Trabalho 2: Regressão Logística

Frederico Schardong

1 Enunciado do trabalho

O enunciado do trabalho[1] propõem a resolução de dois exercícios de regressão logística: linear e não-linear, ambos utilizando um único percéptron. O primeiro consiste na previsão da admissão de estudantes, considerando a nota em duas avaliações. Os possíveis resultados são aprovado ou reprovado. O segundo exercício também consiste em duas dimensões de entrada e uma de saída. O contexto do problema é na aprovação ou reprovação de microchips com base no resultado de dois testes de qualidade. A Figura 1 mostra os dados de entrada dos dois exercícios e as respectivas classificações.

2 Detalhes de implementação

O trabalho foi implementado em Python 3, utilizando as bibliotecas: (i) `numpy` [3] para armazenar e calcular matrizes (semelhante ao Matlab); (ii) `matplotlib` [2] para exibir gráficos referentes aos dados em duas dimensões; e (iii) `scipy` [4], para utilizar funções de otimização. Todas as bibliotecas podem ser instaladas através do gerenciador de pacotes e dependências padrão do Python, o `pip`, através do comando:

```
$ pip install numpy matplotlib scipy
```

O código fonte desta tarefa é publico [2] e possui a licensa MIT. Ele foi submetido junto com este relatório. Para testá-lo basta executar os dois comandos abaixo, pois cada exercício foi implementado em um arquivo diferente.

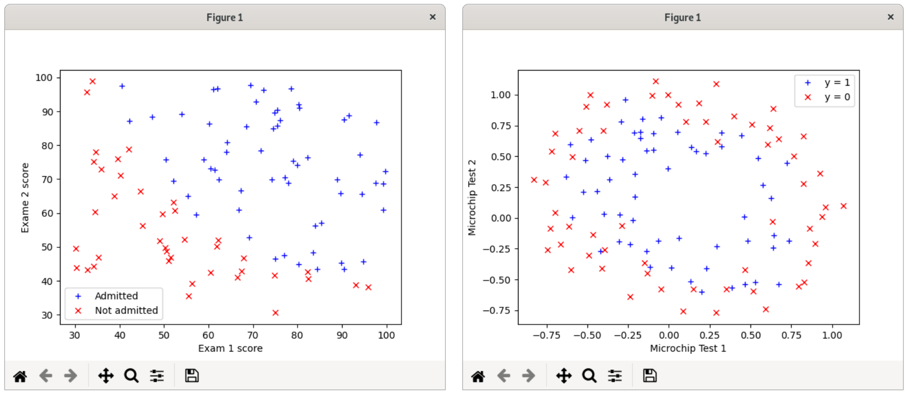
```
$ python3 ex2.py
$ python3 ex2_reg.py
```

3 Resolução do trabalho

Cada dimensão de cada problema é representada como uma entrada do percéptron, como exemplificado na Figura 2, onde três entradas e seus respectivos pesos

[1] Trabalho proposto em `https://www.coursera.org/learn/machine-learning`

[2] Código fonte disponível em `https://github.com/fredericoschardong/programming-exercise-2-logistic-regression-university-of-stanford/`.



(a) Dados de entrada do exercício 1. (b) Dados de entrada do exercício 2.

Figure 1: Dados de entrada e suas classificações.

(representados por w ou θ) são passados ao percéptron. Este, por sua vez, realiza uma soma balanceada das entradas, considerando seus respectivos pesos $\sum_{i=1}^{n} x_i * w_i$. Como os exercícios deste trabalho são de classificação (resultados discretos), e a saída do percéptron é contínua, o resultado precisa ser discretizado. Conforme o código fornecido em Matlab, a discretização dos resultados é dada por $h_\theta(x^{(i)}) \geq 0.5$, sendo $h_\theta(x^{(i)})$ definido na próxima subseção.
.

