INF264
Project 1:
Implementing decision trees

Deadline: September 23th, 23.59
Deliver here:
`https://mitt.uib.no/courses/36686/assignments/60599`

Projects are a compulsory part of the course. This project contributes a total of 25 points to the final grade. You need to upload your answer to MittUiB before 23.59 on September 23th.

The project can be done either alone or in pairs. If you do the work with a pair, add a paragraph to your report explaining the division of labor (Note that both students will get the same grade regardless of the division of labor).

Grading will be based on the following qualities:

- Correctness (your answers/code are correct and clear)
- Clarity of code (documentation, naming of variables, logical formatting)
- Reporting (thoroughness and clarity of the report)

Deliverables. You should deliver **exactly two files**:

1. a PDF report containing an explanation of your approach and design choices to help us understand how your particular implementation works. You can include snippets of code in the PDF to elaborate any point you are trying make.

2. a zip file of your code. We may want to run your code if we feel necessary to confirm that it works the way it should. Please include a README.txt file in your zip file that explains how we should run your code. In case you have multiple files in your code directory, you must mention in the README.txt file which file is the main file that we need to run to execute your entire algorithm. Note that all the numbers that you report should be reproducible from the code that you return.

The reason why we ask you not to put the report inside the zip is that in this way the graders can use MittUiB's Speedgrader-functionality. So please make graders' work easier and follow the instructions.

Programming languages. You are allowed to submit your implementation in the following languages: Python, Matlab, R, C#, Java, Julia. (This list is based on the skills of the graders and no new languages will be added later.)

Teaching assistants support Python. If you choose to implement the project in some other programming language, you are on your own.

Code of conduct. The goal of the project is learning to implement machine learning algorithms. Therefore, you must write the code yourself. You are not allowed to use existing libraries. The decision tree implementation can import only basic packages like `numpy`; for model selection and evaluation as well as for visualisation you can import packages like `sklearn.metrics.accuracy_score` or `matplotlib`. Furthermore, it is not allowed to copy-paste code from online tutorials or similar. In other words, **submitting code that is written by someone else is considered cheating**. If you are unsure whether something is allowed or not, ask the teaching assistants.

Late submission policy: All late submissions will get a deduction of 2 points. In addition, there is a 2-point deduction for every starting 12-hour period. That is, a project submitted at 00.01 on September 24th will get a 4-point deduction and a project submitted at 12.01 on the same day will get a 6-point deduction (and so on). All projects submitted on September 26th or later are automatically failed. (Executive summary: Submit your project on time. The late penalties are designed to be harsh enough so that nobody should benefit by returning their work late. Also note that late penalties can easily drag a good project under the acceptance threshold.) There will be no possibility to resubmit failed projects so start working early.

Based on experiences from the previous years, this is a time-consuming

project. **Start working early!**

1 Tasks

Write a computer program that solves the following tasks. Then write a short report that explain your approach and design choices (Tasks 1.1-1.3) and the results of your experiments as well as the experimental procedure you used (Tasks 1.4-1.5).

1.1 Implement a decision tree learning algorithm from scratch

Implement a greedy algorithm for learning decision trees:

- If all data points have the same label
  - return a leaf with that label
- Else if all data points have identical feature values
  - return a leaf with the most common label
- Else
  - choose a feature that maximizes the information gain
  - split the data based on the value of the feature and add a branch for each subset of data
  - for each branch
    - call the algorithm recursively for the data points belonging to the particular branch

You should use entropy as the impurity measure. Your implementation should have two functions that the users can use:

1. `learn(X, y, impurity_measure='entropy')`
2. `predict(x, tree)`

The function `learn` learns a decision tree classifier from a data matrix $\mathbf{X}$ and a label vector $\mathbf{y}$. We consider the classification task so you can assume that $\mathbf{y}$ consists of categorical variables. You can assume that $\mathbf{X}$ consists of continuous features.

The function `predict` predicts the class label of some new data point $\mathbf{x}$.

Note: If you implement your tree in object-oriented fashion, it is not necessary to include the argument `tree`.

Note: For debugging and visualisation purposes, it may be useful to implement a function that prints the tree.

1.2 Add Gini index

Implement Gini index as an alternative impurity measure. To use the Gini index, you should call your learning function like `learn(X, y, impurity_measure='gini')`.

1.3 Add reduced-error pruning

Decision tree learning is prone to overfitting. To fix this, extend your algorithm to tackle overfitting using reduced-error pruning:

- Divide data to training and pruning data

1. Use the training data to build a full decision tree T^* (using the algorithm from Section 1.1)
2. For each subtree T of T^*
    - If replacing the subtree with the majority label in T (based on training data) does not decrease accuracy on the pruning data
        - Replace T with a leaf node that predicts the majority class

You will need to extend your function's argument list with an additional parameter called `prune` which should by default be set to `False`.

Since pruning should be done inside the `learn` method, the pruning set is a subset of the training set. You can add an additional parameter that specifies which part of the data should be used as a pruning set.

Note that reduced-error pruning starts from the leaves and proceeds bottom-up.

1.4 Evaluate your algorithm

Load the MAGIC Gamma Telescope dataset from MittUiB; alternatively, you can use `https://archive.ics.uci.edu/ml/datasets/MAGIC+Gamma+Telescope`. The task is to predict whether an image of a particle shower represents a hadron or background. The first 10 columns of the data are continuous features extracted from images of the particle showers. The 11th column is the class label (g: gamma (signal), h: hadron (background)).

Assess the performance of your algorithm using an appropriate performance measure. Which setting should you select for this data (entropy or gini, pruning or no pruning)? What is your estimate for the performance of the selected model on unseen data points? Report how you arrived at the conclusions.

Remember to use training, validation, and test sets properly. Note that in the model selection step you select one out of the four models (settings) based on performance on validation data and in the model evaluation step you evaluate the selected model on test data.

1.5 Compare to an existing implementation

Compare your implementation to some existing decision tree implementation. How does your implementation fare against this implementation in terms of accuracy and speed? Can you explain the (possible) differences?

Note: You can compare to, for example, `DecisionTreeClassifier` from `sklearn`.