Trabalho 1: Regressão Linear

Frederico Schardong

1 Enunciado do trabalho

O enunciado do trabalho[1] propõem a resolução de dois exercícios de regressão linear: univariável e multivariável, ambos utilizando um único percéptron. O primeiro consiste na previsão do lucro da abertura de um restaurante *food truck*, considerando o lucro de restaurantes em diferentes cidades como uma função do lucro em relação ao tamanho da população. São fornecidos 97 instâncias para realizar o treinamento do percéptron como pode ser visto na Figura 1.

O segundo exercício, referente a regressão linear multivariável, utiliza dados com três dimensões referente ao preço de casas. São considerados o tamanho da casa, o número de quartos e o preço que foram vendidas. São fornecidas 47 instâncias para treinamento do percéptron e posteriormente realizar a previsão do preço de uma casa com base no tamanho e quantidade de quartos fornecidos.

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); e (ii) e `matplotlib` [2] para exibir gráficos referentes aos dados em duas dimensões, detalhados abaixo. Ambas as bibliotecas podem ser instaladas através do gerenciador de pacotes e dependências padrão do Python, o `pip`, através do comando:

```
$ pip install numpy matplotlib
```

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

```
$ python3 ex1.py
$ python3 ex1_multi.py
```

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

[2] Código fonte disponível em `https://github.com/fredericoschardong/programming-exercise-1-linear-regression-university-of-stanford/`.

Figure 1: Dados de treinamento do exercício 1: regressão linear univariável.

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 (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 este é um problema de regressão, e não de classificação, a saída do percéptron **não** é enviada para uma função de ativação.

O problema em questão consiste em encontrar os melhores pesos para cada entrada. Quando os pesos convergirem para um determinado valor, ou 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 do 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

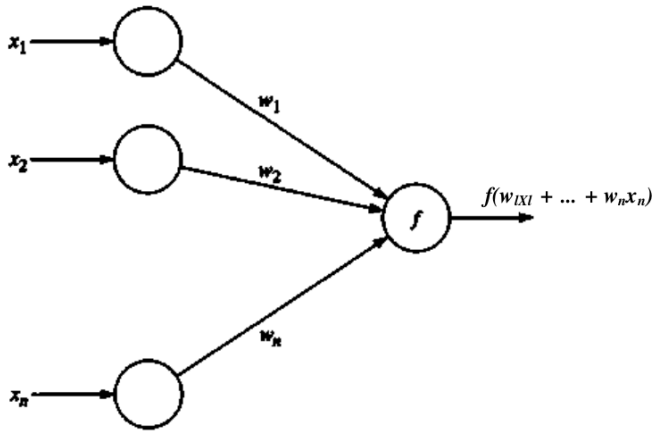

Figure 2: Estrutura e funcionamento de um percéptron [1].

gradiente descentende (através da minimização da função de custo), no lugar desta dimensão é utilizado um valor constante (nesta implementação 1).

Ambos os exercícios foram resolvidos utilizando a técnica de *gradiente descendente em batch* para encontrar os melhores pesos para cada dimensão do problema, de acordo com as Equações 1, 2 e 3.

$$\theta_{j+1} = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \tag{1}$$

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \tag{2}$$

$$h_\theta(x^{(i)}) = \theta^T x^{(i)} \tag{3}$$

Onde $\theta_j \mid j \in \mathbb{N}_0$ representa os valores de θ a cada iteração j; θ é uma matriz $\underset{1\times n}{\theta}$ onde n é a quantidade de dimensões do problema (2 dimensões no primeiro exercício e 3 no segundo); α é a taxa de aprendizado; $J(\theta)$ é a função de custo de mínimos quadráticos a ser minimizada; $h_\theta(x^{(i)})$ 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.

A Equação 1 é implementada na função `gradient_descent`, na linha 5 do arquivo `ex1.py`. A Equação 2 foi implementada como uma subfunção da `gradient_descent`, chamada `compute_cost` na linha 8 do arquivo `ex1.py`. Por fim, a Equação 3 foi implementada em apenas uma linha, como parte da subfunção `compute_cost`, na linha 17. Apesar da regresão multivariável estar codificada no arquivo `ex1_multi.py`, este arquivo inclui e utiliza as funções descritas

acima.

Em ambos os exercícios do trabalho (univariável e multivariável), a taxa de aprendizada sugerida pelo enunciado do trabalho é $\alpha = 0.01$ e inicialmente ($j = 0$) os elementos de θ possuem valor 0. No primeiro exercício a quantidade de iterações sugerida é $\theta_j \mid j \in \mathbb{N}_0 \wedge j < 1500$, a quantidade de dimensões do problema é $n = 2$ (população da cidade; lucro) e o conjunto de treinamento tem tamanho $m = 97$. Já no segundo exercício $\theta_j \mid j \in \mathbb{N}_0 \wedge j < 400$, o problema possui $n = 3$ dimensões (área da casa; número de quartos; preço) e $m = 47$.

