Machine Learning (CS60050) - Assignment 1 Report

Group 3

Vanshita Garg - 19CS10064

Ashutosh Kumar Singh - 19CS30008

Tasks

1. Implement a decision tree model to predict whether or not the patient has diabetes using different impurity measures (information gain and gini index).
2. Report the test accuracy by averaging the results over 10 random 80/20 train-test splits.
3. Study the effects of depth and node count variations on test accuracy and print the decision tree with the best possible depth limit.
4. Perform pruning operation over the tree with the highest test accuracy obtained from Task-2.

Dataset

This dataset contains the diagnostic predictions of whether or not a person has diabetes based on certain measurements of different attributes. In particular, all patients here are females at least 21 years old of Pima Indian heritage. The dataset has the following attributes:

- **Pregnancies** - Number of times pregnant
- **Glucose** - Plasma glucose concentration a 2 hours in an oral glucose tolerance test
- **Blood Pressure** - Diastolic blood pressure (mm Hg)
- **Skin Thickness** - Triceps skin fold thickness (mm)
- **Insulin** - 2-Hour serum insulin (mu U/ml)
- **BMI** - Body mass index (weight in kg/(height in m)^2)
- **Diabetes Pedigree Function**
- **Age** - The person's age in years

The target function is a boolean-valued classification of whether or not a patient has diabetes. The dataset has 768 training examples and is provided in the '.csv' format.

The Decision Tree Algorithm Used

- We have used the ID3 algorithm for constructing the decision tree. However, the standard ID3 algorithm is restricted to attributes that take on a discrete set of values. To incorporate the continuous-valued attributes in our dataset, we try to partition the continuous range of values into a discrete set of intervals.

- For each attribute, candidate thresholds are found out by first sorting its unique values and then finding the mid-points of all consecutive pairs of values. The mid-point that gives the largest information gain/gini gain is selected as the threhold and used for splitting. All data samples for which attribute value $<$ threshold are put in the left child, and samples for which attribute value $\geq$ threshold are put in the right child.
- Also note that as compared to the normal decision tree algorithm where we consider an attribute only once in one branch, here we can split on the basis of the same attribute more than once, because we want to be able to create multiple partitions of the range of values taken by an attribute.

Below is the pseudo-code of the procedure build_tree that explains the details of the algorithm.

```
build_tree(examples, depth):
    if size of examples == 1 or all examples have the same outcome value:
        return a leaf node with label same as the outcome value of the examples
    if depth == max depth:
        return a leaf node with label as the most probable value from the examples
    create node
    for attribute in list of attributes:
        sort the list of unique values of the attribute
        for midpoint of every consecutive pair of values:
            calculate information gain/gini gain on splitting around this value
            update maximum gain if required
        update attribute with maximum gain if required
    best_attr = attribute with maximum gain
    threshold = maximum gain for best_attr
    left_examples = examples with value of best_attr < threshold
    right_examples = examples with value of best_attr ≥ threshold
    node.left_child = build_tree(left_examples, depth + 1)
    node.right_child = build_tree(right_examples, depth + 1)
    return node
```

Some Important Terms and Definitions

Gini Index: $Gini = \sum_i p(i)(1-p(i))$. Considering any attribute A, $Gini(A) = \sum_{v \in A} p(v) \sum_i p(i|v)(1-p(i|v))$. Then we define the gini gain as: $GiniGain(A) = Gini - Gini(A)$.

Information Gain: For a collection S, $Entropy(S) = -\sum_i p(i)\, log_2\, p(i)$. The information gain is the reduction in entropy after choosing an attribute A. Mathematically,

$$InformationGain(S, A) = Entropy(S) \; - \sum_{\substack{v \in \\ values(A)}} \frac{|S_v|}{|S|} Entropy(S_v)$$

Accuracy : $$Accuracy = \frac{No.\ of\ examples\ correctly\ classified}{Total\ no.\ of\ examples}$$

Precision Score : $$Precision = \frac{True\,Positive}{True\,Positive + False\,Positive}$$

Recall Score : $Recall = \dfrac{True\,Positive}{True\,Positive + False\,Negative}$

F1 Score : $F1 = 2 \times \dfrac{Precision \times Recall}{Precision + Recall}$

The F1 score ranges from 0 to 1. The closer it is to 0, the better are our predictions.

Procedure and Results

1. Comparing Impurity Measures

Procedure :

- Split the dataset randomly as an 80/20 split (80% for training and 20% for testing)
- Build the decision tree using the algorithm described above using the two impurity measures (one by one) : 1) Gini Index, and 2) Information Gain.

During splitting, we always try to maximise the gini gain or the information gain. Also note that during this part, the maximum depth of the tree was kept as 5, as this is the optimal depth that we get from part 3.