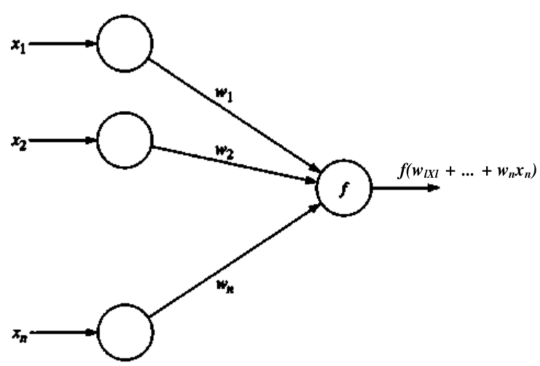


Figure 2: Estrutura e funcionamento de um percéptron [1].

O problema em questão consiste em encontrar os melhores pesos para cada entrada. O fase de treinamento envolve uma função de custo/erro que retorna um valor numérico indicando quão distante dos dados reais são os valores atuais de θ. A fase de treinamento se encerra quando: (i) o erro convergir para determinado valor; ou (ii) quando um número máximo de iterações para obter os melhores pesos for atingido. Novas instâncias de dados podem ser submetidas ao percéptron treinado, que será capaz de prever o resultado do caso fornecido.

É importante observar que apesar do número de dimensões de um problema ser igual ao número de entradas do percéptron, o resultado *i.e.* dimensão dos dados que desejamos chegar através do ajuste dos pesos do percéptron, não é utilizado como uma das entradas do percéptron. A dimensão do resultado é utilizada para guiar a função de *gradiente*, responsável por encontrar os melhores valores de θ, através da minimização da função de custo. No lugar desta dimensão é utilizado um valor constante (nesta implementação 1).

3.1 Regressão Logística Linear

As equações utilizadas para calcular o valor das hipóteses, custo e gradiente fornecidas pelo enunciado do trabalho são mostras nas Equações 1 a 4.

$$g(z) = \frac{1}{1+e^{-z}} \tag{1}$$

$$h_\theta(x^{(i)}) = g(\theta^T x^{(i)}) \tag{2}$$

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}[-y^{(i)}\log_e(h_\theta(x^{(i)})) - (1-y^{(i)})\log_e(1-h_\theta(x^{(i)}))] \tag{3}$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} \tag{4}$$

Onde $g(z)$ é a função sigmoide; $h_\theta(x^{(i)})$ é a função que retorna uma **hipótese de resultado**, *i.e.* um valor hipotético, para uma instância do conjunto de treinamento, representado por $\underset{1\times n}{x^{(i)}}$; onde i é o $i$ésimo dado de um conjunto com total de m dados; e $\underset{1\times 1}{y^{(i)}}$ é o **resultado correto** do $i$ésimo dado; $J(\theta)$ é a função de custo a ser minimizada; θ é uma matriz $\underset{1\times n}{\theta}$ onde $n = 3$ é a quantidade de dimensões do primeiro exercício e θ_j representa os valores de θ na linha 1 e coluna j.

Estas equações foram implementadas no arquivo `ex2.py`. Mais especificamente: a Equação 1 é implementada pela função `sigmoid`, na linha 19; a Equação 2 é implementada pela função `hypothesis` na linha 25; a Equação 3 é implementada pela função `compute_cost` na linha 29; e a Equação 4 é implementada na função `gradient` na linha 38.

O enunciado do trabalho sugere encontrar os melhores valores de θ utilizando a função `fminunc`. Entretanto, a função `fminunc` é exclusiva do Matlab e não possui equivalente no Python. Tentei utilizar a função de otimização `minimize` da biblioteca *scipy*, porém os valores de θ encontrados não produzem um valor de erro adequado, como pode ser visto nas linhas 89 a 99. Consequentemente, o primeiro exercício foi resolvido utilizando a técnica de *gradiente descendente em batch* para encontrar os melhores pesos para cada dimensão do problema. A

Equação 4 foi então modificada para utilizar a técnica de *gradiente descendente em batch* e é representada pela Equação 5 (implementada nas linhas 104 a 120).

$$\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \tag{5}$$

Onde $\alpha = 0.00417$ é a taxa de aprendizado encontrada para produzir o custo 0.203 e a probabilidade de aprovação de 0.776 de uma hipótese de teste com as notas 45 e 85. Foram necessárias 400000 iterações para produzir estes resultados. A flutuação do erro nas iterações pode ser observado na Figura 3.

