DWDM Lab Assignment - 7
CSE 317

30 March 2022

Gyanendra Kumar Shukla
CSE 1
191112040

Department of Computer Science
NIT, Bhopal

Contents

1 Decision Tree algorithms

Use the dataset that we discussed in class (given below) having 14 tuples and 04 attributes along with 01 target attribute. Show all the parameters like, Info_Gain, Gain_Ratio etc for each attribute in output also.

Table 1: Class-labeled Training Tuples from the AllElectronics Customer Database

age	**income**	**student**	**credit_rating**	**buys_computer**
youth	high	no	fair	no
youth	high	no	excellent	no
middle_aged	high	no	fair	yes
senior	medium	no	fair	yes
senior	low	yes	fair	yes
senior	low	yes	excellent	no
middle_aged	low	yes	excellent	yes
youth	medium	no	fair	no
youth	low	yes	fair	yes
senior	medium	yes	fair	yes
youth	medium	yes	excellent	yes
middle_aged	medium	no	excellent	yes
middle_aged	high	yes	fair	yes
senior	medium	no	excellent	no

1.1 ID3 Algorithm

Write a program to construct Decision Tree based on ID3 algorithm.

1.1.1 CODE

```
from collections import defaultdict
import numpy as np
import pandas as pd
import pprint


eps = np.finfo(float).eps


class Dataset:
    """
    The Dataset class is used to store the data and the labels.
    The df attribute is a pandas dataframe.
    """
    age = ['youth', 'youth', 'middle_aged', 'senior', 'senior', 'senior',
    ↪  'middle_aged',
           'youth', 'youth', 'senior', 'youth', 'middle_aged', 'middle_aged',
           ↪  'senior']
    income = ['high', 'high', 'high', 'medium', 'low', 'low', 'low',
              'medium', 'low', 'medium', 'medium', 'medium', 'high', 'medium']
    student = ['no', 'no', 'no', 'no', 'yes', 'yes', 'yes',
               'no', 'yes', 'yes', 'yes', 'no', 'yes', 'no']
    credit_rating = ['fair', 'excellent', 'fair', 'fair', 'fair', 'excellent',
                     'excellent', 'fair', 'fair', 'fair', 'excellent', 'excellent',
                     ↪  'fair', 'excellent']
    buys_computer = ['no', 'no', 'yes', 'yes', 'yes', 'no',
                     'yes', 'no', 'yes', 'yes', 'yes', 'yes', 'yes', 'no']

    dataset = {
        'age': age,
        'income': income,
        'student': student,
        'credit_rating': credit_rating,
        'buys_computer': buys_computer,
    }

    # The pandas dataframe containing the data.
    df = pd.DataFrame(dataset, columns=[
                      'age', 'income', 'student', 'credit_rating', 'buys_computer'])

    @classmethod
    def savetocsv(self):
        self.df.to_csv('data.csv', index=False)


class Id3DecisionTree:
    """
    A Decision Tree class that implements the ID3 algorithm.
    """

    def __init__(self, df) -> None:
        """
        Initializes the decision tree with a dataframe.

        :param df: The dataframe containing the data.
        """
        self.df = df
        self.tree = defaultdict()

    def information_gain_entire(self) -> float:
        """
        Calculates the information gain of the entire dataset.

        :return: The information gain of the entire dataset.
        """
        info = 0
        output_labels = self.df['buys_computer'].unique()
        for output in output_labels:
            pi = self.df['buys_computer'].value_counts()[output] / \
                len(self.df['buys_computer'])
            info += -pi*np.log2(pi)

        return info

    def information_gain_attribute(self, attribute: str) -> float:
        """
        Calculates the information gain of a given attribute.
        :param attribute: The attribute to calculate the information gain of.
        :return: The information gain of the given attribute.
        """

        target_label = self.df['buys_computer'].unique()
        attribute_vars = self.df[attribute].unique()

        info_attr = 0

        for attr in attribute_vars:
            info_feature = 0
            for target in target_label:
                pi_n = len(self.df[attribute][self.df[attribute]
                                              == attr][self.df['buys_computer'] ==
                                              ↪  target])
                pi_d = len(self.df[attribute][self.df[attribute] == attr])
                pi = pi_n/(pi_d+eps)

                info_feature += -pi*np.log2(pi+eps)
            pi_ex = pi_d/len(self.df)
            info_attr += pi_ex*info_feature

        return info_attr

    def information_gain(self) -> defaultdict:
        """
        Calculates the information gain of all the attributes
        except the target label.

        :return: A dictionary containing the information gain of all the attributes.
        """
        info_gain_df = self.information_gain_entire()
        info_gain = defaultdict()
        for attr in self.df.keys()[:-1]:
            info_gain[attr] = info_gain_df - \
                self.information_gain_attribute(attr)
        return info_gain

    def root_attribute(self) -> str:
        """
        Calculates the root attribute of the decision tree.

        :returns: The root attribute of the decision tree.
        """
        ig = [val for (key, val) in self.information_gain().items()]
        return self.df.keys()[:-1][np.argmax(ig)]

    def get_subtable(self, node, value) -> pd.DataFrame:
        """
        Creates a subtable by filtering the dataframe based on the given node and
↪   value.
```

```
        :param node: The column to filter the dataframe by.
        :param value: The attribute in the column to filter the dataframe by.

        :return: A subtable of the dataframe.
        """
        return self.df[self.df[node] == value].reset_index(drop=True)

    def __str__(self):
        return f'{pprint.pformat(self.tree, indent=4)}'

    def print_tree(self, tab=0):
        """
        Prints the decision tree in a readable format.

        :param tab: The number of tabs to indent the tree.
        """
        root = list(self.tree.keys())[0]
        print(root, '?')
        for key, value in self.tree[root].items():
            print('\t'*tab, f"|-{key} -> ", end=' ')
            if isinstance(value, Id3DecisionTree):
                print_tree(value, tab+4)
            else:
                print(value)

    def predict(self, inp: dict):
        """
        Predict the output of the decision tree for a given input.

        :param inp: The input to predict the output of the decision tree for.
        """
        root = list(self.tree.keys())[0]
        subtree = self.tree[root][inp[root]]
        if isinstance(subtree, Id3DecisionTree):
            return make_prediction(subtree, inp)
        return subtree


def build_tree(df: pd.DataFrame) -> Id3DecisionTree:
    """
    Builds a decision tree for the given dataframe
    """
    parent_tree = Id3DecisionTree(df)
    root = parent_tree.root_attribute()
    parent_tree.tree[root] = defaultdict()

    attrOfRootNode = np.unique(parent_tree.df[root])

    for attr in attrOfRootNode:
        subtable = parent_tree.get_subtable(root, attr)
        classValues = np.unique(subtable['buys_computer'])

        if (len(classValues)) == 1:
            parent_tree.tree[root][attr] = classValues[0]
        else:
            parent_tree.tree[root][attr] = build_tree(subtable)

    return parent_tree


def make_prediction(dt, inp):
    """
    Predicts the output of the decision tree for a given input.
    """
    root = list(dt.tree.keys())[0]
    subtree = dt.tree[root][inp[root]]
    if isinstance(subtree, Id3DecisionTree):
        return make_prediction(subtree, inp)
    return subtree


def print_tree(dt, tab=0):
    root = list(dt.tree.keys())[0]
    print(root, '?')
    for key, value in dt.tree[root].items():
        print('\t'*tab, f"|-{key} -> ", end=' ')
        if isinstance(value, Id3DecisionTree):
            print_tree(value, tab+4)
        else:
            print(value)


# Build the decision tree
t = build_tree(Dataset.df)

print("\nThe decision tree is: ")
t.print_tree()

print(f'\nthe information gain is: ', *
      [(k, v) for (k, v) in t.information_gain().items()], sep='\n\t')

inp = {
    'age': 'youth',
    'income': 'low',
    'student': 'no',
    'credit_rating': 'excellent',
}


print(
    f'\nThe prediction for the input {inp} is: \033[1m{t.predict(inp)}\033[0m')
```

1.1.2 OUTPUT

```
The decision tree is:
age ?
 |-middle_aged ->  yes
 |-senior ->  credit_rating ?
                                 |-excellent ->  no
                                 |-fair ->  yes
 |-youth ->  student ?
                                 |-no ->  no
                                 |-yes ->  yes

the information gain is:
        ('age', 0.24674981977443977)
        ('income', 0.029222565658955535)
        ('student', 0.15183550136234225)
        ('credit_rating', 0.048127030408270155)

The prediction for the input {'age': 'youth', 'income': 'low', 'student': 'no',
↪  'credit_rating': 'excellent'} is: no
```

1.2 C4.5 Algorithm

Write a program to construct Decision Tree based on C4.5 algorithm.

1.2.1 CODE

```
from collections import defaultdict
import numpy as np
import pandas as pd
import pprint


eps = np.finfo(float).eps


class Dataset:
    """
    The Dataset class is used to store the data and the labels.
    The df attribute is a pandas dataframe.
    """
    age = ['youth', 'youth', 'middle_aged', 'senior', 'senior', 'senior',
    ↪  'middle_aged',
           'youth', 'youth', 'senior', 'youth', 'middle_aged', 'middle_aged',
           ↪  'senior']
    income = ['high', 'high', 'high', 'medium', 'low', 'low', 'low',
              'medium', 'low', 'medium', 'medium', 'medium', 'high', 'medium']
    student = ['no', 'no', 'no', 'no', 'yes', 'yes', 'yes',
               'no', 'yes', 'yes', 'yes', 'no', 'yes', 'no']
    credit_rating = ['fair', 'excellent', 'fair', 'fair', 'fair', 'excellent',
                     'excellent', 'fair', 'fair', 'fair', 'excellent', 'excellent',
                     ↪  'fair', 'excellent']

    buys_computer = ['no', 'no', 'yes', 'yes', 'yes', 'no',
                     'yes', 'no', 'yes', 'yes', 'yes', 'yes', 'yes', 'no']

    dataset = {
        'age': age,
        'income': income,
        'student': student,
        'credit_rating': credit_rating,
        'buys_computer': buys_computer,
    }

    # The pandas dataframe containing the data.
    df = pd.DataFrame(dataset, columns=[
                      'age', 'income', 'student', 'credit_rating', 'buys_computer'])

# print(Dataset().df)


class C45DecisionTree:
    """
    A decision tree class that implements C4.5 algorithm.
    """

    def __init__(self, df) -> None:
        """
        Initialize the decision tree with a dataframe.

        :param df: pandas dataframe containing the data
        """
        self.df = df
        self.tree = defaultdict()

    def information_gain_entire(self):
        """
        Calculates the information gain of the entire dataset.

        :return: The information gain of the entire dataset.
        """
        info = 0
        output_labels = self.df['buys_computer'].unique()
        for output in output_labels:
            pi = self.df['buys_computer'].value_counts()[output] / \
                len(self.df['buys_computer'])
            info += -pi*np.log2(pi)

        return info

    def information_gain_attribute(self, attribute):
        """
        Calculates the information gain of a given attribute.

        :param attribute: The attribute to calculate the information gain of.
        :return: The information gain of the given attribute.
        """
        target_label = self.df['buys_computer'].unique()
        attribute_vars = self.df[attribute].unique()

        info_attr = 0

        for attr in attribute_vars:
            info_feature = 0
            for target in target_label:
                pi_n = len(self.df[attribute][self.df[attribute]
                                              == attr][self.df['buys_computer'] ==
                                              ↪  target])
                pi_d = len(self.df[attribute][self.df[attribute] == attr])
                pi = pi_n/(pi_d+eps)

                info_feature += -pi*np.log2(pi+eps)
            pi_ex = pi_d/len(self.df)
            info_attr += pi_ex*info_feature

        return info_attr
    def information_gain(self):
        """
        Calculates the information gain of all the attributes
        except the target label.

        :return: A dictionary containing the information gain of all the attributes.
        """
        info_gain_df = self.information_gain_entire()
        info_gain = defaultdict()
        for attr in self.df.keys()[:-1]:
            info_gain[attr] = info_gain_df - \
                self.information_gain_attribute(attr)
        return info_gain

    def split_info_attribute(self, attribute):
        """
        Calculates the split info of a given attribute.

        :param attribute: The attribute to calculate the split info of.

        :return: The split info of the given attribute.
        """
        unique_vals = self.df[attribute].unique()
        split_info = 0
        total_elems = len(self.df)
        for val in unique_vals:
            rat = len(self.df[attribute]
                      [self.df[attribute] == val])/total_elems
            split_info += -rat*np.log2(rat)
        return split_info

    def split_info(self):
        """
        Calculates the split info of all the attributes.

        :return: A dictionary containing the split info of all the attributes.
        """
        split_info_all = defaultdict()
        for attr in self.df.keys()[:-1]:
            split_info_all[attr] = self.split_info_attribute(attr)
        return split_info_all

    def information_gain_ratio(self):
        """
        Calculates the information gain ratio of all the attributes.
