DOKUZ EYLÜL UNIVERSITY

ENGINEERING FACULTY

DEPARTMENT OF COMPUTER ENGINEERING

CME 3402 CONCENPTS OF PROGRAMING LANGUAGES

Decision Tree with Java 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 are intuitive and powerful for solving classification problems by partitioning data based on attribute values. This project implements the **ID3** algorithm from scratch using **Java**, emphasizing manual calculation of entropy and information gain **without relying on external machine learning libraries.**

Key features of this project include:

- Reading input datasets from **CSV** and **TXT** files
- Manually calculating **entropy** and **information gain** at each split
- Recursively building the decision tree structure based on maximum information gain
- Displaying the tree hierarchy in the console output
- Predicting new instances interactively based on user input

This project enhances understanding of the ID3 decision tree mechanism and showcases Java's file handling, recursion, and interactive capabilities.

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:

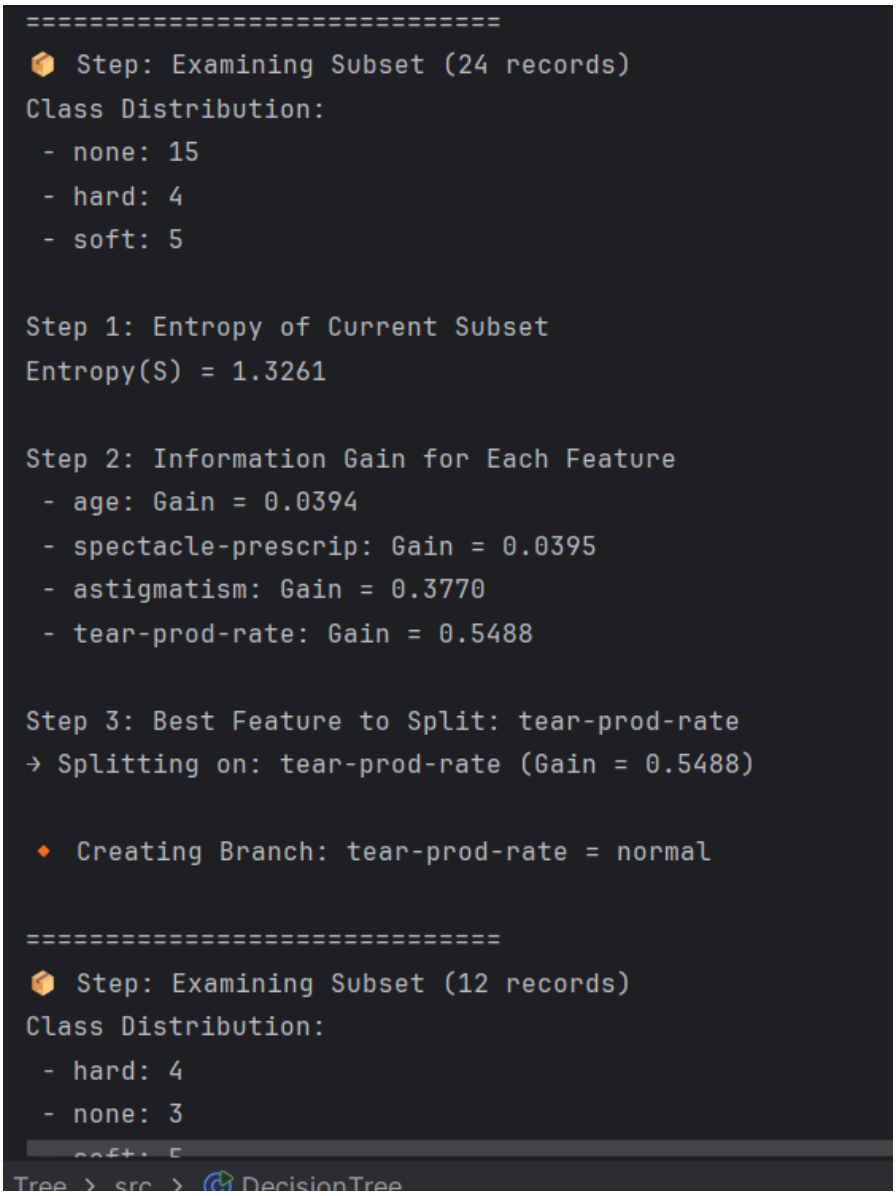

Figure 1: Program output showing entropy calculation for the age attribute. The entropy value of 1.3261 indicates a high level of impurity in the dataset prior to splitting.

