**ID3 Algorithm:**

**ID3(*Examples, Target\_attribute, Attributes*)**

*Examples are the training examples. Target\_attribute is the attribute whose value is to be predicted by the tree. Attributes is a list of other attributes that may be tested by the learned decision tree. Returns a decision tree that correctly classifies the given Examples.*

* Create a *Root* node for the tree
* If all *Examples* are positive, Return the single-node tree *Root*, with label = +
* If all *Examples* are negative, Return the single-node tree *Root*, with label = -
* If *Attributes* is empty, Return the single-node tree *Root*, with label = most common value of *Target\_attribute* in *Examples*
* Otherwise Begin
  + - A ← the attribute from *Attributes* that best\* classifies *Examples*
    - The decision attribute for *Root* ←A
    - For each possible value, υi, of A,
      * Add a new tree branch below *Root*, corresponding to the test A = υi
      * Let *Examplesυi* ,be the subset of *Examples* that have value υi for A
        + If *Examplesυi*, is empty

Then below this new branch add a leaf node with label=most common value of *Target\_attribute* in *Examples*

Else below this new branch add the subtree ID3(*Examplesυi, Target\_attribute, Attributes–{A}*))

* end
* return Root

**Python Code:**

import ast  
import csv  
import sys  
import math  
import os  
def load\_csv\_to\_header\_data(filename):

path = os.path.normpath(os.getcwd() + filename)

fs = csv.reader(open(path))

all\_row = []

for r in fs:

all\_row.append(r)

headers = all\_row[0]  
 idx\_to\_name, name\_to\_idx = get\_header\_name\_to\_idx\_maps(headers)

data = {

'header': headers,

‘rows': all\_row[1:],

'name\_to\_idx': name\_to\_idx,

'idx\_to\_name': idx\_to\_name

}

return data

def get\_header\_name\_to\_idx\_maps(headers):

name\_to\_idx = {}  
 idx\_to\_name = {}  
 for i in range(0, len(headers)):

name\_to\_idx[headers[i]] = i

idx\_to\_name[i] = headers[i]

return idx\_to\_name, name\_to\_idx

def project\_columns(data, columns\_to\_project):

data\_h = list(data['header'])  
 data\_r = list(data['rows'])

all\_cols = list(range(0, len(data\_h)))

columns\_to\_project\_ix = [data['name\_to\_idx'][name] for name in columns\_to\_project]

columns\_to\_remove = [cidx for cidx in all\_cols if cidx not in columns\_to\_project\_ix]

for delc in sorted(columns\_to\_remove, reverse=True):

del data\_h[delc]  
 for r in data\_r:

del r[delc]

idx\_to\_name, name\_to\_idx = get\_header\_name\_to\_idx\_maps(data\_h)

return {'header': data\_h, 'rows': data\_r,

'name\_to\_idx': name\_to\_idx,

'idx\_to\_name': idx\_to\_name}

def get\_uniq\_values(data):

idx\_to\_name = data['idx\_to\_name']

idxs = idx\_to\_name.keys()

val\_map = {}  
 for idx in iter(idxs):

val\_map[idx\_to\_name[idx]] = set( )

for data\_row in data['rows']:  
 for idx in idx\_to\_name.keys():

att\_name = idx\_to\_name[idx]

val = data\_row[idx]  
 if val not in val\_map.keys():

val\_map[att\_name].add(val)

return val\_map

def get\_class\_labels(data, target\_attribute):  
 rows = data['rows']  
 col\_idx = data['name\_to\_idx'][target\_attribute]

labels = {}

for r in rows:  
 val = r[col\_idx]

if val in labels:

labels[val] = labels[val] + 1

else:

labels[val] = 1

return labels

def entropy(n, labels):

ent = 0

for label in labels.keys():

p\_x = labels[label] / n

ent += - p\_x \* math.log(p\_x, 2)

return ent

def partition\_data(data, group\_att):  
 partitions = {}  
 data\_rows = data['rows']  
 partition\_att\_idx = data['name\_to\_idx'][group\_att]

for row in data\_rows:

row\_val = row[partition\_att\_idx]

if row\_val not in partitions.keys():

partitions[row\_val] = { 'name\_to\_idx': data['name\_to\_idx'],

'idx\_to\_name': data['idx\_to\_name'],

'rows': list()

