Implementing decision trees

INF283 - Project - 1

(**Project 1 deadline: 21st of September, 23.59**).
Deliver here: https://mitt.uib.no/courses/12791/assignments

Projects are a compulsory part of the course. You will need to get at least 50% of score to pass this project. This project contributes a total of 17 points to the final grade. You need to upload your answer to MittUIB.no/assignments before 23.59 on the 21st of September. It will then be graded, and the points will be added to your total grade score.

In this project, we implement a decision tree learning algorithm.

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:

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!: This project is about learning, and creating something unique is part of that experience. If we see that you have copy-pasted your answers directly from online tutorials, you have not had the experience of creating, so you will get 0 points. And please don't copy and change a few of the variable names, that is not learning. The only libraries you can import are basic packages like numpy etc. And if you choose to represent you decision tree as a tree class(course book uses a dictionary implementation), you can import a simple tree structure package. No other packages are allowed to be imported. Be noted that pruning is harder on dictionaries compared to the tree implementation (tree implementation can have a isLeaf() function etc.

1. Implement the ID3 algorithm from scratch

Use entropy as the impurity measure. Your implementation should include two functions that the users could use:

1. learn(X, y, impurity_measure='entropy')

This function learns a decision tree classifier from X and y. The function should use a default argument named impurity_measure with a default value 'entropy'. So by default, the learn function would use entropy as an impurity measure for information gain. In other words, if you -- or the user -- call the learn function like this:
learn(X, y, impurity_measure='entropy')
or like this:
learn(X, y)
then in both these cases, your function should learn a decision tree with entropy as impurity measure.

In case you are not familiar what default parameters/arguments are, or how they work, here are some tutorials on it:

1. Default function arguments in python
2. Default function arguments in R

2. predict(x, tree)
This function predicts class label of some new data point x.

All your functions/classes/methods etc. should be documented properly. To learn more on how you should document your code, see the following tutorials below:
Python docstrings (if you are using Python)
R Object documentation (if you are using R)

You are free to implement this decision tree classifier in any programming language you are comfortable with. We, however, will provide solution and support in Python.
NOTE: When you print the tree, it might be useful to have the names of the features (odor etc), input these names into predict and learn as an optional argument to get better visualization of which features it will split, e.g. (learn(X, y, feature_names), predict(x, tree, features_names).

2. Gini Index

So far, your ID3 implementation uses entropy to decide which features to use for splitting the data at each node of the tree. Now we want you to extend your algorithm so that Gini index could be used an alternative impurity measure -- if the user wants so. In other words, if you call your learn() function like this:
learn(X, y, impurity_measure='gini')

then your algorithm should use Gini index as impurity measure instead of entropy for the information gain.

3. Pruning

ID3 is prone to overfit; to fix this, we use pruning.

Extend your algorithm to tackle overfitting using reduced error pruning as shown in weekly exercise 3.

You will need to extend your function argument list with an additional parameter called ‘prune’ which should by default be set to False. In other words, no pruning if the user hasn’t specified anything about pruning. To make this point clear, see the following dummy function calls and their expected behaviour:

If you call the learn function like this:
learn(X, y, impurity_measure=’gini’, pruning=True)
then the function should use gini as impurity measure, and should do pruning of the tree.

If you call the learn function like this:
learn(X, y, impurity_measure=’gini’, pruning=False)
then the function should use gini as impurity measure, and should not do pruning of the tree.

If you call the learn function like this:
learn(X, y)
then the function should use entropy as impurity measure, and should not do pruning of the tree.

If you call the learn function like this:
learn(X, y, pruning=True)
then the function should use entropy as impurity measure, and should do pruning of the tree.

And so on…

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.

Hint:
Pruning starts at the bottom of the tree (leaves), and move upwards to root:
If sum of errors on the pruning set for the children are bigger than the parent node, remove the children nodes from the tree.

4. Classify edible and poisonous mushrooms

Two machine learning experts are on a camping trip, and have collected a bunch of mushrooms from the forest. They want to eat mushrooms, but don't know which ones are dangerous. Since they don't have a computer with them, they ask you to classify their mushrooms for them.
They sent you a link of their data: https://archive.ics.uci.edu/ml/datasets/mushroom
along with information about the data set:
https://archive.ics.uci.edu/ml/machine-learning-databases/mushroom/agaricus-lepiota.names

Classify their data into edible (e) and poisonous (p), using your implementation of ID3. Divide the data first into two sets: training data and test data. (Justify what proportion of the data you use for each set, do you use random sampling for splitting ?) Remember that the pruning data is a excluded subset of the training data.

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 on unseen data points ? (see exercise 3 for hint) Sometimes pruning will reduce the size of the tree, sometimes not. Why ?

NOTE:
This datasets contains ill defined rows, that is rows with ? in them. The ? means the value is not defined (The two guys could not find this features for the specific mushroom). You need to make your "learn" function be able to read the data, so clean up the dataset by removing the rows with ?.

5. Implementation comparison:

Compare your implementation to some existing decision tree implementation. How does your implementation fare against this implementation? Please see below for suggestions for Python and R.

Python:
from sklearn import tree
clf = tree.DecisionTreeClassifier()
R:
caret::train(method = "rpart")
Other language: Include information if you use another language.