exercicio7

February 12, 2021

1 Redes Neurais Artificiais

2 Exercíco 7 - Treinamento RBF

Aluno: Leonam Rezende Soares de Miranda

O objetivo deste exercício é implementar e tetar uma rede neural RBF, com seleção automática de centros e raios, usando técnica de k-médias. 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 construídos diferentes classificadores RBFs (pelo menos 3 para cada base de dados) variando-se o número de centros k para a função de k-médias.

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

2.1 1 Construindo um classificador RBF do zero

Foi construída a classe "RBF" que implementa o classificador RBF, usando a técnica de k-médias conforme o código a seguir:

[66]:
```
from sklearn.base import BaseEstimator
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split

class RBF(BaseEstimator):
    """ Classificador RBF """

    def __init__(self, classification =True):
        self.W = None
        self.p = None
        self.m = None
        self.covlist = []
```

```
        self.n = None
        self.classification = classification

    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 pdfnvar(self, x, m, K):
        """Função radial Gaussiana

            Parâmetros
            ----------
            x: amostra de forma (1, n_características)

            m: vetor de médias de forma (n_características,)

            K: matriz de covariâncias de forma (n_características,␣
↪n_características)

            Retorna
            -------
            p: pdf para cada entrada em um dado cluster determinado po m e K

        """
        p = 1/np.sqrt((2*np.pi) ** self.n * np.linalg.det(K)) * np.exp((-0.5*(x␣
↪- m).T).dot(np.linalg.pinv(K)).dot(x-m))
        return p

    def calcH(self, X: np.ndarray):
        """Função que cálcula a matriz H a a partir dos valores de centros
        e as matrizes de covariâncias de cada centro do modelo

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

            Retorna
            -------
            H: matriz H (saída da pdf de cada neurônio para cada amostra␣
↪acrescida de um bias)
        """

        # número de amostras
        N = X.shape[0]
```

```
        H = np.ones((N, self.p + 1))

        for j in range(N):
            for i in range(self.p):
                mi = self.m[i,:]
                covi = self.covlist[i] + 0.001 * np.identity(X.shape[1])
                H[j,i+1] = self.pdfnvar(X[j,:], mi, covi)

        return H

    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.
            """

        # Valida os rótulos
        if(self.classification):
            X, y = self._check_X_y(X, y)

        #número de neurônios da camada intermediária
        self.p = p

        # dimensão de entrada
        self.n = X.shape[1]

        # Calcula K-médias para a entrada X
        xclust = KMeans(n_clusters=self.p).fit(X)

        # Armazena vetores de centros das funções.
        self.m = xclust.cluster_centers_
```

```
        # Estima matrizes de covariância para todos os centros.
        for i in range(self.p):
            xci = X[(xclust.labels_== i),:]
            covi = np.cov(xci, rowvar=False)
            self.covlist.append(covi)

        # Calcula matriz H
        H = self.calcH(X)

        # 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, cassification = True):
        """ Faz previsões usando o modelo já ajustado

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

        # Calcula matriz H
        H = self.calcH(X)
        if(self.classification):
            return np.sign(np.dot(H,self.W))
        else:
            return 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]
```

