Project 1 – INF264

Felix Anthonisen
Jakob Sverre Alexandersen

September 21 2024

1 Introduction

In this project, we explored the use of decision trees and random forest classifiers for predicting the type or country of origin of coffee and wine, using a dataset containing relevant features. Decision trees, known for their simplicity and interpretability, were implemented alongside the more robust and ensemble-based random forest approach. To improve model performance, we conducted hyperparameter tuning using grid search, aiming to find the optimal configurations for each classifier. The final performance of our custom implementations was then compared against the standard versions provided by scikit-learn, to assess their effectiveness and accuracy

1.1 Division of labour

We both worked in parallel when writing this project. The only times we worked not together was during small optimising steps and fixing typos.

2 Data analysis

The data consists of 2 datasets, one for coffee and one for wine.

2.0.1 Coffee dataset:

The available features for coffee are aroma, flavor, aftertaste, body, acidity, balance, uniformity and sweetness.

These features appear to have a somewhat normal distribution, with most scores concentrated in the middle ranges (between 6 and 8). Slight skewness is visible for some attributes like Acidity and Balance, where most values are towards the higher range.

For uniformity and sweetness the distributions are highly skewed with most of the data points are concentrated at 10.

The target feature for this dataset is country of origin. The targets are relatively balanced with a slight skew towards 1. This skew might impact the performance of our classifiers.

