# COSC 2753 | Machine Learning

## Week 6 Lab Exercises: **Decision Trees**

## Introduction

During the last couple of weeks we learned about the typical ML model development process. In this weeks lab we will explore decision tree based models.

The lab assumes that you have completed the labs for week 2-5. If you havent yet, please do so before attempting this lab.

The lab can be executed on either your own machine (with anaconda installation) or lab computer.

### Objective

- Continue to familiarise with Python and other ML packages.
- Learning classification decision trees from both categorical and continuous numerical data
- Comparing the performance of various trees after pruned.
- Learning regression decision trees and comparing these models to regression models from previous labs.

### Dataset

The data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be (or not) subscribed.

Input variables:

- Bank client data:
  1. age (numeric)
  2. job : type of job (categorical: "admin.","unknown","unemployed","management","housemaid","entrepreneur","student", "blue-collar","self-employed","retired","technician","services")
  3. marital : marital status (categorical: "married","divorced","single"; note: "divorced" means divorced or widowed)
  4. education (categorical: "unknown","secondary","primary","tertiary")
  5. default: has credit in default? (binary: "yes","no")
  6. balance: average yearly balance, in euros (numeric)
  7. housing: has housing loan? (binary: "yes","no")
  8. loan: has personal loan? (binary: "yes","no")
- Related with the last contact of the current campaign:
  9. contact: contact communication type (categorical: "unknown","telephone","cellular")
  10. day: last contact day of the month (numeric)
  11. month: last contact month of year (categorical: "jan", "feb", "mar", ..., "nov", "dec")
  12. duration: last contact duration, in seconds (numeric)
- Other attributes:
  13. campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)
  14. pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric, -1 means client was not previously contacted)
  15. previous: number of contacts performed before this campaign and for this client (numeric)
  16. poutcome: outcome of the previous marketing campaign (categorical: "unknown","other","failure","success")

Output variable (desired target):

```
17. y - has the client subscribed a term deposit? (binary: "yes","no")
```

This dataset is public available for research. The details are described in Moro et al., 2011.

Moro et al., 2011: S. Moro, R. Laureano and P. Cortez. Using Data Mining for Bank Direct Marketing: An Application of the CRISP-DM Methodology. In P. Novais et al. (Eds.), Proceedings of the European Simulation and Modelling Conference - ESM'2011, pp. 117-121, Guimarães, Portugal, October, 2011.

Lets read the data first.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

data = pd.read_csv('./Lab/bank-full.csv', delimiter=';')
data.head()
```

| | age | job | marital | education | default | balance | housing | loan | contact | day | month | duration | campaign | pdays | previous | poutcome | y |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 58 | management | married | tertiary | no | 2143 | yes | no | unknown | 5 | may | 261 | 1 | -1 | 0 | unknown | no |
| 1 | 44 | technician | single | secondary | no | 29 | yes | no | unknown | 5 | may | 151 | 1 | -1 | 0 | unknown | no |
| 2 | 33 | entrepreneur | married | secondary | no | 2 | yes | yes | unknown | 5 | may | 76 | 1 | -1 | 0 | unknown | no |
| 3 | 47 | blue-collar | married | unknown | no | 1506 | yes | no | unknown | 5 | may | 92 | 1 | -1 | 0 | unknown | no |
| 4 | 33 | unknown | single | unknown | no | 1 | no | no | unknown | 5 | may | 198 | 1 | -1 | 0 | unknown | no |

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   age        45211 non-null  int64
 1   job        45211 non-null  object
 2   marital    45211 non-null  object
 3   education  45211 non-null  object
```

```
 4   default    45211 non-null  object
 5   balance    45211 non-null  int64
 6   housing    45211 non-null  object
 7   loan       45211 non-null  object
 8   contact    45211 non-null  object
 9   day        45211 non-null  int64
 10  month      45211 non-null  object
 11  duration   45211 non-null  int64
 12  campaign   45211 non-null  int64
 13  pdays      45211 non-null  int64
 14  previous   45211 non-null  int64
 15  poutcome   45211 non-null  object
 16  y          45211 non-null  object
dtypes: int64(7), object(10)
memory usage: 5.9+ MB
```

The dataset contains categorical and numerical attributes. Lets convert the categorical columns to categorical data type in pandas.

```
for col in data.columns:
    if data[col].dtype == object:
        data[col] = data[col].astype('category')
```

sklearn's classification decision tree learner doesn't work with categorical attributes. It only works with continuous numeric attributes. The target class, however, must be categorical. So the categorical attributed must be converted into a suitable continuous format. Helpfully, Pandas can do this.

First, split the data into the target class and attributes:

```
dataY = data['y']
dataX = data.drop(columns='y')
```

Then use Pandas to generate "numerical" versions of the attributes:

```
dataXExpand = pd.get_dummies(dataX)
dataXExpand.head()
```

| | age | balance | day | duration | campaign | pdays | previous | job_admin. | job_blue-collar | job_entrepreneur | ... | month_jun | month_mar | month_may | month_nov | month_oct | month_sep | poutc |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 58 | 2143 | 5 | 261 | 1 | -1 | 0 | False | False | False | ... | False | False | True | False | False | False | |
| **1** | 44 | 29 | 5 | 151 | 1 | -1 | 0 | False | False | False | ... | False | False | True | False | False | False | |
| **2** | 33 | 2 | 5 | 76 | 1 | -1 | 0 | False | False | True | ... | False | False | True | False | False | False | |
| **3** | 47 | 1506 | 5 | 92 | 1 | -1 | 0 | False | True | False | ... | False | False | True | False | False | False | |
| **4** | 33 | 1 | 5 | 198 | 1 | -1 | 0 | False | False | False | ... | False | False | True | False | False | False | |

5 rows × 51 columns

```
dataXExpand.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 51 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   age                  45211 non-null  int64
 1   balance              45211 non-null  int64
 2   day                  45211 non-null  int64
 3   duration             45211 non-null  int64
 4   campaign             45211 non-null  int64
 5   pdays                45211 non-null  int64
 6   previous             45211 non-null  int64
 7   job_admin.           45211 non-null  bool
 8   job_blue-collar      45211 non-null  bool
 9   job_entrepreneur     45211 non-null  bool
 10  job_housemaid        45211 non-null  bool
 11  job_management       45211 non-null  bool
 12  job_retired          45211 non-null  bool
 13  job_self-employed    45211 non-null  bool
 14  job_services         45211 non-null  bool
 15  job_student          45211 non-null  bool
 16  job_technician       45211 non-null  bool
 17  job_unemployed       45211 non-null  bool
 18  job_unknown          45211 non-null  bool
 19  marital_divorced     45211 non-null  bool
 20  marital_married      45211 non-null  bool
 21  marital_single       45211 non-null  bool
 22  education_primary    45211 non-null  bool
 23  education_secondary  45211 non-null  bool
 24  education_tertiary   45211 non-null  bool
 25  education_unknown    45211 non-null  bool
 26  default_no           45211 non-null  bool
 27  default_yes          45211 non-null  bool
 28  housing_no           45211 non-null  bool
 29  housing_yes          45211 non-null  bool
 30  loan_no              45211 non-null  bool
 31  loan_yes             45211 non-null  bool
 32  contact_cellular     45211 non-null  bool
 33  contact_telephone    45211 non-null  bool
 34  contact_unknown      45211 non-null  bool
 35  month_apr            45211 non-null  bool
 36  month_aug            45211 non-null  bool
 37  month_dec            45211 non-null  bool
 38  month_feb            45211 non-null  bool
 39  month_jan            45211 non-null  bool
 40  month_jul            45211 non-null  bool
 41  month_jun            45211 non-null  bool
 42  month_mar            45211 non-null  bool
 43  month_may            45211 non-null  bool
 44  month_nov            45211 non-null  bool
 45  month_oct            45211 non-null  bool
 46  month_sep            45211 non-null  bool
 47  poutcome_failure     45211 non-null  bool
 48  poutcome_other       45211 non-null  bool
 49  poutcome_success     45211 non-null  bool
 50  poutcome_unknown     45211 non-null  bool
dtypes: bool(44), int64(7)
memory usage: 4.3 MB
```

As you can see, the categories are expanded into boolean (yes/no, that is, 1/0) values that can be treated as continuous numerical values. It's not ideal, but it will allow a correct decision tree to be learned.

� Why is it necessary to convert the attributes into boolean representations, rather than just convert them into integer values? What problem would be caused by converting the attributes into integers?

1. **Avoiding Implicit Ordering**: When you convert categorical data into integers, it introduces an artificial order or hierarchy that doesn't exist in the original data. For example, if you have a 'job' column with categories like 'admin', 'technician', 'entrepreneur', and you map them to integers like 1, 2, 3, the algorithm might incorrectly interpret these numbers as having an order (i.e., 'technician' > 'admin', 'entrepreneur' > 'technician'), which can lead to skewed or incorrect results.
2. **Equal Representation**: One-hot encoding treats all categories equally without implying any sort of ordinal relationship between them. Each category becomes a separate feature with equal weight, which is more representative of the true nature of categorical data.
3. **Model Compatibility**: Many machine learning models are designed to work with continuous data and can misinterpret the nature of the data if it's simply converted to integers. One-hot encoding transforms categorical data into a binary matrix that these models can process more effectively.

### Problems with Converting Attributes into Integers:

1. **Incorrect Assumptions by the Model**: As mentioned, the model might infer a non-existent ordinal relationship between categories. This can lead the model to draw false conclusions about the data.
2. **Distorted Distance Measurements**: In models that calculate distances between data points (like in K-Nearest Neighbors), integer encoding can cause distortion. The model might assume that two data points are close to each other because their integer representations are numerically close, even if they're actually quite different.
3. **Limiting Model Performance**: Integer encoding can limit the ability of the model to learn complex patterns in the data, especially if the categorical variable has no inherent order. This can result in poorer model performance.

The target class also needs to be pre-processed. The target will be treated by sklearn as a category, but sklearn requires that these categories are represented as integers (not strings). To convert the strings into numbers, the preprocessing. LabelEncoder class from sklearn can be used, as shown below. The two print statements show how to convert in both directions (strings to integers, and vice-versa).

```
from sklearn import preprocessing

le = preprocessing.LabelEncoder()
le.fit(dataY)
class_labels = le.inverse_transforam([0,1])
dataY = le.transform(dataY)
```

1. `le = preprocessing.LabelEncoder()` : This line is creating an instance of the `LabelEncoder` class.
2. `le.fit(dataY)` : The `fit` method is used to fit the label encoder instance to the data. This means it examines all the unique values in `dataY`, and assigns each unique value to a unique integer, which will be used to transform the categorical data into numerical data. The unique values are stored in the `classes_` attribute of the `LabelEncoder` instance.
3. `class_labels = le.inverse_transform([0,1])` : The `inverse_transform` method is used to convert the encoded numerical values back into their original categorical form. In this case, it's being used to get the original categorical values for the encoded values 0 and 1. These original values are stored in the `class_labels` variable.
4. `dataY = le.transform(dataY)` : The `transform` method is used to convert the categorical data in `dataY` into numerical data. It does this by replacing each unique value in `dataY` with the integer that was assigned to it when the `fit` method was called. The transformed data is then stored back into the `dataY` variable

We already convert "no"/"yes" into [0,1], try to print labels for checking

```
print(dataY)
print(len(dataY))
print(class_labels)
```

```
[0 0 0 ... 1 0 0]
45211
['no' 'yes']
```

## EDA

> **👉 Task: Since we have covered how to do EDA in the previous labs, this section is left as an exercise for you. Complete the EDA and use the information to justify the decisions made in the subsequent code blocks.**

[ ] ↳ *4 cells hidden*

## Setting up the performance (evaluation) metric

There are many performance metrics that apply to this problem such as `accuracy_score`, `f1_score`, etc. More information on performance metrics available in sklearn can be found at: https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics

The insights gained in the EDA becomes vital in determining the performance metric. Try to identify the characteristics that are important in making this decision from the EDA results. Use your judgment to pick the best performance measure - discuss with the lab demonstrator to see if the performance measure you came up with is appropriate.

In this task, I want to give equal importance to all classes. Therefore I will select `macro-averaged f1_score` as my performance measure and I wish to achieve a target value of 75% f1_score.

F1-score is NOT the only performance measure that can be used for this problem.

## Setup the experiment - data splits

Next **what data should we use to evaluate the performance?**

We can generate "simulated" unseen data in several methods

1. Hold-Out validation
2. Cross-Validation

Lets use hold out validation for this experiment.

> ✍ **Task: Use the knowledge from last couple of weeks to split the data appropriately.**

[ ] ↳ *6 cells hidden*

## Simple decision tree training

Lets train a simple decision tree and visualize it.

[ ] ↳ *24 cells hidden*

## Random forest

Lets make many trees using our dataset. If we run the DT algorithm multiple times on same data, it will result in the same tree. To make different trees we can inject some randomness. Select data data points and features to be used in DT algorithm randomly - this process is called creating boot strapped datasets.

This is automatically done in sklearn for us in the `RandomForestClassifier`. Lets use that on our dataset.

More code can be found here: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[ ] ↳ *10 cells hidden*

## Exercise: Regression Decision Tree

A regression decision tree can also be trained. These are decision trees where the leaf node is a regression function. You will investigate learning regression trees using the boston housing data set from previous labs.

The below code snippet will help get you started. Note that it does not make sense to use entropy for generating splits, so the default method from sklearn will be used. Also note that the DecisionTreeRegressor class uses similar pre-pruning parameters.

[ ] ↳ *4 cells hidden*