MSc Methodology of Biomedical Research,

Biostatistics and Clinical Bioinformatics

University of Thessaly

Faculty of Medicine

Master thesis

Develop a Software in Python for Classification, based on Classification And Regression Trees (CART)

Ανάπτυξη Λογισμικού σε Python για Ταξινόμηση, βασισμένο στα Δέντρα Ταξινόμησης και Παλινδρόμησης (CART)

Emmanouilidou Eleni

Three-member Advisory Committee:
Dr Axel Kowald (Supervisor)
Dr Elias Zintzaras
Dr Ioannis Stefanidis

Academic year
2021

Abstract

Classification is an important part of data analysis and Classification and Regression Trees (CARTs) are a popular Machine Learning method for such a task. In this work a Classification software written in Python programming language is presented, based on CARTs. The algorithm splits the nodes in order to minimize the mixing of the classes in them. In order to split a node it calculates the information gain of all the possible splits and decides to use the one with the highest score. This function is executed recursively until the Tree is complete. Then the software offers textual and graphical visualization of the Classification Tree. The textual form is produced with Python's built-in functions, while the graph format with the Graphviz software in connection with the graphviz package. The software was developed using the PyCharm IDE, and the Iris flower data set was used in the program's execution. CARTs have evolved a lot during years and their potential is great. This algorithm's capabilities can be augmented in various ways that were impossible to include in a thesis of this a length.

Key words:

programming, Python, CART, classification, decision trees, Classification and Regression Trees, Graphviz

Περίληψη

Η Ταξινόμηση αποτελεί σημαντικό κομμάτι της ανάλυσης δεδομένων και τα Δέντρα Ταξινόμησης και Παλινδρόμησης (CARTs) είναι μια δημοφιλής μέθοδος Μηχανικής Μάθησης για τη διεκπεραίωση μιας τέτοιας διαδικασίας. Σε αυτή την εργασία παρουσιάζεται ένα λογισμικό Ταξινόμησης γραμμένο στη γλώσσα προγραμματισμού Python βασισμένο στα CARTs. Ο αλγόριθμος διαχωρίζει του κόμβους ώστε σε κάθε χώρισμα να μειώνεται η νοθεία των κλάσεων. Για να εκτελέσει τον ιδανικό διαχωρισμό υπολογίζει τον δείκτη κέρδους πληροφορίας (information gain) από κάθε πιθανό διαχωρισμό και επιλέγει αυτόν που πετυχαίνει τη μεγαλύτερη τιμή για τον δείκτη. Αυτή η συνάρτηση καλείται αναδρομικά έως ότου το δέντρο ολοκληρωθεί. Το λογισμικό προσφέρει κειμενική και γραφική απεικόνιση του Δέντρου Ταξινόμησης στις οποίες περιλαμβάνεται όλη η διαδικασία που ακολουθήθηκε. Η κειμενική μορφή προκύπτει με χρήση εντολών της Python, ενώ η γραφική με χρήση του λογισμικού Graphviz σε συνδυασμό με το πακέτο graphviz. Το λογισμικό αναπτύχθηκε με χρήση του Ολοκληρωμένο Περιβάλλον Ανάπτυξης PyCharm, και το Iris flower data set χρησιμοποιήθηκε ως σύνολο δεδομένων για την εκτέλεση του προγράμματος. Τα CARTs έχουν εξελιχθεί πολύ μέσα στα χρόνια και οι δυνατότητές τους είναι μεγάλες. Οι δυνατότητες του αλγορίθμου μας μπορούν να αυξηθούν στο μέλλον με ποικίλους τρόπους που είναι αδύνατο να συμπεριληφθούν σε μια εργασία αντίστοιχου μεγέθους.

Λέξεις Κλειδιά:

προγραμματισμός, Python, CART, ταξινόμηση, δέντρα λήψης αποφάσεων, δέντρα ταξινόμησης και παλινδρόμησης, Graphviz

CONTENTS

Introduction

Machine learning (ML) is the field of creating algorithms which make predictions and decisions based on data and the knowledge derived from them (Podgorelec, Kokol, Stiglic, & Rozman, 2002). This thesis is focused on the development of a ML algorithm for Classification of datasets using the Python programming language. Its results are presented using the Iris flower dataset as an example.