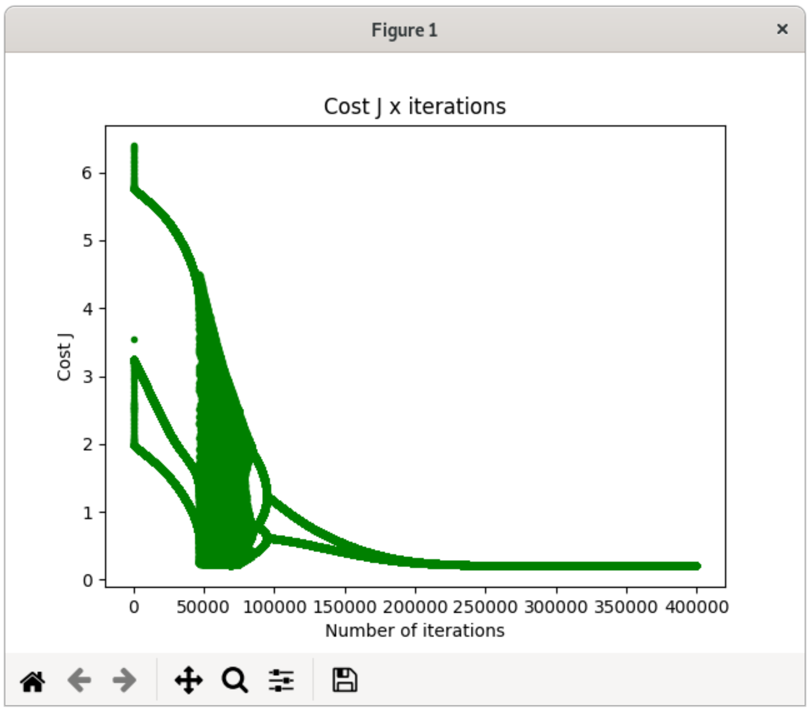


Figure 3: Erro/custo em função do número de iterações.

Por fim, a acurácia das previsões feitas pelo percéptron treinado foi de 89%, que é calculada na linha 133.

3.2 Regressão Logística Não Linear

Como pode ser visto na Figura 1b, a separação entre as duas classes não é linear. Desta forma, o enunciado sugere a normalização dos dados de entrada através da transformação das duas dimensões de entrada (x_1 e x_2) em 28 dimensões (1, x_1, x_2, x_1^2, $x_1 x_2$, x_2^2, x_1^3, ..., $x_1 x_2^5$, x_2^6). A função de custo para regressão

Acurácia		λ	
		0	0.5
Iterações	100	75.42%	74.58%
	1000	83.05%	74.58%

Table 1: Iterações, λ e acurácia resultante.

logistica normalizada, bem como a de gradiente são dadas pelas Equações 6 e 7.

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}[-y^{(i)}\log_e(h_\theta(x^{(i)})) - (1-y^{(i)})\log_e(1-h_\theta(x^{(i)}))] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2 \quad (6)$$

$$\begin{aligned} \frac{\partial J(\theta)}{\partial \theta_j} &= \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} && \text{for } j = 0 \\ \frac{\partial J(\theta)}{\partial \theta_j} &= \left(\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}\right) + \frac{\lambda}{m}\theta_j && \text{for } j \geq 1 \end{aligned} \quad (7)$$

Onde λ é o parâmetro de regularização para prevenir o sobreajuste (*overfitting*) do modelo em relação aos dados de treinamento.

