different age values, guiding the selection of the best attribute for the next node in the decision tree:

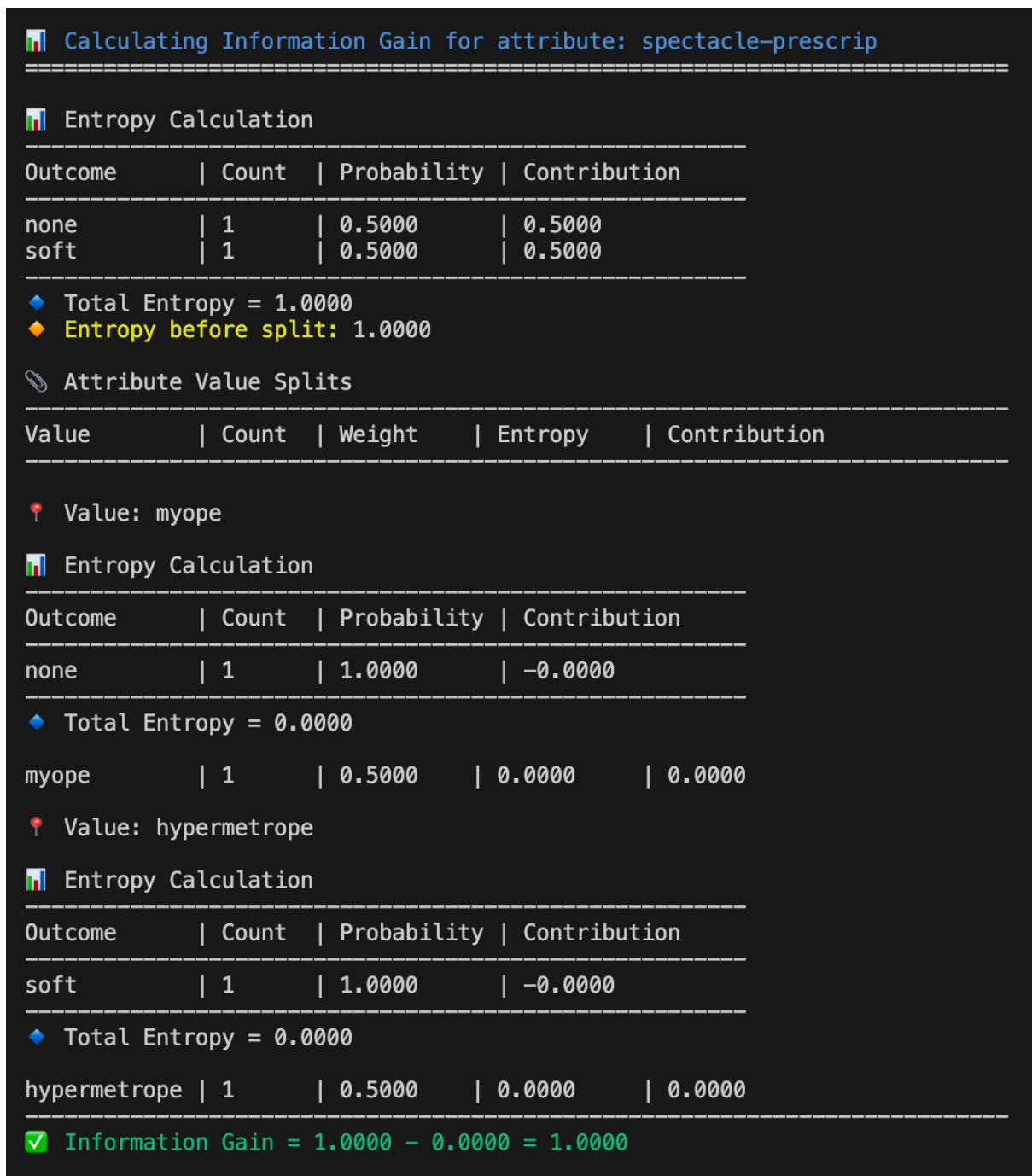


Figure 2: Information Gain calculation for the attribute spectacle-prescrip, demonstrating how splitting the dataset by attribute values myope and hypermetrope reduces entropy from 1.0000 to 0.0000, resulting in an information gain of 1.0000.

## Why Information Gain Matters

Information Gain is crucial because it directly influences how a decision tree learns to classify data. By quantifying how much an attribute reduces uncertainty, it guides the tree-building process to make the most effective splits.

