REPORT ON ANN PROJECT

Artificial Neural Networks lab		GITAM (DEEMED TO BE UNIVERSITY) VISAKHAPATNAM • HYDERABAD • BENGALURU	
Semester	**V**	**Subject**	**Artificial neural networks**
Section	**B22**	**Code**	**19EAI331P**
Name	**K. Yaswanth Kumar**		
Registration No	**122010322024**	**Branch**	**CSE -CS / AIML**
Year	**2022**	**Date**	**22-10-22**

DECISION TREE ALGORITHM

COVID-19 DATASET

K. Yaswanth Kumar

122010322024

B-22, AIML

IMPLEMENTATION USING DECISION TREE ALGORITHM ON DATASET COVID-19

AIM:

Implementation of decision tree algorithm on covid-19 dataset.

Pre-processing:

- In general, every dataset requires pre-processing to get accurate result or maximum accuracy.
- For our dataset covid-19, we applied three methods of pre-processing techniques:
    1. Eliminating Null values/checking null values
    2. Encoding categorial values
    3. Normalization

1. Eliminating Null values/checking null values
    - To check whether there are null values in our dataset, we used a function isnull()
2. Encoding categorial values
    - No need to do this because there are no categorial values in our dataset, all values are binary itself.

3. Normalization
    - For normalization of a specific column called location, we applied it using a function called normalize()
    - As a result, location column is being normalized.

Code for pre-processing:

Code for importing the dataset:

```
data_set = pd.read_excel('covid_19.xlsx')
X = data_set.iloc[:, :-1].values
y = data_set.iloc[:, -1].values
print(X)
```

Code for checking null values:

```
data_set.isnull()
```

The output here is, false in the whole dataset in every column and row.

Code for Normalization:

```
data_set = pd.read_excel('covid_19.xlsx')
x_array = np.array(data_set['location'])
normalized_arr = preprocessing.normalize([x_array])
print(normalized_arr)
```

The output here is, normalized values displayed in location column

Code for the whole pre-processing methods:

```
data_set = pd.read_excel('covid_19.xlsx')
X = data_set.iloc[:, :-1].values
y = data_set.iloc[:, -1].values
print(X)
data_set.isnull()
data_set = pd.read_excel('covid_19.xlsx')
x_array = np.array(data_set['location'])
normalized_arr = preprocessing.normalize([x_array])
print(normalized_arr)
```

Decision tree on dataset – covid-19

- Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems.

- It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

- In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node.

- Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
- The decisions or the test are performed on the basis of features of the given dataset.

- It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.

- It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.

Code:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report


# Function for importing Dataset
def importdata():

    balance_data = pd.read_excel('covid_19.xlsx')
    #The dataset length i.e, number of rows
    #print ("Dataset Length: ", len(balance_data))
    #The dataset shape i.e, number of rows and columns
    #print ("Dataset Shape: ", balance_data.shape)
    # dataset observations
    #print ("Dataset: ",balance_data.head())
    return balance_data

# Function to split the dataset
def splitdataset(balance_data):

    # Separating the target variable
    X = balance_data.values[:,:-1]
    Y = balance_data.values[:, -1]
    # Splitting the dataset into train and test
    X_train, X_test, y_train, y_test = train_test_split(
    X, Y, test_size = 0.2, random_state = 100)
    return X, Y, X_train, X_test, y_train, y_test

# Function to perform training with giniIndex.
def train_using_gini(X_train, X_test, y_train):
```

```
    # Creating the classifier object
    clf_gini = DecisionTreeClassifier(criterion = "gini",
            random_state = 100,max_depth=3, min_samples_leaf=2)

    # Performing training
    clf_gini.fit(X_train, y_train)
    return clf_gini

# Function to perform training with entropy.
def train_using_entropy(X_train, X_test, y_train):

    # Decision tree with entropy
    clf_entropy = DecisionTreeClassifier(
            criterion = "entropy", random_state = 100,
            max_depth = 3, min_samples_leaf = 5)
    # Performing training
    clf_entropy.fit(X_train, y_train)
    return clf_entropy

# Function to make predictions
def prediction(X_test, clf_object):

    # Predicton on test with giniIndex
    y_pred = clf_object.predict(X_test)
    return y_pred

# Function to calculate accuracy
def cal_accuracy(y_test, y_pred):

    #printing the accuracy
    accuracy_accquired =(accuracy_score(y_test,y_pred))
    print ("Accuracy : ",accuracy_accquired*100)

    #printing the report
    print("Report : ",classification_report(y_test, y_pred))
# Driver code
def main():

