RBF

January 8, 2022

1 Exercício 8 - Radial Basis Functons Neural Networks

Aluno: Vítor Gabriel Reis Caitité
Matrícula: 2021712430

1.1 Objetivos

O objetivo do exercício desta semana é combinar os conceitos aprendidos na Unidade 2 e construir uma rede neural que soma elementos das redes RBF e das redes ELM.

As bases de dados a serem estudadas, que pertencem ao repositório UCI Machine Learning Repository [1], são:

- Breast Cancer (diagnostic)
- Statlog (Heart)

Para o exercício desta semana, será construída uma rede RBF com centros e raios atribuídos de forma aleatória aos neurônios.

Além da RBF com centros e raios aleatórios, será construída uma RBF com centros e raios selecionados a partir do k-médias. As acurácias obtidas por cada uma das redes nas duas bases serão apresentadas no formato media +/- desvio e comparadas com os resultados obtidos no exercício 6 para ELMs.

Além disso, será comparado, também, o número de centros necessários para desempenho semelhante entre as redes RBF com centros aleatórios e com centros selecionados por agrupamento (k-médias).

1.2 Importando Bibliotecas e Datasets

[23]:
```python
# Imports
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import math
from sklearn.metrics import confusion_matrix, classification_report,␣
 ↪accuracy_score
from sklearn import preprocessing
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
```

```python
from scipy.spatial import distance
import random
```

1.3 Implementação da Rede RBF com centros e raios selecionados a partir do K-means

A função abaixo é uma implementação possível, em Python, para o algoritmo de treinamento de uma rede RBF com centros e raios selecionados a partir do algoritmo K-means.

[24]:
```python
class RBF:

    def __init__(self, n_neurons):
        self.n_neurons = n_neurons

    def pdfnvar(self, x, m, K, n):
        if n==1:
            r = np.sqrt(K)
            px = (1/(math.sqrt(2*math.pi*r*r))) * math.e**(-0.5 * ((x - m)/
↪r)**2)
        else:
            px = 1/math.sqrt((2*math.pi)**n * np.linalg.det(K)) * math.e ** (-0.
↪5 * np.dot(np.dot(np.transpose(x - m), np.linalg.inv(K)), x-m))
        return px

    def fit(self, X, y):
        N = X.shape[0] # number of samples
        n = X.shape[1] # samples dimension
        # Applying K-mean to separate the clusters:
        kmeans = KMeans(n_clusters=self.n_neurons).fit(X)
        # Capture the centers:
        self.m = kmeans.cluster_centers_
        # Estimate the covariance matrix for all centers:
        self.cov_list = []
        for i in range(0, self.n_neurons):
            ici = np.where(kmeans.labels_ == i)
            Xci = X[ici, :].reshape(ici[0].shape[0], n)
            if n == 1:
                covi = np.var(Xci)
            else:
                covi = np.cov(Xci, rowvar=False)
            self.cov_list.append(covi)
        H = np.ones((N, self.n_neurons+1))
        for j in range(N):
            for i in range(self.n_neurons):
                mi = self.m[i,:]
                covi = self.cov_list[i] + 0.001 * np.diag(np.ones(n))
                H[j, i+1] = self.pdfnvar(X[j, ], mi, covi, n)
        self.W = np.dot(np.linalg.pinv(H), y)
```

```
    def predict(self, X):
        N = X.shape[0] # number of samples
        n = X.shape[1] # samples dimension
        H = np.ones((N, self.n_neurons+1))
        for j in range(N):
            for i in range(self.n_neurons):
                mi = self.m[i,:]
                covi = self.cov_list[i] + 0.001 * np.diag(np.ones(n))
                H[j, i+1] = self.pdfnvar(X[j, ], mi, covi, n)
        y_hat = np.dot(H, self.W)
        return y_hat
```

1.4 Implementação da Rede RBF com centros e raios atribuídos de forma aleatória aos neurônios

A função abaixo é uma implementação possível, em Python, para o algoritmo de treinamento de uma rede RBF com centros e raios atribuídos de forma aleatória aos neurônios. A estratégia utilizada para a construção dos centros foi colocá-los entre 2 pontos escolhidos aleatoriamente do conjunto de treinamento, com o raio da função igual à distância entre os pontos.

[39]:
```
import random
class random_RBF:

    def __init__(self, n_neurons):
        self.n_neurons = n_neurons

    def pdfnvar(self, x, m, K, n):
        if n==1:
            r = np.sqrt(K)
            px = (1/(math.sqrt(2*math.pi*r*r))) * math.e**(-0.5 * ((x - m)/
 ↪r)**2)
        else:
            px = 1/math.sqrt((2*math.pi)**n * np.linalg.det(K)) * math.e ** (-0.
 ↪5 * np.dot(np.dot(np.transpose(x - m), np.linalg.inv(K)), x-m))
        return px

    def fit(self, X, y):
        N = X.shape[0] # number of samples
        n = X.shape[1] # samples dimension

        center = np.zeros((self.n_neurons, n))
        radius = np.zeros(self.n_neurons)
        for index in range(self.n_neurons):
            # Escolhendo 2 pontos aleatoriamente (e garantindo que eles são␣
 ↪diferentes):
            point1 = random.randint(0, N-1)
```

```
            point2 = random.randint(0, N-1)
            while (point1 == point2):
                point2 = random.randint(0, N-1)
            point1 = X[point1,]
            point2 = X[point2,]
            # Centers e Radius:
            center[index,] = (point1+point2)/2
            radius[index] = distance.euclidean(point1, point2)

        # Capture the centers:
        self.m = center
        # Estimate the covariance matrix for all centers:
        self.cov_list = []
        for i in range(0, self.n_neurons):
            ici = np.zeros(N)
            for index in range(N):
                if distance.euclidean(center[i, ], X[index, ]) <= radius[i]:
                    ici[index] = 1
            ici = np.where(ici==1)
            Xci = X[ici, :].reshape(ici[0].shape[0], n)
            if n == 1:
                covi = np.var(Xci)
            else:
                covi = np.cov(Xci, rowvar=False)
            self.cov_list.append(covi)
        H = np.ones((N, self.n_neurons+1))
        for j in range(N):
            for i in range(self.n_neurons):
                mi = self.m[i,:]
                covi = self.cov_list[i] + 0.001 * np.diag(np.ones(n))
                H[j, i+1] = self.pdfnvar(X[j, ], mi, covi, n)
        self.W = np.dot(np.linalg.pinv(H), y)

    def predict(self, X):
        N = X.shape[0] # number of samples
        n = X.shape[1] # samples dimension
        H = np.ones((N, self.n_neurons+1))
        for j in range(N):
            for i in range(self.n_neurons):
                mi = self.m[i,:]
                covi = self.cov_list[i] + 0.001 * np.diag(np.ones(n))
                H[j, i+1] = self.pdfnvar(X[j, ], mi, covi, n)
        y_hat = np.dot(H, self.W)
        return y_hat
```

1.5 Função para captação de resultados

[40]:
```python
def results(X, y, max_iterations, p):
    train_accuracy_RBF = np.zeros(max_iterations)
    test_accuracy_RBF = np.zeros(max_iterations)
    train_accuracy_RBF2 = np.zeros(max_iterations)
    test_accuracy_RBF2 = np.zeros(max_iterations)

    for i in range(0, max_iterations):
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
        # Normalizing data:
        normalizer = preprocessing.Normalizer()
        X_train = normalizer.fit_transform(X_train)
        X_test = normalizer.transform(X_test)

        # RBF
        clf = RBF(p)
        clf.fit(X_train, y_train)
        y_hat_train = clf.predict(X_train)
        y_hat_train = (1*(y_hat_train >= 0)-0.5)*2
        y_hat = clf.predict(X_test)
        y_hat = (1*(y_hat >= 0)-0.5)*2
        train_accuracy_RBF[i] = accuracy_score(y_train, y_hat_train)
        test_accuracy_RBF[i] = accuracy_score(y_test, y_hat)

        # RBF Random centers and radius
        clf = random_RBF(p)
        clf.fit(X_train, y_train)
        y_hat_train=clf.predict(X_train)
        y_hat_train = (1*(y_hat_train >= 0)-0.5)*2
        y_hat=clf.predict(X_test)
        y_hat = (1*(y_hat >= 0)-0.5)*2
        train_accuracy_RBF2[i] = accuracy_score(y_train, y_hat_train)
        test_accuracy_RBF2[i] = accuracy_score(y_test, y_hat)


    print(f"********* Results RBF-Kmeans (p = {p})**************")
    print("Acc train: " + '{:.4f}'.format(train_accuracy_RBF.mean())+ "+/-" +␣
↪'{:.4f}'.format(train_accuracy_RBF.std()))
    print("Acc test: " + '{:.4f}'.format(test_accuracy_RBF.mean()) + "+/-" + '{:
↪.4f}'.format(test_accuracy_RBF.std()))
    print(f"********* Results RBF-Random (p = {p})**************")
    print("Acc train: " + '{:.4f}'.format(train_accuracy_RBF2.mean())+ "+/-" +␣
↪'{:.4f}'.format(train_accuracy_RBF2.std()))
    print("Acc test: " + '{:.4f}'.format(test_accuracy_RBF2.mean()) + "+/-" +␣
↪'{:.4f}'.format(test_accuracy_RBF2.std()))
```

