Lab Class ML:III

Until **Monday, Dec. 9th, 2019, 11:00 am CET**, the following exercises must be submitted:

Students receiving 4.5 ECTS: 2, and 3.
Students receiving 6 ECTS: 1, 2, and 3.

Lab Class General Instructions. Exercises should be completed in groups of three students; which and how many exercises you must submit depends on the number of ECTS credits awarded for the course, based on which degree programme you're studying:

	4.5 ECTS	6 ECTS
Degree programmes	CS4DM, CSM, MI, HCI*	DE, HCI*
	*(HCI students can choose number of credits in electives module)	
For admission to the final exam:		
Must reach at least	80 points	120 points
Out of maximum	120 points	180 points

For each exercise sheet, all required exercises must be submitted until the stated deadline. Each individual exercise will receive a score between zero and ten. Submissions received after the deadline will not be considered for grading. In order to be admitted to the final exams, you must reach the minimum total score shown.
Upload your solutions to Moodle as a single `.pdf` file, plus one `.ipynb` file for any programming exercises. Make it clear which solution corresponds to which exercise. Programming exercises are marked with [P] – their solutions must be documented extensively with docstrings and/or inline comments. ml.weimar.webis.de

Exercise 1 : Decision Trees

Construct by hand a decision tree corresponding to each of the following Boolean functions. The examples $\mathbf{x} \in D$ have one attribute for each Boolean variable A, B, ... in the formula; the target concept $c(\mathbf{x})$ is the truth value of the formula itself. Assume the set D contains examples with all possible combinations of attribute values.

Hint: It may be helpful to write out the set D, i.e., the truth table for the variables and formula, first.

(a) $A \wedge \neg B$

(b) $A \vee (B \wedge C)$

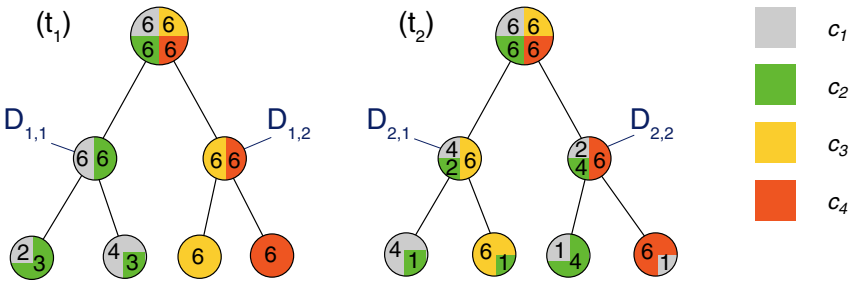
(c) A XOR B

(d) $(A \wedge B) \vee (C \wedge D)$

(e) $(A \vee B) \wedge (C$ XOR $D)$

Exercise 2 : Impurity Functions

Let D be a set of examples over a feature space X and a set of classes $C = \{c_1, c_2, c_3, c_4\}$, with $|D| = 24$. Consider the following illustration of two possible decision trees, t_1 and t_2 – the colors represent the classes present in each document set associated with the nodes of the trees; the numbers denote how many examples of each class are present.



(a) First, consider only the first split that each of the two trees makes: compute $\Delta\iota(\{D_{1,1}, D_{1,2}\})$ and $\Delta\iota(\{D_{2,1}, D_{2,2}\})$ with the misclassification rate $\iota_{misclass}$ and the entropy criterion $\iota_{entropy}$ as splitting criterion. Interpret the results: which of $\{D_{1,1}, D_{1,2}\}$ or $\{D_{2,1}, D_{2,2}\}$ is the better first split?

(b) Which of t_1, t_2 is the better decision tree, and why?

(c) Assuming the splits shown are the only possibilities, which of t_1 or t_2 would the ID3 algorithm construct, and why?

Exercise 3 : P Algorithm ID3 and Measuring Performance

Develop a basic Python implementation of the ID3 algorithm discussed in the lecture. Use the mushroom example data from the slides to develop and test your implementation. For convenience, we will represent the set of examples D as a list of Python dictionaries mapping attributes to their values:

```
mushrooms = \
[{"Color": "red", "Size": "small", "Points": "yes", "Eatability": "toxic"},
 {"Color": "brown", "Size": "small", "Points": "no", "Eatability": "edible"},
 {"Color": "brown", "Size": "large", "Points": "yes", "Eatability": "edible"},
 {"Color": "green", "Size": "small", "Points": "no", "Eatability": "edible"},
 {"Color": "red", "Size": "large", "Points": "no", "Eatability": "edible"}]
```

(a) Implement a function `conditional_entropy(examples, attribute, target)` where `examples` is a list of dictionaries like above, and `attribute` and `target` are strings. The function should return the conditional entropy $H(\texttt{target} \mid \texttt{attribute})$.

You may use the following *doctest*-style test cases to verify your implementation:

```
>>> D = [dict(a=a,b=b,c=c,d=d) for a,b,c,d in zip("1111", "1122", "1212", "1233")]
>>> conditional_entropy(D, "a", "d")
1.5
>>> conditional_entropy(D, "b", "d")
0.5
>>> conditional_entropy(D, "c", "d")
1.0
>>> conditional_entropy(D, "d", "d")
0.0
>>> conditional_entropy(D, "d", "c")
0.5
```

Hints: Refer to the lecturenotes. You may find it easier to first define helper functions that take a set D of examples and compute (1) the prior probabilities $P(B_j)$ for all possible attribute-value assignments B_j, and (2) the posterior probabilities $P(A_i|B_j)$ for possible values A_i of the target attribute $\mathcal{A}$ given the B_j. Remember that $\log_2(0)$ needs to be treated specially (cf. slide ML:III-55).

(b) Develop a class `ID3Node` which holds all the information related to one node in a decision tree, including: which attribute is tested by the node; the node label; the children in the decision tree, and which attribute value corresponds to which child. Your class should have at least the following two methods:

- `create_edge(self, attribute_value, child)`, where `child` is another `ID3Node`, and `attribute_value` is the corresponding value of the attribute tested by `self`.
- `classify(self, example)` where `example` is a single dictionary like those in the list above, and the return value is is one of the possible values of the target attribute.

You may use the following *doctest* example to verify that your implementation can construct a minimal decision tree:

```
>>> t = ID3Node()
>>> t1 = ID3Node()
>>> t2 = ID3Node()
>>> t1.label = True
>>> t1.classify({})
True
>>> t2.label = 42
>>> t2.classify({})
42
>>> t.attribute = "points"
>>> t.create_edge("yes", t1)
>>> t.create_edge("no", t2)
>>> t.classify({"points": "yes"})
True
>>> t.classify({"points": "no"})
42
```

Hint: refer to the documentation if you are new to object-oriented programming with Python.

(c) Implement the function `id3(examples, attributes, target)` that builds a decision tree according to the ID3 algorithm from the lecture and returns the root node. The set of `examples` is represented as a list of dictionaries as above, `attributes` is a set of strings with the names of the predictors, and `target` is a string with the name of the response variable.

```
>>> t = id3(mushrooms_examples, {'Color', 'Size', 'Points'}, 'Eatability')
>>> t.classify({'Color': 'red', 'Points': 'yes', 'Size': 'small'})
'toxic'
>>> t.classify({'Color': 'brown', 'Points': 'yes', 'Size': 'large'})
'edible'
```

(d) Implement the function `misclassification_rate(tree, examples, target)`, where `tree` is the root node of a decision tree, and the other parameters defined as above. Verify that a tree trained using your `id3` function on all five mushroom examples achieves a misclassification rate of zero when tested on the same examples.

(e) Download the "Adult" dataset from the UC Irvine Machine Learning Repository. A dataset description can be found at `https://archive.ics.uci.edu/ml/datasets/Adult`, and the data file itself at `https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data`.

Load the dataset into a `pandas.DataFrame` using the `read_csv` function. For correct operation, you must pass the `names` parameter, and provide the names of all 15 columns that exist in the dataset (see the section "Attribute Information" in the description, and note that the final column, "income" is not listed there explicitly). Remove all continuous attributes, so that 9 columns remain in total. Split this dataset into a training set (the first 20 000 rows) and test set (the remaining 12 561

rows), for example with the help of `DataFrame.iloc`. Finally, apply your `id3` implementation to your training set, and compute the misclassification rate of the resulting tree on your test set. In both cases, use the final column "income" as target.

Hint: Refer to the `DataFrame.to_dict` function (and its parameter `orient`) to help convert the data to the list-of-dictionaries format that your implementation expects.