exercicio5

January 23, 2021

1 Exercíco 5 - Treinamento ELM

Aluno: Leonam Rezende Soares de Miranda

O objetivo dos exercícios é ter um primeiro contato com classificadores não lineares, especificamente ELMs. Utilizando o pacote em R mlbench, foram geradas as seguintes bases de dados, que serão importadas para a realização dos experimentos.

- *mlbench.2dnormals(200)*
- *mlbench.xor(100)*
- *mlbench.circle(100)*
- *mlbench.spirals(100,sd = 0.05)*

Para cada uma das bases, foram construídas diferentes ELMs (pelo menos 3 para cada base de dados) variando-se o número de neurônios da camada intermediária.

Neste primeiro momento, a avaliação da qualidade dos resultados será exclusivamente visual.

1.0.1 1 Construindo um classificador ELM do zero

Foi construída a classe "ELM" que implementa o classificador ELM, conforme o código a seguir:

[133]:
```
from sklearn.base import BaseEstimator
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

class ELM(BaseEstimator):
    """ Classificador ELM """

    def __init__(self):
        self.Z = None
        self.W = None
        self.p = None

    def _check_X_y(self, X, y):
        """ Validate assumptions about format of input data"""
        assert set(y) == {-1, 1}, 'Response variable must be ±1'
        return X, y
```

```
   def fit(self, X: np.ndarray, y: np.ndarray, p):
       """Controi um classificador otimizado a partir do conjunto de␣
↪treinamento (X, y).

        Parâmetros
        ----------
        X: {tipo matriz, matriz esparsa} de forma (n_amostras,␣
↪n_características)
            Os exemplos de entrada de treinamento.

        y: Vetor de formato (n_samples,)
            Os valores alvo (rótulos de classe) do conjunto de treinamento.

        p: número de neurônios da camada intermediária

        Retorna
        -------
        self: objeto
            Estimador ajustado.
        """

       X, y = self._check_X_y(X, y)
       self.p = p
       # Vetor pesos da camada escondida gerado de forma aleatória
       self.Z = np.random.uniform(low=-0.5, high=0.5, size=(X.shape[1] +1,␣
↪self.p))

       H = np.tanh(np.dot(X, self.Z[1:]) + self.Z[0])

       # Vetor pesos entre a camada de saída e a camada intermediária
       self.W = np.dot(np.linalg.pinv(H), y)

       return self

   def predict(self, X):
       """ Make predictions using already fitted model

       Parâmetros
        ----------
       X: {tipo matriz, matriz esparsa} de forma (n_amostras,␣
↪n_características)
            Os exemplos de entrada.
       """

       H = np.tanh(np.dot(X, self.Z[1:]) + self.Z[0])
       return np.sign(np.dot(H,self.W))
```

