**CSE 4633 Artificial Intelligence (AI)**

**Final Project**

**By Ajinkya Nawarkar**

**an839**

**Objective:**

In the last lecture of our class, we learnt about Decision Trees. However, with the restricted time we were only able to go through its basic concept and how they work but I was much more intrigued and so I decided to write a Decision Tree Classifier from scratch as my final project for this class. In the given time frame, I was able to implement and run some experiments on a few datasets to see if it actually worked and it successfully did. I’ve listed my learnings at the end.

**Approach:**

Decision Trees or also known as Classification And Regression Tree algorithm (CART) are popular as a powerful prediction method and mostly used for interpretability. The way they work is very simple to comprehend and have 4 steps:

1. Find features with max. info gain using entropy or Gini index
2. Create best split-split at each level
3. Build the tree
4. Traverse through the tree to predict for a given example

First, using the sample training data, a decision tree is constructed by splitting the dataset into two at optimal point such that the resulting child datasets form with the highest degree of purity as leaf nodes or class predictions. The dataset is recursively split into more than two levels depending on the number of features and unique class values. The optimal point or node at which the dataset should be split is determined using a cost function which calculates the impurity in the dataset. A cost function helps to measure the amount of information gain on a split given a certain feature.

The impurity or the cost function could be computed in a couple of different ways. One way is to use entropy that is too say by what measure is a set of data mixed or unmixed. A set with similar items can be called fully pure whereas one with a mix of different items could be called impure.

For the sake of learning, I used a second way of measuring impurity called Gini index. A Gini score helps to give us an idea of how mixed the classes are in two groups. A score of 0 means perfect separation whereas a split of 50/50 means a score of 0.5. Ideally, you would want to split the tree into two branched for a features which has a Gini index of 0 and maximum information gain.

To predict an example sample, you then just traverse through this built decision tree and determine the prediction with the Leaf Node you end up to starting from the root node.

All of the above steps were implemented with different function in dt\_classifier.py, whereas the prediction class and query class were implemented in separate classes. I used a measure of query which helped in asking question at each level as to what a features points to – a numeric or a text value.

For testing and experimentation purposes, I used standard functions from <https://machinelearningmastery.com> . These included cross validation, train/test split and data sanitization functions.

**Datasets:**

I used four different datasets:

1. Seismic Bumps:
2. Banknote Authentication:

This multivariate data set is used identify between genuine and forged banknote specimens. It contains 4 attributes and 2 classes with 1372 instances.

1. Balance Scale:

This data set is used to model psychological experimental results. Each example is classified as having the balance scale tip to the right, tip to the left, or be balanced. It contains 4 attributes and 3 classes with 625 instances.

1. Qualitative Bankruptcy:

This data set is used to predict bankruptcy from qualitative parameters from experts. Each example feature is classified as Positive, Average, and Negative. It contains 6 attributes and 2 classes with 625 instances.

**Observations:**

I used KFold - Cross Validation for 3 to 5 folds giving each fold per dataset. I did not see any way any kind of a graph would be useful except the class separation for datasets but again not so useful as I only intended a basic operational Decision tree.

The following table shows percent accuracy with the test fold for Seismic dataset.

|  |  |  |  |
| --- | --- | --- | --- |
| Seismic Dataset | K-Folds | Train/Test Split | Accuracy (Percent) |
|  | 3 | 1722/861 | 88.27 |
|  | 5 | 2064/516 | 88.566 |
|  | 7 | 2214/369 | 87.72 |

The following table shows percent accuracy with the test fold for Bank Note dataset.

|  |  |  |  |
| --- | --- | --- | --- |
| Banknote Dataset | K-Folds | Train/Test Split (Percent) | Accuracy (Percent) |
|  | 3 | 914/457 | 97.73 |
|  | 5 | 1096/274 | 98.321 |

The following table shows percent accuracy with the test fold for Balance dataset.

|  |  |  |  |
| --- | --- | --- | --- |
| Balance Dataset | K-Folds | Train/Test Split (Percent) | Accuracy (Percent) |
|  | 3 | 416/208 | 79.04 |
|  | 5 | 500/125 | 77.6 |

The following table shows percent accuracy with the test fold for Bankruptcy dataset.

|  |  |  |  |
| --- | --- | --- | --- |
| Bankruptcy Dataset | K-Folds | Train/Test Split (Percent) | Accuracy (Percent) |
|  | 3 | 200/50 | 100 |
|  | 5 | 166/83 | 100 |

**Complexities and Learnings:**

The first complexity I came across was that my initial version of the implementation only took care of categorical datasets. So it would have only worked for the dataset from the example in class where all input attributes were text and required only a straight comparison.

I had to later tweak my implementation to handle a variety of datasets as you can see from my experiments. This involved almost rewriting and redesigning my implementation with additional classes.

I also noticed that decision trees act like greedy algorithms thus the splits are expected to lead to a global maximum. As you can see from the Bankruptcy dataset, it seems to be a perfect example of overfitting and from my research, one of the ways to avoid this would have been restricting the depth of the tree or by minimizing the no. of samples.

Gini index is widely used for continuous features whereas entropy is more used for categorical features. For Gini index, a lower scores results to be a perfect split and nearly same composition for higher scores.

The speed of the DT prediction also depends on the depth of the tree built, instances where even a nearly equal split but with different resulting sets of features to split further might be a difference in seconds. For example, if you were to use the –dt argument while running on seismic-dataset, you could see more than 30 levels of splits on the decision tree that is printed where as the decision tree printed for the balance tree is much smaller in depth and thus faster.

**Conclusion:**

With given time frame, I was able to accomplish what I had decided which was to implement a Decision tree from scratch and see if it works and understand it better. This was a great learning experience and fun to implement in python. I’ve learnt how incremental and exhaustive the split search could be based on number of attributes and unique class values. By implementing from scratch, I’ve learnt eh concept of purity even better and also realized how decision trees are prone to overfitting and the only way to generalize them is by creating hyperparameters like max depth and min samples size.

If I had enough time, the things that I would’ve tried implementing or researching into were:

* Tree pruning
* Rani forests
* Cross entropy, another impurity index
* Comparison of accuracy and reliability between decision tree algorithms from sklearn and my implementation

**Deliverables:**

I’ve attached all of my project files with datasets I used. I also used arg-parser so you could use different command line arguments to test and see the parsed tree and results. I’ve also attached a README.md file to explain the different command line arguments but you could use the -h option to see all usable options.