Classification and Regression Trees (CARTs)

Classification and Regression Trees are ML prediction models for classification. They use sets of data which include i) predictor variables and ii) the class variable of each instance. The difference between Classification Trees and Regression Trees is that the former analyze a finite number of dependent unordered variables, while in the latter the values have to be continuous or ordered. In this thesis the focus is on Classification Trees. The Tree construction begins with a parent node including the training data based on which the Tree will be built. The data space is divided recursively into child nodes, with each split fitting a prediction model. The result is a procedure that can be represented as a decision tree and aims to generate child nodes that have the minimum mixing of classes (Fig.1). The nodes which do not split anymore are named leaf nodes and classify the data, while the ones that split are called decision nodes. The training data do not usually include the whole data set, because part of it is used as testing data set in order to evaluate the goodness of fit of the Tree. (Loh, 2011)

Classification trees have various advantages. First of all they are white-box algorithms, which means that all the steps of the splitting procedure are revealed. Consequently they do not only produce classifiers but also allow insight and understanding of the predictive structure. Moreover, their representation as a decision tree is applicable to any number of mixed-type variables. As for their comprehensibility, their graphical form is conceptually clear (Breiman, Friedman, Stone, & Olshen, 1984).

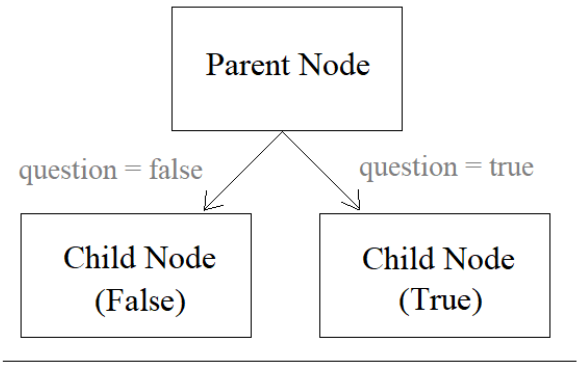


Fig. 1. Example of a decision tree

Attention must be paid to data preparation as CART models do not support missing values, and unbalanced sets produce biased results.

Python Programming Language

The process of Programming includes writing commands that are executed by the computer in order to perform a task. Python is a high-level programming language used to write such instructions. This means that the natural language elements in the source code of a Python program are converted into bytecode that is then executed by the Python virtual machine. It is dynamically typed and is characterized by sensitivity to the difference between uppercase and lowercase. The Python interpreter is freely distributed with its standard library of packages under the Python Software Foundation License, and is very popular because of its accessibility and easy-to-learn syntax. Additionally to the standard library it permits the use of external packages and modules that can be downloaded, a fact that encourages code reuse and constantly augments its capabilities. (Python Software Foundation)

Many Decision Tree learning algorithms have already been developed among which: ID3, C4.5, CART, CHAID, QUEST, GUIDE, CRUISE, and CTREE (Singh & Gupta, 2014). This project is based on the CART algorithm developed in 1984 by Breiman et al. (Breiman, Friedman, Stone, & Olshen, 1984). Those popular algorithms have various capabilities that were impossible to develop in the short length of this thesis. Yet here a comprehensive algorithm is included which can play an important role to understand the methodology of building a tree and extracting a graphical form of it.

Materials and Methods

The computer used for the project runs on the Windows 10 Pro 64-bit operating system. The software was written in Python 3.9.0.

Integrated Development Environment

Programming can be more user-friendly by using an Integrated Development Environment (IDE). IDEs facilitate software development as they combine in a single software application at least a source code editor, building executables, and debugging. In this project the Pycharm 2021.2 Community Edition IDE was used.

Classification method

The building of a Classification and Regression Tree begins with a node that contains the training dataset. In our software the dataset has to have the form of a list of lists. Each sublist includes the values of the variables (numeric and/or string) and the class name (string) of an instance, all separated by commas. The class name is always placed last in the sublist. Two sublists of our list can be seen as an example: [5.1, 3.5, 1.4, 0.2, 'Iris-setosa'], [4.9, 3.0, 1.4, 0.2, 'Iris-setosa'].