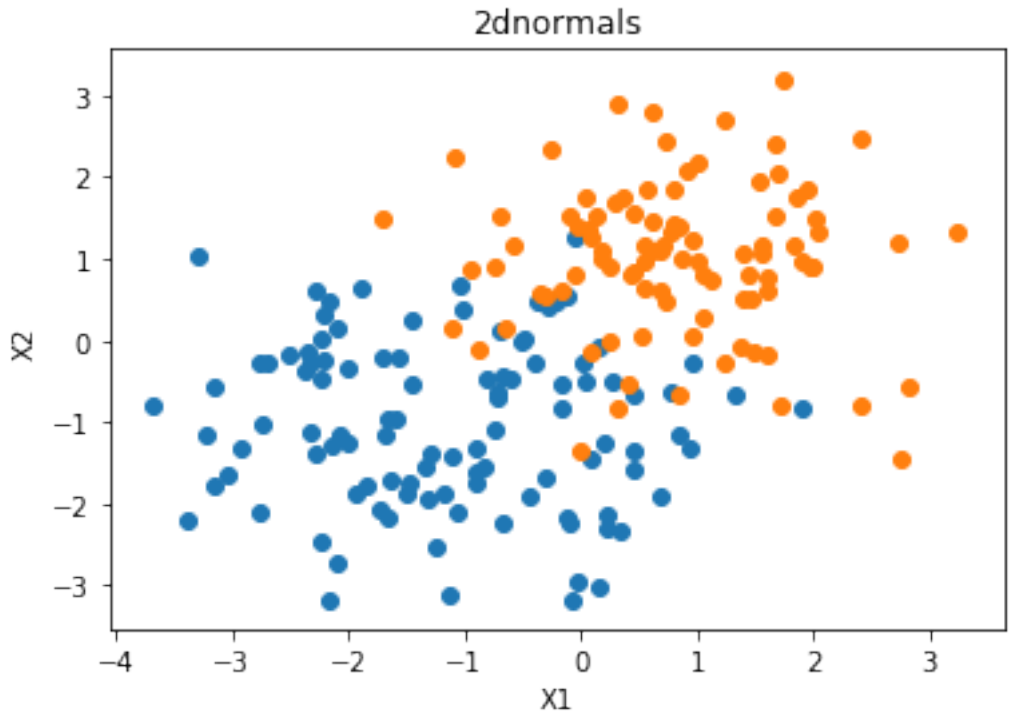
```
    def score(self, X, Y):
        """ Retorna a acurácia do classificador

        Parâmetros
         ----------
        X: {tipo matriz, matriz esparsa} de forma (n_amostras,␣
↪n_características)
             Os exemplos de entrada.

        Y: Vetor de formato (n_samples,)
             Os valores alvo (rótulos de classe) dos exemplos de entrada.
        """
        Y_pred = self.predict(X)
        return np.sum(Y_pred == Y)/Y.shape[0]

    def plot_separacao(self, X, X_1, X_2, ax):
        x_1 = np.linspace(np.floor(X[:,0].min()), np.ceil(X[:,0].max()), 200)
        x_2 = np.linspace(np.floor(X[:,1].min()), np.ceil(X[:,1].max()), 200)

        Contour = np.zeros((x_1.shape[0], x_2.shape[0]))

        for i in range(x_2.shape[0]):
            for j in range(x_1.shape[0]):
                Contour[j,i] = self.predict(np.array([x_2[i], x_1[j]]))


        ax.plot(X_1[:,0], X_1[:,1], 'o')
        ax.plot(X_2[:,0], X_2[:,1], 'o')
        ax.set(xlabel='X2', ylabel='X1')
        ax.pcolor(x_2, x_1, Contour, cmap="RdGy")
        # ax.set_xlim(x_1.min(), x_1.max())
        # ax.set_ylim(x_2.min(), x_2.max())
        ax.set_title('ELM: p = ' + str(self.p))
```

1.1 2. 2dnormals

[135]:
```
# Importação do conjunto de dados
data = pd.read_csv('./2dnormals.csv', delimiter=",", header=None)

#remoção da primeira linha
data = data[1:-1]

# Estração dos atributos
X = data[[1,2]].values.astype(np.float)

# Extração dos rótulos
Y = np.squeeze(data[[3]].values.astype(np.int))
```

