Decision Tree And K-Nearest Neighbor

CSE-0408 Summer 2021

Name:Rukaiya Khan Mithila
Department of Computer Science and Engineering
State University of Bangladesh (SUB)
Dhaka, Bangladesh

***Abstract*—A decision tree is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements.**

***Index Terms*—About decision Tree,decision tree classifiers And I will Try An Problem Solving With Decision Tree Using Python language-matplotlib.pyplot,pandas as pd, numpy as np.**

I. Introduction

Decision Tree : Decision tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.

Decision Trees are a type of Supervised Machine Learning (that is you explain what the input is and what the corresponding output is in the training data) where the data is continuously split according to a certain parameter. The tree can be explained by two entities, namely decision nodes and leaves. The leaves are the decisions or the final outcomes. And the decision nodes are where the data is split. Decision Trees modified An example of a decision tree can be explained using above binary tree. Let's say you want to predict whether a person is fit given their information like age, eating habit, and physical activity, etc. The decision nodes here are questions like 'What's the age?', 'Does he exercise?', 'Does he eat a lot of pizzas'? And the leaves, which are outcomes like either 'fit', or 'unfit'. In this case this was a binary classification problem (a yes no type problem).

II. Literature review

Machine Learning, Tom Mitchell, McGraw Hill, 1997.

In the next post we will be discussing about ID3 algorithm for the construction of Decision tree given by J. R. Quinlan.

III. PROPOSED METHODOLOGY

How to Construction of An Decision Tree :

A tree can be "learned" by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called recursive partitioning. The recursion is completed when the subset at a node all has the same value of the target variable, or when splitting no longer adds value to the predictions. The construction of decision tree classifier does not require any domain knowledge or parameter setting, and therefore is appropriate for exploratory knowledge discovery. Decision trees can handle high dimensional data. In general decision tree classifier has good accuracy. Decision tree induction is a typical inductive approach to learn knowledge on classification.

How to An Decision Tree Representation :

Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance. An instance is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute as shown in the above figure. This process is then repeated for the subtree rooted at the new node.

The decision tree in above figure classifies a particular morning according to whether it is suitable for playing tennis and returning the classification associated with the particular leaf.(in this case Yes or No). For example, the instance

(Outlook = Rain, Temperature = Hot, Humidity = High, Wind = Strong)

IV. Types of Decision Trees

Types of decision trees are based on the type of target variable we have. It can be of two types:

1.Categorical Variable Decision Tree: Decision Tree which has a categorical target variable then it called a Categorical variable decision tree.

2. Continuous Variable Decision Tree: Decision Tree has a continuous target variable then it is called Continuous Variable Decision Tree.

V. Important Terminology related to Decision Trees

1.Root Node: It represents the entire population or sample and this further gets divided into two or more homogeneous sets.

2.Splitting: It is a process of dividing a node into two or more sub-nodes.

3. Decision Node: When a sub-node splits into further sub-nodes, then it is called the decision node.

4.Leaf / Terminal Node: Nodes do not split is called Leaf or Terminal node.

5. Pruning: When we remove sub-nodes of a decision node, this process is called pruning. You can say the opposite process of splitting.

6.Branch / Sub-Tree: A subsection of the entire tree is called branch or sub-tree

7.Parent and Child Node: A node, which is divided into sub-nodes is called a parent node of sub-nodes whereas sub-nodes are the child of a parent node.