- **Maximizing Purity:** Information Gain helps select attributes that produce the purest possible child nodes, which improves classification accuracy.

- **Efficient Learning:** Choosing attributes with high information gain reduces the number of splits needed, resulting in a simpler and more interpretable tree.
- **Preventing Overfitting:** By focusing on attributes that truly reduce uncertainty, the decision tree avoids unnecessary complexity caused by splitting on irrelevant or noisy features.
- **Stepwise Improvement:** At each node, Information Gain ensures that the tree progressively improves its ability to separate classes, leading to better predictions.

In short, Information Gain acts as the compass that steers the decision tree towards the attributes that offer the most meaningful information, making it an essential component in building effective classifiers.

## ENTROPY OUTPUT MODES OPTIONS

To improve usability, the program allows the user to choose **how entropy and information gain details are displayed**. Upon execution, users are prompted with three display options:

1. **Show in Terminal** – Displays entropy and gain calculations directly in the console.
2. **Save to File (entropy\_output.txt)** – Writes detailed output into a text file for reference.
3. **Skip Display** – Proceeds without showing any intermediate calculations.

This flexible output mode ensures that users can either trace the tree-building process in detail or keep the interface clean depending on their needs.

```

How would you like to view entropy and gain calculations?
1. Show in terminal
2. Save to file (entropy_output.txt)
3. Skip showing calculations
Enter choice (1/2/3): █

```

Figure 3: Prompting different way of viewing



Figure 4: Save to File as TXT to view the calculation better for lage datasets

## BUILDING THE TREE USING THE ID3 ALGORITHM

The construction of a decision tree using the ID3 (Iterative Dichotomiser 3) algorithm follows a recursive, top-down approach. At each step, the algorithm selects the attribute that provides the highest Information Gain, aiming to reduce the dataset's impurity as much as possible.

### Tree Construction Steps

1. **Calculate Entropy** of the full dataset to measure initial impurity.
2. **Evaluate Information Gain** for each attribute (excluding the decision column).
3. **Select the Attribute** with the highest gain for splitting.
4. **Create a Decision Node** based on the selected attribute.
5. **Partition the Data** by the values of the selected attribute.
6. **Recurse:**
  - Repeat steps 1–5 for each subset.
  - Exclude previously used attributes.
7. **Terminate** when:
  - All instances in a subset belong to the same class.
  - No attributes remain for further splitting.
  - A predefined condition is met (e.g., max tree depth or minimum gain threshold).

### Missing Attributes in the Final Tree

Not all attributes from the dataset appear in the final decision tree. This is a common and expected outcome when using the ID3 algorithm. Only attributes that contribute meaningfully to reducing entropy—measured via **Information Gain**—are selected at each step of tree construction.

An attribute may be excluded from the final tree due to one or more of the following reasons:

- **Zero or negligible Information Gain:** If an attribute does not significantly reduce uncertainty in the data, the algorithm skips it in favor of more informative attributes.
- **Redundancy or correlation:** The attribute may be closely related to another attribute already used earlier in the tree, making its contribution redundant.



- **Purity of subsets:** During recursive tree construction, some subsets may become pure (i.e., contain only one class) before the algorithm needs to consider all remaining attributes. In such cases, the tree stops growing on that path.

This behavior reflects the **efficiency and selectiveness of the ID3 algorithm**—only attributes that actively improve classification are used. It helps avoid overfitting and keeps the resulting model interpretable and compact.

