CS 6375
ASSIGNMENT 2
Decision Trees

Names of students in your group:
1. Rutuja Kaushike (rnk170000)
2. Akhila Kancharana (axk180025)

CS 6375.502

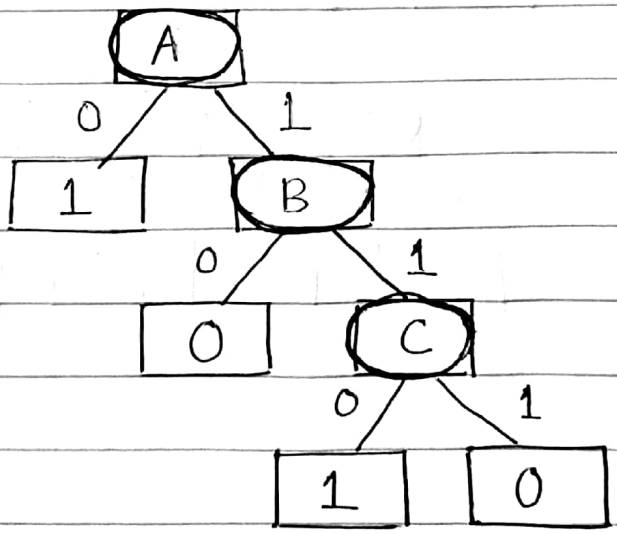
Number of free late days used: 0

Note: You are allowed a **total** of 4 free late days for the **entire semester**. You can use at most 2 for each assignment. After that, there will be a penalty of 10% for each late day.

Q1. Representing boolean functions:

(a) $Y = (\sim A \vee B) \wedge \sim(C \wedge A)$ where $\sim$ represents NOT operator.

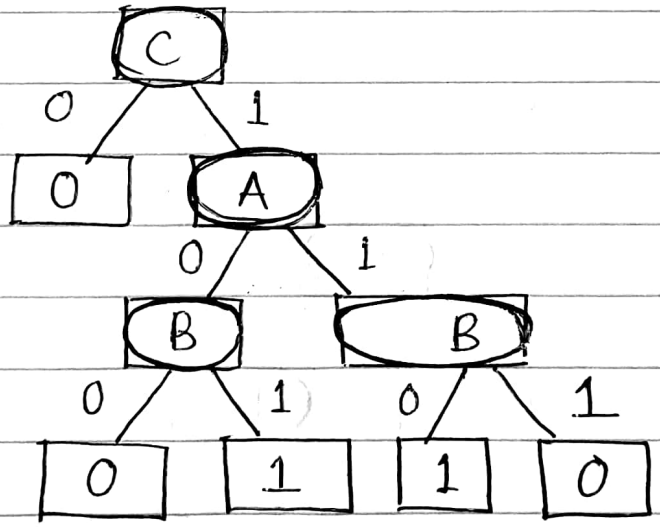
				I	II	III	
A	B	C	$\sim A$	$\sim A \vee B$	$C \wedge A$	$\sim(C \wedge A)$	$I \wedge III$
0	0	0	1	1	0	1	1
0	0	1	1	1	0	1	1
0	1	0	1	1	0	1	1
0	1	1	1	1	0	1	1
1	0	0	0	0	0	1	0
1	0	1	0	0	1	0	0
1	1	0	0	1	0	1	1
1	1	1	0	1	1	0	0



(b) $y = (A \oplus B) \wedge C$ $\oplus$ is XOR operator.

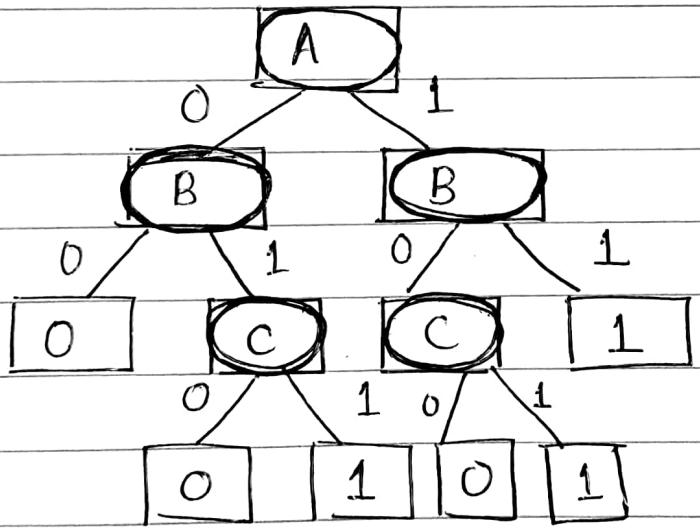
→

			I	
A	B	C	$A \oplus B$	$I \wedge C$
0	0	0	0	0
0	0	1	0	0
0	1	0	1	0
0	1	1	1	1
1	0	0	1	0
1	0	1	1	1
1	1	0	0	0
1	1	1	0	0



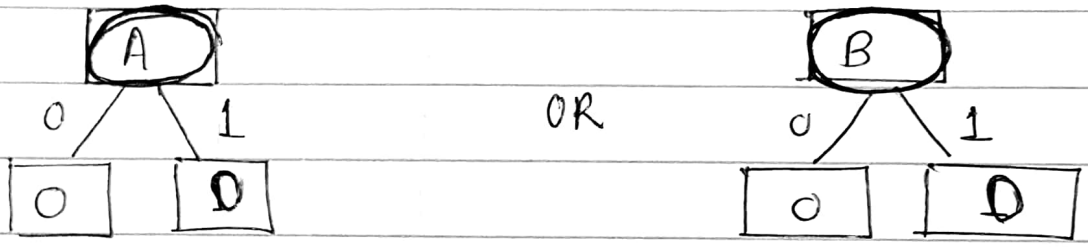
(c) $Y = (A \vee B) \wedge (B \vee C) \wedge (A \vee C)$

			I	II	III	
A	B	C	A∨B	B∨C	A∨C	I∧II∧III
0	0	0	0	0	0	0
0	0	1	0	1	1	0
0	1	0	1	1	0	0
0	1	1	1	1	1	1
1	0	0	1	0	1	0
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	1	1	1	1	1	1



(d) $Y = (A \vee B) \wedge \sim A \wedge \sim B$.

		I			II	
A	B	$A \vee B$	$\sim A$	$\sim B$	$\sim A \wedge \sim B$	$I \wedge II$
0	0	0	1	1	1	0
0	1	1	1	0	0	0
1	0	1	0	1	0	0
1	1	1	0	0	0	0



Q2. ID3 Algorithm.

→

X1	X2	X3	Class (Y)
1	0	0	1
0	1	0	1
0	0	0	0
1	0	1	0
0	0	0	0
1	1	0	1
0	1	1	0
1	0	0	1
0	0	0	0
1	0	0	1

Entropy of Class Y :

$$\text{Entropy}[5+, 5-] = -\frac{5}{10}\log\frac{5}{10} - \frac{5}{10}\log\frac{5}{10}$$

$$= 1.$$

For X1 :

X1 = 0

$$\text{Entropy}[1+, 4-] = -\frac{1}{5}\log\frac{1}{5} - \frac{4}{5}\log\frac{4}{5}$$

$$= -\left[\frac{1}{5}(-2.3219) + \frac{4}{5}(-0.3219)\right]$$

$$= (0.2)(2.3219) + (0.8)(0.3219)$$

$= 0.46438 + 0.25752$

$= 0.7219$

$x_1 = 1$

Entropy $[4+, 1-] = -\frac{4}{5}\log\frac{4}{5} - \frac{1}{5}\log\frac{1}{5}$

Information Gain (X_1) = Entropy (4) - weighted avg. of entropy of (x_1).

