Department of Electrical Engineering
Indian Institute of Technology Kharagpur
Machine Learning for Signal Processing Laboratory
(EE69210)
Spring, 2022-23

Experiment 4:
Decision Trees and Random Forest

Grading Rubric

	Tick the best applicable per row			Points
	Below Expectation	Lacking in Some	Meets all Expectation	
Completeness of the report				
Organization of the report (5 pts)				
Quality of figures (5 pts)				
Implement a decision tree using the dataset and compute the accuracy(30 pts)				
Create bags using the bootstrap algorithm (30 pts)				
Implement a random forest using 50 decision trees and compute the accuracy (40 pts)				
			TOTAL (100 pts)	

1 Code of Conduct

Students are expected to behave ethically both in and out of the lab. Unethical behaviour includes, but is not limited to, the following:

- Possession of another person's laboratory solutions from the current or previous years.
- Reference to, or use of another person's laboratory solutions from the current or previous years.
- Submission of work that is not done by your laboratory group.
- Allowing another person to copy your laboratory solutions or work.
- Cheating on quizzes.

The rules of laboratory ethics are designed to facilitate these goals. We emphasize that laboratory TAs are available to help the student understand the basic concepts and answer the questions asked in the laboratory exercises. By performing the laboratories independently, students will likely learn more and improve their performance in the course as a whole. Please note that it is the student's responsibility to ensure that the content of their graded laboratories is not distributed to other students. If there is any question as to whether a given action might be considered unethical, please see the professor or the TA before you engage in such actions.

2 Introduction

A tree is a collection of nodes and edges organized in a hierarchical structure. Nodes are divided into internal (or split) and terminal (or leaf) nodes. We denote internal nodes with circles and terminal ones with squares. All nodes have exactly one incoming edge. Thus, in contrast to graphs, a tree does not contain loops. Also, this document focuses only on binary trees where each internal node has exactly two outgoing edges.

Let $\mathbf{x}_i$ be a column vector of dimension $D \times 1$ denoting the feature vector, consisting of real-valued numbers such that it is represented as $\mathbf{x}_i \in \mathbb{R}^{D \times 1}$.

$$\mathbf{x}_i = \begin{bmatrix} x^{(0)} \\ x^{(1)} \\ \vdots \\ x^{(D-1)} \end{bmatrix} \tag{1}$$

We further represent the set of all such N samples as a matrix $\mathbf{X}$ formed by stacking all the $\mathbf{x}_i$ in a column-wise manner such that $\mathbf{X} \in \mathbb{R}^{D \times N}$.

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_0, & \mathbf{x}_1, & \cdots, & \mathbf{x}_i, & \cdots, & \mathbf{x}_{N-1} \end{bmatrix} \tag{2}$$

Given a previously unseen data point $\mathbf{x}$, a decision tree hierarchically applies a number of predefined tests. Starting at the root, each split node applies its associated split function to $\mathbf{x}$, depending on the binary test result, and the data is sent to the right or left child. This process is repeated until the data point reaches a leaf node. When discussing tree training, it is convenient to think of subsets of training points associated with different tree branches. For instance S_1 denotes the subset of training points reaching a node and S_1^L, S_1^R denote the subsets going to the left and the right children of the node, respectively. In binary trees the following properties apply: $S_j = S_j^L \cup S_j^R$, $S_j^L \cap S_j^R = \emptyset$, $S_j^L = S_{2j+1}$ and $S_j^R = S_{2j+2}$ for each split node j. If we

split the data horizontally this produces two sets of data. The gain of information achieved by splitting the data is computed as:

$$I = H(S) - \sum_{i=1}^{2} \frac{|S_i|}{|S|} H(S_i) \tag{3}$$

with the Shannon entropy defined mathematically as:

$$H(S) = -\sum_{c \in C} p(c) \log_2(p(c)) \tag{4}$$

where $p(c)$ is the class probabilities at each node.