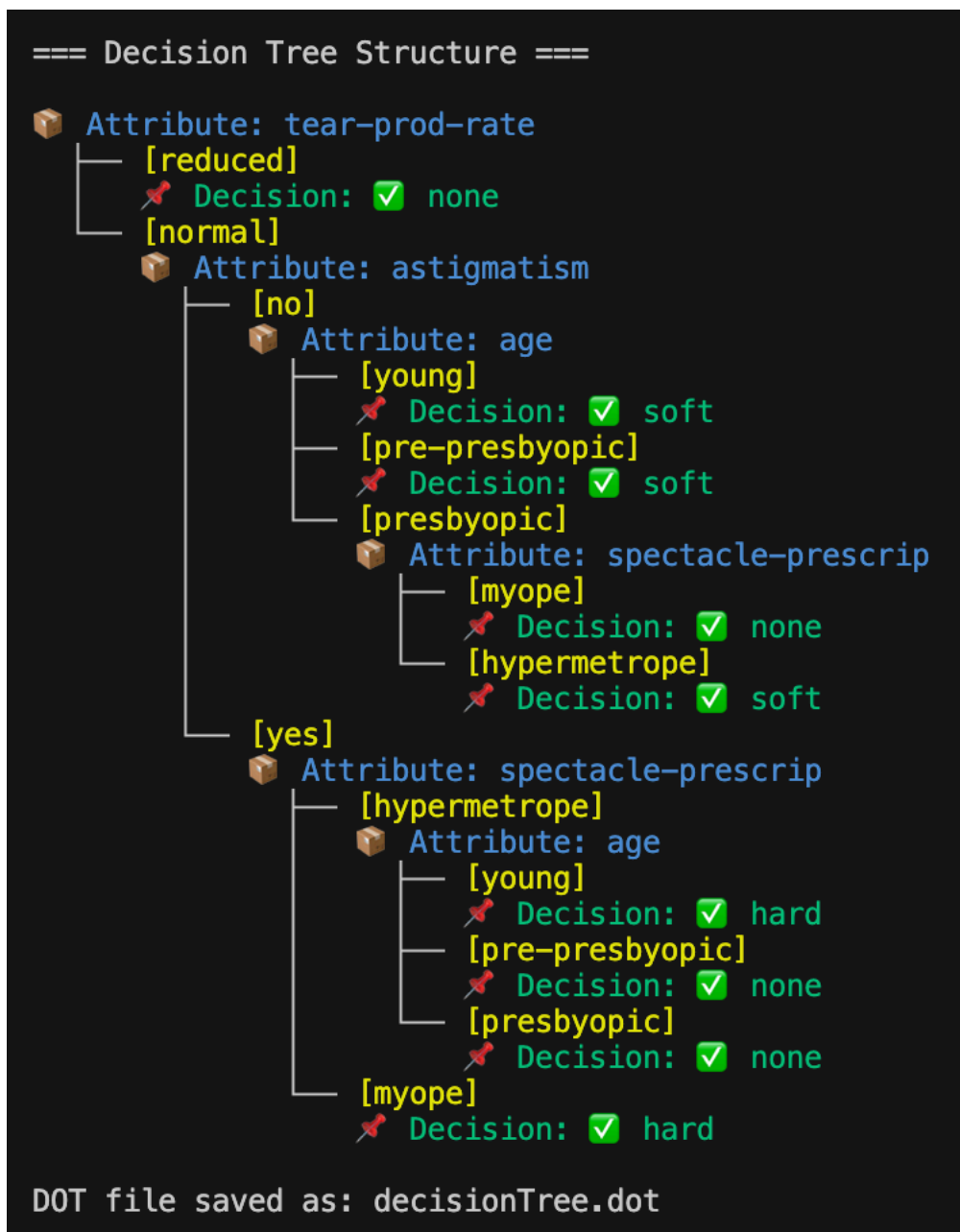


Figure 5: Final decision tree structure generated using the ID3 algorithm. Each node represents a decision point based on an attribute, and each leaf node (📌) indicates the final classification outcome. The tree reflects the step-by-step manual construction derived from entropy and information gain calculations.

## VISUALIZING THE DECISION TREE WITH Graphviz

Once the decision tree is constructed, it can be **exported as a DOT file** (decisionTree.dot)—a plain text format used by **Graphviz** to describe graphs.

To create a **visual representation** of the tree, Graphviz's dot command is used to render the structure into image formats such as PNG or SVG. Depending on the size and complexity of the tree, different rendering options can be applied to ensure clarity and resolution.

### Rendering Commands

- **Standard PNG Output** (for normal-sized trees):

```
$ dot -Tpng -Gdpi=300 decisionTree.dot -o decisionTree.png
```

This command generates a high-resolution PNG image at 300 DPI, suitable for inclusion in documents.

- **Scaled Output for Large Trees:**

If the tree is large and nodes are overlapping or unreadable, scaling can help:

```
$ dot -Tpng -Gdpi=300 -Gscale=2 decisionTree.dot -o decisionTree.png
```

Or to export as a scalable vector image:

```
$ dot -Tsvg decisionTree.dot -o decisionTree.svg
```

### Tree Visualization Examples

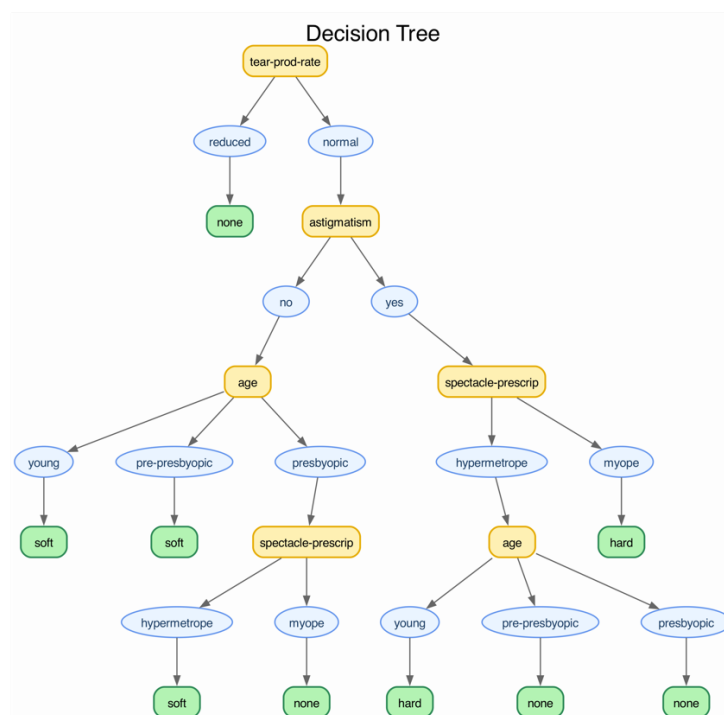


Figure 6: Visualization of a decision tree of moderate size rendered using Graphviz with standard DPI. Each node corresponds to a decision point, and leaf nodes show the final classification outcomes.



Figure 7: Visualization of a larger decision tree with scaled rendering to improve readability. The use of `-Gscale=2` helps accommodate deeper trees with more branches

## SAVING THE DECISION TREE & CREATING VISUALS

Once the decision tree is built using the ID3 algorithm, it is automatically exported as a DOT file and saved in a dedicated folder named `decision_tree/`. This structured separation of output files makes it easier for users to manage multiple datasets.

### Folder Structure

```
decision_tree/
├── weather_decisionTree.dot
├── contact_lenses_decisionTree.dot
├── breast_cancer_decisionTree.dot
```

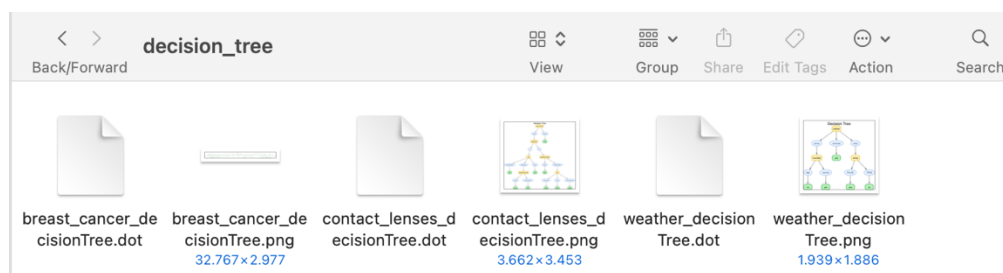


Figure 8: Folder Structure

## Generating Tree Visualizations (User Manual)

To convert a .dot file into a PNG image, the user can run:

```
DOT file saved at: decision_tree/breast_cancer_decisionTree.dot
For high-resolution PNG:
✦ Use: dot -Tpng -Gdpi=300 decision_tree/breast_cancer_decisionTree.dot -o decision_tree/breast_cancer_decisionTree.png
If the Tree is Large:
✦ Use: dot -Tpng -Gdpi=300 -Gscale=2 decision_tree/breast_cancer_decisionTree.dot -o decision_tree/breast_cancer_decisionTree.png
SVG Format:
✦ Use: dot -Tsvg decision_tree/breast_cancer_decisionTree.dot -o decision_tree/breast_cancer_decisionTree.png
```

Figure 9: User Manual in Terminal

## PREDICTION USING THE DECISION TREE

Once the decision tree is constructed using the ID3 algorithm, it becomes a powerful tool for classifying new, unseen instances. Predictions are made by **recursively traversing** the tree from the root to a leaf node based on the input values. This process mirrors how humans make decisions—by answering a series of questions.

