Artificial Intelligence Lab Final

CSE-0408 Summer 2021

SANJIDA AKTER
Department of Computer Science and Engineering
State University of Bangladesh (SUB)
Dhaka, Bangladesh
Muktaeva11@gmail.com

I. K-Nearest Neighbor Problem

A data set with lots of different points and labelled data is the ideal to use. The best languages to use with KNN are R and python. To find the most accurate results from your data set, you need to learn the correct practices for using this algorithm

II. adventage of K-NN

Quick calculation time Simple algorithm – to interpret Versatile – useful for regression and classification High accuracy – you do not need to compare with better-supervised learning models 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.

III. Disadvantage of KNN

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

IV. code

!/usr/bin/env python coding: utf-8
In[14]:
Import necessary modules
from sklearn.neighbors import
KNeighborsClassifier
from sklearn.model$_selectionimport$
train$_test_split$
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$_selectionimport$
train$_test_split$
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$_selectionimport$
train$_test_split$
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$_neighbor$
s=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$_accuracy, label = '$
$Trainingdataset Accuracy'$)
plt.legend()
plt.xlabel('n$_neighbors'$)
plt.ylabel('Accuracy')
plt.show()
In[ ]:

V. Decision Tree

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.

Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal, but are also a popular tool in machine learnin. A decision tree is a support tool with a tree-like structure that models probable outcomes, cost of resources, utilities, and possible consequences. Decision trees provide a way to present algorithms with conditional control statements. They include branches that represent decision-making steps that can lead to a favorable result. Decision trees are one of the best forms of learning algorithms based on various learning methods. They boost predictive models with accuracy, ease in interpretation, and stability. The tools are also effective in fitting non-linear relationships since they can solve data-fitting challenges, such as regression and classifications.

VI. advantage of Decision tree

1. Easy to read and interpret One of the advantages of decision trees is that their outputs are easy to read and interpret without requiring statistical knowledge. For example, when using decision trees to present demographic information on customers, the marketing department staff can read and interpret the graphical representation of the data without requiring statistical knowledge.

The data can also generate important insights on the probabilities, costs, and alternatives to various strategies formulated by the marketing department.

2. Easy to prepare Compared to other decision techniques, decision trees take less effort for data preparation. However, users need to have ready information to create new variables with the power to predict the target variable. They can also create classifications of data without having to compute complex calculations. For complex situations, users can combine decision trees with other methods.

3. Less data cleaning required Another advantage of decision trees is that there is less data cleaning required once the variables have been created. Cases of missing values and outliers have less significance on the decision tree's data.

VII. Disadvantages of Decision Trees

1. Unstable nature One of the limitations of decision trees is that they are largely unstable compared to other decision predictors. A small change in the data can result in a major change in the structure of the decision tree, which can convey a different result from what users will get in a normal event. The resulting change in the outcome can be managed by machine learning algorithms, such as boosting and bagging.

2. Less effective in predicting the outcome of a continuous variable In addition, decision trees are less effective in making predictions when the main goal is to predict the outcome of a continuous variable. This is because decision trees tend to lose information when categorizing variables into multiple categories.

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) ret
urn 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()
```

IX. SUMMARY OF DECISION TREE

Decision trees are used for handling non-linear data sets effectively. The decision tree tool is used in real life in many areas, such as engineering, civil planning, law, and business. Decision trees can be divided into two types; categorical variable and continuous variable decision trees.

ACKNOWLEDGMENT

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