Results :

Impurity Measure	Accuracy (in %)	F1 Score
Gini Index	73.3766	0.63
Information Gain	75.9740	0.66

We can clearly see that the accuracy and the F1 score, both are better in the case of information gain. Hence, we can conclude that information gain is a better measure of impurity for our case as compared to the gini index.
For all the next parts, we fix our impurity measure to be information gain, as it gives better results.

2. Determining Average Accuracy Over 10 Random Splits

Procedure :

- Divide the entire dataset into two parts - 80% training data, and 20% test data.
- Take 1/4th of the training data and keep it aside as the validation set, as it will be used later during pruning. Now our data looks like - 60% training set, 20% validation set and 20% test set.
- Repeat the above steps 10 times, and record the average test accuracy obtained
- Store the decision tree with the best test accuracy, as it will be used later for pruning.

An important point to note here is that we keep the maximum depth as 10 for this part. The reason for this is that we will be pruning the best tree obtained from this part, and hence we should allow it to grow to a sufficient depth. Secondly, after a depth of 10, the test accuracy becomes almost constant, and does not change because of the lack of more training samples. Hence, a maximum depth of 10 is the best choice.
Also note that we take the impurity measure to be information gain as we have already seen that it comes out to be better than gini index in our case.

Results :

Split No.	Test Accuracy (in %)
1	73.3766
2	67.5325
3	70.7792
4	72.0779
5	70.1299
6	67.5325
7	72.7273
8	68.8312
9	72.7273
10	66.8831

Average Test Accuracy: 70.2597%
Best Test Accuracy: 73.3766%

3. Variation of Depth and Number of Nodes

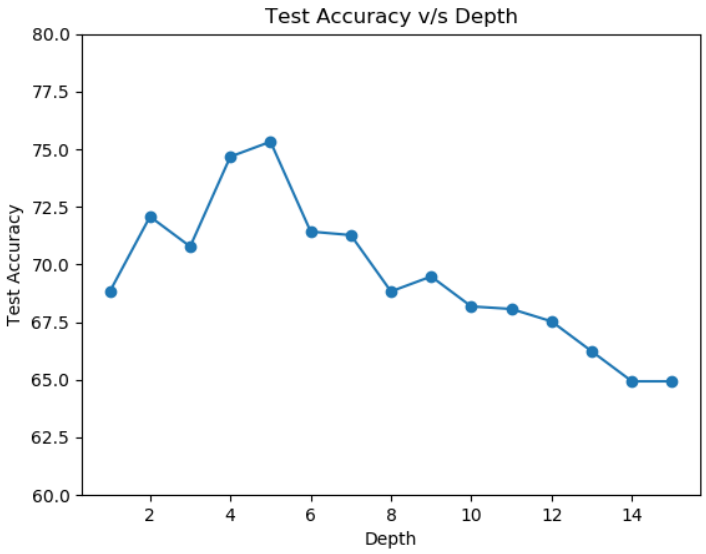
Procedure :

- Vary the maximum depth from 1 to 15 to observe the variation of test accuracy with the maximum depth of the decision tree
- Construct 10 decision trees for each depth using 10 random 80/20 splits, and take the average of the accuracies of each of them to compute the accuracy corresponding to that depth. Record this test accuracy for each depth.

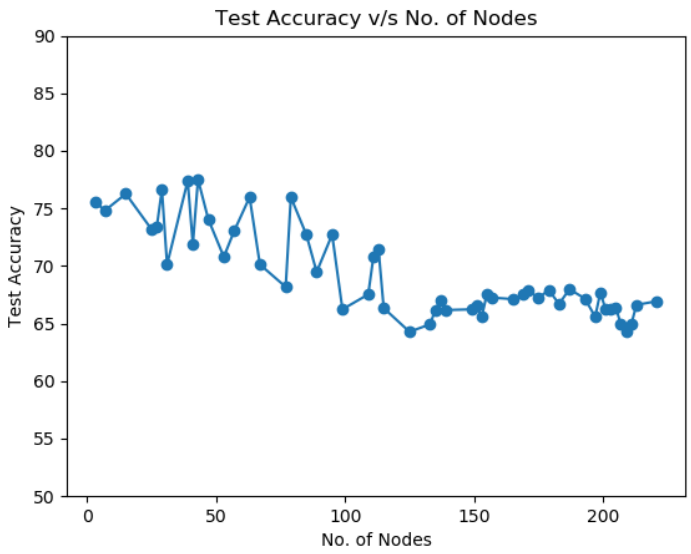
The bigger challenge is to observe the variation of test accuracy with the number of nodes in the tree. If we try to vary the number of nodes over a wide range, then constructing a decision tree for each of them would be computationally very expensive. So, we apply a smart trick here. While varying the depth itself, for each depth from 1 to 15, we are constructing 10 decision trees for each depth. So, for each of these trees, we count the number of nodes, and store them with the corresponding accuracy. This saves us a lot of time and effort, and we are able to combine the process of variation of depth and variation of nodes in a single function. Also, in this manner, we get sufficient number of points for number of nodes in order to plot a graph.