### How Prediction Works

1. **Start at the Root:** The algorithm begins at the root node of the decision tree.
2. **Evaluate Split Attribute:** At each internal node, the tree checks the value of the attribute used to split the dataset at that level.
3. **Follow Matching Branch:** The algorithm follows the branch that matches the input's value for that attribute.
4. **Repeat:** This process continues recursively down the tree until a **leaf node** is reached.
5. **Output Class Label:** The value stored at the leaf node represents the **predicted class** for the input instance.

If the tree encounters a value that was not part of the training dataset (i.e., no branch exists for the given input), the algorithm returns a default prediction such as "Unknown".

### Interactive Prediction in the CLI Application

The `main()` function includes an **interactive prompt** that allows users to:

- Select a dataset (weather.csv, contact\_lenses.csv, or breast\_cancer.csv)
- View the structure of the generated decision tree
- Enter attribute values for prediction one by one
- Receive the predicted class label immediately

Here's a view of how the user interacts with the program:

```
Please enter input values for prediction (type 'exit' at any prompt to quit):
Enter value for outlook (sunny, overcast, rainy): sunny
Enter value for temperature (hot, mild, cool): mild
Enter value for humidity (high, normal): normal
Enter value for windy (FALSE, TRUE): TRUE

>>> Predicted Decision (play): yes
```

*Figure 10: Interactive prediction example using the constructed decision tree. The user is prompted to enter attribute values one by one, and the system outputs the predicted decision based on the input. This process is executed in a loop, continuously accepting new predictions until the user types "exit" at any prompt.*

If an attribute value is entered incorrectly, the program provides helpful feedback and re-prompts the user:

```
Please enter input values for prediction (type 'exit' at any prompt to quit):
Enter value for outlook (sunny, overcast, rainy): SUNNY
❌ Invalid input. Please choose one of the listed valid values.
Enter value for outlook (sunny, overcast, rainy): █
```

Figure 11: Validation of user input during the prediction phase. The program ensures that only valid attribute values—matching the dataset—are accepted. If the user enters an invalid value (e.g., incorrect casing or unrecognized string), the system prompts them to try again. This input loop continues until valid values are provided or the user types "exit" to terminate the session.

Users can type "exit" at any point to terminate the program gracefully.

```
Please enter input values for prediction (type 'exit' at any prompt to quit):
Enter value for outlook (sunny, overcast, rainy): exit
Exiting prediction. Goodbye!
❯ yasi@YASAMINs-MacBook-Pro Decision_Tree_Go % █
```

Figure 12: Graceful termination of the prediction loop. When the user types "exit" at any input prompt, the program halts the prediction process and exits cleanly, displaying a farewell message. This ensures user control and a smooth command-line interaction experience.

## Why This is Important

This prediction phase transforms the decision tree from a static model into a **practical classifier**. By allowing real-time predictions, the system demonstrates:

- The **effectiveness** of the ID3 algorithm in learning from categorical data
- The **interpretability** of decisions through a human-readable structure
- The **applicability** of Go (Golang) in building performant, interactive machine learning tools without external libraries

## PREDICTION PATH VISUALIZATION

Every prediction made through user input is traced from root to leaf. The program **exports each prediction path as a .dot file** stored in a separate folder called `prediction_paths/`.

### Folder Structure

```
prediction_paths/
├── prediction_20250525_152632.dot
├── prediction_20250525_153135.dot
...
```

