Universidad Nacional Autónoma de México

Facultad de Ingeniería

Aprendizaje

Árboles de decisión

M.C. Hugo E. Estrada León

Integrantes:

Bustos Ramírez Luis Enrique
Oropeza Vilchis Luis Alberto

Grupo: 1

Semestre 2019-2

1. Objetivo

- Implementar y utilizar algoritmos de creación de árboles de decisión como ID3 y Random Forest.

2. Introducción

El aprendizaje por árboles de decisión es una de los métodos más utilizados para la inferencia inductiva. Es un método que sirve para aproximar funciones de valores discretos, es robusto al ruido presente en los datos y es capaz de aprender expresiones disyuntivas.

Existen diversos algoritmos para generar árboles de decisión a partir de un conjunto de datos, denominado *conjunto de entrenamiento*. En este trabajo se ilustra la implementación, así como el uso de dos algoritmos ampliamente utilizados en el área, *ID3* y *Random Forest*.

3. Desarrollo

3.1. Implementación del algoritmo ID3 desde cero

La implementación se realizó con el lenguaje de programación python versión 2.7. Código:

```
#! /usr/bin/python
# -*- coding: utf-8 -*-

# Universidad Nacional Autonoma de Mexico
# Asignatura: Aprendizaje
# Programa: Algoritmo ID3
# Descripcion: implementacion basica del algoritmo ID3 para crear
# arboles de decision
#
# Autores:
#               Luis Alberto Oropeza Vilchis
#               Luis Enrique Bustos Ramirez


from __future__ import division  # float division
from math import log
import csv  # To read data
import argparse


# To implement tree as data structure
class Node():

    def __init__(self, label, parent=None, depth=0):
        self.label = label
        self.depth = depth
        self.childs = list()

    def add_child(self, condition, node):
        self.childs.append((condition, node))

    def show(self, condition=''):
        if self.depth == 0:
            print('({0})'.format(self))
        else:
            print('|{0}{1}{2}]-({3})'.format('\t'*(self.depth-1),
                                              '|-[' if self.depth != 1 else '-[',
                                              condition, self))
        for n in self.childs:
            n[1].show(n[0])
            print('|')

    def __str__(self):
```

```
        return self.label


# Class to encapsulate problem data
# filename: file to obtain the data
# target: index of column to be classified, default = lastest column
class CSVData():

    def __init__(self, filename=None, target=-1):
        self.target = target
        self.attributes = list()
        self.data = list()
        self.values = dict()
        if filename != None:
            self.filename = filename
            self.load_data()
            self.obtain_all_values()

    # Loads attributes and rows from csv file
    def load_data(self):
        with open(self.filename) as csvfile:
            data = csv.DictReader(csvfile)
            self.attributes = data.fieldnames
            self.data = [x for x in data]

    # Gets the values from the attributes
    def obtain_all_values(self):
        for attr in self.attributes:
            self.values[attr] = list(set(x[attr] for x in self.data))

    # Returns values from an attribute
    def get_attribute_values(self, attr):
        return self.values[attr]

    # Creates a subset data from a particular value of an attribute
    def filter(self, attribute, value):
        new_attrs = filter(lambda x: x != attribute, self.attributes)
        temp = CSVData()
        temp.set_target(self.target)
        temp.set_data(list(filter(lambda x: x[attribute] == value, self.data)))
        temp.set_attributes(new_attrs)
        return temp

    def size(self):
        return len(self.data)

    def get_attributes(self):
        return self.attributes

    def get_rows(self):
        return self.data

    def get_target(self):
        return self.attributes[self.target]

    def get_target_values(self):
        return self.values[self.get_target()]

    def get_row(self, index):
        return self.data[index]

    def get_num_attributes(self):
        return len(self.attributes)

    def set_data(self, data):
        self.data = data

    def set_attributes(self, attrs):
        self.attributes = attrs
        self.obtain_all_values()

    def set_target(self, t):
```

```
        self.target = t


# Class to encapsulate id3 algorithm operations
# data: instance of CSVData
class ID3():

    def __init__(self, data):
        self.data = data

    def run(self):
        return self.id3(self.data)

    def entropy(self, *args):
        args = filter(lambda x: x > 0, args) # to avoid log(0) error
        total = float(sum(args))
        return sum(map(lambda x: -x/total*log(x/total, 2), args))

    def calc_target_entropy(self, data):
        results = []
        for v in data.get_target_values():
            results.append(data.filter(data.get_target(), v).size())
        return self.entropy(*results)

    def calc_attr_entropy(self, data, attr):
        result = 0.0
        total = data.size()
        for value in data.get_attribute_values(attr):
            subset_data = data.filter(attr, value)
            len_data_value = subset_data.size()
            result += (len_data_value/total)*self.calc_target_entropy(subset_data)
        return result

    def calc_attr_gain(self, data, attr, entropy_set):
        return entropy_set - self.calc_attr_entropy(data, attr)

    def id3(self, data, depth=0):
        entropy_set = self.calc_target_entropy(data)
        if entropy_set == 0:
            return Node(data.get_target_values()[0], depth=depth)
        if data.get_num_attributes() == 1:
            return Node('Not decided', depth=depth)
        gains = [(x, self.calc_attr_gain(data, x, entropy_set)) for x in data.
            get_attributes()[:-1]]
        winner = max(gains, key=lambda x: x[1])
        tree = Node(winner[0], depth=depth)
        for value in data.get_attribute_values(winner[0]):
            tree.add_child(value, self.id3(data.filter(winner[0], value), depth+1))
        return tree


def main():
    parser = argparse.ArgumentParser(description='ID3 algorithm')
    parser.add_argument('file', metavar='file', type=str, help='File path to
        retrieve the data')
    args = parser.parse_args()
    data = CSVData(args.file)
    id3 = ID3(data)
    tree = id3.run()
    tree.show()


if __name__ == '__main__':
    main()
```