1.6 Aplicação na base Breast Cancer

[41]:

```
wdbc_dataset = pd.read_csv('data/WDBC/wdbc.data', names=list(range(0,32)))
# convert to array
y = wdbc_dataset[1].to_numpy()
X = wdbc_dataset.drop([0, 1],axis='columns').to_numpy()
y[np.where(y=='B')] = 1
y[np.where(y=='M')] = -1
y = np.array(y.tolist())
for p in [5, 10, 30, 50, 100]:
    results(X, y, 10, p)
```

```
********* Results RBF-Kmeans (p = 5)**************
Acc train: 0.9020+/-0.0051
Acc test: 0.9009+/-0.0275
********* Results RBF-Random (p = 5)**************
Acc train: 0.8952+/-0.0203
Acc test: 0.8895+/-0.0356
********* Results RBF-Kmeans (p = 10)**************
Acc train: 0.9211+/-0.0066
Acc test: 0.9193+/-0.0203
********* Results RBF-Random (p = 10)**************
Acc train: 0.9207+/-0.0095
Acc test: 0.9175+/-0.0163
********* Results RBF-Kmeans (p = 30)**************
Acc train: 0.9284+/-0.0057
Acc test: 0.9289+/-0.0149
********* Results RBF-Random (p = 30)**************
Acc train: 0.9393+/-0.0060
Acc test: 0.9316+/-0.0218
********* Results RBF-Kmeans (p = 50)**************
Acc train: 0.9389+/-0.0056
Acc test: 0.9140+/-0.0199
********* Results RBF-Random (p = 50)**************
Acc train: 0.9525+/-0.0057
Acc test: 0.9298+/-0.0162
********* Results RBF-Kmeans (p = 100)**************
Acc train: 0.9574+/-0.0049
Acc test: 0.9386+/-0.0166
********* Results RBF-Random (p = 100)**************
Acc train: 0.9754+/-0.0031
Acc test: 0.9491+/-0.0195
```

1.7 Aplicação na base Statlog (Heart)

[42]:
```
statlog_dataset = pd.read_csv('data/statlog/heart.dat', sep="\s+",␣
 ↪engine='python', header=None)
X = statlog_dataset.drop((13), 1).to_numpy()
y = statlog_dataset.iloc[:, 13].to_numpy()
y[y==2] = -1
for p in [5, 10, 30, 50, 100]:
    results(X, y, 10, p)
```

```
********* Results RBF-Kmeans (p = 5)**************
Acc train: 0.6870+/-0.0155
Acc test: 0.6389+/-0.0610
********* Results RBF-Random (p = 5)**************
Acc train: 0.6662+/-0.0259
Acc test: 0.6000+/-0.0637
********* Results RBF-Kmeans (p = 10)**************
Acc train: 0.6907+/-0.0170
Acc test: 0.6352+/-0.0933
********* Results RBF-Random (p = 10)**************
Acc train: 0.7204+/-0.0313
Acc test: 0.6611+/-0.0790
********* Results RBF-Kmeans (p = 30)**************
Acc train: 0.7347+/-0.0137
Acc test: 0.6704+/-0.0624
********* Results RBF-Random (p = 30)**************
Acc train: 0.8139+/-0.0153
Acc test: 0.7426+/-0.0418
********* Results RBF-Kmeans (p = 50)**************
Acc train: 0.7579+/-0.0243
Acc test: 0.6519+/-0.0607
********* Results RBF-Random (p = 50)**************
Acc train: 0.8477+/-0.0189
Acc test: 0.7537+/-0.0389
********* Results RBF-Kmeans (p = 100)**************
Acc train: 0.8532+/-0.0083
Acc test: 0.6537+/-0.0310
********* Results RBF-Random (p = 100)**************
Acc train: 0.9194+/-0.0121
Acc test: 0.6889+/-0.0359
```

1.8 Discussão:

Acima é possível perceber que para todos os números de neurônios maiores que 5 a RBF com centros e raios randômicos obteve resultados superiores a rede RBF com centros selecionados com o algoritmo K-means.

Contudo. ao compararmos os resultados da rede RBF com os da ELM (da lista 6), nota-se que a ELM foi superior em todos os teste.

2 Referências

[1] D. Dua and C. Graff, “UCI machine learning repository,” 2017. [Online]. Available: http://archive.ics.uci.edu/ml