Assignment2_Final

October 5, 2018

1 Assignment 2: Practical data mining project - Implementing ID3 with an Interactive console

by Luke Crawford (12617306) and Jonathan Rau (13112750)

```
In [1]: import pandas as pd
        import torch
        import numpy as np
        from skimage import io, transform
        from math import log
        from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import LabelEncoder
        from decimal import Decimal
```

2 Introduction

We decided to implement the Iterative Dichotomiser 3 (ID3) algorithm to create decision tree classifiers. Decision Trees are generally easier to understand for laymen and can be stored in simple data structures. Our goal is to create a simple means to correctly identify if a given mushroom is poisonous or edible, as well as create a readable and simple design for our ID3 algorithm so others can easily understand our code.

2.1 Methodology

We split our work areas into two: 'Assignment2_Final' (this document) & 'Assignment2_Exploration'. In our Exploration, we explored the data, constructed our algorithm, compared to other methods, ran tests and attempted to build a dictionary for users to easily interact with the data. Assignment2_Final (this document) is the final product which we have developed, including this report, our most recent up to date version of the ID3 algorithm and an interactive console which users can use to build ID3 Decision trees and use their own datasets if they so wish.

2.2 The ID3 Algorithm

2.2.1 Data Structure

The decision tree is designed using the object-oriented programming methodology. Nodes are objects of the decision tree, where each node holds references to its children. As we never traverse the tree from a leaf to the top, we don't store the reference of the parent for each node, which creates spatial efficiency in the tree compared to other ID3 algorithms. Additionally, it is not necessary to retrieve a certain node without traversing the tree beginning from the root. Therefore, we did not need to use a datastructure like an array to store all nodes, which would provide random element reading with O(1) complexity. However, the lookup time complexity to get a specific child of a node is important to make a fast prediction. Hence, we store all children of a node in a dictionary as a member of the node with the branch value as a key.

2.3 Video Presentation

Please see the video presentation linked here: https://www.youtube.com/watch?v=1C7Dq4cbfsg
If there are any problems viewing the video presentation, contact luke.crawford@student.uts.edu.au

2.4 GitHub

Please see our GitHub page linked here for the resources used: https://github.com/badookey/MushroomID3

3 Decision Tree Classifier Source Code

3.1 Node class

```
In [2]: class DecisionTreeNode:
            def __init__(self):
                self.children= dict()

            def add_child(self, child_key, child_value):
                self.children[child_key] = child_value

            def get_children(self):
                return self.children

            def get_attribute(self):
                return self.attribute

            def set_attribute(self, attribute):
                self.attribute = attribute

            def set_label(self, label):
                self.label=label

            def get_label(self):
                return self.label

            def __str__(self, level=1):
                text=""
                if hasattr(self, 'label'):
```

```
            text += "leaf: label = {}".format(self.label)
        else:
            text += "split {}, descendants(".format(self.attribute)
            for value, child in self.children.items():
                text += "\n"+"   "*level+"branch = {}, child node:{}".format(value, chi

            text += ")"
        return text
```

4 Representation of the dataset

We are using the Pandas library to represent the dataset in collections of dataframes and series. For our usage, the biggest advantage of pandas is the ease of data column labelling. This allows us to easily access certain values and present the results in human readable format (no column indexes) without implementing a column-header list to map an index to a name.