Manual de uso

Para su funcionamiento sólo es necesario tener instalado python 2.7. Se utilizaron algunos módulos, sin embargo son parte de la biblioteca estándar de python por lo que no es necesario hacer instalación de dependencias. El programa se realizó con estructura de *script*,

a continuación se listan las opciones disponibles:

- Para mostrar ayuda:
  `$ python id3.py -h`

- Para utilizar el algoritmo:
  `$ python id3.py datos.csv`

  donde **datos.csv** es la ruta del archivo que contiene los datos en formato *csv*

3.2. Utilizando la biblioteca *sklearn*

3.2.1. Algoritmo ID3

```
import numpy as np
import matplotlib.pyplot as plt
import graphviz
from sklearn.metrics import precision_recall_fscore_support
from id3 import Id3Estimator, export_graphviz
from sklearn.datasets import load_digits

digits = load_digits()
size = int (digits.data.shape[0]*0.8)

x_train = digits.data[:size, :]
x_test = digits.data[size:, :]
y_train =digits.target[:size]
y_test = digits.target[size:]

id3 = Id3Estimator()
id3.fit(x_train, y_train)

y_predict = id3.predict(x_test)

print(precision_recall_fscore_support(y_test, y_predict))

export_graphviz(id3.tree_, 'digits.dot', digits.target)
with open("digits.dot") as f:
    dot_graph = f.read()
g = graphviz.Source(dot_graph)
g.render()
g.view()
```

3.2.2. Algoritmo Random Forest

```
import numpy as np
import matplotlib.pyplot as plt
import graphviz
import sklearn.metrics
from sklearn.datasets import load_digits
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import RandomForestClassifier

#Se carga el data set, y se divide en 80% entrenamiento 20% test
digits = load_digits()
size = int (digits.data.shape[0]*0.8)
x_train = digits.data[:size, :]
x_test = digits.data[size:, :]
y_train =digits.target[:size]
y_test = digits.target[size:]

#Se crea un clasificador con 25 arboles
```

```
classifier=RandomForestClassifier(n_estimators=25)
#Se entrena el clasificador con los datos de entrenamiento
classifier=classifier.fit(x_train,y_train)
#Se ingresan los datos de testing para su prediccion
predictions=classifier.predict(x_test)
#Se imprime la precision que tiene el modelo
sklearn.metrics.accuracy_score(y_test, predictions)

model = ExtraTreesClassifier()
model.fit(x_train,y_train)
#print(model.feature_importances_)
#list(model.feature_importances_)
#Se grafican la importancia de los atributos que contiene el data set
from matplotlib import pyplot
pyplot.bar(range(len(model.feature_importances_)), model.feature_importances_)
pyplot.show()

#Se grafica la importancia de cada uno de los arboles que se crearon
trees=range(25)
accuracy=np.zeros(25)
for idx in range(len(trees)):
    classifier=RandomForestClassifier(n_estimators=idx + 1)
    classifier=classifier.fit(x_train,y_train)
    predictions=classifier.predict(x_test)
    accuracy[idx]=sklearn.metrics.accuracy_score(y_test, predictions)

plt.cla()
plt.plot(trees, accuracy)
```

4. Resultados

4.1. Implementacion

El siguiente conjunto de datos se utiliza como prueba para el algoritmo:

Clima	**NV**	**Ventas**	**Region**	**Robo**
Soleado	Alta	Altas	Pueblo	No
Soleado	Alta	Altas	Ciudad	No
Nublado	Alta	Altas	Pueblo	Si
Lluvioso	Media	Altas	Pueblo	Si
Lluvioso	Baja	Normales	Pueblo	Si
Lluvioso	Baja	Normales	Ciudad	No
Nublado	Baja	Normales	Ciudad	Si
Soleado	Media	Altas	Pueblo	No
Soleado	Baja	Normales	Pueblo	Si
Lluvioso	Media	Normales	Pueblo	Si
Soleado	Media	Normales	Ciudad	Si
Nublado	Media	Altas	Ciudad	Si
Nublado	Alta	Normales	Pueblo	Si
Lluvioso	Media	Altas	Ciudad	No

