P04 – Decision Trees

1. Background

In this lab you will use decision trees to classify offers of used cars. This lab is designed to give you a good understanding of how decision trees work and how to apply them. You should also get a feeling for what overfitting is and how to handle it. The accompanying template `decisiontrees_TASK.ipynb` is an IPython notebook. If these notebooks are new to you, have a look at Section 3.

Generally, decision trees are categorized into two classes: binary and non-binary. The binary decisions trees are more common in machine learning because of their reasonable computational complexity. Let's assume that there is a trained binary decision tree. A trained binary decision tree contains information to make a decision at any node of the tree (it knows which feature to split on and the corresponding decision criterion or threshold). Additionally, the class probabilities of the training samples are saved at the leaves of the tree.

The evaluation (test) procedure of a binary tree is as follows:

- A test sample is routed down the tree based on the binary decisions according the certain features at any given node
- Once the sample reaches a leaf, the class is predicted based on the class probability of training samples existing at that leaf

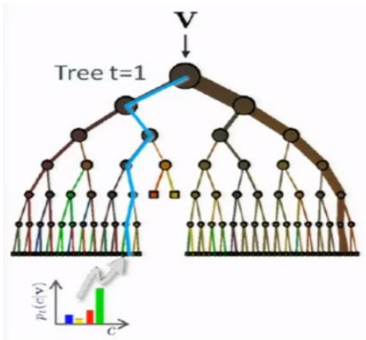


Figure 1: Routing a test sample down the tree to arrive at a probabilistic classification.

Training a classification tree:

- Step 1: Find the most discriminant feature for training
- Step 2: Compute the best decision threshold
- Step 3: Split the training data based on the threshold and chosen feature
- Step 4: Create a subtree for each split, go to 1 until a stopping criterion is met

Examples of stopping criterion for training decision trees are as follows:

- Depth of the tree
- Number of remaining training samples in leaves

The most discriminant feature per node can be determined using one of the following criterions:

- Gini impurity (see this lab)
- Information Gain (see lecture)
- Variance reduction

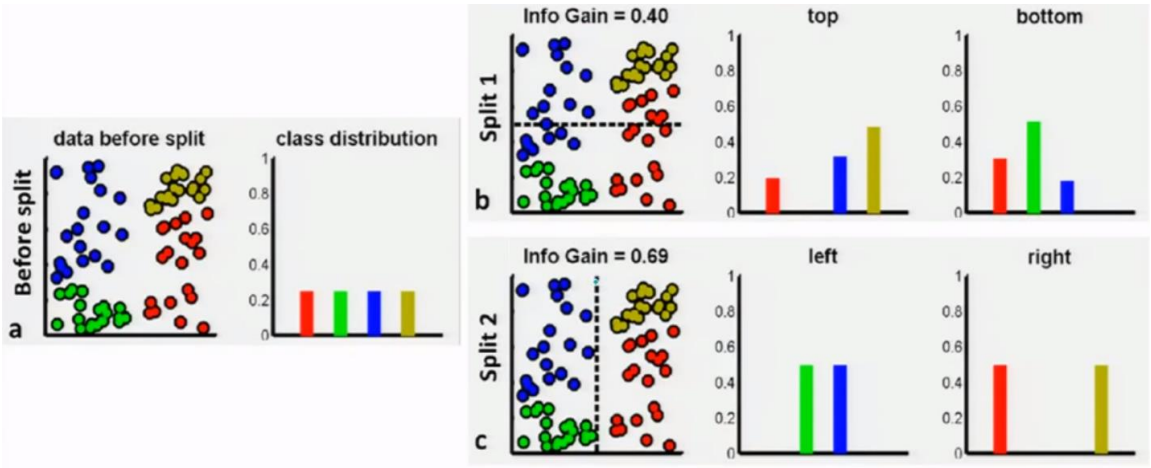


Figure 2: Using the Information Gain criterion to select the most discriminant attribute for splitting. Split 2 (lower right) gives the most information gain and is thus preferred.

Random Forest (RF) introduced by Breiman is an extension of decision trees. A number of decision trees (an ensemble) are trained based on random subsets of features and training samples and then their decisions are combined, leading to robust classifiers (see V09). You can gain additional information through watching the following lecture: https://www.youtube.com/watch?v=3kYujfDgmNk [optional].

Another way of building an ensemble is by using AdaBoost. The essential idea of the AdaBoost algorithm is to combine an ensemble of simple weak-learners (with binary classification accuracy barely over 50%) in order to develop a complex model. The weak learners in our lab are decision trees. Using AdaBoost, we iteratively train decision trees to correct the missclassifications of the previous trees.

Here are the steps for implementing AdaBoost:

- Step 1: Train a classifier based on the weighted samples (initially equal)
- Step 2: Compute predictions
- Step 3: Compute the weighted error rate based on predictions

- Step 4: Update the weights for each sample
- Step 5: Go to 1 on the re-weighted examples until stopping criterion is met

The weighted sum of all decision tree classifications makes up the final decision of an AdaBoost classifier. The decision weights are antiproportional to the weighted error rate computed in step 3. These weights are normalized to one once all decision trees are trained. Further explanation is provided in the following video: https://www.youtube.com/watch?v=ix6IvwbVpw0 [optional].