Results :

Depth	Training Accuracy (in %)	Test Accuracy (in %)
1	76.0586	68.8311
2	76.0591	72.0779
3	76.2214	70.7792
4	78.6644	74.6753
5	**81.5960**	**75.3246**
6	85.0162	71.4285
7	87.7850	71.2751
8	89.4136	68.8311
9	91.8566	69.4805
10	93.9739	68.1818
11	95.4397	68.0717
12	96.9055	67.5324
13	97.8827	66.2337
14	98.8599	64.9350
15	99.3485	64.9350



We get the best depth as 5, as is evident from the results and the plot. The training accuracy increases monotonically and approaches almost 100% at depth 15. This is expected because greater the depth of the tree, the more number of training examples can be accommodated in the decision tree. We can also see that after depth 10, the accuracy becomes very low. This is also in accordance with the theory and happens due to overfitting.



The same thing can be observed in case of the number of nodes too. Due to overfitting, the accuracy decreases and becomes almost constant when we have too many nodes in the tree. Also, since our tree is nearly a binary tree, we can observe that the number of nodes for maximum accuracy comes around the range 40 - 60, which corresponds to a depth of 5.

4. Pruning

Procedure:

- Take the decision tree with best test accuracy from part 2
- Start with the parents of the leaf nodes as leaf nodes cannot be pruned. For every such node, temporarily prune the subtree below, and check the accuracy on the validation set.
- If the accuracy increases, then permanently prune the tree below and make the current node a leaf node by assigning it the label of the majority vote.
- Move up the tree and continue the same process.
- Continue till the accuracy on the validation set increases

We have the decision tree with best test accuracy from part 2, now we apply the pruning operation in this tree. We had partitioned the 80% training data into the 60% grow set which we used to train and build the decision tree, and the 20% validation set. Now we use the validation set. The approach we take is known as reduced-error pruning. The benefit of pruning is that it helps in reducing overfitting significantly.

Results:

	No. of Nodes	Validation Accuracy (in %)	Test Accuracy (in %)
Before Pruning	119	69.7286	73.3766
After Pruning	33	77.2727	76.634

Major Classes and Functions Used in the Code

1. class `Node`

- `is_leaf` : Checks if the given node is a leaf or not.
- `node_count` : Finds the number of nodes in the subtree rooted at the given node.
- `prune` : Prunes a node by recursively first pruning the subtree of this node, and then checks if the current node can be pruned. Pruning continues till the accuracy on the validation set increases.
- `format_string` : Generates the string to be displayed in each node while printing the tree.

2. class `DecisionTree`

- `train` : Trains the decision tree model.
- `build_tree` : Builds the entire decision tree recursively, by deciding which attribute to split on, divides the data into two parts, and then calls the same function for the left and right subtrees.
- `get_best_attribute` : Finds the best attribute on the basis of which splitting should be done at a node, based on the data samples at that node.
- `create_leaf` : Creates and returns a leaf node for the decision tree.
- `predict_one` : Predicts the outcome of a single sample on the basis of the decision tree created by recursively travelling down the tree.
- `predict` : Determines the predictions of the decision tree model on a set of test data.
- `print_tree` : Prints the decision tree in a neat format using the graphviz package, and saves it in a '.png' format.

3. Utility Functions

- `entropy` : Calculates the entropy of a set of values.
- `information_gain` : Calculates the information gain if we split the data at a node on the basis of an attribute into 2 halves. One half with value $<$ split_val, and the other with value $\geq$ split_val.
- `gini_index` : Calculates the gini index of a set of values.
- `gini_gain` : Calculates the gini gain if we split the data on the basis of an attribute into 2 halves.
- `find_best_split` : Finds the best value of an attribute for splitting, given that particular attribute. It first sorts all the unique values of the attribute in ascending order. Then for the mid-point of each pair of consecutive values, it checks the feasibility of that point for splitting.
- `split_data` : Splits a given dataset into a random partition according to the given ratio.
- `split_df_col` : Splits a dataframe into a dataframe having only the attribute columns, and a series having the outcome values.
- `get_pred_accuracy` : Retrieves the predictions and accuracy of the predictions for a given decision tree and test dataset.

- `save_plot` : Creates a y vs. x plot and saves it.
- `precision` : Calculates the precision score based on predictions and true labels
- `recall` : Calculates the recall score based on predictions and true labels.
- `f1_score` : Calculates the F1 score based on predictions and true labels. It is the harmonic mean of the precision and recall.
- `dt_utility` : Returns a decision tree from the given train and test set and its accuracy on the train and test sets.
- `compare_measures` : Compares the impact of the two impurity measures - information gain and gini index by building decision trees using each of them.
- `select_best_tree` : Generates 10 decision trees based on 10 random 80/20 splits of the data, finds the average accuracy and returns the one with the best test accuracy.
- `vary_depth_nodes` : Analyzes the effect of variation in depth, and number of nodes on the performance of the decision tree.