# Exercise 2 - Random Forest Regression Algorithm

Clara Pichler
Hannah Knapp
Sibel Toprakkiran

Group 18

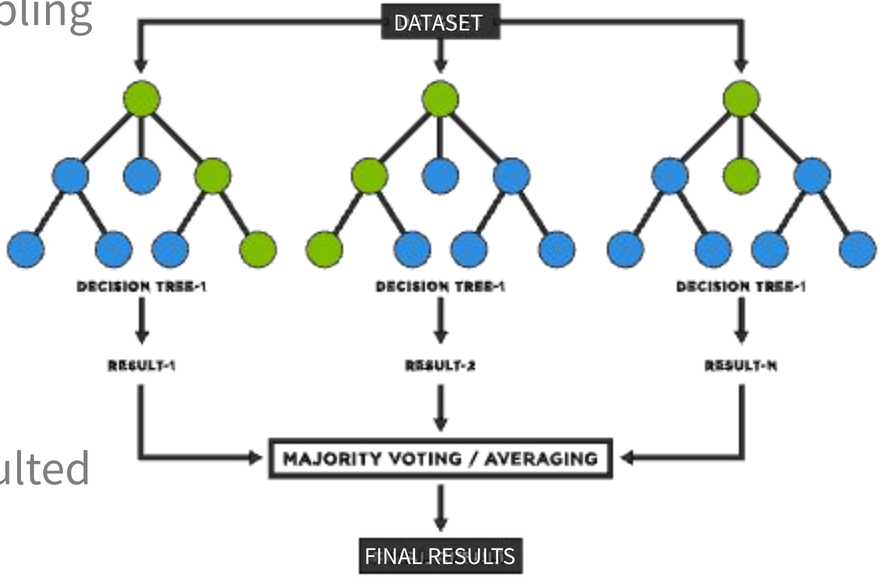
# Random Forest Regressor

Step 1: Creating multiple data frames by resampling the original data

-> Bootstrap

Step 2: Build multiple decision trees regressors

Step 3: Averaging over the predicted values resulted from the decision trees



## Setup

- Python 3.12.5 - Jupiter Notebook files
- Using only numpy and pandas for the implementation of our regressor
- Testing our model with two dataframes:
    - Airfoil Noise Data
    - Abalone Data

- Using Git to collaborate and merge our code
- Multiple meetings on discord to discuss

## Bootstrap

- Method to create multiple subsets of the original data by sampling **with replacement**
- Each subset is used to train one decision tree
- introduces randomness => trees see different views on data
- helps reduce overfitting

```
make_bootstrap(X, y, n_bootstraps = 100)
```

- sample size is the same as the original data
- selecting random data points (indices) from the original data with replacement
- returns a list of tupel containing the bootstraped X and y

## Decision Tree Regressor

Helper Functions:

- Split the data
- Find the best split
- Calculating metrics (for finding the best split)

Tree Building:

- Only using a random subset of the features (`max_features`)
- Recursively splitting data into smaller groups (trees) until a stopping condition is met

## Build Decison Tree

`build_tree(X, y, max_depth, min_samples_split, max_features=”sqrt”, depth=0, metric=”mse”)`

- building a decision tree recursively
- check if stopping conditions are met (leaf node)
  - `max_depth <= depth or len(y) < min_samples_split or mse(y) == 0`
  - return mean of y if conditions are met
- `feature_subset` random *m* features without replacement
  - where *m* is chosen through max_features
- `find_best_split` to get `threshold` and `feature_idx`
- `split_dataset` based on this threshold and feature
- calling `build_tree` for right and left tree with `depth+=1`
- returns dictionary for each tree containing `feature_idx`, `threshold` and left and right tree

# Different Metrics to find Best Split

- MSE
  - Measures average squared difference of actual and predicted values
  - Using weighted version
  - We want to minimize weighted MSE (choose lowest)
- Variance Reduction
  - Measures decrease in variance after splitting
  - effective when capturing variability n target values important
  - Choosing split with highest variance reduction
- MAE
  - Measures average absolute differences of actual and predicted values
  - Using weighted version
  - minimize weighted MAE

dealing better with outliers than MSE

## Find Best Split

`find_best_split(X,y,feature_subset=None, metric=”mse”)`

- depending on the metric:
  - define `best_metric` as `inf` or `-inf`
  - define `is_better` function
- for each feature in `feature_subset` (if `None` then all features are considered)
  - each value is considered as a threshold
  - based on the splitting the data
  - if one split has length of zero => continue
  - else calculating the metric value
  - with `is_better` function check if we found a better split
  - update `best_metric`, `best_feature` and `best_threshold` if that’s the case

# RF Regressor Implementation

Class

- `__init__(self, n_trees=10, max_depth=5, min_sample_split=2, max_features=”sqrt”, metric=”mse”)`
    - initialising the parameters

- `fit(self, X, y)`
    - creates `n_trees` bootstrap samples with `make_bootstraps`
    - builds a decision tree with each sample using `build_tree`
    - saving them as list of dictionaries in `self.trees`

## RF Regressor Implementation - Predicting

`predict_tree(tree, X)`

- Traverses the tree recursively, starting at root node and moving down to a leaf node
- If tree is a leaf node
  - returns mean of the target values of the leaf node
- At each non-leaf node decides which child branch to follow
  - `X[feature_idx] <= threshold`, the function proceeds to the left subtree (`tree["left"]`).

