UNIVERSITÀ DEGLI STUDI DI MILANO
FACOLTÀ DI SCIENZE E TECNOLOGIE

Corso di Laurea Magistrale in Informatica (LM-18)

Statistical Methods for Machine Learning:

Project 2:

Tree predictors for binary classification

Student:
Andrea Moretti, matricola: 47263A

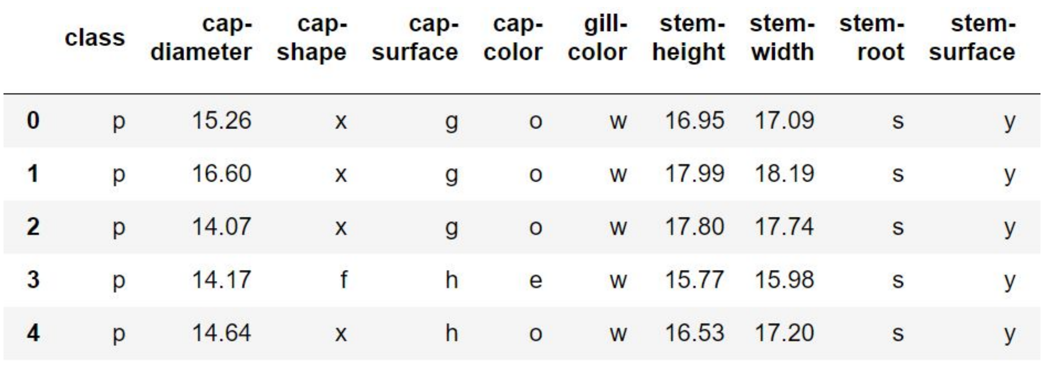
A.Y. 2023/2024

1 Description

The main task of this project is the implementation from scratch of tree predictors for binary classification, in particular to determine whether mushrooms are poisonous from a dataset of mushrooms.

The tree predictors use single-feature binary tests as the decision criterion at any internal node. More precisely, consider thresholds on a single feature in the case of a numerical feature or membership tests in the case of a categorical feature.

The dataset contains 61 069 hypothetical mushrooms with caps based on 173 species (353 mushrooms per species). Each mushroom is identified as definitely edible (label "**e**" or 1) or poisonous (label "**p**" or 0). Of the other 20 features, 17 are nominal and 3 are metrical (Figure 1).



	class	cap-diameter	cap-shape	cap-surface	cap-color	gill-color	stem-height	stem-width	stem-root	stem-surface
0	p	15.26	x	g	o	w	16.95	17.09	s	y
1	p	16.60	x	g	o	w	17.99	18.19	s	y
2	p	14.07	x	g	o	w	17.80	17.74	s	y
3	p	14.17	f	h	e	w	15.77	15.98	s	y
4	p	14.64	x	h	o	w	16.53	17.20	s	y

Figure 1: The dataset of mushrooms

The main classes adopted for this problem are **Node** and **TreeClassifier**. The class **Node** contains a rapresentation of a node componing a tree classifier, in particular it can be a leaf or a decision node. In case of decision node the class stores information about the two child nodes and the decision criterion with the relative threshold value on the feature; in case of leaf the class stores only the predicted label associed.

The class **TreeClassifier** it's the main class and represents the behavior of the binary tree classifier. The constructor initializes the tree with the informations about the splitting criterion to adopte (like gini, entropy etc.) and the stopping criterion, like the maximum tree depth (max_depth) and the minimum samples necessary for a split (min_sample_split).

I've included the possibility to manipulate the behavior for the selection of the threshold in case of numerical features, in fact the constructor also receives a function that reduces the number of value to check for the best split; in my experiments I've chosen a function in order to consider only the 4 evenly spaced values from all the sorted values of a numerical feature (figure 2).

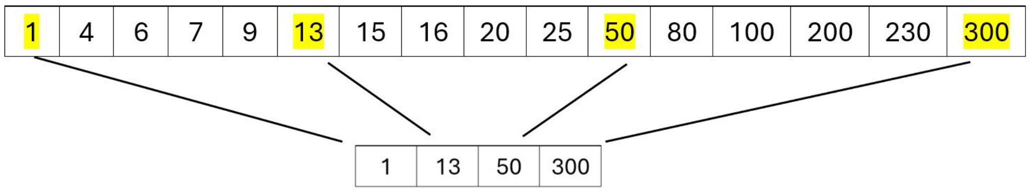


Figure 2: Example of reduction on feature values adopted

This function allows to reduce the number of iterations on the set of values of a feature during the choice of the best splitting threshold.

In the **TreeClassifier** class are also included the methods for the training and the evaluation of the model. The training method ("fit") receives the dataset and the expected labels and creates the structure of the tree, following this idea:

- Start creating the root node;
- For each iteration check the stopping criterion: if true create the leaf and associate the most frequent label then return, otherwise continue;
- Find the value of a feature for the split with the lowest impurity according to the splitting criterion and save it for the decision method;

- Create the two child nodes recalling the method with the new indexes of dataset after the split.

The evaluation method ("evaluate") it's similar to the fit function, it receives the test set and the expected labels, then, using the structure of the tree previously created, it computes the predictions and with the expected outputs evaluates the model using metrics like zero-one loss, precision and recall. It's possible to get the prediction of an example or an array of examples using the method "predict".

Another function I've added to enrich the class **TreeClassifier** it's the "print_tree" method that allows you to print on the command line the structure of the tree with the representation of all nodes and the associated decision criterion.

In order to test the model properly I've also created a function called **CrossValidation** to implement the cross validation method. This function receives the tree classifier, the dataset, the expected labels and a paramter k and produce an estimate of the model performance.

To achieve this behavior the function start sampling the dataset randomly and then split it into the k-fold: for every value of k the model is trained on the k-1 folds and the k$^{\text{th}}$ fold is used as test set. A the end the method evaluates the model and return an average of the metrics.

2 Results

In this section, I present the outcomes derived from the tree classifier model, highlighting key findings that illustrate its effectiveness and applicability. For the evaluation of the model I've used 3 different metrics: the zero-one loss, the precision and the recall. The training of the model is performed on the mushrooms dataset, in particular I divided the entire set with a split of 85/15 for the train/test section.

The first part of the experiments aims to evaluate the performance of the tree using different splitting criteria, in particular:

- Gini function: $2p(1-p)$
- Scaled Entropy: $-\frac{p}{2}\log(p) - \frac{1-p}{2}\log(1-p)$
- Standard deviation: $\sqrt{p(1-p)}$

where p is the mean of the vector of binary labels (composed only by 0, when it's classified as poisonous, and 1 for the edible one) for a cetrain input and corrisponds to the probability to have the label 1.

For every splitting criteria I've computed the cross validation using a tree classifier with the hyperparameter max_depth=5 and min_sample_split=600, obtaining the following results:

Splitting function	**Type**	**zero-one loss**	**Precision**	**Recall**
Gini	Train	0.2885	0.6135	0.9460
	Test	0.2832	0.6205	0.9493
Scaled Entropy	Train	0.2800	0.9096	0.4134
	Test	0.2739	0.9090	0.4180
Standard deviation	Train	0.3148	0.6010	0.8758
	Test	0.3172	0.5935	0.8801

The choice of the hyperparameters is based on a previous test where I noticed that a high value of min_sample_split doesn't change the performance when the value of max_depth is sufficient low (discussed on the next page). In this case I've decided to use a low value of max_depth only to compare the performance between the splitting critria prioritizing a fast creation of the structure of the trees, because a high value of max_depth affects significantly the running time of the algorithm.

The choice of a low value for max_depth shows, as expected, discrete performance, suggesting that the model suffers from underfit; for this reason I decided to do some more tests, in order to understand the limits of tree classifier on this datasets.

This part of experiments consists of selecting a splitting criterion, in my case I've selected the Gini function which performed slightly better in the previous test, and analyze what is the impact of hyperparameters on the performance.

In order to prove what was said before, I've started testing the impact of the min_sample_split setting a value of 10 (instead of 600) and keeping the max_depth to 5:

Hyperparameters	**Type**	**zero-one loss**	**Precision**	**Recall**
max_depth = 5	Train	0.2837	0.6173	0.9480
min_sample_split = 10	Test	0.2840	0.6224	0.9430
max_depth = 5	Train	0.2885	0.6135	0.9460
min_sample_split = 600	Test	0.2832	0.6205	0.9493

The difference between the 2 evaluation of the performance is very slightly. To have a further proof I've trained again the model but this time with min_sample_split setted to 10:

Hyperparameters	**Type**	**zero-one loss**	**Precision**	**Recall**
max_depth = 10	Train	0.1953	0.6966	0.9946
min_sample_split = 10	Test	0.1902	0.7007	0.9964
max_depth = 10	Train	0.2025	0.6921	0.9814
min_sample_split = 600	Test	0.1984	0.6960	0.9837

At this point it's clear that the parameter min_sample_split does not significantly affect the model performance.

However, previous tests have shown that the max_depth can highly change the performance. To verify this I've trained the model with different value of max_depth (keeping the same value of min_sample_split=600) to see how the performance change. The results are:

Hyperparameters	**Type**	**zero-one loss**	**Precision**	**Recall**
max_depth = 7	Train	0.2236	0.6700	0.9772
	Test	0.2245	0.6738	0.9776
max_depth = 13	Train	0.1467	0.8490	0.8149
	Test	0.1492	0.8492	0.8116
max_depth = None	Train	0.0265	0.9709	0.9694
	Test	0.0276	0.9683	0.9708

At this point it's clear the impact of choosing a correct value for max_depth. In particular for the training with max_depth=None the model obtains a very good performance, but it's important to notice that in this training step the computation becomes heavy, in fact the tree has reached a depth of 27, containing a number of nodes between a minimum of $2 \cdot 27 + 1 = 55$ up to a maximum of $2^{28} - 1 = 268\,435\,455$.

