Machine Learning 03
Tuning a hyperparameter of a decision tree

Claudio Sartori

1 Classification with hyperparameter tuning

1.0.1 aim:

Show classification with different strategies for the tuning and evaluation of the classifier 1. simple **holdout** 2. **holdout with validation** train and validate repeatedly changing a hyperparameter, to find the value giving the best score, then test for the final score 4. **cross validation** on training set, then score on test set 5. **bagging** it is an *ensemble* method made available in `scikit-learn`

1.0.2 Workflow

- download the data
- drop the useless data
- separe the predicting attributes X from the class attribute y
- split X and y into training and test
- part 1 - single run with default parameters
  - initialise an estimator with the chosen model generator
  - fit the estimator with the training part of X
  - show the tree structure
  - part 1.1
    - predict the y values with the fitted estimator and the train data
      - compare the predicted values with the true ones and compute the accuracy on the training set
  - part 1.2
    - predict the y values with the fitted estimator and the test data
      - compare the predicted values with the true ones and compute the accuracy on the test set
- part 2 - multiple runs changing a parameter
  - prepare the structure to hold the accuracy data for the multiple runs
  - repeat for all the values of the parameter
    - initialise an estimator with the current parameter value
    - fit the estimator with the training part
    - predict the class for the test part

        * compute the accuracy and store the value
    - find the parameter value for the top accuracy

- part 3 - compute accuracy with cross validation
    - prepare the structure to hold the accuracy data for the multiple runs
    - repeat for all the values of the parameter
        * initialise an estimator with the current parameter value
        * compute the accuracy with cross validation and store the value
    - find the parameter value for the top accuracy
    - fit the estimator with the entire X
    - show the resulting tree and classification report

The data are already in your folder, use the relative path `./uci_breast_tissue_data/BreastTissue.csv`

```
Shape of the input data (1599, 12)
```

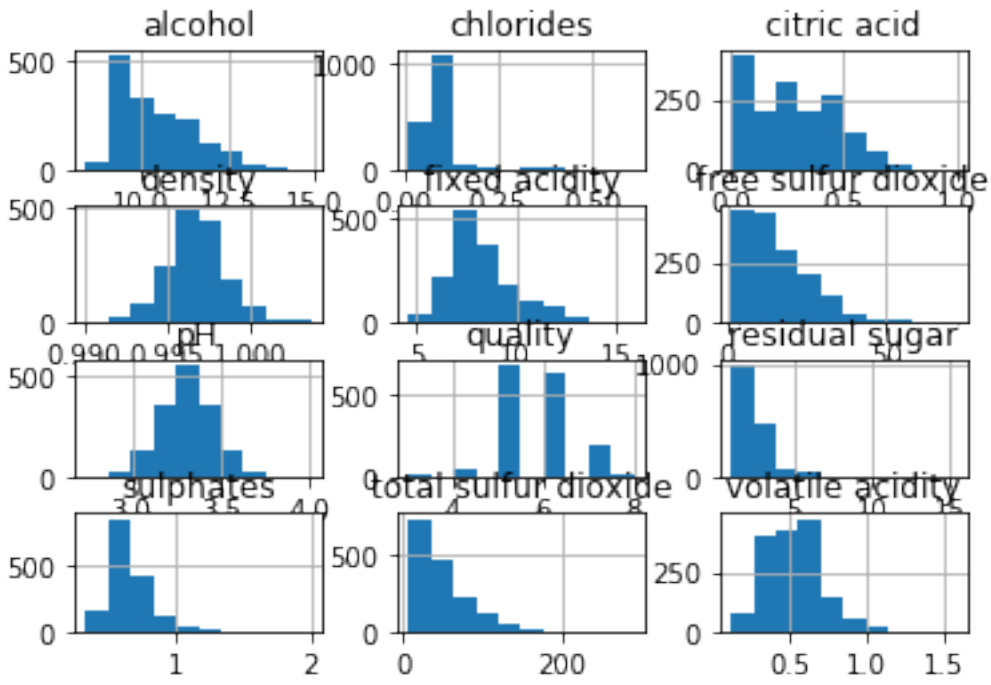
Have a quick look to the data. - use the .shape attribute to see the size - use the `.head()` function to see column names and some data - use the `.hist()` method for an histogram of the columns - use the .unique method to see the class values