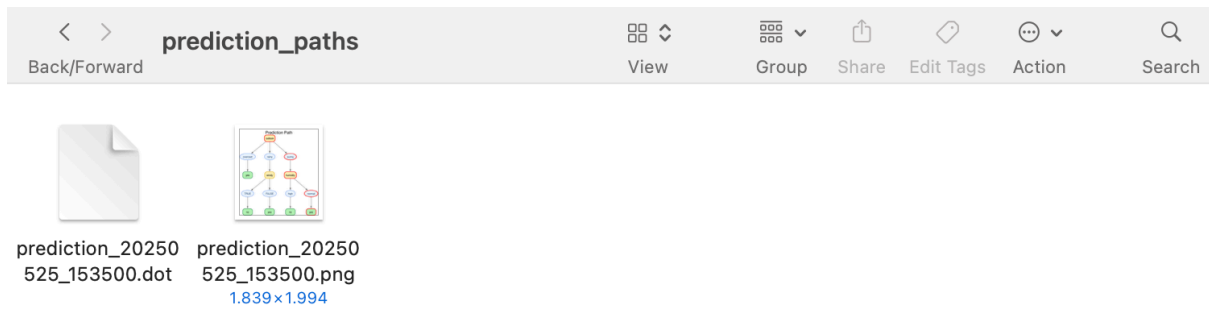


Figure 13: Folder Structure

## How to Generate Prediction Path Images (User Manual)

Each path can be rendered as an image using:

```
Please enter input values for prediction (type 'exit' at any prompt to quit):
Enter value for outlook (sunny, overcast, rainy): rainy
Enter value for temperature (hot, mild, cool): cool
Enter value for humidity (high, normal): normal
Enter value for windy (FALSE, TRUE): TRUE

>>> Predicted Decision (play): no
🌱 Prediction path DOT file saved at: prediction_paths/prediction_20250525_155238.dot
👉 Use: dot -Tpng -Gdpi=300 prediction_paths/prediction_20250525_155238.dot -o prediction_paths/prediction_20250525_155238.png
```

Figure 14: User Manual in Terminal

Only nodes (not arrows) on the prediction path are highlighted in red, providing a clear visual trace without clutter.

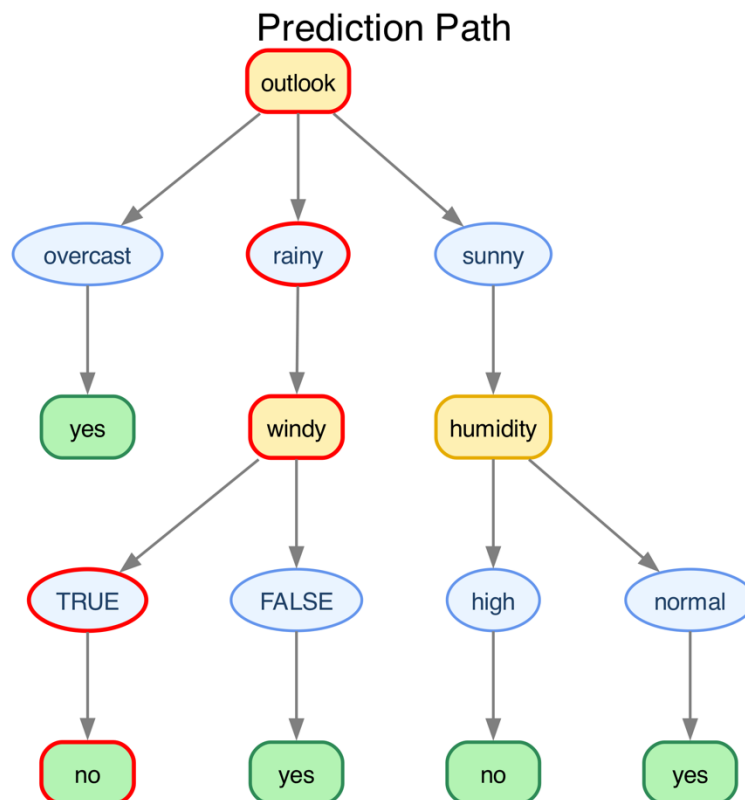


Figure 15: Prediction Path

## HANDLING CSV AND TXT FILES IN GO

To ensure broad compatibility and user flexibility, this project implements a robust mechanism for reading data from both .csv and .txt files. Go's standard libraries — such as encoding/csv, bufio, os, and strings — provide an efficient way to support a variety of data formats without requiring users to preprocess their files externally.

### File Parsing Strategy

The implemented logic allows the program to detect and correctly interpret files with different delimiters, even when the file type is ambiguous (e.g., a .txt file formatted like a CSV).

- **CSV Files**
  - For .csv files, the encoding/csv package is used, which by default expects a comma (,) as the delimiter. However, real-world datasets might use alternative delimiters such as:
    - Semicolon (;)
    - Tab (\t)
    - Dot (.)
    - Pipe (|)
  - The program intelligently **auto-detects** the most suitable delimiter by analyzing the header line and selecting the one that yields the highest field count.
- **TXT Files**
  - .txt files are treated similarly, but the lines are manually read using bufio.NewScanner(), and then split using the detected delimiter.
  - This ensures that even files without a formal structure (like plain text files with tab-separated values) can be processed accurately.

### Delimiter Detection Heuristic

The delimiter is automatically chosen based on a frequency count from the first line (usually the header). The one with the highest occurrence is assumed to be the actual delimiter. This method improves flexibility and avoids hardcoding specific formats.

