Trabalho Final - Inteligência Computacional Aplicada (TIP7077)

Aluno: Carlos Eduardo Sousa Lima

Prof. Guilherme de Alencar Barreto

Questão 02 - Regressão e Ajuste de Curvas - *Real estate valuation dataset*

The data set was randomly split into the training data set (2/3 samples) and the testing data set (1/3 samples).
(https://archive.ics.uci.edu/dataset/477/real+estate+valuation+data+set)

In [ ]:
```
#Bibliotecas Utilizadas
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Regressão Linear Múltipla de Mínimos Quadrados

In [ ]:
```
def R2(y_true, y_pred):

    coef = 1 - (np.power(y_true-y_pred,2).sum()/np.power(y_true-y_true.mean(),2).sum())

    return coef

def RMSE(y_true, y_pred):

    coef = np.power((np.power(y_true - y_pred, 2).sum())/y_true.shape[0], 0.5)

    return coef

data = pd.read_excel("Real estate valuation data set.xlsx", index_col = 0)

X = data.iloc[:, 0:6].to_numpy()

Y = data.iloc[:, -1].to_numpy()
```

```
RMSE_r, tx_ok_20, tx_ok_10, R2_r = [], [], [], []

best_dict = {
    "tx_ok_10":0,
    "tx_ok_20":0,
    "y_pred": 0,
    "y_true": 0,
    "RMSE": 0,
    "R2": 0,
    "R2_aj": 0
}

Nr = 20
for i in range(Nr):
    rand_index = np.random.permutation(X.shape[0])
    X = X[rand_index,:]
    Y = Y[rand_index]

    spt_point = int(2/3*X.shape[0])

    X_train = X[:spt_point,:]
    Y_train = Y[:spt_point]

    X_test = X[spt_point:,:]
    Y_test = Y[spt_point:]

    X_train = np.c_[np.ones(X_train.shape[0]), X_train]
    X_test = np.c_[np.ones(X_test.shape[0]), X_test]
    M = M = np.linalg.lstsq(X_train, Y_train, rcond = -1)[0]

    Y_pred = np.dot(X_test, M)

    count_10 = 0
    count_20 = 0
    for i in range(Y_test.shape[0]):

        if np.abs(Y_test[i] - Y_pred[i]) <= 0.2*np.abs(Y_test[i]):
            count_20 += 1
        if np.abs(Y_test[i] - Y_pred[i]) <= 0.1*np.abs(Y_test[i]):
            count_10 += 1
```

```
    RMSE_r.append(RMSE(Y_test, Y_pred))
    R2_r.append(R2(Y_test, Y_pred))
    tx_ok_10.append(count_10/Y_test.shape[0])
    tx_ok_20.append(count_20/Y_test.shape[0])

    #Para ser a melhor rodada, deve ser melhor nas duas taxas de acerto
    if (best_dict["tx_ok_10"] <= count_10/X_test.shape[0]) & (best_dict["tx_ok_20"] <= count_20/X_test.shape[0]):
        best_dict["y_pred"] = Y_pred
        best_dict["y_true"] = Y_test
        best_dict["R2"] = R2(Y_test, Y_pred)
        best_dict["RMSE"] = RMSE(Y_test, Y_pred)
        best_dict["tx_ok_10"] = count_10/X_test.shape[0]
        best_dict["tx_ok_20"] = count_20/X_test.shape[0]

print("###### Resumo das {} Rodadas ######".format(Nr))
print('''Taxa de acerto para uma taxa de erro admssível de 20%:
Média = {:.2%},
Desv. Pad. = {:.2%},
Máximo = {:.2%},
Mínimo = {:.2%}\n'''.format(np.mean(tx_ok_20),
    np.std(tx_ok_20), np.max(tx_ok_20), np.min(tx_ok_20)))

print('''Taxa de acerto para uma taxa de erro admssível de 10%:
Média = {:.2%},
Desv. Pad. = {:.2%},
Máximo = {:.2%},
Mínimo = {:.2%}\n'''.format(np.mean(tx_ok_10),
    np.std(tx_ok_10), np.max(tx_ok_10), np.min(tx_ok_10)))

print('''R2:
Média = {:.3f},
Desv. Pad. = {:.3f},
Máximo = {:.3f},
Mínimo = {:.3f}\n'''.format(np.mean(R2_r),
    np.std(R2_r), np.max(R2_r), np.min(R2_r)))

print('''RMSE:
Média = {:.3f},
Desv. Pad. = {:.3f},
Máximo = {:.3f},
Mínimo = {:.3f}'''.format(np.mean(RMSE_r),
    np.std(RMSE_r), np.max(RMSE_r), np.min(RMSE_r)))
```