`predict(self, X)`

- Creates array where each row corresponds to predictions from one tree, and each column corresponds to a single data point
- returns array where each value is the averaged prediction for the corresponding data point

## LLM Random Forest Regressor

- Using ChatGPT (4o mini) as our LLM

a very general prompt without giving any details e.g. about hyperparameters

Prompt:
*Can you give me the code for a random forest algorithm from scratch with just numpy and pandas? It should be based on Regression Decision Trees.*

## Comparison with LLM RF Regressor I

logic for building the decision tree and making predictions similar (bootstrapping, building the tree, splitting, finding the best features with threshold)

specific implementation details differ, differences in hyperparameters and modularity

**differences in  hyperparameters and their values**

| | Group 18 | ChatGPT |
|---|---|---|
| Metric (for finding best split) | MSE, MAE, variance reduction | MSE |
| Feature Subset | can handle sqrt, log, none | none |

# Comparison with LLM RF Regressor II

**differences in fitting phase**

- `build_tree` function to create each decision tree as dictionary vs ChatGpt via `DecisionTreeRegressor class`instance.

**differences in prediction phase**

- iterate over the trees and call `predict_tree` vs. ChatGpt implementation calls the `predict` method of each `DecisionTreeRegressor` class instance.

## Comparing with KNN

both can capture non linear relationships

### Random Forest

#### Advantages

- large datasets and high dimensions
- less sensitive to noise and irrelevant features (combining multiple trees)
- can  provide information about feature importance
- can be parallelized

#### Disadvantages

- could be overfitted
- predictions less interpretable
- more complex than KNN, more hyperparameters to be tuned

### KNN

#### Advantages

- easier to understand
- easy to interpret based on nearest neighbours

#### Disadvantages

- computationally expensive for large datasets
- sensitive to noise

## Evaluation

- using two data sets → Airfoil and Abalone
- using different metrics → MSE, MAE and variance reduction
- using different max_features → sqrt(n), log2(n), all features
- running different models with different parameter sets:
  - small n_trees vs. big n_trees
  - small max_depth vs. big n_trees
  - small min_sample split vs. big min_sample split
  - through GridSearchCV, RandomizedSearchCV find best parameters

Default parameters: `n_trees=10, max_depth=5, min_sample_split=2, max_features=”sqrt”, metric=”mse”`

## Experimental results I (Airfoil)

Results with variations in metrics

1. mse:

| | RF_18 | RF_ChatGPT | RF_Sklearn |
|---|---|---|---|
| MSE | 16.74 | 13.74 | 13.22 |
| MAE | 3.34 | 2.88 | 2.78 |
| R^2 | 0.65 | 0.71 | 0.72 |

2. variance reduction:

| | RF_18 | RF_ChatGPT | RF_Sklearn |
|---|---|---|---|
| MSE | 16.07 | - | - |
| MAE | 3.26 | - | - |
| R^2 | 0.66 | - | - |

3. mae:

| | RF_18 | RF_ChatGPT | RF_Sklearn |
|---|---|---|---|
| MSE | 17.94 | - | 13.38 |
| MAE | 3.40 | - | 2.78 |
| R^2 | 0.62 | - | 0.72 |

## Experimental results I (Abalone)

Results with variations in metrics

1. mse:

| | RF_18 | RF_ChatGPT | RF_Sklearn |
|---|---|---|---|
| MSE | 5.62 | 5.03 | 4.91 |
| MAE | 1.66 | 1.56 | 1.56 |
| R^2 | 0.49 | 0.54 | 0.55 |

2. variance reduction:

| | RF_18 | RF_ChatGPT | RF_Sklearn |
|---|---|---|---|
| MSE | 5.51 | - | - |
| MAE | 1.65 | - | - |
| R^2 | 0.50 | - | - |

3. mae:

| | RF_18 | RF_ChatGPT | RF_Sklearn |
|---|---|---|---|
| MSE | 5.70 | - | 5.22 |
| MAE | 1.68 | - | 1.50 |
| R^2 | 0.48 | - | 0.53 |