3.1 Regressão Linear Univariável

As 1500 iterações sugeridas são suficientes para atingir a menor taxa de erro, como pode ser visto na Figura 3, cuja simulação teve 3000 iterações pra mostrar que mais iterações não mudam significativamente o valor do erro. Para analisar o impacto que o número de iterações tem no resultado, a regressão foi executada utilizando 100, 1500 e 3000 iterações, como pode ser visto na Figura 4. É possível verificar que com apenas 100 iterações o resultado previsto está distante das 1500 e 3000 iterações, já estas duas possuem resultados próximos, o que é esperado dado que o erro entre estas configurações é muito próximo.

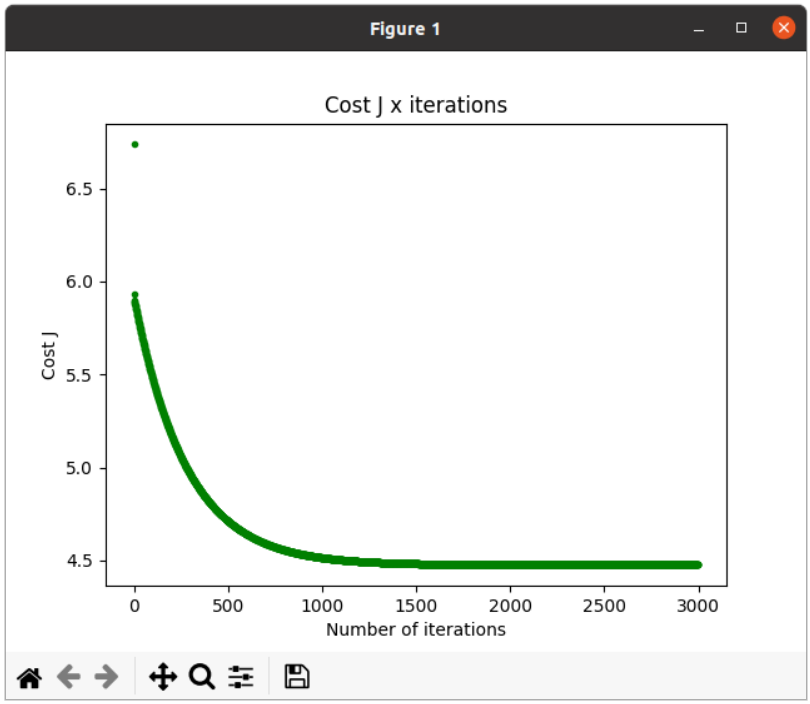

Figure 3: Erro descrescente em função do número de iterações.

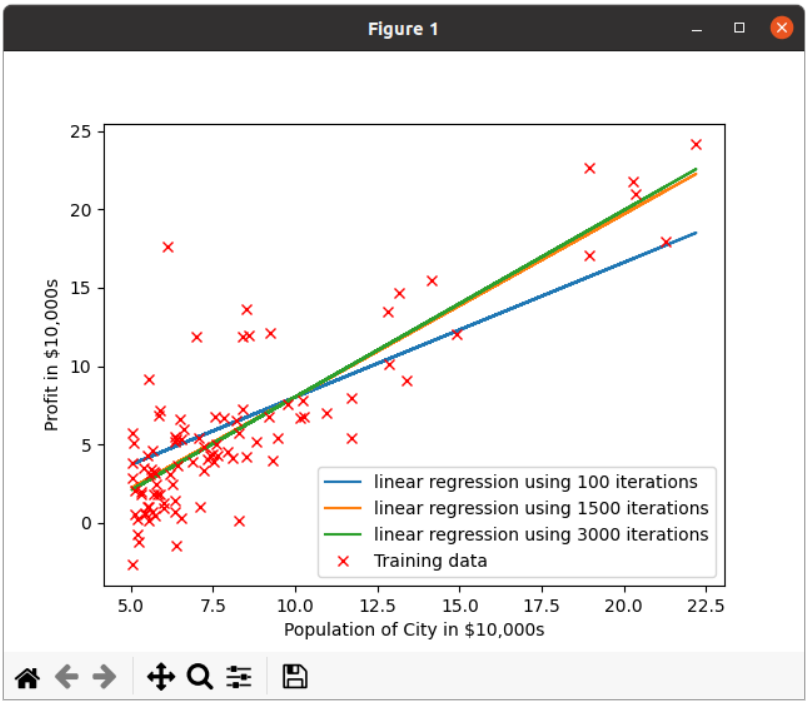

Figure 4: Dados de treinamento e as três regressões lineares.

As previsões de lucro solicitadas foram para cidades com população de 35 e 70 mil habitantes. As previsões para estas populações, considerando as três execuções com iterações diferentes, está resumido na Tabela 3.1. Como existe grande quantidade de treinamento de dados próximos a 70 mil habitantes, todas as três execuções preveram valores próximos dos dados de treinamento. Já para o caso desconhecido, longe dos dados de treinamento, de 30 mil habitantes (os mais próximos são os dados de cidades com 50 mil habitantes), as três previsões são consideravelmente distantes: 1 ordem de magnitude para 100 iterações em relação aos outros; e com 1500 iterações foi 50% maior que 3000 iterações.

Iterações	θ		Predição	
	0	1	35k	70k
100	-0.57655623	0.8595815	24319.79	54405.14
1500	-3.63029144	1.1663623	4519.76	45342.45
3000	-3.87805118	1.1912525	2913.32	44607.16

Table 1: Iterações, θ e previsão de valores para consultas 35k e 70k

3.2 Regressão Linear Multivariável

O enunciado sugere 400 iterações para este exercício. Como pode ser visto na Figura 5, 400 iterações são suficientes para estabilizar a função de custo.

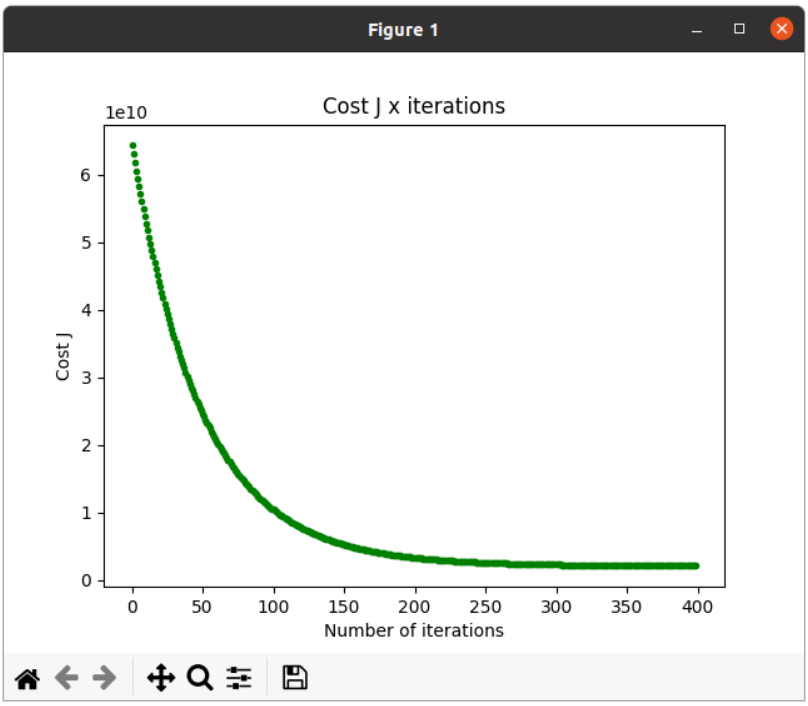

Figure 5: Erro descrescente em função do número de iterações.

O enunciado do exercício então propõem que o problema também pode ser resolvido através da fórmula analítica, que retorna os valores ótimos de θ para os dados de treinamento fornecidos. Ela é encontrada de acordo com a Equação 4, onde $\underset{m\times n}{X}$ representa os dados de treinamento e $\underset{m\times 1}{Y}$ o resultado correto dos dados de treinamento.

$$\theta = (X^T X)^{-1}.X^T Y \tag{4}$$

Por fim, o exercício pede que seja submetido às duas funções (gradiente descendente e fórmula analítica) de regressão já treinadas os dados 1650 (área) e 3 (número de quartos). Os resultados retornados, bem como os três valores de θ são reportados na Tabela 3.2. Como a fórmula analítica é a solução exata (ótima) do problema, é possível verificar que utilizando o método de gradiente descendente com 400 iterações o resultado obtido fica próximo do resultado ótimo (diferença de $\pm 1\%$).

Método	Iterações	Theta			Predição
		0	1	2	1650, 3
Gradiente Descendente	400	334302.06	99411.44	3267.01	289221.54
Fórmula Analítica	-	89597.90	139.21	-8738.01	293081.46

Table 2: Iterações, θ e previsão de valores para uma consulta com 1650 e 3.

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.