$= 1 - \left(\frac{1}{2} \times 0.7219 + \frac{1}{2} \times 0.7219\right)$

$= 1 - 0.7219$

$IG_{x_1} = 0.2781$

For x_2:

$x_2 = 0$

Entropy $[3+, 4-] = -\frac{3}{7}\log\frac{3}{7} - \frac{4}{7}\log\frac{4}{7}$

$x_2 = 1$ $= 0.98527$

Entropy $[2+, 1-] = -\frac{2}{3}\log\frac{2}{3} - \frac{1}{3}\log\frac{1}{3}$

$= 0.9182$

IG_{x_2} = Entropy (4) - Weighted Avg (x_2)

$= 1 - \left(\frac{7}{10} \times 0.98527 + \frac{3}{10} \times 0.9182\right)$

$= 1 - 0.965 = 0.035$

For X_3

$X_3 = 0$

Entropy $[5+, 3-] = -\frac{5}{8}\log\frac{5}{8} - \frac{3}{8}\log\frac{3}{8}$

$= 0.9543$

Entropy $[2+, 2-] = -\frac{2}{2}\log\left(\frac{2}{2}\right) = 0$

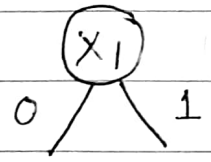
$X_3 = 1$

I.G. = Entropy (Y) - Weighted Avg (X_3)

$X_3 = 1 - \left(\frac{8}{10} \times 0.9543\right)$

$= 1 - 0.76348 = 0.23652$

As IG from X_1 is the highest, root = X_1.



$[1+, 4-]$ $[4+, 1-]$

Consider node X_2

~~$X_2 = 0$~~ =

~~$[0+, 3-]$~~

Now, let's classify further.

To classify again, we need to calculate Information gain for other attributes and select best classifier

for splitting the tree.

Thus, for X_2,

$X_2 = 0$

Entropy $[0+, 3-] = -0 \log_2 0 - \frac{3}{3} \log_2 \frac{3}{3} = 0$

$X_2 = 1$

Entropy $[1+, 1-] = -\frac{1}{2} \log_2 \frac{1}{2} - \frac{1}{2} \log_2 \frac{1}{2} = 1$

Information Gain $= 0.72192 - \frac{3}{5}(0) - \frac{2}{5}(1)$

$\therefore$ Information Gain $= 0.32192$

Now, let's solve for X_3,

for $X_3 = 0$

Entropy $[0+, 3-] = -\frac{1}{4} \log_2 \frac{1}{4} - \frac{3}{4} \log \frac{3}{4}$

$= 0.81127$

for $X_3 = 1$

Entropy $[0+, 1-] = -1 \cdot \log_2 1 \cdot = 0$

Information Gain $= 0.72192 - \frac{4}{5}(0.81127)$

$\therefore$ Information Gain for $X_3 = 0.0729$

Hence, now, we will split on $x_1 = 0$, we will consider x_2 as the next node.

for $x_1 = 0$ ~~and $x_2 = 0$~~,

we have

x_1	x_2	x_3	y
0	1	0	1
0	0	0	0
0	0	0	0
0	1	1	0
0	0	0	0

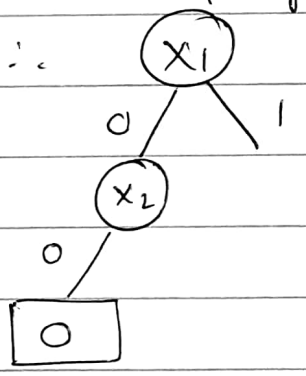
i.e. for $x_1 = 0$ & $x_2 = 0$, $y = 0$

~~$x_3 = 0$~~.

Hence, our tree stops at $x_2 = 0$.

tree in progress will be:

∴



$x_1 = 0$, for $x_2 = 1$,

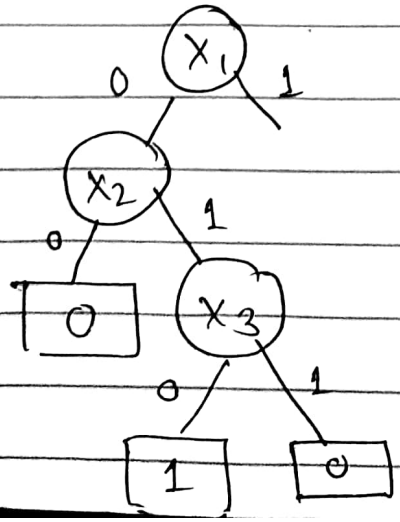
we have

x_1	x_2	x_3	y
0	1	0	1
0	1	1	0

i.e. when, $x_3 = 0$, $y = 1$

~~$x_3 = 1$~~, $y = 0$

∴ our tree will be,



Now, for $x_1 = 1$,

PTO

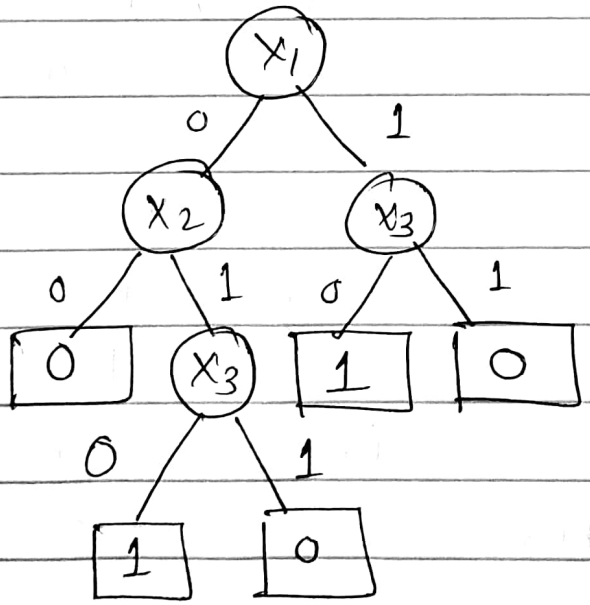
we have

x_1	x_2	x_3	y
1	0	0	1
1	0	1	0
1	1	0	1
1	0	0	1
1	0	0	1

when, $x_1 = 1$, irrespective of what x_2 is if $x_3 = 0$, $y = 1$

and $x_3 = 1$, $y \Rightarrow$

Hence, our final tree is

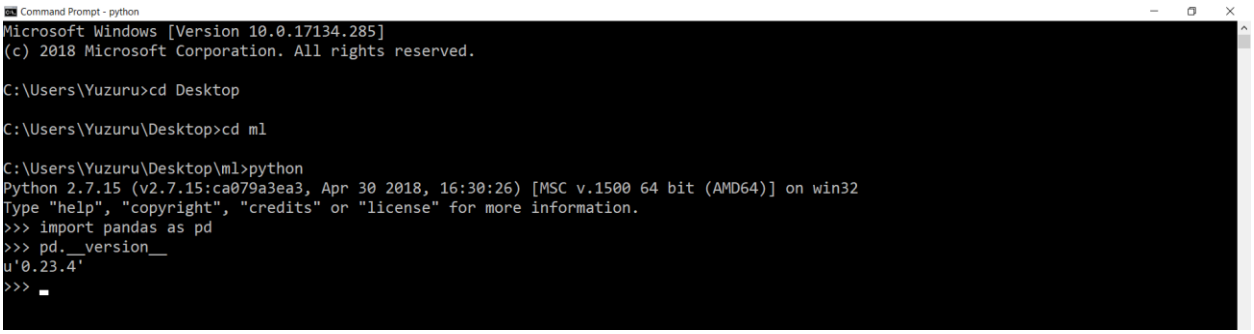


PART-2(Programming Part):

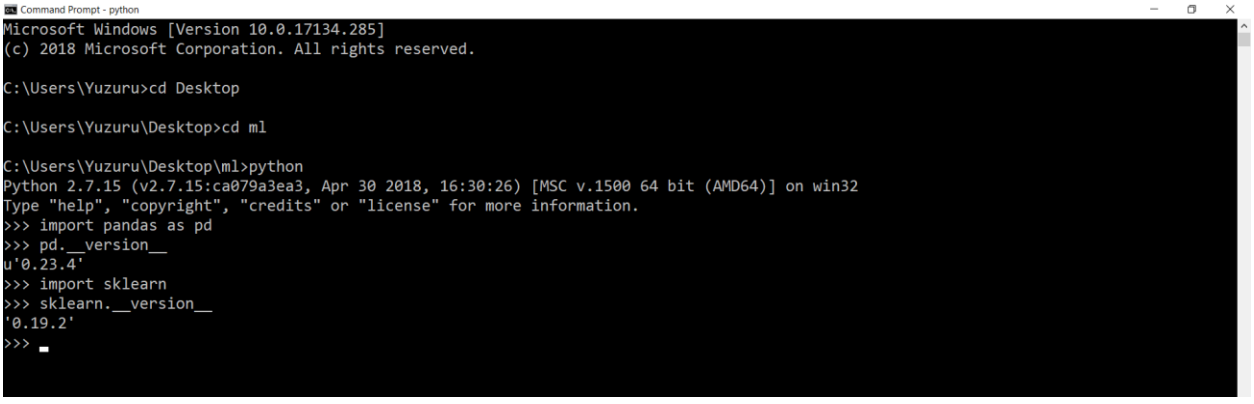
Q1) The code has the following dependencies, which need to be installed before running this code:
(a) Pandas. More details at: https://pandas.pydata.org/
(b) Scikit Learn for only one method in the driver code - train test split
You have to make changes only in the DecisionTree.py file to get everything in the driver.py to run.