```
print("\n###### Melhor Rodada em relação as taxas de acerto para erros admissíveis de 10% e 20% ######")
print("Taxa de acerto para uma taxa de erro admssível de 20% = {:.2%}".format(best_dict["tx_ok_20"]))
print("Taxa de acerto para uma taxa de erro admssível de 10% = {:.2%}".format(best_dict["tx_ok_10"]))
print("R2 = {:.3f}".format(best_dict["R2"]))
print("RMSE = {:.3f}".format(best_dict["RMSE"]))
```

```
###### Resumo das 20 Rodadas ######
Taxa de acerto para uma taxa de erro admssível de 20%:
Média = 68.62%,
Desv. Pad. = 4.44%,
Máximo = 78.99%,
Mínimo = 60.87%

Taxa de acerto para uma taxa de erro admssível de 10%:
Média = 39.02%,
Desv. Pad. = 3.31%,
Máximo = 46.38%,
Mínimo = 35.51%

R2:
Média = 0.561,
Desv. Pad. = 0.067,
Máximo = 0.663,
Mínimo = 0.428

RMSE:
Média = 9.044,
Desv. Pad. = 1.110,
Máximo = 11.068,
Mínimo = 7.447

###### Melhor Rodada em relação as taxas de acerto para erros admissíveis de 10% e 20% ######
Taxa de acerto para uma taxa de erro admssível de 20% = 78.99%
Taxa de acerto para uma taxa de erro admssível de 10% = 42.75%
R2 = 0.630
RMSE = 8.032
```

Yeh e Hsu (2018) apresentam o resultado da análise de regressão multivariada na Tabela 6 de seu artigo. Os resultados são apresentados de acordo com os círculos de abastecimento e demanda apresentados no Trabalho, são eles: Sindia, Danshuei, Wunshan e Beitou. Os resultados também são apresentados em termos da média obtida para cada um desses círculos de abastecimento e demanda.

A base de dados disponibilizada refere-se as informações da círculo de abastecimento e demanda do Sindia, portanto os resultados obtidos nesse trabalho serão comparados aos resultados obtidos para essa região. Os autores destacam que a base de dados total foi divida aleatoriamente em 2/3 para treinamento e 1/3 para teste. As métricas apresentadas pelo autores correspondem a avaliação das predições para os dados de teste.

Tendo em vista essa divisão aleatória realizada pelos autores, decidiu-se rodar o modelo de regressão linear múltipla 20 vezes, gerando uma estatística para essas 4 métricas. Para cada uma dessas rodadas, embaralhou-se a base de dados e realizou-se a separação entre dados de treino e dados de teste, segundo as proporções especificadas pelos autores.

Foram comparadas 4 métricas: i) Taxa de acerto para um erro admissível de 20%, ii) Taxa de acerto para um erro admissível de 10%, iii) R^2 e iv) RMSE. A tabela abaixo apresenta as métricas para a rodada da regressão linear múltipla com melhor taxa de acerto para um erro tolerável de 10% e 20% e os valores obtidos por Yeh e Hsu (2018).

Métrica	Regressão Linear Múltipla	Yeh e Hsu (2018)
i)	78,99%	78,00%
ii)	42,75%	48,80%
iii)	0,630	0,627
iv)	8,032	8,06

Observa-se que o modelo implementado se aproximou dos resultados obtidos por Yeh e Hsu (2018). Exceto pela métrica ii), a a diferença no desempenho do modelo de Regressão Linear Múltipla e os resultados apresentado pelos referidos autores foram insignificantes. O desempenho do modelo implemetando para a métrica ii), por sua vez, ficou abaixo dos obtidos por Yeh e Hsu (2018) em cerca de 6%.

Destaca-se que o desempenho superior ou inferior do modelo implementado em relação aos resultados de Yeh e Hsu (2018) é bastante influenciado pela divisão aleatória da base de dados, uma vez que haverá uma possível combinação de dados de treino e de teste que otimize ou reduza essas métricas.

Extreme Learning Machine

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
```

```
import pandas as pd
import time

def act_fun(u, fun):

    if fun == "step":
        u[np.where(u >= 0)] = 1
        u[np.where(u < 0)] = 0

    elif fun == "tanh":
        u = np.array(list(map(lambda x: (1-np.exp(-x))/(1+np.exp(-x)), u)))

    elif fun == "log":
        u = np.array(list(map(lambda x: 1/(1+np.exp(-x)), u)))

    return u

def norm_data(x):

    for i in range(x.shape[1]):
        x[:,i] = x[:,i]/x[:,i].max()

    return x

def R2(y_true, y_pred):

    coef = 1 - (np.power(y_true-y_pred,2).sum()/np.power(y_true-y_true.mean(),2).sum())

    return coef

def RMSE_calc(y_true, y_pred):

    coef = np.power((np.power(y_true - y_pred, 2).sum())/y_true.shape[0], 0.5)

    return coef

data = pd.read_excel("Real estate valuation data set.xlsx", index_col = 0)