El script muestra la información correspondiente:

```
        ==== Generated Tree ====
(Clima)
|--[Soleado]--(NV)
|               |--[Baja]--(Si)
|
|               |--[Media]--(No)
|
|               |--[Alta]--(No)
|
|
|--[Nublado]--(Si)
|
|--[Lluvioso]--(Region)
|               |--[Pueblo]--(Si)
|
|               |--[Ciudad]--(No)
|
|

        ==== Statistics ====
True Positive: 8
True Negative: 6
False Positive: 0
False Negative: 0
Precision: 100.00%
True Positive Rate: 1.0
False Positive Rate: 0.0

        ==== Evaluating test data ====
Data: {'Robo': 'Si', 'Clima': 'Soleado', 'Region': 'Ciudad', 'NV': 'Media', 'Ventas': 'Normales'}
Prediction: No
```

Figura 1: Salida del script

4.2. Usando bibliotecas

4.2.1. ID3

Se utilizó un conjunto de datos de imágenes que representan digitos, el árbol que resultó de la ejecución:

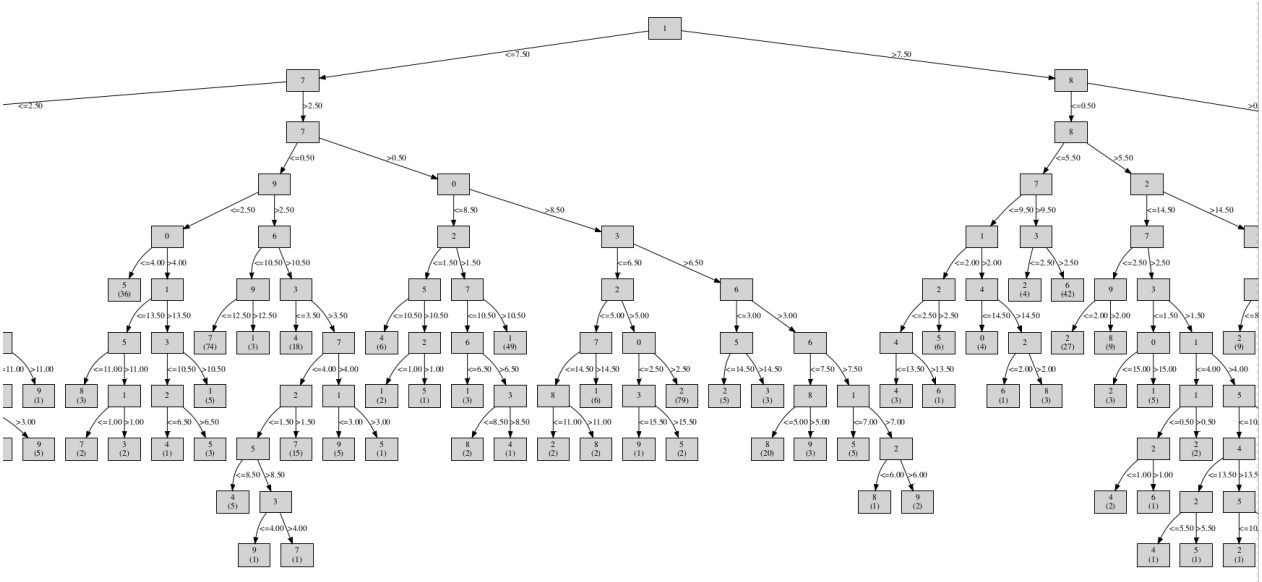


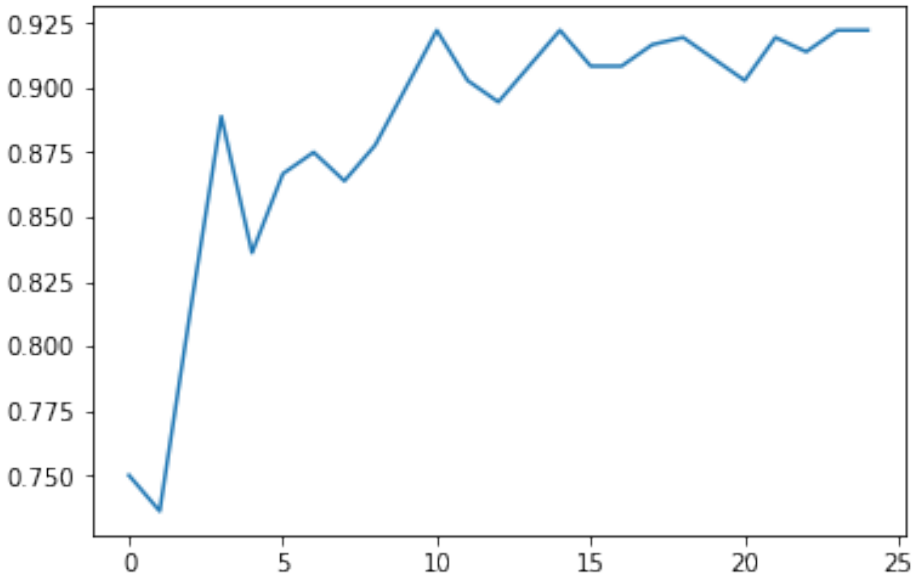
Figura 2: Parte del árbol generado

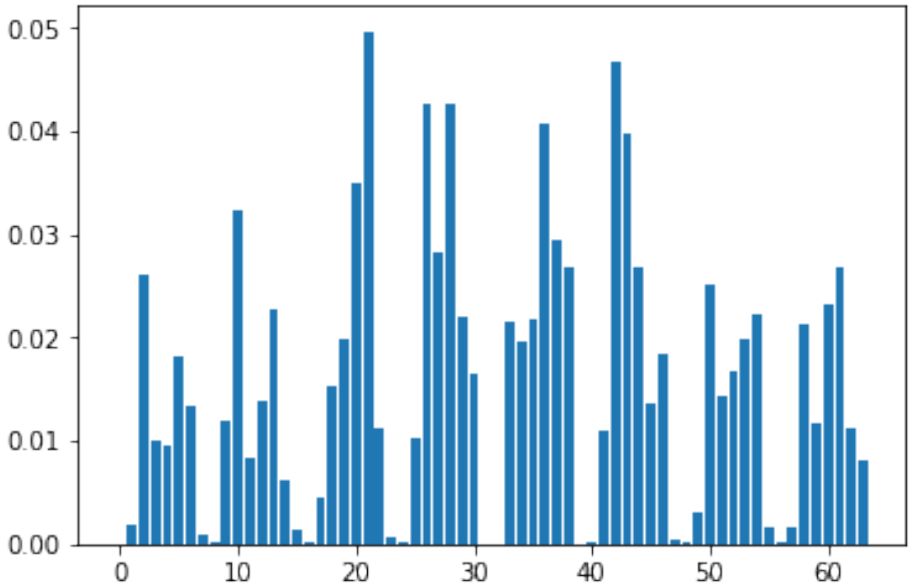
Resultado del *f-score*:

```
 (array([0.91666667, 0.75862069, 0.71794872, 0.80769231, 0.86111111,
0.82926829, 1.        , 0.71111111, 0.71428571, 0.70731707]), array
    ([0.94285714, 0.61111111, 0.8       , 0.56756757, 0.83783784,
0.91891892, 0.86486486, 0.88888889, 0.75757576, 0.78378378]), array
    ([0.92957746, 0.67692308, 0.75675676, 0.66666667, 0.84931507,
0.87179487, 0.92753623, 0.79012346, 0.73529412, 0.74358974]), array([35, 36,
     35, 37, 37, 37, 37, 36, 33, 37]))
```

4.2.2. Random Forest

El resultado de los árboles es:





5. Conclusiones

Bustos Ramírez Luis Enrique

Estos programas me ayudaron a practicar la teoría vista en clase, pero más que eso, tuve un acercamiento a su implementación en lenguaje python; pues a pesar de que ya están implementados ciertos métodos, hay que conocerlos para poder aplicarlos. No sólo sirve copiar y pegar.

Oropeza Vilchis Luis Alberto

Los árboles de decisión me parecen sencillos de entender. Aprender la forma en que se contruyen, así como su funcionamiento es de gran utilidad, ya que nos aporta una nueva forma de poder resolver tareas de decisión automatica de una manera simple. Me quedo con la impresión de que son una herramienta de gran utilidad para conjuntos de datos que no son tan grandes y complejos, debido al tamaño que podría llegar a tener el árbol en el peor de los casos.

Referencias

[1] Bhardwaj, R. & Vatta S. (6, June 2013). Implementation of ID3 Algorithm. International Journal of Advanced Research in Computer Science and Software Engineering, 3, 7.

[2] Bahety, A. Extension and Evaluation of ID3 - Decision Tree Algorithm. Department of Computer Science, University of Maryland.

[3] Debreuve, E. And Introduction to random forest. University Nice Sophia Antipolis.

[4] Castrillejo, A., da Silva, N. & Illanes, G. (19 de Julio de 2010) Consistencia de Random Forest y otros clasificadores promediados. Universidad de la República.

[5] http://www.math.mcgill.ca/yyang/resources/doc/randomforest.pdf