Assignment 3

Problem – To make a growing neural network

Approach – I went for a different approach, then told in class. A decision tree is also a kind of neural network, it grows by partitioning data on best feature on each level (entropy and information theory properties are used).

Similarly, I transformed my image to 16*16 and RGB to greyscale, then each of the of the image pixel acts as feature, I selected best feature at each level. How? I selected the feature which classifies the most images and then that feature is not considered again for making next choices.

Code – Link to Kaggle - https://www.kaggle.com/code/abgo24/notebook33ed841c96

```
import numpy as np

import cv2

from sklearn.metrics import accuracy_score

from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt

import random

import os

import cv2

import networkx as nx


G = nx.DiGraph()

faces=[]

image_target_size=[218,178]

image_dir_map ={"faces":['/kaggle/input/celeba-dataset/img_align_celeba/img_align_celeba'],

            "nfaces":['/kaggle/input/natural-images/natural_images/airplane',

                 '/kaggle/input/natural-images/natural_images/car',

                '/kaggle/input/natural-images/natural_images/cat',

                '/kaggle/input/natural-images/natural_images/dog',

                '/kaggle/input/natural-images/natural_images/flower',

                '/kaggle/input/natural-images/natural_images/fruit',
```

```
                    '/kaggle/input/natural-images/natural_images/motorbike']}


for key,val in image_dir_map.items():
    for image_directory in val:
        for root, _, filenames in os.walk(image_directory):
            if key ==str('faces'):
                img_len=1200
                face=1
            else :
                img_len=200
                face=0
            i=0
            while i<img_len:
                i+=1
                image_path = os.path.join(root, filenames[i])
                image = cv2.imread(image_path)
                image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
                current_size = image.shape[:2]
                if current_size[0] != image_target_size[0] or current_size[1] != image_target_size[1]:
                    image = cv2.resize(image, (image_target_size[1], image_target_size[0]),interpolation =
cv2.INTER_AREA)
                if image.shape[:2][0]==image_target_size[0] and image.shape[:2][1]==image_target_size[1]:
                    image_array = np.array(image)
                    faces.append([image_array , face])
                else:
                    print(f'Not correct size {image_path}')


class SingleLayerNN:
    def __init__(self):
```

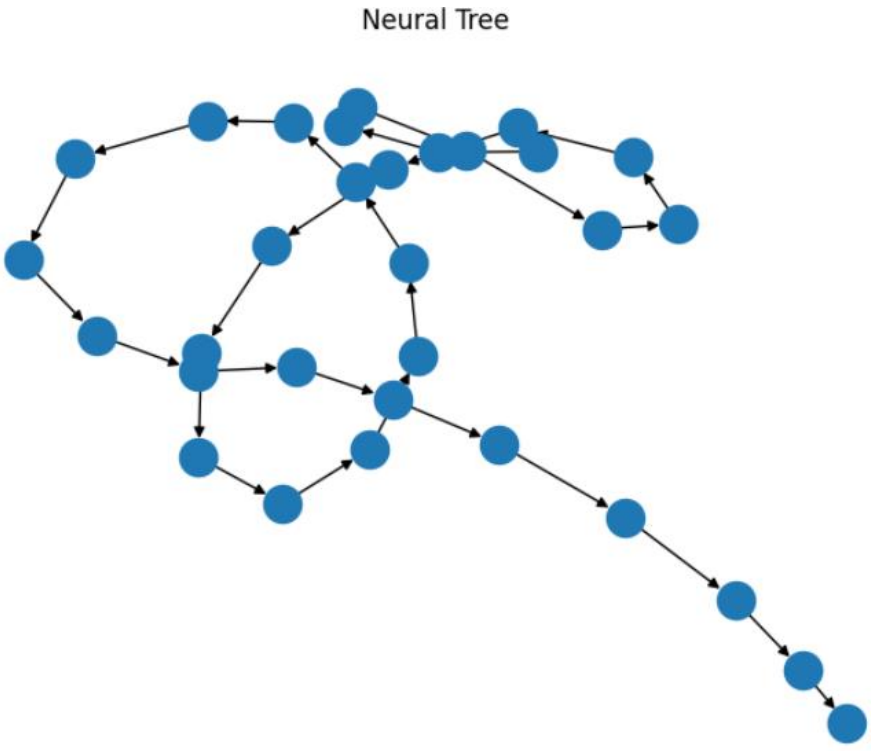
```
        self.W = np.random.randn()
        self.b = 0
    def sigmoid(self, z):
        return 1 / (1 + np.exp(-z))
    def fit(self, X, y, learning_rate=0.01, epochs=25):
        m, n = X.shape
        self.W = np.zeros(n)
        self.b = 0

        for _ in range(epochs):
            z = np.dot(X, self.W) + self.b
            y_pred = self.sigmoid(z)
            dw = (1 / m) * np.dot(X.T, (y_pred - y))
            db = (1 / m) * np.sum(y_pred - y)
            self.W -= learning_rate * dw
            self.b -= learning_rate * db

    def predict(self, X):
        z = np.dot(X, self.W) + self.b
        y_pred = self.sigmoid(z)
        binary_predictions = (y_pred > 0.5).astype(int)
        return binary_predictions


def find_best_feature(X, y,used_features):
    best_feature = None
    best_accuracy = 0.0
    best_predictions = None
    for feature_index in range(X.shape[0]):
```

```
        if(feature_index in used_features):
            continue
        feature_mask = np.zeros(X.shape)
        feature_mask[feature_index] = 1
        masked_X = X * feature_mask
        train_size = int(0.8 * len(y))
        X_train, X_test = masked_X[:train_size], masked_X[train_size:]
        y_train, y_test = y[:train_size], y[train_size:]
        classifier = SingleLayerNN()
        classifier.fit(X_train, y_train)
        predictions = classifier.predict(X_test)
        accuracy = accuracy_score(y_test, predictions)
        if accuracy > best_accuracy:
            best_feature = feature_index
            best_accuracy = accuracy
            best_predictions = predictions

    return best_feature, best_accuracy, best_predictions



class TreeNode:
    def __init__(self, feature_index=None,depth=0):
        self.feature_index = feature_index
        self.left = None
        self.right = None
        self.depth = depth

def build_neural_tree(X, y, max_depth=0,used_features=set()):
    if len(set(y)) == 1 or max_depth==0:
```

```
        return None
    best_feature, best_accuracy, best_predictions = find_best_feature(X, y,used_features)
    used_features.add(best_feature)
    node = TreeNode(feature_index=best_feature, depth=max_depth)
    left_indices = np.where(best_predictions == 0)[0]
    right_indices = np.where(best_predictions == 1)[0]
    node.left = build_neural_tree(X[left_indices], y[left_indices],max_depth-1,used_features)
    node.right = build_neural_tree(X[right_indices], y[right_indices],max_depth-1,used_features)
    return node

X = np.array([((img[0].reshape(-1))/255)*2-1 for img in faces])
y = np.array([img[1] for img in faces])
max_depth=5
root_node = build_neural_tree(X, y,max_depth)

def add_nodes_and_edges(node, parent=None):
    if node is None:
        return
    else :
        node_identifier = generate_unique_identifier()
        G.add_node(node_identifier,feature_index=node.feature_index, depth=node.depth)
        if parent is not None:
            edge_identifier = generate_edge_identifier()
            G.add_edge(node_identifier,edge_identifier)
        if node.left:
            add_nodes_and_edges(node.left, parent=node)
        if node.right:
            add_nodes_and_edges(node.right, parent=node)
```

```
def generate_unique_identifier():
    global node_counter
    node_counter += 1
    return node_counter

def generate_edge_identifier():
    global edge_counter
    edge_counter += 1
    return edge_counter

node_counter = 0
edge_counter = 0
add_nodes_and_edges(root_node)
nx.draw(G)
plt.title("Neural Tree")
plt.axis('off')
plt.show()
```

Graph of neural tree


Neural Tree