exercicio8

February 13, 2021

1 Redes Neurais Artificiais

1.1 Exercíco 8 - Treinamento RBF/ELM

Aluno: Leonam Rezende Soares de Miranda

O objetivo dete exercício é combinar os conceitos aprendidos na Unidade 2 e construir uma rede neural que some elementos das redes RBF e das redes ELM. As basdes de dados estudadas serão:

- Breast Cancer (diagnostic)
- Stalog (Heart)

Será construído uma **rede RBF com centros e raios atribuídos de forma aleatória aps neurônios**. Serão escolhidos os centros entre 2 pontos escolhidos de forma aleatória do conjunto de dados. com raio da função igual à distância entre os pontos.

Além da RBF com centros e raios aleatórios, deve ser construída uma RBF com centros e raios selecionados a partir do k-médias. As acurácias obtidas por cada uma das redes nas duas bases devem ser apresentadas no formato ± e comparadas com os resultados obtidos no exercício 6 para ELMs. O desempenho do modelo deve ser comparado com o desempenho de RBF com centros selecionados por agrupamento (k-medias).

No código abaixo se encontra a classe do classificador RBF que possui funções como fit que treina o modelo, e score que retorna a acurácia do modelo num conjunto passado por parâmetro.

[256]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.base import BaseEstimator
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, plot_roc_curve, roc_curve,␣
↪auc
from sklearn.metrics import plot_confusion_matrix, confusion_matrix
from sklearn.cluster import KMeans
import random

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

```
   def __init__(self, p, classification = True, kmeans=True):
       """ Inicializa a classe

       Parâmetros
       ----------

       p: número de neurônios da camada intermediária

       classification: True para classificação, False para regressão

       kmeans: True para centros calculados por agrupamento (k-medias), False␣
↪para centros aleatórios
       """
       self.W = None
       self.p = p
       self.m = None
       self.covlist = []
       self.n = None
       self.classification = classification
       self.kmeans = kmeans

   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

       """

       K = K + 0.001 * np.identity(X.shape[1])
```

```
        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):
                H[j,i+1] = self.pdfnvar(X[j,:], self.m[i,:], self.covlist[i])
        return H

    def fit(self, X: np.ndarray, y: np.ndarray):
        """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.

            Retorna
            -------
            self: objeto
```

```
            Estimador ajustado.
        """

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

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

        # se centros devem ser selcionados por agrupamento (k-médias)
        if self.kmeans:
            # 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)
        # se centros devem ser selecionados de forma aleatória
        else:
            indicesPontos = set()
            self.m = np.zeros((self.p,X.shape[1]))
            i = 0
            while i < self.p:
                # seleciona 2 pontos de forma aleatória do conjunto de␣
↪treinamento
                indices = tuple(random.sample(range(0, X.shape[0]), 2))
                if indices not in indicesPontos:
                    indicesPontos.add(indices)
                    # Centro é a média dos  pontos
                    self.m[i,:] = (X[indices[0],:] + X[indices[1],:])/2

                    # Raio é a distância entre os dois pontos
                    raio = np.linalg.norm(X[indices[0],:] - X[indices[1],:])

                    # matriz de covariância é dada pelo raio ao quadrado␣
↪multiplicado pela identidadeconsiderando independência entre as variáveis
                    self.covlist.append((raio**2) * np.identity(X.shape[1]))
                else:
                    i -= 1
                i += 1
```

```
        # 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
```

1.1.1 Breast Cancer

A primeira base de dados a ser estudada é a base Breast Cancer (diagnostic). O código a seguir faz a importação e tratamento dos dados.

[293]:
```
# Carregar a base de dados, remover dados faltantes e normalizar os dados
data = load_breast_cancer()

X = data.data
Y = data.target

labels = data.target_names

count = 0
# Loop para verificar se há dados nulos/fatantes no vetor de rótulos Y
for i, j in zip(pd.DataFrame(Y).isnull().sum(), pd.DataFrame(X).isnull().sum()):
    if i!=0 or j!=0:
        count+=1

if count == 0:
    print('Não há dados faltantes!')


# Standardization - Todos os atributos agora terão média zero e variação␣
 ↪unitária
scaler = StandardScaler()
X = scaler.fit_transform(X)

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

```
Y[Y==0] = -1

# Separa os dados de forma aleatória - 70% para treinamento e 30% para testes
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3)
```

```
Não há dados faltantes!
```

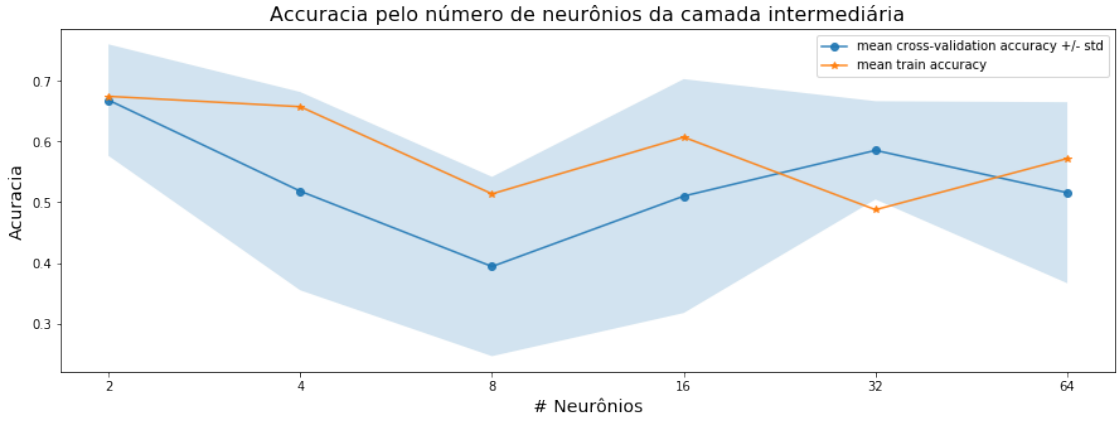
O código a seguir plota a performance estimada com +/- 1 desvio padrão para diferentes quantidades de neurônios na camada intermediária, utilizando validação cruzada com 10 folds.

[258]:
```
def run_cross_validation_on_rbfs(X, y, Nneuronios, kmeans, cv=10):
    """ Método para ajustar RBFs com diferentes números de neurônios na camada␣
↪oculta
        sobre os dados de treinamento usando validação cruzada

    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) do conjunto de treinamento.

    Nneuronios: Lista com quantidade de neurônios (p) que serão avaliados

    kmeans: True para centros calculados por agrupamento (k-medias), False para␣
↪centros aleatórios

    cv: Número de folds que serão utilizados na validação cruzada

    """
    # Listas auxiliares que serão retornadas
    cv_scores_list = []
    cv_scores_std = []
    cv_scores_mean = []
    mean_accuracy_scores_train = []

    for p in Nneuronios:
        RBF_model = RBF(p=p, kmeans=kmeans)
        cv_scores = cross_val_score(RBF_model, X, y, cv=cv, scoring='accuracy')
        cv_scores_mean.append(cv_scores.mean())
        cv_scores_std.append(cv_scores.std())

        accuracy_scores_train = []
        for i in range(cv):
            model = RBF_model.fit(X, y)
            accuracy_scores_train.append(model.score(X, y))
        mean_accuracy_scores_train.append(np.mean(accuracy_scores_train))
```

```
    cv_scores_mean = np.array(cv_scores_mean)
    cv_scores_std = np.array(cv_scores_std)
    mean_accuracy_scores_train = np.array(mean_accuracy_scores_train)
    return cv_scores_mean, cv_scores_std, mean_accuracy_scores_train

# function for plotting cross-validation results
def plot_cross_validation_results(Nneuronios, cv_scores_mean, cv_scores_std,␣
 ↪mean_scores_train, title):
    fig, ax = plt.subplots(1,1, figsize=(15,5))
    ax.plot(list(map(str, Nneuronios)), cv_scores_mean, '-o', label='mean␣
 ↪cross-validation accuracy +/- std', alpha=0.9)
    ax.fill_between(list(map(str, Nneuronios)), cv_scores_mean-cv_scores_std,␣
 ↪cv_scores_mean+cv_scores_std, alpha=0.2)
    ylim = plt.ylim()
    ax.plot(list(map(str, Nneuronios)), mean_scores_train, '-*', label='mean␣
 ↪train accuracy', alpha=0.9)
    ax.set_title(title, fontsize=16)
    ax.set_xlabel('# Neurônios', fontsize=14)
    ax.set_ylabel('Acuracia', fontsize=14)
    ax.set_ylim(ylim)
    ax.set_xticks(list(map(str, Nneuronios)))
    ax.legend()
```

RBF com centros e raios selecionados a partir do k-médias Foi executada validação cruzada com 10 folds para encontrar o modelo que apresente maior generalização em função da quantidade de neurônios da camada intermediária.

[273]:
```
# lista com quantidade de neurônios (p) que serão avaliados
Nneuronios = [2, 4, 8, 16, 32, 64]

cv_scores_mean, cv_scores_std, mean_scores_train=␣
 ↪run_cross_validation_on_rbfs(X_train,
                                                          Y_train,␣
 ↪Nneuronios, kmeans=True)

# plotting accuracy
plot_cross_validation_results(Nneuronios, cv_scores_mean, cv_scores_std,␣
 ↪mean_scores_train,
                              'Accuracia pelo número de neurônios da camada␣
 ↪intermediária')
```



[275]:
```
idx_max = cv_scores_mean.argmax()
best_Nneuronios = Nneuronios[idx_max]
best_score = cv_scores_mean[idx_max]
best_score_std = cv_scores_std[idx_max]
print('Ao utilizar {} neurônios na camada intermediária foi atingindo a melhor␣
 ↪acurácia média de validação cruzada sobre o conjunto de treinamento: {} +/-␣
 ↪{}%'.format(best_Nneuronios, round(best_score*100,5),␣
 ↪round(best_score_std*100, 5)))
```

```
Ao utilizar 2 neurônios na camada intermediária foi atingindo a melhor acurácia
média de validação cruzada sobre o conjunto de treinamento: 64.57692 +/-
7.96608%
```

Ao utilizar mais **2** neurônios na camada intermediária a acurácia média e a acurácia média de validação cruzada diminuem. No código a seguir o modelo **RBF com centros calculado a partir do algoritmo k-médias** com **2** neurônios na camada intermediária, foi treinado 10 vezes obtendo assim a acurácia de treinamento e teste média +/- um desvio padrão.

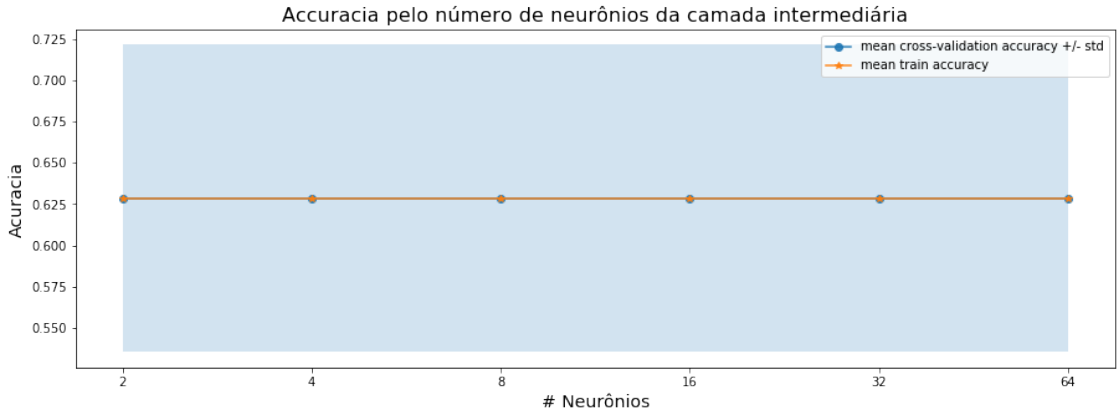
[281]:
```
# function for training and evaluating the ELM model
def run_single_classificator(X_train, y_train, X_test, y_test, p = 10,␣
 ↪kmeans=True, execucoes=1):

    accuracy_train = []
    accuracy_test = []
    for i in range(execucoes):
        model = RBF(p=p, kmeans=kmeans).fit(X_train, y_train)
        accuracy_train.append(model.score(X_train, y_train))
        accuracy_test.append(model.score(X_test, y_test))

    print('Accuracia Média - Conjunto de Treinamento: ' + str(round(np.
 ↪mean(accuracy_train)*100,5)) +'% +/-' + str(round(np.
 ↪std(accuracy_train)*100,5)) + '%')
```

```
    print('Accuracia Média - Conjunto de Teste: ' + str(round(np.
 ↪mean(accuracy_test)*100,5)) +'% +/-' + str(round(np.
 ↪std(accuracy_test)*100,5)) +'%')

    return

# train and evaluate an RBF model
run_single_classificator(X_train, Y_train, X_test, Y_test, best_Nneuronios,␣
 ↪kmeans=True, execucoes=10)
```

```
Accuracia Média - Conjunto de Treinamento: 68.59296% +/-0.0%
Accuracia Média - Conjunto de Teste: 60.81871% +/-0.0%
```

RBF com centros e raios atribuidos de forma aleatória Foi executada validação cruzada com 10 folds para encontrar o modelo que apresente maior generalização em função da quantidade de neurônios da camada intermediária.

[294]:
```
# lista com quantidade de neurônios (p) que serão avaliados
Nneuronios = [2, 4, 8, 16, 32, 64]


cv_scores_mean, cv_scores_std, mean_scores_train=␣
 ↪run_cross_validation_on_rbfs(X_train,
                                                        Y_train,␣
 ↪Nneuronios, kmeans=False)

# plotting accuracy
plot_cross_validation_results(Nneuronios, cv_scores_mean, cv_scores_std,␣
 ↪mean_scores_train,
                               'Accuracia pelo número de neurônios da camada␣
 ↪intermediária')
```



Não houve variação significativa ao variar o número de neurônios. Assim foi escolhido o modelo mais

simples com **2** neurônios. No código a seguir o modelo **RBF com centros e raios atribuidos de forma aleatória** com **2** neurônios na camada intermediária, foi treinado 10 vezes obtendo assim a acurácia de treinamento e teste média +/- um desvio padrão.

[282]:
```
best_Nneuronios = 2
# train and evaluate an RBF model
run_single_classificator(X_train, Y_train, X_test, Y_test, best_Nneuronios,␣
 ↪kmeans=False, execucoes=10)
```