Pandas provides a simple method to load the data from a csv file. Additionally, we are splitting the data into training and testing datasets using the train_test_split method from the sklearn package.

```
In [12]: def load_data(path, header_included_):
             if header_included_:
                 data = pd.read_csv(path)
             else:
                 data = pd.read_csv(path, header=None)
                 add_default = read_Bool("Do you want to add the default mushroom header? (Tru
                 if add_default:
                     assign_mushroom_header(data)


             return data

         def assign_mushroom_header(dataframe_):
             #Input column names from Mushroom Attributes.txt
             columns=['class','cap-shape','cap-surface','cap-color','bruises','odor','gill-atta

             #Rename columns according to their real attributes
             dataframe_.set_axis(columns, axis='columns', inplace=True)

         def split_data(dataframe_, test_size_, label_position):

             indexes = extract_attribute_set(dataframe_, label_position)
             #y is our target class
             #y = dataframe_.iloc[:,label_position]
             #indexes = [i for i in range(dataframe_.columns.size) if i != label_position]
             #x is our attributes
             #x = dataframe_.iloc[:,indexes]
             #x_train_, x_test_, y_train_, y_test_ = train_test_split(x,y,test_size = test_siz
             train_, test_ = train_test_split(dataframe_, test_size = test_size_)


             y_train_ = train_.iloc[:,label_position]
             y_test_ = test_.iloc[:,label_position]
             x_train_ = train_.loc[:,indexes]
             x_test_ = test_.loc[:,indexes]
             return y_train_, x_train_, y_test_, x_test_
```

```
In [4]: def extract_attribute_set(dataframe_, label_position_):
            indexes = [i for i in range(dataframe_.columns.size) if i != label_position_]
            x = dataframe_.iloc[:,indexes]
            return set(x.columns.values.tolist())
```

5 Performance of Statistical methods

As the dataset is divided into subsets to calculate the information gain very often, keeping the target value separate from the attribute values would lead to a large decrease in performance. The subset of the attribute values would have to be joined with the target values for each computation of the information gain for a split attribute. This would be a computational overkill due to it's high time and space complexity. Keeping the attribute and target values together enables us to create subsets of the data based on the attributes in one shot.

```
In [5]: def entropy(target_):
            h = 0
            for label_ in target_.unique():
                h += -((target_[target_==label_].size / target_.size)* log(target_[target_==lal
            return h

        def determine_split_attribute(data_, label_position_, attributes_):
            best_attribute_ = None
            best_gain_ = 0
            base_entropy_ = entropy(data_.iloc[:,label_position_])
            for attribute_ in attributes_:
                x_select_ = data_.loc[:,[attribute_, data_.columns[label_position_]]]
                information_gain_ = base_entropy_
                for value_ in x_select_.loc[:,attribute_].unique():
                    #split_ = pd.concat([x_select_[x_select_==value_], target_], axis=1, join=
                    split_entropy_ = entropy(x_select_.iloc[:,label_position_])
                    information_gain_ -= split_entropy_ * (x_select_.size / data_.size)

                if information_gain_ >= best_gain_:
                    best_attribute_ = attribute_
                    best_gain = information_gain_

            return best_attribute_
```

6 Structure of the ID3 algorithm

We implemented the ID3 algorithm in an recursive approach. The algorithm splits the data based on the attribute which leads to the highest information gain and performs a recursive call to create the child nodes. The Terminate condition for the recursion is reached when the subset contains one or less categories of target values.

```
In [6]: #param attributes_ should be a set of attributes
        #param target_ should be a series (like y_train)
        #param data_ should be a dataframe (like x_train)
        def build_decision_tree(data_, attributes_, label_position_):
            node_ = DecisionTreeNode()
            if data_.iloc[:,label_position_].unique().size==1:
                node_.set_label(data_.iloc[0,label_position_])
                return node_

            if len(attributes_) == 0:
                node_.set_label(data_.iloc[:,label_position_].value_counts().head(1).last_vali
                return node_

            else:
                split_attribute_ = determine_split_attribute(data_, label_position_, attributes
                print('splitting on: {}'.format(split_attribute_))
                node_.set_attribute(split_attribute_)
                split_select_ = data_.loc[:,split_attribute_]
                for split_value_ in split_select_.unique():
                    child_data_ = data_[data_[split_attribute_] == split_value_]
                    child_attributes_ = attributes_
                    child_attributes_.remove(split_attribute_)
                    #print('child_attributes: {}'.format(child_attributes_))
                    node_.add_child(split_value_, build_decision_tree(child_data_,child_attribu
                    #print('currend subtree: {}'.format(node_))

                    #as we are handling references, we have to add the attribut again
                    child_attributes_.add(split_attribute_)

            return node_
```