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:

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

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
- $Values(A)$ is the set of all possible values for attribute A
- SvSv 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:

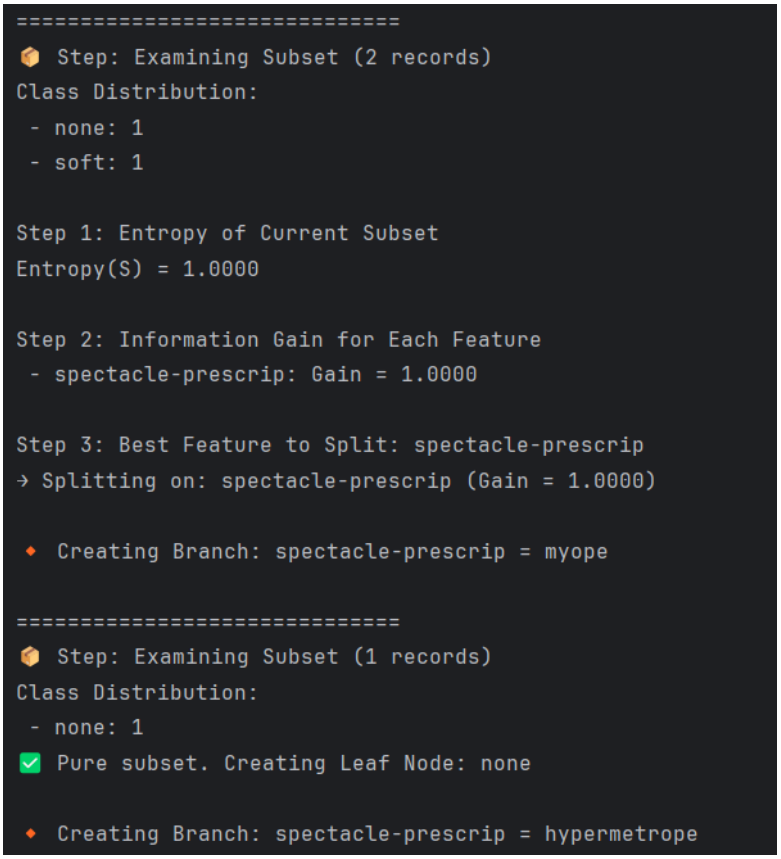

Figure 2: Information Gain calculation for the attribute spectacle-prescrip, demonstrating how splitting the dataset by attribute values myope and hypermetrope reduces entropy from 1.0000 to 0.0000, resulting in an information gain of 1.0000.

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.

Subset Formation: Partitioning the Dataset After Splitting

Once the attribute with the highest Information Gain is selected, the dataset is split into subsets based on the values of that attribute. Each subset represents a branch in the decision tree, containing only the data instances that match the corresponding value.

This step is essential because it narrows down the focus to more homogenous groups, reducing entropy and making classification more accurate in later steps.

Example from the Project:

The image below shows how the dataset is divided into subsets based on the attribute spectacle-prescrip. This is the chosen attribute due to its high Information Gain. The split creates two branches: one for "myope" and another for "hypermetrope".

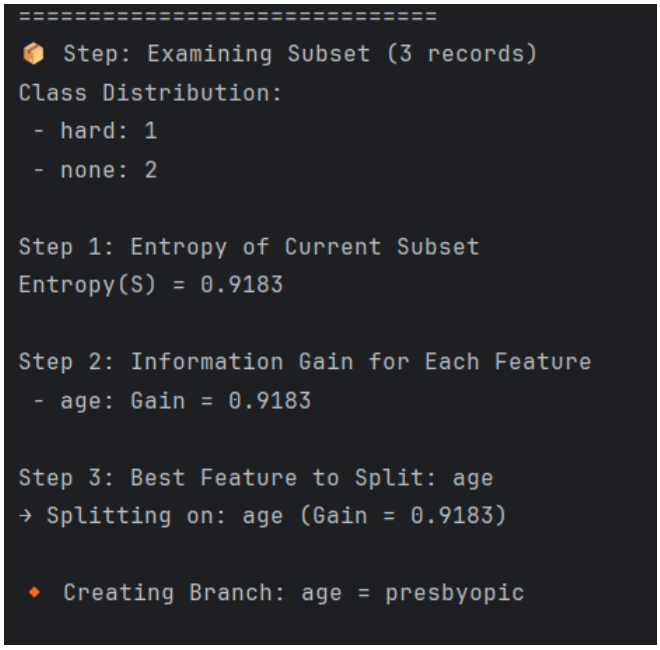

Figure 3: This figure illustrates the subset analysis of 3 records, where the class distribution is mixed (hard: 1, none: 2), resulting in an entropy of 0.9183. The feature 'age' yields the highest information gain and is selected for the next split, creating a new branch for the value 'presbyopic'.

Reaching Leaf Nodes: Final Classification Outcomes

After a series of splits, if all instances in a subset belong to the same class or no further gain can be achieved, a leaf node is created. Leaf nodes are the terminal points in the decision tree and represent the final classification outcome for a specific path through the tree.

This marks the completion of that decision path, and no further splitting is performed on that subset.

Example from the Project:

In the image below, each 📌 symbol marks a leaf node, showing the final decision made by the tree based on previous attribute splits. For example, the leftmost branch terminates in a classification of "no".

