UNIVERSIDADE FEDERAL DE MINAS GERAIS

ELT131 – REDES NEURAIS ARTIFICIAIS

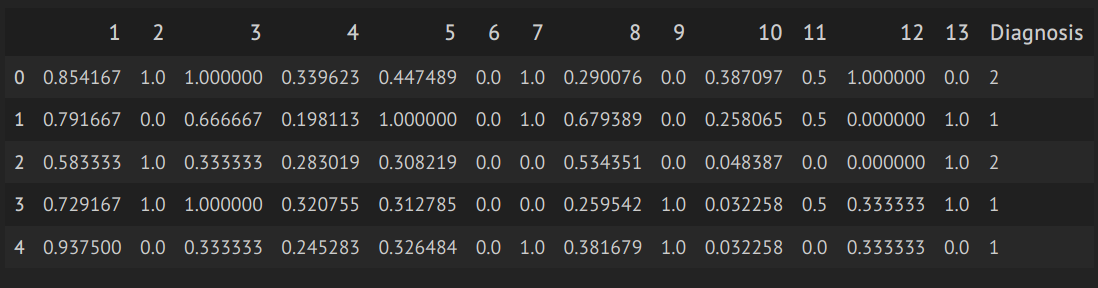
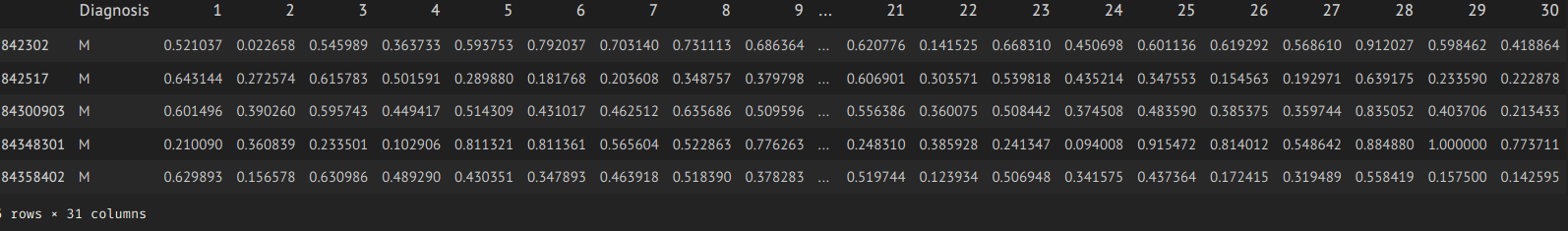
RELATÓRIO EXERCÍCIO 6

Gabriel Saraiva Espeschit – 2015065541

15 de setembro de 2020

Utilizando o módulo *Pandas,* os dados do *Breast Cancer (diagnostic)* e *Statlog (Heart)* foram lidos e colocados no formato de *dataframes* do *Pandas*. Em seguida, os atributos foram normalizados utilizando a função dada na guia para restringi-los para valores entre 0 e 1:

Visualizamos os dados do *dataframe* de câncer e coração (dispostos abaixo) para verificar que estavam corretamente normalizados e pronto para serem divididos em atributos e classes de teste e de treino.

Em seguida os *dataframes* de cada *dataset* foram convertidos em duas *arrays* de *Numpy.* Uma contendo somente os atributos e outra contendo somente as classes (diagnósticos) referente a cada atributo. Para cada *dataset* as classes foram convertidas para que ficasse inteligível para as funções de *Perceptreon* e *ELM* desenvolvidas para os exercícios passados. Isto é, as classificações ficaram expressas em -1/1 e 0/1 para serem usadas nos métodos de *ELM* e *Perceptron* respectivamente.

Utilizando a função *train\_test\_split* do módulo *SciKitLearn,* dividimos os datasets em instâncias para treino e instâncias para validação numa quantia de 70/30 para treinar as ELMs. Sendo assim, ficamos com vetores de treino e teste com as seguintes dimensões:

Dimensão atributos de treino *dataset* de câncer: (398, 30)

Dimensão do vetor de classes de treino *dataset* de câncer: (398, 1)

Dimensão atributos de teste *dataset* de câncer: (171, 30)

Dimensão do vetor de classes de teste *dataset* de câncer: (171, 1)

Dimensão atributos de treino *dataset* de coração: (189, 13)

Dimensão do vetor de classes de treino *dataset* de coração: (189, 1)

Dimensão atributos de teste *dataset* de coração: (81, 13)

Dimensão do vetor de classes de teste *dataset* de coração: (81, 1)

Em seguida, treinou a função ELM desenvolvida anteriormente com 5, 10, 20, 50, 100 e 300 neurônios. Calculamos a média da acurácia e o desvio padrão em cima dos dados de teste para 5 execuções de treinamento do algoritmo. Os resultados se encontram abaixo para cada *dataset*.

Acurácia câncer com 5 neurônios: 0.89 ± 0.03 %

Acurácia câncer com 10 neurônios: 0.93 ± 0.03 %

Acurácia câncer com 30 neurônios: 0.97 ± 0.01 %

Acurácia câncer com 50 neurônios: 0.98 ± 0.0 %

Acurácia câncer com 100 neurônios: 0.97 ± 0.01 %

Acurácia câncer com 300 neurônios: 0.86 ± 0.02 %

Acurácia coração com 5 neurônios: 0.67 ± 0.0 %

Acurácia coração com 10 neurônios: 0.77 ± 0.0 %

Acurácia coração com 30 neurônios: 0.8 ± 0.0 %

Acurácia coração com 50 neurônios: 0.75 ± 0.0 %

Acurácia coração com 100 neurônios: 0.72 ± 0.0 %

Acurácia coração com 300 neurônios: 0.62 ± 0.0 %

Podemos ver que, para a base de dados de câncer de mama o melhor número de neurônios é 50, pois apresenta melhor acurácia e menor desvio padrão, ou seja, os resultados são bem consistentes. Para a base de dados Heart, todos os resultados são consistentes entre si, porém apresentam uma acurácia mais baixa que para base de dados de câncer, o que é de se esperar, pois há um maior número de atributos e entidades na base de dados de câncer que na do coração.

Outro fenômeno que se pode observar é o *overfitting* que ocorre quando começa a se aumentar demais a quantidade de neurônios na modelo ELM.

Em seguida, se dividiu os dados para treinar o *perceptron* da mesma forma que fizemos com as ELMs resultando nas mesmas dimensionalidades que anteriormente. Calculamos a acurácia e o desvio padrão para 5 execuções de treino do *perceptron* tanto para o *dataset* de cancer, quanto para o do coração. Os resultados estão dispostos a seguir.

Acurácia câncer para perceptron: 0.98 ± 0.0 %

Acurácia coração para perceptron: 0.78 ± 0.02 %

Podemos concluir que ambas as formas de classificação produzem resultados semelhantes. Cada método tem suas vantagens e desvantagens: por um lado, as ELMs não precisam de um processo iterativo para chegar em uma solução, mas, por outro lado, para o caso dos *perceptrons* não é necessário ficar calibrando o número de neurônios a serem usados. Além disso, as operações feitas pelo *perceptron* são computacionalmente menos custosas que das ELMs, onde é necessário calcular a pseudo-inversa de uma matriz. O código utilizado para obter esse resultado, pode ser verificado no apêndice a seguir.

# Apêndice

import pandas as pd

import numpy as np

import func\_rna as func

# Utilizando o módulo Pandas, lemos os datasets de cancer e heart disponibilizados no relatório

colnames = list(map(str, range(1, 31)))

colnames.insert(0, 'Diagnosis')

colnames\_heart = list(map(str, range(1, 14)))

colnames\_heart.append('Diagnosis')

breast\_cancer = pd.read\_csv('wdbc.data', index\_col=0, names = colnames, header = None)

heart = pd.read\_csv('heart.dat', header = None, delim\_whitespace=True, names = colnames\_heart, index\_col = False)

# Fazemos a normalização dos features dos datasets

breast\_cancer\_x = breast\_cancer.iloc[:,1:31]

breast\_cancer\_x = (breast\_cancer\_x-breast\_cancer\_x.min())/(breast\_cancer\_x.max()-breast\_cancer\_x.min())

breast\_cancer.iloc[:,1:31] = breast\_cancer\_x

heart\_x = heart.iloc[:,0:13]

heart\_x = (heart\_x-heart\_x.min())/(heart\_x.max()-heart\_x.min())

heart.iloc[:,0:13] = heart\_x

# Visualizando o dataset

print(breast\_cancer.head())

# Visualizando o dataset Heart

print(heart.head())

# Covertemos os dados do dataset de Cancer de pandas dataframes para arrays de Numpy

X\_cancer = breast\_cancer.iloc[:,1:31].to\_numpy()

y\_cancer = breast\_cancer.Diagnosis.to\_numpy().reshape((-1,1))

# Aqui nos vamos adequar o Y para que o algoritmo de perceptron e ELM consiga fazer a classificação corretamente

y\_cancerP = np.where(y\_cancer == 'M', 1, 0)

y\_cancerELM = np.where(y\_cancer == 'M', 1, -1)

