

Final Assignment Report : Decision Tree Algorithm and K-Nearest Neighbour Algorithm

CSE-0408 Summer 2021

Ahasanul Bari Rifat
Department of Computer Science and Engineering
State University of Bangladesh (SUB)
Dhaka, Bangladesh
ahsanulbari.rifat88@gmail.com

Abstract—Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules.

An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If $k = 1$, then the object is simply assigned to the class of that single nearest neighbor

Index Terms—Python

I. INTRODUCTION

Decision tree are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches. It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.

KNN also known as K-nearest neighbour is a supervised and pattern classification learning algorithm which helps us find which class the new input(test value) belongs to when k nearest neighbours are chosen and distance is calculated between them.

II. WHY USE DECISION TREES?

Below are the two reasons for using the Decision tree:

- Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.
- The logic behind the decision tree can be easily understood because it shows a tree-like structure.

III. WHY OR HOW USE K-NEAREST NEIGHBOUR ALGORITHM?

KNN algorithm can be used for both classification and regression problems. The KNN algorithm uses 'feature similarity' to predict the values of any new data points.

IV. EXAMPLE OF DECISION TREE AND KNN ALGORITHM

An example of a decision tree can be explained using above binary tree.

KNN also known as K-nearest neighbour is a supervised and pattern classification learning algorithm which helps us find which class the new input(test value) belongs to when k nearest neighbours are chosen and distance is calculated between them.

V. ADVANTAGES AND DISADVANTAGES OF DECISION TREE ALGORITHM

Advantages :

- It is easy to grasp because it follows a constant method that somebody follows whereas creating any call-in real-life.
- It is terribly helpful for the resolution of decision-related issues.

Disadvantages :

- It may have an associate overfitting issue, which might be resolved exploitation the Random Forest formula.
- For a lot of category labels, the process quality of the choice tree could increase.

VI. ADVANTAGES AND DISADVANTAGES OF KNN ALGORITHM

Advantages :

- New data can be added without effecting the algorithm performance or accuracy.
- Versatile – useful for regression and classification.
- High accuracy – you do not need to compare with better-supervised learning models.

Disadvantages :

- accuracy depends on the quality of the data.
- With large data, the prediction stage might be slow.
- Sensitive to the scale of the data and irrelevant features.
- Require high memory – need to store all of the training data

VII. REPORT CODE OF DECISION TREE ALGORITHM

```
jupyter treecodecision Last Checkpoint: a few seconds ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

In [ ]: import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import math
import copy

dataset = pd.read_csv('Book1.csv')
X = dataset.iloc[:, 1:].values
# print(X)
attribute = ['outlook', 'temp', 'humidity', 'wind']

class Node(object):
    def __init__(self):
        self.value = None
        self.decision = None
        self.children = None

def findEntropy(data, rows):
    yes = 0
    no = 0
    ans = -1
    idx = len(data[0]) - 1
    entropy = 0
    for i in rows:
        if data[i][idx] == 'Yes':
            yes = yes + 1
        else:
```

Fig. 1.

```
jupyter treecodecision Last Checkpoint: a few seconds ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

x = yes/(yes+no)
y = no/(yes+no)
if x != 0 and y != 0:
    entropy = -1 * (x*math.log2(x) + y*math.log2(y))
if x == 1:
    ans = 1
if y == 1:
    ans = 0
return entropy, ans

def findMaxGain(data, rows, columns):
    maxGain = 0
    retidx = -1
    entropy, ans = findEntropy(data, rows)
    if entropy == 0:
        print("Yes")
    else:
        print("No")
    return maxGain, retidx, ans

for j in columns:
    mydict = {}
    idx = j
    for i in rows:
        key = data[i][idx]
        if key not in mydict:
            mydict[key] = 1
        else:
            mydict[key] = mydict[key] + 1
    gain = entropy
```