A)

a) **Pandas(install):** Screen shot after installation and running pandas version



b) **Scikit (install):** Screen shot after installation and running scikit version.



Q2) The code for each decision node in class Decision Node stores information about non-leaf nodes. Besides the information given in the code on line 278 of the original code, you need to add following properties for each node:

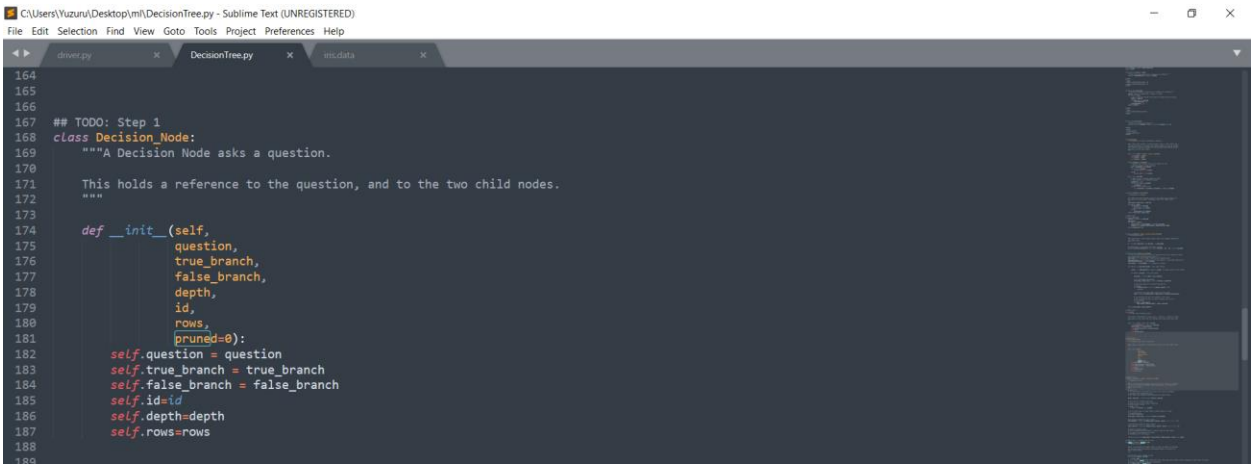
- depth of the node - starting from the root node, which is at depth 0
- id of the node, using the following schema:

The root node will have an id of 0. We assume that each node will split into two branches only - left and right. If a node has id of n, then its left child will have id of 2n + 1 and right child will have id of 2n + 2. This will ensure that there is no clash in node ids.

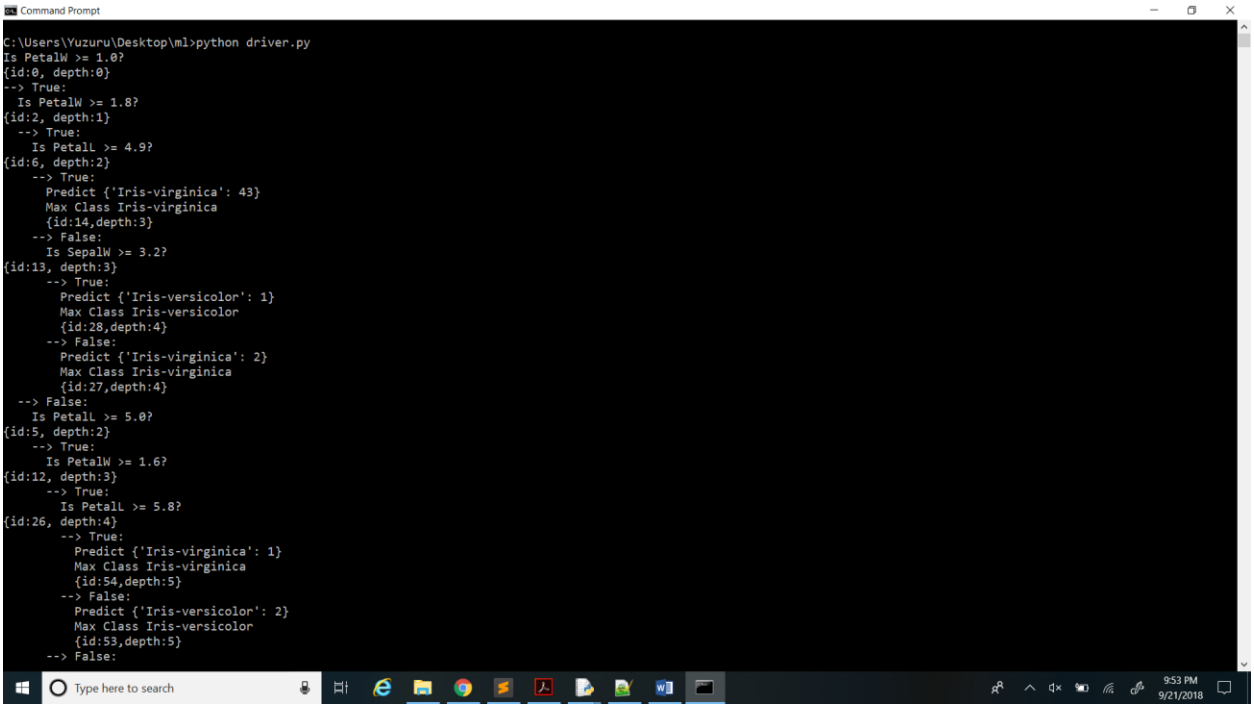
- the rows of data that it contains

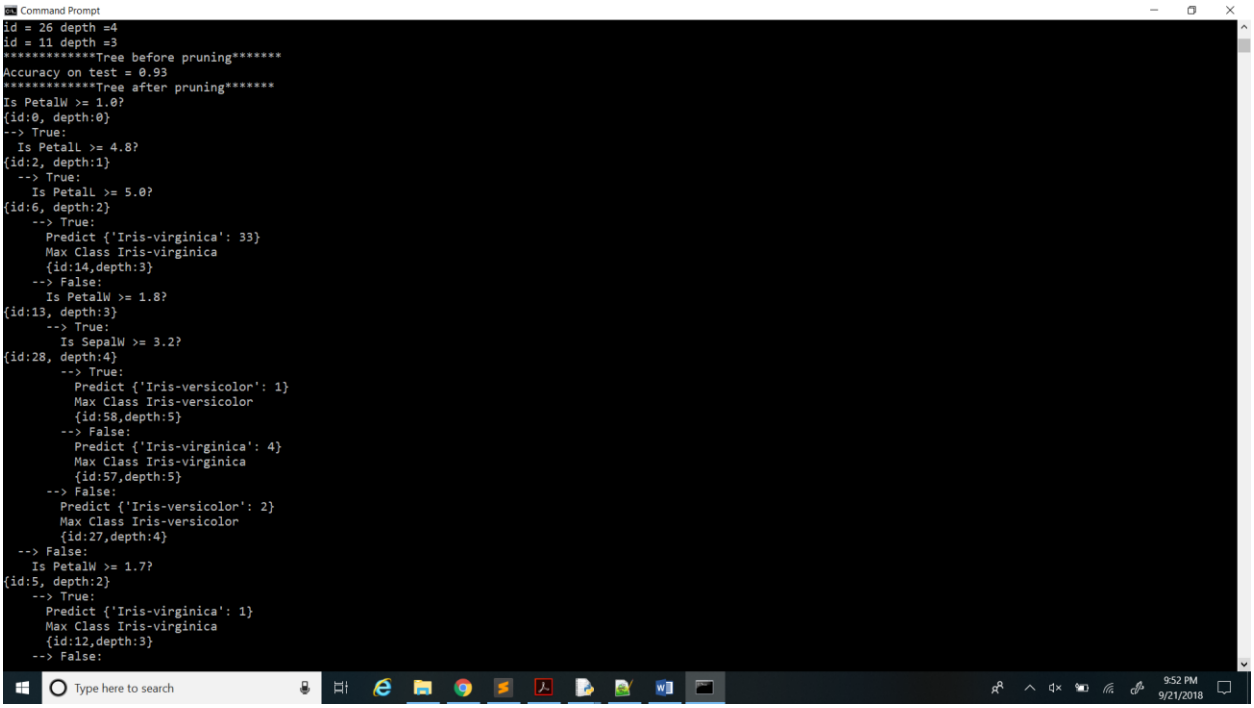
A2) Printed id's and depth of nodes.

Code:



Output:


Command Prompt
C:\Users\Yuzuru\Desktop\ml>python driver.py
Is PetalW >= 1.0?
{id:0, depth:0}
--> True:
Is PetalW >= 1.8?
{id:2, depth:1}
--> True:
Is PetalL >= 4.9?
{id:6, depth:2}
--> True:
Predict {'Iris-virginica': 43}
Max Class Iris-virginica
{id:14,depth:3}
--> False:
Is SepalW >= 3.2?
{id:13, depth:3}
--> True:
Predict {'Iris-versicolor': 1}
Max Class Iris-versicolor
{id:28,depth:4}
--> False:
Predict {'Iris-virginica': 2}
Max Class Iris-virginica
{id:27,depth:4}
--> False:
Is PetalL >= 5.0?
{id:5, depth:2}
--> True:
Is PetalW >= 1.6?
{id:12, depth:3}
--> True:
Is PetalL >= 5.8?
{id:26, depth:4}
--> True:
Predict {'Iris-virginica': 1}
Max Class Iris-virginica
{id:54,depth:5}
--> False:
Predict {'Iris-versicolor': 2}
Max Class Iris-versicolor
{id:53,depth:5}
--> False:
Type here to search
9:53 PM
9/21/2018


Command Prompt
id = 26 depth =4
id = 11 depth =3
**********Tree before pruning*******
Accuracy on test = 0.93
**********Tree after pruning*******
Is PetalW >= 1.0?
{id:0, depth:0}
--> True:
Is PetalL >= 4.8?
{id:2, depth:1}
--> True:
Is PetalL >= 5.0?
{id:6, depth:2}
--> True:
Predict {'Iris-virginica': 33}
Max Class Iris-virginica
{id:14,depth:3}
--> False:
Is PetalW >= 1.8?
{id:13, depth:3}
--> True:
Is SepalW >= 3.2?
{id:28, depth:4}
--> True:
Predict {'Iris-versicolor': 1}
Max Class Iris-versicolor
{id:58,depth:5}
--> False:
Predict {'Iris-virginica': 4}
Max Class Iris-virginica
{id:57,depth:5}
--> False:
Predict {'Iris-versicolor': 2}
Max Class Iris-versicolor
{id:27,depth:4}
--> False:
Is PetalW >= 1.7?
{id:5, depth:2}
--> True:
Predict {'Iris-virginica': 1}
Max Class Iris-virginica
{id:12,depth:3}
--> False:
Type here to search
9:52 PM
9/21/2018

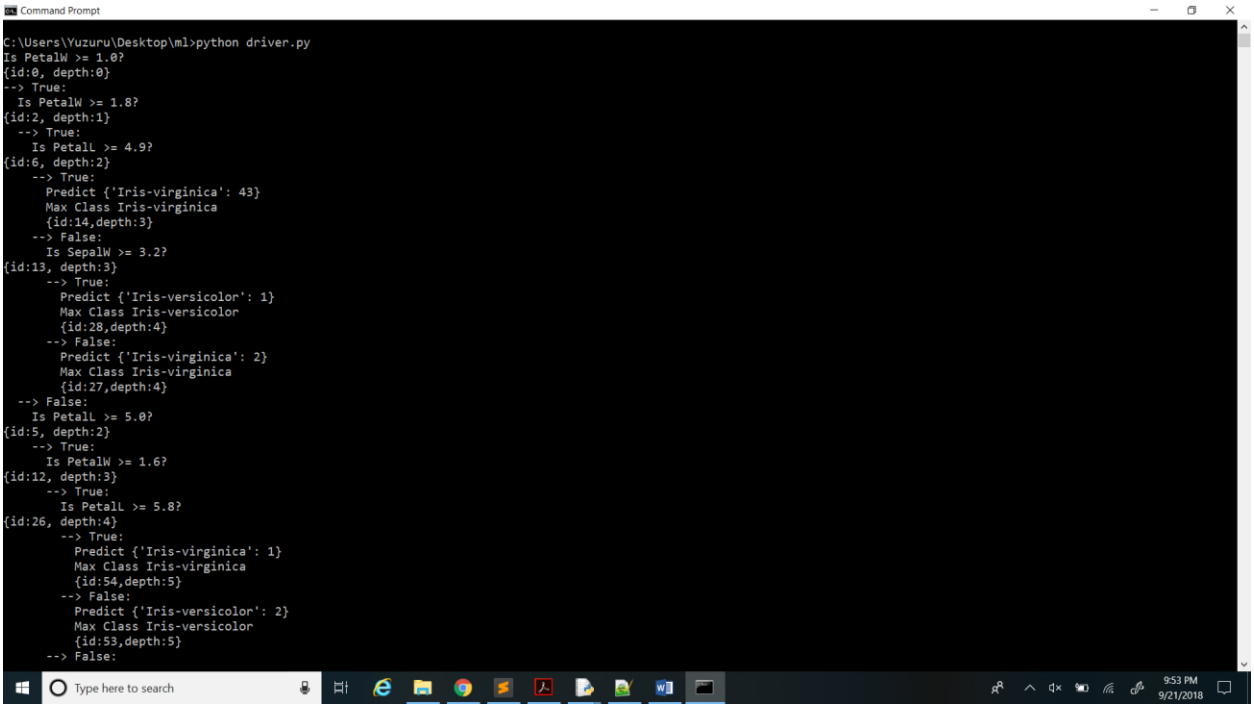
Q3) We will modify the definition of the leaf node in line 267 so that it stores the following properties besides the ones given:

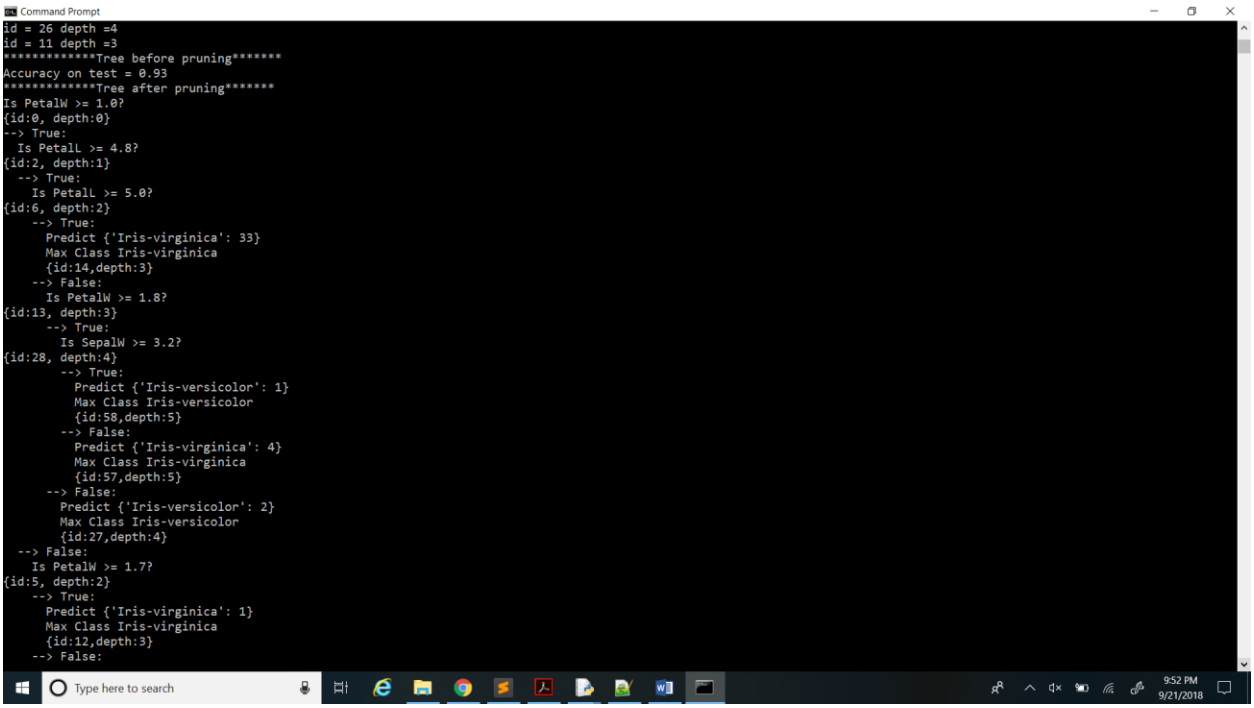
- predicted class label. Note that the predictions is a dictionary that contains the class counts. You will need to extract the majority class in it and output that.
- id of the leaf node
- depth of the leaf node

Code:

```
150  ## TODO: Step 2
151  class Leaf:
152      """A Leaf node classifies data.
153
154      This holds a dictionary of class (e.g., "Apple") -> number of times
155      it appears in the rows from the training data that reach this leaf.
156      """
157
158      def __init__(self, rows, id, depth):
159          self.predictions = class_counts(rows)
160          count_temp=self.predictions
161          self.predictions_cls=max(count_temp)
162          self.id=id
163          self.depth=depth
164
```

Output:





Q4) Next, you need to modify the build tree function. This is the main function that builds the tree from top (root mode) to bottom (leaf nodes) recursively. You need to modify the tree definition based on the conditions of parts 1 and 2 above. To construct each node, you need to pass besides the rows, other parameters such as depth, id, and header information. The header information is necessary because we will not hard code this information, like it is done in line 27 of the original code. The new definition of the build tree function would be as follows:

Make sure you finish and test this function before going ahead. Also, the example finds the best split using the find best split function that internally uses the function info gain function, which uses the Gini index for impurity (line 138 of original code). In class, we have studied Entropy measure. You would need to replace the gini function with an entropy function.

A4) Build Tree:

```
def build_tree(rows, header, depth=0, id=0):
    """Builds the tree.

    Rules of recursion: 1) Believe that it works. 2) Start by checking
    for the base case (no further information gain). 3) Prepare for
    giant stack traces.
    """
    # depth = 0
    # Try partitioing the dataset on each of the unique attribute,
    # calculate the information gain,
    # and return the question that produces the highest gain.

    gain, question = find_best_split(rows, header)

    # Base case: no further info gain
    # Since we can ask no further questions,
    # we'll return a leaf.
    if gain == 0:
        return Leaf(rows, id, depth)

    # If we reach here, we have found a useful feature / value
    # to partition on.
    # nodeList.append(id)
    true_rows, false_rows = partition(rows, question)

    # Recursively build the true branch.
    true_branch = build_tree(true_rows, header, depth + 1, 2 * id + 2 )

    # Recursively build the false branch.
    false_branch = build_tree(false_rows, header, depth + 1, 2 * id + 1)

    # Return a Question node.
    # This records the best feature / value to ask at this point,
    # as well as the branches to follow
    # depending on on the answer.

    return Decision_Node(question, true_branch, false_branch, depth, id, rows)
```

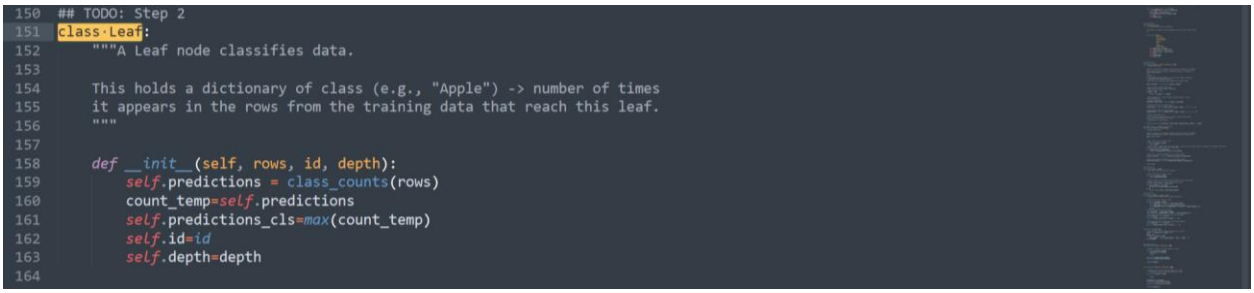
Entropy:

```
## TODO: Step 3
def entropy(rows):
    counts = class_counts(rows)
    impurity=0
    for lbl in counts:
        prob_of_lbl = counts[lbl] / float(len(rows))
        impurity += (prob_of_lbl*(math.log(prob_of_lbl,2)))
    return (impurity*-1)
```

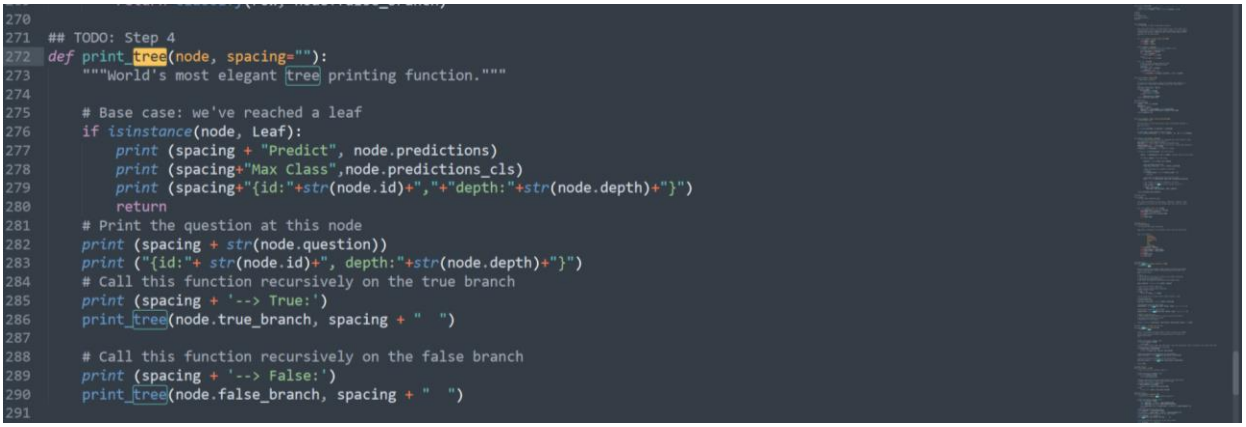
Q5) Now, we will modify the print tree function to print more details from the tree. Besides the current details that are outputted, we would also like to output the depth, node id of the decision nodes. For the leaf label, we need to see the predicted class label. You can output the predicted dictionary also, but that is optional.

A5)

Code: Max Prediction and printing

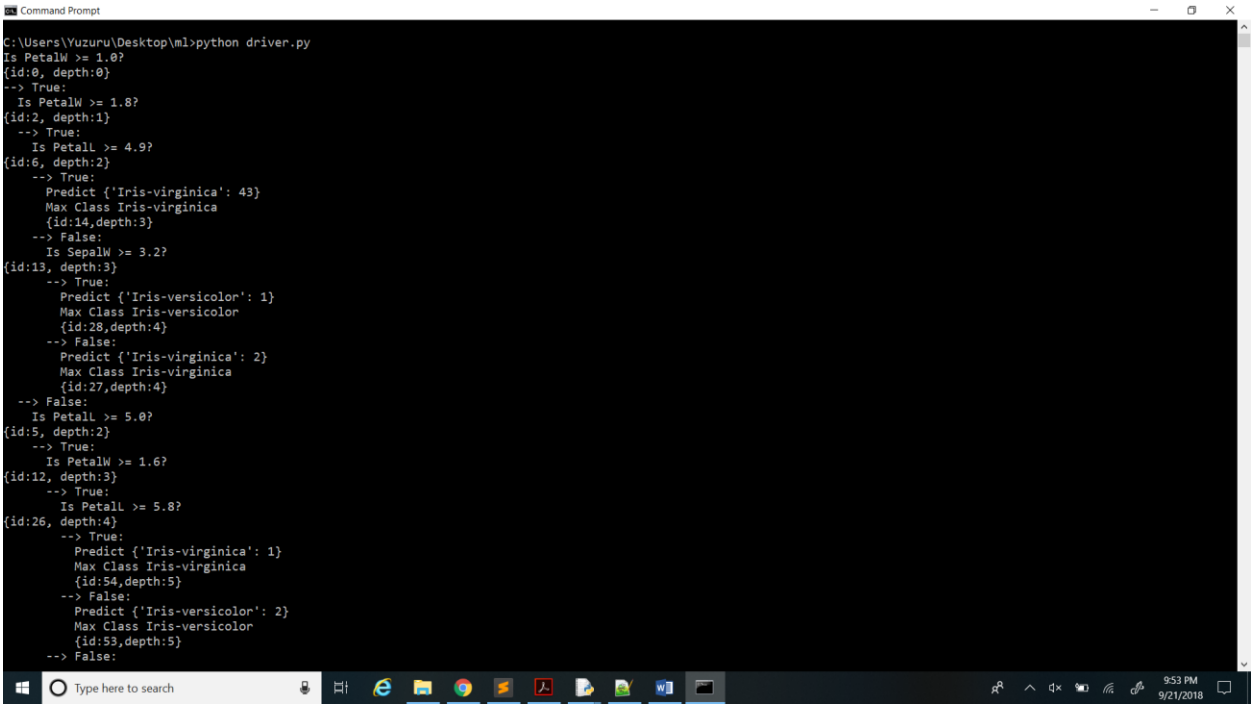


```
## TODO: Step 2
class Leaf:
    """A Leaf node classifies data.

    This holds a dictionary of class (e.g., "Apple") -> number of times
    it appears in the rows from the training data that reach this leaf.
    """

    def __init__(self, rows, id, depth):
        self.predictions = class_counts(rows)
        count_temp=self.predictions
        self.predictions_cls=max(count_temp)
        self.id=id
        self.depth=depth

```



```

## TODO: Step 4
def print_tree(node, spacing=""):
    """World's most elegant tree printing function."""

    # Base case: we've reached a leaf
    if isinstance(node, Leaf):
        print (spacing + "Predict", node.predictions)
        print (spacing+"Max Class",node.predictions_cls)
        print (spacing+"{id:"+str(node.id)+","+"depth:"+str(node.depth)+"}")
        return
    # Print the question at this node
    print (spacing + str(node.question))
    print ("{id:"+ str(node.id)+", depth:"+str(node.depth)+"}")
    # Call this function recursively on the true branch
    print (spacing + '--> True:')
    print_tree(node.true_branch, spacing + "  ")

    # Call this function recursively on the false branch
    print (spacing + '--> False:')
    print_tree(node.false_branch, spacing + "  ")

```

Output: Max Class Prediction at Leaf nodes.



Q6) Next, you will write two functions - one that returns a list of all inner nodes (i.e. those nodes that are not leaf nodes) and another function that returns a list of leaf nodes. Note that you have to return the nodes, not just their ids. You can call these functions getInnerNodes and getLeafNodes.
Both of these should have one input parameter - the root node of the tree.
You can see an example of this in the print tree function - without the spacing parameter, of course.

A6) Code:

```
## TODO: Step 5
def getLeafNodes(node, leafNodes =[]):

    # Returns a list of all leaf nodes of a tree
    if isinstance(node, Leaf):
        leafNodes.append(node)
        return

    getLeafNodes(node.true_branch)
    getLeafNodes(node.false_branch)

    return leafNodes


def getInnerNodes(node, innerNodes =[]):

    # Returns a list of all non-leaf nodes of a tree
        # Returns a list of all leaf nodes of a tree
    if isinstance(node, Leaf):

        return

    innerNodes.append(node)
    getInnerNodes(node.true_branch)
    getInnerNodes(node.false_branch)

    return innerNodes
```

Output:

```
 {id:1,depth:1}
********* Leaf nodes ***************
id = 14 depth =3
id = 28 depth =4
id = 27 depth =4
id = 54 depth =5
id = 53 depth =5
id = 25 depth =4
id = 24 depth =4
id = 23 depth =4
id = 1 depth =1
********* Non-leaf nodes ***************
id = 0 depth =0
id = 2 depth =1
id = 6 depth =2
id = 13 depth =3
id = 5 depth =2
id = 12 depth =3
id = 26 depth =4
id = 11 depth =3
************Tree before pruning*******
```

Q7) We will modify the classify function so that it outputs just the predicted class label, instead of class counts. The classify function works on any 1 row of data. We will also create a function called computeAccuracy that takes in a set of rows and returns their accuracy.
Below is the signature of that function:

A7) Code:

Classify:

```
254
255 ## TODO: Step 6
256 def classify(row, node):
257     """See the 'rules of recursion' above."""
258
259     # Base case: we've reached a leaf
260     if isinstance(node, Leaf):
261         return node.predictions
262
263     # Decide whether to follow the true-branch or the false-branch.
264     # Compare the feature / value stored in the node,
265     # to the example we're considering.
266     if node.question.match(row):
267         return classify(row, node.true_branch)
268     else:
269         return classify(row, node.false_branch)
270
```

Accuracy:

```
330
331 ## TODO: Step 6
332 def computeAccuracy(rows, node):
333     totalRows = len ( rows )
334     numAccurate = 0
335     for row in rows :
336         pred_label=classify(row,node)
337         if row[-1] in pred_label.keys():
338             numAccurate = numAccurate + 1
339
340     return round( numAccurate/float(totalRows) ,2)
341
342
```

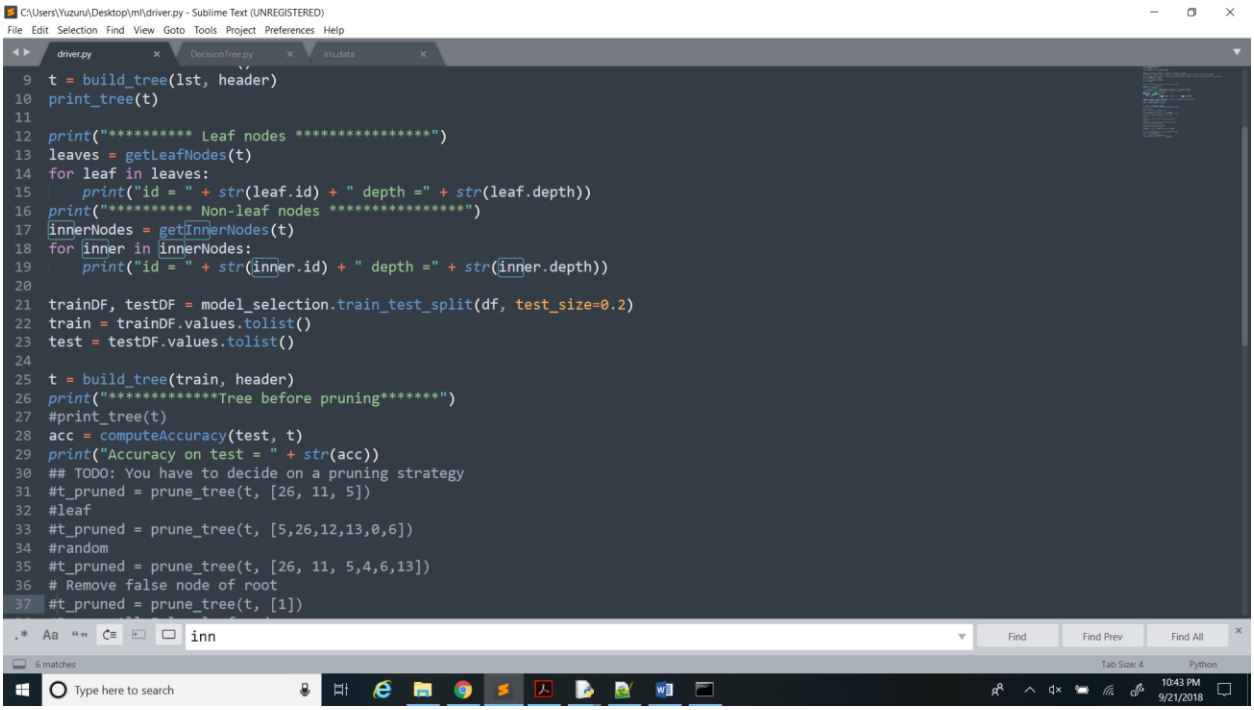
Output: Accuracy after pruning is slightly greater.

```
Accuracy on test = 0.93
*************Tree after pruning*******
Is PetalW >= 1.0?
{id:0, depth:0}
--> True:
  Is PetalW >= 1.8?
{id:2, depth:1}
  --> True:
    Is PetalL >= 4.9?
{id:6, depth:2}
    --> True:
      Predict {'Iris-virginica': 34}
      Max Class Iris-virginica
      {id:14,depth:3}
```

```
   --> False:
     Predict {'Iris-virginica': 1, 'Iris-versicolor': 38}
     Max Class Iris-virginica
     {id:11,depth:3}
--> False:
  Predict {'Iris-setosa': 43}
  Max Class Iris-setosa
  {id:1,depth:1}
Accuracy on test = 0.97
```

Q8) For testing our model, we will use Scikit Learn's train test split method. This will allow you to choose a random sample of the dataset for training and remaining for test.

A8) Code:



Q9) You would need to decide a pruning strategy?

A9)

(a) Prune each node that is 1 level above the leaf e.g. node 2 in the figure above and see if it helps improve the test error
Code:

```
#leaf
t_pruned = prune_tree(t, [5,26,12,13,0,6])
```

Output: Getting an over fit of data, therefore accuracy is same.

```
*************Tree before pruning*******
Accuracy on test = 0.97
*************Tree after pruning*******
Predict {'Iris-virginica': 37, 'Iris-setosa': 40, 'Iris-versicolor': 43}
Max Class Iris-virginica
{id:0,depth:0}
Accuracy on test = 0.97

C:\Users\Yuzuru\Desktop\ml>
```

(b) Randomly select n nodes and prune them as a batch, and see if it improves accuracy

Code: t_pruned = prune_tree (t, [26, 11, 5,4,6,13])

Output: After pruning the accuracy is greater.

```
*************Tree before pruning*******
Accuracy on test = 0.93
*************Tree after pruning*******
Is PetalW >= 1.0?
{id:0, depth:0}
--> True:
  Is PetalW >= 1.8?
{id:2, depth:1}
  --> True:
    Predict {'Iris-virginica': 39, 'Iris-versicolor': 1}
    Max Class Iris-virginica
    {id:6,depth:2}
  --> False:
    Predict {'Iris-virginica': 3, 'Iris-versicolor': 36}
    Max Class Iris-virginica
    {id:5,depth:2}
--> False:
  Predict {'Iris-setosa': 41}
  Max Class Iris-setosa
  {id:1,depth:1}
Accuracy on test = 1.0

C:\Users\Yuzuru\Desktop\ml>
```

(c) Anything else that you can think of

Code:

a) Remove false node of root

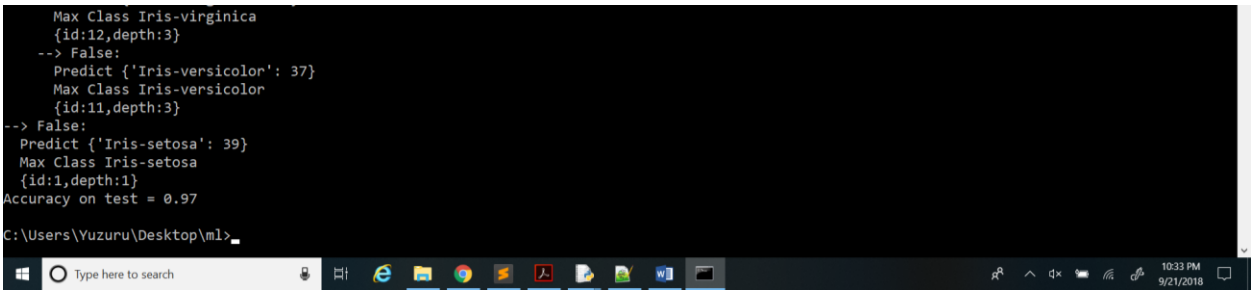
t_pruned = prune_tree (t, [1])

Output: Overfitting happens when you remove the false node of root.

b) Remove All False leaf nodes

t_pruned = prune_tree (t, [1, 11,25,53,27])

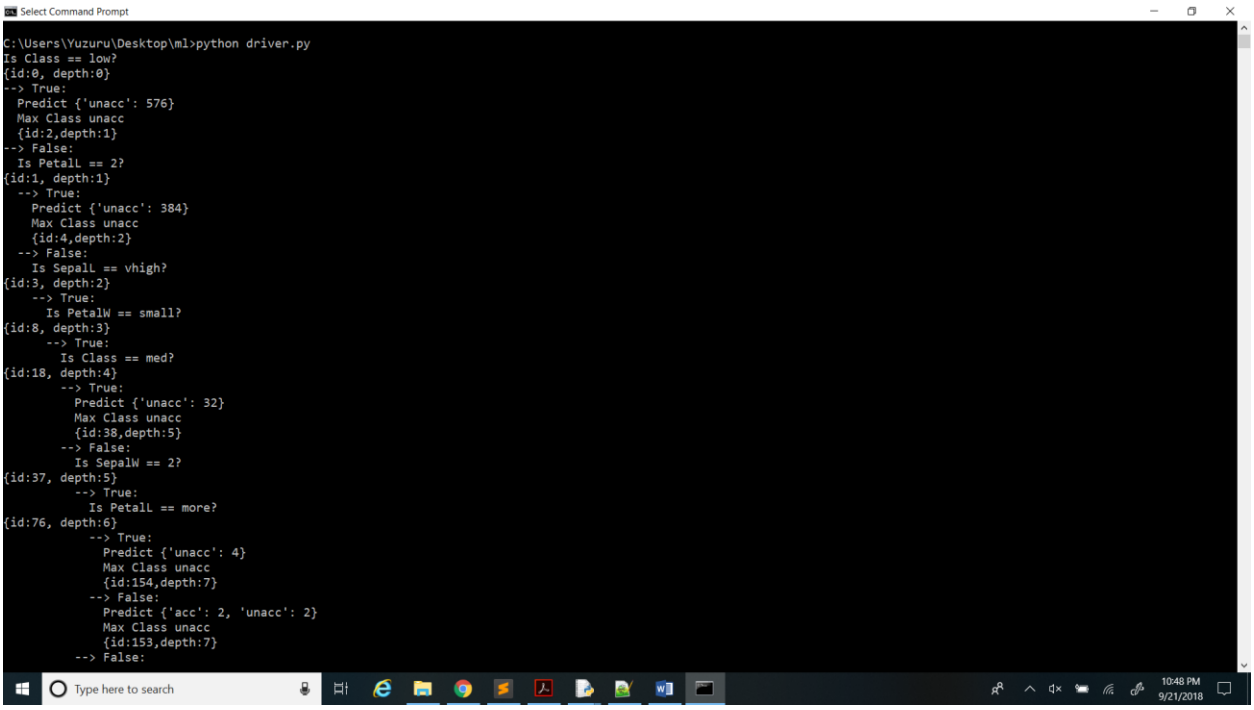
output: accuracy is greater after pruning all false nodes.