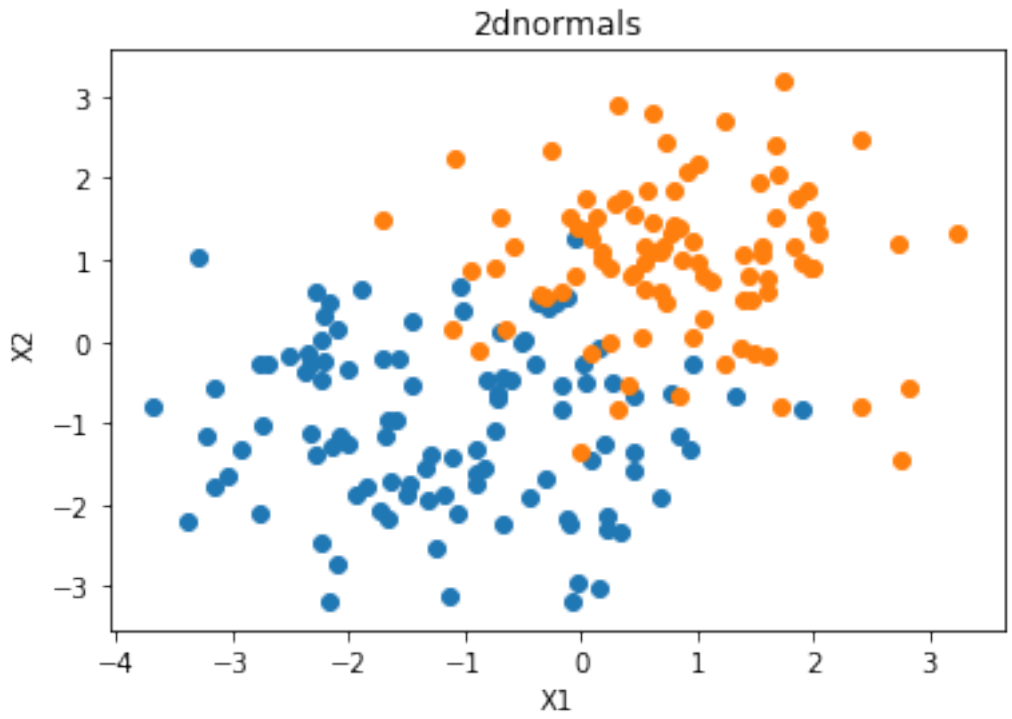
```
    def plot_separacao(self, X, X_1, X_2, ax):
        if(self.n == 2):
            x_1 = np.linspace(np.floor(X[:,0].min()), np.ceil(X[:,0].max()),␣
↪100)
            x_2 = np.linspace(np.floor(X[:,1].min()), np.ceil(X[:,1].max()),␣
↪100)

            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.reshape(np.array([x_2[i],␣
↪x_1[j]]),(1,2)))


            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="binary")
            # ax.set_xlim(x_1.min(), x_1.max())
            # ax.set_ylim(x_2.min(), x_2.max())
            ax.set_title('RBF: p = ' + str(self.p))
        else:
            return
```

2.2 1.1 2dnormals

[3]:
```
# 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
```

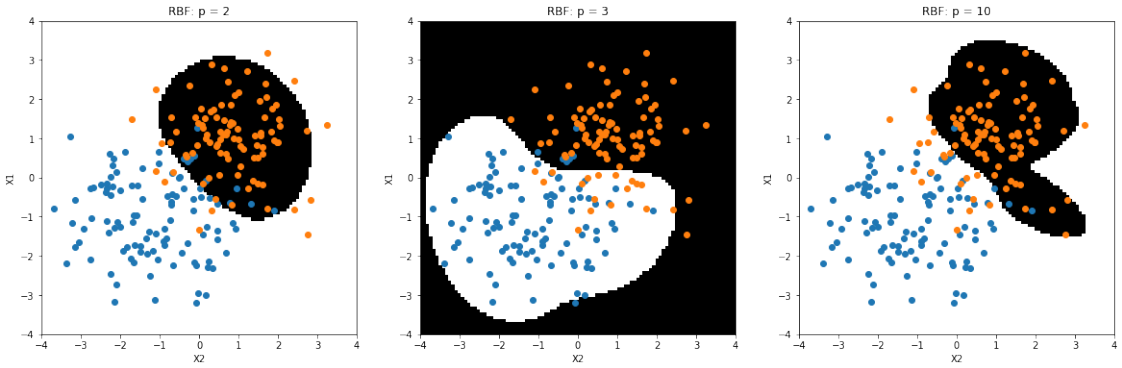
```
plt.scatter(X_1[:,0], X_1[:,1])
plt.scatter(X_2[:,0], X_2[:,1])
plt.title('2dnormals')
plt.xlabel('X1')
plt.ylabel('X2')
```

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



[4]:
```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(20,6))

# Treina o 1º modelo : p = 2
model1 = RBF().fit(X, Y, 2)
acuracia1 = model1.score(X, Y)
model1.plot_separacao(X, X_1, X_2, ax1)

# Treina o 2º modelo : p = 3
model2 = RBF().fit(X, Y, 3)
acuracia2 = model2.score(X, Y)
model2.plot_separacao(X, X_1, X_2, ax2)

# Treina o 3º modelo : p = 10
model3 = RBF().fit(X, Y, 10)
acuracia3 = model3.score(X, Y)
```

```
model3.plot_separacao(X, X_1, X_2, ax3)
```

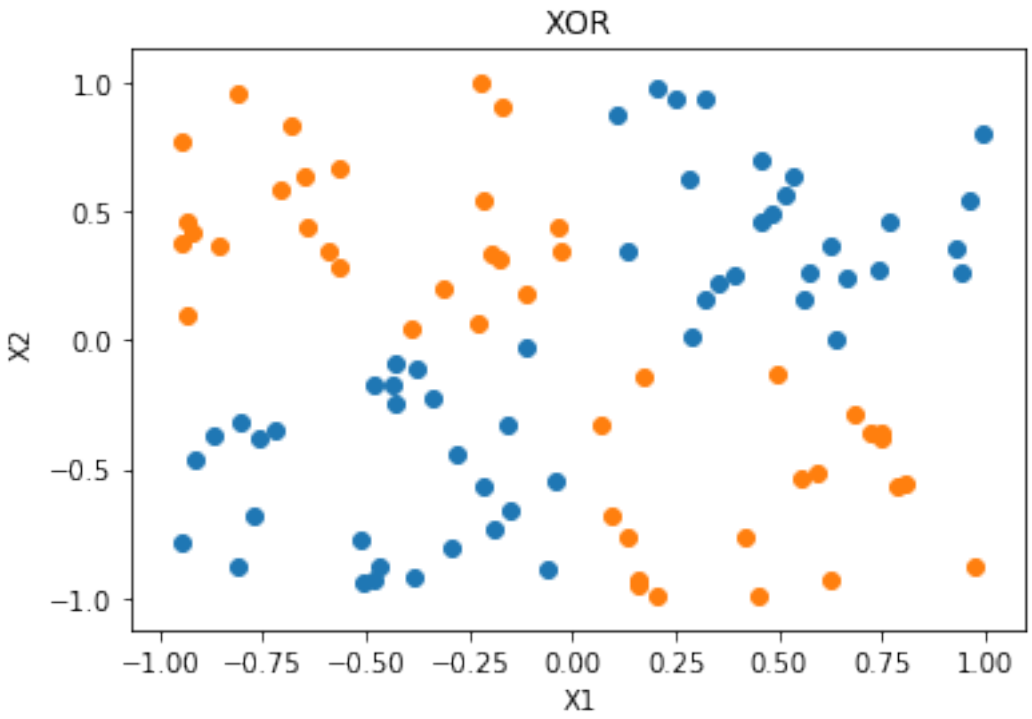


Para todos os valores de p, observa-se que ocorreu sobreajuste, pois não foi gerada a superfície de separação esperada (reta).

