Assignment - 1 Report

Decision Tree

—

Group-6

Aaditya Agrawal, 19CS10003
Debanjan Saha, 19CS30014

Dataset

Pima Indians Diabetes Database:
https://www.kaggle.com/uciml/pima-indians-diabetes-database

Procedure

About the Dataset

The dataset was obtained from Kaggle and is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

Analysing the Data

The dataset was in the form of a csv . It consists of several medical predictor variables and one target variable, ***Outcome*****.** The data had the following attributes:

1. Pregnancies
2. Glucose level
3. Blood Pressure
4. Skin Thickness
5. Insulin
6. BMI (Body Mass Index)

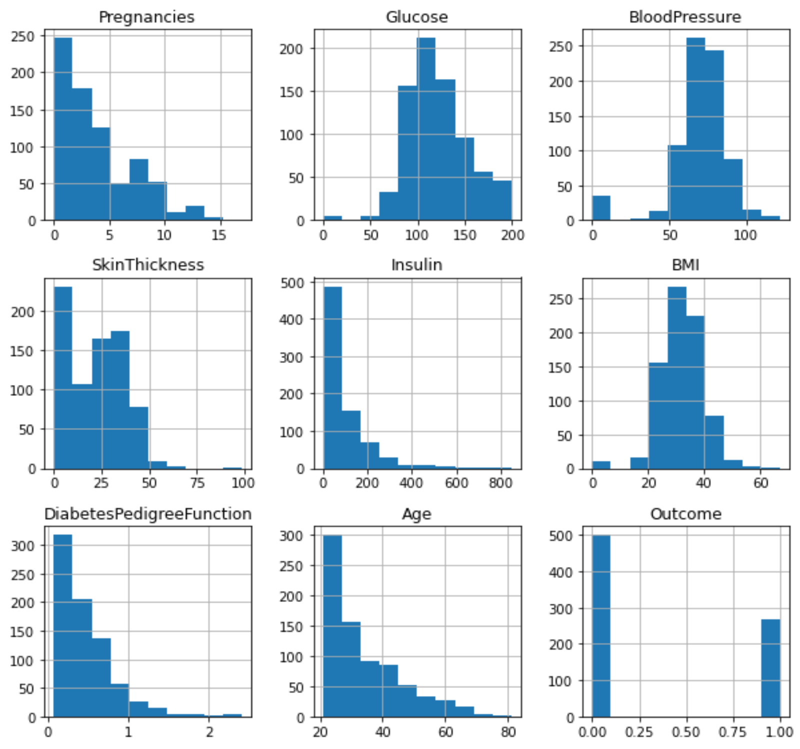
7. Diabetes Pedigree Function
8. Age

The dataset has 768 rows (observations). The visualization of data helps identify if there is missing data in the dataset. So we checked the following statistics related to the data.

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

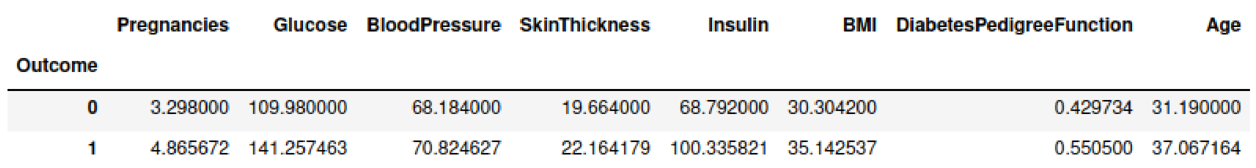
Count represents the number of Non-NULL entries in each column. As we observe, the count for each column is equal to the number of observations, therefore, there is no data missing. Thus, there is no need for data imputation.

We plotted histograms to understand the distribution of data within attributes.



From the above histogram, we find that the median and mean of BMI lies around 30 which indicates that a lot of patients who were observed were obese(BMI >= 30).

To observe the correlation of attributes with the outcome, we grouped the observations by outcome and found the mean for each of the attributes. Also we can see that the people having diabetes are approximately half of the people without diabetes in our database. We can also see that the glucose distribution is similar to a uniform distribution. Hence, we got a visualisation that it might come out to be the root node of our decision tree because of a uniform division and later this came true as we generated our final decision tree at a later stage in this assignment.



