TDT4171 - Assignment 6
Learn Decision Tree

Arthur Testard - id: 105022

Exercise 1

Both implementation of IMPORTANCE function versions as LEARN-DECISION-TREE can be found on `assignment_6.py`. Both are implemented as they are defined in pseudo code in Chapter 19 of [RN10].

Implementation of `random` version of IMPORTANCE function is simple as picking up a random value in the list of available attributes in the current node. Implementation of `information_gain` version consists in calculating the following value:

$$a^* = argmax_a B(\frac{p}{p+n}) - \sum_{v \in V(a)} \frac{p_v + n_v}{p+v} B(\frac{p_v}{p_v + n_v})$$

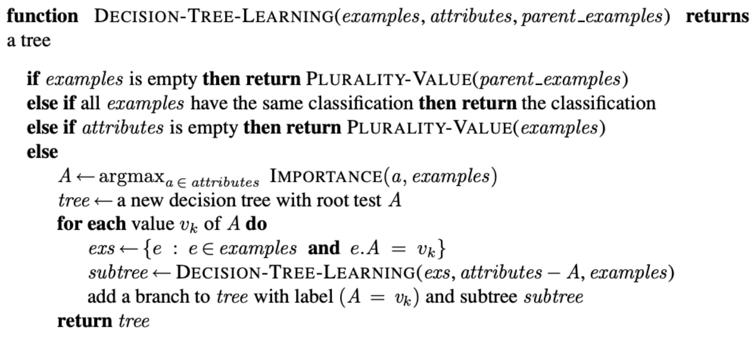
Where:
- $B(q)$: entropy of a Boolean random variable that is true with probability q.

$$B(q) = qlog_2(q) + (1-q)log_2(1-q)$$

- p: number of positive examples.
- n: number of negative examples.
- p_v: number of positive examples for a given value in the set of values for attribute a.
- n_v: number of negative examples for a given value in the set of values for attribute a.
- $V(a)$: set of values for attribute a.

Notice that in our data from `train.csv` and `test.csv`, we have for any attribute, $V(a) = \{1, 2\}$. Notice that in output we only have two classes (True or False), but we could add more and then the formula of a^* would be different (starting by the definition of B).

LEARN-DECISION-TREE is simply the implementation of the pseudo-code algorithm given by Figure 1 from the book.



```
function DECISION-TREE-LEARNING(examples, attributes, parent_examples) returns
a tree
  if examples is empty then return PLURALITY-VALUE(parent_examples)
  else if all examples have the same classification then return the classification
  else if attributes is empty then return PLURALITY-VALUE(examples)
  else
      $A \leftarrow \text{argmax}_{a \in attributes}$ IMPORTANCE(a, examples)
      tree ← a new decision tree with root test A
      for each value $v_k$ of A do
          exs ← $\{e : e \in examples$ and $e.A = v_k\}$
          subtree ← DECISION-TREE-LEARNING(exs, attributes − A, examples)
          add a branch to tree with label $(A = v_k)$ and subtree subtree
      return tree
```

Figure 1: The decision-tree learning algorithm from [RN10]. The function IMPORTANCE is described in above. The function PLURALITY-VALUE selects the most common output value among a set of examples, breaking ties randomly (its implementation is already done in the given file).

Firstly, I wanted to verify my implementation by using a more simple dataset. I thus took the examples for the restaurant domain from the book, given in Figure 2. It consists in the following table that I had to translate to values from 1 to the number of values associated to an attribute. For example, $V(Patrons) = \{1, 2, 3\}$. My implementation gave me the tree given on Figure 3.

An issue then was to take into account the lack of data in some cases (Patrons $\rightarrow$ Wait/Estimate $\rightarrow$ 0 - 10 $\rightarrow$ *No data*). That is the reason why, for a given attribute a, $V(a)$ is implemented as it is on line 129. This is a point which seems covered by the book but the point is not here for that implementation.

Example	Input Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*\$\$\$*	*No*	*Yes*	*French*	*0–10*	$y_1 =$ *Yes*
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*\$*	*No*	*No*	*Thai*	*30–60*	$y_2 =$ *No*
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*\$*	*No*	*No*	*Burger*	*0–10*	$y_3 =$ *Yes*
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*\$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 =$ *Yes*
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*\$\$\$*	*No*	*Yes*	*French*	*>60*	$y_5 =$ *No*
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*\$\$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 =$ *Yes*
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*\$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 =$ *No*
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*\$\$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 =$ *Yes*
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*\$*	*Yes*	*No*	*Burger*	*>60*	$y_9 =$ *No*
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*\$\$\$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} =$ *No*
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*\$*	*No*	*No*	*Thai*	*0–10*	$y_{11} =$ *No*
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*\$*	*No*	*No*	*Burger*	*30–60*	$y_{12} =$ *Yes*

Figure 2: Examples for the restaurant domain.