During training, the optimal parameters θ_j of the jth split node need to be computed. This is done here by maximizing an information gain objective function:

$$\theta_j^* = \arg\max_{\theta_j} \{I_j\} \tag{5}$$

with

$$I_j = I(S_j; S_j^L; S_j^R; \theta_j^*) \tag{6}$$

The symbols S_j, S_j^L, S_j^R denote the sets of training points before and after the split. During training, information that is useful for prediction in testing will be learned for all leaf nodes. In the case of classification, each leaf may store the empirical distribution over the classes associated with the subset of training data that has reached that leaf. The probabilistic leaf predictor model for the t-th tree is

$$p_t(c|\mathbf{x}) \tag{7}$$

with $c \in c_1, c_2, \ldots, c_k$ indexing the class. In more conventional decision trees, the estimate is

$$c^* = \arg\max_{c} \{p_t(c|\mathbf{x})\} \tag{8}$$

In a forest with T trees, we have $t \in 1, 2, \ldots, T$. All trees are trained independently. During testing, each test point v is simultaneously pushed through all trees (starting at the root) until it reaches the corresponding leaves. Combining all tree predictions into a single forest prediction may be done by a simple averaging operation. For instance, in classification,

$$p(c|\mathbf{x}) = \frac{1}{T} \sum_{t=1}^{T} p_t(c|\mathbf{x}) \tag{9}$$

3 Datasets

3.1 Two Class Dataset

3.1.1 XOR

This code generates a two-class dataset of 200 samples, with 100 samples per class, and visualizes it using the scatter function from Matplotlib. The data is generated by randomly sampling points from two rectangles centered at (0, 0) and (1, 1), respectively. The label1 and label2 arrays are used to specify the class labels for each sample, with class 1 denoted by red points and class 2 denoted by blue points.

```
import numpy as np
import matplotlib.pyplot as plt

x1 = np.concatenate([np.random.uniform(0, 1, 100), np.random.uniform(-1, 0,
    100)])
```

```
y1 = np.concatenate([np.random.uniform(-1, 0, 100), np.random.uniform(0, 1, 100)])
label1 = np.zeros_like(x1)  # create a list of zeros of the same shape as x1 and y1

x2 = np.concatenate([np.random.uniform(0, 1, 100), np.random.uniform(-1, 0, 100)])
y2 = np.concatenate([np.random.uniform(0, 1, 100), np.random.uniform(-1, 0, 100)])
label2 = np.ones_like(x2)  # create a list of ones of the same shape as x2 and y2

plt.scatter(x1, y1, c='red', label='Class 1')
plt.scatter(x2, y2, c='blue', label='Class 2')
plt.legend()
plt.show()
```

