Final Assignment:Decision Tree and K-Nearest Neighbor Problem

CSE-0408 Summer 2021

Name:Nishu Akter
Department of Computer Science and Engineering
State University of Bangladesh (SUB)
Dhaka, Bangladesh
email:nishuakter309@gmail.com

***Abstract*—code-1: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.**

code-2:K-nearest neighbor (KNN) algorithm is a nonparametric classification method often used in pattern classification for handling binary and multiclass problems in different domains. Although KNN algorithm is a simple data mining approach but it may give inaccurate results when training datasets are imbalanced and incomplete.

n

***Index Terms*—The word mostly used in your report.**

I. Introduction

code-1:A decision tree is a tree-like graph with nodes representing the place where we pick an attribute and ask a question; edges represent the answers the to the question; and the leaves represent the actual output or class label. They are used in non-linear decision making with simple linear decision surface.

code-2:The K-nearest neighbors (KNN) algorithm is a type of supervised machine learning algorithms. KNN is extremely easy to implement in its most basic form, and yet performs quite complex classification tasks. KNN is a non-parametric learning algorithm, which means that it doesn't assume anything about the underlying data. The KNN algorithm is one of the simplest of all the supervised machine learning algorithms. It simply calculates the distance of a new data point to all other training data points.

II. Varients

code-1:Types of Decisions tree There are two main types of decision trees that are based on the target variable, i.e., categorical variable decision trees and continuous variable decision trees.

1. Categorical variable decision tree A categorical variable decision tree includes categorical target variables that are divided into categories. For example, the categories can be yes or no. The categories mean that every stage of the decision process falls into one category, and there are no in-betweens.

2. Continuous variable decision tree A continuous variable decision tree is a decision tree with a continuous target variable. For example, the income of an individual whose income is unknown can be predicted based on available information such as their occupation, age, and other continuous variables.

code-2: There are many variants of KNN algorithm proposed in previously done studies which tried to overcome these shortcomings. Some of them are described in the subsequent part of this section.
A. Locally Adaptive KNN
B. Weight Adjusted KNN
C.Adaptive KNN
D:KNN with Shared Nearest Neighbours
E:KNN with Mahalanobis Metric

III. Algorithm step

code-1: Step-1: Begin the tree with the root node, says S, which contains the complete dataset.
Step-2: Find the best attribute in the dataset using Attribute Selection Measure (ASM).
Step-3: Divide the S into subsets that contains possible values for the best attributes.
Step-4: Generate the decision tree node, which contains the best attribute.
Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

code-2: Step-1: Select the number K of the neighbors
Step-2: Calculate the Euclidean distance of K number of neighbors
Step-3: Take the K nearest neighbors as per the calculated Euclidean distance.

Step-4: Among these k neighbors, count the number of the data points in each category.
Step-5: Assign the new data points to that category for which the number of the neighbor is maximum.
Step-6: Our model is ready.

IV. Advantages and Disadvantages

code-1:Advantages of the Decision Tree It is simple to understand as it follows the same process which a human follow while making any decision in real-life. It can be very useful for solving decision-related problems. It helps to think about all the possible outcomes for a problem. There is less requirement of data cleaning compared to other algorithms.

Disadvantages of the Decision Tree The decision tree contains lots of layers, which makes it complex. It may have an overfitting issue, which can be resolved using the Random Forest algorithm. For more class labels, the computational complexity of the decision tree may increase.

code-2:Advantages of KNN Algorithm: It is simple to implement. It is robust to the noisy training data It can be more effective if the training data is large.
Disadvantages of KNN Algorithm: Always needs to determine the value of K which may be complex some time. The computation cost is high because of calculating the distance between the data points for all the training samples.

V. Code

code-1: import numpy as np
from itertools import groupby
import math
import collections
from copy import deepcopy
import pickle

class TreeNode:
def ${}_{init}{}_{(self,split,col_{i}ndex):self.col_{i}d=col_{i}ndexself.split_{v}alue=splitself.parent=Noneself.left=Noneself.right=None}$
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$_{t}ree(trainData, [])$
$saveTree(self.treemodel)$

def accuracy$_{c}onfusion_{m}atrix(self, testData)$:
$printsthetreeconfusionmatrixalongwiththeaccuracy$
$build_{c}onfusion_{m}atrix(self.treemodel, testData)$