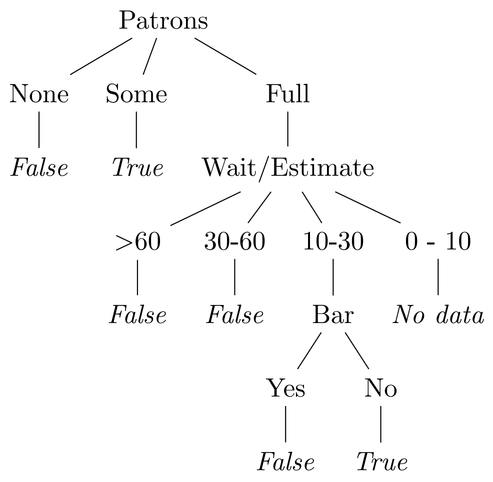


Figure 3: Tree given by the implementation of LEARN-DECISION-TREE and information gain version of IMPORTANCE

Thus, I started to study the results using the data from `train.csv` and `test.csv` using both versions of IMPORTANCE function. Not surprisly, I get an accuracy of 1.0 on the train set for both versions. However, the results on `test.csv` vary depending on the version of IMPORTANCE function. Some values of the results can be found on Figure 4.

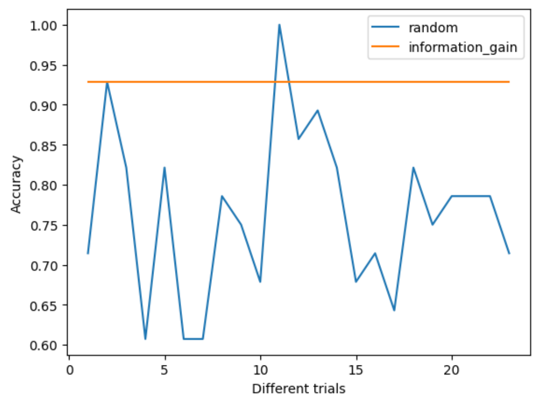
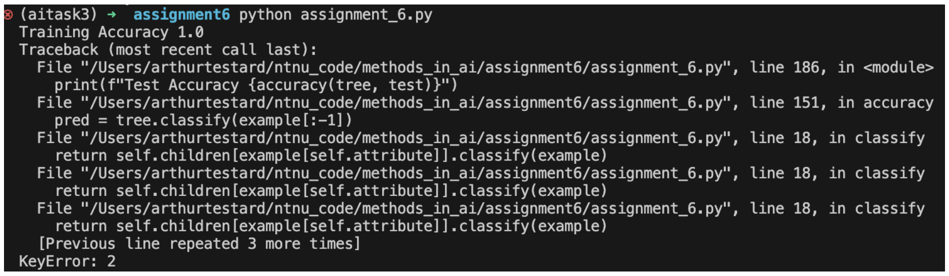


Figure 4: Accuracy of the model using random or information gain version of IMPORTANCE for different trials

As we can see, the accuracy on `information_gain` version does not vary depending on the trial. We have this result because the training set does not change on a different trial and that the algorithm is deterministic. In the mean time, `random` version is not deterministic, so we found different values for the accuracy at the end. For 23 trials, I get a mean accuracy, for `random` version of approximately 0.76, which is lower than 0.93 the approximated value of the accuracy using `information_gain` version. However, we can notice that the test accuracy using `random` version has been once equal to 1, surprisingly. But, as far as I can see, that result comes from the size of the dataset which is pretty small so, luckily sometime we get a great accuracy like this. However this value is just due to luck and should not be considered as an argument if we have to decide between both versions. Probably, if we use a different test set, we would have a lower value of the test accuracy.

As a conclusion, I would say that with that example, `information_gain` version seems to be a better version of IMPORTANCE function rather than `random` one. The random aspect does not seem very desirable in that case. Also the test set is maybe to small to conclude if `random` version can be efficient here (according to the point on the test accuracy equal to 1). Most of the time, its accuracy is lower than the deterministic version, `information_gain`.

One last point consider a more technical aspect of the implementation. I sometimes get the error showed by Figure 5 using `random` version. However, I considered that it was not an important point to explore here, because it were working most of the time.



```
⊗ (aitask3) ➜  assignment6 python assignment_6.py
 Training Accuracy 1.0
 Traceback (most recent call last):
   File "/Users/arthurtestard/ntnu_code/methods_in_ai/assignment6/assignment_6.py", line 186, in <module>
     print(f"Test Accuracy {accuracy(tree, test)}")
   File "/Users/arthurtestard/ntnu_code/methods_in_ai/assignment6/assignment_6.py", line 151, in accuracy
     pred = tree.classify(example[:-1])
   File "/Users/arthurtestard/ntnu_code/methods_in_ai/assignment6/assignment_6.py", line 18, in classify
     return self.children[example[self.attribute]].classify(example)
   File "/Users/arthurtestard/ntnu_code/methods_in_ai/assignment6/assignment_6.py", line 18, in classify
     return self.children[example[self.attribute]].classify(example)
   File "/Users/arthurtestard/ntnu_code/methods_in_ai/assignment6/assignment_6.py", line 18, in classify
     return self.children[example[self.attribute]].classify(example)
   [Previous line repeated 3 more times]
 KeyError: 2
```

Figure 5: The error get sometimes using error version of IMPORTANCE

References

[RN10] Stuart J Russell and Peter Norvig. *Artificial intelligence a modern approach.* London, 2010.