}

partitions[row\_val]['rows'].append(row)

return partitions

def avg\_entropy\_w\_partitions(data, splitting\_att, target\_attribute):

# find uniq values of splitting att  
 data\_rows = data['rows']  
 n = len(data\_rows)

partitions = partition\_data(data, splitting\_att)

avg\_ent = 0

for partition\_key in partitions.keys():

partitioned\_data = partitions[partition\_key]

partition\_n = len(partitioned\_data['rows'])

partition\_labels = get\_class\_labels(partitioned\_data, target\_attribute)

partition\_entropy = entropy(partition\_n, partition\_labels)  
 avg\_ent += partition\_n / n \* partition\_entropy

return avg\_ent, partitions

def most\_common\_label(labels):  
 mcl = max(labels, key=lambda k: labels[k])

return mcl

def id3(data, uniqs, remaining\_atts, target\_attribute):

labels = get\_class\_labels(data, target\_attribute)

node = {}

if len(labels.keys()) == 1:  
 node['label'] = next(iter(labels.keys()))

return node

if len(remaining\_atts) == 0:

node['label'] = most\_common\_label(labels)

return node

n = len(data['rows'])

ent = entropy(n, labels)

max\_info\_gain = None

max\_info\_gain\_att = None

max\_info\_gain\_partitions = None

for remaining\_att in remaining\_atts:  
 avg\_ent, partitions = avg\_entropy\_w\_partitions(data, remaining\_att, target\_attribute)

info\_gain = ent - avg\_ent  
 if max\_info\_gain is None or info\_gain > max\_info\_gain:

max\_info\_gain = info\_gain

max\_info\_gain\_att = remaining\_att

max\_info\_gain\_partitions = partitions

if max\_info\_gain is None:  
 node['label'] = most\_common\_label(labels)

return node

node['attribute'] = max\_info\_gain\_att

node['nodes'] = {}

remaining\_atts\_for\_subtrees = set(remaining\_atts)

remaining\_atts\_for\_subtrees.discard(max\_info\_gain\_att)

uniq\_att\_values = uniqs[max\_info\_gain\_att]

for att\_value in uniq\_att\_values:  
 if att\_value not in max\_info\_gain\_partitions.keys():

node['nodes'][att\_value] = {'label': most\_common\_label(labels)}

continue  
 partition = max\_info\_gain\_partitions[att\_value]  
 node['nodes'][att\_value] = id3(partition, uniqs, remaining\_atts\_for\_subtrees, target\_attribute)

return node

def load\_config(config\_file):  
 with open(config\_file, 'r') as myfile:

data = myfile.read().replace('\n', '')

return ast.literal\_eval(data)

def pretty\_print\_tree(root):

stack = []

rules = set()

def traverse(node, stack, rules):

if 'label' in node:

stack.append(' THEN ' + node['label'])

rules.add(''.join(stack))  
 stack.pop()

elif 'attribute' in node:  
 ifnd = 'IF ' if not stack else ' AND '

stack.append(ifnd + node['attribute'] + ' EQUALS ')

for subnode\_key in node['nodes']:

stack.append(subnode\_key)

traverse(node['nodes'][subnode\_key], stack, rules)

stack.pop()

stack.pop()

traverse(root, stack, rules)

print(os.linesep.join(rules))

def main():  
 argv ='tennis.cfg'

print("Command line args are {}: ".format(argv))

config = load\_config(argv)

data = load\_csv\_to\_header\_data(config['data\_file'])  
 data = project\_columns(data, config['data\_project\_columns'])

target\_attribute = config['target\_attribute']

remaining\_attributes = set(data['header'])

remaining\_attributes.remove(target\_attribute)

uniqs = get\_uniq\_values(data)  
 root = id3(data, uniqs, remaining\_attributes, target\_attribute)

pretty\_print\_tree(root)

if \_\_name\_\_ == "\_\_main\_\_": main()

Output:

