Assignment No: Decision tree and k-Nearest Neighbor

CSE-0408 Summer 2021

Name: Shubho Chandra Shil (ID:UG02-48-18-021)
Department of Computer Science and Engineering
State University of Bangladesh (SUB)
Dhaka, Bangladesh
email: shubho.sarker12@gmail.com

***Abstract*—Decision tree classifiers are widely recognized as one of the most well-known approaches for representing data classification in classifiers.**

n

***Index Terms*—Python, Artificial Intelligence,Decision Tree.**

I. Introduction

Technology has advanced significantly in recent years, particularly in the field of Machine Learning (ML), which is effective for minimizing human labor. A decision tree is a graphical representation of all possible solutions to a decision. These days, tree-based algorithms are the most commonly used algorithms in the case of supervised learning scenarios. They are easier to interpret and visualize with great adaptability. Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance.

Decision tree is a flowchart-like tree structure where an internal node represents feature, the branch represents a decision rule, and each leaf node represents the outcome

In this project, a thorough implementation of the most recent and most effective techniques to decision trees in several fields of machine learning that have been developed by researchers over the last three years is carried out. Making a decision tree is also a part of the process.

II. Literature Review

Lertworaprachaya et al.,2014 [1] proposed a new model for compose decision trees using interval-valued fuzzy membership values.

For diabetes mellitus prediction, Zou et al. used decision trees, Random Forest (RF), and neural network techniques. Physical research data for hospitals in Luzhou, China is included in the dataset. There are 14 different characteristics to consider. The training array extracts data from 68994 healthy humans and diabetic patients at random. To reduce dimensionality, they exploited the full significance of minimum Redundancy Maximum Relevance (mRMR) and Principal Component Analysis (PCA). In certain instances, the effects of RF, as opposed to the other classifiers, appeared to be larger. Furthermore, in the Luzhou data collection, 0.8084 is the best result.

III. Decision Tree Algorithm

Decision trees are a simple classification tool capable of separating records of data into specific categories by proposing a series of questions.Decision trees are commonly used due to many factors, including their relatively small learning curve for interpretability

Decision trees are a strong tool that may be utilized in a variety of domains, including machine learning, image processing, and pattern recognition. DT is a sequential model that effectively and cohesively connects a series of fundamental tests in which a numeric feature is compared to a threshold value in each test. The numerical weights in the neural network of connections between nodes are far more difficult to construct than the conceptual rules. DT is primarily used for grouping purposes. Furthermore, in Data Mining, DT is an often used classification model. Each tree is made up of its nodes and branches. Each subset defines a value that the node can take, whereas each node represents features in a category to be categorised. Decision trees offer a wide range of applications due to their straightforward analysis and precision across many data types.

IV. Types of Nodes

A decision tree consists of three types of nodes:
Decision nodes — typically represented by squares
Chance nodes — typically represented by circles
End nodes — typically represented by triangles

V. Advantages

A. Are simple to understand and interpret. People are able to understand decision tree models after a brief explanation.

B. Help determine worst, best and expected values for different scenarios.

C. Use a white box model. If a given result is provided by a model.

D. Can be combined with other decision techniques.

VI. Disadvantages

A.They are unstable, meaning that a small change in the data can lead to a large change in the structure of the optimal decision tree.

B. They are often relatively inaccurate.

C.Calculations can get very complex, particularly if many values are uncertain and/or if many outcomes are linked.

VII. Conclusion

This assignment is based on a graphic representation of a decision tree. A data-set is given for the training and visualization of this decision tree.