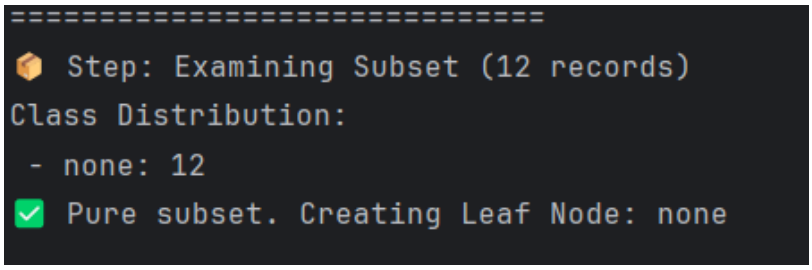

Figure 4: This figure shows a subset of 12 records with a uniform class distribution (none: 12), resulting in zero entropy. Since the data is pure, no further splitting is needed, and a leaf node is created with the classification "none".

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. **Entropy Calculation:**
   The method calculateEntropy computes the entropy of the current dataset subset, measuring the impurity of the target classification. This is done by counting occurrences of each class label and applying the entropy formula.
2. **Information Gain Calculation:**
   For each candidate attribute, calculateInformationGain is called to evaluate how effectively splitting by that attribute reduces entropy. The code partitions the dataset by each attribute's possible values, computes the weighted entropy of the subsets, and subtracts it from the original entropy to obtain the Information Gain.
3. **Selecting the Best Attribute:**
   The algorithm iterates over all remaining attributes (excluding previously used ones) to find the attribute with the highest Information Gain, which maximizes the reduction in uncertainty.Create a Decision Node based on the selected attribute.
4. **Node Creation:**
   A new TreeNode is created representing the decision node for the selected attribute. If the dataset is pure (all examples have the same label), the node is a leaf containing the class label.
5. **Partitioning and Recursion:**
   The dataset is split into subsets according to the selected attribute's values. The algorithm recursively calls buildTree on each subset with the reduced set of attributes, building subtrees until stopping conditions are met.
6. **Stopping Criteria:**
   Recursion terminates when:
   - All instances in a subset have the same classification (pure subset).
   - There are no remaining attributes to split on.

- The dataset subset is empty, in which case the most common class from the parent subset is assigned.

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).

Handling Missing or Conflicting Data:

The implementation proactively addresses instances of conflicting or ambiguous data—cases where identical attribute combinations yield differing class labels. Prior to tree construction, users can specify a resolution strategy: either labeling such nodes as “Can’t decide” to explicitly represent ambiguity or defaulting to the most common class label to enforce deterministic outcomes. This design choice ensures the decision tree remains logically coherent and interpretable even when faced with noisy or inconsistent datasets

Optimization and Model Parsimony:

By rigorously excluding attributes that do not contribute to entropy reduction and halting recursion upon reaching pure nodes, the algorithm avoids overfitting and produces a concise, generalizable decision tree. This selective splitting enhances model interpretability and computational efficiency without compromising predictive performance.

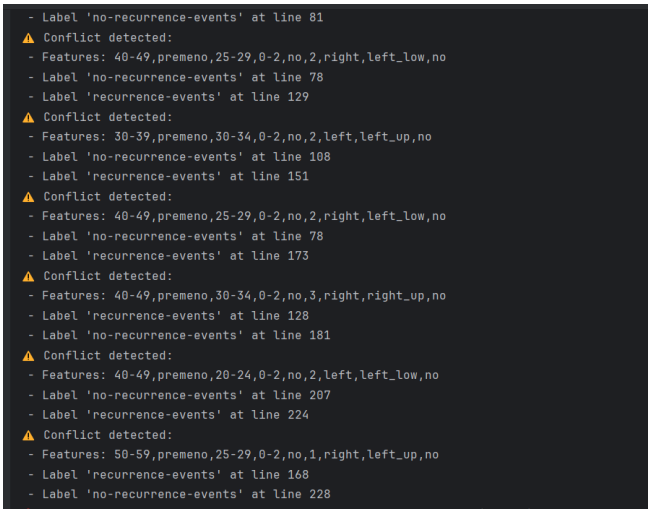

Figure 5: Conflict detection during tree construction. The system identifies samples with identical features but conflicting labels, prompting the user to resolve before continuing.

- Final Decision Tree Text-Based format in terminal:

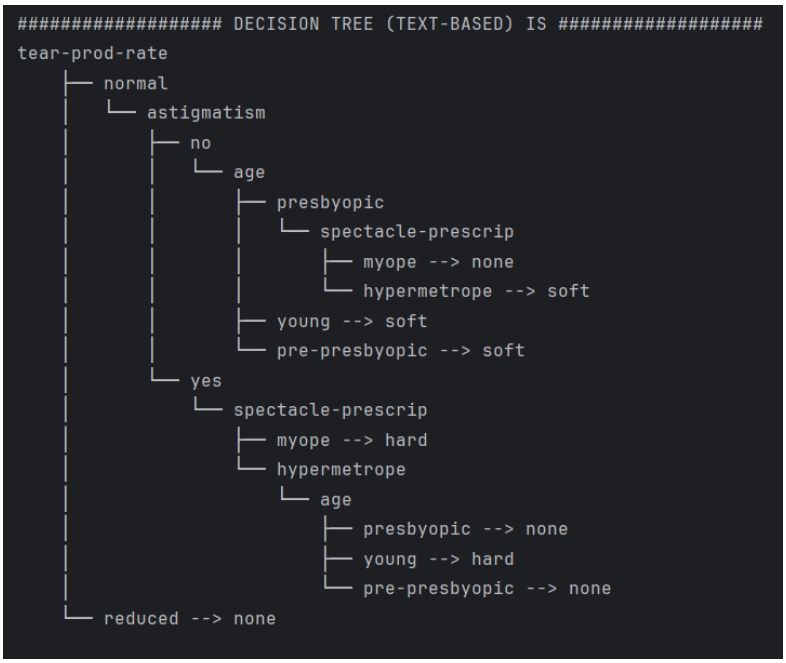

Figure 6: Final decision tree structure generated using the ID3 algorithm. Each node represents a decision point based on an attribute, and each leaf node (📌) indicates the final classification outcome. The tree reflects the step-by-step manual construction derived from entropy and information gain calculations.

VISUALIZING THE DECISION TREE WITH Graphviz

After the decision tree has been constructed by the Java implementation of the ID3 algorithm, it is exported as a DOT file (Tree.dot). This file is a plain text representation of the tree structure formatted according to the Graphviz DOT language, which enables efficient visualization of hierarchical data.

To generate a graphical representation of the decision tree, users utilize Graphviz's dot command-line tool. This tool converts the DOT file into commonly used image formats such as PNG or SVG, allowing for easy inspection, analysis, and inclusion in reports or presentations.

Tree Visualization Examples

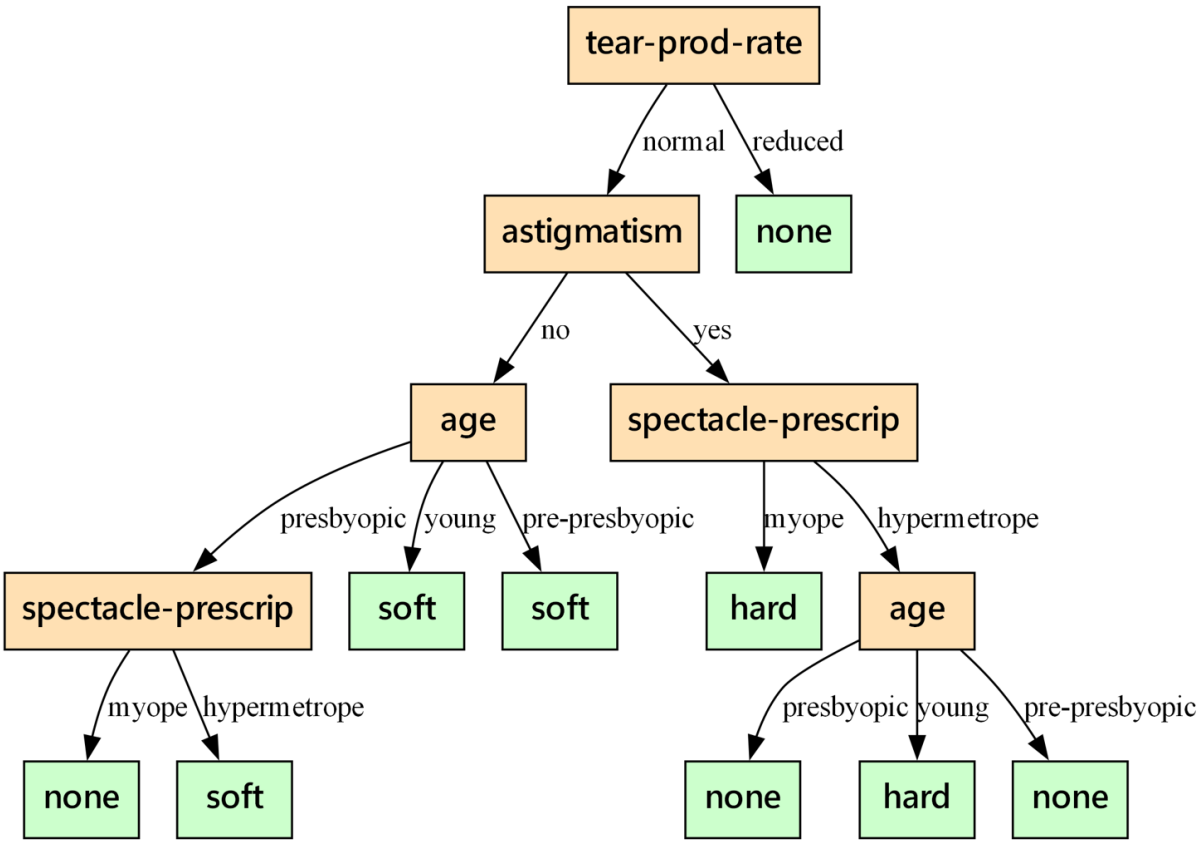

Figure 7: Visualization of a decision tree of moderate size rendered using Graphviz with standard DPI. Each node corresponds to a decision point, and leaf nodes show the final classification outcomes.

