Decision Tree Implementation

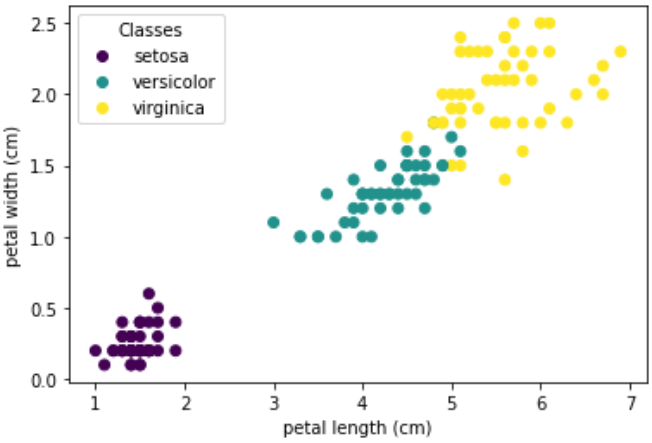
To implement a functional decision tree only one class called DecisionTree is used to build the whole decision tree algorithm and save the training results instead of creating different classes to represent the decision tree nodes, branches and methods needed. The decision tree is trained on the training dataset by first finding the root node using the find_root method. The find_root method iterates through all possible columns of the dataset as potential branch nodes, determines the gini impurity and returns the column with the lowest possible gini impurity. This procedure is in line with the CART decision tree algorithm discussed in the lecture. Once the root node is established, the divide_df method is used to divide the initial dataset into subsets based on the decision boundary established by the root node. The new datasets are each used to create a decision tree child node, another instance of the DecisionTree class. The child nodes are further split up in the same manner as in the initial split until one of two stopping criteria is reached. The two implemented stopping criteria are max_tree_dept and min_leaf_instances. Max_tree_depts stops the learning process once the decision tree reaches a prespecified dept while min_leaf_instances stops the training of a branch node once only a prespecified number of datapoints are left in the data subsets. The static method gini_impurity was implemented to not only work on categorical but also continuous datasets by creating 5 thresholds based on which continuous data columns were converted into categorical data using >= and < operations and returning the column and threshold with the lowest gini impurity. When predictions are being made, the algorithm traverses down the decision tree nodes until a leaf node is reached. Then the target category of the highest number of training instances in that leaf node is returned. The decision tree structure can be visualized using the __str__ method to clearly see the decision tree boundaries and training instances, that reached certain tree branches or leaf nodes.

The space complexity of this decision tree implementation depends on the amount of columns n in the dataset. In a worst-case scenario, where all columns are splitting up the dataset poorly and no stopping criteria are in place, the space complexity will be O(n) because n will be equal to the number of nodes created in the decision tree. The space complexity in this implementation is greatly reduced because the dataset is not saved multiple times during the splits, but only views of the initial dataset are created. The time complexity of the algorithm depends on the columns n and is equal to $O(n^2)$ in the worst-case scenario, where every column is used as a node in the decision tree. The squared complexity is caused by a for loop needed to compare all columns for every split in the decision tree.

Overall the algorithm overperforms a base decision tree algorithm because continuous datasets can be processed, overfitting can be prevented by means of two different stopping criteria and predictions can be made even if the testing data contains partially missing information.

Demonstration of the Decision Tree Algorithm on the Iris Dataset

The effectiveness of the implemented decision tree was demonstrated on the iris dataset. The iris dataset is a popular online dataset with 150 datapoints, that uses four physical attributes of flowers as for example their sepal length and petal width to identify three different iris flower species (Anderson, 1936; Fisher, 1936; Pedregosa et al; 2011). The iris dataset was chosen for several reasons to demonstrate the effectiveness of the decision tree implementation discussed above. First, three iris species are classified in the iris dataset, which makes the prediction of iris species a multiclass instead of a binary classification problem. This adds additional complexity compared to the weather dataset and ensures that the implemented decision tree can make multiclass predictions too. Additionally, all features in the iris dataset are continuous instead of categorical, which is why several adjustments had to be made to the base decision tree algorithm to ensure compatibility with continuous as well as categorical features. Finally, the three iris species in the iris dataset cannot perfectly be linearly separated based on the provided four features (see **Figure 1**), which can cause decision tree algorithms to easily overfit the training dataset as an additional challenge.




```
Instances: 120
+Root: petal length (cm)
         Split Criterion: <2.475
           Instances: 41
           Class 0: 41
         Split Criterion: >=2.475
           Instances: 79
           Class 1: 42
           Class 2: 37
             Root: petal width (cm)
                   Split Criterion: <1.75
                     Instances: 45
                     Class 1: 41
                     Class 2: 4
                       Root: sepal length (cm)
                             Split Criterion: <7.2
                               Instances: 44
                               Class 1: 41
                               Class 2: 3
                             Split Criterion: >=7.2
                               Instances: 1
                               Class 2: 1
                   Split Criterion: >=1.75
                     Instances: 34
                     Class 2: 33
                     Class 1: 1
```

Figure 1: Illustration of the petal width and petal length features of the iris dataset (right) and part of the learned decision tree structure from the iris dataset (left). Iris setosa can easily be separated from iris versicolor and iris virginica, which was clearly achieved by the implemented decision tree algorithm.

The implemented decision tree algorithm performed extremely well on the iris dataset with an overall accuracy of 96% on the testing dataset. Additionally, the recall and precision values for all classes were higher than 89%. The high performance was possible due to the max_dept feature of the decision tree implementation, that limited the decision tree dept and prevented the algorithm from overfitting the training data set. The petal length was determined to be the most important feature to separate the iris species, with iris setosa being significantly easier separatable from the other species than iris versicolor and iris virginica (see **Figure 1**). This suggests the hypothesis that iris versicolor and iris virginica are biologically more closely related to each other than to iris setosa, which could be further investigated. Such a hypothesis would be hard to establish using machine learning methods other than decision trees, as the classification decisions would be concealed from the practitioner.

References

[1] Fisher, R. A., 1936. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2), 179–188.

[2] Anderson, E., 1936. The species problem in Iris. Annals of the Missouri Botanical Garden, 23(3), 457–509.

[3] Pedregosa et al., 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.