Fig. 2.

```
jupyter treecodecision Last Checkpoint: a minute ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

# print(mydict)
for key in mydict:
    yes = 0
    no = 0
    for k in rows:
        if data[k][j] == key:
            if data[k][1] == 'Yes':
                yes = yes + 1
            else:
                no = no + 1
    # print(yes, no)
    x = yes/(yes+no)
    y = no/(yes+no)
    # print(x, y)
    if x != 0 and y != 0:
        gain += (mydict[key] * (x*math.log2(x) + y*math.log2(y)))/14
    # print(gain)
    if gain > maxGain:
        # print("hello")
        maxGain = gain
        retidx = j

return maxGain, retidx, ans

def buildTree(data, rows, columns):
    maxGain, idx, ans = findMaxGain(X, rows, columns)
    root = Node()
    root.children = []
    # print(maxGain)
```

Fig. 3.

```
jupyter treecodecision Last Checkpoint: a minute ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

newcolumns.remove(idx)
for key in mydict:
    newrows = []
    for i in rows:
        if data[i][idx] == key:
            newrows.append(i)
    # print(newrows)
    temp = buildTree(data, newrows, newcolumns)
    temp.decision = key
    root.children.append(temp)
    return root

def traverse(root):
    print(root.decision)
    print(root.value)
    n = len(root.children)
    if n > 0:
        for i in range(0, n):
            traverse(root.children[i])

def calculate():
    rows = [i for i in range(0, 14)]
    columns = [i for i in range(0, 4)]
    root = buildTree(X, rows, columns)
    root.decision = 'Start'
    traverse(root)

calculate()
```

Fig. 4.

VIII. REPORT CODE OF KNN ALGORITHM

```
jupyter knncodemy Last Checkpoint: a minute ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

In [1]: import numpy as np
from itertools import groupby
import math
import collections
from copy import deepcopy
import pickle

class TreeNode:
    def __init__(self, split, col_index):
        self.col_id = col_index
        self.split_value = split
        self.parent = None
        self.left = None
        self.right = None

class Tree():
    def __init__(self):
        self.treemodel = None

    def train(self, trainData):
        #Attributes/Last Column is class
        self.createTree(trainData)

    def createTree(self, trainData):
        #create the tree
        self.treemodel = build_tree(trainData, [])
        saveTree(self.treemodel)
```

Fig. 5.

```
jupyter knncodemy Last Checkpoint: 2 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

def accuracy_confusion_matrix(self, testData):
    #prints the tree confusion matrix along with the accuracy
    build_confusion_matrix(self.treemodel, testData)

#returns the best split on the data instance along
with the splitted dataset and column index
def getBestSplit(data):
    #set the max information gain
    maxInfoGain = -float('inf')

    #convert to array
    dataArray = np.asarray(data)

    #to extract rows and columns
    dimension = np.shape(dataArray)

    #iterate through the matrix
    for col in range(dimension[1]-1):
        dataArray = sorted(dataArray, key=lambda x: x[col])
        for row in range(dimension[0]-1):
            val1 = dataArray[row][col]
            val2 = dataArray[row+1][col]
            expectedSplit = (float(val1)+float(val2))/2.0
            InfoGain, l, r = calcInfoGain(data, col, expectedSplit)
            if (InfoGain > maxInfoGain):
                maxInfoGain = InfoGain
                best = (col, expectedSplit, l, r)

    return best

#This method is used to calculate the gain and returns
the left and right data as per the split
```

Fig. 6.

```

jupyter kncodemy Last Checkpoint 2 minutes ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
+ + + + + Run Code
def calcInfoGain(data,col,split):
    totalLen = len(data)
    infoGain = entropy(data)

    left_data, right_data = getDataSetSplit(data,split,col)

    infoGain = infoGain - ((len(left_data)/totalLen) * entropy(left_data))
    infoGain = infoGain - ((len(right_data)/totalLen) * entropy(right_data))

    return infoGain,left_data,right_data

def getDataSetSplit(data, split, col):
    l_data=[]
    r_data=[]

    for val in data:
        if(val[col]<split):
            l_data.append(val)
        else:
            r_data.append(val)

    return l_data,r_data

#calculates the entropy of the data set provided
def entropy(data):
    totalLen = len(data)
    entropy = 0
    group_by_class = groupby(data, lambda x:x[5])
    for key,group in group_by_class:
        grp_len = len(list(group))
        entropy+= -(grp_len/totalLen)*math.log((grp_len/totalLen),2)

```

Fig. 7.

```

jupyter kncodemy Last Checkpoint 2 minutes ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
+ + + + + Run Code
#this counts all the column class variable row values and finds most common in it
return collections.Counter(np.asarray(data[:,5])).most_common(1)[0][0]

else:
    bestsplit= getBestSplit(data)
    node = Treenode(bestsplitt[1],bestsplitt[0])
    node.left= build_tree(bestsplitt[2],data)
    node.right= build_tree(bestsplitt[3],data)
    return node

#this method is used to classify the test set with the model created
def classify(tree, row):
    if type(tree)==str:
        return tree
    if row[tree.col_id]<tree.split_value:
        return classify(tree.left, row)
    else:
        return classify(tree.right, row)

#this method saves the decision tree model using pickle package
def saveTree(tree):
    decisionTree= deepcopy(tree)
    pickle.dump(decisionTree,open('model.pkl','wb'))

#this method creates a confusion matrix and finds accuracy for test dataset
def build_confusion_matrix(tree, data):
    confusion_mat = [[0 for row in range(4)]for col in range(4)]

    total_len=len(data)
    num_correct_instances=0;
    num_incorrect_instances = 0;

```

Fig. 8.

IX. ASSIGNMENT OUTPUT (DECISION TREE ALGORITHM PROBLEM SOLVING)

```

calculate()

Start
outlook
Sunny
No
Overcast
Yes
Rain
wind
Weak
Yes
Strong
No
Overcast
Yes
Sunny
temp
Mild
No
Cool
Yes
Hot
Yes

```

Fig. 9.

X. ASSIGNMENT OUTPUT (KNN ALGORITHM OUTPUT)

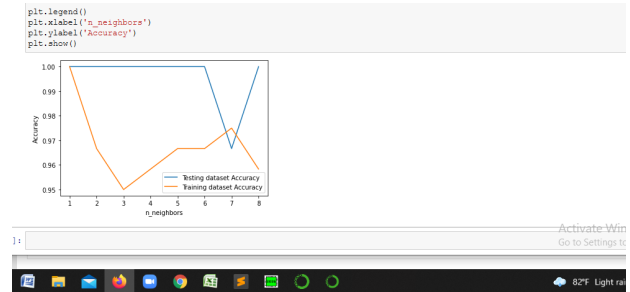


Fig. 10.

ACKNOWLEDGMENT

I would like to thank my honourable **Khan Md. Hasib Sir** for his time, generosity and critical insights into this project.

REFERENCES

- [1] Ben-Haim, Yael, and Elad Tom-Tov. "A Streaming Parallel Decision Tree Algorithm." Journal of Machine Learning Research 11.2 (2010).
- [2] Freund, Yoav, and Llew Mason. "The alternating decision tree learning algorithm." icml. Vol. 99. 1999.
- [3] Su, Jiang, and Harry Zhang. "A fast decision tree learning algorithm." Aaai. Vol. 6. 2006.
- [4] Priyam, Anuja, et al. "Comparative analysis of decision tree classification algorithms." International Journal of current engineering and technology 3.2 (2013): 334-337.
- [5] Soucy, P., Mineau, G. W. (2001, November). A simple KNN algorithm for text categorization. In Proceedings 2001 IEEE International Conference on Data Mining (pp. 647-648). IEEE.
- [6] Deng, Z., Zhu, X., Cheng, D., Zong, M., Zhang, S. (2016). Efficient kNN classification algorithm for big data. Neurocomputing, 195, 143-148.
- [7] Cheng, D., Zhang, S., Deng, Z., Zhu, Y., Zong, M. (2014, December). kNN algorithm with data-driven k value. In International Conference on Advanced Data Mining and Applications (pp. 499-512). Springer, Cham.