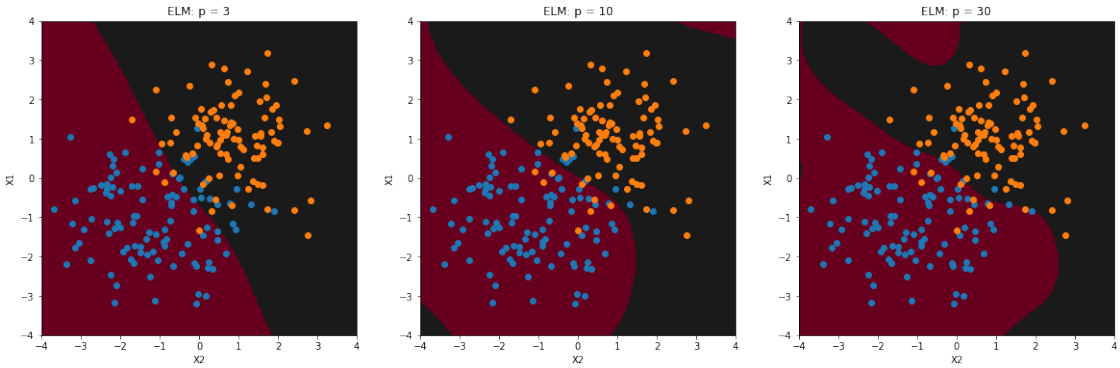
```
# Mapeia os rótulos binários de forma que negativo = -1 e positivo = +1
Y[Y==2] = -1

# Separa os pontos da classe positiva dos pontos da classe negativa
X_1 = np.vstack((X[Y==-1,0], X[Y==-1,1])).T
X_2 = np.vstack((X[Y==1,0], X[Y==1,1])).T

# plota os pontos
plt.scatter(X_1[:,0], X_1[:,1])
plt.scatter(X_2[:,0], X_2[:,1])
plt.title('2dnormals')
plt.xlabel('X1')
plt.ylabel('X2')
```

[135]: `Text(0, 0.5, 'X2')`



[136]:
```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(20,6))

# Treina o 1º modelo : p = 3
model1 = ELM().fit(X, Y, 3)
acuracia1 = model1.score(X, Y)
model1.plot_separacao(X, X_1, X_2, ax1)
```

```
# Treina o 2º modelo : p = 10
model2 = ELM().fit(X, Y, 10)
acuracia2 = model2.score(X, Y)
model2.plot_separacao(X, X_1, X_2, ax2)

# Treina o 3º modelo : p = 30
model3 = ELM().fit(X, Y, 30)
acuracia3 = model3.score(X, Y)
model3.plot_separacao(X, X_1, X_2, ax3)
```



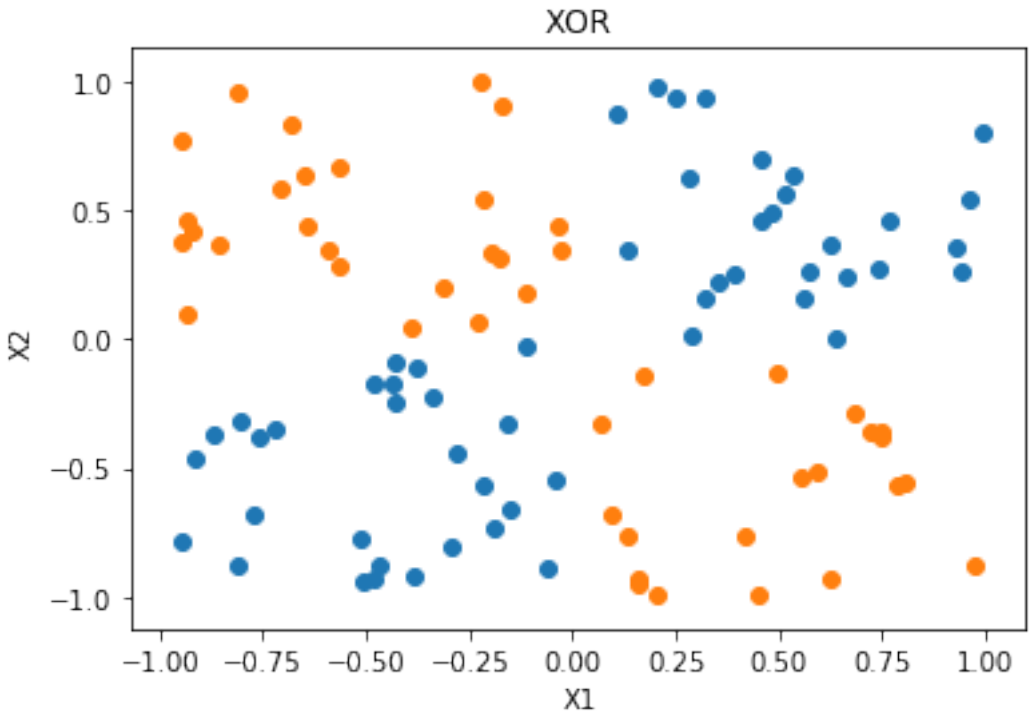
É nítido que para $p = 3$ foi o obtido o separador com melhor generalização, sendo mais robusto a variâncias do dado. Para $p = 30$ e $p = 10$ verifica-se que ocorreu sobreajuste.

