# Programming Assignment 1 - Implement a fixed depth decision tree.

Team Details:

1) Rahul Pasunuri ([rahupasu@indiana.edu](mailto:rahupasu@indiana.edu))

2) Huzefa Dargahwala([hydargah@indiana.edu](mailto:hydargah@indiana.edu))

Introduction:

This project implements a fixed depth decision tree algorithm using Python (2.7). The decision tree is built in a top down approach, and decision to split at every node is done in a greedy fashion. The algorithm uses "Information gain" as the deciding factor for splitting a node. This algorithm does binary splits using the one vs rest approach for every possible feature, i.e. every (feature, value) pair is considered as a feasible criterion for splitting at that node. And the split which gives us the maximum Information gain will be employed. The learnt tree is then applied on two datasets and the results are compared with the J48 implementation of Weka.

Few Definitions:

* Entropy: This is used to measure the impurity or uncertainty of a given distribution [2].

* Information Gain: This is used to measure the expected reduction in uncertainty of the distribution with the partitioning of the samples based on an attribute [2].

Here, S is the collection of samples, and A is the attribute which is being in consideration for the split.

Criterion for stopping the tree from splitting:

We stop building the tree, when at least one of the following criterion is met:

1. There is no information gain in splitting any node further.
2. The tree grew to the maximum allowed depth size, which is given in the command line arguments.
3. Splits with all possible criterion are done, i.e., there is no new (feature, value) pair, with which a new split can happen.

Inductive bias of the learnt tree:

Shorter trees are preferred over larger trees, and the splits with highest information gain are kept close to the node.

Running Application:

The syntax of the command which has to be executed can be seen below:

python decTree.py <training-file-name> <test-file-name> [<threshold> [depth]]

Arguments:

1) training-file-name: This is the name of the file which contains the training dataset.

2) test-file-name: This is the name of the file which contains the test dataset.

3) threshold: This is the threshold parameter, which is of float type. The split at a node is done, only if the maximum achievable "Information gain" at that node is greater than this threshold parameter. This is an optional parameter, and if it is not set then it will be defaulted to the value '0'.

4) depth: This is the maximum depth of the decision tree, which has to be learnt. This is also an optional parameter, and the default value of it is "10".

Note that, in-order to override the depth parameter, even the threshold parameter has to be overriden.

Examples:

1) python decTree.py "zoo-train.csv" "zoo-test.csv" 0.001 5

Here, "zoo-train.csv" is the name of the training dataset, and "zoo-test.csv" is the name of the test dataset. Here, '0.001' is used as the threshold parameter, and '5' is used as the maximum allowed depth of the decision tree.

2) python decTree.py "foodInspectionTrain.csv" "foodInspectionTest.csv"

Here, as the values of the threshold and depth are not given, they will be defaulted to 0, 10 respectively.

3) python decTree.py "foodInspectionTrain.csv" "foodInspectionTest.csv" 0.001

Here, the value of the threshold will be overriden to '0.001', but the maximum depth of the tree will be given the default value.

Project's File structure:

1) createDataSet.py

Results:

Misclassification rate vs depth:

The below plot shows the variation of the misclassification rate with the maximum allowed depth of the tree.



Misclassification

rate

Maximum allowed depth of the tree.

As, you can see, the misclassification rate stops decreasing after depth 5. This is because the tree stops growing after depth 5, since it doesn't get any Information gain from further splitting. The minimum misclassification rate achieved by this tree is 0.1142,

Confusion Matrices:

Trees learnt with our algorithm with depths 1 & 2 using the “zoo-train.csv” dataset was employed to test the dataset “zoo-test.csv”. The confusion matrices of the test results can be seen below.

The first row in the below matrices represent the predicted labels, and the 1st column represents the actual labels of the data set.