For every application of the model on this dataset the tree never shown to suffer from overfit, this is probability due to the fact that the dataset contains explanatory examples and also the random sampling allows to divide correctly the train and the test set.

I've also tested how the tuning of hyperparameters has impact on the performance for either the others splitting critria.

The scaled entropy has the same behavior of the gini function, in fact the model's results are almost the same:

Hyperparameters	**Type**	**zero-one loss**	**Precision**	**Recall**
max_depth = 7	Train	0.2583	0.6384	0.9634
	Test	0.2560	0.6458	0.9638
max_depth = 10	Train	0.2007	0.6967	0.9695
	Test	0.1960	0.7058	0.9730
max_depth = 13	Train	0.1561	0.8344	0.8085
	Test	0.1545	0.8464	0.8052
max_depth = None	Train	0.0234	0.9797	0.9675
	Test	0.0257	0.9784	0.9633

The splitting criterion that behaves differently is the standard deviation, which shows a slightly distinct pattern compared to the Gini and scaled entropy functions.

Hyperparameters	**Type**	**zero-one loss**	**Precision**	**Recall**
max_depth = 7	Train	0.2764	0.6334	0.9007
	Test	0.2826	0.6269	0.8980
max_depth = None	Train	0.0493	0.9520	0.9362
	Test	0.0491	0.9528	0.9382

Overall, it appears to perform slightly worse than the other criteria, not only in the final performance but also for the execution time, this is probably caused by the fact that the execution without the limit on the max_depth reaches a depth of 30.

An important reason because an high value for the hyperparameter max_depth can only increase the performance of the tree classifier is related to the property of splitting criterion. In fact if the formula used to compute the impurity of the split is a concave function, it's possible to demonstrate, via Jensen's inequality, that after a split the training error never increases. All the splitting criteria used are concave (figure 3).

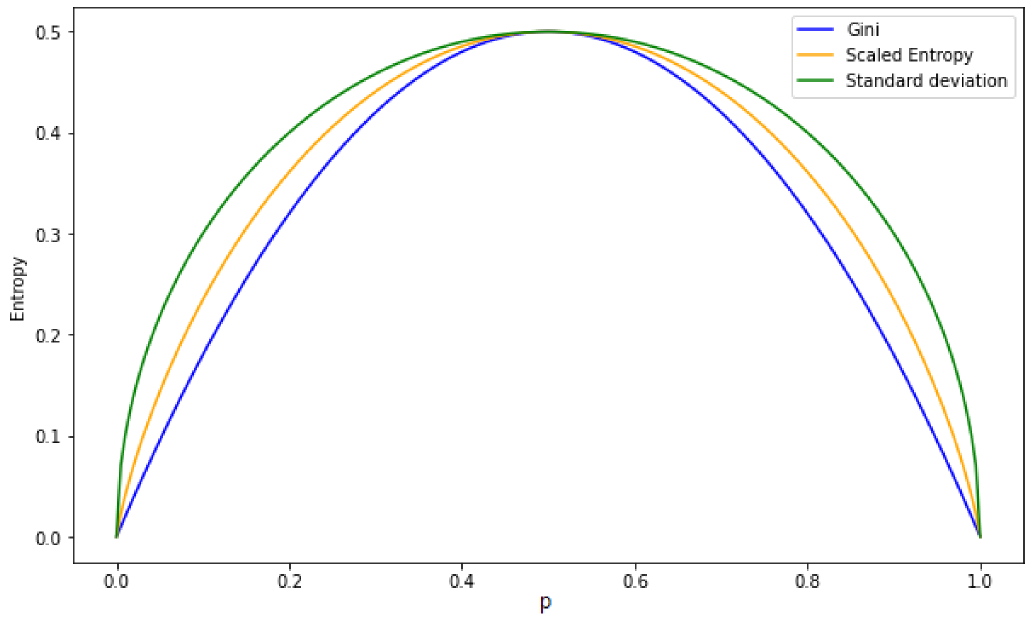


Figure 3: This graph illustrates the entropy functions plotted

A difference between the standard deviation and the others criteria, strictly related to the dataset, is the most important feature separating the majority of the data chosen during the train: according to the standard deviation the more influent feature is the $ring - type$ and if it has a value different of z surely the prediction is 0, so the mushroom is poisonous; the gini function and the scaled entropy, instead, found that the feature with the highest impurity is the $stem - width$ when it has a value < 6.56.