Università degli Studi di Milano

Data Science for Economics

Machine Learning Experimental Project:

Tree predictors for binary classification

Student:
Beatrice Cagnin

ACADEMIC YEAR 2023-2024

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

Contents

1 Introduction **4**

2 Dataset **5**

2.1 Data preparation 6

3 Decision Tree classifier for binary classification **7**

3.1 Decision Tree classifier from scratch 7

3.2 Hyperparameter tuning 9

3.3 Random Forest classifier 9

4 Conclusion **10**

5 References **11**

1 Introduction

The aim of this project is the implementation from scratch of tree predictors for binary classification, to determine whether mushrooms are poisonous.

The project starts with the definition of classes for the internal nodes and for the tree predictors themselves.

After having analyse the dataset, transforming all the categorical variables into numerical ones using One-hot encoding, it has been divided into training and test set to apply the algorithms.

The tree predictors have been trained according to the classes, adopting two stopping criteria (maximum tree depth and minimum number of splits), whose optimal values have been found given a specific range, and three criteria for the expansion of the leaves (Gini, Entropy and Squared Error).

Since evaluating the model on the whole dataset does not conduct to a good approximation, a way to improve is applying hyperparameter tuning, using RandomizedSearchCV, according to the splitting criteria and the stopping criteria previously adopted.

The next step is the implementation of Random Forest classifier by reusing the already implemented tree predictor structure, so that the building of several independent trees leads to a more robust model.

The models have been evaluated through cross-validation scores and validation curves.

2 Dataset

The dataset is accessed from the UCI Machine Learning Repository with the ID '848' and it includes 61069 hypothetical mushrooms with caps based on 173 species (353 mushrooms per species). Each mushroom is identified as edible, poisonous, or of unknown edibility and not recommended (the latter class was combined with the poisonous class).

The variables describing mushrooms are:

- cap-diameter (quantitative): float number in cm
- cap-shape (qualitative): bell=b, conical=c, convex=x, flat=f, sunken=s, spherical=p, others=o
- cap-surface (qualitative): fibrous=i, grooves=g, scaly=y, smooth=s, shiny=h, leathery=l, silky=k, sticky=t, wrinkled=w, fleshy=e
- cap-color (qualitative): brown=n, buff=b, gray=g, green=r, pink=p, purple=u, red=e, white=w, yellow=y, blue=l, orange=o, black=k
- does-bruise-bleed (qualitative): bruises-or-bleeding=t,no=f
- gill-attachment (qualitative): adnate=a, adnexed=x, decurrent=d, free=e, sinuate=s, pores=p, none=f, unknown=?
- gill-spacing (qualitative): close=c, distant=d, none=f
- gill-color (qualitative): brown=n, buff=b, gray=g, green=r, pink=p, purple=u, red=e, white=w, yellow=y, blue=l, orange=o, black=k, none=f
- stem-height (quantitative): float number in cm
- stem-width (quantitative): float number in mm
- stem-root (qualitative): bulbous=b, swollen=s, club=c, cup=u, equal=e, rhizomorphs=z, rooted=r
- stem-surface (qualitative): fibrous=i, grooves=g, scaly=y, smooth=s, shiny=h, leathery=l, silky=k, sticky=t, wrinkled=w, fleshy=e, none=f
- stem-color (qualitative): brown=n, buff=b, gray=g, green=r, pink=p, purple=u, red=e, white=w, yellow=y, blue=l, orange=o, black=k, none=f
- veil-type (qualitative): partial=p, universal=u
- veil-color (qualitative): brown=n, buff=b, gray=g, green=r, pink=p, purple=u, red=e, white=w, yellow=y, blue=l, orange=o, black=k, none=f
- has-ring (qualitative): ring=t, none=f
- ring-type (qualitative): obwebby=c, evanescent=e, flaring=r, grooved=g, large=l, pendant=p, sheathing=s, zone=z, scaly=y, movable=m, none=f, unknown=?