2. Exercise

At the top of the template, there are some helper functions, which should make it easier for you to implement your own decision trees, as well as some data preprocessing code. The main class of helper functions is called "Tree_node". Overall, you are asked to complete the following tasks:

1. ***Start small***
   At first you are asked to train a tree-stump (a tree of depth 1). In order to do so you need to complete the function `find_best_split()` which finds the best split with respect to the Gini impurity criterion. Don't get fancy here – it is best to try every possible split (this is why decision trees are slow for numerical variables). The function `find_best_split()` finds the best split through an exhaustive search of all possible features and split values. The training data and target column (labels) are inputs of this function and it returns the best split and the corresponding divided set.
   During working on `find_best_split()`, you will make use of the following functions:
   - `divideset(training_data, given_feature, value)`: divides the training data into two subsets based on a binary decision of a value for a given feature
   - `gini_impurity(data, labels, weights=None)`: computes the Gini impurity of a subset of data given the labels. In order to handle cost-aware loss functions later on, we can incorporate weights in the calculation; however, the classes are equally weighted by default

2. ***Confusing error rates***
   Compute the confusion matrix (predicted vs actual label[1]), as well as the overall prediction error, for the training set and the test set. Any comments?

[1] See https://en.wikipedia.org/wiki/Confusion_matrix for an explanation.

- Hint: Use the `classify` method from the `Tree_node` class to compute the predictions for a confusion matrix

3. ***We have to go deeper***[2]
   Complete the `train_tree()` method to recursively train deeper trees. If you have reached a leaf, return `in_data`, otherwise return a new `Tree_node`. Train a tree of depth 5: does this tree perform better than the tree stump from task 2.1?

4. ***This is much easier!***
   Use the `sk-learn`[3] class `DecisionTreeClassifier()` to train another tree of depth 5 (also based on the Gini impurity). Does this perform different from your own implementation?
   - Hint: you have to use the encoded versions of the data because the `DecisionTreeClassifier` only takes numerical labels.

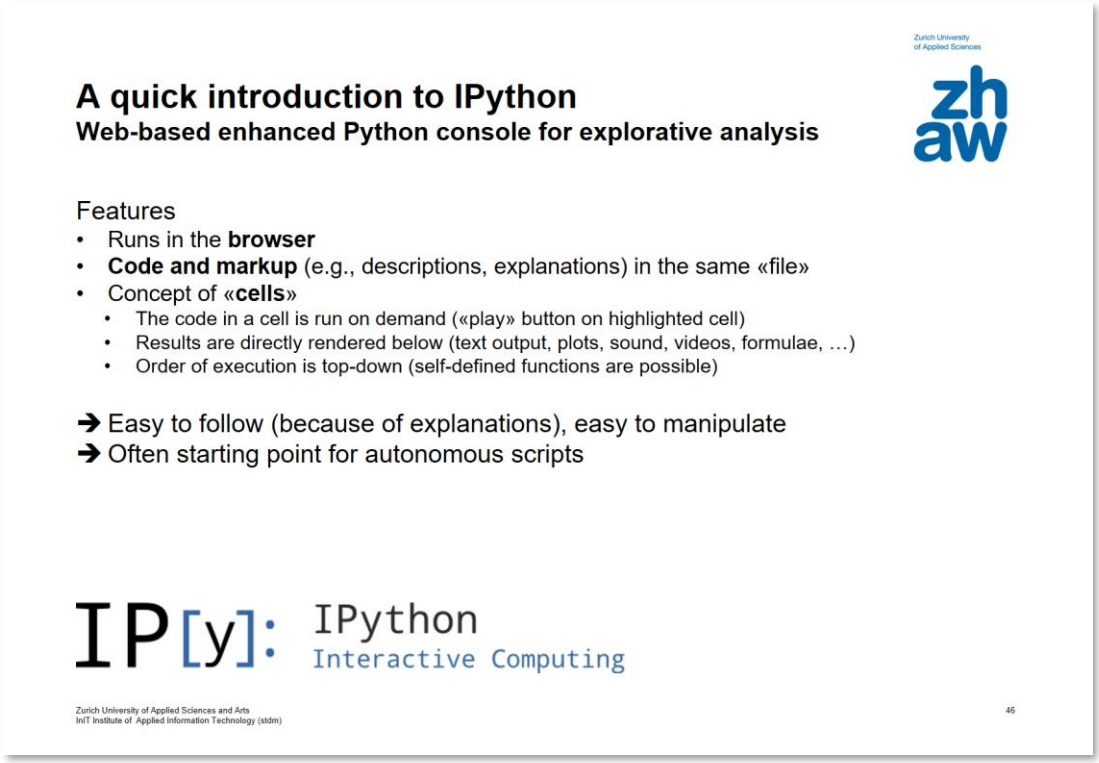
5. ***Machine-learners cookbook,***
   step 1: throw more computing power at the problem
   step 2: if necessary repeat step 1
   Create your own implementation of AdaBoost by completing the skeleton `ada_boost_trees`. For performance reasons, base this algorithm on the `DecisionTreeClassifier` and not on your own decision tree implementation. Use decision trees of depth 5 as base classifiers. Does the boosted tree perform better?