Listing 1: XOR Dataset

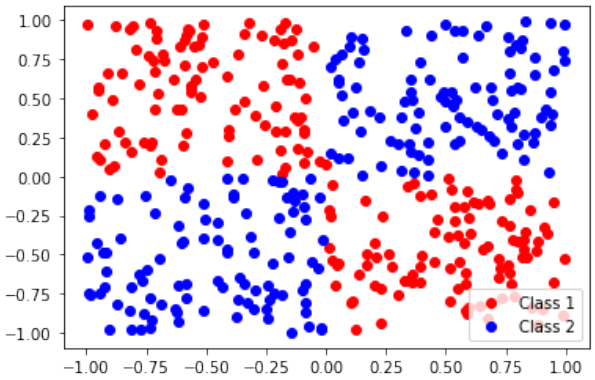

Figure 1: XOR Dataset

3.2 Three Class Datasets

3.2.1 Concentric Circles

This code generates a three-class concentric circles dataset with 300 samples in total (100 per class) and a certain amount of Gaussian noise. The circles are centered at (0, 0), (0, 0), and (0, 0), respectively, with radii of 1, 2, and 3. The X array holds the data points and the y array holds the corresponding class labels, with class 0 denoted by blue points, class 1 denoted by green points, and class 2 denoted by red points

```
import numpy as np
import matplotlib.pyplot as plt

# Set the number of samples and noise level
n = 300
noise = 0.06

# Generate the data
theta = np.linspace(0, 2*np.pi, n)
r1 = np.random.normal(0, noise, n) + 1
r2 = np.random.normal(0, noise, n) + 2
r3 = np.random.normal(0, noise, n) + 3
x1 = r1*np.cos(theta)
y1 = r1*np.sin(theta)
x2 = r2*np.cos(theta)
y2 = r2*np.sin(theta)
```

```
x3 = r3*np.cos(theta)
y3 = r3*np.sin(theta)

# Concatenate the data and labels
X = np.concatenate([np.vstack([x1, y1]), np.vstack([x2, y2]), np.vstack([x3, y3])], axis=1)
y = np.concatenate([np.zeros(n), np.ones(n), np.full(n, 2)])

# Plot the data
plt.scatter(X[0, :], X[1, :], c=y, cmap=plt.cm.brg)
plt.axis('equal')
plt.show()
```

Listing 2: Concentric Circles Dataset

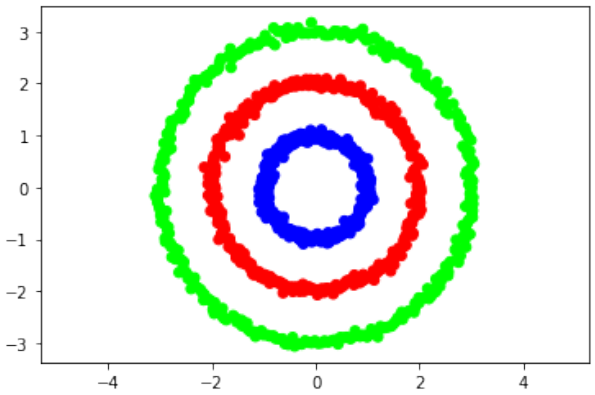

Figure 2: Concentric Circles Dataset

3.2.2 Unbalanced Spiral

This code generates a 2D dataset consisting of 3 classes arranged in a spiral pattern. The dataset has a total of 600 points, with 200 points per class. The variable X is a matrix of size 600x2, where each row represents a data point in 2D space. The variable y is a vector of size 600, where each element represents the class label (0, 1, or 2) of the corresponding data point in X.

The data points are generated by first generating a set of radii and angles, which are used to generate polar coordinates. The radius r is incremented for each class to form the spiral pattern. The angle t is randomly perturbed with Gaussian noise to introduce some variability in the data.

The resulting dataset is visualized using a scatter plot, with each point colored according to its class label. The cmap parameter of the scatter function specifies the color map to be used, which in this case is the brg (blue-red-green) colormap.

```
N = 200 # number of points per class
D = 2 # dimensionality
K = 3 # number of classes
X = np.zeros((N*K,D)) # data matrix (each row = single example)
y = np.zeros(N*K, dtype='uint8') # class labels
for j in range(K):
  ix = range(N*j,N*(j+1))
  r = np.linspace(0.0,1,N) # radius
  r = r+j*0.5
  #print(r)

  t = np.linspace(j*4,(j+1)*4,N) + np.random.randn(N)*0.2 # theta
  #print(t)
  X[ix] = np.c_[r*np.sin(t), r*np.cos(t)]
  y[ix] = j
```

```
spiral4= np.column_stack((X,y))
#np.savetxt("spiral5.csv", spiral4, delimiter=",", header="x,y,label", comments
    ="", fmt='%.5f')
# lets visualize the data:
plt.scatter(X[:, 0], X[:, 1], c=y, s=40, cmap='brg')
plt.show()
```