[2]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

Use the `hist` method of the DataFrame to show the histograms of the attributes

NB: a semicolon at the end of a statement suppresses the `Out[]`



Print the unique class labels (hint: use the `unique` method of pandas Series

```
[3 4 5 6 7 8]
```

Split the data into the predicting values X and the class y Drop also the columns which are not relevant for training a classifier, if any

The method "drop" of dataframes allows to drop either rows or columns - the "axis" parameter chooses between dropping rows (axis=0) or columns (axis=1)

Another quick look to data

[6]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	\
0	7.4	0.70	0.00	1.9	0.076	
1	7.8	0.88	0.00	2.6	0.098	
2	7.8	0.76	0.04	2.3	0.092	
3	11.2	0.28	0.56	1.9	0.075	
4	7.4	0.70	0.00	1.9	0.076	

	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	\
0	11.0	34.0	0.9978	3.51	0.56	
1	25.0	67.0	0.9968	3.20	0.68	
2	15.0	54.0	0.9970	3.26	0.65	
3	17.0	60.0	0.9980	3.16	0.58	
4	11.0	34.0	0.9978	3.51	0.56	

```
   alcohol
0      9.4
1      9.8
2      9.8
3      9.8
4      9.4
```

[7]:
```
0    5
1    5
2    5
3    6
4    5
Name: quality, dtype: int64
```

1.1 Prepare a simple model selection: holdout method

- Split X and y in train and test
- Show the number of samples in train and test, show the number of features

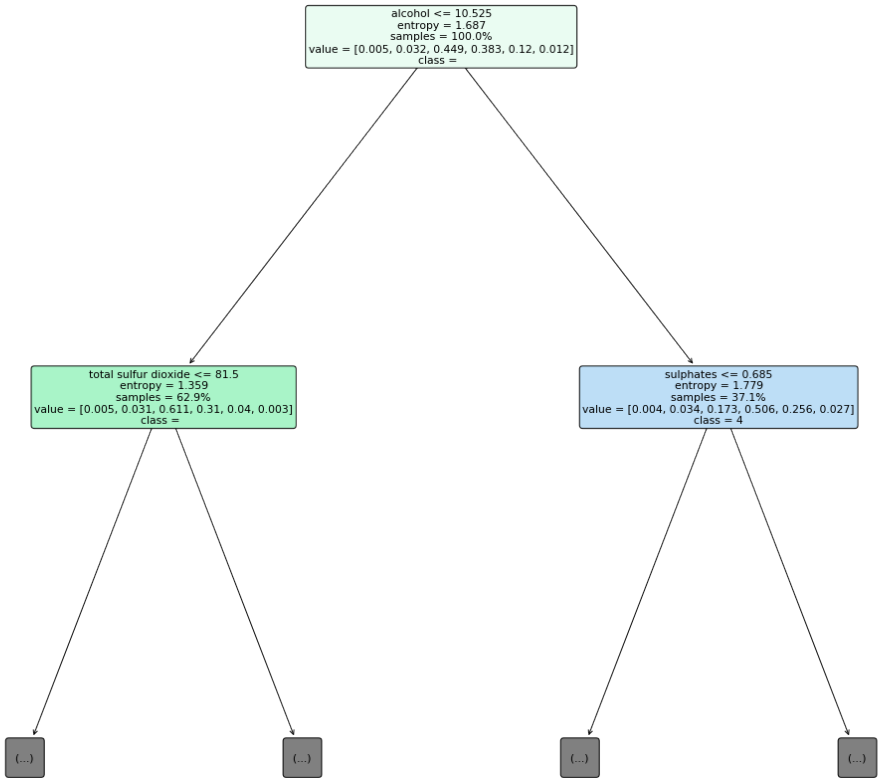
```
There are 1199 samples in the training dataset
There are 400 samples in the testing dataset
Each sample has 11 features
```

1.2 Part 1

- Initialize an estimator with the required model generator `tree.DecisionTreeClassifier(criterion="entropy")`
- Fit the estimator on the train data and target

Look at the tree structure - the feature names are used to show the tests in the nodes - they are the column names in the X - the class names - the attribute `estimator.classes_` contains the array of classes detected in the target; if the classes are numbers they have to be transformed in strings with `str()` - the dept of the visualization can be limited with the parameter `max_depth`

```
plt.figure(figsize = (20,20))  tree.plot_tree(estimator           , filled=True
, feature_names = X.columns           , class_names = str(estimator.classes_)
, rounded = True           , proportion = True           , max_depth = 1
);
```



1.2.1 Part 1.1

Let's see how it works on training data - predict the target using the fitted estimator on the training data - compute the accuracy on the training set using `accuracy_score(<target>,<predicted_target) * 100`

`The accuracy on training set is 100.0%`

1.2.2 Part 1.2

That's more significant: how it works on test data - use the fitted estimator to predict using the test features - compute the accuracy and store it on a variable for the final summary - store the maximum depth of the tree, for later use - `fitted_max_depth = estimator.tree_.max_depth` - store the range of the parameter which will be used for tuning - `parameter_values =`

`range(1,fitted_max_depth+1)` - print the accuracy on the test set and the maximum depth of the tree

```
The accuracy on test set is 55.8%
The maximum depth of the fitted tree is 17
```

1.3 Part 2

Optimising the tree: limit the maximum tree depth. We will use the three way splitting: `train, validation, test`. For simplicity, since we already splitted in *train* and *test*, we will furtherly split the *train* - split the training set into two parts: **train_t** and **val** - max_depth - pruning the tree cutting the branches which exceed max_depth - the experiment is repeated varying the parameter from 1 to the depth of the unpruned tree - the scores for the various values are collected and plotted

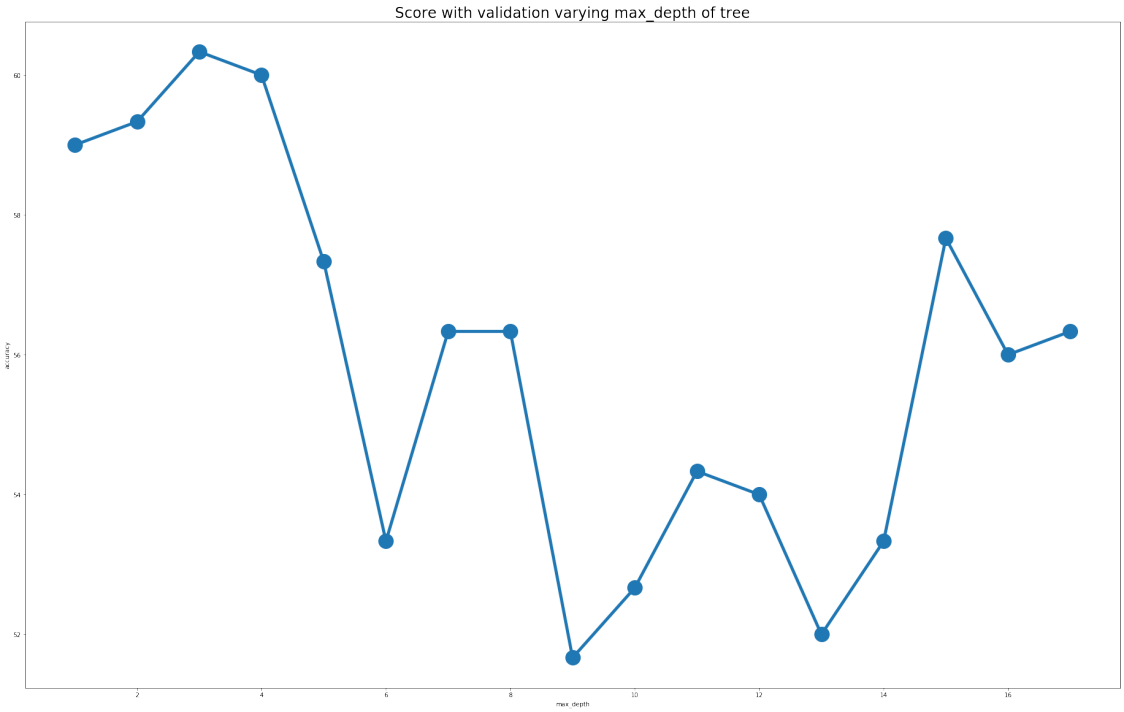
```
There are 899 samples in the training dataset
There are 300 samples in the validation dataset
```

1.3.1 Loop for computing the score varying the hyperparameter

- initialise a list to contain the scores
- loop varying `par` in `parameter_values`
  - initialize an estimator with a DecisionTreeClassifier, using `par` as maximum depth and `entropy` as criterion
  - fit the estimator on the `train_t` part of the features and the target
  - predict with the estimator using the validation features
  - compute the score comparing the prediction with the validation target and append it to the end of the list

1.3.2 Plot the results

Plot using the `parameter_values` and the list of `scores`



1.3.3 Fit the tree after validation and print summary

- store the parameter value giving the best score with `np.argmax(scores)`
- initialize an estimator as a DecisionTreeClassifier, using the best parameter value computed above as maximum depth and `entropy` as criterion
- fit the estimator using the `train` part
- use the fitted estimator to predict using the test features
- compute the accuracy on the test and store it on a variable for the final summary
- print the accuracy on the test set and the best parameter value

```
The top accuracy is 56.5%
Obtained with max_depth = 3
```

1.4 Part 3 - Tuning with Cross Validation

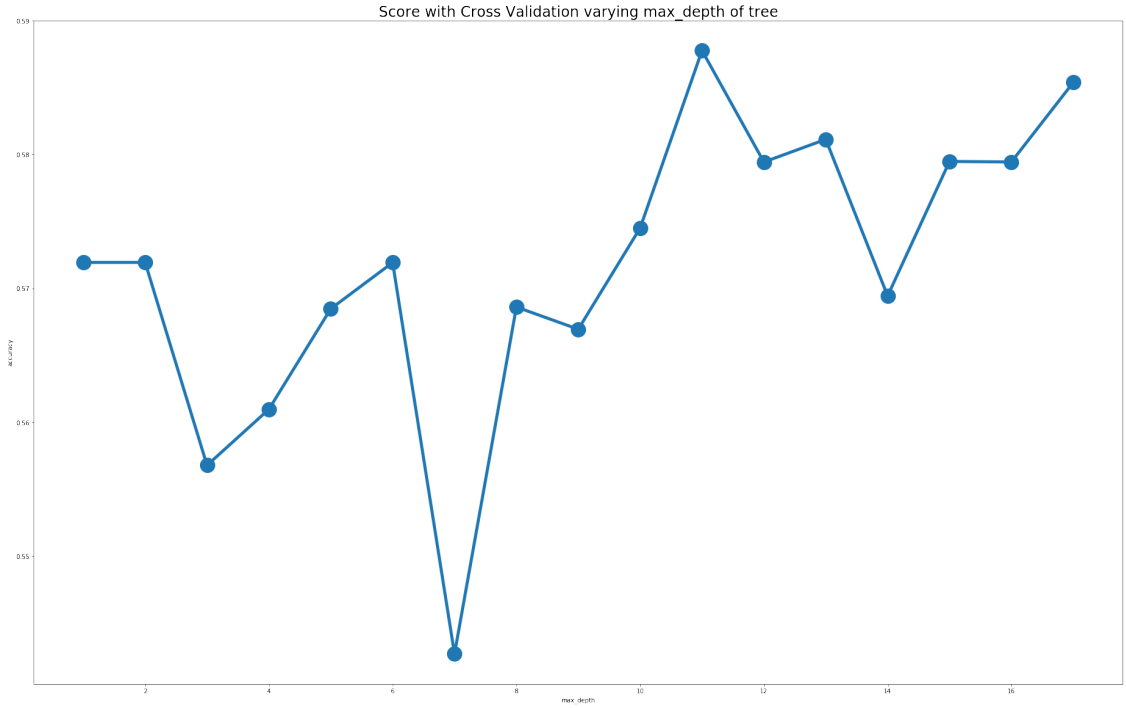
Optimisation of the hyperparameter with **cross validation** (cv suffix in the variable names). Now we will tune the hyperparameter looping on cross validation with the **training set**, then we will fit the estimator on the training set and evaluate the performance on the **test set**

- initialize an empty list for the scores
- loop varying `par` in `parameter_values`
  - initialize an estimator with a DecisionTreeClassifier, using `par` as maximum depth and `entropy` as criterion
  - compute the score using the estimator on the `train` part of the features and the target using

        * `cross_val_score(estimator, X_train, y_train, scoring='accuracy', cv = 5)`
        * the result is list of scores
    - compute the average of the scores and append it to the end of the list
- print the scores

```
[0.5719425034423883, 0.5719425034423883, 0.5567881329436533, 0.5609721472724145,
0.5684801424047305, 0.5719328293004885, 0.5426980949754188, 0.5686105170310605,
0.5669506744322675, 0.5745110660595276, 0.5877682351914538, 0.5794380948559146,
0.581133010110152, 0.569440387077933, 0.5794840742673417, 0.5794453947231194,
0.5853841873757648]
```

Plot using the `parameter_values` and the list of `scores`



1.4.1 Fit the tree after cross validation and print summary

- store the parameter value giving the best score with `np.argmax(scores)`
- initialize an estimator as a DecisionTreeClassifier, using the best parameter value computed above as maximum depth and `entropy` as criterion
- fit the estimator using the `train` part
- use the fitted estimator to predict using the test features
- compute the accuracy on the test and store it on a variable for the final summary
- print the accuracy on the test set and the best parameter value

```
The accuracy on test set tuned with cross_validation is 57.8% with depth 11
```

```
print(classification_report(y_test, y_predicted))
```

```
              precision    recall  f1-score   support

           3       0.20      0.25      0.22         4
           4       0.15      0.13      0.14        15
           5       0.60      0.74      0.66       143
           6       0.68      0.53      0.59       179
           7       0.45      0.51      0.48        55
           8       0.00      0.00      0.00         4

    accuracy                           0.58       400
   macro avg       0.35      0.36      0.35       400
weighted avg       0.59      0.58      0.57       400
```

- **micro**: Calculate metrics globally by counting the total true positives, false negatives and false positives.
- **macro**: Calculate metrics for each label, and find their unweighted mean. This does not take label imbalance into account.
- **weighted**: Calculate metrics for each label, and find their average weighted by support (the number of true instances for each label). This alters 'macro' to account for label imbalance; it can result in an F-score that is not between precision and recall.

```
print(confusion_matrix(y_test, y_predicted))
```

```
[[  1   1   1   0   1   0]
 [  3   2   5   4   1   0]
 [  1   8 106  22   6   0]
 [  0   1  57  94  25   2]
 [  0   1   7  17  28   2]
 [  0   0   2   1   1   0]]
```

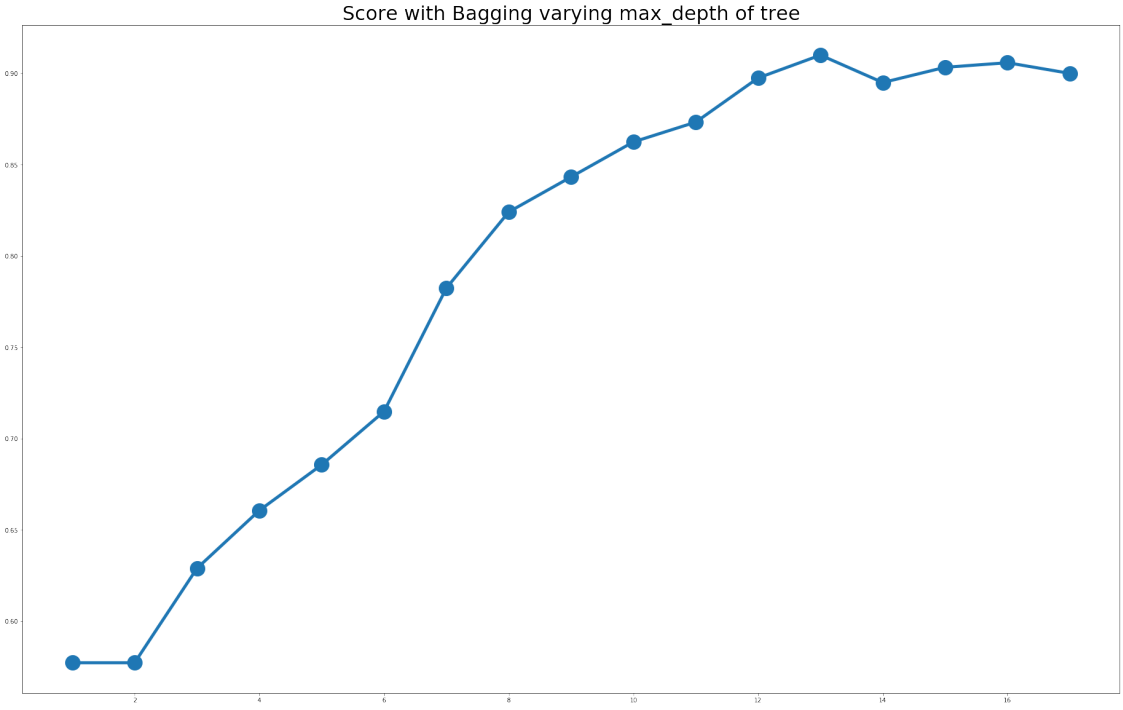
2 4. Tuning with an ensemble method

We will use the **bagging** method, made available by `scikit-learn`, for documentation see the pdf file provided, or the online documentation. - initialize an empty list for the scores - loop varying `par` in `parameter_values` - initialize an estimator with a BaggingClassifier applied to a DecisionTreeClassifier, using `par` as maximum depth and `entropy` as criterion (see below the statement) - fit the estimator on the `train` part - compute the score using the `score` method of the estimator on the `train` part of the features and the target, append the score to the end of the list - print the scores

```
estimator_bagging = BaggingClassifier(tree.DecisionTreeClassifier(criterion="entropy",
max_depth = par)                                        , max_samples=0.5,
max_features=0.5)
```

```
[0.5771476230191827, 0.5771476230191827, 0.6288573811509591, 0.6605504587155964,
0.6855713094245204, 0.7147623019182652, 0.7823185988323603, 0.8240200166805671,
0.8432026688907422, 0.8623853211009175, 0.8732276897414513, 0.8974145120934112,
0.9099249374478732, 0.8949124270225187, 0.9032527105921602, 0.9057547956630525,
0.8999165971643036]
```

Plot the scores, as done in the previous cases



2.0.1 Fit the tree after bagging and print summary

- store the parameter value giving the best score with `np.argmax(scores)`
- initialize an estimator as above, using the best parameter value computed above as maximum depth and `entropy` as criterion
- fit the estimator using the `train` part
- use the fitted estimator to predict using the test features
- compute the accuracy on the test and store it on a variable for the final summary
- print the accuracy on the test set and the best parameter value

```
The accuracy on test set tuned with bagging is 59.8%
Obtained with max_depth = 13
```

Print a summary of the four experiments

```
                                    Accuracy   Hyperparameter
Simple HoldOut and full tree        :   55.8%      17
HoldOut and tuning on validation set:   56.5%      3
CrossValidation and tuning          :   57.8%      11
Ensemble Bagging and tuning         :   59.8%      13

The scikit-learn version is 0.21.3.
```

2.0.2 Suggested exercises

- try other datasets
- try to optimise the parameters “min_impurity_decrease” “min_samples_leaf” and “min_samples_split”