6. ***This is much easier! v2.0***
   Compare your AdaBoost implemention against `sk-learn`'s `AdaBoostClassifier`: are there significant performance differences?

[2] A famous meme in machine learning due to the «Inception» architecture of deep neural networks; the curious finds satisfaction at https://medium.com/initialized-capital/we-need-to-go-deeper-a-practical-guide-to-tensorflow-and-inception-50e66281804f.
[3] Scikit-learn is the standard machine learning library in Python: https://scikit-learn.org/stable/.

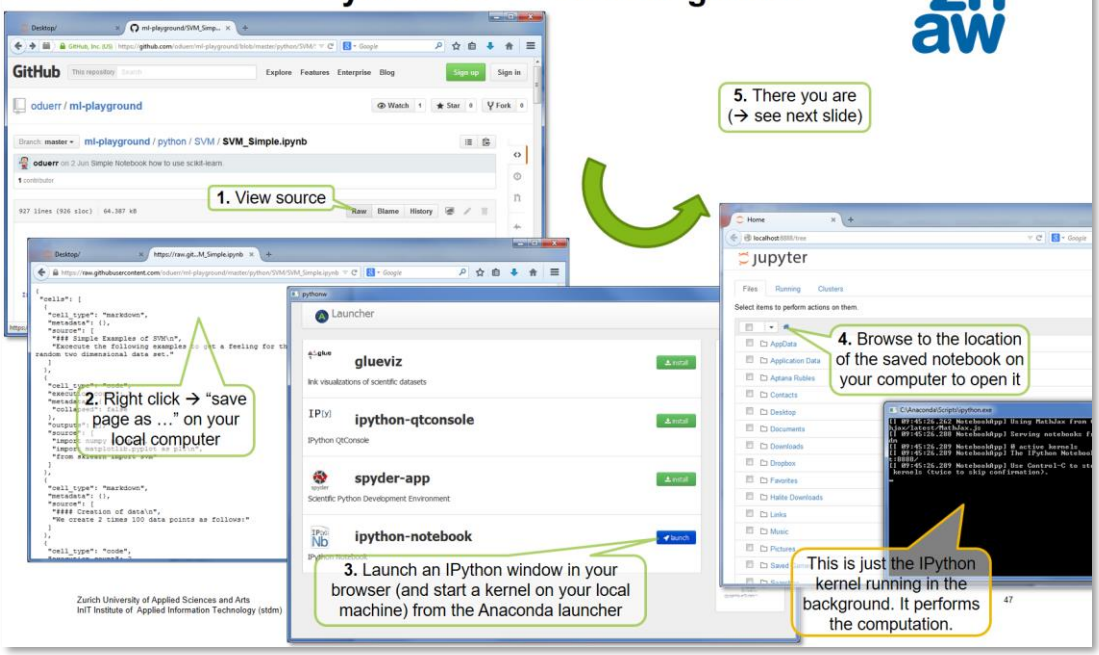
3. A quick introduction to IPython

IPython notebooks (also called Jupyter notebooks[4] since a while, where project Jupyter offers kernels also for many other languages besides Python) are basically code & documentation together in your browser for exploratory work. The following slides give an overview and quick-start guide:



[4] See https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html for more background, a brief description of the main concepts, and further links.

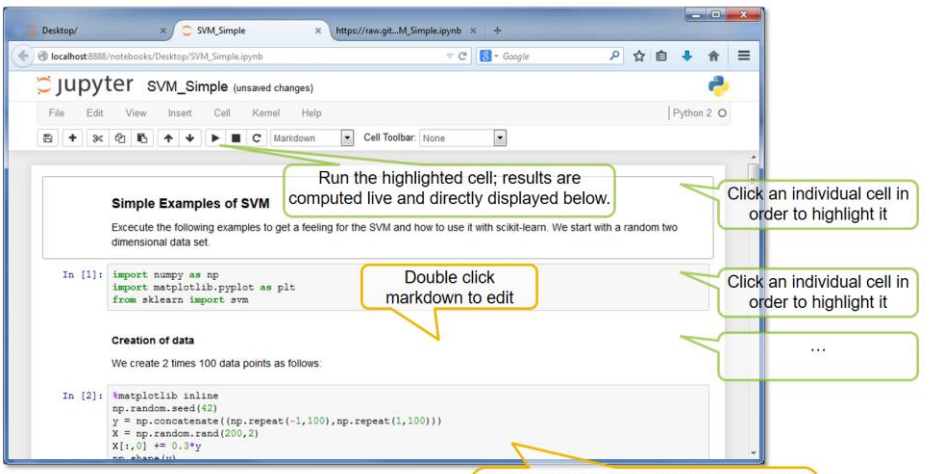
How to run an IPython notebook from github?



Zurich University of Applied Sciences and Arts
InIT Institute of Applied Information Technology (stdm)

47

How to operate an IPython notebook?



Zurich University of Applied Sciences and Arts
InIT Institute of Applied Information Technology (stdm)

48