Decision Tree

Decision Tree is a decision rule based tree structure constructed from the data samples. It utilizes different tree based algorithms, such as, ID3 (Iterative Dichotomiser 3), C4.5, C5.0, and CART (Classification and Regression Trees). In our study, we use a decision tree learning algorithm provided in Russell and Norvig's book "Artificial Intelligence, A Modern Approach" that constructs non binary trees using the feature and threshold with the support of the largest information gain at each node.

Intrusion Detection

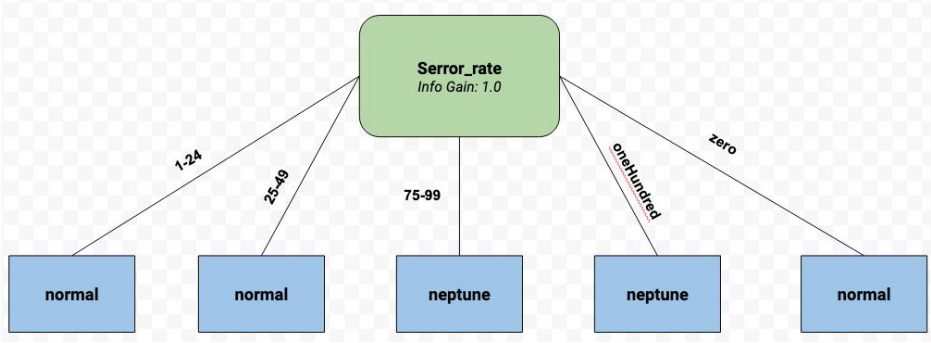
Intrusion detection is an area of computer security that focuses on detecting attacks reliably. Intrusion detection systems (IDS) usually have a knowledge base containing rules that characterize attacks. To differentiate between instances of normal and attack behavior of the network traffic, we work with the IDS dataset that contains records of network activities that are normal or part of a denial of service (DOS) attack(s) called Neptune (aka SYN-flood).
In this project, we build the tree using Python 3, evaluate the constructed decision tree with the KDD datasets as well as a toy example dataset, and achieve satisfactory performance.

Experimental Methodology

The implementation takes account of three files passed as arguments, such as, train, test, and attribute files. The learner builds the decision tree with the given train set. With the best choice of information gain, it chooses the attribute to perform the split on the dataset and records the first feature as the root of the tree. Until no more examples are left, all examples have the same class, and no more attributes to split, the algorithm recursively iterates through the train set to learn the decision rules with the help of entropy and information gain. To attempt a better implementation, the tree holds nodes that have got one *dictionary* variable to store its details, e.g., name, condition to reach the node, and information gain as well as three pointers to connect it with other nodes in a tree structure. As the tree can very likely be non-binary trees, the pointers surves the purpose of connecting the parent, leftmost child, and siblings of the nodes.
After the tree is built, we print out the constructed decision tree. We traverse through it in order to predict the class of each record in both train and test set. Comparing the predicted results with the given results, we compute the train set accuracy and test set accuracy.

Results

With the train set given to us, the following tree is achieved --



From the tree above, we can come to the conclusion that visiting one attribute is suffice to generate class labels, e.g., normal and neptune (attack). To further describe, the learning algorithm finds that the conditions (such as, 1-24, 25-49, 75-99, oneHundred, and zero) found in the attribute, *serror_rate*, has been deterministic enough that no other feature needs to be visited to perform the split on. Hence, we achieve one layer decision tree.

On the other hand, the accuracy has been 100% when tested with both train and test set. Thus, we do find any false positives or false negatives examples in any case.

Discussion

The reason behind achieving the steller results is that the constructed decision tree from the train set has been perfectly formulated to find the class up on inspecting only one attribute. It is a rare case; however, the implemented algorithm is tested with a toy dataset (a very small version of the example given in the book of Russell & Norvig: Will the customer wait for a table?) to investigate its correctness of the approach. The example sets, used to evaluate the algorithm, are provided in the developed software to run.

Conclusion

We use the decision tree algorithm provided in Russell and Norvig's book "Artificial Intelligence, A Modern Approach." We build the solution using Python 3 with the help of Pandas library to access the datasets along with some other commonly used Python libraries, evaluate the constructed decision tree with the KDD datasets as well as a toy example dataset, and achieve stellar performance.

Appendix

1. Output of the program

```
$ python3 decision_tree.py ids-train.txt ids-test.txt ids-attr.txt
> Train, test, and attribute files are read properly
Training Set Shape: (800, 12)
Testing Set Shape: (200, 12)

============ Decision Tree ============
serror_rate (Info Gain: 1.0 and Root)
normal (Info Gain: None and Condition from its parent, serror_rate, is 1-24)
normal (Info Gain: None and Condition from its parent, serror_rate, is 25-49)
neptune (Info Gain: None and Condition from its parent, serror_rate, is 75-99)
neptune (Info Gain: None and Condition from its parent, serror_rate, is oneHundred)
normal (Info Gain: None and Condition from its parent, serror_rate, is zero)

============ Performance ============
Train Accuracy: 1.0000
Test Accuracy: 1.0000
```

2. Libraries used in the program

```
import sys
import math
import pandas as pd
import numpy as np
from typing import Dict
```

3. Decision Tree Learning algorithm

function DECISION-TREE-LEARNING(*examples*, *attributes*, *default*)
returns a decision tree
  inputs: *examples*, set of examples
    attributes, set of attributes
    default, default value for the goal predicate

  if *examples* is empty **then return** *default*
  else if all *examples* have the same classification
    then return the classification
  else if *attributes* is empty
    then return MAJORITY-VALUE(*examples*)
  else
    best ← CHOOSE-ATTRIBUTE(*attributes*, *examples*)
    tree ← a new decision tree with root test *best*
    for each value v_i of *best* **do**
      $examples_i$ ← {elements of *examples* with *best* = v_i}
      subtree ← DECISION-TREE-LEARNING($examples_i$,
        attributes − *best*, MAJORITY-VALUE(*examples*))
      add a branch to *tree* with label v_i and subtree *subtree*
    end
    return *tree*