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
def calcInfoGain(data,col,split):
totalLen = len(data)
infoGain = entropy(data)

left$_{d}ata, right_{d}ata = getDataSplit(data, split, col)$

infoGain = infoGain- (len(left$_{d}ata)/totalLen *$
$entropy(left_{d}ata))$
$infoGain = infoGain - (len(right_{d}ata)/totalLen *$
$entropy(right_{d}ata))$

return infoGain,left$_{d}ata, right_{d}ata$

def getDataSplit(data, split, col):
l$_{d}ata = []$
$r_{d}ata = []$

for val in data:
if(val[col]¡split):
l$_{d}ata.append(val)$
$else$:
$r_{d}ata.append(val)$

return l$_{d}ata, r_{d}ata$

calculates the entropy of the data set provided
def entropy(data):
totalLen = len(data)
entropy = 0

group$_by_class = groupby(data, lambda x : x[5])$
$for key, group in group_by_class :$
$grp_len = len(list(group))$
$entropy+ = -(grp_len/totalLen) * math.log((grp_len/totalLen), 2) return entropy$

this method builds the decision tree recursively until the leaf nodes are reached
def build$_tree(data, parent_data) :$
$code to find out if the class variable is all one value$
$count = 0;$
$group_by_class = groupby(data, lambda x : x[5])$

finds out if all the instances have the same class or not for key,group in group$_by_class :$
$count = count + 1;$

if same class for all instances then return the leaf node class value
if(count==1):
return data[0][5];

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(bestsplit[1],bestsplit[0])
node.left= build$_tree(bestsplit[2], data)$
$node.right = build_tree(bestsplit[3], data)$
$return node$

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$_id] <= tree.split_value :$
$return classify(tree.left, row)$
$else :$

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$_len = len(data)$
$num_correct_instances = 0;$
$num_incorrect_instances = 0;$

map required to build the confusion matrix map='B':0,'G':1,'M':2, 'N':3

for row in data:
actual$_class = row[5]$
$predicted_class = classify(tree, row)$
$if(actual_class == predicted_class) :$
$num_correct_instances = num_correct_instances + 1 confusion_mat[map.get(actual_class)][map.get(actual_class)] = confusion_mat[map.get(actual_class)][map.get(actual_class)] + 1$
$else :$
$num_incorrect_instances = num_incorrect_instances + 1 confusion_mat[map.get(actual_class)][map.get(predicted_class)] = confusion$

print("Accuracy of the model:",(num$_correct_instances/total_len) * 100, "print("Correct instances", num_correct_instances)$
$print("Incorrect instances", num_incorrect_instances)$

print$_map = 0 :' B', 1 :' G', 2 :' M', 3 :' N'$
$print("Confusion Matrix : ")$

print(" B G M N")

ind=0;
printing matrix
for row in confusion$_mat :$
$print(print_map.get(ind), "", row)$
$ind+ = 1$

code-2:!/usr/bin/env python
coding: utf-8

In[14]:

Import necessary modules
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model$_selection import train_test_split$
$from sklearn.datasets import load_iris$

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$_selectionimporttrain_test_split$
$fromsklearn.datasetsimportload_iris$

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$_selectionimporttrain_test_split$
$fromsklearn.datasetsimportload_iris$
$importnumpyasnp$
$importmatplotlib.pyplotasplt$

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$_accuracy, label$ $='$
$TestingdatasetAccuracy')$
$plt.plot(neighbors, train_accuracy, label$ $='$
$TrainingdatasetAccuracy')$

plt.legend()
plt.xlabel('n$_neighbors')$
$plt.ylabel('Accuracy')$
$plt.show()$

In[ ]:

VI. Conclusion

code-1:Decision trees assist analysts in evaluating upcoming choices. The tree creates a visual representation of all possible outcomes, rewards and follow-up decisions in one document. . Decision tree analysis involves making a tree-shaped diagram to chart out a course of action or a statistical probability analysis.

code-2:KNN is a simple yet powerful classification algorithm. It requires no training for making predictions, which is typically one of the most difficult parts of a machine learning algorithm. The KNN algorithm have been widely used to find document similarity and pattern recognition. It has also been employed for developing recommender systems and for dimensionality reduction and pre-processing steps for computer vision, particularly face recognition tasks.

Acknowledgment

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

References

[1] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.
[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.
[3] I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.
[4] K. Elissa, "Title of paper if known," unpublished.
[5] R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.
[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].
[7] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.