Final Report

CSE-0408 Summer 2021

Bm Pasha
Department of Computer Science and Engineering
State University of Bangladesh (SUB)
Dhaka, Bangladesh
email : bmpasha72@gmail.com

***Abstract*——This paper introduced for Decision tree and KNN (k-nearest neighbors algorithm) using Python language.**

n

***Index Terms*—Language : Python**

I. Introduction

Decision tree A decision tree is a flowchart-like structure in which each internal node represents a ”test” on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). k-nearest neighbors The k-nearest neighbors algorithm is a simple, supervised machine learning algorithm that can be used to solve both classification and regression problems. It’s easy to implement and understand, but has a major drawback of becoming significantly slows as the size of that data in use grows.

II. Literature Review

I am using python to solving the two algorithms Decision tree and k-nearest neighbors

III. Proposed Methodology

• To gain insights of supervised and unsupervised machine learning techniques; • To be able to implement simple classification and regression algorithms using Python Libraries.

IV. KNN Advanteges and Disadvatages

Advantages • Quick calculation time. • Simple algorithm – to interpret. • Versatile – useful for regression and classification. Disadvantages • Does not work well with large dataset. • Does not work well with high dimensions.

V. Decision Tree Advantages and disadvantages

Advantages – Does not require normalization of data. – Does not require scaling of data as well. Disadvantages – Often involves higher time to train the model. – Training is relatively expensive as the complexity and time has taken are more.

VI. Knn Algorithm Pseudo Code

– Calculate “d(x, xi)” i =1, 2, ..., n; where d denotes the Euclidean distance between the points. – Arrange the calculated n Euclidean distances in non decreasing order. – Let k be a +ve integer, take the first k distances from this sorted list. – Find those k-points corresponding to these k distances. – Let ki denotes the number of points belonging to the ith class among k points i.e. k 0 – If ki ¿kj i j then put x in class i.

VII. Conclusion and Future Work

The BFS algorithm is useful for analyzing the nodes in a graph and constructing the shortest path of traversing through these.

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.

VIII. Decision Tree Code

In [2]:
```
import numpy as np
from itertools import groupby
import math
import collection
import pickle
```

In [ ]:
```
class TreeNode:
    def __init__(self,split,col_
        self.col_id= col_index
        self.split_value= split
        self.parent=None
        self.left= None
        self.right= None
```

In [4]:
```
class Tree():

    def __init__(self):
        self.treemodel = None

    def train(self,trainData):
        #Attributes/Last Column
        self.createTree(trainDat
    def createTree(self,trainDat
        #create the tree
        self.treemodel=build_tre
        saveTree(self.treemodel)

    def accuracy_confusion_matri
        #prints the tree confusi
        build_confusion_matrix(s

    #returns the best split on t
#with the splitted dataset and c
def getBestSplit(data):
    #set the max information gai
    maxInfoGain = -float('inf')

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

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

    #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
```

```
def build_tree(data,parent_data):
    #code to find out if the class variable is a
    count=0;
    group_by_class= groupby(data, lambda x:x[5])

#finds out if all the instances have the same cl
    for key,group in group_by_class:
        count=count+1;

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

    elif(len(data)==0):
        #this counts all the column class variab
        return collections.Counter(np.asarray(da

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

#this method creates a confusion matrix and find
def build_confusion_matrix(tree, data):
    confusion_mat = [[0 for row in range(4)]for c

    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_ins
```

```
    for row in data:
        actual_class = row[5]
        predicted_class=classify(tree, row)
        if(actual_class==predicted_class):
            num_correct_instances=num_correct
            confusion_mat[map.get(actual_clas
        else:
            num_incorrect_instances=num_incor
            confusion_mat[map.get(actual_clas

    print("Accuracy of the model:",(num_corre
    print("Correct instances",num_correct_ins
    print("Incorrect instances",num_incorrect
    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
```

IX. K-NN Algorithm Code

```
In [1]: from collections import Counter
        import math

        def knn(data, query, k, distance_fn, ch
            neighbor_distances_and_indices = []

            # 3. For each example in the data
            for index, example in enumerate(dat
                # 3.1 Calculate the distance be
                # example from the data.
                distance = distance_fn(example[

                # 3.2 Add the distance and the
                neighbor_distances_and_indices.

            # 4. Sort the ordered collection of
            # smallest to largest (in ascending
            sorted_neighbor_distances_and_indic

            # 5. Pick the first K entries from
            k_nearest_distances_and_indices = s

            # 6. Get the labels of the selected
            k_nearest_labels = [data[i][-1] for

            # 7. If regression (choice_fn = mea
            # 8. If classification (choice_fn =
            return k_nearest_distances_and_indi

        def mean(labels):
            return sum(labels) / len(labels)

        def mode(labels):
            return Counter(labels).most_common(

        def euclidean_distance(point1, point2):
            sum_squared_distance = 0
            for i in range(len(point1)):
                sum_squared_distance += math.po
            return math.sqrt(sum_squared_distan
```

```
def main():
    '''
    # Regression Data
    #
    # Column 0: height (inches)
    # Column 1: weight (pounds)
    '''
    reg_data = [
        [65.75, 112.99],
        [71.52, 136.49],
        [69.40, 153.03],
        [68.22, 142.34],
        [67.79, 144.30],
        [68.70, 123.30],
        [69.80, 141.49],
        [70.01, 136.46],
```

```python
    clf_data = [
       [22, 1],
       [23, 1],
       [21, 1],
       [18, 1],
       [19, 1],
       [25, 0],
       [27, 0],
       [29, 0],
       [31, 0],
       [45, 0],
    ]
    # Question:
    # Given the data we have, does a 33 year old like pineapples on their pizza?
    clf_query = [33]
    clf_k_nearest_neighbors, clf_prediction = knn(
        clf_data, clf_query, k=3, distance_fn=euclidean_distance, choice_fn=mode
    )

if __name__ == '__main__':
    main()
```