Listing 3: Unbalanced Spiral Dataset

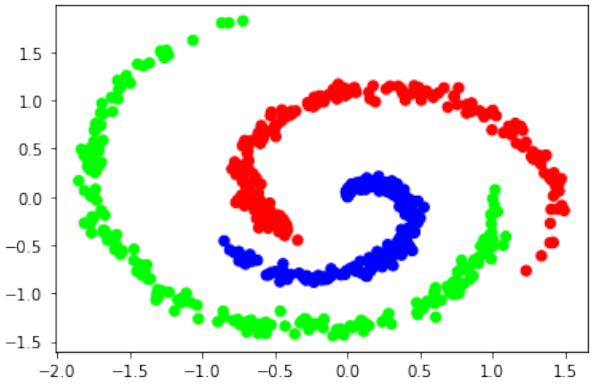

Figure 3: Unbalanced Spiral Dataset

3.2.3 Balanced Spiral

The dataset has the same number of points and classes as before, but the radius increment for each class is now smaller (j*0.05 instead of j*0.5), resulting in a tighter spiral pattern. The angle t is also perturbed with smaller Gaussian noise (np.random.randn(N)*0.02 instead of np.random.randn(N)*0.2).

```
import numpy as np
N = 200 # number of points per class
D = 2 # dimensionality
K = 3 # number of classes
X = np.zeros((N*K,D)) # data matrix (each row = single example)
y = np.zeros(N*K, dtype='uint8') # class labels
for j in range(K):
  ix = range(N*j,N*(j+1))
  r = np.linspace(0.0,1,N) # radius
  r = r+j*0.05
  #print(r)

  t = np.linspace(j*4,(j+1)*4,N) + np.random.randn(N)*0.02 # theta
  #print(t)
  X[ix] = np.c_[r*np.sin(t), r*np.cos(t)]
  y[ix] = j
spiral5= np.column_stack((X,y))
np.savetxt("spiral3u.csv", spiral5, delimiter=",", header="x,y,label", comments
    ="", fmt='%.5f')
# lets visualize the data:
plt.scatter(X[:, 0], X[:, 1], c=y, s=40, cmap='brg')
plt.show()
```

Listing 4: Balanced Spiral Dataset

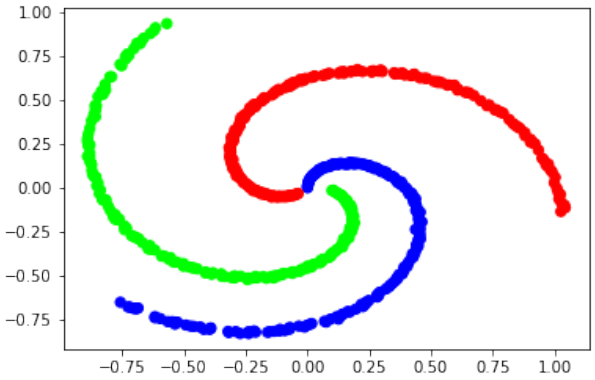

Figure 4: Balanced Spiral Dataset

3.3 Five Class Datasets

3.3.1 Unbalanced Spiral

This code generates a 2D dataset consisting of 5 classes arranged in a spiral pattern. The dataset has a total of 1000 points, with 200 points per class. The variable X is a matrix of size 1000x2, where each row represents a data point in 2D space. The variable y is a vector of size 1000, where each element represents the class label (0, 1, 2, 3, or 4) of the corresponding data point in X.

The data points are generated in a similar way as before, with each class having a different radius increment and angle perturbation. The resulting dataset is visualized using a scatter plot, with each point colored according to its class label. The cmap parameter of the scatter function specifies the color map to be used, which in this case is the brg colormap.

```
N = 200 # number of points per class
D = 2 # dimensionality
K = 5 # number of classes
X = np.zeros((N*K,D)) # data matrix (each row = single example)
y = np.zeros(N*K, dtype='uint8') # class labels
for j in range(K):
  ix = range(N*j,N*(j+1))
  r = np.linspace(0.0,1,N) # radius
  r = r+j*0.5
  #print(r)

  t = np.linspace(j*4,(j+1)*4,N) + np.random.randn(N)*0.2 # theta
  #print(t)
  X[ix] = np.c_[r*np.sin(t), r*np.cos(t)]
  y[ix] = j
spiral4= np.column_stack((X,y))
#np.savetxt("spiral5.csv", spiral4, delimiter=",", header="x,y,label", comments
    ="", fmt='%.5f')
# lets visualize the data:
plt.scatter(X[:, 0], X[:, 1], c=y, s=40, cmap='brg')
plt.show()
```