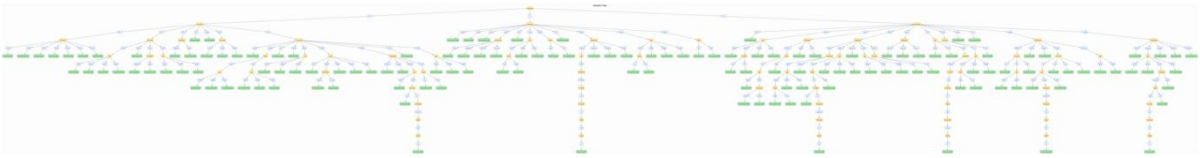

Figure 8: Visualization of a larger decision tree with scaled rendering to improve readability. The use of -Gscale=2 helps accommodate deeper trees with more branches

SAVING THE DECISION TREE & CREATING VISUALS

Upon completion of the decision tree construction using the ID3 algorithm, the tree is programmatically exported as a DOT file (highlighted_tree.dot). This file is systematically saved within a dedicated directory named decision_tree/.

This organized folder structure promotes efficient management and retrieval of tree files, especially when working with multiple datasets or generating numerous visualizations. By maintaining a clear separation of output files, users can effortlessly navigate and utilize the saved decision trees for further analysis, visualization, or reporting.

Folder Structure

decision_tree/
└── highlighted_tree.dot

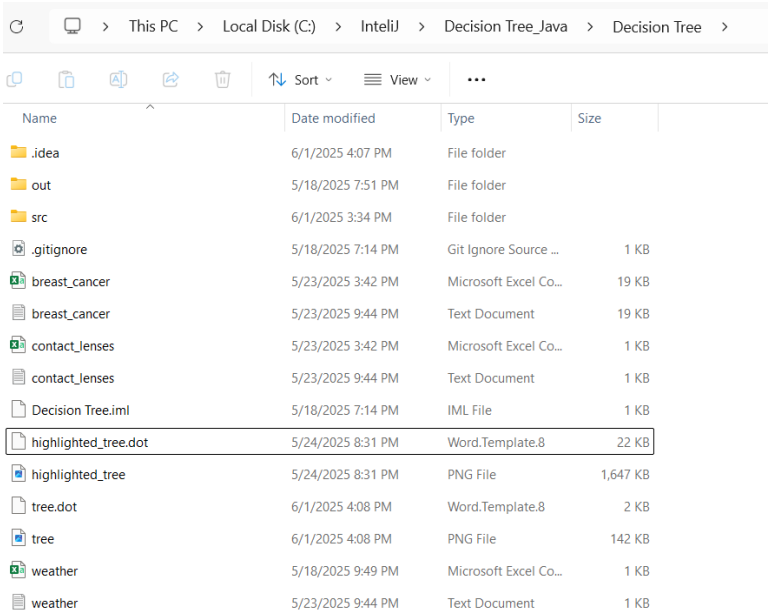

Figure 9: Folder Structure

PREDICTION USING THE DECISION TREE

Once the decision tree is constructed using the ID3 algorithm, it serves as an effective classifier capable of predicting the class labels for new, unseen instances. Prediction is performed by recursively traversing the tree from the root node to a leaf node, using the input attribute values to guide the path—much like answering a series of hierarchical questions.

How Prediction Works

1. Start at the Root: **The traversal begins at the root node of the decision tree.**
2. Evaluate Split Attribute: **At each internal node, the algorithm inspects the attribute used for splitting at that node.**
3. Follow Matching Branch: **Based on the input instance's attribute value, the traversal follows the corresponding branch.**
4. Recursive Descent: **This process repeats recursively, moving down the tree until a leaf node is reached.**
5. Output Class Label: **The class label stored in the leaf node is returned as the prediction for the input instance.**

In cases where an input value does not match any branch (i.e., a value unseen during training), the algorithm returns a default output such as "Unknown" to indicate uncertainty.

Interactive Prediction in the CLI Application

The program's main method provides an interactive command-line interface (CLI) that enables users to:

- Select from multiple datasets (e.g., weather.csv, contact_lenses.csv, or breast_cancer.csv).
- View the structure of the constructed decision tree.
- Input attribute values sequentially for classification.
- Receive immediate prediction results based on the entered data.

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

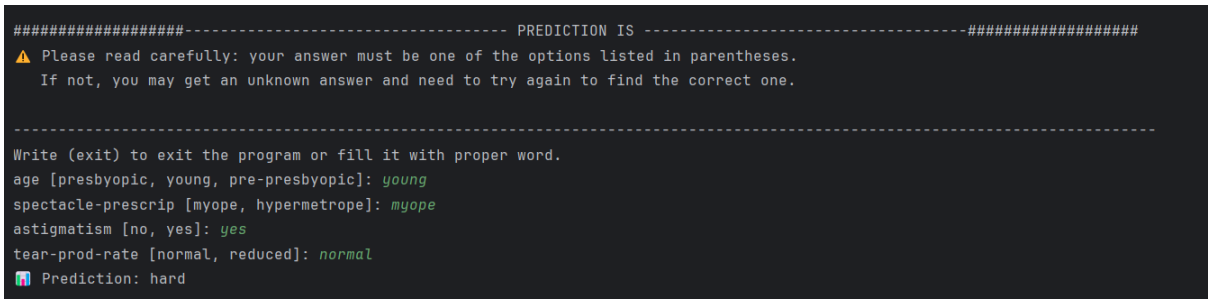

Figure 10: Interactive prediction example using the constructed decision tree. The user is prompted to enter attribute values one by one, and the system outputs the predicted decision based on the input. This process is executed in a loop, continuously accepting new predictions until the user types "exit"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