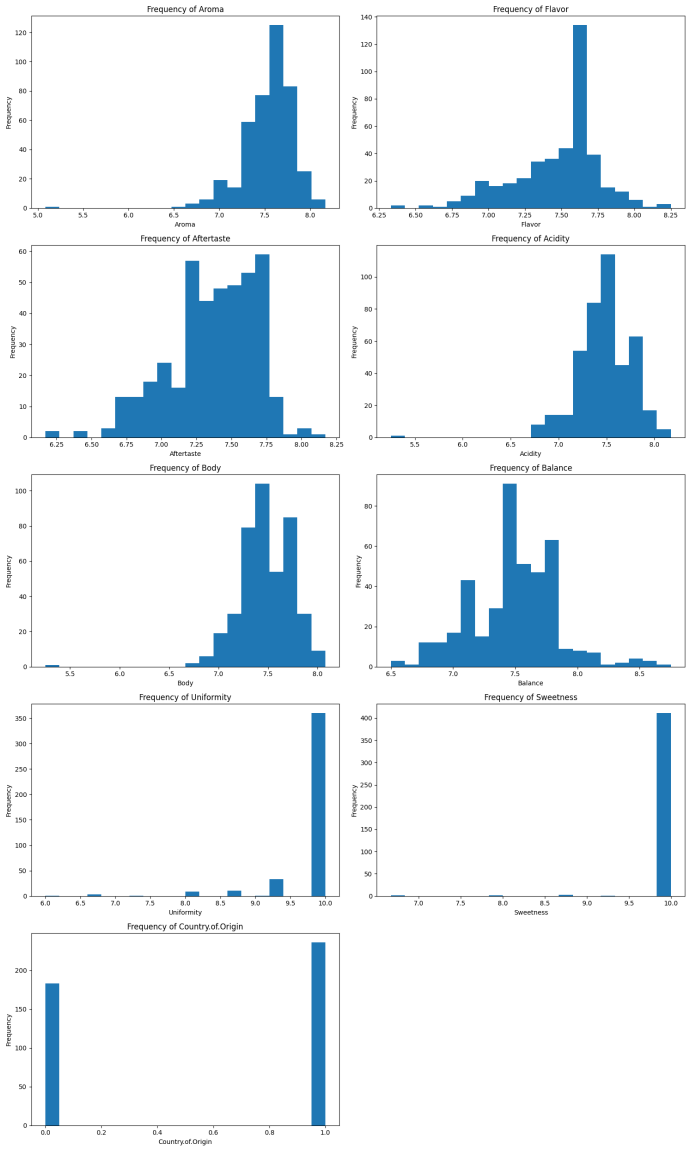

Figure 1: Frequencies for features and target for the coffee dataset

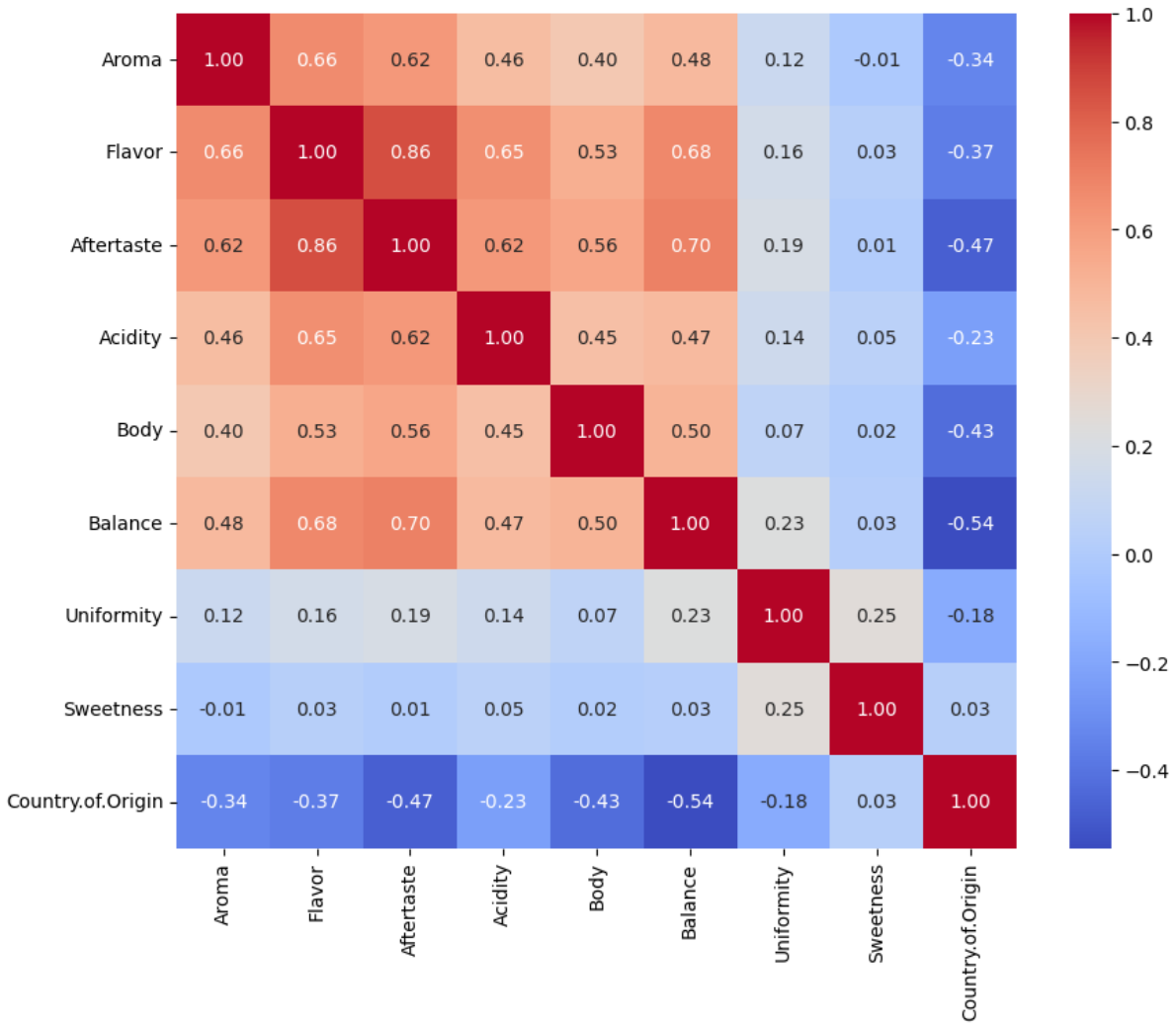

Figure 2: Correlation matrix between variables in the coffee dataset

2.0.2 Wine dataset:

The available features for wine are citric acid, sulphates,residual sugar, pH and alcohol.

Citric Acid, Sulphates, and Residual Sugar show irregular or skewed distributions, meaning certain characteristics might be rare or highly specific to certain wines.

The dataset seems relatively balanced for pH and alcohol.

The target feature for this dataset is type. The targets are balanced, which should be good for classification.