1.2 3. XOR

[137]:
```
# Importação do conjunto de dados
data = pd.read_csv('./xor.csv', delimiter=",", header=None)

#remoção da primeira linha
data = data[1:-1]

# Estração dos atributos
X = data[[1,2]].values.astype(np.float)

# Extração dos rótulos
Y = np.squeeze(data[[3]].values.astype(np.int))

# Mapeia os rótulos binários de forma que negativo = 0 e positivo = +1
Y[Y==2] = -1

# Separa os pontos da classe positiva dos pontos da classe negativa
X_1 = np.vstack((X[Y==-1,0], X[Y==-1,1])).T
X_2 = np.vstack((X[Y==1,0], X[Y==1,1])).T
```

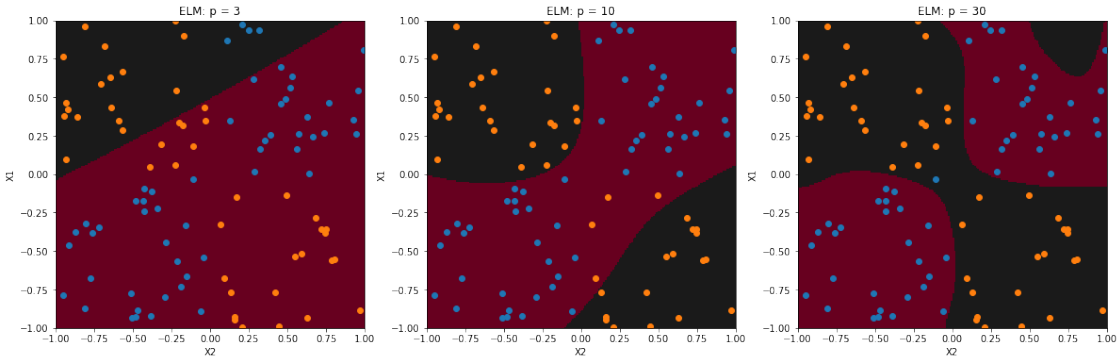
```
# plota os pontos
plt.scatter(X_1[:,0], X_1[:,1])
plt.scatter(X_2[:,0], X_2[:,1])
plt.title('XOR')
plt.xlabel('X1')
plt.ylabel('X2')
```

[137]: `Text(0, 0.5, 'X2')`



[138]:
```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(20,6))

# Treina o 1º modelo : p = 3
model1 = ELM().fit(X, Y, 3)
acuracia1 = model1.score(X, Y)
model1.plot_separacao(X, X_1, X_2, ax1)

# Treina o 2º modelo : p = 10
model2 = ELM().fit(X, Y, 10)
acuracia2 = model2.score(X, Y)
model2.plot_separacao(X, X_1, X_2, ax2)

# Treina o 3º modelo : p = 30
```

```
model3 = ELM().fit(X, Y, 30)
acuracia3 = model3.score(X, Y)
model3.plot_separacao(X, X_1, X_2, ax3)
```