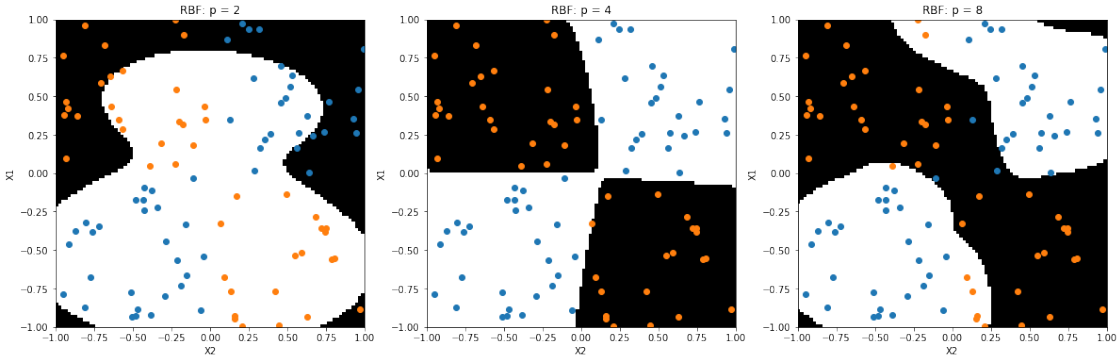
2.3 1.2 XOR

[5]:
```
# 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')
```

[5]:
```
Text(0, 0.5, 'X2')
```



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

# Treina o 1º modelo : p = 2
model1 = RBF().fit(X, Y, 2)
acuracia1 = model1.score(X, Y)
model1.plot_separacao(X, X_1, X_2, ax1)

# Treina o 2º modelo : p = 4
model2 = RBF().fit(X, Y, 4)
acuracia2 = model2.score(X, Y)
model2.plot_separacao(X, X_1, X_2, ax2)

# Treina o 3º modelo : p = 8
model3 = RBF().fit(X, Y, 8)
acuracia3 = model3.score(X, Y)
model3.plot_separacao(X, X_1, X_2, ax3)
```