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

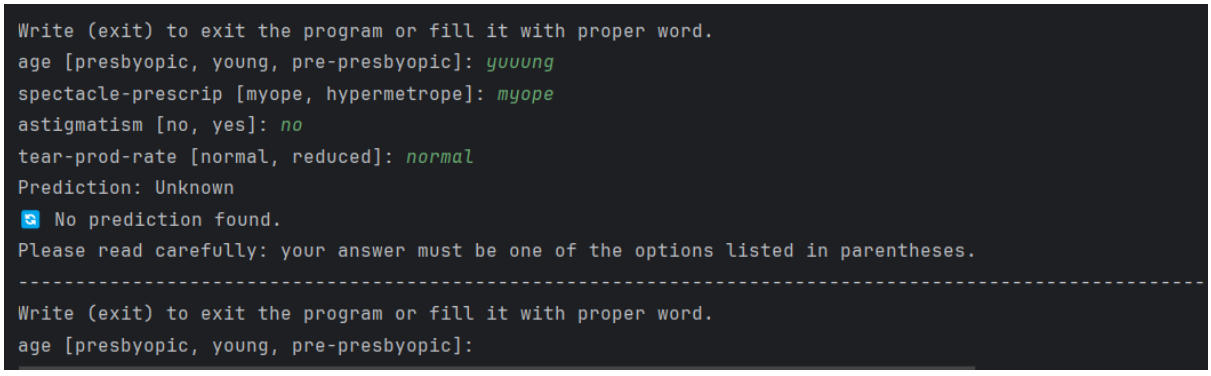

Figure 11: User input validation during prediction. The program checks each input value against the structure of the constructed decision tree. If a value does not exist in the tree (e.g., an unexpected branch), the system throws an error and reports it as an incorrect input. However, if the input is valid but leads to a non-matching path, the prediction proceeds correctly without interruption.

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

```
Write (exit) to exit the program or fill it with proper word.
age [presbyopic, young, pre-presbyopic]: exit
Exiting...

Process finished with exit code 0
```

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 This interactive prediction capability elevates the decision tree from a static data structure to a practical, real-world classifier by:

- Demonstrating the efficacy of the ID3 algorithm in handling categorical data.
- Providing transparent and interpretable decision paths easily understood by users.
- Showcasing how Java can be leveraged to develop performant and user-friendly machine learning tools without relying on external dependencies.

PREDICTION PATH VISUALIZATION

To further aid interpretability, each prediction's traversal path from root to leaf is recorded and exported as a DOT file. In the generated visualizations, only the nodes along the prediction path are highlighted (typically in red), clearly tracing the decision steps without cluttering the graph.

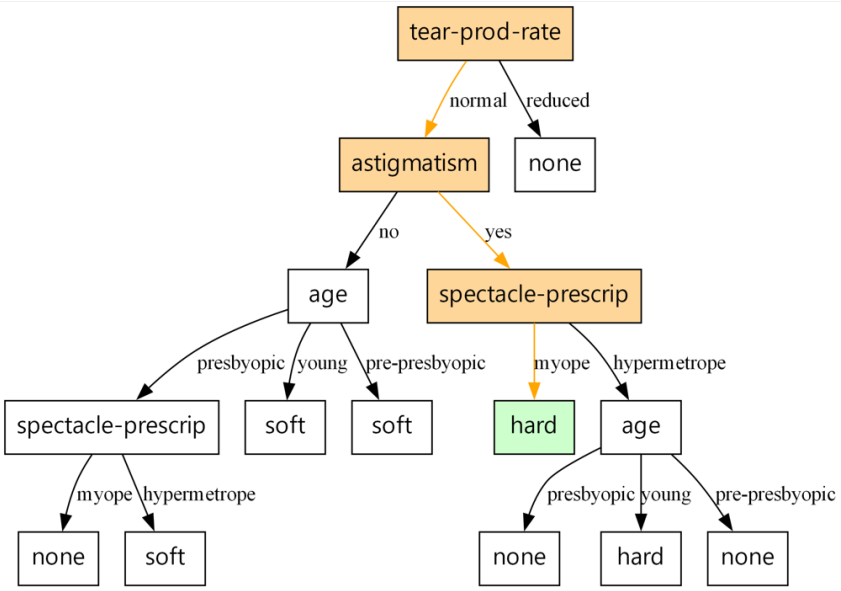

Figure 13: Predicition Path

HANDLING CSV AND TXT FILES IN JAVA

To maximize usability and ensure compatibility with diverse data sources, the project incorporates a robust file-reading mechanism capable of processing both .csv and .txt files. Leveraging Java's standard I/O libraries—such as java.io.BufferedReader, java.io.FileReader, and java.util.StringTokenizer—the implementation supports a wide range of data formats without requiring manual preprocessing by the user.

