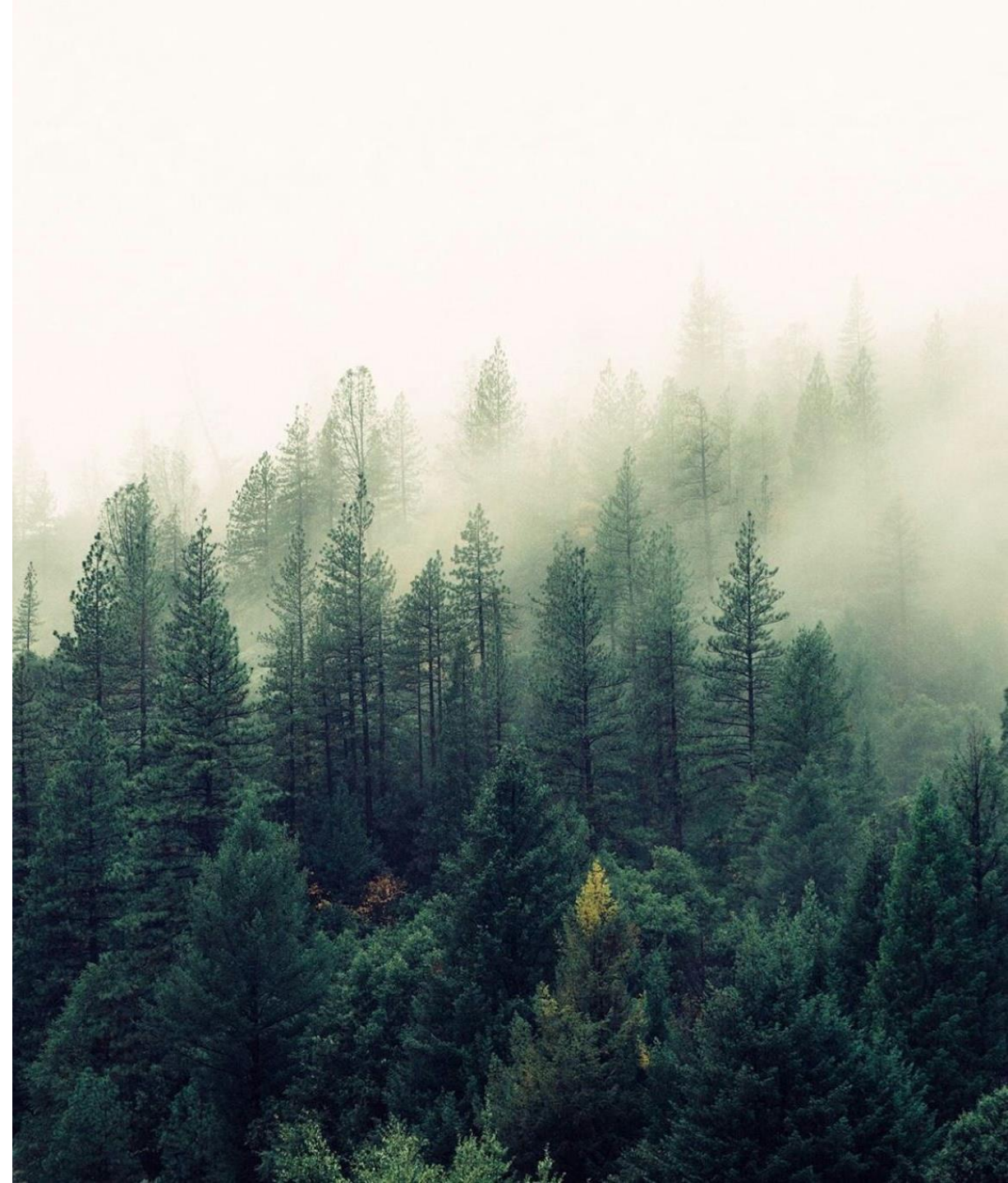


---

# RANDOM FOREST CLASSIFIER IMPLEMENTATION

- Python Programming Final Project
- Authors:
- Jakub Łudzik & Filip Żurek



---

# THEORETICAL BACKGROUND (ALGORITHM)

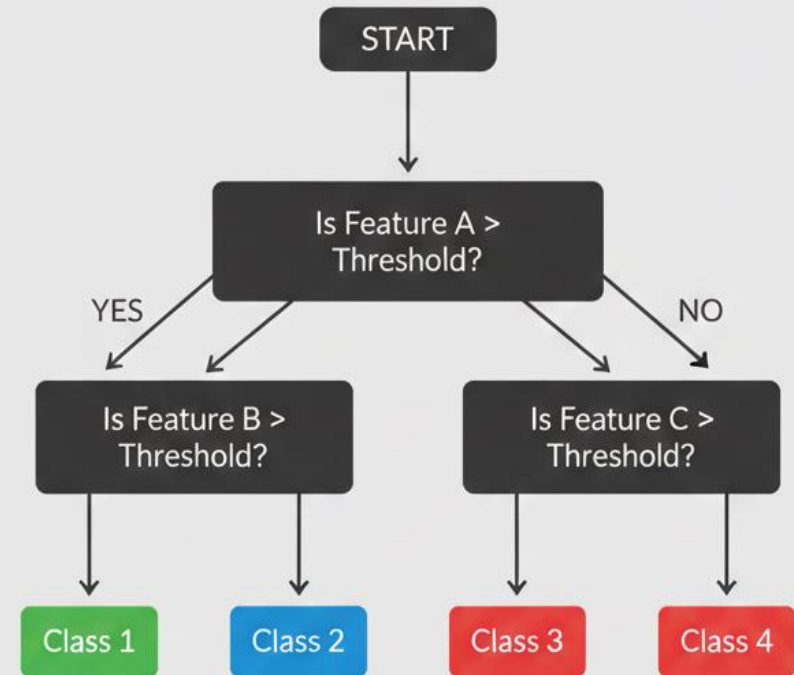
**Select Random Data:** The algorithm takes your main dataset and randomly selects different samples (small parts) of the data to create many separate groups.

**Build Many Trees:** It builds a separate Decision Tree for each of these groups. Each tree learns from its specific data and looks at a random set of features.

**Make Predictions:** When you have new data to test, the algorithm asks every single tree to make a prediction independently.

**Count the Votes:** Finally, the algorithm combines all the results.

---



DECISION TREE ALGORITHM

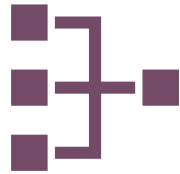
---

# IMPLEMENTATION DETAILS



## Tech Stack:

Python 3.13,  
NumPy,  
Pandas.



## Core Classes:

*Node*: Stores feature index, threshold, and value.

*DecisionTree*: Handles fit() (recursive growth) and predict().

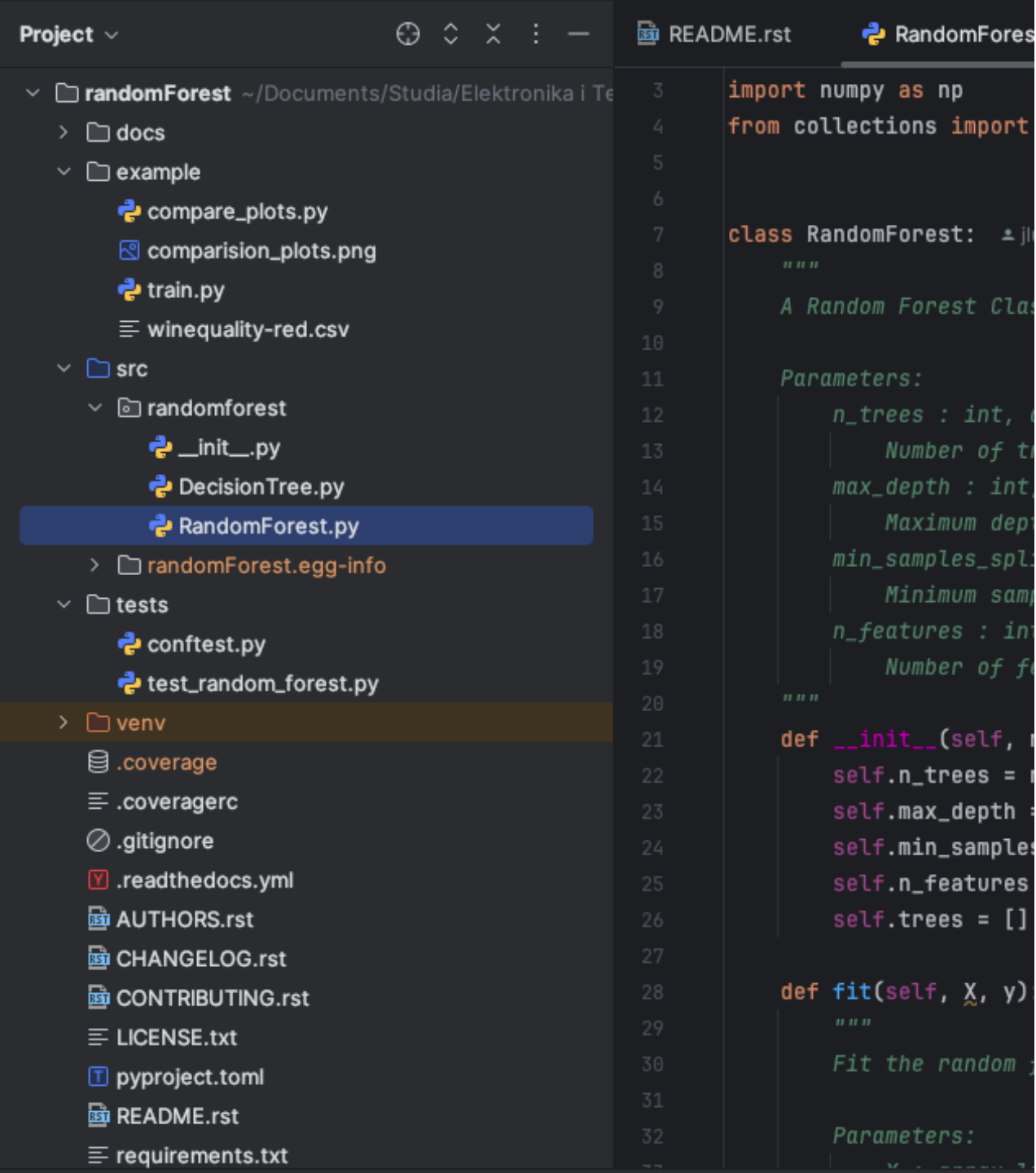
*RandomForest*: Manages the ensemble of trees.



## Key Metrics:

Shannon Entropy,  
Information Gain.

---



---

# SOFTWARE ENGINEERING & TOOLS



**Project Structure:**  
Generated via  
**PyScaffold**.



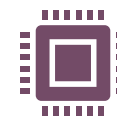
**Environment:**  
VirtualEnv  
managed  
dependencies.



**Testing:** Unit tests  
with **pytest**.



**Documentation:**  
Auto-generated  
HTML with  
**Sphinx**.



**CI/CD & Hygiene:** Clean  
repo, `.gitignore`,  
formatted code.

---

---

# DOCUMENTATION



Full API Reference  
generated from docstrings.



Hosted on **GitHub Pages**.



Includes installation guide  
and usage examples.

The screenshot shows a web browser with two tabs: 'jludzik/randomForest' and 'randomForest — randomFore'. The address bar shows 'jludzik.github.io/randomForest/'. The page title is 'randomForest'. Below the title is a search bar with a 'Go' button. A 'Navigation' sidebar on the left lists: Overview, Contributions & Help, License, Authors, Changelog, and Module Reference. The main content area starts with the text 'This is the documentation of **randomForest**.' followed by a 'Note:' section. The note explains that the documentation is formatted in reStructuredText and provides instructions on how to add additional pages and link them. It also mentions that the documentation can refer to other Python packages and that the 'autodoc' extension is activated by default. At the bottom, there is a 'Contents' section with a list of links: Overview, Authors, Project Description, Requirements and Installation, Usage, Contributions & Help, and Contribution Policy.

**randomForest**

Search  Go

**randomForest**

This is the documentation of **randomForest**.

**Note:**

This is the main page of your project's [Sphinx](#) documentation. It is formatted in [reStructuredText](#). Add additional pages by creating rst-files in `docs` and adding them to the [toctree](#) below. Use then [references](#) in order to link them from this page, e.g. [Contributors](#) and [Changelog](#).