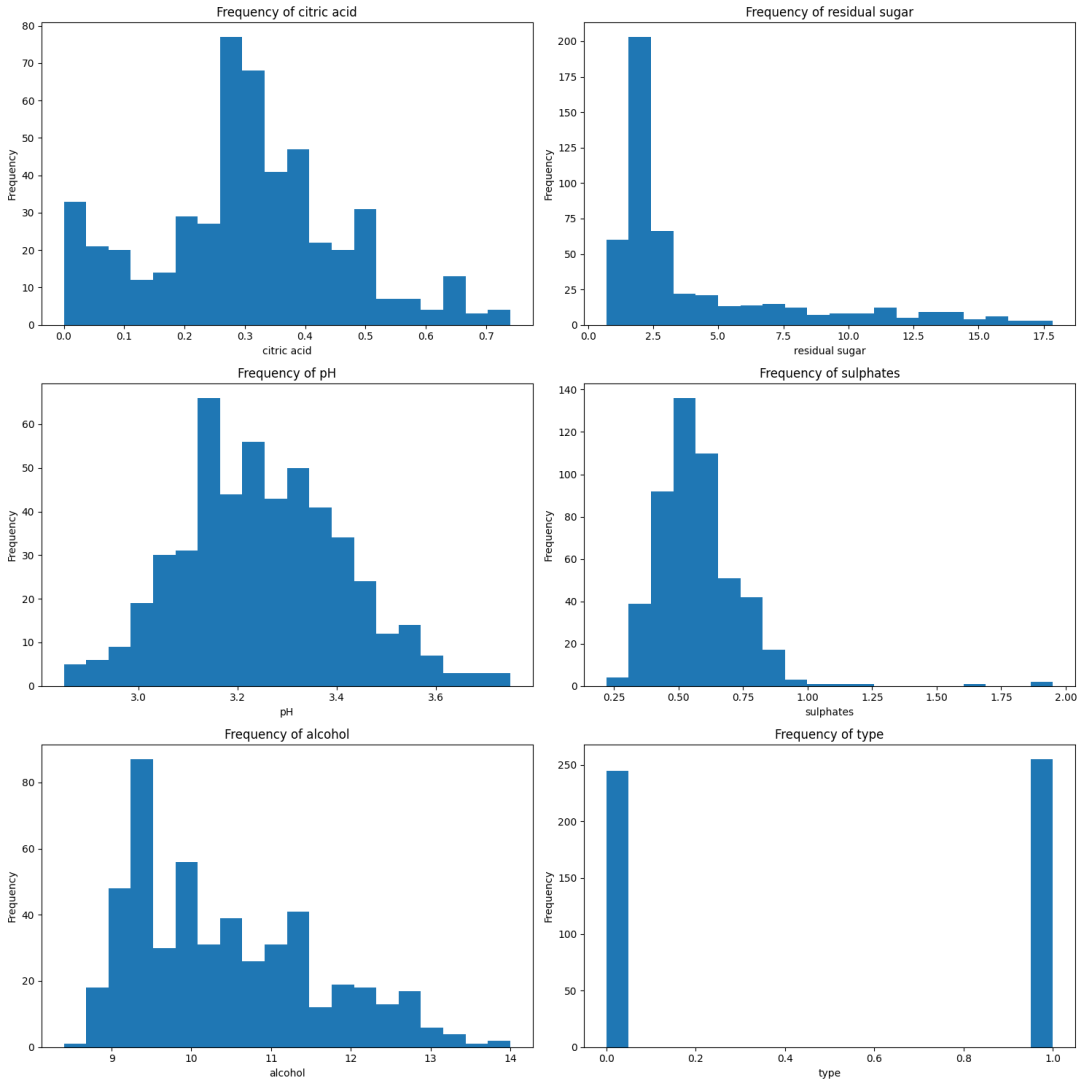

Figure 3: Frequencies for features and target for the wine dataset

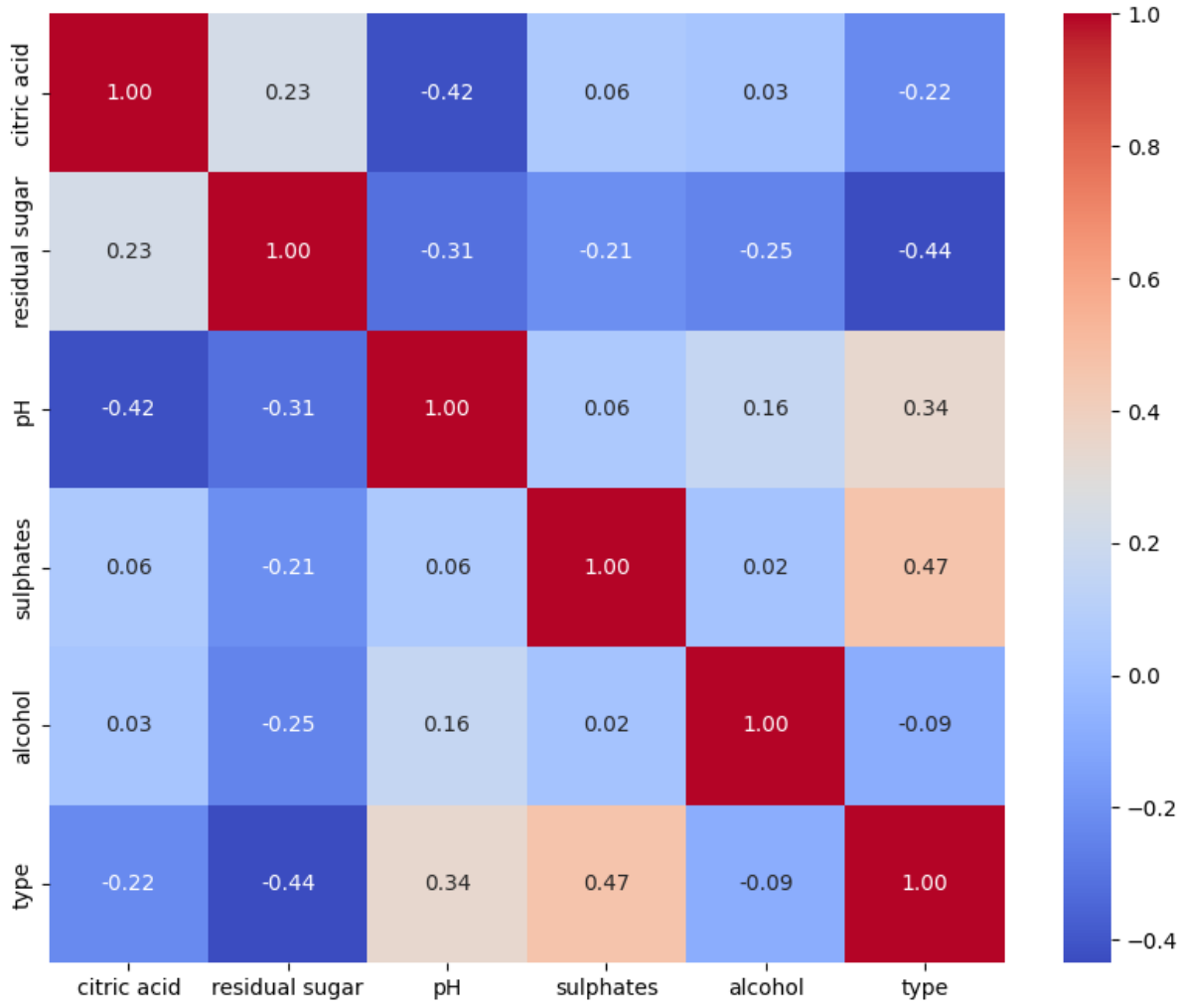

Figure 4: Correlation matrix between variables in the wine dataset

2.1 Data preprocessing

We decided that further data preprocessing was unnecessary. Before applying the decision tree and random forest classifiers, we reviewed the datasets and determined that the data was already in a form that could be directly utilized by these algorithms. The key reasons are outlined below:

Clean Data: Both the coffee and wine datasets appeared to be free of missing values or outliers that could heavily skew the models. Decision trees and random forests are relatively robust to outliers, as they focus on partitioning the feature space based on thresholds, so no preprocessing was required to handle potential noise in the data.