Then the first main step of the algorithm is to look for the best criterion to split the parent node. If each attribute is considered as a column with values ['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm'], each value of each column is used to ask a question of the following form:

- For numeric variables:
  Is the value of the subject's attribute ≥ the value we are testing?
- For string variables:
  Is the value of the subject's attribute the same as the value we are testing?

for each subject of the dataset.

Each time the subjects are split in two child nodes based on the result of the question (True/False), and the success of each split is calculated. In Classification Trees as success is considered the minimization of the impurity (mixing) in the child nodes. In our CART method the Gini impurity measurement is used. At each split the Gini impurity is calculated for the parent node and the two child nodes separately:

$$I_G = 1 - \sum_{j=1}^{c} (p_j{}^2),$$

Where I_G is the Gini Impurity and p is the proportion of the samples that belong to class c for a particular node.

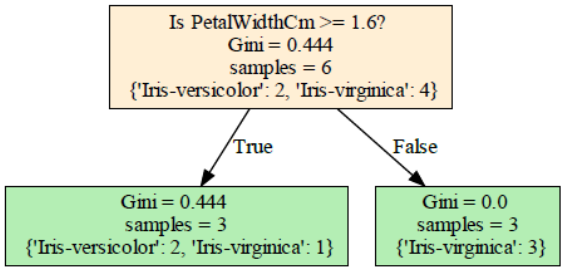


Fig. 2. Part of the Iris flower data set Classification Tree

For the nodes in Fig. 2 the impurity is:

- For the Parent Node:

$I_{G1} = 1 - [(\frac{2}{6})^2 + (\frac{4}{6})^2] = 0.444$

- For the true Child Node:

$I_{G2} = 1 - [(\frac{2}{3})^2 + (\frac{1}{3})^2] = 0.444$

- For the false Child Node:

$I_{G3} = 1 - (\frac{3}{3})^2 = 0$

In the example above, "samples" equals to the number of instances of the node, 'Iris-versicolor' and 'Iris-virginica' are names of different classes. I_{G3} is equal to 0 because the false Child Node has no mixing of classes. All samples belong to the 'Iris-virginica' class.

The next step is to calculate the impurity decrease for each potential split. For this reason the Information gain is calculated in the following way:

$$\mathrm{IG}(\mathrm{D}_p,\mathrm{f}) = I_G(D_p) - \frac{N_{true}}{N} I_G(D_{true}) - \frac{N_{false}}{N} I_G(D_{false})$$

Where f is the question being asked, D_p is the dataset of the parent node, D_{true} is the dataset of the True child node and D_{false} in the dataset of the False child node. I_G is the Gini Impurity of each node and N is the number of instances in each node (N_{true}: in the True node, N_{false}: in the False node, N: in the parent node).

For the node in Fig. 2, the information gain is:

$$\mathrm{IG}(\mathrm{D}_p,\mathrm{f}) = I_G(D_p) - \frac{N_{true}}{N} I_G(D_{true}) - \frac{N_{false}}{N} I_{G3}(D_{false}) =$$

$$= 0.444 - \left(\frac{3}{6}\right) \times 0.444 - \left(\frac{3}{6}\right) \times 0 =$$

$$= 0.222$$

Checking all the possible decisions one by one, each time the algorithm finds a higher information gain it saves it in a variable named max_info_gain, replacing the previous

value. After asking all the possible questions the max_info_gain and its corresponding questions are kept, and the parent node finally splits according to it.

Then the algorithm recursively splits the child nodes following the same algorithm until the splitting stops in two cases. The first case is when no information gain occurs from the split (max_info_gain = 0) and the decision node does not split, it becomes a leaf node. The second case is when the depth of the tree reaches a prefixed value and has to stop. The depth of the tree is the number of splits a Tree can make before reaching a leaf node and in our algorithm the limit was set to 4. Setting a depth limit before the Tree fully grows is called Prepruning. Pruning can also happen after a Tree is fully grown by subtracting the deepest nodes of it. Pruning and Prepruning help the algorithm because they produce Trees that do not overfit to the training sample and can generalize more efficiently.

Graph Visualization

The Classification Tree of the software is visualized in two forms. The first form is textual using Pythons' built-in functions.