7 Predicting Data

Making predictions is done by traversing the tree, deciding which path to follow based on the attribute values until finding an end leaf. There is one edge case on the mushroom dataset, where an attribute's value was not present in the training set. We simply choose the children on a pseudorandom basis (which makes the ordering of the data structure not always the same on multiple run-throughs).

```
In [7]: #data should be a dataframe (like x_train)
        #root should be a a DecisionTreeNode (returned from build_decision_tree)
        def make_prediction(root, data):
            predictions = dict()
            for i, point in data.iterrows():
                current_node_ = root
                not_predicted = True
                while not_predicted:
                    if(hasattr(current_node_, 'label')):
                        predictions[i]=current_node_.get_label()
                        not_predicted = False
                    else:
                        split_value = point[current_node_.get_attribute()]
                        try:
                            current_node_ = current_node_.get_children()[split_value]
                        except KeyError:
                            current_node_ = list(current_node_.get_children().values())[0]
            #wrap the result in a Series to make the calculation of the accuracy easier
            result = pd.Series(predictions)
            return result
```

8 Creating an interface

We wrap the methods build_decision_tree and make_predictions in a class to provide an interface, which provides methods with a standard method signature.

```
In [8]: class DecisionTreeClassifier:
            def fit(self, label_, values_, label_position_):
                data_ = pd.concat([label_,values_], axis=1)
                self.tree=build_decision_tree(data_, extract_attribute_set(data_, label_positi

            def predict(self, data_):
                return make_prediction(self.tree, data_)

            def print_model(self):
                print(self.tree)
```

```
In [9]: def print_accuracy(real_values, predicted_values):
            stats= pd.crosstab(index = predicted_values, columns=real_values, margins=True, ro
            accuracy = np.sum(real_values == predicted_values) / predicted_values.size
            print(stats)
            print("The accuracy is: {}".format(accuracy))
```

8.1 Command Line Interface Functions

The following code is simply to provide a command line to end users in order to construct their own ID3 Decision Trees and provide their own datasets to use.

```
In [10]: def read_Bool(msg):
             text = input(msg)
```

```python
    if text == "True":
        text = True
    elif text == "False":
        text = False
    else:
        return read_Bool(msg)
    return text
def read_float(msg):
    try:
        d = float(input(msg))
    except:
        print("invalid input, please try again")
        read_float(msg)
    if 0<d<1:
        return d
    print("the relative size has to be between 0.0 and 1.0")
    read_float(msg)

def read_int(msg):
    try:
        i = int(input(msg))
    except:
        print("invalid input, please try again")
        read_int(msg)
    return i

def main():
    path = input("enter the absolute path of the dataset: ")
    header_included = read_Bool("is the header included in the dataset?: (True/False)
    data = load_data(path, header_included)
    split_size = read_float("enter the relative size of the test set: ")
    model = DecisionTreeClassifier()
    label_position = read_int("enter the position of the class label in the dataset: 
    y_train, x_train, y_test, x_test = split_data(data, split_size, label_position)
    print("Training the model")
    model.fit(y_train, x_train, label_position)
    print('your model: \n\n')
    model.print_model()
    print("\n\nevaluating on test set:")
    predict = model.predict(x_test)
    print('\n\n')
    print_accuracy(y_test, predict)
```

8.2 Command Line Interface

Simply type main() in order to access the command line.

Please note the example (In [13]), which has an accuracy of 99.87%

	Exploration (Neural Network)	Final (ID3)
Accuracy	93%	99.8%

```
In [13]: main()
```

```
enter the absolute path of the dataset: Data/Mushrooms.txt
is the header included in the dataset?: (True/False)False
Do you want to add the default mushroom header? (True/False)True
enter the relative size of the test set: 0.4
enter the position of the class label in the dataset: 0
Training the model
splitting on: cap-shape
splitting on: ring-type
splitting on: stalk-root
splitting on: habitat
splitting on: stalk-surface-above-ring
splitting on: gill-attachment
splitting on: stalk-surface-below-ring
splitting on: veil-color
splitting on: ring-number
splitting on: spore-print-color
splitting on: gill-size
splitting on: cap-color
splitting on: gill-color
splitting on: stalk-shape
splitting on: odor
splitting on: stalk-shape
splitting on: odor
splitting on: gill-color
splitting on: gill-size
splitting on: cap-color
splitting on: stalk-shape
splitting on: odor
splitting on: gill-size
splitting on: cap-color
splitting on: stalk-shape
splitting on: odor
splitting on: gill-size
splitting on: cap-color
splitting on: stalk-shape
splitting on: odor
splitting on: habitat
splitting on: stalk-root
splitting on: stalk-surface-above-ring
splitting on: habitat
splitting on: ring-type
splitting on: stalk-root
splitting on: habitat
splitting on: stalk-surface-above-ring
splitting on: gill-attachment
splitting on: stalk-surface-below-ring
splitting on: veil-color
splitting on: ring-number
splitting on: spore-print-color
splitting on: gill-size
splitting on: gill-size
splitting on: cap-color
splitting on: stalk-shape
splitting on: habitat
splitting on: stalk-surface-above-ring
splitting on: gill-attachment
splitting on: stalk-surface-below-ring
splitting on: veil-color
splitting on: ring-number
splitting on: spore-print-color
splitting on: gill-color
splitting on: gill-size
splitting on: cap-color
splitting on: stalk-shape
splitting on: odor
splitting on: gill-size
splitting on: cap-color
splitting on: stalk-shape
splitting on: odor
splitting on: gill-size
splitting on: cap-color
splitting on: stalk-shape
splitting on: odor
splitting on: gill-size
splitting on: cap-color
splitting on: gill-color
splitting on: stalk-shape
splitting on: odor
splitting on: stalk-shape
splitting on: odor
splitting on: stalk-root
splitting on: stalk-surface-above-ring
splitting on: gill-color
splitting on: ring-type
splitting on: stalk-root
splitting on: stalk-surface-above-ring
splitting on: habitat
splitting on: stalk-root
splitting on: habitat
splitting on: ring-type
splitting on: stalk-root
splitting on: stalk-surface-above-ring
your model:


split cap-shape, descendants(
   branch = f, child node:split ring-type, descendants(
      branch = p, child node:split stalk-root, descendants(
         branch = e, child node:split habitat, descendants(
            branch = u, child node:split stalk-surface-above-ring, descendants(
               branch = s, child node:split gill-attachment, descendants(
                  branch = f, child node:split stalk-surface-below-ring, descendants(
                     branch = s, child node:split veil-color, descendants(
                        branch = w, child node:split ring-number, descendants(
                           branch = o, child node:split spore-print-color, descendants(
                              branch = k, child node:split gill-size, descendants(
                                 branch = n, child node:split cap-color, descendants(
                                    branch = n, child node:split gill-color, descendants(
                                       branch = p, child node:split stalk-shape, descendants(
                                          branch = e, child node:split odor, descendants(
                                             branch = p, child node:leaf: label = p
                                             branch = n, child node:leaf: label = e))
                                       branch = k, child node:split stalk-shape, descendants(
                                          branch = e, child node:split odor, descendants(
                                             branch = n, child node:leaf: label = e
                                             branch = p, child node:leaf: label = p))
                                       branch = g, child node:leaf: label = e
                                       branch = w, child node:leaf: label = p
                                       branch = n, child node:leaf: label = p)
                                    branch = w, child node:leaf: label = p
                                    branch = g, child node:leaf: label = e))
                              branch = n, child node:split gill-color, descendants(
                                 branch = n, child node:split gill-size, descendants(
                                    branch = n, child node:split cap-color, descendants(
                                       branch = g, child node:leaf: label = e
                                       branch = w, child node:leaf: label = p
                                       branch = n, child node:split stalk-shape, descendants(
                                          branch = e, child node:split odor, descendants(
                                             branch = p, child node:leaf: label = p
                                             branch = n, child node:leaf: label = e))))
                                 branch = w, child node:leaf: label = p
                                 branch = g, child node:leaf: label = e
                                 branch = p, child node:split gill-size, descendants(
                                    branch = n, child node:split cap-color, descendants(
                        branch = n, child node:split stalk-shape, descendants(
                          branch = e, child node:split odor, descendants(
                            branch = p, child node:leaf: label = p
                            branch = n, child node:leaf: label = e))
                        branch = w, child node:leaf: label = p))
                    branch = k, child node:split gill-size, descendants(
                      branch = n, child node:split cap-color, descendants(
                        branch = w, child node:leaf: label = p
                        branch = n, child node:split stalk-shape, descendants(
                          branch = e, child node:split odor, descendants(
                            branch = n, child node:leaf: label = e
                            branch = p, child node:leaf: label = p))
                        branch = g, child node:leaf: label = e))))))))
      branch = g, child node:leaf: label = p)
    branch = b, child node:split habitat, descendants(
      branch = d, child node:leaf: label = e
      branch = p, child node:leaf: label = e
      branch = g, child node:leaf: label = p
      branch = u, child node:leaf: label = p
      branch = l, child node:leaf: label = p
      branch = m, child node:leaf: label = p)
    branch = ?, child node:leaf: label = e
    branch = r, child node:leaf: label = e)
  branch = e, child node:split stalk-root, descendants(
    branch = ?, child node:split stalk-surface-above-ring, descendants(
      branch = s, child node:split habitat, descendants(
        branch = d, child node:leaf: label = p
        branch = w, child node:leaf: label = e
        branch = p, child node:leaf: label = p
        branch = l, child node:leaf: label = p)
      branch = k, child node:leaf: label = p)
    branch = e, child node:leaf: label = e
    branch = b, child node:leaf: label = e
    branch = c, child node:leaf: label = p)
  branch = l, child node:leaf: label = p
  branch = f, child node:leaf: label = e
  branch = n, child node:leaf: label = p)
branch = x, child node:split ring-type, descendants(
  branch = p, child node:split stalk-root, descendants(
    branch = b, child node:split habitat, descendants(
      branch = d, child node:split stalk-surface-above-ring, descendants(
        branch = s, child node:split gill-attachment, descendants(
          branch = f, child node:split stalk-surface-below-ring, descendants(
            branch = s, child node:split veil-color, descendants(
              branch = w, child node:split ring-number, descendants(
                branch = o, child node:split spore-print-color, descendants(
                  branch = k, child node:split gill-size, descendants(
                    branch = b, child node:leaf: label = e
```

```
                        branch = n, child node:leaf: label = p)
                     branch = n, child node:split gill-size, descendants(
                        branch = b, child node:leaf: label = e
                        branch = n, child node:split cap-color, descendants(
                           branch = w, child node:split stalk-shape, descendants(
                              branch = e, child node:leaf: label = p
                              branch = t, child node:leaf: label = e)
                           branch = g, child node:leaf: label = p
                           branch = p, child node:leaf: label = p
                           branch = y, child node:leaf: label = e))
                     branch = u, child node:leaf: label = e)))))
      branch = y, child node:leaf: label = e)
   branch = g, child node:leaf: label = p
   branch = u, child node:leaf: label = p
   branch = p, child node:leaf: label = e)
branch = c, child node:leaf: label = e
branch = e, child node:split habitat, descendants(
   branch = u, child node:split stalk-surface-above-ring, descendants(
      branch = s, child node:split gill-attachment, descendants(
         branch = f, child node:split stalk-surface-below-ring, descendants(
            branch = s, child node:split veil-color, descendants(
               branch = w, child node:split ring-number, descendants(
                  branch = o, child node:split spore-print-color, descendants(
                     branch = n, child node:split gill-color, descendants(
                        branch = n, child node:split gill-size, descendants(
                           branch = n, child node:split cap-color, descendants(
                              branch = w, child node:leaf: label = p
                              branch = n, child node:split stalk-shape, descendants(
                                 branch = e, child node:split odor, descendants(
                                    branch = n, child node:leaf: label = e
                                    branch = p, child node:leaf: label = p))