### Post-Processing for Clean Data

Regardless of the file type or delimiter:

- Leading and trailing whitespace from each field is **trimmed** to ensure consistency.
- The header row is extracted for attribute labeling.
- The remaining lines are parsed into the data matrix.

### Benefits of This Approach

- **User-Friendly:** Users are not required to know or specify the delimiter.
- **Flexible Input:** Works with .csv and .txt formats, even when the content varies.
- **Error Resilience:** Provides error handling for malformed input or unsupported file structures.
- **Code Reuse:** Centralizes file reading into a single function for both formats.

## HANDLING CONFLICTING DATA (CAN'T DECIDE CASES)

During the dataset analysis, certain entries were identified **where all input attribute values were identical**, but the **corresponding outcomes differed**. These conflicting cases represent **inconsistencies** in the data that can hinder deterministic classification.

To address this, the system prompts the user to select a conflict resolution strategy before constructing the decision tree:

- **Option 1: Use "Can't decide"**

When selected, the decision tree **labels all conflicting cases as "Can't decide"**.

These nodes become leaf nodes and explicitly mark regions of ambiguity, ensuring that the model does not infer misleading outcomes.

- **Option 2: Use Most Common Outcome**

Alternatively, the user can instruct the algorithm to resolve conflicts by automatically selecting the **most frequent** outcome among the conflicting rows. This enables the tree to proceed with a majority-based decision in uncertain regions.

This flexible conflict-handling approach ensures the decision tree remains logically sound while adapting to different user preferences. It enhances both the **interpretability** and **practicality** of the model, especially when working with noisy or real-world datasets.



```

⚠ Conflict detected:
- Features: 40-49,premeno,25-29,0-2,no,2,right,left_low,no
- Outcome 'no-recurrence-events' at lines: [78]
- Outcome 'recurrence-events' at lines: [129 173]
⚠ Conflict detected:
- Features: 40-49,premeno,30-34,0-2,no,3,right,right_up,no
- Outcome 'recurrence-events' at lines: [128]
- Outcome 'no-recurrence-events' at lines: [181]
⚠ Conflict detected:
- Features: 30-39,premeno,0-4,0-2,no,2,right,central,no
- Outcome 'no-recurrence-events' at lines: [81]
- Outcome 'recurrence-events' at lines: [57]
⚠ Conflict detected:
- Features: 50-59,premeno,25-29,0-2,no,1,right,left_up,no
- Outcome 'recurrence-events' at lines: [168]
- Outcome 'no-recurrence-events' at lines: [228]
⚠ Conflict detected:
- Features: 40-49,premeno,20-24,0-2,no,2,left,left_low,no
- Outcome 'recurrence-events' at lines: [224]
- Outcome 'no-recurrence-events' at lines: [207]
⚠ Conflict detected:
- Features: 30-39,premeno,30-34,0-2,no,2,left,left_up,no
- Outcome 'no-recurrence-events' at lines: [108]
- Outcome 'recurrence-events' at lines: [151]

There are conflicts in the data (same inputs → different outcomes).
1. Use 'Can't decide' for conflicting inputs
2. Automatically choose the most common decision
Choose how to handle them (1/2): 2

```

Figure 16: Example of dataset option of how to handle these conflicts

## HOW TO RUN A GO PROGRAM AND WHY CHOOSE GO

### Running a Go Program

Running a Go program is straightforward and efficient, making it a great choice for building and testing applications quickly.

#### 1. Install Go

Download and install Go from the official site [golang.org](https://golang.org) for your operating system.

#### 2. Write Your Code

Create a .go file with your source code, for example main.go.

#### 3. Build or Run Directly

- To **run** the program without building a binary: go run main.go
- To **build** the program into an executable binary: go build main.go

Then run the generated executable (./main on Linux/macOS or main.exe on Windows).

#### 4. Manage Dependencies

Go uses modules (go.mod) to manage dependencies, simplifying project setup and version control.

#### Why Choose Go for This Project?

Go (or Golang) is more than just a programming language; it's a powerful tool that offers unique advantages, especially for projects involving data processing, server-side programming, and system tools.

#### Key Benefits:

- **Fast Compilation & Execution**

Go compiles directly to machine code, producing fast, efficient executables. This speeds up development cycles and runtime performance.

- **Simple & Clear Syntax**

Go's syntax is concise and easy to learn, reducing bugs and improving code readability—perfect for collaborative projects.

- **Built-in Concurrency**

Goroutines and channels enable simple, lightweight concurrent programming, helping to efficiently handle multiple tasks like file reading and processing.

