Final Report: A. K-Nearest Neighbors Problem & B. Decision Tree Problem

CSE-0408 Summer 2021

Md.Rafiul Alam Durjoy
Department of Computer Science and Engineering
State University of Bangladesh (SUB)
Dhaka, Bangladesh
mdurjoy148@gmail.com

***Abstract*—A. K-nearest-neighbor (KNN) classification is one of the most fundamental and simple classification methods and should be one of the first choices for a classification study when there is little or no prior knowledge about the distribution of the data.**

B. 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*—KNN, DECISION TREE, PYTHON**

I. Introduction

A. K-Nearest Neighbors:

The k-nearest neighbour method (KNN) is a classical, widely used prediction method in the fields of machine learning and data mining. This is mostly due to its simplicity and versatility in predicting many different types of data. It was first introduced in 1951 by E. Fix and J. L. Hodges in their unpublished report for the US Air Force School of Aviation Medicine. In 1967 Cover and Hart (1967) formalized the original idea and discovered the main properties of this method. KNN is an instance-based, or lazy, method since it does not require building a model to represent the underlying statistics and distributions of the original training data. Instead, it works directly on the actual instances of the training data. This section provides an overview of current uses of the KNN method and how it compares to other machine learning methods. The KNN method is defined and its extensions are detailed. Furthermore, our evaluation methodology of the results is outlined.

B. 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). The paths from root to leaf represent classification rules.

II. Literature Review

A. K-Nearest Neighbors:

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. Also, the random oversampling and undersampling methods, the class overlapping problem, and evaluation measures will be reviewed.

B. Decision Tree:

The Decision Tree is an important classification method in data mining classification. It is commonly used in marketing, surveillance, fraud detection, scientific discovery. As the classical algorithm of the decision tree ID3, C4.5, C5.0 algorithms have the merits of high classifying speed, strong learning ability and simple construction.

III. Proposed Methodology

A. K-Nearest Neighbors:

kNN stands for k-Nearest Neighbours. It is a supervised learning algorithm.kNN is very simple to implement and is most widely used as a first step in any machine learning setup. It is often used as a benchmark for more complex classifiers such as Artificial Neural Networks (ANN) and Support Vector Machines (SVM).

Breaking it Down – Pseudo Code of KNN:
1.Calculate the distance between test data and each row of training data.
2.Sort the calculated distances in ascending order based on distance values.
3.Get top k rows from the sorted array.
4.Get the most frequent class of these rows.
5.Return the predicted class.

KNN example:
KNN also known as K-nearest neighbour is a supervised

and pattern classification learning algorithm which helps us find which class the new input(test value) belongs to when k nearest neighbours are chosen and distance is calculated between them.

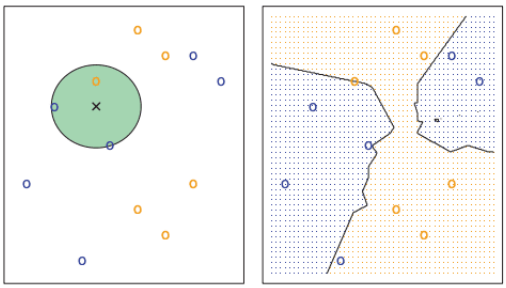

Fig. 1. K-Nearest Neighbors.

Here is step by step on how to compute K-nearest neighbors KNN algorithm:
1.Determine parameter K = number of nearest neighbors.
2.Calculate the distance between the query-instance and all the training samples.
3.Sort the distance and determine nearest neighbors based on the K-th minimum distance.

Code For KNN:

In [2]:
```
#!/usr/bin/env python
# coding: utf-8


# 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))
```

Fig. 2. Code Knn.

```
# 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)

# Calculate the accuracy of the model
print(knn.score(X_test, y_test))


# Import necessary modules
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
import numpy as np
import matplotlib.pyplot as plt

irisData = load_iris()
```

Fig. 3. Code Knn.

```
# 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()
```

Fig. 4. Code Knn.

```
[1 0 2 1 1 0 1 2 2 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0]
0.9666666666666667
```

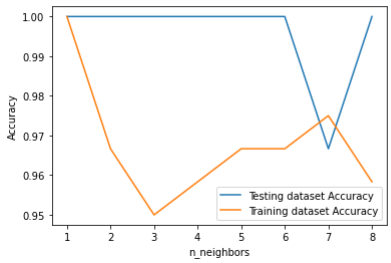

Fig. 5. Output Knn.