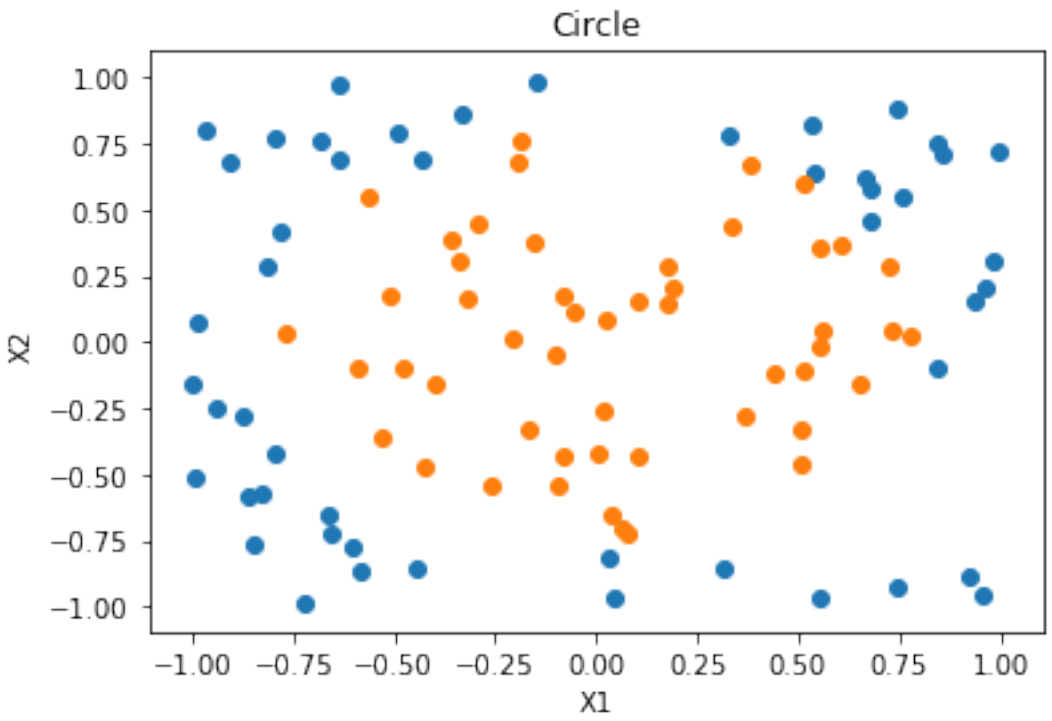


Conforme o esperado, o separador com $p = 4$ obteve melhor desempenho, apesar de ser verificado certo sorbeajuste. Para $p = 2$, ocorreu subajuste e para $p = 8$ o modelo passou a se especialiar nos dados de treinamento, demonstrando forte tendencia sobreajuste caso fossem utilizados mais neurônios na camada intermediária.

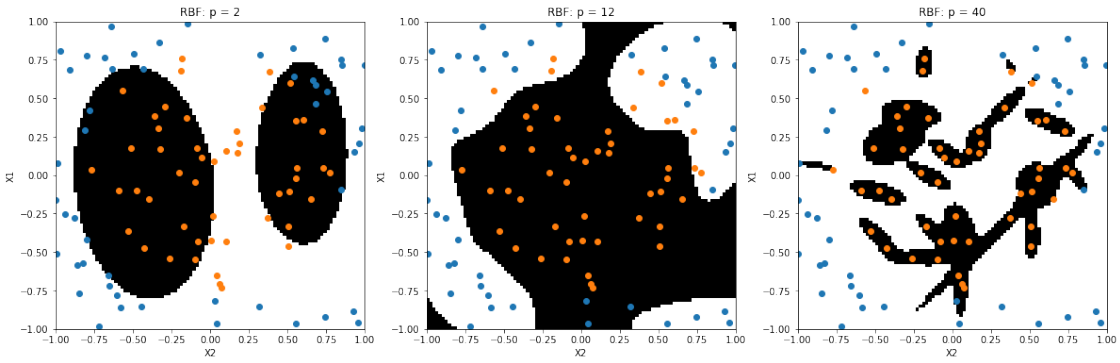
2.4 1.3 Circle

[70]:
```
# 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')
```

[70]:
```
Text(0, 0.5, 'X2')
```



[76]:
```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(20,6))

# Treina o 1º modelo : p = 2
model1 = RBF().fit(X, Y, 2)
acuracia1 = model1.score(X, Y)
model1.plot_separacao(X, X_1, X_2, ax1)

# Treina o 2º modelo : p = 12
model2 = RBF().fit(X, Y, 12)
acuracia2 = model2.score(X, Y)
model2.plot_separacao(X, X_1, X_2, ax2)

# Treina o 3º modelo : p = 40
model3 = RBF().fit(X, Y, 40)
acuracia3 = model3.score(X, Y)
model3.plot_separacao(X, X_1, X_2, ax3)
```

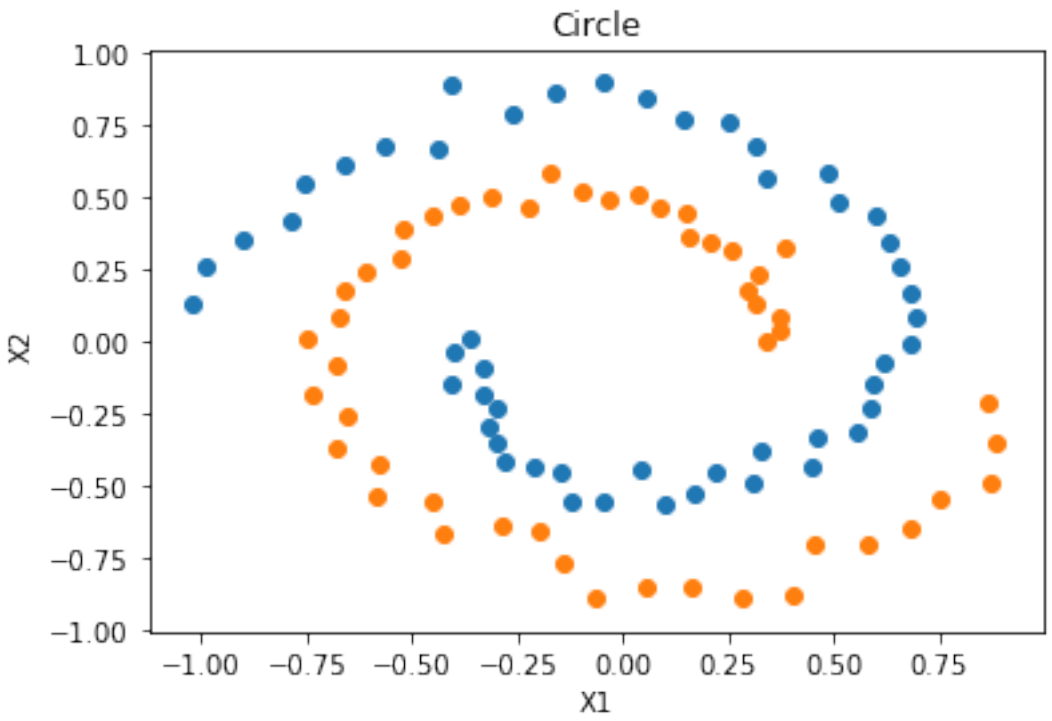


Vizualmente o separador com $p = 12$ obteve melhor generalização, pois a separação é a que mais se assemelha a um circulo. É nítido que para p=*2* ocorreu underfitting e que para $p = 40$ ocorreu overfitting.

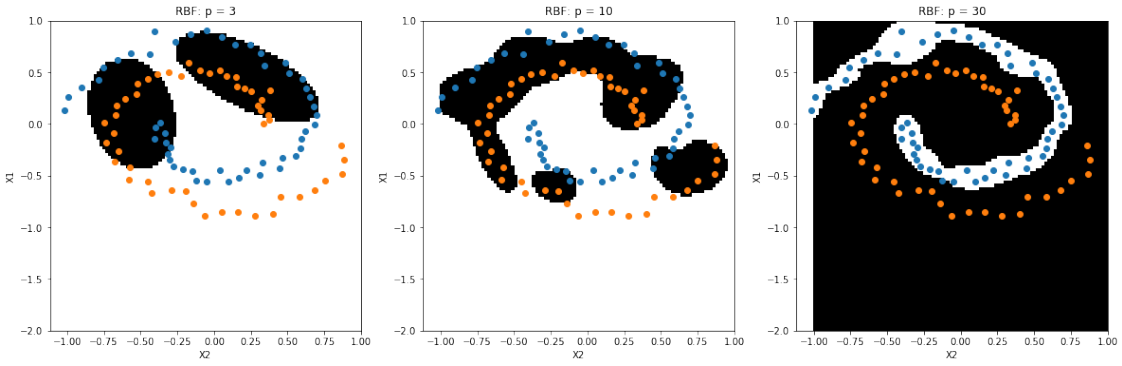
2.5 1.4 Spirals

[9]:
```
# 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')
```

[9]:
```
Text(0, 0.5, 'X2')
```



[10]:
```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(20,6))