The second form is graph visualization using the Graphviz software. Graphviz is an open source software that can be downloaded under the The Common Public License. The graphviz package was installed and imported in the algorithm. Graphviz takes descriptions of graphs in simple text language and makes diagrams in various formats, such as .png or .pdf. It also allows the choice of different colours, fonts, shapes, node layouts etc for the extracted file (The Graphviz Authors). In our software a directed graph (Digraph) is created, which means that the nodes of the Tree are connected with arrows pointing from the parent node to the child node. In order for a graph to be created, Graphviz needs the description of each node and each edge that connects the nodes, along with their attributes.

The Iris flower data set

The Iris flower data set is a data set of 150 flowers belonging to three species of the iris genus: Iris setosa, Iris versicolor and Iris virginica. It includes the measurements of the sepal length, sepal width, petal length and petal width of 50 flowers of each species and was introduced as a problem of taxonomy to be addressed with linear discriminant analysis (Fisher, 1936). The dataset is open sourced, with class imbalances and no missing values and is conceptually understandable, so it is very popular as a test sample for many classification techniques in machine learning.

Data reading and preparation

The Iris flower data set was downloaded in as a Comma Separated Values (csv) file in our software's directory. Using the csv package from the Python Standard Library, the file was read as a list of lists named 'data', but all the values were considered to be strings.

```
with open('Iris_dataset.csv', newline='') as f:
    reader = csv.reader(f)
    data = list(reader)
```

In order to give the appropriate format to the file, the first value of each list was deleted as it was the id of the flower and it was not useful.

```
for list in data:  # deletes the Id of the instances
    del list[0]
```

Then a new list was created including only the first list of the file's lists. This new list held the headers of the columns. The headers were then deleted from the initial list because they would interfere with the algorithm.

```
headers = data[0]  # list of columns' headers
del data[0]
```

The iris data set includes the class names and 4 attributes for each flower which are numeric. In order to change the string values of the attributes to numeric, the class names were separated in another list of lists named 'target_values' and deleted from the initial list.

```
target_values = []  # separate the target values because they are str
for list in data:
    target_values.append(list[-1])
    del (list[-1])
```

All the values of the attributes were appended as floats in another new list of lists named 'iris_data'.

```
iris_data = []
for list in data:  # str to num
    iris_data_list = []
    for i in list:
        iris_data_list.append(float(i))
    iris_data.append(iris_data_list)
```

Finally the class names were reappended in the list with the numeric values and the data set was ready for use.

```
for idx, i in enumerate(target_values):  # iris_data appends target
values again
    iris_data[idx].append(i)
```

