UNIVERSIDADE FEDERAL DE MINAS GERAIS

ELT131 – REDES NEURAIS ARTIFICIAIS

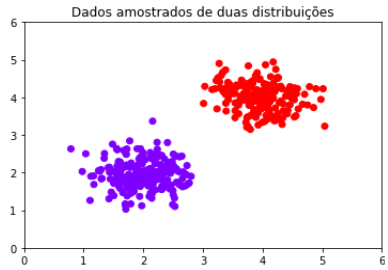
RELATÓRIO TREINAMENTO PERCEPTRON SIMPLES

Gabriel Saraiva Espeschit – 2015065541

31 de agosto de 2020

EXERCÍCIO 1

Utilizando a função *cria_dados, f*oram gerados duas distribuições normais, com 200 amostras cada, centradas em M1 = [2, 2,] e M2 = [4, 4] e com desvio padrão de 0,4 para ambas. Os dados gerados podem ser vistos abaixo.



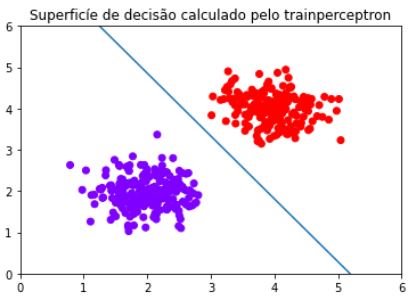
Com parâmetros de *eta* e *tol* em 0.001, o vetor de pesos *w* foi calculado utilizando a função *trainperceptron.*

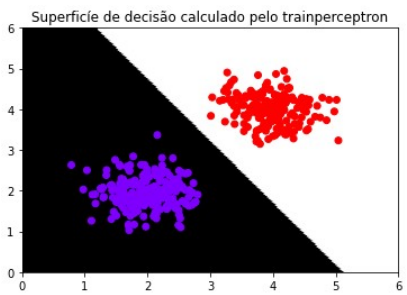
A função *trainperceptron* aplica a metodologia aprendida para calcular os valores do peso do perceptron utilizando o erro do y previsto pela função com os pesos atuais e o y real, assim como a taxa de aprendizado *eta*. Isso é feito de forma iterativa até que uma tolerância (*tol*) mínima seja respeitada ou um número máximo de iterações seja percorrido.

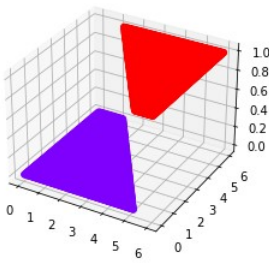
Com isso, foi possível visualizar a superfície de separação de três maneiras:

- Calculando os parâmetros da reta de separação como representado abaixo, e plotando a reta de separação.
- Criando um *grid* de pontos que englobam a região de interesse e utilizando a função *yperceptron*, pode-se classificar os pontos desse *grid* e plotá-los em um gráfico bidimensional.
- Podemos fazer a mesma coisa que acima, porém com um gráfico tridimensional.

As três formas de visualização dos dados podem ser vistas abaixo.



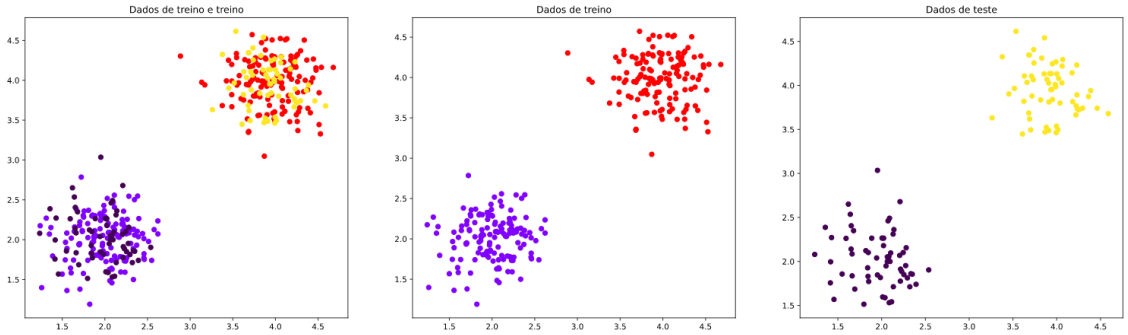




Com isso podemos ver que todas nossas funções estão desempenhando conforme esperado e o algoritmo é capaz de criar uma superfície de decisão para classificação dos dados gerados. No exercício seguinte, veremos como podemos causar que essa superfície seja indevidamente projetada devido à divisão de dados de treino e teste. O código correspondente a esse exercício poderá ser verificado no apêndice A1 desse relatório.

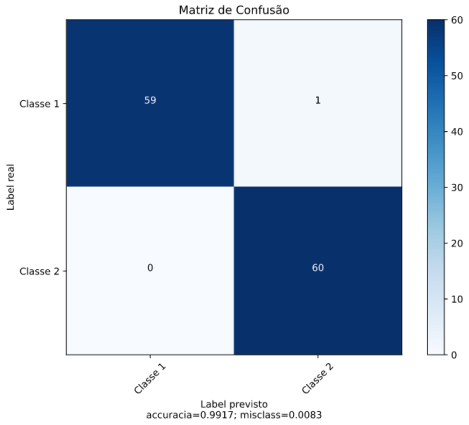
EXERCÍCIO 2

Nesse exercício, os mesmos parâmetros para criação de dados foram usados que no primeiro, exceto pelo desvio padrão o qual teve valor de 0,3. A diferença esteve na separação desses dados em dados de treino e dados de teste. Para tal foi utilizada a função *train_test_balanceados*. Essa função randomiza a matriz contendo os dados do problema e retorna *train_v* dados para treino e 1-*train_v* dados para teste. Nesse exercício, optamos por usar 70% de dados de treino. Abaixo podemos ver como que ficou essa divisão.

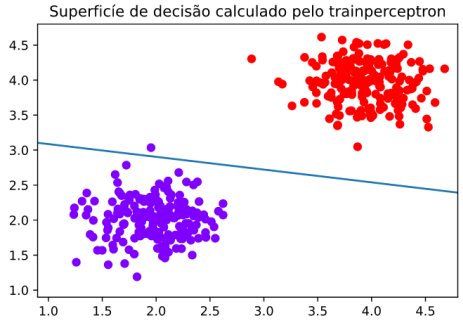


Em seguida o procedimento foi semelhante ao adotado no primeiro exercício. Os dados de treino foram utilizados para calcular o vetor peso. Em seguida, utilizando a função *yperceptron,* que calcula a resposta do perceptron dado os pesos *w* e os valores *xin* de entrada, os valores de teste foram usados para verificar a acurácia do perceptron treinado. A acurácia obtida foi de 99.17%.

Uma matriz de confusão foi gerada utilizando o modulo *confusion_matrix* do pacote *sklearn.* Para deixar a matriz mais visual, utilizou-se o código encontrado na documentação do módulo[1]. Abaixo podemos verificar a matriz de confusão gerada.



Como podemos ver 1 ponto foi indevidamente classificado. Plotando a superfície de decisão fica evidente o ponto que causou o único erro do sistema treinado.



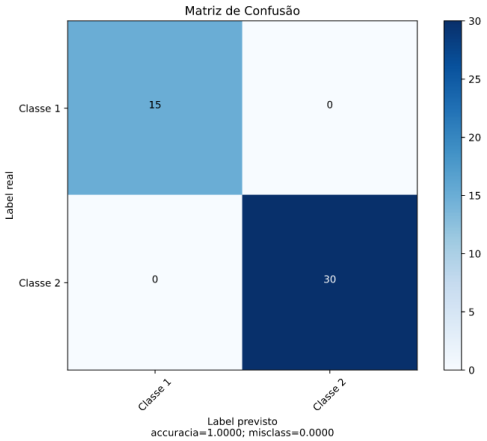
O código correspondente a esse exercício poderá ser verificado no apêndice A2 desse relatório.

EXERCÍCIO 3

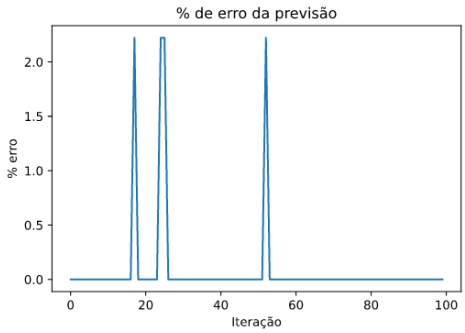
Para o terceiro exercício, foi utilizado a base de dados *iris,* obtida por meio do pacote *sklearn*. Foi necessário fazer uma adequação dos dados para que ele pudessem ser usados da forma correta para esse exercício. Primeiramente, trocamos todos os elementos da classe 2 para elementos da classe 1.

Em seguida, as características e as classes do dado *iris* foram unidas em uma única matriz de tamanho 150x5 que, por sua vez, foi alimentada na função *train_test_balanceado* onde foi capaz de se obter os dados de treino (70% do total) e de teste (30%).

Em seguida o mesmo processo do exercício 2 foi feito: calculamos os pesos w_i usando a função *trainperceptron*, fizemos as previsões para y com a função *yperceptron* e visualizamos a matriz de confusão e acurácia:



Por fim, esse processo foi feito de forma iterativa, por 100 iterações, armazenando em cada uma delas a porcentagem de erros cometidos. Abaixo pode ser visualizado o gráfico de % de erros por iteração.



Em conclusão, podemos observar que o *perceptron* é adequado para fazer a separação de dados utilizando um semiplano. O código correspondente a esse exercício poderá ser verificado no apêndice A3 desse relatório. As funções que foram desenvolvidas pelo aluno e utilizadas em todos exercícios estão no apêndice B.

Referências

[1] **Confusion matrix — scikit-learn 0.23.2 documentation**. Disponível em: <https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html>. Acesso em: 31 ago. 2020.

[2] **NumPy Documentation**. Disponível em: <https://numpy.org/doc/>. Acesso em: 31 ago. 2020.

[3] BRAGA, A.P. Notas de Aula.

Apêndice A1 – Código do Exercício 1

```
import numpy as np
import matplotlib.pyplot as plt
import perceptron as fp
centros = [[2,2], [4,4]]
desvios_padroes = [0.4, 0.4]
num_amostras = 200
num_distrbuições = 2
X = fp.cria_dados(num_amostras, centros, desvios_padroes,
                  num_distrbuições)

plt.scatter(X[:, 0], X[:, 1], c=X[:, 2], cmap='rainbow')
plt.title('Dados amostrados de duas distribuições')
plt.xlim(0, 6)
plt.ylim(0, 6)
plt.show()

wt, evac = fp.trainperceptron(xin = X[:, :2], yd = X[:, 2], eta = 0.001,
                              tol = 0.001, maxepocas = 100)
b, w1, w2 = wt
#Calculando a reta da superfície de separação
x = -b/w1
y = -b/w2
d = y
c = -y/x
line_x_coords = np.array([0, x])
line_y_coords = c * line_x_coords + d
plt.plot(line_x_coords, line_y_coords)
plt.scatter(X[:, 0], X[:, 1], c = X[:, 2], cmap='rainbow')
plt.title('Superficíe de decisão calculado pelo trainperceptron')
plt.xlim(0, 6)
plt.ylim(0, 6)
plt.show()

x1_lin = np.linspace(0, 6, 200)
x2_lin = np.linspace(0, 6, 200)
X1, X2 = np.meshgrid(x1_lin, x2_lin)
X_lin = np.column_stack((X1.ravel(),X2.ravel()))
pred_class = fp.yperceptron(xin=X_lin, w = wt)
plt.scatter(X_lin[:, 0], X_lin[:, 1], c = (pred_class), cmap='gist_gray')
plt.scatter(X[:, 0], X[:, 1], c = X[:, 2], cmap='rainbow')
plt.title('Superficíe de decisão calculado pelo trainperceptron')
plt.xlim(0, 6)
plt.ylim(0, 6)
plt.show()

ax = plt.axes(projection='3d')
ax.scatter3D(X_lin[:,0], X_lin[:,1], pred_class, c=pred_class,
             cmap = 'rainbow' )
plt.show()
```

Apêndice A2 – Código do Exercício 2

```
import numpy as np
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import perceptron as fp

cria_dados(num_amostras, centros, des_p, dist)
centros = [[2, 2], [4, 4]]
des_p = [0.3, 0.3]
num_amostras = 200
dist = 2
dados = fp.cria_dados(num_amostras, centros, des_p, dist)

train, test = fp.train_test_balanceados(dados, 0.7)

fig, ax = plt.subplots(1, 3, figsize=(25,7))
ax[0].scatter(train[:,0], train[:,1], c = train[:,2], cmap='rainbow')
ax[0].scatter(test[:,0], test[:, 1], c = test[:,2])
ax[0].set_title('Dados de treino e treino')
ax[1].scatter(train[:,0], train[:,1], c = train[:,2], cmap='rainbow')
ax[1].set_title('Dados de treino')
ax[2].scatter(test[:,0], test[:, 1], c = test[:,2])
ax[2].set_title('Dados de teste')
plt.show()

wt, e = fp.trainperceptron(train[:,:2], train[:,2], 0.001, 0.001, 100)

y_pred = fp.yperceptron(test[:, :2], wt)

certos = 0
errados = 0
for classe_prevista, classe_real in zip(y_pred, test[:,2]):
    if (classe_prevista == classe_real):
        certos += 1
    else:
        errados += 1
print(f'Numero de acertos: {certos}\nNúmeros de erros: {errados}\n
Porcentagem de acertos: {certos/(certos+errados)*100:.2f}%')

conf_mat = confusion_matrix(test[:,2], y_pred)
fp.plot_confusion_matrix(conf_mat, ['Classe 1', 'Classe 2'], title='Matriz de
Confusão', cmap='Blues', normalize = False)

b, w1, w2 = wt
x = -b/w1
y = -b/w2
d = y
c = -y/x
line_x_coords = np.array([0, x])
line_y_coords = c * line_x_coords + d
plt.plot(line_x_coords, line_y_coords)
plt.scatter(dados[:, 0], dados[:, 1], c = dados[:, 2], cmap='rainbow')
plt.title('Superficíe de decisão calculado pelo trainperceptron')
plt.xlim(0.9, 4.8)
plt.ylim(0.9, 4.8)
plt.show()
```

Apêndice A3 – Código do Exercício 3

```
import numpy as np
import matplotlib.pyplot as plt
import perceptron as fp
from sklearn import datasets
from sklearn.metrics import confusion_matrix

iris = datasets.load_iris()
X = iris.data
y = iris.target.T
y[y[:]==2] = 1
y = y.reshape(-1,1)
data = np.append(X,y,1)

train, test = fp.train_test_balanceados(data, train_v=0.7)

wt, evel = fp.trainperceptron(train[:,:-1], train[:, -1], 0.001, 0.001, 100)

y_pred = fp.yperceptron(test[:,:-1], wt)

conf_mat = confusion_matrix(test[:,-1], y_pred)
fp.plot_confusion_matrix(conf_mat, ['Classe 1', 'Classe 2'],
                         title='Matriz de Confusão', cmap='Blues', normalize = False)

erro = []
for i in range(100):
    train, test = fp.train_test_balanceados(data, train_v=0.7)
    wt, evel = fp.trainperceptron(train[:,:-1], train[:, -1], 0.001, 0.001, 100)
    y_pred = fp.yperceptron(test[:,:4], wt)
    certos = 0
    errados = 0
    for classe_prevista, classe_real in zip(y_pred, test[:,-1]):
        if (classe_prevista == classe_real):
            certos += 1
        else:
            errados += 1
    erro.append(errados/(certos+errados)*100)

plt.plot(erro)
plt.title('% de erro da previsão')
plt.ylabel('% erro')
plt.xlabel('Iteração')
plt.show()
```

Apêndice B – Funções utilizadas em todos exercícios

```
import numpy as np
def cria_dados (num_amostras, centros, des_padroes, num_dist, dim
=2):
'''
Função que gera conjuntos de dados utilizando a distribuição normal
de num_dists 'classes' diferentes
Entradas:
- num_amostras: numero de amostras por conjunto de dados
- centros: listas contendo os centros de cada distribuição normal
- des_padroes: lista contendo os desvios padrões de cada
distribuição normal
- num_dist: numero de classes a ser criado
- dim: numero de dimensões de cada conjunto de dados
Retorna:
data_set: data-set criado
'''
data_set = np.empty((num_amostras*num_dist, dim+1))
for i in range(num_dist):
x = np.random.normal(loc=centros[i], scale=des_padroes[i],
size=(num_amostras,dim))
dist_col = np.full((num_amostras, 1), i)
data_set[num_amostras*i:(i+1)*num_amostras, :] = np.append(x,
dist_col, 1)
return data_set

def trainperceptron (xin, yd, eta, tol, maxepocas):
'''
Função que aplica o metodo para treinamento de perceptron
yd: tem que ser garado para as xin (concatenado xall), metade 0 e
metade 1
xin: Entrada Nxn de dados de matriz
eta: Peso de atualizacao do passo
tol: tolerancia do erro
maxepocas: numero maximo de epocas permitido
retorna:
- wt: parametros da função avaliada
- evec: erro médio por época
'''

N = xin.shape[0] #recebe as linhas
n = xin.shape[1] # recebe as colunas
xin = np.append(np.ones((N,1)), xin,axis = 1)

wt = np.random.randn(n+1, 1)*0.01
nepocas = 0
eepoca = tol+1
# inicializa vetor erro evec
evec = np.empty([maxepocas+1, 1])
while ((nepocas < maxepocas) and (eepoca>tol)): #eepocas erro
da epoca e tol tolerancia
ei2 = 0
if (nepocas+1)%10 == 0:
print(f'Epoca: {nepocas+1}')
#sequencia aleatoria para treinamento
xseq = (np.arange(N))
np.random.shuffle(xseq)
for i in range(N):
#padrao para sequencia aleatoria
irand = xseq[i]
yhati = (np.matmul(xin[None, irand, :], wt)) >= 0
ei = yd[irand]-yhati
dw = eta * ei * xin[None, irand, :]
#atualizacao do peso w
wt = wt + dw.T
#erro acumulado
ei2 = ei2+ei*ei
#numero de epocas
nepocas = nepocas+1
evec[nepocas] = ei2/N
#erro por epoca
eepoca = evec[nepocas]
return wt, evec[1:nepocas]

def yperceptron(xin, w):

'''
Função que retorna a saída de um sistema cujo parametros
foram obtidos usando a função trainadaline
xin: vetor x de entrada
w: parametros a serem considerados
retorna: vetor y correspondente ao modelo com parametros w
'''

return ((np.matmul(np.append(np.ones((xin.shape[0],1)), xin,axis = 1
), w))>=0).astype(int)
```

```
def train_test_balanceados(data, train_v = 0.9, dist = 2):
'''
Fazer divisão em dados de teste e treino
Argumentos:
dados = dados que se deseja dividir
train_v = Porcentagem de dados de treino (deve ser menor que 1)
dist = número de distribuições no meu dataframe
Retorna: Vetores X, Y de treino e teste adequadamente distribuidos
'''
assert(train_v<=1), 'train_v deve ser menor que 1'
np.random.shuffle(data)
X_train, y_train, X_test, y_test = [], [], [], []

for i in range(dist):
dados = np.copy(data[data[:, -1] == i])
train, test = dados[:int(0.7*dados.shape[0])],
dados[int(0.7*dados.shape[0]):]
    X_train.append(train[:, :train.shape[1]-1])
    y_train.append(train[:,train.shape[1]-1:])
    X_test.append(test[:, :train.shape[1]-1])
    y_test.append(test[:,train.shape[1]-1:])
    X_train = np.concatenate(X_train)
    y_train = np.concatenate(y_train)
    train = np.append(X_train, y_train, 1)
    X_test = np.concatenate(X_test)
    y_test = np.concatenate(y_test)
    test = np.append(X_test, y_test, 1)

    np.random.shuffle(train)
    np.random.shuffle(test)

    return (train, test)
```