A nova função de custo é implementada como a função `cost_function_reg`, na linha 72 do arquivo `ex2_reg.py`, e a nova função de gradiente é implemtada como a função `gradient` na linha 92. Por fim, como sugerido no enunciado do trabalho, diferentes λ foram utilizados para treinar e testar o modelo. Foram utilizados os valores de λ de 0 e 0.5 e para cada um destes foram testadas 500 e 1000 iterações no método de gradiente descendente, todas as quatro configurações utilizando α de 0.1. O limite de decisão não linear para estas diferentes configurações são mostrados na Figura 4, e a acurácia na Tabela 1

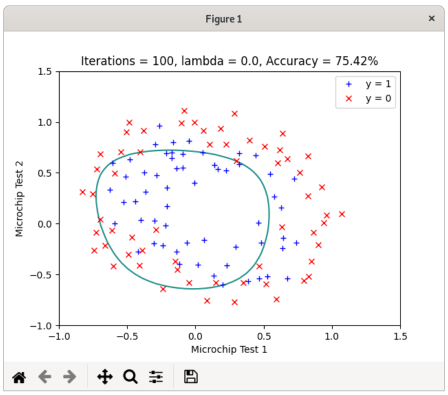
References

[1] Kishan Mehrotra, Chilukuri K Mohan, and Sanjay Ranka. *Elements of artificial neural networks*. MIT press, 1997.

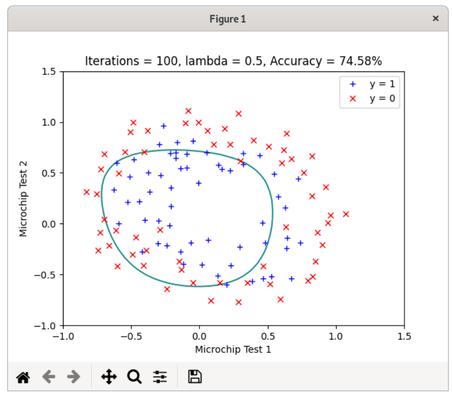
[2] John D Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in science & engineering* 9.3 (2007), pp. 90–95.

[3] Stéfan van der Walt, S Chris Colbert, and Gael Varoquaux. "The NumPy array: a structure for efficient numerical computation". In: *Computing in Science & Engineering* 13.2 (2011), pp. 22–30.

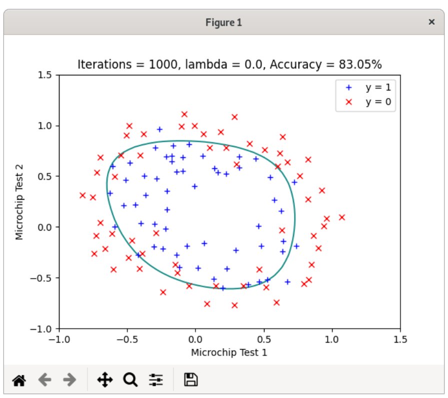
[4] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: `https://doi.org/10.1038/s41592-019-0686-2`.



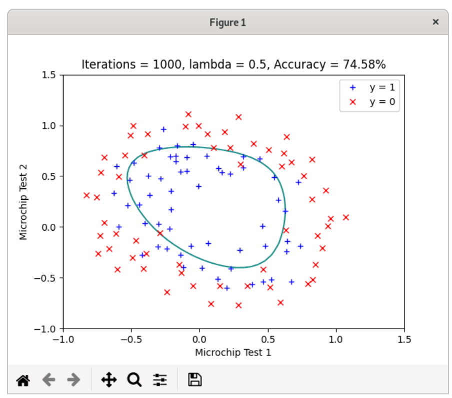
(a) 100 iterações e $\lambda = 0$.



(b) 100 iterações e $\lambda = 0.5$.



(c) 1000 iterações e $\lambda = 0$.



(d) 1000 iterações e $\lambda = 0.5$.

Figure 4: Dados de entrada e suas classificações.