- spore-print-color (qualitative): brown=n, buff=b, gray=g, green=r, pink=p, purple=u, red=e, white=w, yellow=y, blue=l, orange=o, black=k
- habitat (qualitative): grasses=g, leaves=l, meadows=m, paths=p, heaths=h, urban=u, waste=w, woods=d
- season (qualitative): spring=s, summer=u, autumn=a, winter=w
- class (qualitative) - TARGET: edible=e, poisonous=p

2.1 Data preparation

In order to prepare the dataset for the application of the algorithms, some techniques have been implemented.

The target variable 'class' has been transformed into numerical type, by indicating '1' for poisonous mushrooms and '0' for edible.

It has made a check for missing values: first of all columns containing a lot of them have been dropped, then also rows containing remaining null values.

In this way it has been obtained a dataset without NA values, with 22239 mushrooms, 9884 edible and 12355 poisonous, described by 15 features (the previous ones without 'veil-type', 'spore-print-color', 'veil-color', 'stem-surface' and 'stem-root').

The algorithms that have been used for this project do not work with categorical variables, so they have been transformed into numerical ones through One-hot encoding. This method applies to values that are not ranked, it creates a new dummy feature for each value in the range of the categorical feature, where the number of dummy variables is equal to the cardinality of the range. Indeed, the number of features increased from 15 to 84.

The final step to proceed is the splitting of the dataset into training set (60%) and test set (40%) .

3 Decision Tree classifier for binary classification

A decision tree is a predictor, $h : X \to Y$, that predicts the label associated with an instance x by traveling from a root node of a tree to a leaf; in this case the focus is on the binary classification setting, namely Y = {0, 1} for edible and poisonous mushrooms.

A decision tree algorithm is built starting from a single-node tree, the root, and assigning it a label according to a majority vote among all labels over the training set. Then the tree is grown performing a series of iterations, by picking a leaf and replacing it with an internal node (labeled by "tests") and two new leaves (labeled by elements of Y). At each iteration the effect of splitting a single leaf is examined, defining a 'gain' measure to quantify the improvement due to the split, so that, among all possible splits, there's the possibility to choose either the one that maximizes the gain and perform it or choose not to split the leaf. The algorithm for the case of binary features is $X = \{0, 1\}^d$, it works by recursive calls and returns a decision tree.

The simplest definition of the 'gain' is the decrease in the training error, that can be written as a sum of contributions due to all leaves

$l_s(h_T) = \sum_l \psi\left(\frac{N_l^+}{N_l}\right) N_l \quad \text{with} \quad \psi = \min\{a, 1-a\}$ defined on [0,1]

with $N_l = |S_l| = N_l^- + N_l^+$, where S_l is the subset of training examples that are routed to l, $N_l^+ = |S_l^+|$ associated to $y_t = 1$ and $N_l^- = |S_l^-|$ associated to $y_t = 0$. A split never increases the training error under the condition that ψ is a concave function.

The splitting criterion used in the project for selecting both the leaf to expand and the decision criterion to adopt in the new internal node are:

- Gini function: $\psi_2(p) = 2p(1-p)$
- Entropy / Information gain: $\psi_3(p) = -p\log_2(p) - (1-p)\log_2(1-p)$
- Squared error (usually used for regression tasks)

The stopping criterion chosen to arrest the construction of the decision tree are:

- max_depth: maximum depth of the tree
- min_samples_split: minimum number of samples splits reaching the leaf

3.1 Decision Tree classifier from scratch

The structure for the nodes, TreeNode, has the following attributes: the 'decision_function', used by the current node returning a Boolean value as output, 'left' and 'right' children, that can be nodes themselves, a 'is_leaf' to check if the node is a leaf, and the 'prediction', that represents the class label (1 - poisonous or 0 - edible).

The attributes of the structure for the binary tree predictor, DecisionTree, are: 'splitting_criterion' for selecting both the leaf to expand and the decision criterion to adopt in the new internal node, it can be 'gini', 'entropy' or 'squared_error', and 'max_depth', 'min_samples_split' as stopping criteria to arrest the construction of the decision tree.

Before applying the algorithm DecisionTree on the training set, cross validation has been computed on stopping criteria in order to find the best values of 'max_depth' and 'min_samples_split' to be put calling the model.