Balanced Target Variables: As shown in the histograms, the target variables (Country of Origin for coffee and Type for wine) were relatively balanced, with both classes well-represented in the datasets. As a result, we did not need to do any processing here.

Handling of Skewed Features: While some features showed slight skewness (e.g., residual sugar in wine, uniformity in coffee), the decision tree-based models used (both decision trees and random forests) are non-parametric. These models do not assume a normal distribution in the data, making them robust to skewness and non-linear relationships.

3 Implementation

3.1 Decision Tree classifier:

We used the decision tree classifier to predict the type / country of origin of coffee and wine, respectively. The user is able to decide between a range of hyperparameters to tune the model.

Hyperparameter	Description	Default value	Possible values
`max_depth`	The maximum depth of the tree	`None`	`None` or `int`
`criterion`	The function to measure the quality of a split	`gini`	`gini`, `entropy`
`max_features`	The maximum amount of features	`None`	`log2`, `sqrt`, and `None`
`random_state`	The seed for all random functions	`0`	`int`

How it works:

The decision tree classifier `fit` method operates by recursively splitting the dataset into subsets based on the feature that yields the most informative split. This process continues until the data is fully partitioned into subsets, effectively creating a tree structure.

To evaluate the quality of each split, the classifier employs metrics such as Gini impurity or entropy to calculate the information gain we would get if this split was performed.

More Key parameters in the decision tree algorithm include:

- **max_depth**: This parameter limits the depth of the tree, ensuring that no node exceeds a specified level. This helps prevent overfitting by controlling the complexity of the model. Formally, for any node n, the condition $n_{depth} \leq$ `max_depth` must hold
- **max_features**: This parameter specifies the maximum number of features to consider when searching for the best split at each node. By limiting the features, the algorithm introduces randomness, which can enhance model generalization.
- **random_state**: This parameter serves as a seed for any random processes within the algorithm, ensuring reproducibility of results across different runs.

Information Gain

We calculate IG for each feature and choose the feature with the highest IG to split the data on. The IG is calculated as follows:

$$IG(x, y) = H(y) - H(y|x) \tag{1}$$

Where:

- $IG(x, y$ is the information gain
- $H(y)$ is the entropy / gini index of the target variable
- $H(y|x)$ is the conditional entropy / gini index of the target variable given the feature

3.2 Random Forest classifier

The Random Forest Classifier (RFC) is built on top of the decision tree classifier. It leverages the concept of bagging (Bootstrap Aggregating) to enhance model performance. In bagging, multiple decision trees are trained on different subsets of the training data. This approach helps to reduce overfitting and increase the diversity among the individual trees, resulting in a more robust model. To make predictions, the RFC aggregates the outputs of all the individual trees, employing a majority voting mechanism. The class label that receives the most votes across the ensemble of trees is chosen as the final prediction. The performance of the RFC can be fine-tuned through several hyperparameters, including:

Hyperparameter	**Description**	**Default value**	**Possible values**
`n_estimators`	The number of trees to train	`100`	`int`
`max_depth`	The maximum depth of the tree	`None`	`None` or `int`
`criterion`	The function to measure the quality of a split	`gini`	`gini`, `entropy`
`max_features`	The maximum amount of features	`sqrt`	`log2`, `sqrt`, and `None`
`random_state`	The seed for all random functions	`0`	`int`

4 Model selection and evaluation

Model Selection Technique

We utilized grid search for hyperparameter tuning, which systematically explores all possible combinations of the specified hyperparameters to find the best configuration. This exhaustive approach allows us to ensure that we thoroughly evaluate the potential hyperparameter combinations.

Scoring Metric

The primary scoring metric we used for evaluation was accuracy. Accuracy is straightforward to compute and interpret, providing a clear measure of how well the model performs. However, we acknowledged its limitation in scenarios with class imbalance, where it may not adequately reflect model performance. To mitigate the risk of overfitting and ensure a robust evaluation, we employed cross-validation using sklearn's GridSearchCV. This technique allows us to assess model performance across different subsets of the training data, providing a more reliable estimate of how the model will perform on unseen data.

Justification of Choices

The choice of hyperparameters was guided by the need to balance model complexity and generalization. Grid search provides a thorough and methodical way to optimize these hyperparameters, while accuracy – although simple – serves as a clear baseline metric. Cross-validation further enhances our evaluation by ensuring that the results are not biased due to a particular train-test split.

4.1 Hyperparameter tuning for the decision tree classifier

Hyperparameter	Potential values
max_depth	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, None]
criterion	["entropy", "gini"]
max_features	["log2", "sqrt", None]

4.2 Hyperparameter tuning for the random forest classifier

Hyperparameter	Potential values
n_estimators	[5, 6, 7, 8, 9, 10, 25, 30, 50, 75, 100]
max_depth	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, None]
criterion	["entropy", "gini"]
max_features	["log2", "sqrt", None]

5 Wine dataset

After tuning the hyperparameters by using Grid Search Cross Validation, we got the following sets of optimal hyperparameters for each classifier:

5.1 Our Decision Tree Classifier

Hyperparameter	Value
max_depth	None
criterion	entropy
max_features	None

5.2 Our Random Forest Classifier

Hyperparameter	Value
n_estimators	100
max_depth	5
criterion	entropy
max_features	sqrt

5.3 SKLearn Decision Tree Classifier

Hyperparameter	Value
max_depth	7
criterion	entropy
max_features	None

5.4 SKLearn Random Forest Classifier

Hyperparameter	Value
n_estimators	75
max_depth	9
criterion	entropy
max_features	log2

5.5 Results

Model	Training	Validation
Our decision tree	1	0.84
Our random forest	0.9971	0.87333
SKL decision tree	0.9485	0.7866
SKL random forest	0.9971	0.87333

6 Coffee dataset

After tuning the hyperparameters by using Grid Search Cross Validation, we got the following sets of optimal hyperparameters for each classifier:

6.1 Our Decision Tree Classifier

Hyperparameter	Value
max_depth	None
criterion	entropy
max_features	None

6.2 Our Random Forest Classifier

Hyperparameter	Value
n_estimators	100
max_depth	5
criterion	entropy
max_features	sqrt

6.3 SKLearn Decision Tree Classifier

Hyperparameter	Value
max_depth	3
criterion	entropy
max_features	None

6.4 SKLearn Random Forest Classifier

Hyperparameter	Value
n_estimators	25
max_depth	4
criterion	entropy
max_features	None

6.5 Results

Model	Training	Validation
Our decision tree	1	0.73
Our random forest	0.86	0.8015
SKL decision tree	0.8464	0.7936
SKL random forest	0.8737	0.8095

7 Discussion

7.1 Comparing Decision Trees and Random Forests

The Random Forest Classifier consistently outperformed the Decision Tree Classifier across both datasets. The RFC achieved a validation score of **0.8733** compared to the Decision Tree's **0.84** in the wine dataset. On the coffee dataset, the results were not as good as on the wine dataset, but the RFC still outperformed the Decision Tree with a validation score of **0.8015** for RFC compared to **0.73** for the Decision Tree. This demonstrates the RFC's superior ability to generalize, owed to its ensemble approach.

7.2 Comparison to Existing Implementations

In terms of performance, our implementations were generally head to head with SKLearn's versions, achieving similar performance stats, with our model being beat marginally in some cases.

7.3 Results on the Coffee Dataset

For the Coffee dataset, the hyperparameters remained fairly consistent with those used in the wine dataset:. Interestingly, while the hyperparameters were the same, the performance metrics varied between datasets, indicating that the models' effectiveness can depend significantly on the underlying data characteristics.

For instance, the observant reader may have noticed the differences between our correlation matrices. The coffee dataset has stronger correlations between feautures, which may introduce a bias when training a classifier model.

In addition to this, the slight skew in the coffee dataset (namely our target variable – country of origin) may have had an added negative effect on the model performance.

8 Conclusion

This project provided valuable insights into the implementation and evaluation of machine learning models. We learned about the importance of model selection and hyperparameter tuning, which are critical for improving predictive performance. In future projects, we would focus on data cleaning to remove statistical outliers, explore a broader range of models, and potentially expand the hyperparameter grid, despite the increased computational costs. Challenges encountered included optimizing hyperparameters while balancing model complexity and efficiency. However, our systematic approach helped mitigate these issues and led to successful implementations.