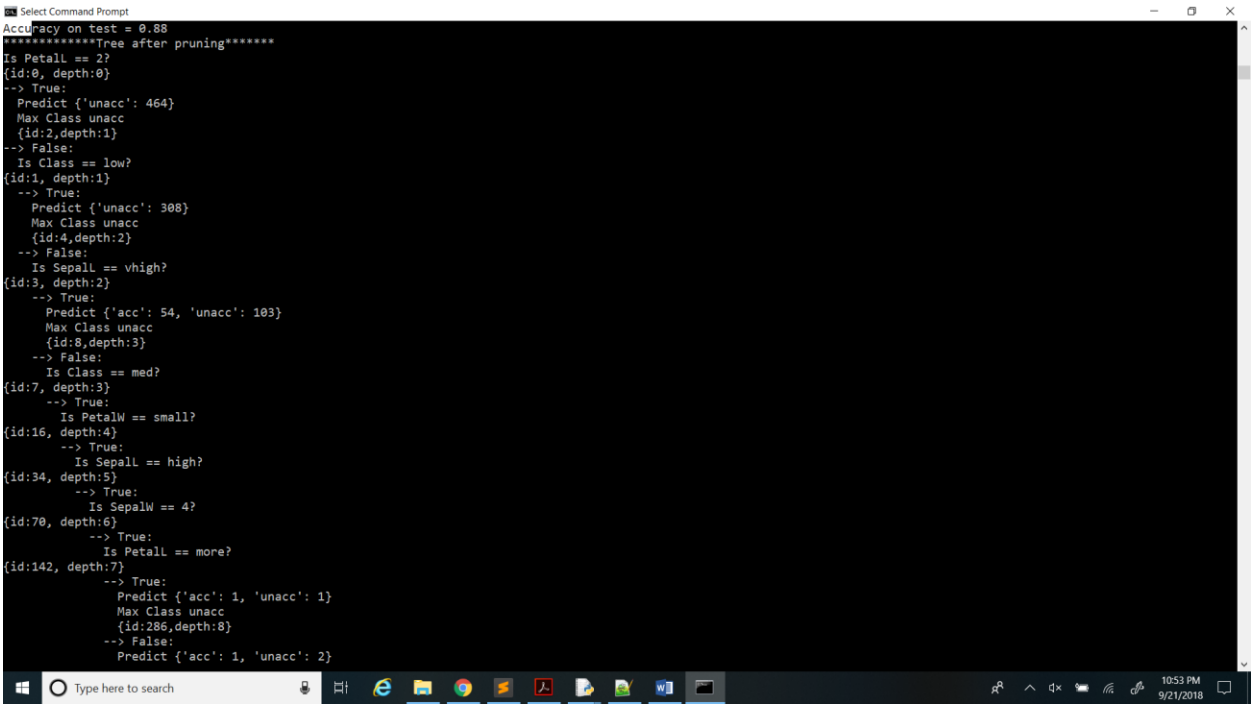


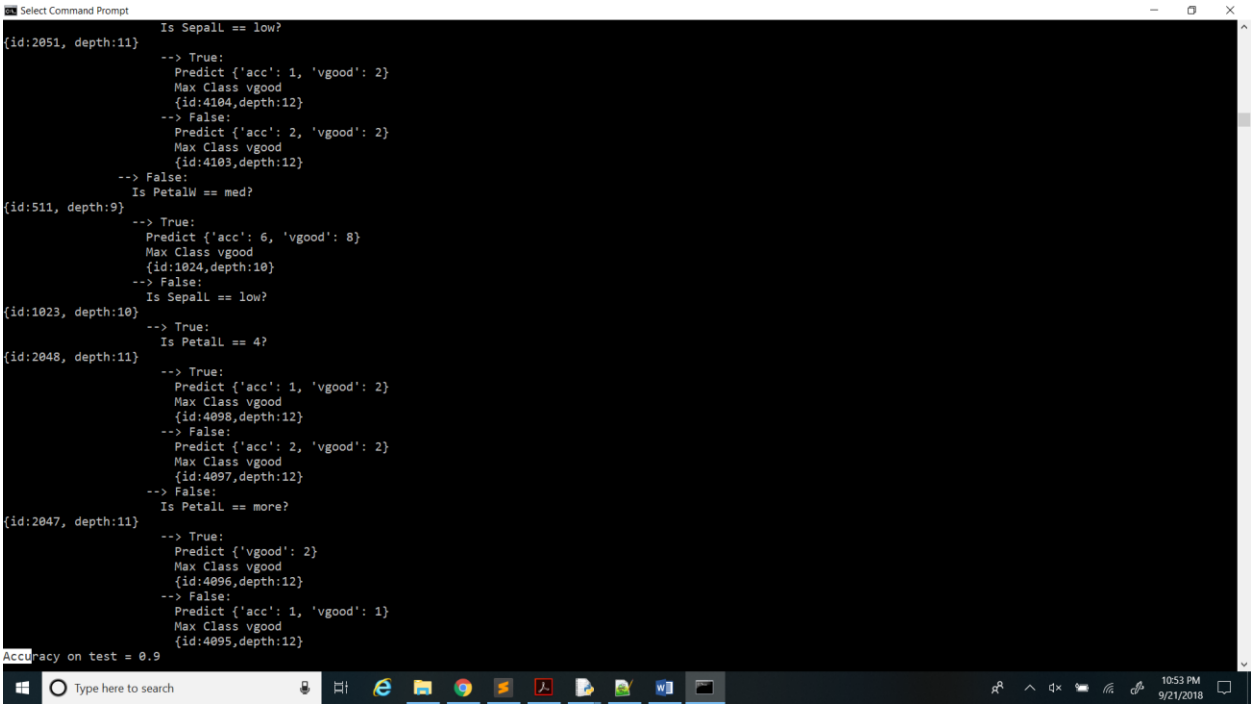
Q10) After making the above changes, you have to get everything in the driver.py file to run without error. and then choose another dataset from UCI Machine Learning dataset that is suitable for classification tasks and get it to work using the code that you created earlier.

A10) Car Data Set:

Output: As Data Set is too big we are just adding imp parts of output. Accuracy after pruning (0.9) is greater than (0.88)




Select Command Prompt
Accuracy on test = 0.88
************Tree after pruning*******
Is PetalL == 2?
{id:0, depth:0}
--> True:
Predict {'unacc': 464}
Max Class unacc
{id:2,depth:1}
--> False:
Is Class == low?
{id:1, depth:1}
--> True:
Predict {'unacc': 308}
Max Class unacc
{id:4,depth:2}
--> False:
Is SepalL == vhigh?
{id:3, depth:2}
--> True:
Predict {'acc': 54, 'unacc': 103}
Max Class unacc
{id:8,depth:3}
--> False:
Is Class == med?
{id:7, depth:3}
--> True:
Is PetalW == small?
{id:16, depth:4}
--> True:
Is SepalL == high?
{id:34, depth:5}
--> True:
Is SepalW == 4?
{id:70, depth:6}
--> True:
Is PetalL == more?
{id:142, depth:7}
--> True:
Predict {'acc': 1, 'unacc': 1}
Max Class unacc
{id:286,depth:8}
--> False:
Predict {'acc': 1, 'unacc': 2}
Type here to search
10:53 PM
9/21/2018


Select Command Prompt
Is SepalL == low?
{id:2051, depth:11}
--> True:
Predict {'acc': 1, 'vgood': 2}
Max Class vgood
{id:4104,depth:12}
--> False:
Predict {'acc': 2, 'vgood': 2}
Max Class vgood
{id:4103,depth:12}
--> False:
Is PetalW == med?
{id:511, depth:9}
--> True:
Predict {'acc': 6, 'vgood': 8}
Max Class vgood
{id:1024,depth:10}
--> False:
Is SepalL == low?
{id:1023, depth:10}
--> True:
Is PetalL == 4?
{id:2048, depth:11}
--> True:
Predict {'acc': 1, 'vgood': 2}
Max Class vgood
{id:4098,depth:12}
--> False:
Predict {'acc': 2, 'vgood': 2}
Max Class vgood
{id:4097,depth:12}
--> False:
Is PetalL == more?
{id:2047, depth:11}
--> True:
Predict {'vgood': 2}
Max Class vgood
{id:4096,depth:12}
--> False:
Predict {'acc': 1, 'vgood': 1}
Max Class vgood
{id:4095,depth:12}
Accuracy on test = 0.9
Type here to search
10:53 PM
9/21/2018