        The information gain ratio is the information gain divided by the split info.

        :return: A dictionary containing the information gain ratio of all the
↪   attributes.
        """
        info_gain = self.information_gain()
        split_info = self.split_info()
        info_gain_ratio = defaultdict()
        for attr in self.df.keys()[:-1]:
            info_gain_ratio[attr] = info_gain[attr]/(split_info[attr]+eps)
        return info_gain_ratio

    def root_attribute(self):
        """
        Calculates the root attribute of the decision tree.

        :returns: The root attribute of the decision tree.
        """
        ig = [val for (key, val) in self.information_gain_ratio().items()]
        return self.df.keys()[:-1][np.argmax(ig)]

    def get_subtable(self, node, value):
        """
        Creates a subtable by filtering the dataframe based on the given node and
↪   value.
        It also removes the node from the dataframe.

        :param node: The column to filter the dataframe by.
        :param value: The attribute in the column to filter the dataframe by.

        :return: A subtable of the dataframe.
        """
        return self.df[self.df[node] == value].drop([node],
        ↪   axis=1).reset_index(drop=True)

    def __str__(self):
        return f'{pprint.pformat(self.tree, indent=4)}'

    def print_tree(self, tab=0):
        """
        Prints the decision tree in a readable format.

        :param tab: The number of tabs to indent the tree.
        """
        root = list(self.tree.keys())[0]
        print(root, '?')
        for key, value in self.tree[root].items():
            print('\t'*tab, f"|-{key} -> ", end=' ')
            if isinstance(value, C45DecisionTree):
                print_tree(value, tab+4)
            else:
                print(value)

    def predict(self, inp):
        """
        Predict the output of the decision tree given an input.

        :param inp: The input to predict the output of the decision tree.
        """
        root = list(self.tree.keys())[0]
        subtree = self.tree[root][inp[root]]
        if isinstance(subtree, C45DecisionTree):
            return make_prediction(subtree, inp)
        return subtree


def build_tree(df):
    """
    Builds a decision tree for the given input dataframe.
    """
    parent_tree = C45DecisionTree(df)
    root = parent_tree.root_attribute()
    parent_tree.tree[root] = defaultdict()

    attrOfRootNode = np.unique(parent_tree.df[root])

    for attr in attrOfRootNode:
        subtable = parent_tree.get_subtable(root, attr)
        classValues = np.unique(subtable['buys_computer'])

        if (len(classValues)) == 1:
            parent_tree.tree[root][attr] = classValues[0]
        else:
            parent_tree.tree[root][attr] = build_tree(subtable)

    return parent_tree


def make_prediction(dt, inp):
    """
    Predicts the output of the decision tree given an input.
    """
    root = list(dt.tree.keys())[0]
    subtree = dt.tree[root][inp[root]]
    if isinstance(subtree, C45DecisionTree):
        return make_prediction(subtree, inp)
    return subtree


def print_tree(dt, tab=0):
    root = list(dt.tree.keys())[0]
    print(root, '?')
    for key, value in dt.tree[root].items():
        print('\t'*tab, f"|-{key} -> ", end=' ')
        if isinstance(value, C45DecisionTree):
            print_tree(value, tab+4)
        else:
            print(value)


print("The training dataset is: ")
print(Dataset.df)
```

```python
# Build the decision tree
t = build_tree(Dataset.df)

print("\nThe decision tree is: ")
t.print_tree()

print(f'\nthe information gain is: ', *
      [(k, v) for (k, v) in t.information_gain().items()], sep='\n\t')
print(f'\nthe information gain ratio is: ', *
      [(k, v) for (k, v) in t.information_gain_ratio().items()], sep='\n\t')

inp = {
    'age': 'youth',
    'income': 'low',
    'student': 'no',
    'credit_rating': 'excellent',
}


print(
    f'\nThe prediction for the input {inp} is: \033[1m{t.predict(inp)}\033[0m')
```

1.2.2 OUTPUT

```
The decision tree is:
age ?
 |-middle_aged ->  yes
 |-senior ->  credit_rating ?
                                 |-excellent ->  no
                                 |-fair ->  yes
 |-youth ->  student ?
                                 |-no ->  no
                                 |-yes ->  yes

the information gain is:
        ('age', 0.24674981977443977)
        ('income', 0.029222565658955535)
        ('student', 0.15183550136234225)
        ('credit_rating', 0.048127030408270155)

the information gain ratio is:
        ('age', 0.15642756242117553)
        ('income', 0.018772646222241924)
        ('student', 0.15183550136234222)
        ('credit_rating', 0.04884861551152149)

The prediction for the input {'age': 'youth', 'income': 'low', 'student': 'no',
↪  'credit_rating': 'excellent'} is: no
```