B. Decision Tree:

Decision trees are powerful and popular tools for classification and prediction. Decision trees represent rules, which can be understood by humans and used in knowledge system such as database. A decision tree is a hierarchical model for supervised learning whereby the local region is identified in a sequence of recursive splits in a smaller number of steps. A decision tree is composed of internal decision nodes decision node and terminal leaves. Each decision node m implements a test function fm(x) with discrete outcomes labelling the branches. Given an input, at each node, a test is applied and one of the branches is taken depending on the outcome. This process starts at the root and is repeated recursively until a leaf node is hit, at which point the value written in the leaf constitutes the output. A decision tree is also a nonparametric model in the sense that we do not assume any parametric form for the class densities and the tree structure is not fixed a priori but the tree grows, branches and leaves are added, during learning depending on the complexity of the problem inherent in the data.
Decision tree is a classifier in the form of a tree structure which consists of:
Decision node: specifies a test on a single attribute.
Leaf node: indicates the value of the target attribute.
Edge: split of one attribute.
Path: a disjunction of test to make the final decision.

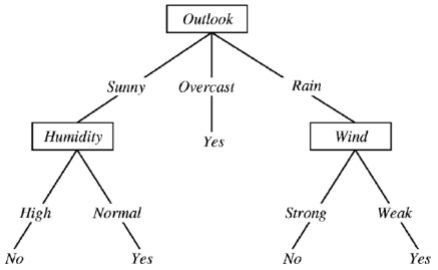

Fig. 6. A Decision Tree.

Decision trees classify instances or examples by starting at the root of the tree and moving through it until a leaf node.

Advantages of Decision trees:
1. Easy to interpret the decision rules.
2. Nonparametric so it is easy to incorporate a range of numeric or categorical data layers and there is no need to select unimodal training data.
3. Robust with regard to outliers in training data.

Disadvantages of Decision Trees:
1. Decision trees tend to over fit training data which can give poor results when applied to the full data set.
2. Not possible to predict beyond the minimum and maximum limits of the response variable in the training data.

In [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_index):
        self.col_id= col_index
        self.split_value= split
        self.parent=None
        self.left= None
        self.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_tree(trainData,[])
        saveTree(self.treemodel)
```

Fig. 7. code for decision tree.

```
    def accuracy_confusion_matrix(self,testData):
        #prints the tree confusion matrix along with the accuracy
        build_confusion_matrix(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
```

Fig. 8. code for decision tree.

```
    totalLen = len(data)
    infoGain = entropy(data)

    left_data, right_data =getDataSplit(data,split,col)

    infoGain = infoGain- (len(left_data)/totalLen * entropy(left_data))
    infoGain = infoGain- (len(right_data)/totalLen * entropy(right_data))

    return infoGain,left_data,right_data


def getDataSplit(data, split, col):
    l_data=[]
    r_data=[]

    for val in data:
        if(val[col]<split):
            l_data.append(val)
        else:
            r_data.append(val)

    return l_data,r_data

#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
```

Fig. 9. code for decision tree.

```
    #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];

    elif(len(data)==0):
        #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:
        return classify(tree.right, row)
```

Fig. 10. code for decision tree.

```
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
    else:
        num_incorrect_instances=num_incorrect_instances+1
        confusion_mat[map.get(actual_class)][map.get(predicted_class)]=confusion_mat[map.get(actual_class)][map.get(predicte

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
```

Fig. 11. code for decision tree.

IV. Conclusion

A. K-Nearest Neighbors:

K-Nearest neighbor classification is a general technique to learn classification based on instance and do not have to develop an abstract model from the training data set. However the classification process could be very expensive because it needs to compute the similiary values individually between the test and training examples.

B. Decision Tree:

Classification in data mining assigns data samples to target classes. For example identify loan applicants as medium, high, low in a bank data samples. The goal is to predict accurately the class to which the data samples belong to. Decision trees are prone to overfitting as we can see in the above experiment where the tree probably overfit the data giving an accuracy of 100.

Acknowledgment

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

References

[1] Wang, H.: Nearest Neighbours without k: A Classification Formalism based on Probability, technical report, Faculty of Informatics, University of Ulster, N.Ireland, UK (2002)

[2] Introduction to Data Mining, Pang-Ning Tan, Michael Steinbach, Vipin Kumar, Published by Addison Wesley.

[3] "Decision Tree: Towards an Alternative to Deep Neural Networks" by Zhi-Hua Zhou and Ji Feng.