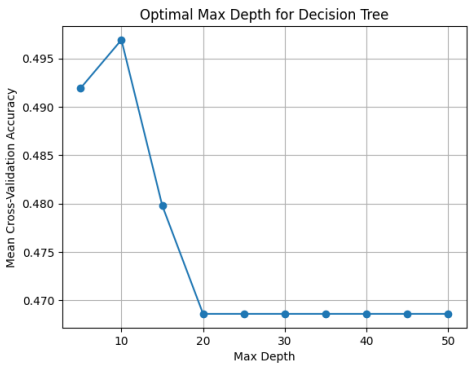


Figure 1: Optimal max_depth - best value = 10

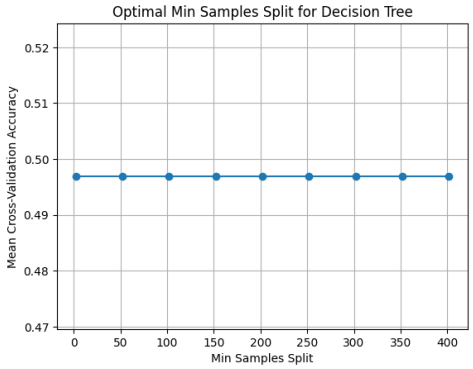


Figure 2: Optimal min_samples_split - best value = 2

As stated by the plots, the best values are max_depth = 10 and min_samples_split = 2, even if the accuracy associated to each is rather low. These values have been used to train three different trees on the training set, one for each splitting criteria while taking the stopping criteria fixed.

GINI

```
tree_gini = DecisionTree(splitting_criterion='gini', max_depth=10, min_samples_split=2)
tree_gini.fit(X_train, y_train)

y_train_pred_gini = tree_gini.predict(X_train)
```

ENTROPY

```
tree_entropy = DecisionTree(splitting_criterion='entropy', max_depth=10, min_samples_split=2)
tree_entropy.fit(X_train, y_train)

y_train_pred_entropy = tree_entropy.predict(X_train)
```

SQUARED ERROR

```
tree_squared_error = DecisionTree(splitting_criterion='squared_error', max_depth=10, min_samples_split=2)
tree_squared_error.fit(X_train, y_train)

y_train_pred_squared_error = tree_squared_error.predict(X_train)
```

In order to evaluate the performance of the models, training errors according to the zero-one loss and test errors have been computed. The training errors are 0.0137 applying Gini, 0.1207 for Entropy and 0.0137 for Squared Error; the test errors are 0.0160 according to Gini, 0.1230 for Entropy and 0.0160 for Squared Error. In general these values are particularly low, giving evidence of the absense of overfitting.

However the previous accuracy, that is the result of (1 - error), is low, indicating that there can be something wrong. In order to try to improve it, hyperparameter tuning has been applied.

3.2 Hyperparameter tuning

Hyperparameters are special parameters whose values must be determined before the training phase. In this case a dictionary for 'splitting_criterium - Gini, Entropy, Squared_Error', 'max_depth: range[5, 51, 5]' and 'min_samples_split: range[2, 451, 50]' has been provided, defining the hyperparameter space for tuning a decision tree model using RandomizedSearchCV.

RandomizedSearchCV is a function of the scikit-learn package used to perform hyperparameter tuning. It searches for the best hyperparameters by randomly sampling a specified number of parameter combinations from the provided search space. Unlike GridSearchCV, which tries all possible combinations, RandomizedSearchCV selects a random subset of hyperparameters to evaluate, which makes it faster and more efficient, especially when there are many hyperparameters and potential values to consider.

Again the DecisionTree classifier has been adopted inside the RandomizedSearchCV function, the new model has been trained indicating as optimal hyperparameters 'min_samples_split = 2', 'max_depth = 40' and 'criterion = Gini'.

Also in this case the best cross-validation accuracy results very high, 0.9993, as well as the accuracy of the test part, 0.9996. As before, seems that there's no overfitting, but the mean of cross-validation scores, that evaluates the performance of the model on the whole dataset, results rather low, 0.4686. Since these parameters are already the result of RandomizedSearchCV, even though they might represent the best found in this progect, overfitting is still a potential issue due to the nature of the dataset, model complexity, and data split.

3.3 Random Forest classifier

Random Forest algorithms can be another way to try to combat overfitting by averaging multiple decision trees, each trained on a random subset of the data and features. They build several trees, 50 in this project, each trained on a different bootstrapped sample of the data. They introduce additional randomness by selecting a random subset of features at each split. This decorrelates the trees, leading to a more robust model.

Again the accuracy of the model on the training set is high, 0.9996, while the mean of cross-validation scores, evaluating the performance on the whole dataset, results 0.5009, slightly better than the case with hyperparameter tuning (0.4686) but still low.

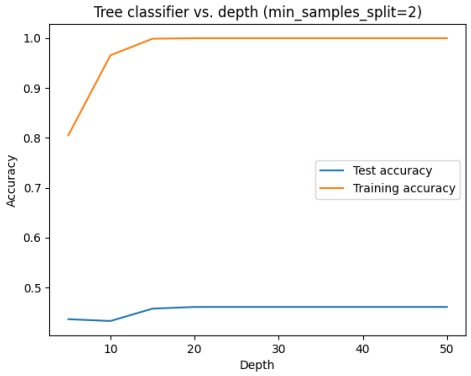
4 Conclusion

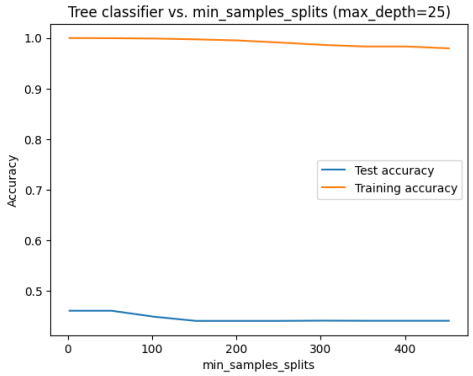
To evaluate the results, validation curves have been compared.

Validation curves are a useful tool for assessing how well a model generalizes to unseen data.

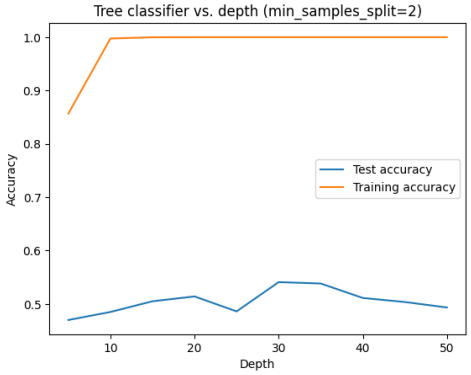
In this case, the curves shows the performance of the model on training and test set of each stopping criterion, keeping fixed the other at the best value suggested by the hyperparameter tuning.

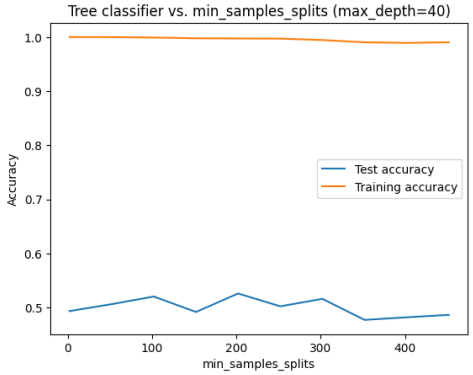
Hyperparameter tuning with RandomSearchCV





Random Forest





All the plots suggest overfitting, indeed the models perform very well on training data but poorly on test data.

When a model shows very high accuracy on the training data but the validation curves indicate overfitting, it means that the model is performing well on the data it was trained on, but is failing to generalize to unseen or validation data. The goal should be to strike a balance between underfitting and overfitting by tuning the hyperparameters, simplifying the model, or employing regularization techniques to improve generalization.

5 References

Shalev-Shwartz and Ben-David, “Understanding Machine Learning: From Theory to Algorithms”, Chapter 18