Final Report

CSE-0408 Summer 2021

Name:Nymul Islam Moon
ID:UG02-47-18-017
Department of Computer Science and Engineering
State University of Bangladesh (SUB)
Dhaka, Bangladesh
Gmail:towkir1997islam@gmail.com

***Abstract*—This paper introduced for Decision tree and KNN (k-nearest neighbors algorithm) using Python language.**
***Index Terms*——Languages : 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 Advantages and Disadvantages

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 Pseudocode:

- 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. Figures

*New Text Document.txt - Notepad

File Edit Format View Help

```
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))
[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
In [13]:
# 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
```

Fig. 1. code

*New Text Document.txt - Notepad

File Edit Format View Help

```
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()
```

Fig. 2. code