X = data.iloc[:, 0:6].to_numpy()

Y = data.iloc[:, -1].to_numpy()
```

```
X = norm_data(X)
Y = norm_data(np.expand_dims(Y,1))

fun_type = "log"
Nr = 10
q = 5
RMSE_r = []
R2_r = []
R2aj_r = []
tx_ok_20 = []
tx_ok_10 = []

best_dict = {
    "tx_ok_10":0,
    "tx_ok_20":0,
    "y_pred": 0,
    "y_true": 0,
    "RMSE": 0,
    "R2": 0,
    "R2_aj": 0
}
tic = time.perf_counter()
for r in range(Nr):
    rand_index = np.random.permutation(X.shape[0])

    X = X[rand_index,:]
    Y = Y[rand_index]

    spt_point = int(2/3*X.shape[0])
    X_train = X[:spt_point,:]
    Y_train = Y[:spt_point]

    X_test = X[spt_point:,:]
    Y_test = Y[spt_point:]

    W = np.random.normal(loc = 0, scale = 0.1, size = (q, X.shape[1]+1))

    Z = []

    for i in range(X_train.shape[0]):
        x = np.append(-1, X_train[i,:])
        U = np.dot(W, x)
```

```
        z = act_fun(U, fun_type)
        z = np.append(-1, z)
        Z.append(z)
    Z = np.array(Z)
    M = np.dot(np.linalg.pinv(Z), Y_train)

    count_20 = 0
    count_10 = 0
    y_pred = []
    RMSE = 0
    for j in range(X_test.shape[0]):
        x = np.append(-1, X_test[j,:])
        U1 = np.dot(W, x)
        z = act_fun(U1, fun_type)
        z = np.append(-1, z)
        y = np.dot(z, M)
        y_pred.append(y)
        RMSE = RMSE + RMSE_calc(Y_test[j], y)
        if np.abs(Y_test[j] - y) <= 0.2*np.abs(Y_test[j]):
            count_20 += 1
        if np.abs(Y_test[j] - y) <= 0.1*np.abs(Y_test[j]):
            count_10 += 1
    y_pred = np.array(y_pred)

    RMSE_r.append(RMSE)
    R2_r.append(R2(Y_test, y_pred))
    tx_ok_20.append(count_20/X_test.shape[0])
    tx_ok_10.append(count_10/X_test.shape[0])

    #Para ser a melhor rodada, deve ser melhor nas duas taxas de acerto
    if (best_dict["tx_ok_10"] <= count_10/X_test.shape[0]) & (best_dict["tx_ok_20"] <= count_20/X_test.shape[0]):
        best_dict["y_pred"] = y_pred
        best_dict["y_true"] = Y_test
        best_dict["R2"] = R2(Y_test, y_pred)
        best_dict["RMSE"] = RMSE
        best_dict["tx_ok_10"] = count_10/X_test.shape[0]
        best_dict["tx_ok_20"] = count_20/X_test.shape[0]

toc = time.perf_counter()

print("\n###### Resumo das {} Rodadas ######".format(Nr))
print('''Taxa de acerto para uma taxa de erro admssível de 20%:
```

```
Média = {:.2%},
Desv. Pad. = {:.2%},
Máximo = {:.2%},
Mínimo = {:.2%}\n'''.format(np.mean(tx_ok_20),np.std(tx_ok_20), np.max(tx_ok_20), np.min(tx_ok_20)))
print('''Taxa de acerto para uma taxa de erro admssível de 10%:
Média = {:.2%},
Desv. Pad. = {:.2%},
Máximo = {:.2%},
Mínimo = {:.2%}\n'''.format(np.mean(tx_ok_10), np.std(tx_ok_10), np.max(tx_ok_10), np.min(tx_ok_10)))
print('''R2:
Média = {:.3f},
Desv. Pad. = {:.3f},
Máximo = {:.3f},
Mínimo = {:.3f}\n'''.format(np.mean(R2_r), np.std(R2_r), np.max(R2_r), np.min(R2_r)))
print('''RMSE:
Média = {:.3f},
Desv. Pad. = {:.3f},
Máximo = {:.3f},
Mínimo = {:.3f}'''.format(np.mean(RMSE_r), np.std(RMSE_r), np.max(RMSE_r), np.min(RMSE_r)))

print("\n###### Melhor Rodada em relação as taxas de acerto para erros de 10% e 20% ######")
print("Taxa de acerto para uma taxa de erro admssível de 20% = {:.2%}".format(best_dict["tx_ok_20"]))
print("Taxa de acerto para uma taxa de erro admssível de 10% = {:.2%}".format(best_dict["tx_ok_10"]))
print("R2 = {:.3f}".format(best_dict["R2"]))
print("RMSE = {:.3f}".format(best_dict["RMSE"]))

print("\nTempo de Calibração e Validação = {:.3f} segundos, Rodadas = {}".format(toc - tic, Nr))
```

```
###### Resumo das 10 Rodadas ######
Taxa de acerto para uma taxa de erro admssível de 20%:
Média = 71.30%,
Desv. Pad. = 5.88%,
Máximo = 82.61%,
Mínimo = 62.32%

Taxa de acerto para uma taxa de erro admssível de 10%:
Média = 42.90%,
Desv. Pad. = 6.48%,
Máximo = 52.90%,
Mínimo = 32.61%

R2:
Média = 0.612,
Desv. Pad. = 0.080,
Máximo = 0.763,
Mínimo = 0.434

RMSE:
Média = 6.944,
Desv. Pad. = 0.743,
Máximo = 7.943,
Mínimo = 5.426

###### Melhor Rodada em relação as taxas de acerto para erros de 10% e 20% ######
Taxa de acerto para uma taxa de erro admssível de 20% = 82.61%
Taxa de acerto para uma taxa de erro admssível de 10% = 52.90%
R2 = 0.763
RMSE = 5.426

Tempo de Calibração e Validação = 0.271 segundos, Rodadas = 10
```

