Assignment 1 - Decision Trees and Clustering Techniques

Aprendizagem Computacional - MEI | Computação Neuronal e Sistemas Difusos - MIEB

by Catarina Silva and Marco Simões

_

This assignment will assess the students knowledge on the following Machine Learning topics:

- Decision Trees
- Clustering Techniques

The assignment is split into two sub-assignments: 1-a) Decision Trees (first week) and 1-b) Clustering Techniques (second week).

Students should implement their solutions and answering the questions directly in the notebooks, and submit both files together in Inforestudante before the deadline: *06/10/2021*

Conditions:

- *Groups:* two elements of the same PL class
- *Duration:* 2 weeks
- *Workload:* 8h per student

Assignment 1 - a) Decision Trees

Consider the depression dataset, from Agresti, A. (2019). *An introduction to categorical data analysis (2nd ed.). John Wiley & Sons.* This dataset is composed by evaluations of 335 patients during 3 phase treatment. We want to learn a decision tree that, given the attributes A - Diagnosis Severity (0: Mild, 1: Severe), B - Treatment Type (0: Standard, 1: New drug) and C - Follow Up Time (0: 1 week, 1: 2 weeks, 2: 4 weeks), predicts D - Depression Outcome (0: Normal, 1: Abnormal).

In [ ]:

```
import pandas as pd
import numpy as np
... # TODO add extra imports if needed


# load data
data = pd.read_csv('depression.csv')
```

Ex. 1

Create a function `attr_probs( data, attr )` that, given the dataset (`data`) and a attribute id (`attr`), computes the percentage of cases with Abnormal treatment outcome (D) for each attribute *value*. The function should return a dictionary with the different attribute values as keys and the correspondent percentages as values. Example: `attr_probs( data, 'A')` -> returns `{0: 0.30, 1: 0.23}`

In [ ]:

```
OUTCOME = 'D'

def attr_probs( data, attr):

    probs = {}
    # TODO CODE HERE

    return probs
```

Ex. 2

Create a function `entropy( probs )` that, given a list probability values, returns the correspondent **entropy** value.

In [ ]:

```
def entropy( probs ):
    # TODO CODE HERE
```

In [ ]:

```
# example
print(entropy([2/8, 0/8, 4/8, 2/8])) # should print 1.5
```

Ex. 3

Create a function `gain( data, attr )` to compute the gain of an attribute. Make use of the functions developed in the previous exercises.

In [ ]:

```
def gain( data, attr ):
    # TODO CODE HERE
```

Ex. 4

Run the following code to compute the gain for the different attributes. In what does those results influence the design of the decision tree?

In [ ]:
```
ATTRS = ['A', 'B', 'C']
for attr in ATTRS:
    print('Gain {attr}: {gain:.2f}'.format(attr=attr, gain=gain(data, attr)))
```

Answer:

```
TODO write answer here ...
```

Ex. 5

Split the dataset into two sets (train set and test set), assigning randomly 70% of the cases to the train set and the remaining 30% to the test set. Use the `train_test_split` method from the `sklearn.model_selection` module, specifying the `random_state` with a value of 7 for reproducibility purposes.

Train a `DecisionTreeClassifier` (from the `sklearn.tree` module) using the training data. Enforce the use of the `entropy` criterion instead of the `gini` criterion.

Resort to the function `export_text` from the `sklearn.tree` module to visualize the structure of the resulting tree. Are the results of **Ex. 4** congruent with the tree obtained here? Justify.

In [ ]:
```
# TODO CODE HERE
```

Answer:

```
TODO write answer here ...
```

Ex 6

Looking for the structure of the tree printed, evaluate the following cases (by hand) and provide the outcome class for each case, as well as the path from the root to the leaf (meaning, provide the conditions it evaluated as true to reach that class).

Cases:

c1 = (A=1, B=0, C=2)

c2 = (A=0, B=0, C=0)

c3 = (A=0, B=0, C=1)

c4 = (A=1, B=1, C=0)

Example:

case: cx = (A=1, B=1, C=1)

path: (C <= 1.5) --> (A > 0.5) --> (C > 0.5) --> (B > 0.5) --> class 1

Answer:

case: c1 = (A=1, B=0, C=2)

path: `TODO write answer here ...`

_

case: c2 = (A=0, B=0, C=0)

path: `TODO write answer here ...`

_

case: c3 = (A=0, B=0, C=1)

path: `TODO write answer here ...`

_

case: c4 = (A=1, B=1, C=0)

path: `TODO write answer here ...`

Ex. 7

Apply the decision tree trained in the previous exercise to the test data. Compare the predicted labels to the true labels, generating a confusion matrix (you can use the `confusion_matrix` function of the `sklearn.metrics` module for that). Report the **percentage** of `True Positives, True Negatives, False Positives and False Negatives`, as well as the metrics `accuracy, precision, recall and f1-score`.

In [ ]:
```
# TODO CODE HERE
```

Ex. 8

Repeat the process of spliting the data, training the classifier and testing the classifier 100 times (use the values from 0 to 99 as `random_state` for the `train_test_split` function). Plot the accuracy across the 100 repetitions, reporting also its mean value and standard deviation.

In [ ]:

```
# TODO CODE HERE
```