- **Robust Standard Library**

Go comes with a rich standard library that supports file I/O, networking, encoding, and more — minimizing external dependencies.

- **Cross-Platform Support**

Compiled Go programs run on all major operating systems without modification.

- **Static Typing & Safety**

Strong typing helps catch errors at compile-time, increasing code reliability.

- **Easy Deployment**

Go produces standalone binaries that don't require runtime dependencies, simplifying deployment in various environments.

- **Great Tooling**

Built-in tools for formatting, testing, and profiling streamline development.

## **Beyond Syntax: Why Learning Go is Valuable**

- **Industry Demand**

Go is widely used by major companies (Google, Docker, Kubernetes, etc.) for cloud services, microservices, and networking tools.

- **Efficient Development**

Its combination of simplicity, speed, and powerful features lets you prototype and scale projects quickly.

- **Improved Problem Solving**

Go's focus on concurrency and simplicity teaches you to write clean, efficient, and scalable code—skills that transfer to many other languages and paradigms.

# CONCLUSION

This project transcended a mere exercise in implementing a decision tree algorithm; it evolved into a comprehensive exploration of fundamental machine learning principles and software development practices. By meticulously building the ID3 algorithm from the ground up, we developed an intimate understanding of core concepts such as entropy, information gain, and the recursive nature of decision tree construction. This hands-on approach allowed us to appreciate the mathematical underpinnings that guide the algorithm's decision-making process, beyond simply applying pre-built libraries or black-box solutions.

Implementing every step manually—from calculating entropy values and selecting the best attribute splits to constructing and pruning the tree—greatly enhanced both our theoretical knowledge and coding proficiency. This granular involvement ensured that we could troubleshoot, optimize, and extend the algorithm with confidence and clarity.

Choosing Go as the implementation language introduced an additional layer of challenge and growth. Go's minimalist syntax, strong typing, and built-in concurrency features not only improved the performance and maintainability of our code but also broadened our programming paradigm. The experience demonstrated Go's suitability for data processing and algorithmic applications, highlighting its advantages in speed, cross-platform support, and tooling ecosystem. This practical exposure to Go empowers us with a versatile skill set applicable in modern software and data engineering domains.

The resulting tool is a robust, user-friendly command-line application that supports:

- Flexible data ingestion from CSV or TXT files with automatic delimiter detection, enabling diverse dataset formats.
- Precise computation of entropy and information gain to guide attribute selection rigorously.
- Stepwise construction of an interpretable decision tree structure, which can be visualized via DOT files for graphical analysis.
- Interactive prediction capabilities, allowing users to input new data and obtain classification results in real-time.
- Graceful handling of invalid inputs and continuous user interaction until termination, enhancing usability.

Beyond achieving the technical milestones, this project exemplifies the synergy between algorithmic rigor, programming language mastery, and practical software engineering. It serves as a strong foundation for future explorations in machine learning, data science, and systems development, equipping us with both the knowledge and the tools to tackle more complex challenges.

In essence, this journey not only delivered a fully functional decision tree implementation but also cultivated a deeper appreciation for the interplay between theory and practice, and the power of choosing the right tools for real-world problems.

## References

- [1] J. R. Quinlan, “Induction of Decision Trees,” *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986, doi: 10.1007/BF00116251.
- [2] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann, 2011.
- [3] T. M. Mitchell, *Machine Learning*. New York, NY, USA: McGraw-Hill, 1997.
- [4] A. Sharma and A. Dey, “A comparative study of classification algorithms for decision tree,” *2012 International Conference on Advanced Computing & Communication Technologies*, Rohtak, India, 2012, pp. 426–430, doi: 10.1109/ACCT.2012.61.
- [5] N. B. Karayiannis and G. A. Mihailidis, “Decision Trees in Machine Learning,” *IEEE Potentials*, vol. 38, no. 2, pp. 29–34, Mar.-Apr. 2019, doi: 10.1109/MPOT.2018.2879561.
- [6] A. Alomari, S. Sulaiman, and R. Razali, “An Investigation on the Use of Decision Trees for Classification,” *2021 IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE)*, Penang, Malaysia, 2021, pp. 60–65, doi: 10.1109/ISCAIE51753.2021.9431840.
- [7] The Go Programming Language, “Go Documentation.” [Online]. Available: <https://golang.org/doc/>
- [8] Graphviz – Graph Visualization Software. [Online]. Available: <https://graphviz.org/>