Yeh e Hsu (2018) apresentam o resultado da predição dos modelos de redes neurais na Tabela 7 de seu artigo. Os resultados são apresentados de acordo com os círculos de abastecimento e demanda apresentados no Trabalho, são eles: Sindia, Danshuei, Wunshan e Beitou. Os resultados também são apresentados em termos da média obtida para cada um desses círculos de abastecimento e demanda.

A base de dados disponibilizada refere-se as informações da círculo de abastecimento e demanda do Sindia, portanto os resultados obtidos nesse trabalho serão comparados aos resultados obtidos para essa região. Yeh e Hsu (2018) destacam que a base de dados total foi divida

aleatoriamente em 2/3 para treinamento e 1/3 para teste. As métricas apresentadas pelo autores correspondem a avaliação das predições para os dados de teste.

A ELM implementada possui a seguinte arquitetura (6, 5, 1). Adotou o número de neurônio ocultos $(q = 5)$ igual ao especificado por Yeh e Hsu (2018). A ELM foi rodada 10 vezes e, para cada uma dessas rodadas, embaralhou-se a base de dados e realizou-se a separação entre dados de treino e dados de teste, segundo as proporções especificadas pelos autores. Cabe destacar que esses dados de treino e teste foram previamente normalizados, trazendo todos os valores para uma escala comum sem distorcer as diferenças nos intervalos de valores. Como foi adotado a função de ativação logística, essa nova escala comum adotada foi $x_{norm} \in [0, 1]$

Foram comparadas 4 métricas: i) Taxa de acerto para um erro admissível de 20%, ii) Taxa de acerto para um erro admissível de 10%, iii) R^2 e iv) RMSE. A tabela abaixo apresenta as métricas para a rodada da ELM com melhor taxa de acerto para um erro tolerável de 10% e 20% e os valores obtidos por Yeh e Hsu (2018).

