
RANDOM FOREST CLASSIFIER IMPLEMENTATION

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



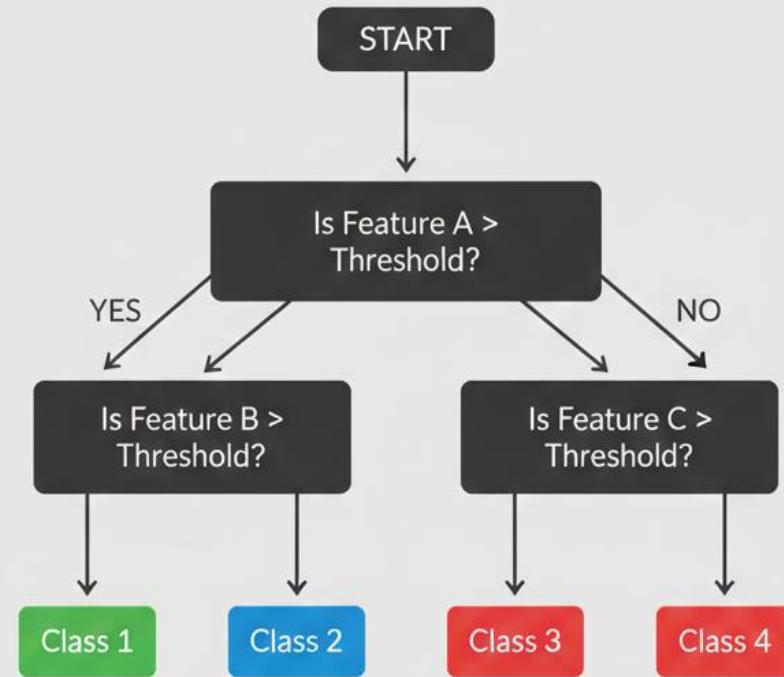
THEORETICAL BACKGROUND (ALGORITHM)

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.

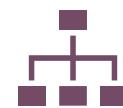
Project ▾

randomForest ~ /Documents/Studia/Elektronika i Te
docs
example
compare_plots.py
comparision_plots.png
train.py
winequality-red.csv
src
randomforest
__init__.py
DecisionTree.py
RandomForest.py
randomForest.egg-info
tests
conftest.py
test_random_forest.py
venv
.coverage
.coveragerc
.gitignore
.readthedocs.yml
AUTHORS.rst
CHANGELOG.rst
CONTRIBUTING.rst
LICENSE.txt
pyproject.toml
README.rst
requirements.txt

README.rst RandomFores

```
3 import numpy as np
4 from collections import
5
6
7 class RandomForest: __jl
8 """
9     A Random Forest Class
10
11     Parameters:
12         n_trees : int,
13             Number of trees in the forest.
14         max_depth : int,
15             Maximum depth of the individual decision trees.
16         min_samples_split : int,
17             Minimum number of samples required to split an internal node.
18         n_features : int,
19             Number of features to consider when looking for the best split.
20
21     def __init__(self, n_trees=100, max_depth=10, min_samples_split=2, n_features=10):
22         self.n_trees = n_trees
23         self.max_depth = max_depth
24         self.min_samples_split = min_samples_split
25         self.n_features = n_features
26         self.trees = []
27
28     def fit(self, X, y):
29         """
30             Fit the random forest.
31
32             Parameters:
```

SOFTWARE ENGINEERING & TOOLS



Project Structure:
Generated via
PyScaffold.



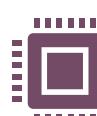
Environment:
VirtualEnv
managed
dependencies.



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.

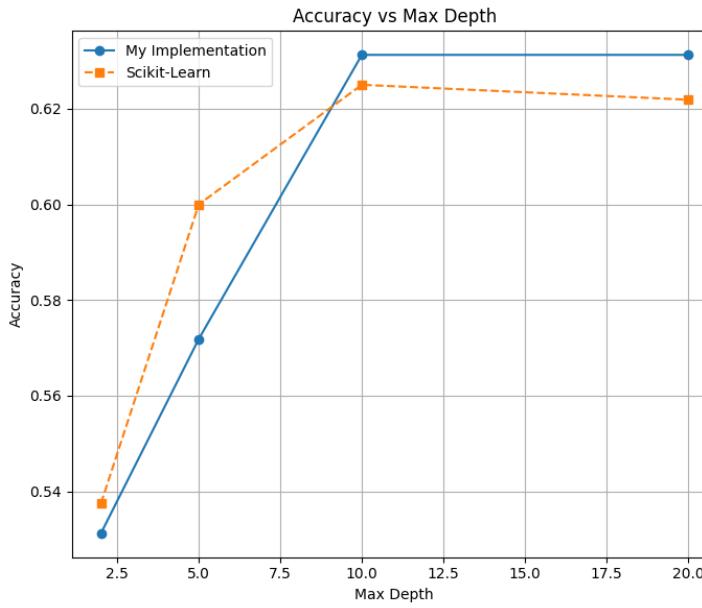
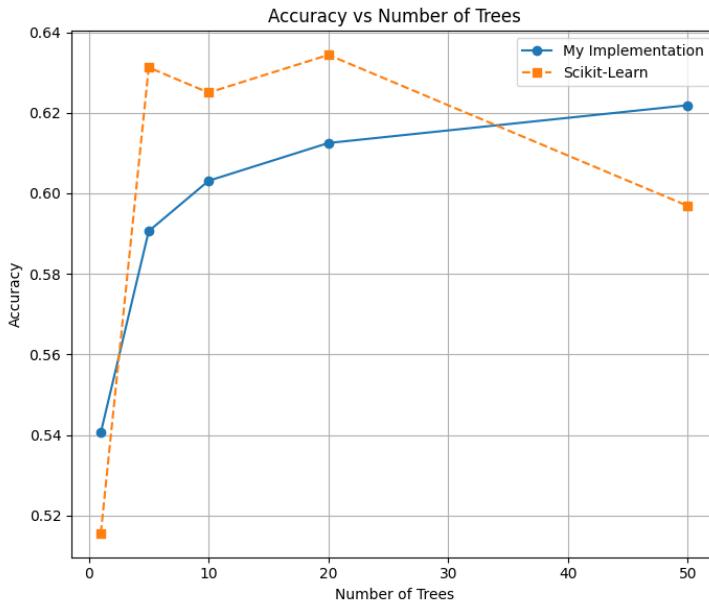
The screenshot shows a web browser displaying the documentation for a project named 'randomForest'. The URL in the address bar is `jiudzik.github.io/randomForest/`. The page title is 'randomForest' and the subtitle is 'This is the documentation of **randomForest**'. On the left, there is a 'Navigation' sidebar with links to 'Overview', 'Contributions & Help', 'License', 'Authors', 'Changelog', and 'Module Reference'. The main content area contains a 'Note' section which explains that this is the main page of the Sphinx documentation, formatted in reStructuredText, and provides instructions for adding more pages. It also mentions the possibility of referring to other Python packages using Python domain syntax. Below the note, there is a section about the `autodoc` extension being activated by default, which includes documentation from docstrings. The style can be written in Google style (recommended!), NumPy style, or classical style. At the bottom, there is a 'Contents' section with links to 'Overview', 'Authors', 'Project Description', 'Requirements and Installation', 'Usage', 'Contributions & Help', and 'Contribution Policy'.

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.