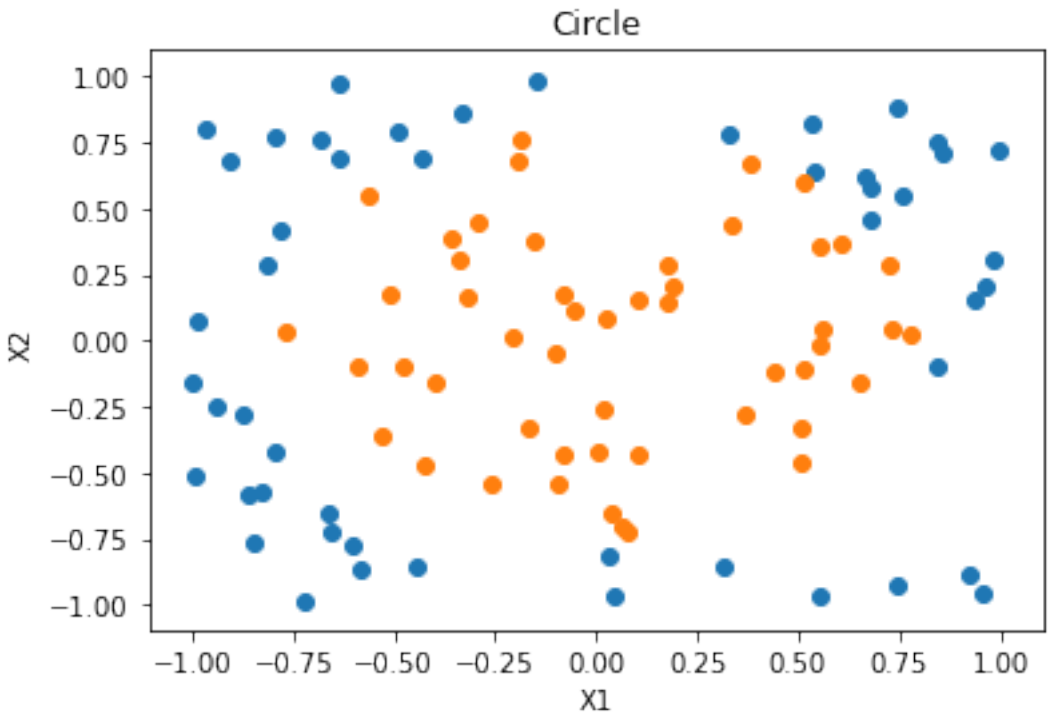


Vizualmente o separador com $p = 30$ obteve melhor generalização, sendo mais robusto a variâncias do dado. Para $p = 3$ e $p = 10$ ocorreu *underfitting*.