## Experimental results II (Airfoil)

Results with variations in max_features

1. max_features = √n:

| | RF_18 | RF_ChatGPT | RF_Sklearn |
|---|---|---|---|
| MSE | 16.74 | - | 15.32 |
| MAE | 3.34 | - | 3.12 |
| R^2 | 0.65 | - | 0.68 |

2. max_features = log2(n):

| | RF_18 | RF_ChatGPT | RF_Sklearn |
|---|---|---|---|
| MSE | 15.37 | - | 16.68 |
| MAE | 3.10 | - | 3.29 |
| R^2 | 0.68 | - | 0.65 |

3. max_features = n

| | RF_18 | RF_ChatGPT | RF_Sklearn |
|---|---|---|---|
| MSE | 14.35 | 13.74 | 13.92 |
| MAE | 2.90 | 2.88 | 2.79 |
| R^2 | 0.70 | 0.71 | 0.71 |

# Experimental results II (Abalone)

## Results with variations in max_features

1. max_features = √n:

| | RF_18 | RF_ChatGPT | RF_Sklearn |
|---|---|---|---|
| MSE | 5.62 | - | 5.55 |
| MAE | 1.66 | - | 1.66 |
| R^2 | 0.49 | - | 0.50 |

2. max_features = log2(n):

| | RF_18 | RF_ChatGPT | RF_Sklearn |
|---|---|---|---|
| MSE | 5.84 | - | 5.78 |
| MAE | 1.70 | - | 1.69 |
| R^2 | 0.47 | - | 0.48 |

3. max_features = n

| | RF_18 | RF_ChatGPT | RF_Sklearn |
|---|---|---|---|
| MSE | 4.89 | 5.03 | 5.01 |
| MAE | 1.54 | 1.56 | 1.57 |
| R^2 | 0.56 | 0.54 | 0.55 |

# Experimental results III (Airfoil)

## Results with variations in n_trees

1. n_trees = 10:

| | RF_18 | RF_ChatGPT | RF_Sklearn |
|---|---|---|---|
| MSE | 16.74 | 13.74 | 13.59 |
| MAE | 3.34 | 2.88 | 2.84 |
| R^2 | 0.65 | 0.71 | 0.71 |

2. n_trees = 25:

| | RF_18 | RF_ChatGPT | RF_Sklearn |
|---|---|---|---|
| MSE | 15.25 | 13.13 | 13.15 |
| MAE | 3.15 | 2.78 | 2.80 |
| R^2 | 0.68 | 0.72 | 0.72 |

3. n_trees = 50:

| | RF_18 | RF_ChatGPT | RF_Sklearn |
|---|---|---|---|
| MSE | 15.13 | 13.11 | 13.03 |
| MAE | 3.13 | 2.79 | 2.79 |
| R^2 | 0.68 | 0.73 | 0.73 |

# Experimental results III (Abalone)

## Results with variations in n_trees

1. n_trees = 10:

| | RF_18 | RF_ChatGPT | RF_Sklearn |
|---|---|---|---|
| MSE | 5.62 | 5.03 | 5.12 |
| MAE | 1.66 | 1.56 | 1.57 |
| R^2 | 0.49 | 0.54 | 0.54 |

2. n_trees = 25:

| | RF_18 | RF_ChatGPT | RF_Sklearn |
|---|---|---|---|
| MSE | 5.59 | 4.99 | 4.99 |
| MAE | 1.65 | 1.56 | 1.56 |
| R^2 | 0.49 | 0.55 | 0.55 |

3. n_trees = 50:

| | RF_18 | RF_ChatGPT | RF_Sklearn |
|---|---|---|---|
| MSE | 5.48 | 4.86 | 4.92 |
| MAE | 1.63 | 1.54 | 1.55 |
| R^2 | 0.50 | 0.56 | 0.55 |

## Experimental results IV (Airfoil)

Results with variations in max_depth

1. max_depth = 3:

| | RF_18 | RF_ChatGPT | RF_Sklearn |
|---|---|---|---|
| MSE | 24.32 | 23.28 | 23.22 |
| MAE | 4.02 | 3.85 | 3.79 |
| R^2 | 0.49 | 0.51 | 0.51 |

2. max_depth = 5:

| | RF_18 | RF_ChatGPT | RF_Sklearn |
|---|---|---|---|
| MSE | 16.74 | 13.74 | 12.81 |
| MAE | 3.34 | 2.88 | 2.75 |
| R^2 | 0.65 | 0.71 | 0.73 |

3. max_depth = 7:

| | RF_18 | RF_ChatGPT | RF_Sklearn |
|---|---|---|---|
| MSE | 10.19 | 8.10 | 7.90 |
| MAE | 2.49 | 2.18 | 2.17 |
| R^2 | 0.79 | 0.83 | 0.83 |

# Experimental results IV (Abalone)

## Results with variations in max_depth

1. max_depth = 3:

| | RF_18 | RF_ChatGPT | RF_Sklearn |
|---|---|---|---|
| MSE | 6.58 | 6.11 | 6.18 |
| MAE | 1.83 | 1.75 | 1.75 |
| R^2 | 0.40 | 0.45 | 0.44 |

2. max_depth = 5:

| | RF_18 | RF_ChatGPT | RF_Sklearn |
|---|---|---|---|
| MSE | 5.62 | 5.03 | 5.12 |
| MAE | 1.66 | 1.56 | 1.57 |
| R^2 | 0.49 | 0.54 | 0.54 |

3. max_depth = 7:

| | RF_18 | RF_ChatGPT | RF_Sklearn |
|---|---|---|---|
| MSE | 5.14 | 4.86 | 4.69 |
| MAE | 1.57 | 1.52 | 1.51 |
| R^2 | 0.53 | 0.56 | 0.57 |

# Experimental results V (Airfoil)

## Results with variations in min_sample split

1. min_sample split = 2:

| | RF_18 | RF_ChatGPT | RF_Sklearn |
|---|---|---|---|
| MSE | 16.74 | 13.74 | 12.81 |
| MAE | 3.34 | 2.88 | 2.75 |
| R^2 | 0.65 | 0.71 | 0.73 |

2. min_sample split = 4:

| | RF_18 | RF_ChatGPT | RF_Sklearn |
|---|---|---|---|
| MSE | 16.22 | 12.76 | 13.74 |
| MAE | 3.26 | 2.74 | 2.91 |
| R^2 | 0.66 | 0.73 | 0.71 |

3. min_sample split = 6:

| | RF_18 | RF_ChatGPT | RF_Sklearn |
|---|---|---|---|
| MSE | 17.85 | 12.41 | 13.71 |
| MAE | 3.39 | 2.73 | 2.83 |
| R^2 | 0.63 | 0.74 | 0.71 |

# Experimental results V (Abalone)

Results with variations in min_sample split

1. min_sample split = 2:

| | RF_18 | RF_ChatGPT | RF_Sklearn |
|---|---|---|---|
| MSE | 5.62 | 5.03 | 5.12 |
| MAE | 1.66 | 1.56 | 1.57 |
| R^2 | 0.49 | 0.54 | 0.54 |

2. min_sample split = 4:

| | RF_18 | RF_ChatGPT | RF_Sklearn |
|---|---|---|---|
| MSE | 5.50 | 4.91 | 4.94 |
| MAE | 1.63 | 1.55 | 1.55 |
| R^2 | 0.50 | 0.55 | 0.55 |

3. min_sample split = 6:

| | RF_18 | RF_ChatGPT | RF_Sklearn |
|---|---|---|---|
| MSE | 5.52 | 5.08 | 4.91 |
| MAE | 1.66 | 1.57 | 1.54 |
| R^2 | 0.50 | 0.54 | 0.55 |

# Experimental results VI

1. Results with RandomizedSearchCV (Airfoil)

   Best Parameters: RF: `max_depth=15, max_features=None, n_estimators=150` ; K-NN: `n_neighbors=7`

| | RF_18 | RF_ChatGPT | RF_Sklearn | K-NN |
|---|---|---|---|---|
| MSE | 3.77 | 3.84 | 3.43 | 37.86 |
| MAE | 1.37 | 1.38 | 1.35 | 4.86 |
| R^2 | 0.92 | 0.92 | 0.93 | 0.19 |

2. Results with RandomizedSearchCV (Abalone)

   Best Parameters: RF: `max_depth=15, max_features='sqrt', min_samples_split=3` ; K-NN: `n_neighbors=7, weights='distance'`

| | RF_18 | RF_ChatGPT | RF_Sklearn | K-NN |
|---|---|---|---|---|
| MSE | 4.82 | 4.78 | 4.64 | 4.86 |
| MAE | 1.54 | 1.54 | 1.51 | 1.52 |
| R^2 | 0.56 | 0.57 | 0.58 | 0.54 |

## Conclusions

- Our Random Forest does relatively well in comparison with the RF from ChatGPT and the RF from SKlearn
- There could be improvement in optimization to make the Code faster
- Make the Code modular with adding a Class for the Decision Trees
- Our RF is slower than the RF from Sklearn but faster than the one from ChatGPT
- We get the best results if the number of Trees gets bigger and the max_depth gets bigger
- But to reduce Overfitting the number of max_depth should be lower
- K-NN does not perform as well as RF and for Airfoil it performs really bad
- The Regressors work better for the Airfoil Dataset than the Abalone Dataset