Results

Software

The algorithm that was developed is the following. First there is the import statement for the packages used. Then the Functions and the Classes are defined:

```
import csv
import random
import graphviz

# Classification Definitions

def get_unique_numbers(numbers):
    """It takes a group of numbers and creates a list of them with no
duplicate values."""
    list_of_unique_numbers = []
    unique_numbers = set(numbers)
    for number in unique_numbers:
        list_of_unique_numbers.append(number)
    return list_of_unique_numbers


def numeric_data(value):
    """Checks if a value is numeric"""
    return isinstance(value, int) or isinstance(value, float)


def type_counts(rows):
    """Counts the number of instances with each target value in the
node"""
    counts = {}
    for row in rows:
        target_val = row[-1]
        if target_val not in counts:
            counts[target_val] = 0
        counts[target_val] += 1
    return counts


def instances_split(rows, column, unique_vals):
    """Asks all the possible questions.
        Each time splits the rows of a node in true or false based on
the question.
        If true it appends it to the 'true_group',
        else to the 'false_group'"""

    true_group, false_group = [], []  # The instances will be split
in two arrays

    for row in rows:
        if numeric_data(row[column]):  # question if numeric
            criterion = row[column] >= unique_vals

        else:  # question if str
            criterion = row[column] == unique_vals

        if criterion:
            true_group.append(row)
        else:
```

```
            false_group.append(row)
    return true_group, false_group


def gini_impurity(rows):
    """Calculates the GI = Gini impurity"""
    gini = 1
    counts = type_counts(rows)
    for value in counts:  # checks each target value
        gini -= (counts[value] / len(rows)) ** 2
    return gini


def information_gain(left_node, right_node, parent_node):
    """Information Gain (IG) is the reduction of entropy by the
dataset transformation.
    IG = GI(parent node) -
    GI(left child node) weighted -
    GI(right child node) weighted"""
    weight_left = len(left_node) / (len(left_node) + len(right_node))
    weight_right = len(right_node) / (len(left_node) +
len(right_node))
    return gini_impurity(parent_node) - gini_impurity(left_node) *
weight_left - gini_impurity(
        right_node) * weight_right


def unique_values_for_questions(parent_node, headers):
    """Returns a dictionary with keys: the header names, and items:
the unique values of each column."""
    unique_vals = {}
    for i in range(len(parent_node[0]) - 1):  # i in number of
characteristics
        vals = []
        for row in parent_node:
            vals.append(row[i])
        unique_vals[headers[i]] = get_unique_numbers(vals)
    # print("The dictionary of unique values for each column is:",
unique_vals)
    return unique_vals


def ask_best_question(parent_node):
    """Asks all the possible questions. Then finds the question with
the highest info gain."""

    max_info_gain = 0
    ideal_key_num = 0
    ideal_val = 0
    ideal_key = ''
    question = ''
    unique_vals = unique_values_for_questions(parent_node, headers)

    for idx, key in enumerate(unique_vals.keys()):  # checking column
by column: idx = 0, 1,... and key = column name
        for i in range(len(unique_vals[key])):  # check each unique
value of the key

            true_group, false_group = instances_split(parent_node,
idx, unique_vals[key][i])
```

```
            info_gain = information_gain(true_group, false_group,
parent_node)
            # print(info_gain)

            if info_gain > max_info_gain:
                max_info_gain = info_gain
                ideal_key = key
                ideal_key_num = idx
                ideal_val = unique_vals[key][i]
                question = 'Is %s %s %s?' % (ideal_key, '>=',
str(ideal_val))
    # print("The best question is", ideal_key, ">=", ideal_val, "?",
"(Information gain =", max_info_gain, ")")
    return max_info_gain, ideal_key, ideal_key_num, ideal_val,
question


class LeafNode:
    """A leaf node is not split anymore. It classifies data.
        It holds a dictionary with key: the unique target values of
the dataset
        and value: the number of instances of each target value in
the node"""

    def __init__(self, rows):
        self.predictions = type_counts(rows)
        self.gini = gini_impurity(rows)
        self.node_name = str(random.random())
        self.len = len(rows)


class DecisionNode:
    """A decision node is split based on another question.
        It holds a reference to the question and the child nodes that
occur from it."""

    def __init__(self, rows, question, true_branch, false_branch):
        self.gini = gini_impurity(rows)
        self.predictions = type_counts(rows)
        self.question = question
        self.true_branch = true_branch
        self.false_branch = false_branch
        self.node_name = str(random.random())
        self.len = len(rows)

def build_tree(parent_node, depth):
    """It uses a recursive function to create the whole
classification tree"""
    global name
    name += 1

    if depth < 4:
        depth += 1
    else:
        return LeafNode(parent_node)

    max_info_gain, ideal_key, ideal_key_num, ideal_val, question =
ask_best_question(
        parent_node)  # Best question for the branch

    if max_info_gain == 0:  # If information gain = 0 the instances
```

```
do not split further, we have a leaf node
        return LeafNode(parent_node)

    else:
        true_group, false_group = instances_split(parent_node,
ideal_key_num,
                                                  ideal_val)  # If
info gain != 0 -> split (decision node)

        true_branch = build_tree(true_group,
                                 depth=depth)  # the build_tree
function is recursively called to split the true_branch
        false_branch = build_tree(
            false_group, depth=depth)  # the build_tree function is
recursively called to split the false_branch

        return DecisionNode(parent_node, question, true_branch,
false_branch)

# End of Classification Definitions
```