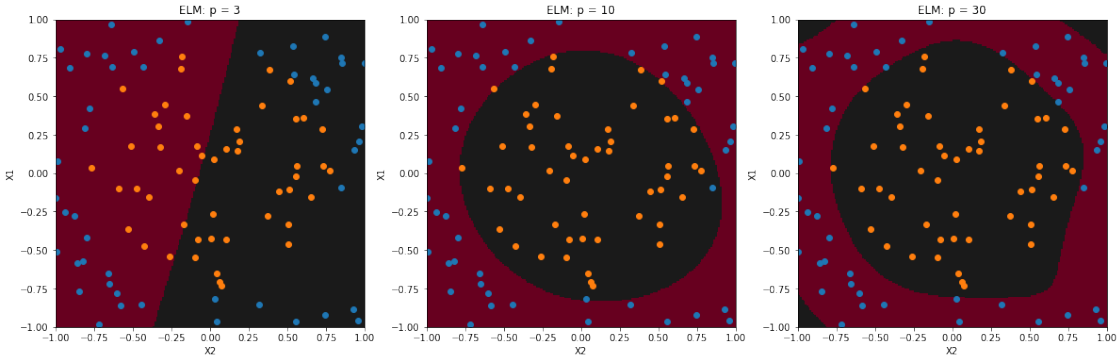
1.3 4. Circle

[140]:
```
# Importação do conjunto de dados
data = pd.read_csv('./circle.csv', delimiter=",", header=None)

#remoção da primeira linha
data = data[1:-1]

# Estração dos atributos
X = data[[1,2]].values.astype(np.float)

# Extração dos rótulos
Y = np.squeeze(data[[3]].values.astype(np.int))

# Mapeia os rótulos binários de forma que negativo = 0 e positivo = +1
Y[Y==2] = -1

# Separa os pontos da classe positiva dos pontos da classe negativa
X_1 = np.vstack((X[Y==-1,0], X[Y==-1,1])).T
X_2 = np.vstack((X[Y==1,0], X[Y==1,1])).T

# plota os pontos
plt.scatter(X_1[:,0], X_1[:,1])
plt.scatter(X_2[:,0], X_2[:,1])
plt.title('Circle')
plt.xlabel('X1')
plt.ylabel('X2')
```

[140]: Text(0, 0.5, 'X2')



[141]:
```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(20,6))

# Treina o 1º modelo : p = 3
model1 = ELM().fit(X, Y, 3)
acuracia1 = model1.score(X, Y)
model1.plot_separacao(X, X_1, X_2, ax1)

# Treina o 2º modelo : p = 10
model2 = ELM().fit(X, Y, 10)
acuracia2 = model2.score(X, Y)
model2.plot_separacao(X, X_1, X_2, ax2)

# Treina o 3º modelo : p = 30
model3 = ELM().fit(X, Y, 30)
acuracia3 = model3.score(X, Y)
model3.plot_separacao(X, X_1, X_2, ax3)
```

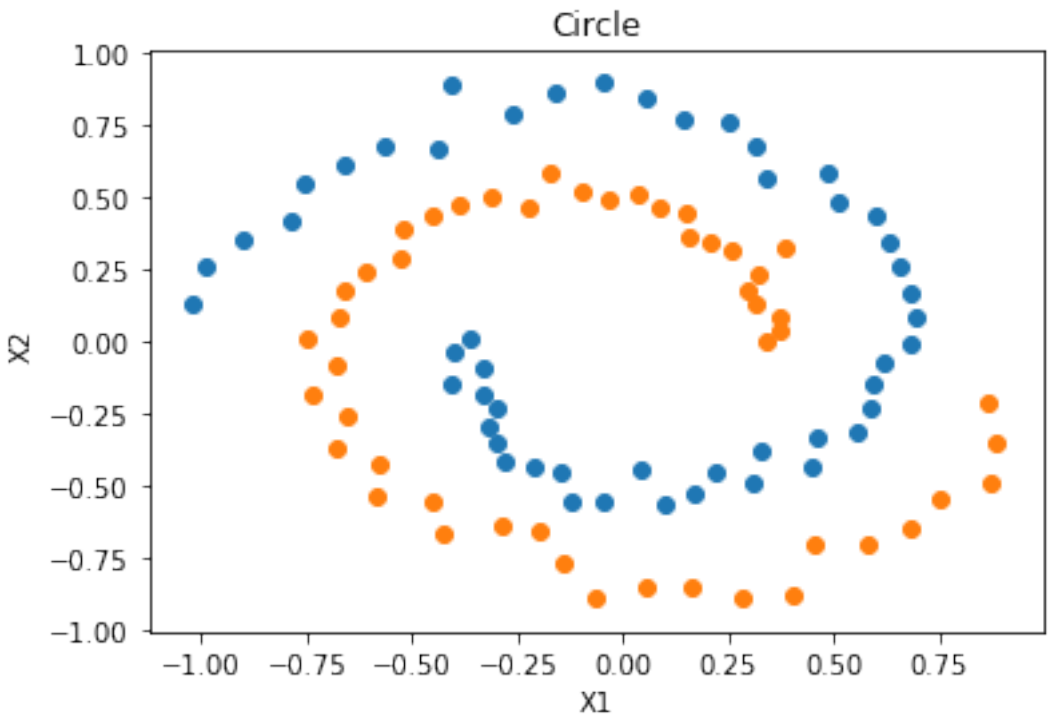


Vizualmente o separador com $p = 10$ obteve melhor generalização, pois a separação é a que mais se assemelha a um circulo, sendo mais robusto a variâncias do dado. É nítido que para p=3 ocorreu underfitting. Para $p = 30$ o erro encontrado foi o menos, mas ocorreu sobreajuste.