# Treina o 1º modelo : p = 3
model1 = RBF().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 = RBF().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 = RBF().fit(X, Y, 30)
acuracia3 = model3.score(X, Y)
model3.plot_separacao(X, X_1, X_2, ax3)
```



Vizualmente o separador com $p = 20$ obteve melhor acurácia, mas observa-se que ocorreu sobreajuste, pois não há uma separação suave dos dados. É nítido que para *p=3* e *p=10* ocorreu underfitting.

Em todos os experimentos acredito que poderiam ser obtidos resultados melhores se fosse realizado alguma regularização, como a regressão ridge (regularização L2).

2.6 2. Sinc

A segunda parte desta atividade consiste em contruir uma rede RBF para aproximar a função *sinc* acrescida de um ruído aussiano:

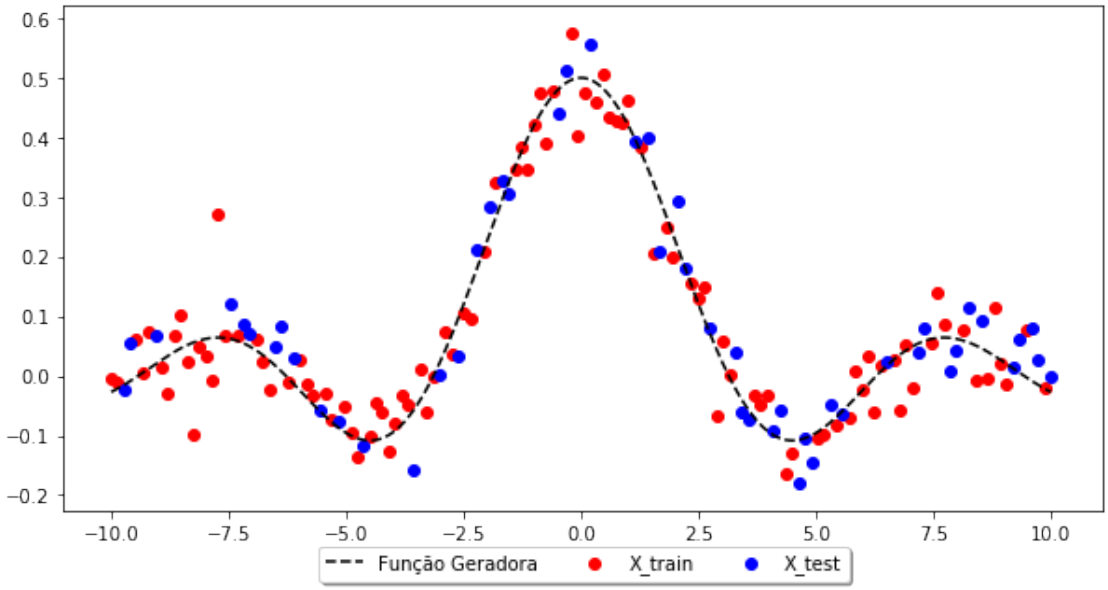
- $sinc(x) = \frac{sin(x)}{x}$

Serão utilizadas 100 amostras (x, y) para treinamento, em que $y = sinc(x) + \epsilon$, com $\epsilon \sim \mathcal{N}(0, 0.05)$. Assim como na primeira parte, serão ajustadas 3 redes RBBFs, com diferentes valores de k (número de centros). As 3 redes devem ter o seu desempenho comparado com um conjunto de 50 amostras (geradas da mesma forma que as amostras de treinamento). A métrica a ser usada é o erro quadrático médio, definido como: $MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

[111]:
```
# 1000 linearly spaced numbers
x = np.linspace(-10, 10,1000)

# the function, which is y
y = (1/2) * np.sin(x)/x

# Amostras - 150 pontos igualmente espaçados entre -10 e 10
X = np.reshape(np.linspace(-10, 10, 150), (150, 1))
Y = (1/2) * np.sin(X)/X + np.random.normal(0,0.05,len(X)).reshape((X.
 ↪shape[0],1))

# Separa os dados de forma aleatória - 66% para treinamento e 33% para testes
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33)

# Gera figura da função geradora juntamente das amostras de treinamento e teste
fig, ax = plt.subplots(1,1, figsize=(10,5))
```

```
ax.scatter(X_train, Y_train, c='red', label='X_train')
ax.scatter(X_test, Y_test, c='blue', label='X_test')
# plot the function
plt.plot(x,y, '--k', label='Função Geradora')

ax.legend(loc='upper center', bbox_to_anchor=(0.5, -0.05), shadow=True, ncol=3)
```

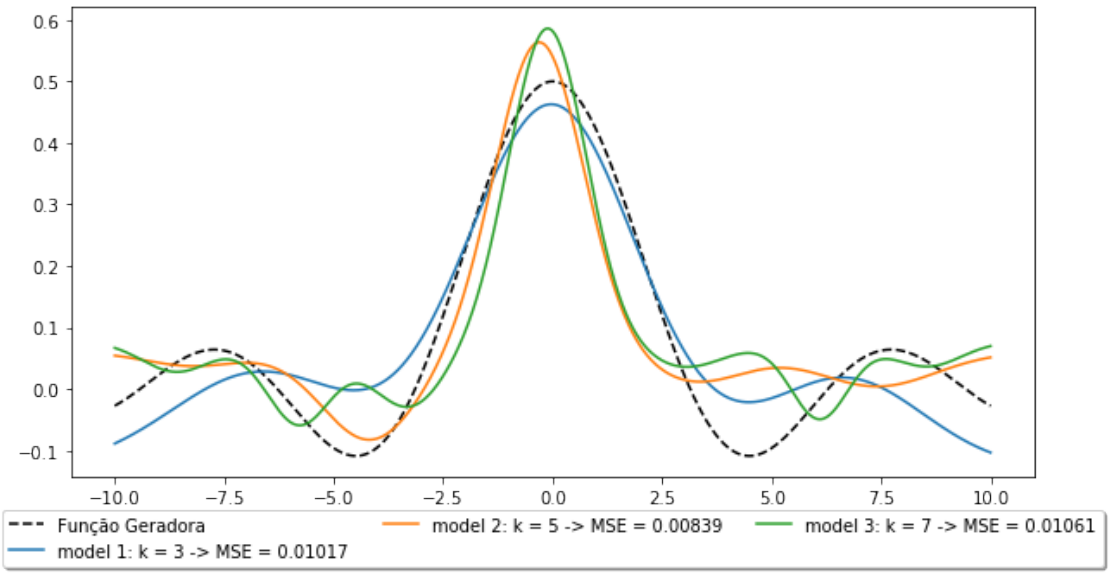
[111]: <matplotlib.legend.Legend at 0x200d98ffb08>



[113]:
```
# Gera figura da função geradora juntamente das regressões dos modelos com␣
 ↪diferentes números de centros
fig, ax = plt.subplots(1,1, figsize=(10,5))

plt.plot(x,y, '--k', label='Função Geradora')

Z = np.zeros((x.shape[0]))
# Treina o 1º modelo : k = 3
model1 = RBF(classification=False).fit(X_train, Y_train, 3)
Y_pred = model1.predict(X_test)
MSE1 = (1/Y_test.shape[0])*np.sum((Y_test-Y_pred)**2)
for i in range(x.shape[0]):
        Z[i] = model1.predict(np.reshape(x[i],(1,1)))
plt.plot(x,Z, label='model 1: k = 3 -> MSE = '+str(round(MSE3,5)))

# Treina o 1º modelo : k = 5
model2 = RBF(classification=False).fit(X_train, Y_train, 5)
```

```
Y_pred = model2.predict(X_test)
MSE2 = (1/Y_test.shape[0])*np.sum((Y_test-Y_pred)**2)
for i in range(x.shape[0]):
        Z[i] = model2.predict(np.reshape(x[i],(1,1)))
plt.plot(x,Z, label='model 2: k = 5 -> MSE = '+str(round(MSE2,5)))

# Treina o 1º modelo : k = 7
model3 = RBF(classification=False).fit(X_train, Y_train, 7)
Y_pred = model3.predict(X_test)
MSE3 = (1/Y_test.shape[0])*np.sum((Y_test-Y_pred)**2)
for i in range(x.shape[0]):
        Z[i] = model3.predict(np.reshape(x[i],(1,1)))
plt.plot(x,Z, label='model 3: k = 7 -> MSE = '+str(round(MSE3,5)))

ax.legend(loc='upper center', bbox_to_anchor=(0.5, -0.05), shadow=True, ncol=3)
```

[113]: <matplotlib.legend.Legend at 0x200d755dd88>



O modelo 2 com 5 centros foi o que apresentou menor erro médio quadrático.