Acknowledgment

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

VIII. Code

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import math
import copy
dataset = pd.read_csv('tennis.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()
```

***Abstract*—K-Nearest Neighbors method is one of methods used for classification which calculate a value to find out the closest in distance. In this Assignment we are going to implement K-Nearest Neighbors using Jupyter Notebook.**

n

***Index Terms*—Machine Learning, Python, K nearest neighbors..**

I. Introduction

The K-Nearest-Neighbors (KNN) is a nonparametric classification algorithm, i.e. it does not make any presumptions on the elementary dataset. It is known for it's simplicity and effectiveness. It is a supervised learning algorithm. A labeled training dataset is provided where the data points are categorized into various classes, so that the class of the unlabeled data can be predicted.In Classification, different characteristics determine the class to which the unlabeled data belongs. KNN is mostly used as a classifier. It is used to classify data based on closest or neighbouring training examples in a given region.

In this assignment, we will implement another widely used machine learning classification technique called K-nearest neighbors (KNN). Our focus will be primarily on how does the algorithm work and how does the input parameter affects the output/prediction.

II. Literature Review

Along the years, a great effort was done in the scientific community in order to solve or mitigate the imbalanced dataset problem. Specifically for KNN, there are several balancing methods based on this algorithm. This section will provide a bibliographic review about the KNN and its derivate algorithms for dataset balancing.

III. KNN Algorithm

We can implement a KNN model by following the below steps:

1) Load the data 2) Initialise the value of k 3) For getting the predicted class, iterate from 1 to total number of training data points a) Calculate the distance between test data and each row of training data. Here we will use Euclidean

distance as our distance metric since it's the most popular method. The other metrics that can be used are Chebyshev, cosine, etc.

b) Calculate the distance between test data and each row of training data. Here we will use Euclidean distance as our distance metric since it's the most popular method. The other metrics that can be used are Chebyshev, cosine, etc.

c) Sort the calculated distances in ascending order based on distance values

d) Get top k rows from the sorted array

e) Get the most frequent class of these rows

f) Return the predicted class

IV. Advantages

A. No Training Period: KNN is called Lazy Learner.It does not learn anything in the training period. It does not derive any discriminative function from the training data

In future, what you bring in your project and the idea of your work.

B. Since the KNN algorithm requires no training before making predictions, new data can be added seamlessly which will not impact the accuracy of the algorithm.

C. KNN is very easy to implement. There are only two parameters required to implement KNN i.e. the value of K and the distance function.

V. Disadvantages

A. Does not work well with large dataset.

B. Does not work well with high dimensions.

C. Need feature scaling: We need to do feature scaling before applying KNN algorithm to any dataset.

D. Sensitive to noisy data, missing values and outliers: KNN is sensitive to noise in the dataset.

VI. Conclusion

This assignment is based on a graphic representation of a KNN model accuracy. A data-set is given for the training and visualization of this KNN model accuracy.

VII. Conclusion

This assignment is based on a graphic representation of a KNN model accuracy. A data-set is given for the training and visualization of this KNN model accuracy.

Acknowledgment

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

VIII. Code

```
!/usr/bin/env python
In[14]:
Import necessary modules
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selectionimporttrain_test_split
from sklearn.datasets import load_iris
Loading data
irisData = load_iris()
Create feature and target arrays
X = irisData.data
y = irisData.target
Split into training and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
knn = KNeighborsClassifier(n_neighbors = 7)
knn.fit(X_train, y_train)
Predict on dataset which model has not seen before
print(knn.predict(X_test))
In[13]:
Import necessary modules
from sklearn.neighbors import KNeighborsClassifier
```

from sklearn.model$_selectionimporttrain_test_split$
from sklearn.datasets import load$_iris$
Loading data
irisData = load$_iris()$
Create feature and target arrays
X = irisData.data y = irisData.target
Split into training and test set
X$_train, X_test, y_train, y_test$ = $train_test_split(X, y, test_size = 0.2, random_state = 42)$
knn = KNeighborsClassifier(n$_neighbors = 7)$
knn.fit(X$_train, y_train)$
Calculate the accuracy of the model
print(knn.score(X$_test, y_test)$)
In[12]:
Import necessary modules
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model$_selectionimporttrain_test_split$
from sklearn.datasets import load$_iris$
import numpy as np
import matplotlib.pyplot as plt
irisData = load$_iris()$
Create feature and target arrays
X = irisData.data
y = irisData.target
Split into training and test set
X$_train, X_test, y_train, y_test$ = $train_test_split(X, y, test_size = 0.2, random_state = 42)$
neighbors = np.arange(1, 9)
train$_accuracy = np.empty(len(neighbors))$
test$_accuracy = np.empty(len(neighbors))$
Loop over K values
for i, k in enumerate(neighbors):
knn = KNeighborsClassifier(n$_neighbors = k)$
knn.fit(X$_train, y_train)$
Compute training and test data accuracy
train$_accuracy[i] = knn.score(X_train, y_train)$
test$_accuracy[i] = knn.score(X_test, y_test)$
Generate plot
plt.plot(neighbors, test$_accuracy, label =' Testing dataset Accuracy')$
plt.plot(neighbors, train$_accuracy, label =' Training dataset Accuracy')$
plt.legend()
plt.xlabel('n$_neighbors')$
plt.ylabel('Accuracy')
plt.show()
In[ ]: