ASU ID 1233271600

Name : Suhas Raghavendra

In [1]:
```
import sys; sys.path.append('../..') ; sys.path.append('..') ; from my_utils

import torch
import torch.nn as nn
import torch.utils.data as data
import torch.optim as optim
# dummy trainloader
trainloader = data.DataLoader(data.TensorDataset(torch.Tensor(1), torch.Tens
device = torch.device('cpu')

import matplotlib.pyplot as plt
```

In this homework, there are three different datasets consisting of 2-dimensional input features and binary class labels, and you will be asked to implement machine learning classifiers.

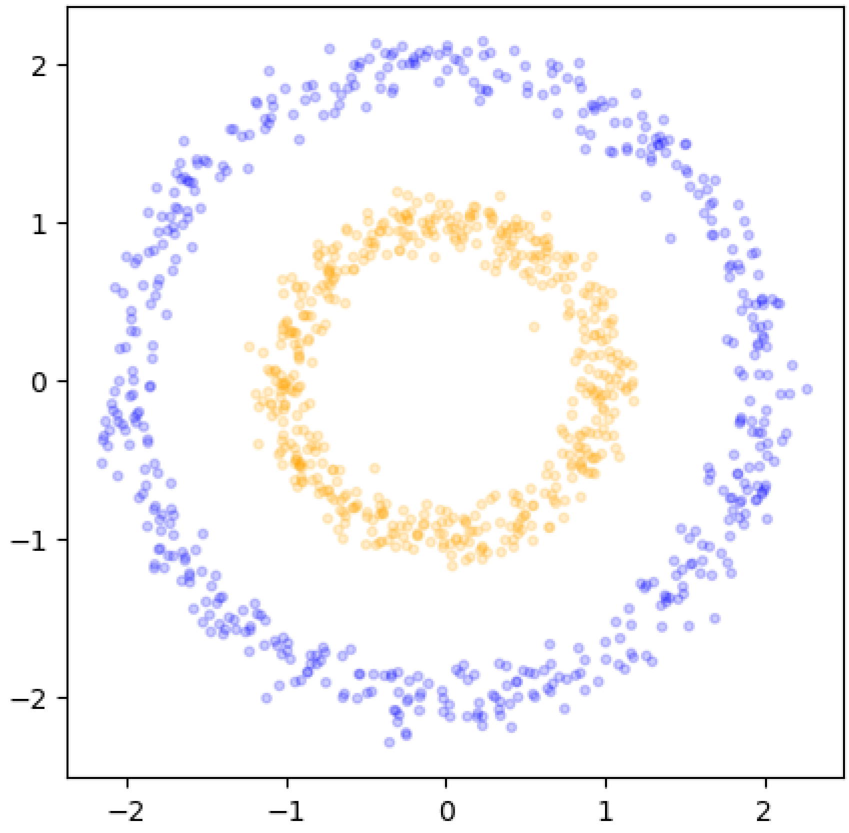
Let's begin by importing some libaries.

Next, we set a random seed for reproducibility.

In [3]:
```
import numpy as np
import random

seed = 0
np.random.seed(seed)
torch.random.manual_seed(seed)
random.seed(seed)
```

Concentric annuli

In [5]:
```python
X, y = sample_annuli()
fig, ax = plt.subplots(1,1, figsize=(5,5))
plot_scatter(ax, X, y)
```



[2pt] Let's start by implmenting a logistic regression model (like in HW2). Fill the template below to complete the logisitc regression model. Use the binary cross entropy loss, torch.nn.BCELoss.

(i) Complete the model, (ii) finish the training loop, (iii) present the results with a figure (see the example below) and the classification accuracy

In [7]:
```python
class Model(nn.Module):
    def __init__(self,device="cpu"):
        super(Model, self).__init__()
        self.linear = nn.Linear(2, 1)

    def forward(self, x):
        return torch.sigmoid(self.linear(x))
```

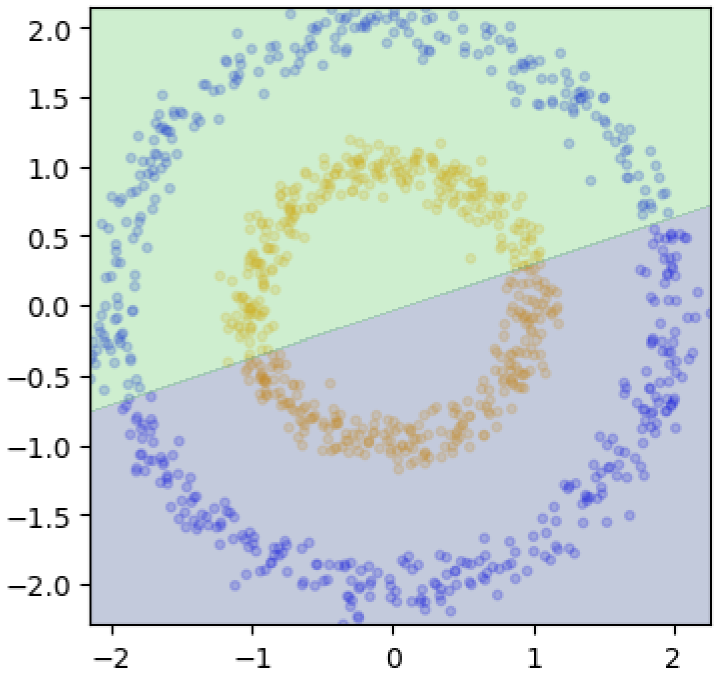
In [8]:
```python
model = Model().to(device)
```

In [9]:
```python
optimizer = optim.AdamW(model.parameters(), lr=1e-2, weight_decay=1e-6)
```

In [10]:
```python
# complete the following training loop.
criterion = nn.BCELoss()


for itr in range(1, 1001):
    optimizer.zero_grad()
    yh = model(X)
    loss = criterion(yh, y.unsqueeze(1).float())
    loss.backward()
    optimizer.step()
```

In [11]:
```python
# visualize the result and report the accuracy
with torch.no_grad():
    fig, ax = plt.subplots(figsize=(4, 4))
    plot_scatter(ax, X, y)
    x_min, x_max = X[:, 0].min(), X[:, 0].max()
    y_min, y_max = X[:, 1].min(), X[:, 1].max()
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100),
                         np.linspace(y_min, y_max, 100))
    grid_points = np.c_[xx.ravel(), yy.ravel()]
    Z = model(torch.Tensor(grid_points).to(device)).cpu().numpy()
    Z = Z.reshape(xx.shape)
    ax.contourf(xx, yy, Z, alpha=0.3, levels=np.linspace(0, 1, 3))
    predictions = (model(X) > 0.5).float()
    accuracy = (predictions == y.unsqueeze(1)).float().mean()

    print(f"Accuracy: {accuracy.item():.2f}")

plt.show()
```

```
Accuracy: 0.53
```



It is obvious that the logistic regression would not be able to distinguish two classes (not linearly separate data). You will have to build another model.

[4pt] [Feature engineering] In the class template below, implement your own model that will achieve 100% accuracy in classifying the data poitns in training set. There is one restriction; you are allowed to use "one" linear layer for your implementation as in the logistic regression model above. But you are allowed to use as many nonlinear functions as needed to engineer hand-crafted features.

Plot the classification result in the format of an image shown below (i.e., scatter plot with different colors for each label) and compute and print out the accuracy.

2pts for model + training

1pts for plotting

In [13]:
```
class Model(nn.Module):
    def __init__(self, device="cpu"):
        super(Model, self).__init__()
        self.linear = nn.Linear(1, 1)

    def forward(self, x):
        r = torch.sqrt(torch.sum(x ** 2, dim=1, keepdim=True))
        output = torch.sigmoid(self.linear(r))
        return output
```

In [14]:
```
model = Model().to(device)
```

In [15]:
```
optimizer = optim.AdamW(model.parameters(), lr=1e-2, weight_decay=1e-6)
```

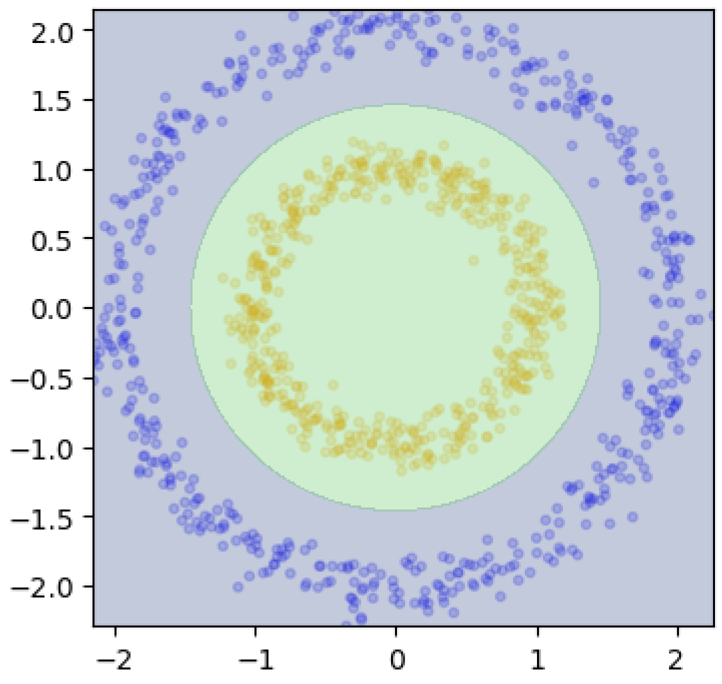
In [16]:
```
criterion = nn.BCELoss()


for itr in range(1, 1001):
    optimizer.zero_grad()
    yh = model(X)
    loss = criterion(yh, y.unsqueeze(1).float())
    loss.backward()
    optimizer.step()
```

In [17]:
```
print(f"Iteration {itr}, Loss: {loss.item():.4f}")
```

```
Iteration 1000, Loss: 0.1219
```

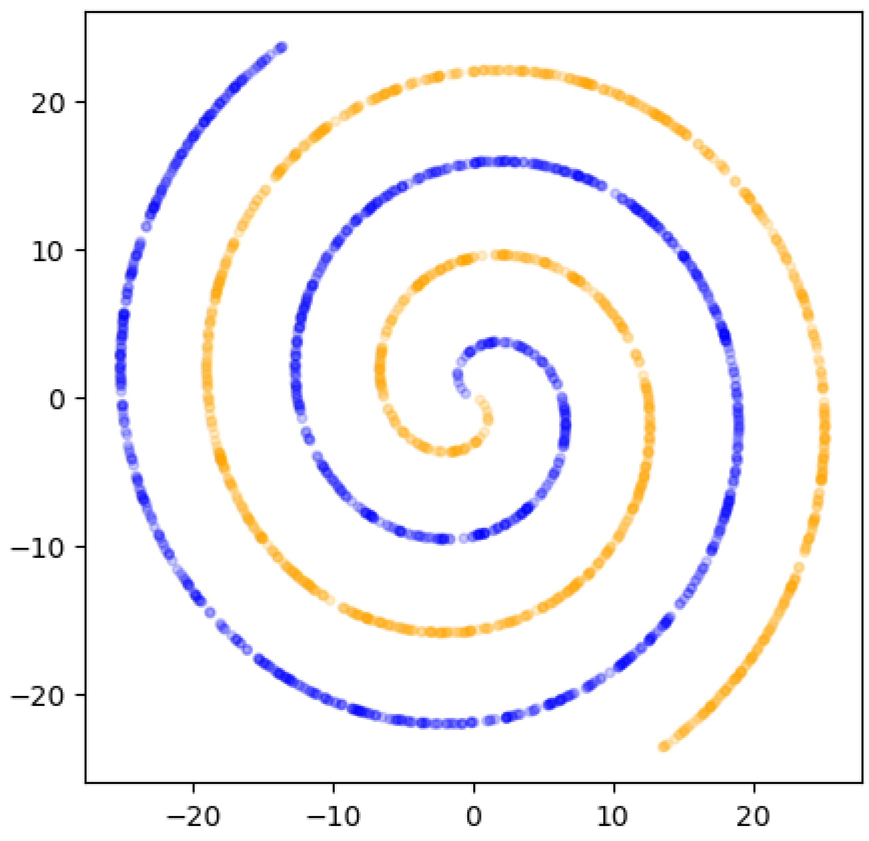
In [18]:
```python
with torch.no_grad():
    fig, ax = plt.subplots(figsize=(4, 4))
    plot_scatter(ax, X, y)

    xx, yy = np.meshgrid(np.linspace(X[:, 0].min(), X[:, 0].max(), 100),
                         np.linspace(X[:, 1].min(), X[:, 1].max(), 100))
    grid_points = np.c_[xx.ravel(), yy.ravel()]
    Z = model(torch.Tensor(grid_points).to(device)).cpu().numpy()
    Z = Z.reshape(xx.shape)

    ax.contourf(xx, yy, Z, alpha=0.3, levels=np.linspace(0, 1, 3))

    predictions = (model(X) > 0.5).float()
    accuracy = (predictions == y.unsqueeze(1)).float().mean()
    print(f"Accuracy: {accuracy.item():.2f}")

plt.show()
```

```
Accuracy: 1.00
```



Spiral dataset

In [20]:
```
X, y = sample_spiral()
fig, ax = plt.subplots(1,1, figsize=(5,5))
plot_scatter(ax, X, y)
```



It's obvious that neither the logistic regression nor the model you developed for the second dataset would not work for this dataset.

[3pt] implemente a neural network of your choice (any type of NNs is allowed) and achieve 100% classification accuracy

Plot the classification result in the format of an image shown below (i.e., scatter plot with different colors for each label) and compute and print out the accuracy.

1pts for model + training

1pts for plotting

1pts for computing accuracy

In [51]:
```python
class Model(nn.Module):
    def __init__(self, device="cpu"):
        super(Model, self).__init__()
        self.layers = nn.Sequential(
            nn.Linear(2, 64),
            nn.ReLU(),
            nn.Linear(64, 64),
            nn.ReLU(),
            nn.Linear(64, 1)
        )

    def forward(self, x):
        x = self.layers(x)
        return torch.sigmoid(x)
```

In [53]:
```python
model = Model().to(device)
```

In [55]:
```python
optimizer = optim.AdamW(model.parameters(), lr=1e-2, weight_decay=1e-6)
```

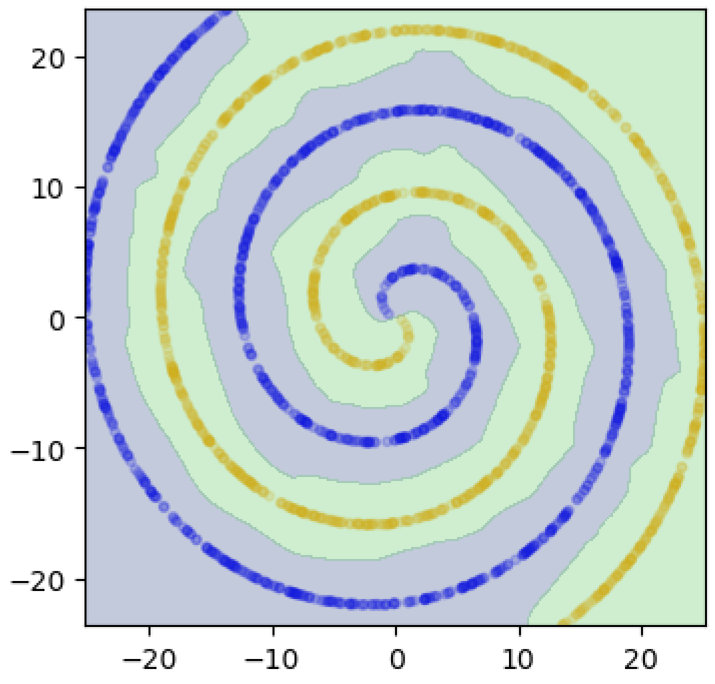
In [57]:
```python
criterion = nn.BCELoss()

for iteration in range(1, 1001):
    optimizer.zero_grad()
    yh = model(X)
    loss = criterion(yh, y.unsqueeze(1).float())
    loss.backward()
    optimizer.step()
```

In [59]:
```python
print(f"Iteration {iteration}, Loss: {loss.item():.4f}")
```

```
Iteration 1000, Loss: 0.0009
```

In [63]:
```
with torch.no_grad():
    fig, ax = plt.subplots(figsize=(4, 4))
    plot_scatter(ax, X, y)

    xx, yy = np.meshgrid(
        np.linspace(X[:, 0].min(), X[:, 0].max(), 100),
        np.linspace(X[:, 1].min(), X[:, 1].max(), 100)
    )

    Z = model(torch.Tensor(np.c_[xx.ravel(), yy.ravel()]).to(device)).cpu()
    Z = Z.reshape(xx.shape)

    ax.contourf(xx, yy, Z, alpha=0.3, levels=np.linspace(0, 1, 3))

    preds = (model(X) > 0.5).float()
    accuracy = (preds == y.unsqueeze(1)).float().mean()
    print(f"Accuracy: {accuracy.item():.2f}")

plt.show()
```

```
Accuracy: 1.00
```


20
10
0
−10
−20
−20
−10
0
10
20