Universidade Federal de Santa Catarina
Programa de Pós-Graduação em Ciência da Computação
Inteligência Artificial Conexionista

Augusto André Souza Berndt 202000735

Exercício 2.

A biblioteca numpy foi utilizada para realizar os problemas do exercício. Aideia é realizar uma regressão logística, primeiramente se realiza a visualização dos dados, utilizando matplotlib.

```
# Load data
# The first two columns contains the exam scores and the thi
# contains the label.
data = np.loadtxt(os.path.join('Data', 'ex2data1.txt'), deli
X, y = data[:, 0:2], data[:, 2]
```

Figura 1.

Antes do primeiro desafio, é feita a leitura dos dados disponíveis em *"Data/ex2data1.txt",* que dizem respeito a notas de alunos que foram admitidos ou não em uma universidade. As duas primeiras colunas são armazenadas em X, representando a nota de cada aluno nas duas provas, em y é armazenado os targets, que é um valor binário (1 ou 0) representando se o aluno foi admitido ou não na universidade.

A respeito do primeiro desafio, o código resposta já é fornecido, fazemos então uma analise do que é feito. Se utiliza desta parte do código (figuras 2 e 3) para fazer uma visualização dos dados de maneira "crua". Podemos ver que existe um clara divisória entre os alunos que foram admitidos e os que não foram. É interessante criar um modelo matemático capaz de identificar esta divisora automaticamente, capaz também de generalizar para novos indivíduos ainda não vistos.

```
    ============
    Plot the positive and negative examples on a 2D plot, us
    option 'k*' for the positive examples and 'ko' for the n
    """
    # Create New Figure
    fig = pyplot.figure()

    # ====================== YOUR CODE HERE ================
    # Find Indices of Positive and Negative Examples
    pos = y == 1
    neg = y == 0

    # Plot Examples

    # add axes labels
#     pyplot.xlabel('Exam 1 score')
#     pyplot.ylabel('Exam 2 score')
#     pyplot.legend(['Admitted', 'Not admitted'])
#     pass
    pyplot.plot(X[pos, 0], X[pos, 1], 'k*', lw=2, ms=10)
    pyplot.plot(X[neg, 0], X[neg, 1], 'ko', mfc='y', ms=8, m

    # ======================================================
# plotData(X, y)
```

Figura 2.

```
plotData(X, y)
# # add axes labels
pyplot.xlabel('Exam 1 score')
pyplot.ylabel('Exam 2 score')
pyplot.legend(['Admitted', 'Not admitted'])
pass
```

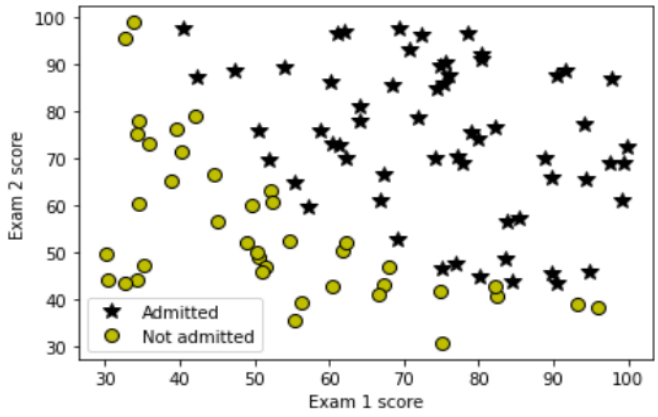


Figura 3.

Para realizar a demonstração é utilizado a biblioteca matplotlib. Os alunos que foram admitidos, ou seja, aqueles em X que tem um valor positivo para y são selecionados com a variável pos (*pos = y==1*) inversamente para a variável neg. Com X[pos,0] e X[pos,1] se seleciona os valores do exame 1 (primeira coluna 0) e exame 2 (segunda coluna 1) dos indivíduos que foram admitidos e se desenha uma estrela para cada um desses pontos com o parâmetro k* do pytplot.plot. O mesmo é feito para indivíduos não admitidos, com um círculo amarelo.