# Fazemos o mesmo para o dataset heart

X\_heart = heart.iloc[:,0:13].to\_numpy()

y\_heart = heart.Diagnosis.to\_numpy().reshape((-1,1))

y\_heartP = np.where(y\_heart == 1, 1, 0)

y\_heartELM = np.where(y\_heart == 1, 1, -1)

print(f'X Cancer Shape: {X\_cancer.shape}\nY Cancer shape: {y\_cancerP.shape}')

print(f'X Heart Shape: {X\_heart.shape}\nY Heart shape: {y\_heartP.shape}')

# Dividimos os dados para treinar o ELM

from sklearn.model\_selection import train\_test\_split as tts

X\_cancer\_train, X\_cancer\_test, y\_cancerELM\_train, y\_cancerELM\_test = tts(X\_cancer, y\_cancerELM, test\_size=0.3)

X\_heart\_train, X\_heart\_test, y\_heartELM\_train, y\_heartELM\_test = tts(X\_heart, y\_heartELM, test\_size=0.3)

print(f'X Cancer Training Shape: {X\_cancer\_train.shape}\nY Cancer Training shape: {y\_cancerELM\_train.shape}\nX Cancer Testing Shape: {X\_cancer\_test.shape}\nY Cancer Testing Shape: {y\_cancerELM\_test.shape}')

print(f'\n\nX Heart Training Shape: {X\_heart\_train.shape}\nY Heart Training shape: {y\_heartELM\_train.shape}\nX Heart Testing Shape: {X\_heart\_test.shape}\nY Heart Testing Shape: {y\_heartELM\_test.shape}')

# Treinando a ELM para base de dados Cancer

num\_neuronios = [5, 10, 30, 50, 100, 300]

for neuronio in num\_neuronios:

accuracy = []

for i in range(5):

W\_cancer, H\_cancer, Z\_cancer = func.ELM\_train(X\_cancer\_train, y\_cancerELM\_train, neuronio)

y\_h\_cancer = func.ELM\_y(X\_cancer\_test, W\_cancer, Z\_cancer)

accuracy.append(((y\_h\_cancer - y\_cancerELM\_test)\*\*2/4).mean())

print(f'Accurácia CANCER com {neuronio} neuronios: {round(np.mean(accuracy), 2)} \u00B1 {round(np.std(accuracy), 2)} %')

# Treinando a ELM para base de dados Heart

for neuronio in num\_neuronios:

accuracy = []

for i in range(5):

W\_heart, H\_heart, Z\_heart = func.ELM\_train(X\_heart\_train, y\_heartELM\_train, neuronio)

y\_h\_heart = func.ELM\_y(X\_heart\_test, W\_heart, Z\_heart)

accuracy = ((y\_h\_heart - y\_heartELM\_test)\*\*2/4).mean()

print(f'Accurácia HEART com {neuronio} neuronios: {round(np.mean(accuracy), 2)} \u00B1 {round(np.std(accuracy), 2)} %')

# Agora dividimos os dados para treinar o perceptron

X\_cancer\_train, X\_cancer\_test, y\_cancerP\_train, y\_cancerP\_test = tts(X\_cancer, y\_cancerP, test\_size=0.3)

X\_heart\_train, X\_heart\_test, y\_heartP\_train, y\_heartP\_test = tts(X\_heart, y\_heartP, test\_size=0.3)

print(f'X Cancer Training Shape: {X\_cancer\_train.shape}\nY Cancer Training shape: {y\_cancerP\_train.shape}\nX Cancer Testing Shape: {X\_cancer\_test.shape}\nY Cancer Testing Shape: {y\_cancerP\_test.shape}')

print(f'\n\nX Heart Training Shape: {X\_heart\_train.shape}\nY Heart Training shape: {y\_heartP\_train.shape}\nX Heart Testing Shape: {X\_heart\_test.shape}\nY Heart Testing Shape: {y\_heartP\_test.shape}')

# Treinando o perceptron para base de dados Cancer

accuracy = []

for i in range(5):

wt\_cancer, e\_cancer = func.trainperceptron(X\_cancer\_train, y\_cancerP\_train, 0.0001, 0.0001, 100)

y\_cancer\_pred = func.yperceptron(X\_cancer\_test, wt\_cancer)

accuracy = (y\_cancer\_pred == y\_cancerP\_test).mean()

print(f'Accurácia CANCER para perceptron: {round(np.mean(accuracy), 2)} \u00B1 {round(np.std(accuracy), 2)} %')

# Treinando o perceptron para base de dados Heart

accuracy = []

for i in range(5):

wt\_heart, e\_heart = func.trainperceptron(X\_heart\_train, y\_heartP\_train, 0.0001, 0.0001, 100)

y\_heart\_pred = func.yperceptron(X\_heart\_test, wt\_heart)

accuracy.append((y\_heart\_pred == y\_heartP\_test).mean())

print(f'Accurácia HEART para perceptron: {round(np.mean(accuracy), 2)} \u00B1 {round(np.std(accuracy), 2)} %')