Listing 5: Unbalanced Spiral Dataset

3.3.2 Balanced Spiral

This code generates a 2D dataset consisting of 5 classes arranged in a spiral pattern. The dataset has a total of 1000 points, with 200 points per class. The variable X is a matrix of size 1000x2, where each row represents a data point in 2D space. The variable y is a vector of size 1000,

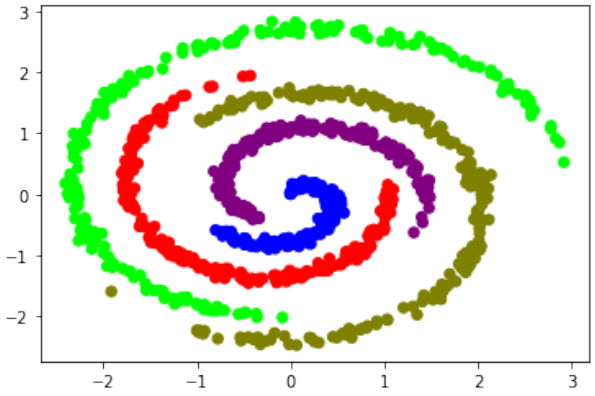

Figure 5: Unbalanced Spiral Dataset

where each element represents the class label (0, 1, 2, 3, or 4) of the corresponding data point in X.

The main difference with the previous code is that the radius increment for each class has been reduced from 0.5 to 0.05, making the spirals tighter. Additionally, the angle perturbation has been reduced from 0.2 to 0.02, making the spirals smoother.

```
import numpy as np
import numpy as np
N = 200 # number of points per class
D = 2 # dimensionality
K = 5 # number of classes
X = np.zeros((N*K,D)) # data matrix (each row = single example)
y = np.zeros(N*K, dtype='uint8') # class labels
for j in range(K):
  ix = range(N*j,N*(j+1))
  r = np.linspace(0.0,1,N) # radius
  r = r+j*0.05
  #print(r)

  t = np.linspace(j*4,(j+1)*4,N) + np.random.randn(N)*0.02 # theta
  #print(t)
  X[ix] = np.c_[r*np.sin(t), r*np.cos(t)]
  y[ix] = j
spiral5= np.column_stack((X,y))
#np.savetxt("spiral5u.csv", spiral5, delimiter=",", header="x,y,label",
    comments="", fmt='%.5f')
# lets visualize the data:
plt.scatter(X[:, 0], X[:, 1], c=y, s=40, cmap='brg')
plt.show()
```

Listing 6: Balanced Spiral Dataset

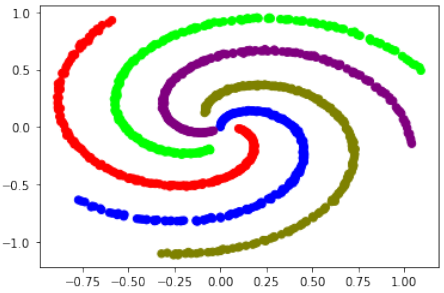

Figure 6: Balanced Spiral Dataset

4 Objectives

4.1 Assignments to solve and report

Write the codes for the above tasks in the .ipynb file, and submit the executed file. Submit a separate pdf describing your observations and reasoning while executing these experiments.

1. Generate and visualize all the datasets in 2D space.
2. Write a generic code to create one decision tree which can be used to classify any number of classes. Using the generated decision tree, classify XOR, Concentric circles, and spiral datasets.
3. Show the information gain at each node in the tree and show the feature number and associated threshold value at each node in the tree. Also, show the probability distribution over classes at the leaf nodes.
4. Take 70% of the dataset as training data and 30% as test test. Compute the accuracy using the generated decision tree for all datasets.
5. Create bags for each dataset using the bootstrap algorithm. Show that each bag must have the same class distribution as the class distribution of the original dataset.
6. Create a random forest with 10 trees such that each tree is trained with a different bag. Take 70% of each bag as training data and 30% as the test set. Compute the accuracy of the random forest using the generated decision trees using Equation (9) for all datasets.
7. Visualize the decision boundaries for one decision tree in 2D space. Also, visualize the decision boundaries for the random forest with 10 trees on 2D space.