It is also possible to refer to the documentation of other Python packages with the [Python domain syntax](#). By default you can reference the documentation of [Sphinx](#), [Python](#), [NumPy](#), [SciPy](#), [matplotlib](#), [Pandas](#), [Scikit-Learn](#). You can add more by extending the `intersphinx_mapping` in your Sphinx's `conf.py`.

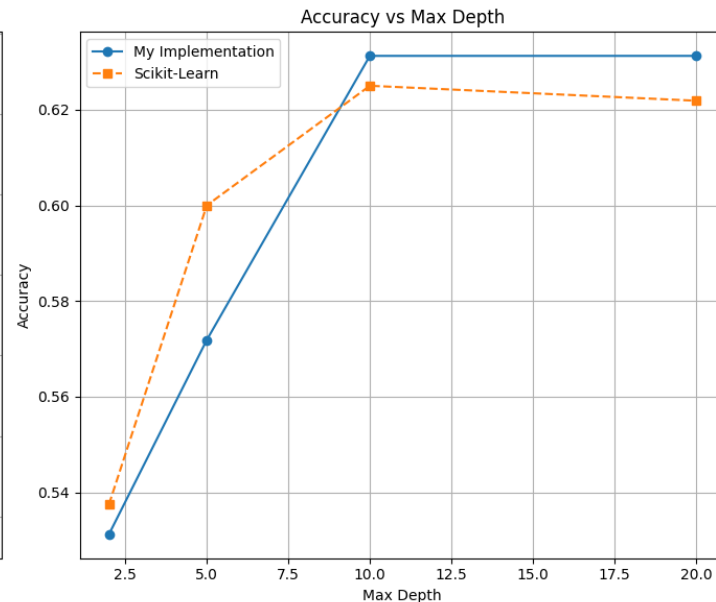
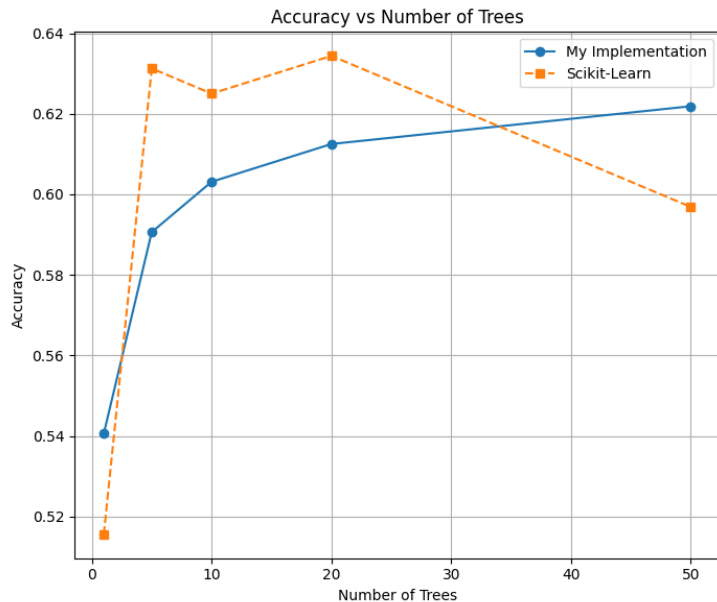
The pretty useful extension [autodoc](#) is activated by default and lets you include documentation from docstrings. Docstrings can be written in [Google style](#) (recommended!), [NumPy style](#) and [classical style](#).

**Contents**

- [Overview](#)
  - [Authors](#)
  - [Project Description](#)
  - [Requirements and Installation](#)
  - [Usage](#)
- [Contributions & Help](#)
  - [Contribution Policy](#)

# RESULTS & COMPARISON (SCIKIT-LEARN)

## PARITY CHECK: OURS VS. SCIKIT-LEARN



**Accuracy:** Comparable results (differences < 1-2%).



**Convergence:** Both models improve with depth and tree count.



**Performance:** Scikit-Learn is faster (C-optimized optimizations) vs. Pure Python.

# SUMMARY & FUTURE WORK

---

- **Goal achieved:** Fully functional classifier built from scratch.
- **Learnings:** Deep understanding of entropy and ensemble methods.
- **Future improvements:**
  - Parallel processing (multiprocessing) to speed up training.
  - Support for regression (Random Forest Regressor).
  - Pruning (tree trimming) to prevent overfitting.