                              branch = g, child node:leaf: label = e))
                        branch = p, child node:split gill-size, descendants(
                           branch = n, child node:split cap-color, descendants(
                              branch = g, child node:leaf: label = e
                              branch = n, child node:split stalk-shape, descendants(
                                 branch = e, child node:split odor, descendants(
                                    branch = n, child node:leaf: label = e
                                    branch = p, child node:leaf: label = p))
                              branch = w, child node:leaf: label = p))
                        branch = g, child node:leaf: label = e
                        branch = k, child node:split gill-size, descendants(
                           branch = n, child node:split cap-color, descendants(
                              branch = n, child node:split stalk-shape, descendants(
                                 branch = e, child node:split odor, descendants(
                                    branch = p, child node:leaf: label = p
                                    branch = n, child node:leaf: label = e))
                              branch = g, child node:leaf: label = e
```

```
                          branch = w, child node:leaf: label = p))
                      branch = w, child node:leaf: label = p)
                    branch = k, child node:split gill-size, descendants(
                      branch = n, child node:split cap-color, descendants(
                        branch = g, child node:leaf: label = e
                        branch = w, child node:leaf: label = p
                        branch = n, child node:split gill-color, descendants(
                          branch = p, child node:split stalk-shape, descendants(
                            branch = e, child node:split odor, descendants(
                              branch = p, child node:leaf: label = p
                              branch = n, child node:leaf: label = e))
                          branch = w, child node:leaf: label = p
                          branch = n, child node:split stalk-shape, descendants(
                            branch = e, child node:split odor, descendants(
                              branch = p, child node:leaf: label = p
                              branch = n, child node:leaf: label = e))
                          branch = g, child node:leaf: label = e
                          branch = k, child node:leaf: label = p)))))))))
        branch = g, child node:leaf: label = p)
      branch = ?, child node:leaf: label = e
      branch = r, child node:leaf: label = e)
    branch = e, child node:split stalk-root, descendants(
      branch = ?, child node:split stalk-surface-above-ring, descendants(
        branch = k, child node:leaf: label = p
        branch = s, child node:split gill-color, descendants(
          branch = b, child node:leaf: label = p
          branch = e, child node:leaf: label = e
          branch = w, child node:leaf: label = e))
      branch = e, child node:leaf: label = e
      branch = b, child node:leaf: label = e)
    branch = l, child node:leaf: label = p
    branch = f, child node:leaf: label = e
    branch = n, child node:leaf: label = p)
  branch = k, child node:split ring-type, descendants(
    branch = e, child node:split stalk-root, descendants(
      branch = ?, child node:split stalk-surface-above-ring, descendants(
        branch = k, child node:leaf: label = p
        branch = s, child node:split habitat, descendants(
          branch = l, child node:leaf: label = p
          branch = p, child node:leaf: label = p
          branch = d, child node:leaf: label = p
          branch = w, child node:leaf: label = e))
      branch = b, child node:leaf: label = e
      branch = c, child node:leaf: label = p)
    branch = p, child node:split stalk-root, descendants(
      branch = b, child node:split habitat, descendants(
        branch = d, child node:leaf: label = e
        branch = l, child node:leaf: label = p)
```

```
         branch = ?, child node:leaf: label = e)
      branch = n, child node:leaf: label = p)
   branch = b, child node:split ring-type, descendants(
      branch = p, child node:split stalk-root, descendants(
         branch = c, child node:leaf: label = e
         branch = ?, child node:leaf: label = e
         branch = b, child node:split stalk-surface-above-ring, descendants(
            branch = s, child node:leaf: label = p
            branch = y, child node:leaf: label = e))
      branch = e, child node:leaf: label = p)
   branch = s, child node:leaf: label = e
   branch = c, child node:leaf: label = p)


evaluating on test set:



actual        e     p   All
predicted
e          1713     0  1713
p             4  1533  1537
All        1717  1533  3250
The accuracy is: 0.9987692307692307
```

```
In [ ]: main()
```

8.3 Conclusion

In conclusion, the above ID3 implementation is a concise and simple interactive program which allows anyone to classify data efficiently and easily. The design of the code and command-line make it so anyone can modify the algorithm to their particular needs with ease. This is due to the object oriented design of the code and the simple, but powerful commands of the interface. Anyone can use any