File Parsing Strategy

The program intelligently detects and interprets files with varying delimiters, accommodating datasets that may not strictly conform to the .csv standard:

- **CSV Files**
  - .csv files are parsed by reading lines and splitting them using an auto-detected delimiter. Although commas (,) are the default delimiter, the program recognizes alternative delimiters commonly found in real-world datasets, including:
    - Semicolon (;)
    - Tab (\t)
    - Dot (.)
    - Space ( )

    Delimiter detection is performed by analyzing the header line to identify which delimiter yields the highest number of fields, thereby selecting the most appropriate separator automatically.

- **TXT Files**

  - .txt files are handled similarly, with lines read manually and then split based on the detected delimiter. This approach allows processing of loosely structured files such as tab-separated text files without requiring special user intervention.

Delimiter Detection Heuristic

A simple yet effective heuristic determines the delimiter by counting the frequency of candidate characters in the header line. The delimiter with the highest occurrence is chosen, enhancing flexibility and avoiding assumptions about the input format.

Post-Processing for Clean Data

After reading and splitting the data, the program performs several cleaning steps to maintain data integrity:

- Trims leading and trailing whitespace from all fields.
- Extracts the first line as the attribute header.
- Parses the remaining lines into a structured data matrix.

Benefits of This Approach

- User-Friendly: **No need for users to specify delimiters or preprocess files.**
- Flexible Input Support: **Works seamlessly with .csv and .txt files, even with varied formatting.**
- Robustness: **Incorporates error handling for malformed or inconsistent input files.**
- Code Reusability: **Consolidates file reading logic into a unified process, reducing redundancy.**

HANDLING CONFLICTING DATA (CAN'T DECIDE CASES)

During dataset preprocessing, some records were identified where multiple entries shared identical input attribute values but differed in their corresponding output labels. These conflicts pose challenges for deterministic classification and can degrade the decision tree's reliability.

To address this issue, the system provides the user with two configurable conflict-resolution strategies prior to tree construction:

- **Option 1: Label as "Can't Decide"**

  When selected, all conflicting cases are assigned a special class label—"Can't decide." These leaf nodes explicitly mark ambiguous regions in the data where the model cannot confidently infer a single outcome, preserving the integrity and interpretability of the decision tree.

- **Option 2: Majority Class Resolution**

  Alternatively, the user can opt for automatic conflict resolution by assigning the most frequent outcome among the conflicting entries. This majority-vote approach

allows the tree to continue growing by prioritizing the prevalent class, facilitating practical decision-making despite data noise.

This flexible conflict-handling mechanism enhances the model's adaptability, making it suitable for real-world datasets that often contain inconsistencies or labeling errors. It improves both the accuracy and explanatory power of the resulting decision tree by allowing users to tailor how ambiguous data is treated.

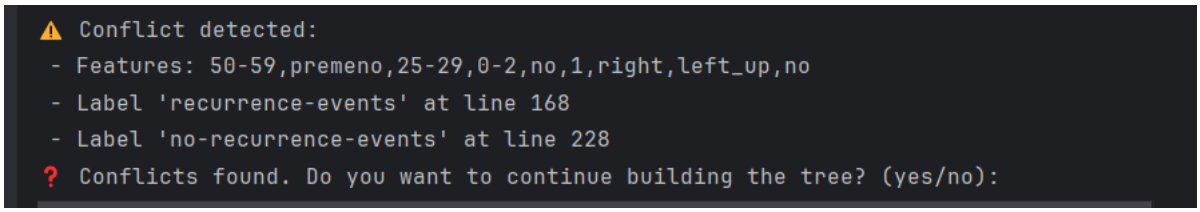

Figure 14: Example of dataset option of how to handle these conflicts

RUNNING A JAVA PROGRAM AND WHY JAVA WAS CHOSEN

Running the Program in IntelliJ IDEA

Java, combined with IntelliJ IDEA, offers a robust, developer-friendly environment for building and testing complex algorithms like the ID3 decision tree. Running the application within IntelliJ is straightforward and efficient.

Steps to Run the Project:

1. **Install Java Development Kit (JDK)**
   Download and install the latest JDK from
   https://www.oracle.com/java/technologies/javase-downloads.html or use OpenJDK.
2. **Set Up IntelliJ IDEA**
   Open IntelliJ IDEA and either:
   - Create a new Java project and import your .java source files, or
   - Open an existing project directory.
3. **Configure the SDK**
   Ensure the correct JDK is selected via:
   File → Project Structure → Project SDK

4. **Build the Project**
   IntelliJ will automatically compile the project on save. Alternatively, go to:
   Build → Build Project to compile manually.
5. **Run the Program**
   Right-click on the main() method (typically in Main.java) and select:
   Run 'Main'.
   You can also create a run configuration to specify command-line arguments or working directories.
6. **Handling Dependencies**
   If your project includes external libraries, use IntelliJ's built-in support for Maven or Gradle to manage dependencies efficiently. For standard projects, Java's built-in packages are often sufficient.