Após a visualização do dataset de entrada é feita a implementação de uma função sigmoid, esta função deve funcionar tanto para listas como matrizes de entrada e disponibiliza uma saída com shape equivalente ao de entrada onde os elementos passam pela função sigmoid, um a um. A solução foi feita com a biblioteca *math* para utilizar a primitiva *exp()*.

```
def sigmoid(z):
    """
    Compute sigmoid function given the input z.

    Parameters
    ----------
    z : array_like
        The input to the sigmoid function. This can be a 1-D vector
        or a 2-D matrix.

    Returns
    -------
    g : array_like
        The computed sigmoid function. g has the same shape as z, since
        the sigmoid is computed element-wise on z.

    Instructions
    ------------
    Compute the sigmoid of each value of z (z can be a matrix, vector or scalar).
    """
    # convert input to a numpy array
    z = np.array(z)

    # You need to return the following variables correctly
    g = np.zeros(z.shape)

    # ====================== YOUR CODE HERE ======================
    g = 1/(1+(math.exp(-z)))


    # =============================================================
    return g
```

Figura 4.

A função implementada foi experimentada com algumas valores de entrada, como consta na Figura 5. Os valores obtidos como saída estão de acordo com o esperado de uma função sigmoide.

```
g( 0 ) =  0.5
g( -1 ) =  0.2689414213699951
g( -2 ) =  0.11920292202211755
g( -10 ) =  4.5397868702434395e-05
g( -35 ) =  6.305116760146985e-16
g( -40 ) =  4.248354255291589e-18
g( -100 ) =  3.7200759760208356e-44
g( -200 ) =  1.3838965267367376e-87
g( 0.1 ) =  0.52497918747894
g( 0.01 ) =  0.5024999791668749
g( 0.0001 ) =  0.5000249999999792
g( 2 ) =  0.8807970779778823
g( 3 ) =  0.9525741268224334
g( 5 ) =  0.9933071490757153
g( 35 ) =  0.9999999999999993
g( 100 ) =  1.0
```

Figura 5.

A Figura 6 mostra a implementação do custo *J* e do gradiente *grad* para valores de theta como por padrão do exercício, todos em zero.

```
    # ===================== YOUR CODE HERE ======================
    h=sigmoid(np.dot(X,theta))
    J= 1/m * (np.sum((-y*np.log(h))-((1-y)*np.log(1-h))))
    grad= 1/m * np.dot(X.T,h-y)
    # =============================================================
    return J, grad
```

e you are done call your `costFunction` using two test cases for θ by executing the next cell.

```
# Initialize fitting parameters
initial_theta = np.zeros(n+1)

cost, grad = costFunction(initial_theta, X, y)

print('Cost at initial theta (zeros): {:.3f}'.format(cost))
print('Expected cost (approx): 0.693\n')

print('Gradient at initial theta (zeros):')
print('\t[{:.4f}, {:.4f}, {:.4f}]'.format(*grad))
print('Expected gradients (approx):\n\t[-0.1000, -12.0092, -11.2628]\n')

# Compute and display cost and gradient with non-zero theta
test_theta = np.array([-24, 0.2, 0.2])
cost, grad = costFunction(test_theta, X, y)

print('Cost at test theta: {:.3f}'.format(cost))
print('Expected cost (approx): 0.218\n')

print('Gradient at test theta:')
print('\t[{:.3f}, {:.3f}, {:.3f}]'.format(*grad))
print('Expected gradients (approx):\n\t[0.043, 2.566, 2.647]')
```

```
Cost at initial theta (zeros): 0.693
Expected cost (approx): 0.693

Gradient at initial theta (zeros):
        [-0.1000, -12.0092, -11.2628]
Expected gradients (approx):
        [-0.1000, -12.0092, -11.2628]

Cost at test theta: 0.218
Expected cost (approx): 0.218

Gradient at test theta:
        [0.043, 2.566, 2.647]
Expected gradients (approx):
        [0.043, 2.566, 2.647]
```

Figura 6.