    # Building Phase
    data = importdata()
    X, Y, X_train, X_test, y_train, y_test = splitdataset(data)
    clf_gini = train_using_gini(X_train, X_test, y_train)
```

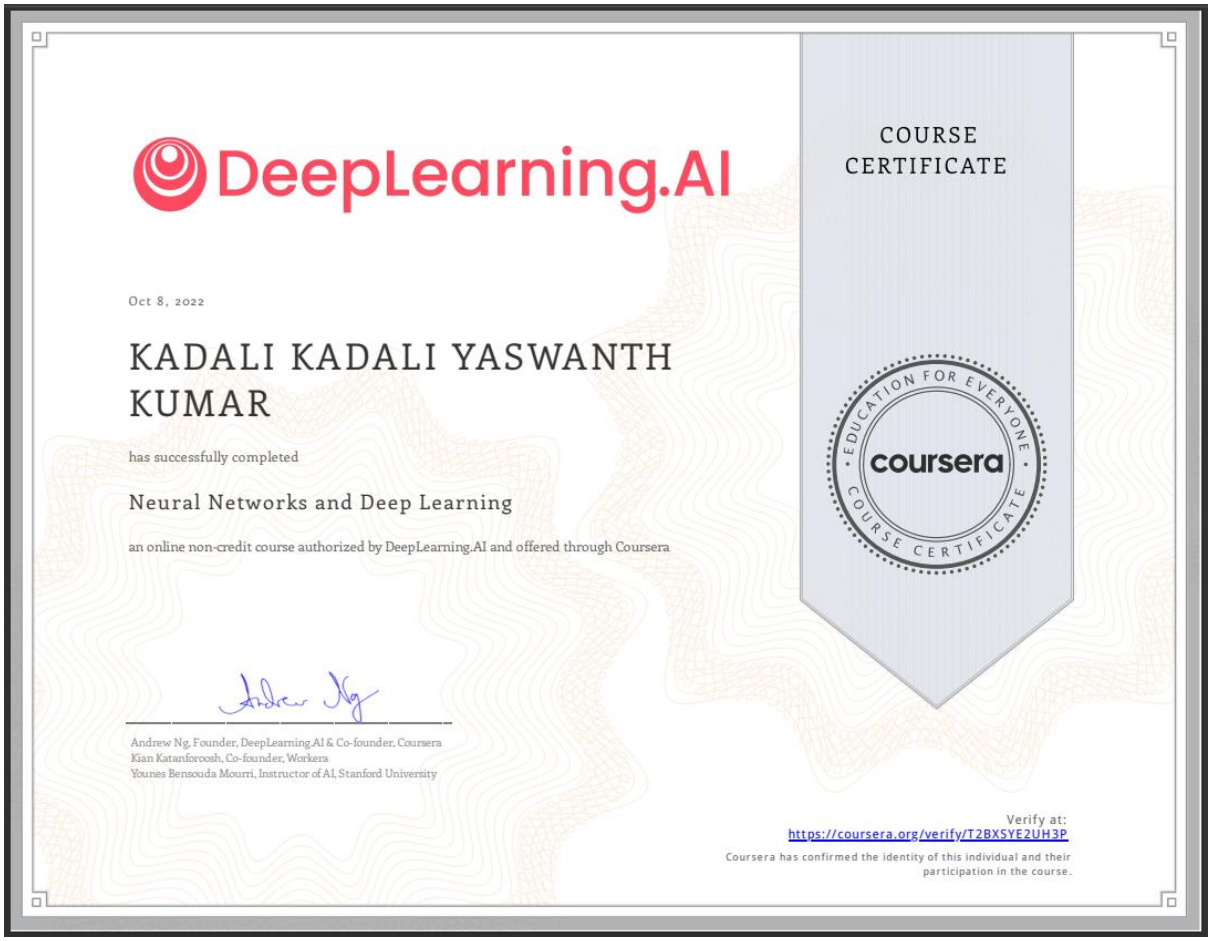
```
    clf_entropy = tarin_using_entropy(X_train, X_test, y_train)

    # Operational Phase
    # Prediction of results using gini
    y_pred_gini = prediction(X_test, clf_gini)
    cal_accuracy(y_test, y_pred_gini)

    # Prediction of results using entropy
    y_pred_entropy = prediction(X_test, clf_entropy)


# Calling main function
if __name__=="__main__":
    main()
```

OUTPUT:

Accuracy: 95.95375722543352

Report :	**precision**	**recall**	**f1-score**	**support**
0.0	**0.98**	**0.97**	**0.98**	**152**
1.0	**0.82**	**0.86**	**0.84**	**21**
accuracy			**0.96**	**173**
macro avg	**0.90**	**0.92**	**0.91**	**173**
weighted avg	**0.96**	**0.96**	**0.96**	**173**

Coursera certificate:



Coursera certificate link:

https://www.coursera.org/account/accomplishments/verify/T2BXSYE2UH3P

Project Referenes:

https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm