Programming Assignment 1 - Implement a fixed depth decision tree.

Team Details:

1) Rahul Pasunuri (rahupasu@indiana.edu)
2) Huzefa Dargahwala (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].

$$H(Y) = \sum p_i \, log_2 p_i$$

- 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].

$$\text{Gain}(S,A) = H(S) - \sum_{v \in \text{values}(A)} \left(\frac{|S_v|}{|S|}\right) H(S_v)$$

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

Algorithm:

1. Create a root node, and choose the best (feature-f, value-v) pair for the root node.
2. If there is no such pair, or any one of the stopping criterion is met, then this node is a leaf node. Skip to point '5'.
3. Separate the training set into sets S1, and S2, where S1 has samples which have the feature f's value as 'v', and S2 = training set – S1.
4. Follow steps 1 &2, to create the right sub tree and the left sub tree, using S1 and S2 as training datasets respectively.
5. When no further splitting is possible, create a leaf Node with all the occurring class labels and the number of times each occurs. (The most probable one is chosen as the class label)
6. The tree is ready for testing.

Choosing the best (feature, value) pair:

We will split at that criterion which gives us the maximum Information gain. For this, we imitate splitting on every possible (feature, value) pairs and check the Information gain, and we choose the (feature, value) pair, which gives us the maximum Information gain (which must be greater than 0).

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. (Occam's Razor)

Running the Application:

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

```
python decTree.py >output.txt
```

Arguments:

1) output.txt : Give the output in a nice txt file rather than a command line. This argument is optional

Project's Contents:

1) Node.py:

This file defines the class 'Node', which is used to represent a node in the tree, which could be either a decision node or a leaf node. Each node then points to its left node, and its right node. Also, each node saves the feature index which was used for the split on that node, and also saves the value of the feature which was used for the binary split. All the samples, which match the value for the feature will be re-directed to the right sub tree, and other samples will be re-directed to the left sub tree (one vs rest approach). When the dataset cannot be split further, Node becomes the leaf node, with all the possible class labels.

2) decTree.py

This file has the logic to create the decision tree using the training dataset, and then classify the test dataset. We create trees of depths starting from 1 to 16 and save them in a text format in the "results" directory.

3) output.txt

This file contains the trees modeled by our algorithm in a text format. It includes the trees trained for both the zoo and food-inspection datasets with depths from 1 to 16.

4) directory – results/

This contains the classified and expected labels for both the zoo and food-inspection datasets. These are computed for all the trees with depths from 1 to 16.

5) directory – "weka datasets"

This directory contains zoo and food-inspection datasets in arff format, and also contains the test results of weka.

6) README

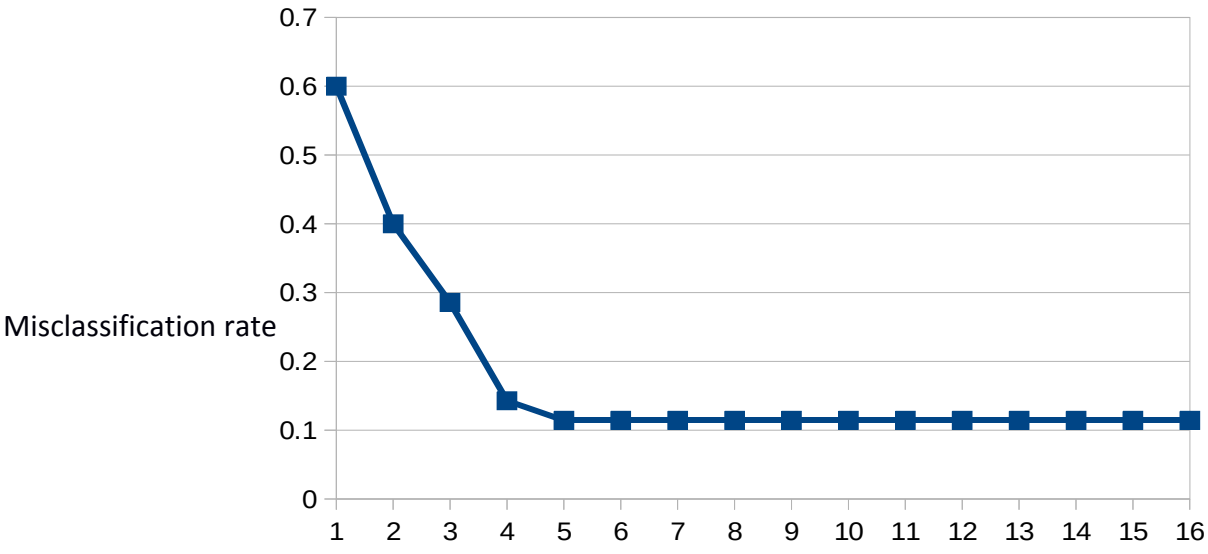
This file contains a brief description on how to run the project and also has some details on the algorithm used.

Results:

Zoo Datasets

Misclassification rate vs depth:

The below plot shows the variation of the misclassification rate with the 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. The structure of the tree is given in the section where we compare our results with Weka.

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

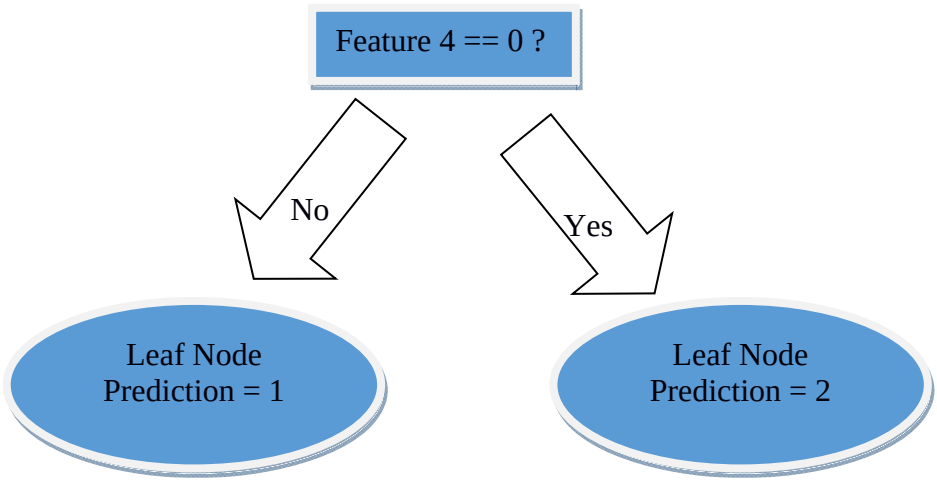
Structure of the tree with Depth = 1:

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 prediction 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 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 for Depth = 2:



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 evaluates to NO), or the right sub tree (if the answer evaluates to Yes).

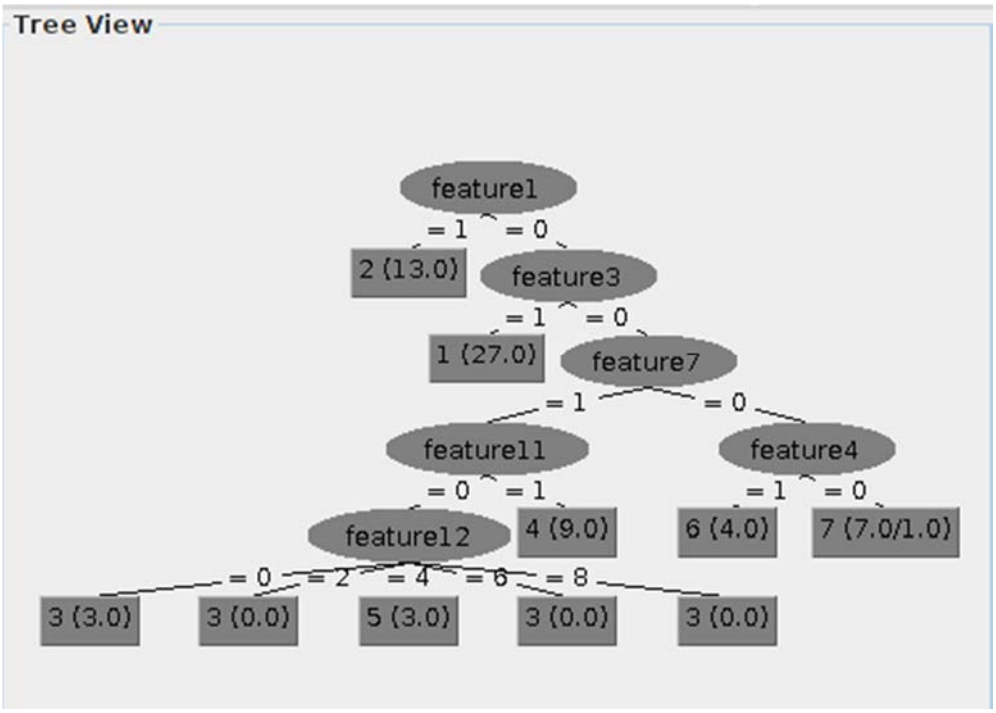
Comparison with Weka:

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.)