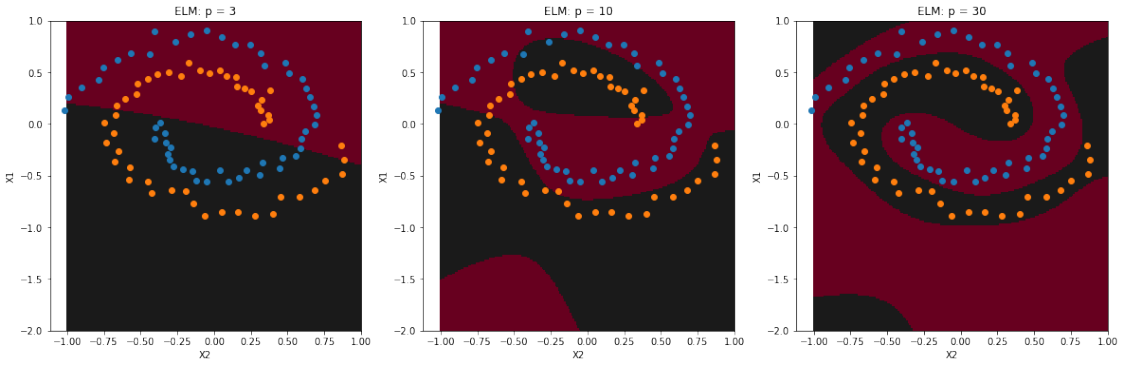
1.4 5. Spirals

[19]:
```
# Importação do conjunto de dados
data = pd.read_csv('./spirals.csv', delimiter=",", header=None)

#remoção da primeira linha
data = data[1:-1]

# Estração dos atributos
X = data[[1,2]].values.astype(np.float)

# Extração dos rótulos
Y = np.squeeze(data[[3]].values.astype(np.int))

# Mapeia os rótulos binários de forma que negativo = 0 e positivo = +1
Y[Y==2] = -1

# Separa os pontos da classe positiva dos pontos da classe negativa
X_1 = np.vstack((X[Y==-1,0], X[Y==-1,1])).T
X_2 = np.vstack((X[Y==1,0], X[Y==1,1])).T

# plota os pontos
plt.scatter(X_1[:,0], X_1[:,1])
plt.scatter(X_2[:,0], X_2[:,1])
plt.title('Circle')
plt.xlabel('X1')
plt.ylabel('X2')
```

[19]:
```
Text(0, 0.5, 'X2')
```



[134]:
```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(20,6))

# Treina o 1º modelo : p = 3
model1 = ELM().fit(X, Y, 3)
acuracia1 = model1.score(X, Y)
model1.plot_separacao(X, X_1, X_2, ax1)

# Treina o 2º modelo : p = 10
model2 = ELM().fit(X, Y, 10)
acuracia2 = model2.score(X, Y)
model2.plot_separacao(X, X_1, X_2, ax2)

# Treina o 3º modelo : p = 30
model3 = ELM().fit(X, Y, 30)
acuracia3 = model3.score(X, Y)
model3.plot_separacao(X, X_1, X_2, ax3)
```



Vizualmente o separador com $p = 30$ obteve melhor generalização, sendo mais robusto a variâncias do dado. É nítido que para $p=3$ ocorreu underfitting. Talvez escolhendo Um valor de p entre 10 e 30 fosse encontrado um separador melhor.