VI. Code

```
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.childs = 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:
no = no + 1
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:
"""if ans == 1:
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
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.childs = []
print(maxGain
)
if maxGain == 0:
if ans == 1:
root.value = 'Yes'
else:
root.value = 'No'
return root
root.value = attribute[idx]
mydict =
for i in rows:
key = data[i][idx]
if key not in mydict:
mydict[key] = 1
else:
mydict[key] += 1
newcolumns = copy.deepcopy(columns)
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.childs.append(temp)
return root
def traverse(root):
print(root.decision)
print(root.value)
n = len(root.childs)
if n ¿ 0:
for i in range(0, n):
traverse(root.childs[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. 1. Rule Set Of Decision Tree

***Abstract*—KNN (k-nearest neighbor) is an extensively used classification algorithm owing to its simplicity, ease of implementation and effectiveness. It is one of the top ten data mining algorithms, has been widely applied in various fields. KNN has few shortcomings affecting its accuracy of classification. It has large memory requirements as well as high time complexity. Several techniques have been proposed to improve these shortcomings in literature. In this paper, we have first reviewed some improvements made in KNN algorithm. Then, we have proposed our novel improved algorithm. It is a combination of dynamic selected, attribute weighted and distance weighted techniques. We have experimentally tested our proposed algorithm in Net Beans IDE, using a standard UCI dataset-Iris. The accuracy of our algorithm is improved with a blend of classification and clustering techniques. Experimental results have proved that our proposed algorithm performs better than conventional KNN algorithm.**

***Index Terms*—K-Nearest Neighbors is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining and intrusion detection. And I will Try An Problem Solving With Decision Tree Using Python language-matplotlib.pyplot,pandas as pd, numpy as np.**

VII. Introduction

9Are you venturing into machine learning? Here is a quick introduction to the simplest machine language algorithms – KNN – which will help you grasp its key dynamics.

K-Nearest Neighbors algorithm (or KNN) is one of the most used learning algorithms due to its simplicity. So what is it?

KNN is a lazy learning, non-parametric algorithm. It uses data with several classes to predict the classification of the new sample point. KNN is non-parametric since it doesn't make any assumptions on the data being studied, i.e., the model is distributed from the data.

What does it mean to say KNN is a lazy algorithm? It means it doesn't use the training data points to make any generalisation. Which implies:

You expect little to no explicit training phase,

The training phase is pretty fast,

KNN keeps all the training data since they are needed during the testing phase. Most data does not obey the typical theoretical assumptions, like when we consider a model like linear regression, which makes KNN crucial when studying data with little or no prior knowledge.

VIII. Literature review

KNN was born out of research done for the armed forces. Fix and Hodge – two officers of USAF School of Aviation Medicine – wrote a technical report in 1951 introducing the KNN algorithm.

IX. Where to use KNN

KNN can be used in both regression and classification predictive problems. However, when it comes to industrial problems, it's mostly used in classification since it fairs across all parameters evaluated when determining the usability of a technique

1. Prediction Power

2.Calculation Time

3. Ease to Interpret the Output

KNN algorithm fairs across all parameters of considerations. But mostly, it is used due to its ease of interpretation and low calculation time.

X. How does KNN works?

The k-nearest neighbor algorithm stores all the available data and classifies a new data point based on the similarity measure (e.g., distance functions). This means when new data appears. Then it can be easily classified into a well-suited category by using K- NN algorithm.

Suppose there are two classes, i.e., Class A and Class B, and we have a new unknown data point "?", so this data point will lie in which of these classes. To solve this problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the class of a particular dataset. The data point is classified by a majority vote of its neighbors, with the data point being assigned to the class most common amongst its K nearest neighbors measured by a distance function.

Consider the below diagram:

Here, we can see that if k = 3, then based on the distance function used, the nearest three neighbors of the data point is found and based on the majority votes of its neighbors, the data point is classified into a class. In the case of k = 3, for the above diagram, it's Class B. Similarly, when k = 7, for the above diagram, based on the majority votes of its neighbors, the data point is classified to Class A.

XI. KNN Advantages

Some Advantages of KNN:

1.Quick calculation time

2.Simple algorithm – to interpret