Métrica	Extreme Learning Machine	Yeh e Hsu (2018)
i)	82,61%	78,00%
ii)	52,90%	48,80%
iii)	0,763	0,627
iv)	5,426	8,06

Observa-se que a rede ELM implementada apresentou um desempenho superior à MLP implementada pelos autores para todas as métricas avaliadas. Esse melhor desempenho pode ser oriundo da aleatoriedade implementada na separação da base de treino e teste, existindo uma combinação de dados de treino e de teste que otimize essas métricas. Além do melhor desempenho, destaca-se a velocidade de treinamento e validação do algoritmo implementado, cerca de 0,271 segundos.

MLP

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from collections import namedtuple
import time
```

```
def act_fun(u, fun):
    action_fun = namedtuple("act_fun", ["f", "df"])
    if fun == "step":
        u[np.where(u >= 0)] = 1
        u[np.where(u < 0)] = 0
        du = np.nan
    elif fun == "tanh":
        u = np.array(list(map(lambda x: (1-np.exp(-x))/(1+np.exp(-x)), u)))
        du = 0.5*(1-np.power(u,2)) + 0.05
    elif fun == "log":
        u = np.array(list(map(lambda x: 1/(1+np.exp(-x)), u)), dtype = "float")
        du = u*(1-u)
    return (action_fun(f = u, df = du))

def norm_data(x):

    for i in range(x.shape[1]):
        x[:,i] = x[:,i]/x[:,i].max()
        # x[:,i] = 2*((x[:,i] - x[:,i].min())/(x[:,i].max() - x[:,i].min())) - 1
        #x_norm = 2*((x - x.min())/(x.max() - x.min()) - 1
    return x

def R2(y_true, y_pred):

    coef = 1 - (np.power(y_true-y_pred,2).sum()/np.power(y_true-y_true.mean(),2).sum())

    return (coef)

def RMSE_calc(y_true, y_pred):

    coef = np.power((np.power(y_true - y_pred, 2).sum())/y_true.shape[0], 0.5)

    return coef

def Hardamad_Prod(a, b):

    prod = np.array([np.multiply(x,y) for x, y in zip(a,b)])

    return prod

data = pd.read_excel("Real estate valuation data set.xlsx", index_col = 0)
```

```
X_real = data.iloc[:, 0:6].to_numpy()

Y_real = data.iloc[:, -1].to_numpy()

X = norm_data(X_real)
Y = norm_data(np.expand_dims(Y_real,1))

fun_type = "log"
Nr = 10
q = 5
eta = 0.6
Ne = 1500
mon = 0.9

RMSE_r = []
R2_r = []
tx_ok_20 = []
tx_ok_10 = []

best_dict = {
    "tx_ok_10":0,
    "tx_ok_20":0,
    "y_pred": 0,
    "y_true": 0,
    "RMSE": 0,
    "R2": 0,
    "R2_aj": 0
}

tic = time.perf_counter()

for r in range(Nr):
    rand_index = np.random.permutation(X.shape[0])

    X = X[rand_index,:]
    Y = Y[rand_index]

    spt_point = int(2/3*X.shape[0])
    X_train = X[:spt_point,:]
    Y_train = Y[:spt_point,:]

    X_test = X[spt_point:,:]
```

```
Y_test = Y[spt_point:,:]

W = {
    0: np.random.rand(q, X_train.shape[1]+1)*0.01,
    1: np.random.rand(Y_train.shape[1], q+1)*0.01
}
W_old = W.copy()

for e in range(Ne):
    rand_index = np.random.permutation(X_train.shape[0])
    X_train = X_train[rand_index,:]
    Y_train = Y_train[rand_index,:]


    for i in range(X_train.shape[0]):
        x = np.append(-1, X_train[i,:])
        U1 = np.dot(W[0], x)
        z, dz = act_fun(U1, fun_type)
        z = np.append(-1, z)
        U2 = np.dot(W[1], z)
        y, dy = act_fun(U2, fun_type)

        err = Y_train[i,:] - y

        err = np.expand_dims(Hardamad_Prod(err, dy), 1)
        x = np.expand_dims(x,1)
        z = np.expand_dims(z,1)

        W_aux = W.copy()

        DDi = Hardamad_Prod(dz, np.dot(W[1][:,1:].T, err))

        W[0] = W[0] + eta*np.dot(DDi, x.T) + mon*(W[0] - W_old[0])
        W[1] = W[1] + eta*np.dot(err, z.T) + mon*(W[1] - W_old[1])

        W_old = W_aux.copy()