Why Choose Java for This Project?

Java is a well-established, versatile, and reliable programming language, making it an excellent choice for implementing machine learning algorithms like ID3. Its rich ecosystem and strong tooling support simplify the development of data-centric, interactive command-line applications.

Key Advantages of Java:

1. **Object-Oriented Design**
   Java's strong OOP paradigm makes it easier to model components such as DecisionTreeNode, Dataset, and TreeBuilder using classes and inheritance.
2. **Cross-Platform Compatibility**
   Java programs run on any OS with a Java Virtual Machine (JVM), ensuring portability.
3. **Rich Standard Library**
   Java includes robust libraries for file I/O (java.io, java.nio), data structures (java.util.*), and string processing, which are essential for reading datasets, parsing files, and building trees.
4. **Strong Typing & Compile-Time Safety**
   Java's static type system helps catch errors early, reducing runtime bugs and improving code quality.

5. **IDE Support**
   IntelliJ IDEA, one of the most powerful Java IDEs, offers features like code completion, refactoring tools, integrated debugging, and version control—all of which speed up development and debugging.
6. **Community & Documentation**
   Java has extensive documentation, community support, and mature learning resources—ideal for both beginners and experienced developers.

Why Java Is a Practical Choice for Decision Tree Projects

- **Readable & Maintainable Code:** Java's clear syntax and modular structure promote maintainable code, which is important for recursive algorithms like ID3.
- **Efficient Development Workflow:** Tools like IntelliJ simplify iteration and testing.
- **Reusability**: Core logic (e.g., entropy calculation, tree traversal, prediction) can be encapsulated and reused across different datasets.

CONCLUSION

This project extended far beyond the technical implementation of the ID3 algorithm—it served as a deep dive into the core principles of decision tree learning and the disciplined software engineering practices necessary to realize such an algorithm from scratch. By choosing not to rely on external libraries or machine learning frameworks, we were able to directly engage with foundational concepts like entropy, information gain, and recursive tree construction. This hands-on experience brought clarity to how theoretical ideas translate into practical, working code.

Implementing the decision tree manually in Java allowed us to reinforce both mathematical intuition and coding proficiency. We developed each core component of the algorithm—from entropy calculation and attribute selection to data partitioning and tree generation—through rigorous logic and structured abstraction. This direct control over every step of the process enabled us to debug effectively, extend functionality with precision, and ensure the model's correctness. Rather than treating the algorithm as a black box, we understood the intricate mechanics that drive its decision-making behavior.

Choosing Java as the programming language provided several advantages that enhanced the project's structure and maintainability. Java's object-oriented design enabled us to modularize the algorithm cleanly, defining distinct classes and methods for dataset handling, tree nodes, and user interaction. Its static typing system and powerful standard libraries improved both reliability and performance, particularly for tasks involving file I/O, string processing, and dynamic user input. Additionally, IntelliJ IDEA's tooling environment greatly facilitated development, offering real-time feedback, automatic refactoring, and robust debugging capabilities.

One of the notable accomplishments of this project was the implementation of flexible data ingestion. The system supports both .csv and .txt files with intelligent delimiter detection, making it adaptable to real-world datasets. Furthermore, we introduced a user-driven resolution strategy for handling conflicting data entries, ensuring that the model remains logical even when the training data contains inconsistencies. These additions reflect a strong focus on usability, flexibility, and robustness.

The application also includes an interactive command-line interface, allowing users to input new data and receive instant predictions. The decision tree can be visualized by exporting its structure to a DOT file, which can be rendered using tools like Graphviz. This visual capability enhances interpretability and provides a valuable aid for analysis and presentation. The ability to trace prediction paths and identify decision boundaries within the tree further demonstrates the clarity and transparency of the model.

Ultimately, this project demonstrates the power of combining algorithmic understanding with deliberate software engineering. By constructing a functional and user-friendly decision tree classifier entirely in Java, we not only gained practical machine learning experience but also honed our skills in writing maintainable, scalable, and efficient code. This work forms a strong foundation for future exploration in areas such as ensemble methods, decision tree pruning, and real-time classification systems. It exemplifies the importance of deeply understanding the tools and methods we use, and of choosing technologies that align with the goals of both the system and its users.

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] Oracle, *The Java™ Tutorials: Working with CSV Files and Streams*. [Online]. Available: https://docs.oracle.com/javase/tutorial/essential/io/

[8] Graphviz – Graph Visualization Software. [Online]. Available: https://graphviz.org/

[9] B. Lutz, *Java for Data Science*, Birmingham, UK: Packt Publishing, 2016.

[10] A. Mohan, “Decision Tree Implementation in Java from Scratch,” *Towards Data Science*, [Online]. Available: https://towardsdatascience.com (Accessed: May 2025).

[11] IntelliJ IDEA Documentation, “Running and Debugging Java Applications.” [Online]. Available: https://www.jetbrains.com/help/idea/running-applications.html