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
Outcome								
0	3.298000	109.980000	68.184000	19.664000	68.792000	30.304200	0.429734	31.190000
1	4.865672	141.257463	70.824627	22.164179	100.335821	35.142537	0.550500	37.067164

From the above table, we understand that diabetic women tend to have higher number of pregnancies, higher levels of glucose, higher Blood Pressure, Skin Thickness, Insulin, BMI, Diabetes Pedigree Function and Age. Women that have Diabetes have an average of insulin that is higher than the normal range.

After analysing the statistics and the histograms, we moved on to the coding part. We designed the model and created the helper functions.

Decision Tree Model

We have used the **ID3 algorithm** for constructing the decision tree. We split the dataset in the ratio of 60:20:20 as Training, Validation and Test Set respectively. Following is the description of the model class and various functions which were used while creating the model.

Main functions

- **build_decision_tree:** It builds the decision tree while using utility functions. It is a recursive function that calls itself to build the left and right subtree. The function chooses the best attribute to classify the node into two children nodes. The caller passes a pointer to the function which measures the impurity at a node. In our implementation, the impurity is measured using information gain or Gini index.
- **get_best_attr:** This function uses the above mentioned pointer to a function to calculate the gain and selects the best attribute using the data available at the node. This attribute is used in **build_decision_tree** to classify nodes.

- **predict_list:** This is a function which predicts the output for a list of data points on the basis of the root of the decision tree passed as parameter.
- **prune:** It prunes the decision tree that was built using **build_decision_tree.** It uses the validation set to prune the tree.
- **print_decision_tree:** This function prints the decision tree graph that was created using Graphviz package and saves the output in a pdf file.
- **get_best_depth:** This function calculates the best possible depth for our decision tree and generates 2 plots of accuracy vs depth and accuracy vs no. of nodes. It returns the best depth achieved along with its accuracy and the decision tree root node.

Utility functions

- **get_data_from_csv:** It loads the dataset into a pandas dataframe and returns it.
- **convert_data:** This function transforms the data from a dataframe to an array of dictionaries.
- **train_test_split:** Splits the data in the ratio of 80:20 between training and test after randomly shuffling the data. It also returns the list of attributes,
- **train_valid_split:** Splits the training data from above in a 75:25 ratio to create the final training and validation set.
- **remove_children:** Removes the children of a node and replaces the current node's value by the majority of the value of its children.
- **restore_children:** Restores the children of the current node.
- **count_nodes:** Recursive function to count the number of nodes in the subtree of the current node.
- **get_majority_leaf_value:** Finds out if the leaf nodes in the subtree have majority value true or false.
- **predict:** It predicts the output for a single sample on the basis of the root of the decision tree passed as parameter
- **details:** Returns a string related to the information of the node for passing it to graphviz package.
- **calculate_gini_index, calculate_information_gain:** These functions calculate the gini index and information gain respectively.
- **get_accuracy:** Given the decision tree and test input-output, it predicts the output of the model and returns the accuracy after comparing with the test output.

- **get_error:** Given the decision tree and test input-output, it predicts the output of the model and returns the error after comparing with the test output.

Pruning Process:

- We applied post-pruning to reduce overfitting. We first grew the tree using the training set, then applied **reduced-error** pruning which is a cross-validation approach. We followed the following steps:
    1. We start pruning from the parent of leaf nodes. For a node, we remove its children, making it a leaf node and replacing it with the majority vote. We then find the accuracy of this temporary tree on the validation set.
    2. If the error is reduced on the validation set, we remove the children permanently else we restore the children.
    3. We keep on doing this till no further pruning is possible.
- To summarize, we use the training set to **grow**, validation set to **prune**, and test set to estimate the **accuracy**.

Results

All the results below were obtained at the time when we ran the program. Due to the random splitting, the results you may obtain can be different.