A Figura 7 apresenta um custo inicial para um theta inicial de valores aleatorios. O custo inicial se mostra muito maior do que o iniciado com zeros, isto é esperado pois não se deve utilizar valores muito altos de theta.

```
# Initialize fitting parameters
# initial_theta = np.zeros(n+1)
initial_theta = np.random.randn(n+1)
cost, grad = costFunction(initial_theta, X, y)
print('Cost at initial theta (zeros): {:.3f}'.format(cost))
print('Expected cost (approx): 0.693\n')
print('Gradient at initial theta (zeros):')
print('\t[{:.4f}, {:.4f}, {:.4f}]'.format(*grad))
print('Expected gradients (approx):\n\t[-0.1000, -12.0092, -11.2628]\n')
# Compute and display cost and gradient with non-zero theta
test_theta = np.array([-24, 0.2, 0.2])
cost, grad = costFunction(test_theta, X, y)
print('Cost at test theta: {:.3f}'.format(cost))
print('Expected cost (approx): 0.218\n')
print('Gradient at test theta:')
print('\t[{:.3f}, {:.3f}, {:.3f}]'.format(*grad))
print('Expected gradients (approx):\n\t[0.043, 2.566, 2.647]')
```

```
Cost at initial theta (zeros): 16.517
Expected cost (approx): 0.693

Gradient at initial theta (zeros):
        [-0.5884, -43.7988, -43.9524]
Expected gradients (approx):
        [-0.1000, -12.0092, -11.2628]

Cost at test theta: 0.218
Expected cost (approx): 0.218

Gradient at test theta:
        [0.043, 2.566, 2.647]
Expected gradients (approx):
        [0.043, 2.566, 2.647]
```

Figura 7.

Em seguida se faz a busca por um theta inicial com a função optimize do scipy. Os valores por padrão são demonstrados na Figura 8.

```
Cost at theta found by optimize.minimize: 0.203
Expected cost (approx): 0.203

theta:
        [-25.161, 0.206, 0.201]
Expected theta (approx):
        [-25.161, 0.206, 0.201]
```

Once `optimize.minimize` completes, we want to use the final value for θ to visualize the decision boundary on the training data as shown in the figure below.

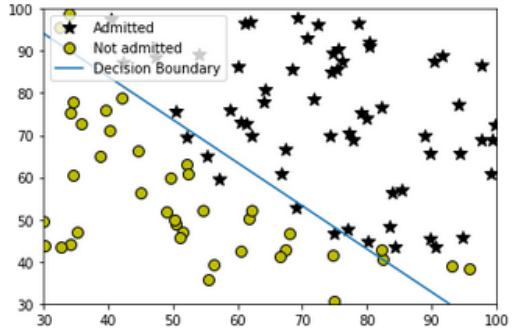


Figura 8.

Já para valores aleatórios se obteve uma divisão por zero, provavelmente se deve ao problema de *vanishing gradient.* A figura 9 demonstra a tentativa para valores aleatórios de theta inicial.

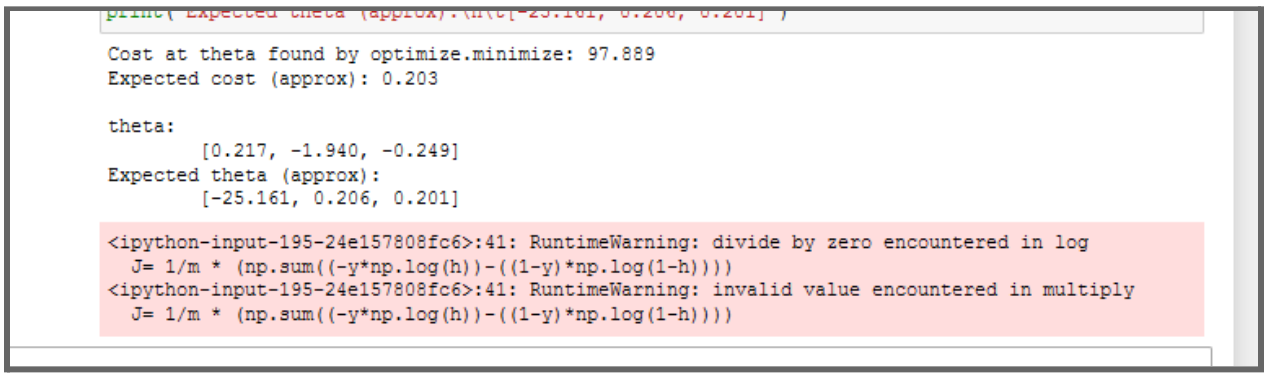
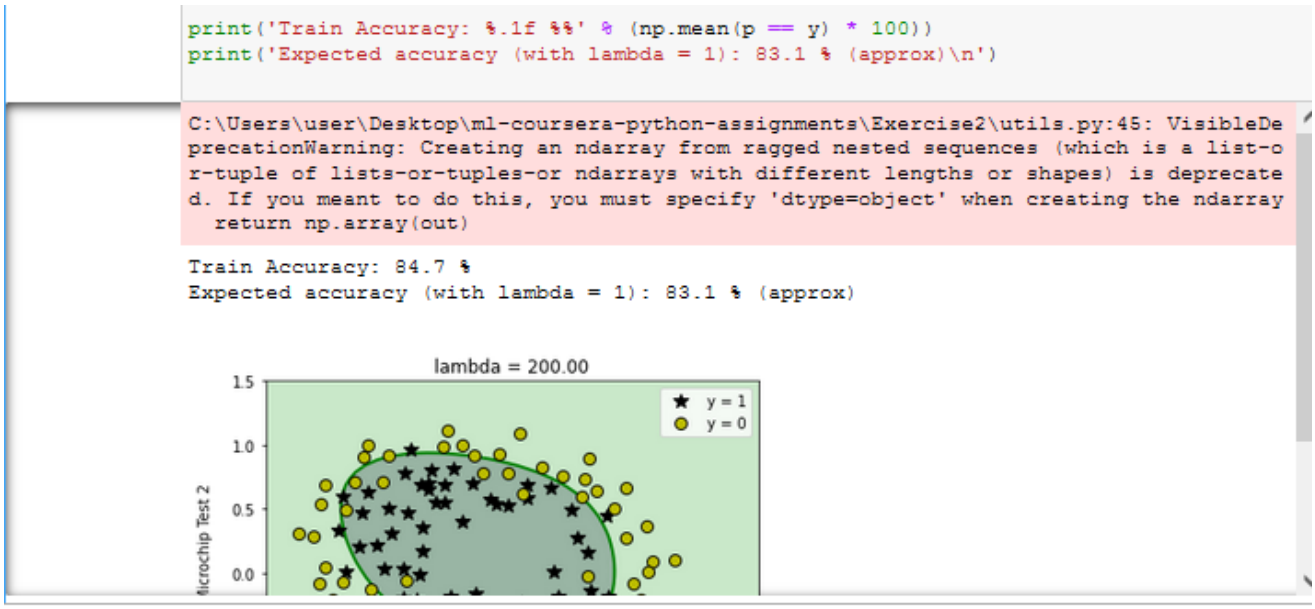


Figura 9.

Na segunda parte do exercício se faz uma regressão logística regularizada. E se utiliza outro dataset composto de microchips

Pelo visto há um bug conhecido e em aberto na versão em python do exercício 2. A figura 10 mostra a mesagem de warning. E há um issue em aberto no github do curso desde junho com relação a este mesmo problema:
https://github.com/dibgerge/ml-coursera-python-assignments/issues/77



Type *Markdown* and LaTeX: α^2

Figura 10.

No entanto, mesmo se substituindo os valores de lambda a acurácia não se demonstra alterar, algo não está acontecendo