3.Versatile – useful for regression and classification

4.High accuracy – you do not need to compare with better-supervised learning models

5.No assumptions about data – no need to make additional assumptions, tune several parameters, or build a model. This makes it crucial in nonlinear data case.

XII. KNN Disadvantages

Some Disadvantages of KNN

1.Accuracy depends on the quality of the data

2.With large data, the prediction stage might be slow

3.Sensitive to the scale of the data and irrelevant features

4. Require high memory – need to store all of the training data

5.Given that it stores all of the training, it can be computationally expensive

XIII. Algorithm

The Crisp K-NN Algorithm

Let W = xx, x2, · · ·, xn be a set of n labeled samples.Thealgorithm is as follows:

BEGIN

Input y9 of unknown classification.

Set , ^{Kn}.

Initialize / = 1.

DO UNTIL (AT-nearest neighbors found)

Compute distance from y to xt.

IF (/ ¡ K) THEN

Include xt in the set of Af-nearest neighbors ELSE IF (x; is closer to y than any previous nearest neighbor) THEN

Delete farthest in the set of $-nearestneighborsIncludex, inthesetofAf-nearestneighbors.$

END IF Increment /.

BEGIN

Input x, of unknown classification.

Set K, 1 ¡ K ¡ n.

Initialize / = 1.

DO UNTIL (AT-nearest neighbors to x found) Compute distance from x to x.

I F (i ¡ K) THEN

Include JC,· in the set of AT-nearest neighbors ELSE I F (xt closer to x than any previous nearest neigh-bor)

T H E N

Delete the farthest of the -nearest neighbors Include xt in the set of -nearest neighbors.

E N D I F

E N D DO UNTIL

Initialize i = 1.

DO UNTIL (x assigned membership in all classes) Compute W,(JC) using (1).

Increment i.

END DO UNTIL

E N D

XIV. Code

Import necessary modules

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model$_s election import train_test_split$

from sklearn.datasets import load$_i ris$

import numpy as np

import matplotlib.pyplot as plt

irisData = load$_i ris()$

Create feature and target arrays

X = irisData.data

y = irisData.target

Split into training and test set

$X_t rain, X_t est, y_t rain, y_t est = train_t est_s plit(X, y, test_s ize = 0.2, random_s tate = 42)$

neighbors = np.arange(1, 9)

train$_a ccuracy = np.empty(len(neighbors))$

test$_a ccuracy = np.empty(len(neighbors))$

Loop over K values

for i, k in enumerate(neighbors):

knn = KNeighborsClassifier(n$_n eighbors = k$)

knn.fit(X$_t rain, y_t rain$)

Compute training and test data accuracy

train$_a ccuracy[i] = knn.score(X_t rain, y_t rain)$

test$_a ccuracy[i] = knn.score(X_t est, y_t est)$

Generate plot

plt.plot(neighbors, test$_a ccuracy, label = 'TestingdatasetAccuracy'$)

plt.plot(neighbors, train$_a ccuracy, label = 'TrainingdatasetAccuracy'$)

plt.legend()

plt.xlabel('n$_n eighbors'$)

plt.ylabel('Accuracy')

plt.show()

XV. How can is KNN implemented?

In the example shown above following steps are performed:

1.The k-nearest neighbor algorithm is imported from the scikit-learn package.

2.Create feature and target variables.

3.Split data into training and test data.

4.Generate a k-NN model using neighbors value.

5.Train or fit the data into the model.

6.Predict the future.

XVI. CONCLUSION

A fuzzy ÀT-NN decision rule and a fuzzy prototype decision rule have been developed along with three methods for assigning membership values to the sample sets. The fuzzy /iT-nearest neighbor and fuzzy nearest prototype algorithms developed and investigated in this report show useful results. In particular, concerning the fuzzy -nearest neighbor algorithm with fuzzy X-nearest neighbor initialization, the mem- bership assignments produced for classified samples tend to possess desirable qualities. That is, an incorrectly classified sam- ple will not have a membership in any class close to one while a correctly classified sample does possess a membership in

the correct class close to one. The fuzzy nearest prototype classifier, while not producing error rates as low as the fuzzy nearest neighbor classifier, is computationally attractive and also pro- duces membership assignments that are desirable.

ACKNOWLEDGMENT

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

REFERENCES

[1] Safavian, S. Rasoul, and David Landgrebe. "A survey of decision tree classifier methodology." IEEE transactions on systems, man, and cybernetics 21.3 (1991): 660-674.

[2] Safavian, S. R., Landgrebe, D. (1991). A survey of decision tree classifier methodology. IEEE transactions on systems, man, and cybernetics, 21(3), 660-674

[3] Safavian, S. Rasoul, and David Landgrebe. "A survey of decision tree classifier methodology." IEEE transactions on systems, man, and cybernetics 21, no. 3 (1991): 660-674.

[4] Zhang, Min-Ling, and Zhi-Hua Zhou. "ML-KNN: A lazy learning approach to multi-label learning." Pattern recognition 40.7 (2007): 2038-2048.

[5] Zhang, M. L., Zhou, Z. H. (2007). ML-KNN: A lazy learning approach to multi-label learning. Pattern recognition, 40(7), 2038-2048.

[6] Zhang, M.L. and Zhou, Z.H., 2007. ML-KNN: A lazy learning approach to multi-label learning. Pattern recognition, 40(7), pp.2038-2048.

[7] Zhang ML, Zhou ZH. ML-KNN: A lazy learning approach to multi-label learning. Pattern recognition. 2007 Jul 1;40(7):2038-48.