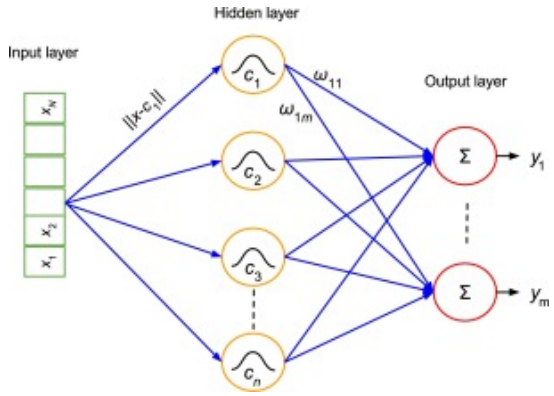
Exercício 7

Arthur Felipe Reis Souza

April 29, 2024

1 Introduction

As redes neurais do tipo RBF (Radial Basis Function) são redes neurais com uma camada de entrada, uma camada intermediária e uma camada de saída. A camada intermediária desse tipo de rede aplica N (N é o número de neurônios da camada intermediária) funções radiais sobre os dados de entrada, gerando uma saída H que alimentará a camada de saída. Uma função radial é uma função que mensura a distância entre os pontos e um valor arbitrário. A distância pode ser entendida como uma medida de dissimilaridade entre dois ou mais valores. Há várias métricas de distâncias que podem ser utilizadas, sendo elas: Euclidean distance, Manhattan distance, Minowski distance (forma generalizada das duas anteriores). Portanto, a ideia principal das redes do tipo RBF é: pegar as entradas da rede e aplicar sobre essa entrada um conjunto de N funções radiais (função gaussiana é comumente utilizada). A saída de cada função radial será combinada linearmente com os parâmetros da camada de saída da rede, que poderá aplicar uma função de ativação sigmoidal ou identidade sobre essa combinação.



Faz-se mister, portanto, encontrar a localização dos centros, ou do valor central dessas funções radiais que irão aplicar, sobre os dados de entrada, uma

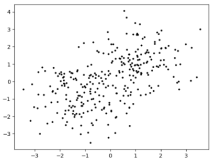
função de dissimilaridade. Para encontrar os pontos centrais dessas funções radiais, foi utilizado um método de clustering (agrupamento) de dados chamado KMeans. O algoritmo de clustering KMeans é um algoritmo de aprendizado não supervisionado (sem informações diretas da saída), e tem por objetivo agrupar os dados em K clusters. A lógica do algoritmo KMeans é: definir as coordenadas iniciais dos K clusters e calcular a distância de cada ponto à esses clusters. Cada ponto será atribuído ao cluster mais próximo, e fazendo isso para todos os pontos teremos K agrupamentos. Após ter todos os pontos agrupados, os centros de cada cluster irão se mover para o ponto médio entre os pontos de cada agrupamento. Esse método é iterativo e não supervisionado, ou seja, irá parar de reagrupar os dados apenas perante a um critério de parada que será obtido com a minimização da discrepância entre as variâncias de cada cluster.

A primeira parte do exercícios 7 consiste em utilizar, nos dados obtidos no último Exercício ELM, o algoritmo KMeans para agrupar os mesmos. Após aplicar o algoritmo KMeans, usando K = 5, obtemos os seguintes resultados abaixo :

1.1 2dnormals

Dados para K = 5, dataset normals :

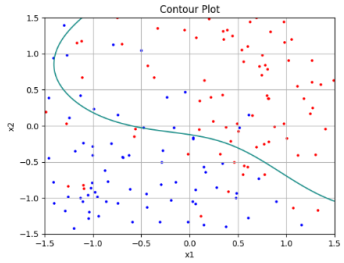
```
[18]: import numpy as np
      import matplotlib.pyplot as plt
      import pandas as pd
      import rpy2.robjects as ro
      from functools import partial
      from rpy2.robjects import pandas2ri
      from rpy2.robjects.packages import importr
      from rpy2.ipython import html
      html.html_rdataframe = partial(html.html_rdataframe, table_class = "docutils")
      %load_ext rpy2.ipython

      The rpy2.ipython extension is already loaded. To reload it, use:
        %reload_ext rpy2.ipython

[19]: %%R
      library(mlbench)
      datasetnormals <- mlbench.2dnormals(300)
      XR_datasetnormals <- datasetnormals$x
      LABELSR_datasetnormals <- datasetnormals$classes

[20]: from rpy2.robjects import numpy2ri
      numpy2ri.activate()
      x_df_normals = np.array(ro.r['XR_datasetnormals'])
      labels_df_normals = np.array(ro.r['LABELSR_datasetnormals'])
      labels_df_normals[labels_df_normals == 1] = -1
      labels_df_normals[labels_df_normals == 2] = 1

[21]: plt.scatter(x_df_normals[:, 0], x_df_normals[:, 1], color = 'black', s = 5)
      plt.show()
```



```
[25]: r = 0.6
      p = cluster_centers.shape[0]
      m = ret_[0]
      covi = ret_[1]
      r = ret_[2]
      W = ret_[3]
      H = ret_[4]

[26]: plt.scatter(X_train[:, 0], X_train[:, 1], c=cluster_labels, cmap='viridis', marker='o', edgecolors='k')
      plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1], c='red', marker='x', s=100, label='Cluster Centers')

      # Add labels and legend
      plt.xlabel('Feature 1')
      plt.ylabel('Feature 2')
      plt.title('KMeans Clustering')
      plt.legend()

      # Show the plot
      plt.show()
```

A superfície de separação para K = 5 está registrada abaixo :

```
[53]: plt.contour(seqx1x2, seqx1x2, MZ,  levels = 1)
      plt.scatter(x1_points[:, 0], x1_points[:, 1], color = 'red', label = 'data1', s = 5)
      plt.scatter(x2_points[:, 0], x2_points[:, 1], color = 'blue', label = 'data2', s = 5)
      plt.xlabel('x1')
      plt.ylabel('x2')
      plt.title('Contour Plot')
      plt.xlim(-1.5, 1.5)
      plt.ylim(-1.5, 1.5)
      plt.grid(True)
      plt.show()
```



Aumentando muito o valor de K teoricamente levaria o modelo ao overfitting. Porém, esse conjunto de dados é facilmente separável, e não foi possível observar um overfitting visualmente, porém com um K muito grande o modelo erra mais nos dados de teste. O valor de K que levará o modelo a ter uma melhor performance é obtido com o algoritmo de Grid Search Cross-Validation, e está registrado abaixo :

```
[107]: from sklearn.metrics import accuracy_score
       lst_acc_test = list()
       lst_param = list()

       for i in range(1, 200, 1):
           kmeans = KMeans(n_clusters = i, random_state = 0, n_init = 'auto').fit(X_train)
           cluster_labels = kmeans.labels_
           cluster_centers = kmeans.cluster_centers_
           ret_ = train_RBF.trainRBF(X_train, y_train, i, r)

           yhat_test = train_RBF.y_RBF(X_test, ret_)
           yhat_test[yhat_test <= -0.5] = -1
           yhat_test[yhat_test > -0.5] = 1
           yhat_train = train_RBF.y_RBF(X_train, ret_)
           yhat_train[yhat_train <= -0.5] = -1
           yhat_train[yhat_train > -0.5] = 1
           lst_acc_test.append(accuracy_score(y_test, yhat_test))
           lst_param.append(i)
       lst_acc_test = np.array(lst_acc_test)
       lst_param = np.array(lst_param)
```

```
H shape : (210, 1)
Haug shape : (210, 2)
yin shape : (210,)
W shape : (2,)
H shape : (210, 2)
Haug shape : (210, 3)
yin shape : (210,)
W shape : (3,)
H shape : (210, 3)
Haug shape : (210, 4)
yin shape : (210,)
W shape : (4,)
H shape : (210, 4)
Haug shape : (210, 5)
yin shape : (210,)
W shape : (5,)
H shape : (210, 5)
Haug shape : (210, 6)
```

```
[108]: idx = np.argmax(lst_acc_test)
       print(f"The value of K that give us the best performance in test data is : {lst_param[idx]}")
```

```
The value of K that give us the best performance in test data is : 10
```

1.2 xor

Dados para K = 5, dataset xor :

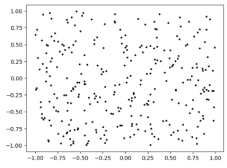
```
[17]: import numpy as np
      import matplotlib.pyplot as plt
      import pandas as pd
      import rpy2.robjects as ro
      from functools import partial
      from rpy2.robjects import pandas2ri
      from rpy2.robjects.packages import importr
      from rpy2.ipython import html
      html.html_rdataframe = partial(html.html_rdataframe, table_class = "docutils")
      %load_ext rpy2.ipython
```

```
The rpy2.ipython extension is already loaded. To reload it, use:
  %reload_ext rpy2.ipython
```

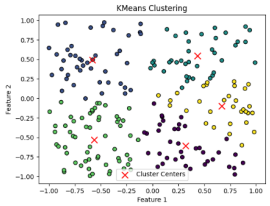
```
[18]: %%R
      library(mlbench)
      datasetxor <- mlbench.xor(300)
      XR_datasetxor <- datasetxor$x
      LABELSR_datasetxor<- datasetxor$classes
```

```
[19]: from rpy2.robjects import numpy2ri
      numpy2ri.activate()
      x_df_xor = np.array(ro.r['XR_datasetxor'])
      labels_df_xor = np.array(ro.r['LABELSR_datasetxor'])
      labels_df_xor[labels_df_xor == 1] = -1
      labels_df_xor[labels_df_xor == 2] = 1
```

```
[20]: plt.scatter(x_df_xor[:, 0], x_df_xor[:, 1], color = 'black', s = 5)
      plt.show()
```

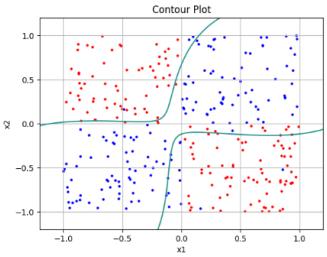


```
[21]: from sklearn.model_selection import train_test_split
      X_train, X_test, y_train, y_test = train_test_split(x_df_xor, labels_df_xor, random_state = 0, train_size = 0.7)

[22]: from sklearn.cluster import KMeans
      kmeans = KMeans(n_clusters = 5, random_state = 0, n_init = 'auto').fit(X_train)
      cluster_labels = kmeans.labels_
      cluster_centers = kmeans.cluster_centers_
      p = cluster_centers.shape[0]

[23]: plt.scatter(X_train[:, 0], X_train[:, 1], c=cluster_labels, cmap='viridis', marker='o', edgecolors='k')
      plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1], c='red', marker='x', s=100, label='Cluster Centers')

      # Add labels and legend
      plt.xlabel('Feature 1')
      plt.ylabel('Feature 2')
      plt.title('KMeans Clustering')
      plt.legend()

      # Show the plot
      plt.show()
```



A superfície de separação para K = 5 está registrada abaixo :

```
[190]: plt.contour(seqx1x2, seqx1x2, MZ,  levels = 1)
       plt.scatter(x1_points[:, 0], x1_points[:, 1], color = 'red', label = 'data1', s = 5)
       plt.scatter(x2_points[:, 0], x2_points[:, 1], color = 'blue', label = 'data2', s = 5)
       plt.xlabel('x1')
       plt.ylabel('x2')
       plt.title('Contour Plot')
       plt.xlim(-1.2, 1.2)
       plt.ylim(-1.2, 1.2)
       plt.grid(True)
       plt.show()
```



Aumentando muito o valor de K teoricamente levaria o modelo ao overfitting. O valor ideal de K é obtido com o metodo Grid Search CV e está registrado abaixo :

```
[201]: from sklearn.metrics import accuracy_score
       lst_acc_test = list()
       lst_param = list()

       for i in range(1, 200, 1):
           kmeans = KMeans(n_clusters = i, random_state = 0, n_init = 'auto').fit(X_train)
           cluster_labels = kmeans.labels_
           cluster_centers = kmeans.cluster_centers_
           ret_ = train_RBF.trainRBF(X_train, y_train, i, r)


           yhat_test = train_RBF.y_RBF(X_test, ret_)
           yhat_test[yhat_test <= -0.5] = -1
           yhat_test[yhat_test > -0.5] = 1
           yhat_train = train_RBF.y_RBF(X_train, ret_)
           yhat_train[yhat_train <= -0.5] = -1
           yhat_train[yhat_train > -0.5] = 1
           lst_acc_test.append(accuracy_score(y_test, yhat_test))
           lst_param.append(i)
       lst_acc_test = np.array(lst_acc_test)
       lst_param = np.array(lst_param)
```

```
M shape : (210, 1)
Maug shape : (210, 2)
yin shape : (210,)
W shape : (2,)
H shape : (210, 2)
Maug shape : (210, 3)
yin shape : (210,)
W shape : (3,)
H shape : (210, 3)
Maug shape : (210, 4)
yin shape : (210,)
W shape : (4,)
H shape : (210, 4)
Maug shape : (210, 5)
yin shape : (210,)
W shape : (5,)
H shape : (210, 5)
Maug shape : (210, 6)
```

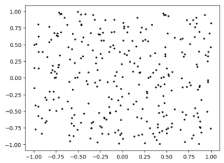
```
[202]: idx = np.argmax(lst_acc_test)
       print(f"The value of K that give us the best performance in test data is : {lst_param[idx]}")
```

```
The value of K that give us the best performance in test data is : 44
```

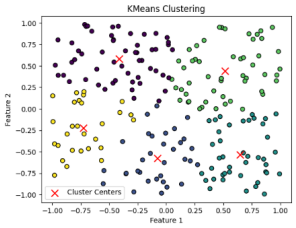
1.3 circle

Dados para K = 5, dataset circle :

```
[1]: import numpy as np
     import matplotlib.pyplot as plt
     import pandas as pd
     import rpy2.robjects as ro
     from functools import partial
     from rpy2.robjects import pandas2ri
     from rpy2.robjects.packages import importr
     from rpy2.ipython import html
     html.html_rdataframe = partial(html.html_rdataframe, table_class = "docutils")
     %load_ext rpy2.ipython

[2]: %%R
     library(mlbench)
     datasetcircle <- mlbench.circle(300)
     XR_datasetcircle <- datasetcircle$x
     LABELSR_datasetcircle <- datasetcircle$classes

     In addition: Warning message:
     In (function (package, help, pos = 2, lib.loc = NULL, character.only = FALSE,  :
       'library "/usr/lib/R/site-library" contains no packages

[3]: from rpy2.robjects import numpy2ri
     numpy2ri.activate()
     x_df_circle = np.array(ro.r('XR_datasetcircle'))
     labels_df_circle = np.array(ro.r('LABELSR_datasetcircle'))
     labels_df_circle[labels_df_circle == 1] = -1
     labels_df_circle[labels_df_circle == 2] = 1

[4]: plt.scatter(x_df_circle[:, 0], x_df_circle[:, 1], color = 'black', s = 5)
     plt.show()
```

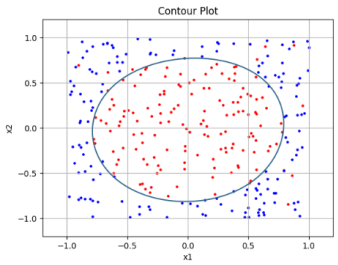


```
[6]: from sklearn.cluster import KMeans
     kmeans = KMeans(n_clusters = 5, random_state = 0, n_init = 'auto').fit(X_train)
     cluster_labels = kmeans.labels_
     cluster_centers = kmeans.cluster_centers_

[7]: plt.scatter(X_train[:, 0], X_train[:, 1], c=cluster_labels, cmap='viridis', marker='o', edgecolors='k')
     plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1], c='red', marker='x', s=100, label='Cluster Centers')

     # Add labels and legend
     plt.xlabel('Feature 1')
     plt.ylabel('Feature 2')
     plt.title('KMeans Clustering')
     plt.legend()

     # Show the plot
     plt.show()
```



A superfície de separação para K = 5 está registrada abaixo :

```
[46]: plt.contour(seqx1x2, seqx1x2, MZ,  levels = 1)
plt.scatter(x1_points[:, 0], x1_points[:, 1], color = 'red', label = 'data1', s = 5)
plt.scatter(x2_points[:, 0], x2_points[:, 1], color = 'blue', label = 'data2', s = 5)
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Contour Plot')
plt.xlim(-1.2, 1.2)
plt.ylim(-1.2, 1.2)
plt.grid(True)
plt.show()
```



Aumentando muito o valor de K teoricamente levaria o modelo ao overfitting. O valor ideal de K é obtido com o metodo Grid Search CV e está registrado abaixo :

```
[56]: from sklearn.metrics import accuracy_score
lst_acc_test = list()
lst_param = list()

for i in range(1, 200, 1):
    kmeans = KMeans(n_clusters = i, random_state = 0, n_init = 'auto').fit(X_train)
    cluster_labels = kmeans.labels_
    cluster_centers = kmeans.cluster_centers_
    ret_ = train_RBF.trainRBF(X_train, y_train, i, r)

    yhat_test = train_RBF.y_RBF(X_test, ret_)
    yhat_test[yhat_test <= -0.5] = -1
    yhat_test[yhat_test > -0.5] = 1
    yhat_train = train_RBF.y_RBF(X_train, ret_)
    yhat_train[yhat_train <= -0.5] = -1
    yhat_train[yhat_train > -0.5] = 1
    lst_acc_test.append(accuracy_score(y_test, yhat_test))
    lst_param.append(i)
lst_acc_test = np.array(lst_acc_test)
lst_param = np.array(lst_param)
```

```
H shape : (210, 1)
Haug shape : (210, 2)
yin shape : (210,)
W shape : (2,)
H shape : (210, 2)
Haug shape : (210, 3)
yin shape : (210,)
W shape : (3,)
H shape : (210, 3)
Haug shape : (210, 4)
yin shape : (210,)
W shape : (4,)
H shape : (210, 4)
Haug shape : (210, 5)
yin shape : (210,)
W shape : (5,)
H shape : (210, 5)
Haug shape : (210, 6)
```

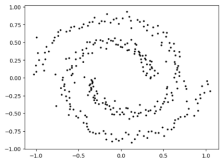
```
[57]: idx = np.argmax(lst_acc_test)
print(f"The value of K that give us the best performance in test data is : {lst_param[idx]}")
```

```
The value of K that give us the best performance in test data is : 82
```

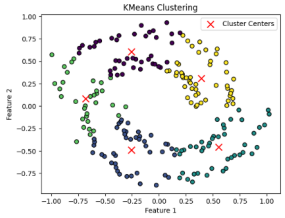
1.4 spirals

Dados para K = 5, dataset spirals :

```
[1]: import numpy as np
     import matplotlib.pyplot as plt
     import pandas as pd
     import rpy2.robjects as ro
     from functools import partial
     from rpy2.robjects import pandas2ri
     from rpy2.robjects.packages import importr
     from rpy2.ipython import html
     html.html_rdataframe = partial(html.html_rdataframe, table_class = "docutils")
     %load_ext rpy2.ipython

[2]: %%R
     library(mlbench)
     datasetspirals <- mlbench.spirals(300, sd = 0.05)
     XR_datasetspirals <- datasetspirals$x
     LABELSR_datasetspirals <- datasetspirals$classes

     In addition: Warning message:
     In (function (package, help, pos = 2, lib.loc = NULL, character.only = FALSE,  :
       library '/usr/lib/R/site-library' contains no packages

[3]: from rpy2.robjects import numpy2ri
     numpy2ri.activate()
     X_df_espirals = np.array(ro.r['XR_datasetspirals'])
     labels_df_espirals = np.array(ro.r['LABELSR_datasetspirals'])
     labels_df_espirals[labels_df_espirals == 1] = -1
     labels_df_espirals[labels_df_espirals == 2] = 1

[4]: plt.scatter(X_df_espirals[:, 0], X_df_espirals[:, 1], color = 'black', s = 5)
     plt.show()
```

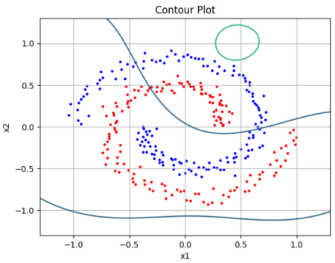


```
[6]: from sklearn.cluster import KMeans
     kmeans = KMeans(n_clusters = 5, random_state = 0, n_init = 'auto').fit(X_train)
     cluster_labels = kmeans.labels_
     cluster_centers = kmeans.cluster_centers_
     p = cluster_centers.shape[0]

[7]: plt.scatter(X_train[:, 0], X_train[:, 1], c=cluster_labels, cmap='viridis', marker='o', edgecolors='k')
     plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1], c='red', marker='x', s=100, label='Cluster Centers')

     # Add labels and legend
     plt.xlabel('Feature 1')
     plt.ylabel('Feature 2')
     plt.title('KMeans Clustering')
     plt.legend()

     # Show the plot
     plt.show()
```



A superfície de separação para K = 5 está registrada abaixo :

```
[102]: plt.contour(seqx1x2, seqx1x2, MZ,  levels = 1)
       plt.scatter(x1_points[:, 0], x1_points[:, 1], color = 'red', label = 'data1', s = 5)
       plt.scatter(x2_points[:, 0], x2_points[:, 1], color = 'blue', label = 'data2', s = 5)
       plt.xlabel('x1')
       plt.ylabel('x2')
       plt.title('Contour Plot')
       plt.xlim(-1.3, 1.3)
       plt.ylim(-1.3, 1.3)
       plt.grid(True)
       plt.show()
```



Aumentando muito o valor de K teoricamente levaria o modelo ao overfitting. Como pode ser observado para K = 5, o modelo não faz uma boa aproximação.

Para encontrar o melhor valor de K foi utilizado o algoritmo de Grid, que registra abaixo o melhor valor de K e também a superfície de separação para esse valor de K.

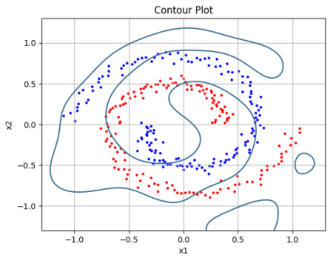
```
[113]: from sklearn.metrics import accuracy_score
       lst_acc_test = list()
       lst_param = list()
       for i in range(1, 200, 1):
           kmeans = KMeans(n_clusters = i, random_state = 0, n_init = 'auto').fit(X_train)
           cluster_labels = kmeans.labels_
           cluster_centers = kmeans.cluster_centers_
           ret_ = train_RBF.trainRBF(X_train, y_train, i, r)

           yhat_test = train_RBF.y_RBF(X_test, ret_)
           yhat_test[yhat_test <= -0.5] = -1
           yhat_test[yhat_test > -0.5] = 1
           yhat_train = train_RBF.y_RBF(X_train, ret_)
           yhat_train[yhat_train <= -0.5] = -1
           yhat_train[yhat_train > -0.5] = 1
           lst_acc_test.append(accuracy_score(y_test, yhat_test))
           lst_param.append(i)
       lst_acc_test = np.array(lst_acc_test)
       lst_param = np.array(lst_param)
```

```
H shape : (210, 1)
Haug shape : (210, 2)
yin shape : (210,)
W shape : (2,)
H shape : (210, 2)
Haug shape : (210, 3)
yin shape : (210,)
W shape : (3,)
H shape : (210, 3)
Haug shape : (210, 4)
yin shape : (210,)
W shape : (4,)
H shape : (210, 4)
Haug shape : (210, 5)
yin shape : (210,)
W shape : (5,)
H shape : (210, 5)
Haug shape : (210, 6)
```

```
[114]: idx = np.argmax(lst_acc_test)
       print(f"The value of K that give us the best performance in test data is : {lst_param[idx]}")
```

```
The value of K that give us the best performance in test data is : 33
```

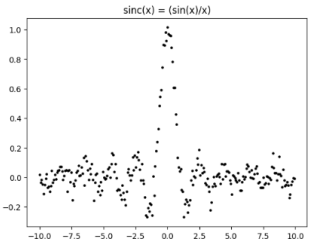
```
[144]: plt.contour(seqx1x2, seqx1x2, MZ,  levels = 1)
       plt.scatter(x1_points[:, 0], x1_points[:, 1], color = 'red', label = 'data1', s = 5)
       plt.scatter(x2_points[:, 0], x2_points[:, 1], color = 'blue', label = 'data2', s = 5)
       plt.xlabel('x1')
       plt.ylabel('x2')
       plt.title('Contour Plot')
       plt.xlim(-1.3, 1.3)
       plt.ylim(-1.3, 1.3)
       plt.grid(True)
       plt.show()
```



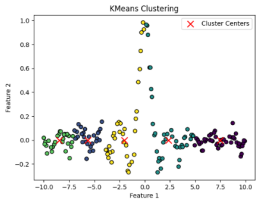
2 Sinc(x)

A figura abaixo retrata os dados gerados por uma função sinc(x) com 1 certo ruído gaussiano. Também mostra o resultado do clustering através do algoritmo KMeans :

```
[1]: import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt

[2]: x_sinc = np.arange(start = -10, stop = 10, step = 0.08)
     y_sinc = np.sinc(x_sinc)
     y_noise = y_sinc + np.random.normal(loc = 0, scale = 0.05, size = 250)

[3]: plt.scatter(x_sinc, y_noise, color = 'black', s = 5)
     plt.title('sinc(x) = (sin(x)/x)')
     plt.show()
```




```
[4]: from sklearn.model_selection import train_test_split
     X_train, X_test, y_train, y_test = train_test_split(x_sinc, y_noise, random_state = 0, train_size = 0.7)

[5]: from sklearn.cluster import KMeans
     X_train = X_train.reshape(-1, 1)
     y_train = y_train.reshape(-1, 1)
     kmeans = KMeans(n_clusters = 5, random_state = 0, n_init = 'auto').fit(X_train)
     cluster_labels = kmeans.labels_
     cluster_centers = kmeans.cluster_centers_

[6]: plt.scatter(X_train, y_train, c=cluster_labels, cmap='viridis', marker='o', edgecolors='k')
     plt.scatter(cluster_centers[:, 0], np.zeros((cluster_centers.shape[0])), c='red', marker='x', s=100, label='Cluster Centers')

     # Add labels and legend
     plt.xlabel('Feature 1')
     plt.ylabel('Feature 2')
     plt.title('KMeans Clustering')
     plt.legend()

     # Show the plot
     plt.show()
```

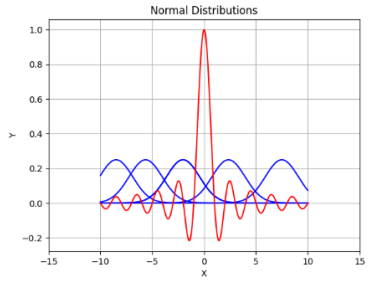


O resultado do erro para K = 5 neurônios está registrado abaixo :

```
[9]: yhat_test = train_RBF.y_RBF(X_test, ret_)
     yhat_train = train_RBF.y_RBF(X_train, ret_)

     You're having the error tuple index out of range. So we will change the numer of dimensions...

[10]: from sklearn.metrics import mean_squared_error
      error_train = mean_squared_error(y_train, yhat_train)
      error_test = mean_squared_error(y_test, yhat_test)
      print(f"The train error is {error_train}")
      print(f"The test error is {error_test}")

      The train error is 0.0440084446602437
      The test error is 0.05120952912089694
```

O plot abaixo mostra as funções gaussianas utilizadas na camada intermediária, e também a função sinc(x) que gerou os dados.

```
[146]: create_graphics(cluster_centers)  # Plot normal distributions with 5 centers
plt.xlabel('X')
plt.xlim(-15, 15)
plt.ylabel('Y')
plt.title('Normal Distributions')
plt.grid(True)
plt.show()
```



É possível observar que, aumentando o número de funções radiais (neuronios da camada intermediária), o modelo tenderá a ter um overfitting. O valor ideal de K, usando o algoritmo de Grid Search foi de 24 neurônios radiais. A imagem abaixo mostra o algoritmo de Grid Search para encontrar o valor de K o qual o modelo terá a melhor performance.

```
[34]: from sklearn.metrics import accuracy_score
lst_error_test = list()
lst_param = list()
for i in range(1, 200, 1):
    kmeans = KMeans(n_clusters = i, random_state = 0, n_init = 'auto').fit(X_train)
    cluster_labels = kmeans.labels_
    cluster_centers = kmeans.cluster_centers_
    ret_ = train_RBF.trainRBF(X_train, y_train, i, 0.6)
    yhat_test = train_RBF.y_RBF(X_test, ret_)

    error_test = mean_squared_error(y_test, yhat_test)
    lst_error_test.append(error_test)
    lst_param.append(i)
lst_error_test = np.array(lst_acc_test)
lst_param = np.array(lst_param)
```

```
yin shape : (175, 1)
W shape : (5, 1)
You're having the error tuple index out of range. So we will change the numer of dimensions...
H shape : (175, 5)
Haug shape : (175, 6)
yin shape : (175, 1)
W shape : (6, 1)
You're having the error tuple index out of range. So we will change the numer of dimensions...
H shape : (175, 6)
Haug shape : (175, 7)
yin shape : (175, 1)
W shape : (7, 1)
You're having the error tuple index out of range. So we will change the numer of dimensions...
H shape : (175, 7)
Haug shape : (175, 8)
yin shape : (175, 1)
W shape : (8, 1)
You're having the error tuple index out of range. So we will change the numer of dimensions...
H shape : (175, 8)
```

```
[36]: idx = np.argmin(lst_error_test)
print(f"Using {lst_param[idx]} radial neurons in the hidden layer the model has the better performance.")
```

```
Using 24 radial neurons in the hidden layer the model has the better performance.
```