Confusion Matrix with depth - 1:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **1** | 14 | 0 | 0 | 0 | 0 | 0 | 0 |
| **2** | 7 | 0 | 0 | 0 | 0 | 0 | 0 |
| **3** | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| **4** | 4 | 0 | 0 | 0 | 0 | 0 | 0 |
| **5** | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| **6** | 3 | 0 | 0 | 0 | 0 | 0 | 0 |
| **7** | 4 | 0 | 0 | 0 | 0 | 0 | 0 |

Structure of the Tree :

The tree has a single node which is a “decision stump”. Here, the tree is the decision stump. Here, the node observes that class '1' occurs maximum number of times. And at the leaf node, we find the label which occurs the maximum number of times in its samples at that node, and uses that as the prediciton label. So, it labels everything as '1'. Hence, in the confusion matrix, only the first column has non-zero values, and the other class has 0 values, as the our tree doesn't predict those class labels.

Confusion Matrix with depth - 2:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **1** | 14 | 0 | 0 | 0 | 0 | 0 | 0 |
| **2** | 0 | 7 | 0 | 0 | 0 | 0 | 0 |
| **3** | 0 | 2 | 0 | 0 | 0 | 0 | 0 |
| **4** | 0 | 4 | 0 | 0 | 0 | 0 | 0 |
| **5** | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| **6** | 0 | 3 | 0 | 0 | 0 | 0 | 0 |
| **7** | 0 | 4 | 0 | 0 | 0 | 0 | 0 |

Structure of the Tree :

Feature 4 == 0 ?

Ye

s

NO

(Here, rhombus shapes are used to represent decision nodes, and oval shapes are used to represent leaf nodes. )

The root node has criterion based on feature 4. Based on the result of the criterion, we must proceed either to the left sub tree (if the answer evalues to NO), or the right sub tree(if the answer evaluates to Yes) .

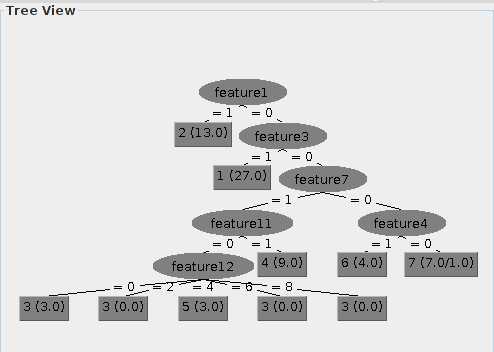
Comparison with Weka:

We used J48 decision tree algorithm in weka on the zoo-dataset. The confusion matrix of the test results can be seen below.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **1** | 14 | 0 | 0 | 0 | 0 | 0 | 0 |
| **2** | 0 | 7 | 0 | 0 | 0 | 0 | 0 |
| **3** | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| **4** | 0 | 0 | 0 | 4 | 0 | 0 | 0 |
| **5** | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| **6** | 0 | 0 | 0 | 0 | 0 | 2 | 1 |
| **7** | 0 | 0 | 0 | 0 | 0 | 0 | 4 |

The misclassification rate on this test set is '0.085'. The J48 tree wrongly classified 3 out of 35 samples, and the tree modeled by us wrongly classified 4 out of the 35 samples.

The decision tree learnt by weka can be seen below.



The depth of the tree learnt by weka is 5, and has 10 leaf nodes. (Here, feature1 is the first column in the dataset, feature2 is 2nd column, and so on.)

Food-Inspection dataset:

Similarly, trees learnt with our algorithm with depths 1 & 2 using the “zoo-train.csv” dataset were employed to test the dataset “zoo-test.csv”. The confusion matrices of the test results can be seen below.

Confusion Matrix with depth - 1:

Confusion Matrix with depth – 2:

Explanation of the above Confusion Matrices:

Limitations of the project:

1. The algorithm only supports binary splits.
2. The algorithm does not work as expected if the test/training dataset has missing values.
3. Our algorithm doesn't do backtracking. So, it may not converge to the global optimum.
4. The algorithm doesn't handle data which have features with continuous values.
5. All the attributes in this algorithm are given equal weights. So, it might not work as expected, when different weights are assigned to the attributes.

Summary:

In this project, we have a modeled a simple decision tree and compared it with the weka's J48 decision tree. Though, our algorithm has some limitations, the algorithm performed quite well compared to the J48 algorithm.

References:

1) Programming Collective Intelligence, Building Smart Web 2.0 Applications, O'REILLY, First Edition, by Toby Segaran.

2) Machine Learning, McGraw Hill Education Private Limited, by Tom M. Mitchell.