The next step is to define the functions of the visualization part of the algorithm (textual and graphical):

```
# Visualization Definitions


def print_tree(node, spacing="   "):  # textual
    if isinstance(node, LeafNode):
        print(spacing + 'Predict', node.predictions)
        print(spacing + 'The Gini impurity is ' + str(node.gini))
        return

    if isinstance(node, DecisionNode):
        print(spacing + str(node.question))
        print(spacing + 'Predict', node.predictions)
        print(spacing + 'The Gini impurity is ' + str(node.gini))

        print(spacing + '--> True:')
        print_tree(node.true_branch, spacing + " ")

        print(spacing + '--> False')
        print_tree(node.false_branch, spacing + " ")


def graph_tree(tree, parent_node_name, graph, edge_label):
    """Produces a graph of the Classification Tree using Graphviz.
        The tree.predictions are used as names for the nodes
        because at least with this dataset they never repeat
themselves."""

    global k
    k += 1

    if isinstance(tree, LeafNode):
        graph.node(name=tree.node_name,
                   label='%s %.5s \n %s %s \n %s'%
```

```
                        ('Gini =', tree.gini, 'samples =', tree.len,
tree.predictions),
                    shape='box', style='filled',
fillcolor='darkseagreen2')
        graph.edge(parent_node_name, tree.node_name,
label=edge_label)

        return

    if isinstance(tree, DecisionNode):
        graph.node(name=tree.node_name,
                   label='%s \n %s %.5s \n %s %s \n %s'%
                         (tree.question, 'Gini =', tree.gini,
'samples =', tree.len, tree.predictions),
                   shape='box', style='filled',
fillcolor='papayawhip')
        if k > 1:
            graph.edge(parent_node_name, tree.node_name,
label=edge_label)
        parent_node_name = tree.node_name

        graph_tree(tree.true_branch, parent_node_name, graph,
edge_label='True')
        graph_tree(tree.false_branch, parent_node_name, graph,
edge_label='False')
        return

# End of Visualization Definitions
```

After all the Functions and Classes are defined, the dataset has to be read and prepared to be used in the correct format. The Iris flower data set was saved as a .csv file in the same directory as the python project. Then, the preparation of the dataset is the following:

```
# Opening and preparing the dataset


with open('Iris_dataset.csv', newline='') as f:
    reader = csv.reader(f)
    data = list(reader)

for list in data:  # deletes the Id of the instances
    del list[0]

headers = data[0]  # list of columns' headers
del data[0]


target_values = []  # separate the target values because they are str
for list in data:
    target_values.append(list[-1])
    del (list[-1])

iris_data = []
for list in data:  # str to num
    iris_data_list = []
```

```
    for i in list:
        iris_data_list.append(float(i))
    iris_data.append(iris_data_list)

for idx, i in enumerate(target_values):  # iris_data appends target
values again
    iris_data[idx].append(i)


# End of opening and preparing dataset file
```

Finally there is the part of the software which analyzes the dataset, trains the Decision Tree and visualizes the result using the functions and classes defined at the beginning:

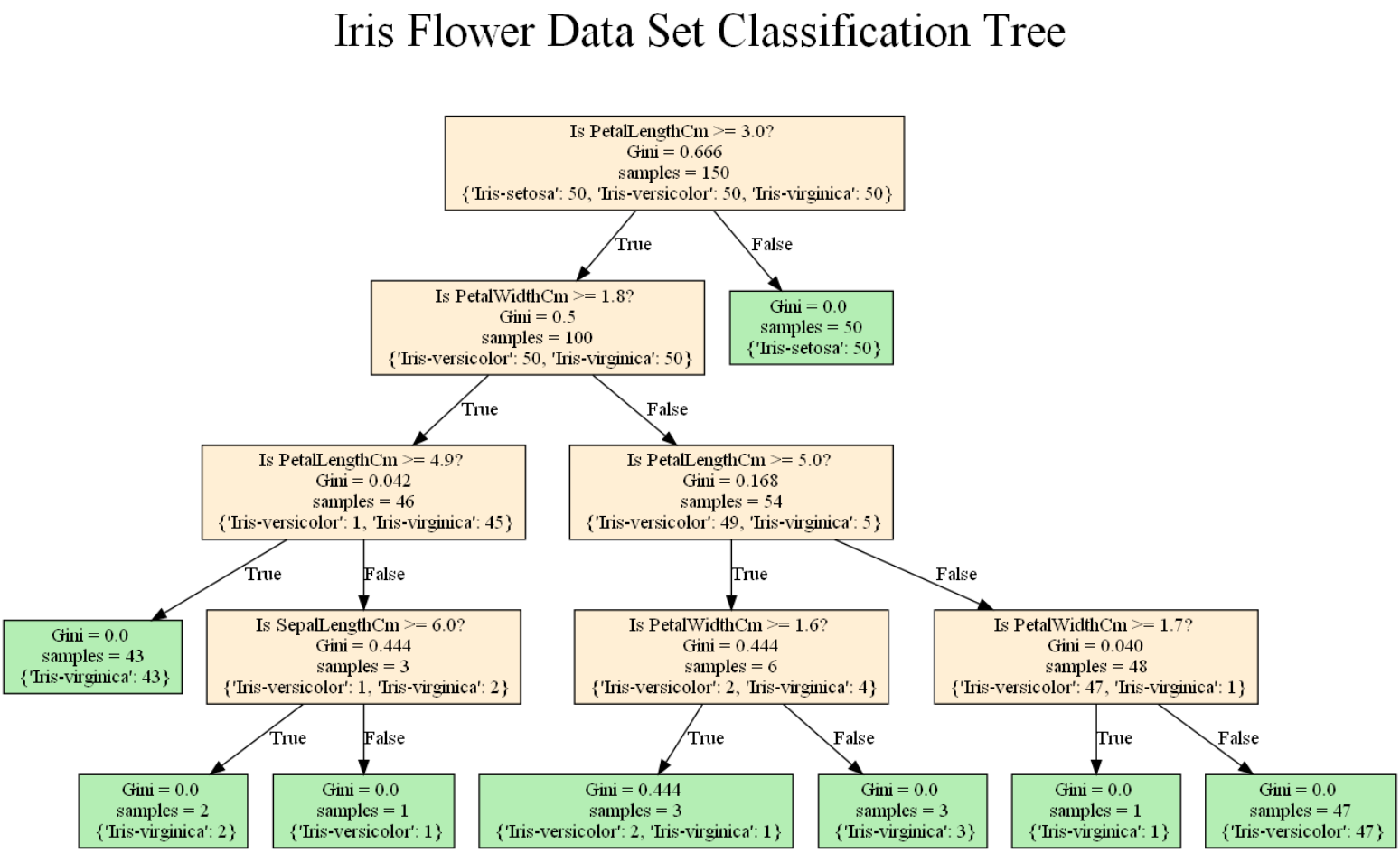
```
name = 0
depth = 0
k = 0
parent_node_name = ''

my_tree = build_tree(iris_data, depth=depth)
print_tree(my_tree)

g = graphviz.Digraph(name='Classification Tree', format='png')
g.attr(label='Iris Flower Data Set Classification Tree\n ',
fontsize='34', labelloc='t')
graph_tree(my_tree, parent_node_name, g, edge_label='')

print(g.source)
g.view()
```

Visualization

When we run the program the textual visualization output is the following. Here the colors of the splitting questions and the True/False answers are different than the PyCharm output for optical reasons. Each decision node contains i) the question for its split, ii) the number of instances of each class that the node contains, ii) its Gini impurity. If the Node has no printed question it means that it is a leaf node.

```
Is PetalLengthCm >= 3.0?
Predict {'Iris-setosa': 50, 'Iris-versicolor': 50, 'Iris-virginica': 50}
The Gini impurity is 0.6666666666666665
--> True:
 Is PetalWidthCm >= 1.8?
 Predict {'Iris-versicolor': 50, 'Iris-virginica': 50}
 The Gini impurity is 0.5
 --> True:
  Is PetalLengthCm >= 4.9?
  Predict {'Iris-versicolor': 1, 'Iris-virginica': 45}
  The Gini impurity is 0.04253308128544431
  --> True:
   Predict {'Iris-virginica': 43}
   The Gini impurity is 0.0
  --> False
   Is SepalLengthCm >= 6.0?
   Predict {'Iris-versicolor': 1, 'Iris-virginica': 2}
   The Gini impurity is 0.4444444444444444
   --> True:
    Predict {'Iris-virginica': 2}
    The Gini impurity is 0.0
   --> False
    Predict {'Iris-versicolor': 1}
    The Gini impurity is 0.0
 --> False
  Is PetalLengthCm >= 5.0?
  Predict {'Iris-versicolor': 49, 'Iris-virginica': 5}
  The Gini impurity is 0.1680384087791495
  --> True:
   Is PetalWidthCm >= 1.6?
   Predict {'Iris-versicolor': 2, 'Iris-virginica': 4}
   The Gini impurity is 0.4444444444444444
   --> True:
    Predict {'Iris-versicolor': 2, 'Iris-virginica': 1}
    The Gini impurity is 0.4444444444444445
   --> False
    Predict {'Iris-virginica': 3}
    The Gini impurity is 0.0
  --> False
   Is PetalWidthCm >= 1.7?
   Predict {'Iris-versicolor': 47, 'Iris-virginica': 1}
   The Gini impurity is 0.040798611111111174
   --> True:
    Predict {'Iris-virginica': 1}
    The Gini impurity is 0.0
   --> False
    Predict {'Iris-versicolor': 47}
    The Gini impurity is 0.0
--> False
 Predict {'Iris-setosa': 50}
 The Gini impurity is 0.0
```

Using Graphviz the graphic visualization was also possible. A .png file format was chosen and the extracted file looks as it follows. Decision Nodes are colored light orange, Leaf Nodes green, and they are all labeled with their Gini impurity (Gini), the number of instances they include (samples), and the number of instances belonging to each class. Decision Nodes also include in their label the question that represents the splitting criterion.



Discussion

A Classification software was developed including visualization of the classification, both textually and graphically. The methodology was based on Classification and Regression Trees. The software was written in Python programming language using only the language's built-in functions, except for three cases. The first was in the line reading the .csv dataset file, where the 'csv' package of the Standard Library was used. Then, the 'random' package of the Standard Library was used in the two classes of the program (LeafNode, DecisionNode) in order to generate a unique name for each node. The last package that was imported was graphviz in order to use the Graphviz software for the graph visualization of the Tree. The fact that the classification algorithm is written in pure Python makes it comprehensible for all Python users.

A limitation of the method is that it cannot handle missing data. For the Iris flower data set we made the assumption that no data was missing and it was true, but if the set has missing values they have to be handled, for example with a deletion or imputation method, while preparing the dataset. Attention in the dataset preparation is also important because the "dataset preparation" part of the algorithm can be used unmodified only for a specific file format, which includes solely numeric variables except for the class names. Since categorical values can be handled in the "classification" algorithm, a personalization of the data preparation allows the use of it. One last point that has to be mentioned is the maximum depth (prepruning), as depth=4 could be insufficient for big datasets. The depth limit can easily be changed by changing the "if" statement in the "build_tree" function (if depth < 4), where 4 is the maximum depth.

A possible next step for this algorithm could be the separation of the data in training data, which will be used to build the tree, and testing data, which will be used to assess its predictive accuracy. An evaluation of the accuracy will also permit a more advanced pruning method, where the depth is chosen by the algorithm based on accuracy.

Conclusions

Classification and Regression Trees have many advantages and are a popular Machine Learning method. Our software is focused on classification and can handle both numeric and string attributes of the dataset with the appropriate preparation of it. The software can be used in various fields for decision making including health sciences (for classification, diagnosing etc.). In the present project the Iris flower data set was used to train a Classification Tree, where the Classifiers were chosen based on the information gain of each split. The Tree was prepruned at depth = 4 many of the leaf nodes were pure. The software additionally to choosing the classifiers and splitting the data also produces visual results of all the steps in the classification process. The visualization is both textual and graphical using the Graphviz software.

Due to the limited time available for the execution of this project, the original dataset was not divided in training and testing data. Future development of the algorithm should include this addition since it opens the way for the assessment of the accuracy of the Classification Tree, knowledge that is important for its use as a prediction model.

References

Breiman, L., Friedman, J., Stone, C., & Olshen, R. (1984). *Classification and regression trees.* Chapman and Hall, New York.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, pp. 179–188.

Loh, W.-Y. (2011, January). Classification and Regression Trees. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, pp. 14 - 23, DOI:10.1002/widm.8.

Podgorelec, V., Kokol, P., Stiglic, B., & Rozman, I. (2002, October). Decision trees: an overview. *Journal of Medical Systems, Kluwer Academic/Plenum Press*, pp. Vol. 26, Num. 5, pp. 445-463.

Singh, S., & Gupta, P. (2014, July). Comparative study ID3, CART and C4.5 Decision Tree Algorithm: A Survey. *International Journal of Advanced Information Science and Technology (IJAIST)*, pp. Vol.27, No.27, ISSN: 2319:2682.

Python Software Foundation. (n.d.). *Python*. Retrieved from python.org

The Graphviz Authors. (n.d.). *Graphviz*. Retrieved from https://graphviz.org