RMSE_test = 0
count_20 = 0
count_10 = 0
y_pred = []
for j in range(X_test.shape[0]):
```

```
        x = np.append(-1, X_test[j,:])
        U1 = np.dot(W[0],x)
        z = act_fun(U1, fun_type).f
        z = np.append(-1, z)
        U2 = np.dot(W[1], z)
        y = act_fun(U2, fun_type).f
        y_pred.append(y)
        err = Y_test[j,:] - y

        RMSE_test = RMSE_test + RMSE_calc(Y_test[j,:], y)

        if np.abs(err) <= 0.2*np.abs(Y_test[j,:]):
            count_20 += 1
        if np.abs(err) <= 0.1*np.abs(Y_test[j,:]):
            count_10 += 1

    RMSE_r.append(RMSE_test)
    R2_r.append(R2(Y_test, y_pred))
    tx_ok_20.append(count_20/X_test.shape[0])
    tx_ok_10.append(count_10/X_test.shape[0])

    #Para ser a melhor rodada, deve ser melhor nas duas taxas de acerto
    if (best_dict["tx_ok_10"] <= count_10/X_test.shape[0]) & (best_dict["tx_ok_20"] <= count_20/X_test.shape[0]):
        best_dict["y_pred"] = y_pred
        best_dict["y_true"] = Y_test
        best_dict["R2"] = R2(Y_test, y_pred)
        best_dict["RMSE"] = RMSE_test
        best_dict["tx_ok_10"] = count_10/X_test.shape[0]
        best_dict["tx_ok_20"] = count_20/X_test.shape[0]

toc = time.perf_counter()

print("\n###### Resumo das {} Rodadas ######".format(Nr))
print('''Taxa de acerto para uma taxa de erro admssível de 20%:
Média = {:.2%},
Desv. Pad. = {:.2%},
Máximo = {:.2%},
Mínimo = {:.2%}\n'''.format(np.mean(tx_ok_20),np.std(tx_ok_20), np.max(tx_ok_20), np.min(tx_ok_20)))
print('''Taxa de acerto para uma taxa de erro admssível de 10%:
Média = {:.2%},
Desv. Pad. = {:.2%},
Máximo = {:.2%},
```

```
Mínimo = {:.2%}\n'''.format(np.mean(tx_ok_10), np.std(tx_ok_10), np.max(tx_ok_10), np.min(tx_ok_10)))
print('''R2:
Média = {:.3f},
Desv. Pad. = {:.3f},
Máximo = {:.3f},
Mínimo = {:.3f}\n'''.format(np.mean(R2_r), np.std(R2_r), np.max(R2_r), np.min(R2_r)))
print('''RMSE:
Média = {:.3f},
Desv. Pad. = {:.3f},
Máximo = {:.3f},
Mínimo = {:.3f}'''.format(np.mean(RMSE_r), np.std(RMSE_r), np.max(RMSE_r), np.min(RMSE_r)))

print("\n###### Melhor Rodada em relação as taxas de acerto para erros admissíveis de 10% e 20% ######")
print("Taxa de acerto para uma taxa de erro admssível de 20% = {:.2%}".format(best_dict["tx_ok_20"]))
print("Taxa de acerto para uma taxa de erro admssível de 10% = {:.2%}".format(best_dict["tx_ok_10"]))
print("R2 = {:.3f}".format(best_dict["R2"]))
print("RMSE = {:.3f}".format(best_dict["RMSE"]))

print("\nTempo de Calibração e Validação = {:.3f} segundos, Rodadas = {}, Épocas = {}".format(toc - tic, Nr, Ne))
```

```
###### Resumo das 10 Rodadas ######
Taxa de acerto para uma taxa de erro admssível de 20%:
Média = 67.68%,
Desv. Pad. = 11.39%,
Máximo = 80.43%,
Mínimo = 38.41%

Taxa de acerto para uma taxa de erro admssível de 10%:
Média = 38.70%,
Desv. Pad. = 7.96%,
Máximo = 48.55%,
Mínimo = 18.12%

R2:
Média = 0.492,
Desv. Pad. = 0.204,
Máximo = 0.686,
Mínimo = -0.034

RMSE:
Média = 7.771,
Desv. Pad. = 1.452,
Máximo = 11.351,
Mínimo = 6.277

###### Melhor Rodada em relação as taxas de acerto para erros admissíveis de 10% e 20% ######
Taxa de acerto para uma taxa de erro admssível de 20% = 80.43%
Taxa de acerto para uma taxa de erro admssível de 10% = 44.20%
R2 = 0.614
RMSE = 6.281