```
Accuracia Média - Conjunto de Treinamento: 64.57286% +/-0.0%
Accuracia Média - Conjunto de Teste: 58.47953% +/-0.0%
```

No ELM do exercício 6 foi obtida acurácia de trienamento de 97% e de teste de 96%. Assim, para a Base de dados **Breast Cancer**, o algoritmo ELM obteve os melhores resultados.

1.1.2 Statlog (Heart)

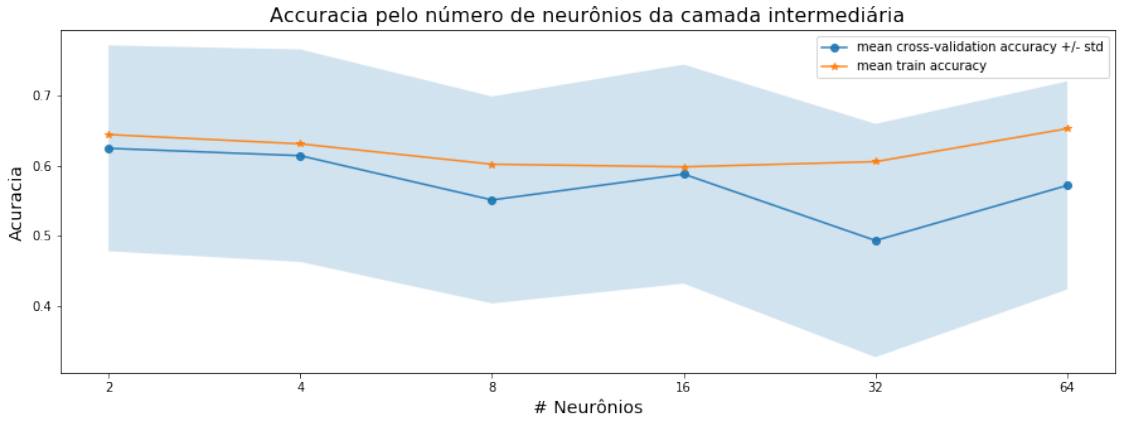
A segunda base de dados a ser estudada é a base Stalog (Heart). O código a seguir faz a importação e tratamento dos dados.

[283]:
```
data = []

with open('heart.dat','r') as token:
    for line in token:
        data.append(line.split())

data = np.asarray(data)

Y = np.squeeze(data[:,-1].astype(np.int))
X = data[:,0:-1]

# Standardization - Todos os atributos agora terão média zero e variação␣
 ↪unitária
scaler = StandardScaler()
X = scaler.fit_transform(X)

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

# Separa os dados de forma aleatória - 70% para treinamento e 30% para testes
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3)
```

RBF com centros e raios selecionados a partir do k-médias Foi executada validação cruzada com 10 folds para encontrar o modelo que apresente maior generalização em função da quantidade de neurônios da camada intermediária.

[286]:
```
# lista com quantidade de neurônios (p) que serão avaliados
Nneuronios = [2, 4, 8, 16, 32, 64]
```

```
cv_scores_mean, cv_scores_std, mean_scores_train=␣
↪run_cross_validation_on_rbfs(X_train,
                                                    Y_train,␣
↪Nneuronios, kmeans=True)

# plotting accuracy
plot_cross_validation_results(Nneuronios, cv_scores_mean, cv_scores_std,␣
↪mean_scores_train,
                              'Accuracia pelo número de neurônios da camada␣
↪intermediária')
```



[287]:
```
idx_max = cv_scores_mean.argmax()
best_Nneuronios = Nneuronios[idx_max]
best_score = cv_scores_mean[idx_max]
best_score_std = cv_scores_std[idx_max]
print('Ao utilizar {} neurônios na camada intermediária foi atingindo a melhor␣
↪acurácia média de validação cruzada sobre o conjunto de treinamento: {} +/-␣
↪{}%'.format(best_Nneuronios, round(best_score*100,5),␣
↪round(best_score_std*100, 5)))
```

```
Ao utilizar 2 neurônios na camada intermediária foi atingindo a melhor acurácia
média de validação cruzada sobre o conjunto de treinamento: 62.48538 +/-
14.6806%
```

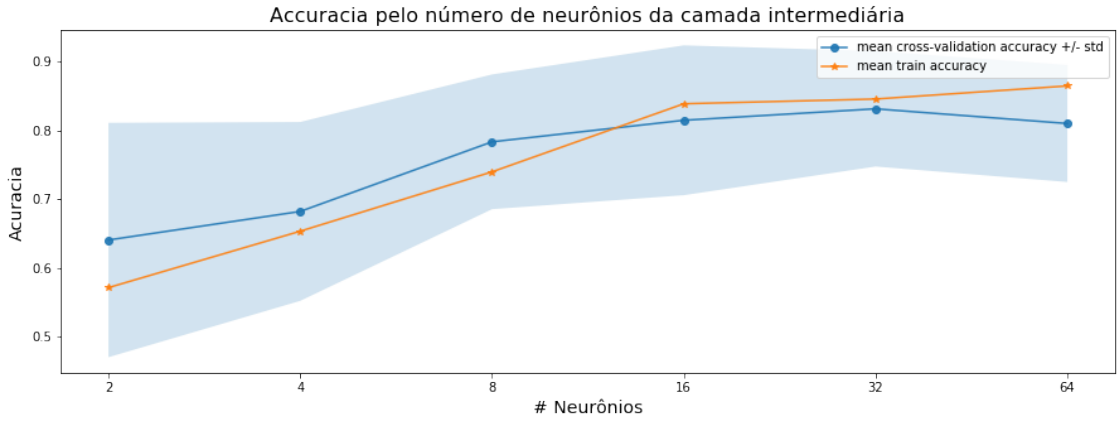
Assim com na base de dados ***Breast Cancer*** Ao utilizar mais **2** neurônios na camada intermediária a acurácia média e a acurácia média de validação cruzada diminuem. No código a seguir o modelo **RBF com centros calculado a partir do algoritmo k-médias** com **2** neurônios na camada intermediária, foi treinado 10 vezes obtendo assim a acurácia de treinamento e teste média +/- um desvio padrão.

[288]:
```
# train and evaluate an RBF model
run_single_classificator(X_train, Y_train, X_test, Y_test, best_Nneuronios,␣
 ↪kmeans=True, execucoes=10)
```

```
Accuracia Média - Conjunto de Treinamento: 64.70899% +/-0.47619%
Accuracia Média - Conjunto de Teste: 58.2716% +/-0.74074%
```

RBF com centros e raios atribuídos de forma aleatória Foi executada validação cruzada com 10 folds para encontrar o modelo que apresente maior generalização em função da quantidade de neurônios da camada intermediária.

[289]:
```
# lista com quantidade de neurônios (p) que serão avaliados
Nneuronios = [2, 4, 8, 16, 32, 64]


cv_scores_mean, cv_scores_std, mean_scores_train=␣
 ↪run_cross_validation_on_rbfs(X_train,
                                                    Y_train,␣
 ↪Nneuronios, kmeans=False)

# plotting accuracy
plot_cross_validation_results(Nneuronios, cv_scores_mean, cv_scores_std,␣
 ↪mean_scores_train,
                               'Accuracia pelo número de neurônios da camada␣
 ↪intermediária')
```



[290]:
```
idx_max = cv_scores_mean.argmax()
best_Nneuronios = Nneuronios[idx_max]
best_score = cv_scores_mean[idx_max]
best_score_std = cv_scores_std[idx_max]
```

```
print('Ao utilizar {} neurônios na camada intermediária foi atingindo a melhor␣
 ↪acurácia média de validação cruzada sobre o conjunto de treinamento: {} +/-␣
 ↪{}%'.format(best_Nneuronios, round(best_score*100,5),␣
 ↪round(best_score_std*100, 5)))
```

```
Ao utilizar 32 neurônios na camada intermediária foi atingindo a melhor acurácia
média de validação cruzada sobre o conjunto de treinamento: 83.12865 +/-
8.38121%
```

Ao utilizar mais **32** neurônios na camada intermediária a acurácia média e a acurácia média de validação cruzada diminui, demostrando sobreajuste. No código a seguir o modelo **RBF com centros calculado a partir do algoritmo k-médias** com **32** neurônios na camada intermediária, foi treinado 10 vezes obtendo assim a acurácia de treinamento e teste média +/- um desvio padrão.

[291]:
```
# train and evaluate an RBF model
run_single_classificator(X_train, Y_train, X_test, Y_test, best_Nneuronios,␣
 ↪kmeans=False, execucoes=10)
```

```
Accuracia Média - Conjunto de Treinamento: 85.92593% +/-0.95238%
Accuracia Média - Conjunto de Teste: 81.11111% +/-1.99451%
```

No ELM do exercício 6 foi obtida acurácia de trienamento de 86.77% e de teste de 86.41%. Assim, para a Base de dados **Stalog (heart)**, o algoritmo ELM também obteve os melhores resultados.