The depth of the tree built by our algorithm is also 5. The **best tree learnt by our algorithm** looks like:

```
Split on Column : 3 with criteria : 0
Left Branch ->  Leaf Node : {'1': 27}
Right Branch ->  Split on Column : 7 with criteria : 1
    Left Branch ->  Split on Column : 1 with criteria : 1
        Left Branch ->  Split on Column : 9 with criteria : 1
            Left Branch ->  Leaf Node : {'7': 5}
            Right Branch ->  Split on Column : 2 with criteria : 1
                Left Branch ->  Leaf Node : {'7': 1}
                Right Branch ->  Leaf Node : {'6': 5}
        Right Branch ->  Leaf Node : {'2': 13}
    Right Branch ->  Split on Column : 11 with criteria : 1
        Left Branch ->  Split on Column : 12 with criteria : 0
            Left Branch ->  Leaf Node : {'5': 3}
            Right Branch ->  Leaf Node : {'3': 3}
        Right Branch ->  Leaf Node : {'4': 9}
```

Food-Inspection Dataset

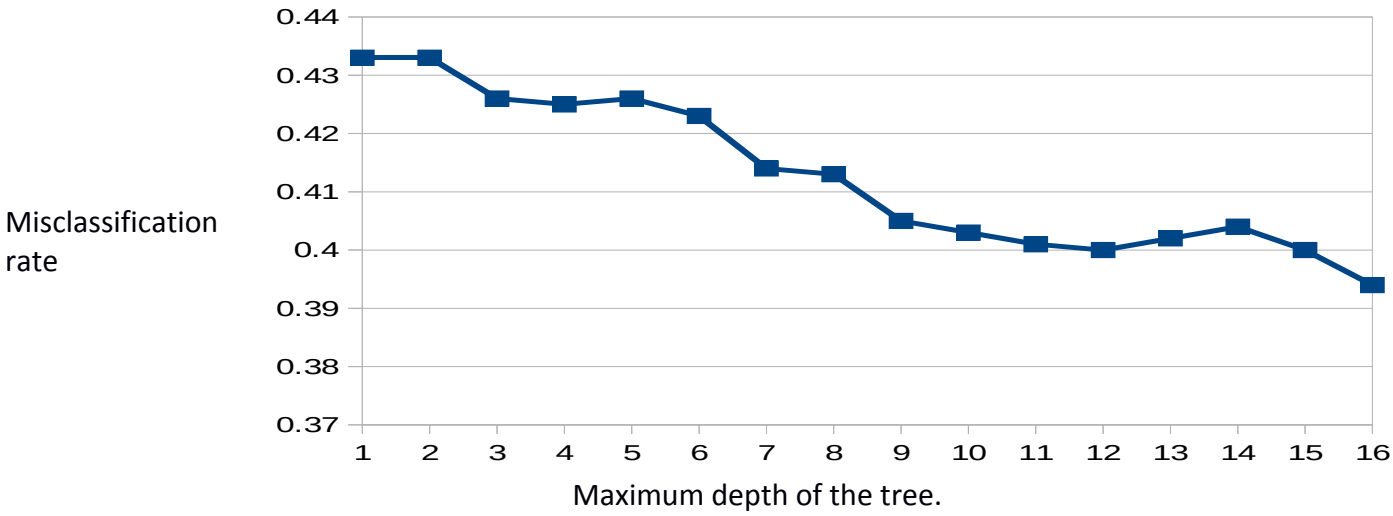
Trees learnt with our algorithm with depths 1 & 2 using the “foodInspectionTrainPruned.csv” dataset were employed to test the dataset “foodInspectionTestPruned.csv”. The confusion matrices of the test results can be seen below.

Pre-processing on the dataset:

When we first tried to run the algorithm on the dataset as-is, the algorithm massively overfitted the data because of the number of continuous values present. We then talked to our professor and he suggested that we discretize the dataset. To make the features discrete, some preprocessing has been done on the zoo dataset, which are:

1) Removed the last two digits of the zip code.
2) Only the 'day' value is considered from the date feature.
3) The decimal points in the longitude and latitude are truncated.
4) The “string” features are replaced with the first character in the string.

Misclassification rate vs Maximum depth of the tree:



Confusion Matrix with Depth = 1:

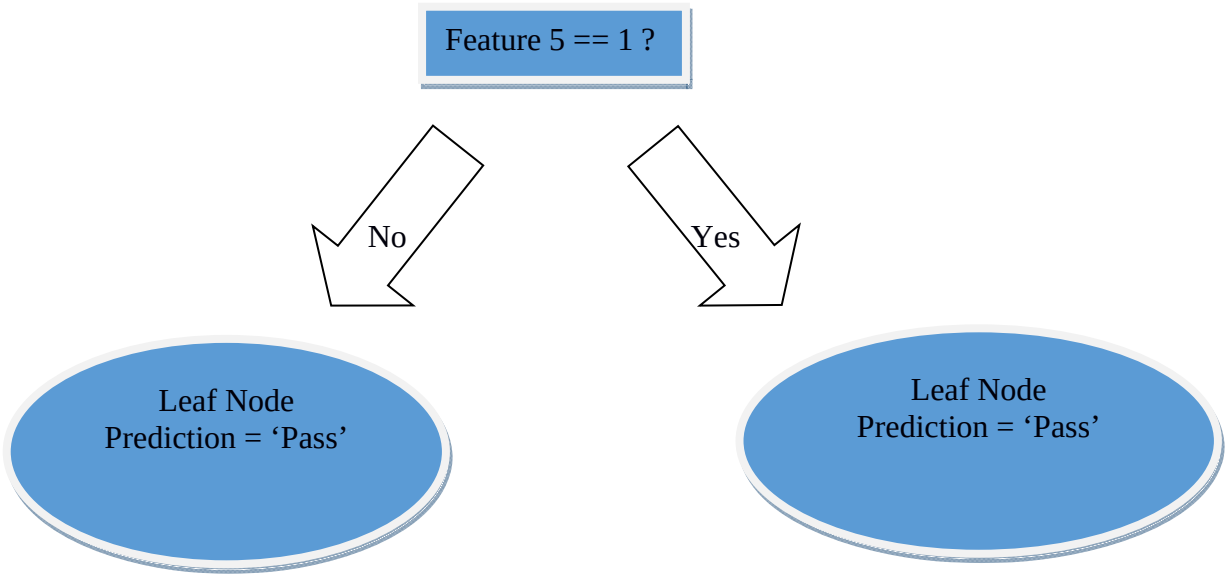
	Pass	Fail	Pass w/ Conditions	No Entry	Out of Business
Pass	567	0	0	0	0
Fail	193	0	0	0	0
Pass w/ Conditions	114	0	0	0	0
No Entry	48	0	0	0	0
Out of Business	78	0	0	0	0

Here, our learnt tree is a simple decision stump, which blindly predicts every sample as “Pass”. This is the reason for other columns having only '0' in them.

Confusion Matrix with Depth = 2:

	Pass	Fail	Pass w/ Conditions	No Entry	Out of Business
Pass	567	0	0	0	0
Fail	193	0	0	0	0
Pass w/ Conditions	114	0	0	0	0
No Entry	48	0	0	0	0
Out of Business	78	0	0	0	0

Structure of the Tree for Depth = 2:



Here, though a split is occurring at the root node, the leaf nodes still predict every sample as “Pass”, as the number of “Pass” samples are more in both the branches. More number of splits are necessary to predict other labels as well.

Comparison with Weka:

The processed datasets were run using Weka's J48. The trained tree was a simple **decision stump**, which predicts everything as “Pass”.

	Pass	Fail	Pass w/ Conditions	No Entry	Out of Business
Pass	567	0	0	0	0
Fail	193	0	0	0	0
Pass w/ Conditions	114	0	0	0	0
No Entry	48	0	0	0	0
Out of Business	78	0	0	0	0

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.