Tempo de Calibração e Validação = 1675.135 segundos, Rodadas = 10, Épocas = 1500
```

Yeh e Hsu (2018) apresentam o resultado da predição dos modelos de redes neurais na Tabela 7 de seu artigo. Os resultados são apresentados de acordo com os círculos de abastecimento e demanda apresentados no Trabalho, são eles: Sindia, Danshuei, Wunshan e Beitou. Os resultados também são apresentados em termos da média obtida para cada um desses círculos de abastecimento e demanda.

A base de dados disponibilizadas refere-se as informações da círculo de abastecimento e demanda do Sindia, portanto os resultados obtidos nesse trabalho serão comparados aos resultados obtidos para essa região. Yeh e Hsu (2018) destacam que a base de dados total foi divida

aleatoriamente em 2/3 para treinamento e 1/3 para teste. As métricas apresentadas pelo autores correspondem a avaliação das predições para os dados de teste.

A MLP implementada possui uma a seguinte arquitetura (6, 5, 1). Adotou-se o número de neurônio ocultos $(q = 5)$, taxa de aprendizado $(\eta = 0,6)$ e o número de épocas $(Ne = 1500)$ iguais ao especificado por Yeh e Hsu (2018). Yeh e Hsu (2018) não destacam a utilização do fator de momento, todavia optou-se pela sua utilização na MLP implementada no presente projeto, visando uma modificação mais estável dos pesos sinápticos ao longo do treinamento da rede. O fator de momento considerado foi igual a $0,9$

Tendo em vista a divisão aleatória da base de dados total em dados de treino e teste realizada por Yeh e Hsu (2018), a MLP implementada foi rodada 10 vezes e, para cada uma dessas rodadas, embaralhou-se a base de dados total e realizou-se a separação entre dados de treino e dados de teste, segundo as proporções especificadas pelos autores.

Da mesma forma que para a ELM, a base de dados total foi previamente normalizada antes do treinamento e validação da MLP, trazendo todos os valores para uma escala comum sem distorcer as diferenças nos intervalos de valores. Como foi adotado a função de ativação logística, essa nova escala comum adotada foi $x_{norm} \in [0, 1]$. Busca-se, com esse processo, melhorar a estabilidade e convergência do processo de aprendizado.

Foram comparadas 4 métricas: i) Taxa de acerto para um erro admissível de 20%, ii) Taxa de acerto para um erro admissível de 10%, iii) R^2 e iv) RMSE. A tabela abaixo apresenta as métricas para a rodada da MLP com melhor taxa de acerto para um erro de 10% e 20% e os valores obtidos por Yeh e Hsu (2018).

Métrica	MLP (6,5,1)	Yeh e Hsu (2018)
i)	80,43%	78,00%
ii)	44,20%	48,80%
iii)	0,614	0,627
iv)	6,281	8,06

Observa-se que a MLP implementada teve resultados próximos aos obtidos por Yeh e Hsu (2018). A MLP implementada teve um desempenho um pouco melhor para as métricas i) e iv), ficando atrás nas métricas ii) e iii). Apesar da arquitetura semelhante da MLP implementada no presente trabalho e a implementada por Yeh e Hsu (2018), destaca-se que a divisão aleatória da base de dados pode gerar combinações de dados de treino e de teste capazes de dificultar ou melhorar o treinamento e generalização da rede neural. Como pode ser

observado na estatística das métricas adotadas para as 10 rodadas, o R^2 teve um valor mínimo de -0,034 e um valor máximo de 0,614, evidenciando a influência dessa divisão aleatória da base de dados no desempenho da rede neural.

Comparando a MLP implementada com a ELM implementada no item anterior, pode-se constatar que a ELM teve um melhor desempenho em todas as métricas avaliadas, todavia, a influência da divisão aleatória da base de dados também é válida para essa comparação. Além dessas métricas, observa-se um tempo de treinamento e validação consideravelmente menor para ELM (0,271 segundos) em relação a MLP (1675,135 segundos), o que torna a ELM bem mais atrativa para solução do problema proposto.

Referências

Yeh, I-C.; Hsu, T-K. Building real estate valuation models with comparative approachthrough case-based reasoning, **Applied Soft Computing**, 65, 2018, p. 260-271, doi: 10.1016/j.asoc.2018.01.029
