# Exercício 10

March 13, 2021

**Aluno**: Leonam Rezende Soares de Miranda

O objetivo dos exercícios desta semana é utilizar redes MLP para resolver problemas multidimensionais, a partir de bases de dados reais. Assim serão aplicadas MLP em duas bases de dados: *Boston Housing* e *Statlog (Heart)*. Neste exercício serão empregados pacotes de treinamento de redes neurais fornecidos pelo *Scikit-Learn*.

## 0.1 Boston Housing

O código a seguir faz a importação dos pacotes que serão utilizados e da base de dados

[99]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor, MLPClassifier
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score,␣
↪roc_auc_score

boston = load_boston()
print(boston.data.shape)
```

```
(506, 13)
```

A seguir, será feito o pré-processamento dos dados.

[118]:
```
X = boston.data
Y = boston.target

# Divide os atributos em conjunto de treinamento e de testes na razaõ 75%/ 25%
X_train, X_test, y_train, y_test = train_test_split(X, Y)

# Aplica a mesma escala para ambos os datasets
scaler = StandardScaler()
X_train_scl = scaler.fit_transform(X_train)
X_test_scl = scaler.transform(X_test) # note that we transform rather than␣
↪fit_transform
```

No codigo abaixo será feito uma busca em grid, avaliando o MSE médio de validação cruzada com 10 folds, para encontrar a melhor arquitetura de rede neural (variando o número de neurônios e funções de ativação). As funções de ativação da camada intermediária que serão avaliadas estão definidas a seguir:

- ***identity***, no-op activation, useful to implement linear bottleneck, returns f(x) = x
- ***logistic***, the logistic sigmoid function, returns f(x) = 1 / (1 + exp(-x)).
- ***tanh***, the hyperbolic tan function, returns f(x) = tanh(x).
- ***relu***, the rectified linear unit function, returns f(x) = max(0, x)

Serão avaliados modelos com número de neurônios na camada intermediária iguais a $2\ \forall i \in [1, 11]$.

[103]:
```
NNeuronios = [(2,), (4,), (8,), (16,), (32,), (64,), (128,), (256,), (512,),␣
↪(1024,), (2048,)]
activations = ['identity', 'logistic', 'tanh', 'relu']

parameters = {'hidden_layer_sizes':NNeuronios, 'activation':activations}
model = MLPRegressor(tol=0.5e-2)

clf = GridSearchCV(model, parameters, scoring='neg_mean_squared_error', n_jobs␣
↪= -1, cv = 10)
clf.fit(X_train_scl, y_train)
```

```
C:\Users\Leonam\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:582:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

[103]:
```
GridSearchCV(cv=10, estimator=MLPRegressor(tol=0.005), n_jobs=-1,
             param_grid={'activation': ['identity', 'logistic', 'tanh', 'relu'],
                         'hidden_layer_sizes': [(2,), (4,), (8,), (16,), (32,),
                                                (64,), (128,), (256,), (512,),
                                                (1024,), (2048,)]},
             scoring='neg_mean_squared_error')
```

[104]:
```
idx = np.where(clf.cv_results_['mean_test_score'] == np.max(clf.
↪cv_results_['mean_test_score']))[0][0]
best_MSE = clf.cv_results_['mean_test_score'][idx]
best_MSE_std = clf.cv_results_['std_test_score'][idx]

best_Nneuronios = clf.best_params_['hidden_layer_sizes']
best_activation = clf.best_params_['activation']
```

```
print('Ao utilizar {} neurônios na camada intermediária e função de ativação␣
↪\'{}\' foi encontrado o menor MSE médio de validação cruzada sobre o␣
↪conjunto de treinamento: {} +/- {}%'.format(best_Nneuronios,␣
↪best_activation, round(-1*best_MSE,3), round(best_MSE_std, 3)))
```

```
Ao utilizar (2048,) neurônios na camada intermediária e função de ativação
'relu' foi encontrado o menor MSE médio de validação cruzada sobre o conjunto de
treinamento: 14.063 +/- 7.63%
```

Após encontrar os parâmetros do melhor modelo, o mesmo foi validado sobre o conjunto de testes. Serão feitas 10 execuções diferentes e ao final será apresentado o $MSE\_mdio \pm desvio\_padrao$ e o $R2\_score\_medio \pm desvio\_padrao$.

[105]:
```
execucoes = 10

MSE_array = np.zeros(execucoes)
R2scores_array = np.zeros(execucoes)

for i in range(execucoes):
    regression_model = MLPRegressor(hidden_layer_sizes=best_Nneuronios,␣
↪activation=best_activation, max_iter=2000)
    regression_model.fit(X_train_scl, y_train)
    MSE_array[i] = mean_squared_error(y_test, regression_model.
↪predict(X_test_scl))
    R2scores_array[i] = r2_score(y_test, regression_model.predict(X_test_scl))

print('MSE médio e desvio padrão sobre o conjunto de testes após dez tentativas␣
↪{} +/- {}'.format(round(np.mean(MSE_array),3), round(np.std(MSE_array),3)))
print('R2 score médio e desvio padrão sobre o conjunto de testes após dez␣
↪tentativas {} +/- {}'.format(round(np.mean(R2scores_array),3), round(np.
↪std(R2scores_array),3)))
```

```
MSE médio e desvio padrão sobre o conjunto de testes após dez tentativas 9.158
+/- 0.504
R2 score médio e desvio padrão sobre o conjunto de testes após dez tentativas
0.884 +/- 0.006
```

## 0.2 Statlog (Heart)

A mesma análise que foi feita na base de dados *Boston Housing* será feita a seguir na base de dados *Statlog (Heart)*.