Average accuracy over 10 random 80/20 splits:

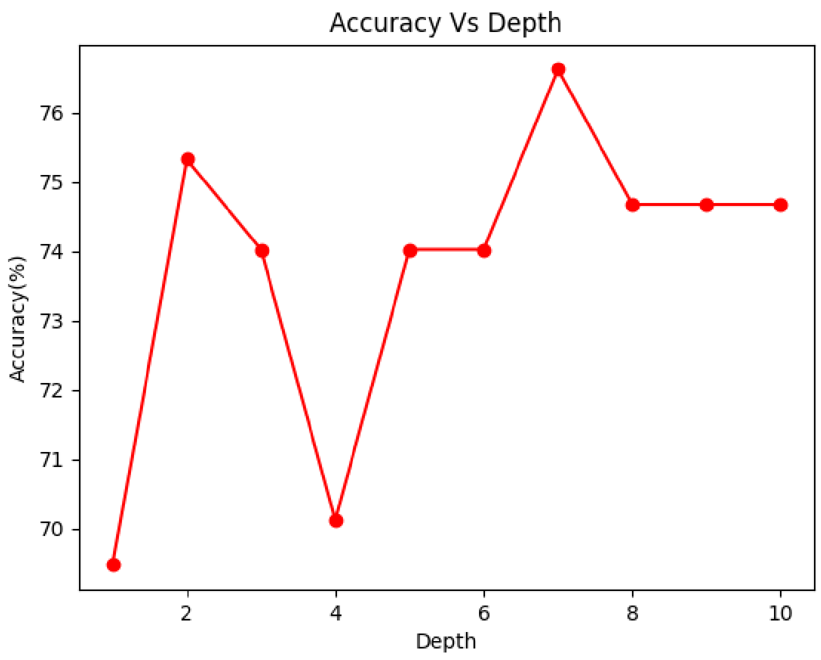
1. Using Gini index as measure of impurity: **68.127%**
2. Using Information Gain as measure of impurity: **69.316%**

Depth variation and Best possible depth limit

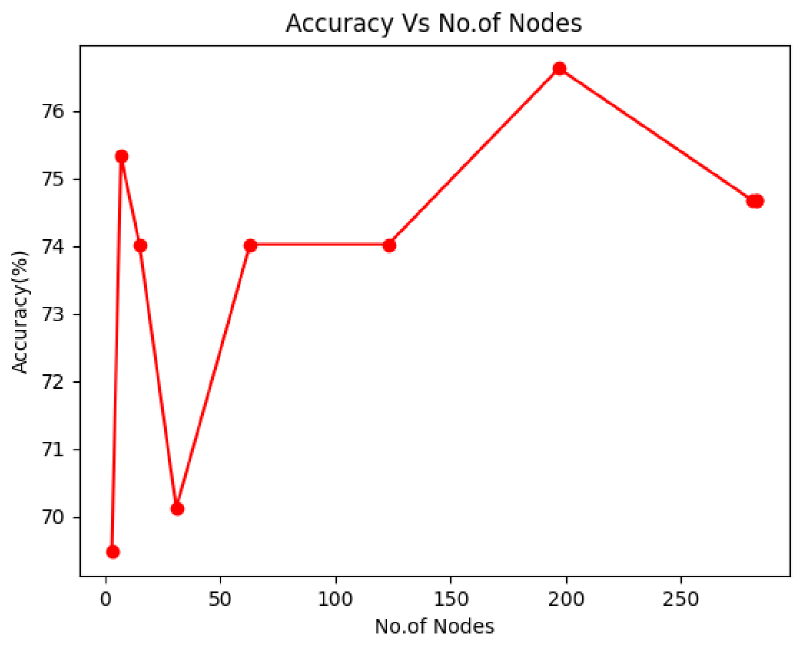
We found that the depth limit = **7** gives the best result on the test set. We set the max height as **10** (since the number of attributes were 9) , and found out the trees corresponding to that maximum depth. The table and plots are given below.

Max Depth	Test Accuracy

1	69.48051948051948
2	75.32467532467533
3	74.02597402597402
4	70.12987012987013
5	74.02597402597402
6	74.02597402597402
7	76.62337662337663
8	74.67532467532467
9	74.67532467532467
10	74.67532467532467



We also plotted the accuracy of test v/s total number of nodes. The plot is given below:



Pruning

- **Before pruning:** Number of nodes: **149** , Accuracy: **79.870%** (Accuracy of Best Tree)
- **After pruning:** Number of nodes: **93**, Accuracy: **80.519%**

Decision Tree with the hierarchical representation of Attributes:

The resultant tree is available in the submitted zip folder.

It can be found at the location: **pruned_tree.gv.pdf**

Total time taken by code during execution: **10.3286 seconds**