No código a seguir foi feita a importação e o pré-processamento dos dados.

[124]:
```
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]

# 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)

# Aplica a mesma escala para ambos os datasets
scaler = StandardScaler()
X_train_scl = scaler.fit_transform(X_train)
X_test_scl = scaler.transform(X_test) # note that we transform rather than␣
↪fit_transform
```

No codigo abaixo será feito uma busca em grid, avaliando o AUC médio de validação cruzada com 10 folds, para encontrar a melhor arquitetura de rede neural (variando o número de neurônios e funções de ativação).

[125]:
```
NNeuronios = [(2,), (4,), (8,), (16,), (32,), (64,), (128,), (256,), (512,),␣
↪(1024,), (2048,)]
activations = ['identity', 'logistic', 'tanh', 'relu']

parameters = {'hidden_layer_sizes':NNeuronios, 'activation':activations}
model = MLPClassifier()

clf = GridSearchCV(model, parameters, scoring='roc_auc', n_jobs = -1, cv = 10)
clf.fit(X_train_scl, Y_train)
```

```
C:\Users\Leonam\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:582:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

[125]:
```
GridSearchCV(cv=10, estimator=MLPClassifier(), n_jobs=-1,
             param_grid={'activation': ['identity', 'logistic', 'tanh', 'relu'],
                         'hidden_layer_sizes': [(2,), (4,), (8,), (16,), (32,),
                                                (64,), (128,), (256,), (512,),
                                                (1024,), (2048,)]},
             scoring='roc_auc')
```

[127]:
```
idx = np.where(clf.cv_results_['mean_test_score'] == np.max(clf.
↪cv_results_['mean_test_score']))[0][0]
best_AUC = clf.cv_results_['mean_test_score'][idx]
best_AUC_std = clf.cv_results_['std_test_score'][idx]
```

```
best_Nneuronios = clf.best_params_['hidden_layer_sizes']
best_activation = clf.best_params_['activation']

print('Ao utilizar {} neurônios na camada intermediária e função de ativação␣
 ↪\'{}\' foi encontrado o maior AUC médio de validação cruzada sobre o␣
 ↪conjunto de treinamento: {} +/- {}%'.format(best_Nneuronios,␣
 ↪best_activation, round(best_AUC,3), round(best_AUC_std, 3)))
```

```
Ao utilizar (32,) neurônios na camada intermediária e função de ativação
'logistic' foi encontrado o maior AUC médio de validação cruzada sobre o
conjunto de treinamento: 0.933 +/- 0.082%
```

Após encontrar os parâmetros do melhor modelo, o mesmo foi validado sobre o conjunto de testes. Serão feitas 10 execuções diferentes e ao final será apresentado o $AUC\_mdio \pm desvio\_padrao$ e a $Acurcia\_media \pm desvio\_padrao$.

[128]:
```
execucoes = 10

AUC_array = np.zeros(execucoes)
Accuracy_array = np.zeros(execucoes)

for i in range(execucoes):
    classifier_model = MLPClassifier(hidden_layer_sizes=best_Nneuronios,␣
 ↪activation=best_activation, max_iter=2000)
    classifier_model.fit(X_train_scl, Y_train)
    AUC_array[i] = roc_auc_score(Y_test, classifier_model.predict(X_test_scl))
    Accuracy_array[i] = accuracy_score(Y_test, classifier_model.
 ↪predict(X_test_scl))

print('AUC médio e desvio padrão sobre o conjunto de testes após dez tentativas␣
 ↪{} +/- {}'.format(round(np.mean(AUC_array),3), round(np.std(AUC_array),3)))
print('Acurácia média e desvio padrão sobre o conjunto de testes após dez␣
 ↪tentativas {} +/- {}'.format(round(np.mean(Accuracy_array),3), round(np.
 ↪std(Accuracy_array),3)))
```

```
AUC médio e desvio padrão sobre o conjunto de testes após dez tentativas 0.789
+/- 0.01
Acurácia média e desvio padrão sobre o conjunto de testes após dez tentativas
0.796 +/- 0.01
```

A partir desse exercício pode-se observar a importância na seleção de hiperparâmetros nos modelos, pois isso impacta bastante em seus desempenhos, e como a escolha dos melhores hiperparâmetros varia bastante em função da base de dados. Entretanto, deve-se levar em consideração que a busca em grid realizando validação cruzada para seleção de modelos é um processo muito custoso, dependedo da quantidade de hiperparâmetros a serem avaliados.

Nesse trabalho foram avaliadas várias funções de ativação. As funções de ativação “tanh” e “logistic” (sigmoid) possuem derivadas próximas de zero quando as suas entradas são muito pequenas ou

muito grandes, e isso pode tornar a convergência do gradiente descendente muito lenta. Por isso, para camadas intermediárias de redes neurais de multiplas camadas, o mais comum na literatura é usar a função de ativação $ReLU$ (rectify linear unit), ou a sua variante *leaky* $ReLU$, pois assim a rede neural tendenrá a treinar mais ráído. As funções de ativação "tanh" e "logistic" (sigmoid) são mais comumente utilizadas na camada de saída para realizar classificação binária.