Trabalho 1: Regressão Linear

Thiago Dong Chen
RA 187560
thiagodchen@gmail.com

Naomi Takemoto
RA 184849
naomitkm1@gmail.com

I. Introdução

Atribuir preço justo a um produto não é uma tarefa simples se considerarmos todos os atributos que podem influenciar em seu preço. No comércio de diamantes o problema de atribuição do valor também ocorre, frequentemente os vendedores tabelam os preços dos diamantes de maneira superfaturada, com interesse de lucrar mais sobre o cliente. Com um conjunto de dados grande e uma ferramenta de *Machine Learning* é possível gerar um modelo que prediz o preço dos diamantes a partir de suas características. Assim, o modelo pode ajudar o consumidor a ter noção do preço do diamante. Dessa forma, podemos diminuir o conflito de interesse entre o comprador e vendedor de diamantes.

O objetivo deste trabalho é estudar a aplicabilidade do método de regressão linear para estimar o preço dessas gemas. Para tanto, foi utilizada uma base de dados contendo informações acerca de 45849 amostras, com características expressas em 9 atributos como comprimento, largura, profundidade, peso, corte, cor, claridade, etc. Estas 4 últimas são consideradas as mais importantes na determinação do preço [1]. Existem várias formas de se aplicar o método de regressão linear, alguns deles iterativos, como *Batch Gradient Descent*, e suas variações *Stochastic Gradient Descent* e *Mini Batch Gradient Descent*. O problema da regressão foi tratado também com o método dos mínimos quadrados através da resolução das equações normais.

II. Teoria

Existem várias maneiras de se tentar descrever o que é um algoritmo de *machine learning*, uma das mais conhecidas foi proposta por Tom Mitchell em 1997:

> Um algoritmo é dito de aprendizagem automática se ele é capaz de aprender a partir do acúmulo de experiência (E), com respeito a alguma classe de atividades (T) e uma medida de performance (P) se sua performance na tarefa T, medida por P, melhora com experiência E. [2].

Dentre as diferentes classes de tarefas a serem tratadas, estão os problemas de classificação, regressão, tradução, transcrição, etc. A predição de preços de diamantes é um problema de regressão, já que o interesse é encontrar uma função f, tal que y = f(x) + ϵ, onde x é usualmente um dado de múltiplas *features* e f(x) descreve a parte de y que é dependente de x ϵ é a parte não dependente. O interesse nesta tarefa é estimar a função f(x), uma possível abordagem seria supor f(x) uma relação linear. A ideia da regressão linear, como o nome sugere, é encontrar um conjunto de parâmetros que definem a melhor reta que descreve a relação entre x (dado de entrada com nx parâmetros) e y, valor real associado a x. A noção de "melhor" vem da ideia de que uma reta ótima minimiza uma determinada função de custo, que no contexto da aprendizagem de máquina é também a medida de performance P. A métrica mais comum para a medida de performance para esse tipo de regressão é conhecida como (*mean squared error*), fórmula a seguir:

$$J(\theta_0, \theta_1, ..., \theta_{nx}) = \frac{1}{2m}\sum_{i}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 \quad (1)$$

$$h_\theta = \theta_0 x_0 + \theta_1 x_1 + ... + \theta_{nx} x_n x \quad (2)$$

III. Soluções Propostas

Para predizer os preços dos diamantes é proposto, nesse trabalho, o uso do método estatístico de regressão linear [3]. Encontrar a melhor reta que se ajusta via força bruta não é viável computacionalmente, pois existem infinitas retas possíveis. Assim, para atualizar os parâmetros da equação da regressão, é proposto o uso do método iterativo *Gradient Descent* [3], que usa como base o cálculo das derivadas parciais, que busca a convergência para um mínimo local.

O algoritmo de *Gradient Descent* possui variantes [4] [9]:

- ***Batch Gradient Descent (GD)*** O *Batch Gradient Descent* computa o gradiente usando todo o conjunto de dados. Ela é ótima para casos aonde a superfície de convergência é convexa. Nesses casos, ela converge diretamente para o mínimo local.

  repeat

  $$\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i}^{m}(h_\theta(x^{(i)}) - y^{(i)}).x_j^{(i)} \quad (3)$$

  for j := 0...n
  until convergence
- ***Stochastic Gradient Descent (SDG)*** O SDG computa o gradiente usando apenas uma amostra. Normalmente o SDG funciona melhor do que o *Batch Gradient Descent* para casos não convexos, que possuem muitos locais de máximos e mínimos. Uma outra vantagem do SGD é computacionalmente mais rápido para conjunto de dados grande, pois um conjunto grande não pode ser ocupado na RAM, fazendo que a vetorização dos dados seja menos eficiente.

repeat

for i := 1...m **do**

$$\theta_j := \theta_j - \alpha \sum_{i}^{m} (h_\theta(x^{(i)}) - y^{(i)}).x_j^{(i)} \quad (4)$$

for j := 0...n
end for
until convergence

- ***Mini-Batch Gradient Descent (MBGD)*** O *Mini-Batch Gradient Descent* não utiliza todo o conjunto de dados a cada iteração para atualização de parâmetros, também não utiliza apenas uma amostra. O *Mini-Batch* utiliza uma quantidade amostras definida pelo programador. A vantagem de utilizar esse algoritmo é que ela permite a performance intermediária entre o *Batch Gradient Descent* e o SGD. Além disso, permite sair dos mínimos locais e dando outra chance de achar um mínimo melhor do que a anterior.

repeat

for i := 1...m, b **do**

$$\theta_j := \theta_j - \alpha \frac{1}{b} \sum_{i}^{i+b-1} (h_\theta(x^{(i)}) - y^{(i)}).x_j^{(i)} \quad (5)$$

for j := 0...n
end for
until convergence

Além dos métodos iterativos, a regressão linear também permite ser resolvida de maneira analítica, utilizando a equação normal [7] [8]. A tabela I compara a equação normal com o *gradient descent*.

Table I
COMPARAÇÃO DA EQUAÇÃO NORMAL COM O GRADIENT DESCENT

Gradient Descent	***Equação Normal***
Precisa escolher a *learning rate*	Não precisa de *learning rate*
Precisa de muitas iterações	Não precisa de iteração
$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$
Funciona bem para n grande	Lento para n grande

O que faz o algoritmo da equação normal ser custoso computacionalmente é a necessidade de inverter a matriz ($\mathcal{O}(n^3)$). Mas para um conjunto de dados pequeno, a equação normal (6) é uma opção.

$$\theta = (X^T X)^{-1} X^T y \quad (6)$$

Um dos maiores desafios em *Machine Learning* é fazer um algoritmo que não seja bom somente em ajustar no conjunto de treinamento, mas também seja bom em prever em dados novos (conjunto de teste). O fenômeno do modelo se ajustar somente ao conjunto de treinamento é chamado de *overfitting*. Existe várias estratégias para diminuir *overfitting*. Essas estratégias são conhecidas como regularização [6]. Nesse trabalho, somaremos o erro quadrático médio dos parâmetros na função de custo, para tornar o modelo mais generalista.

IV. EXPERIMENTOS E DISCUSSÕES

Divisão do conjunto de dados

Quando se pretende abordar um problema com o uso de técnicas de *machine learning*, alguns cuidados quanto ao tratamento dos dados são necessários. O primeiro passo é dividir o *dataset* disponível tem três subconjuntos, treinamento, validação e teste.

O dado fornecido já está dividido em treinamento e teste [10]. Para ter um conjunto de dados para escolher o melhor modelo, o conjunto de treinamento foi dividido em treinamento (80%) e validação (20%), utilizando a ferramenta *sklearn.model_selection.train_test_split*.

Experimento 1: comparando algoritmos

No primeiro experimento foram comparados as técnicas GD, SGD, MBGD e as equações normais, mantendo o *learning rate* em 0.000000001 e o número de atualizações em 1000. Foram tomadas então medidas de tempo de treinamento e de custo.

Figure 1. Treinamento Gradient Descent - custo calculado em relação aos m exemplos

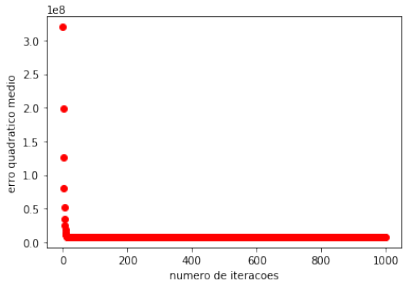
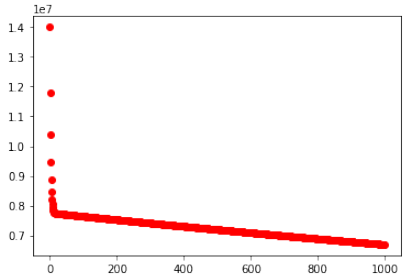
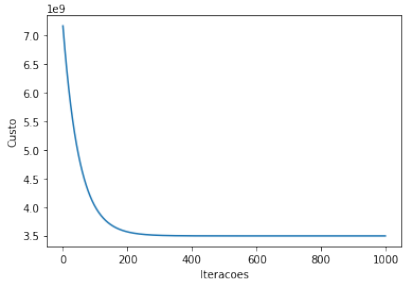


Figure 2. Treinamento Stochastic Gradient Descent, custo calculado em relação aos m exemplos



De acordo com a tabela, é possível perceber que o método com o menor custo e o menor erro foi a equação normal, no caso do problema da predição de preços de diamantes, esse resultado era esperado uma vez que o número de *features* do dado de entrada era relativamente pequeno, fazendo com que a inversão da matriz $(X^T X)$ fosse pouco custosa. O segundo

Figure 3. Treinamento Stochastic Gradient Descent - custo calculado em relação aos m exemplos



menor custo ocorreu para o modelo SDG implementado pela dupla. No entanto, o tempo de execução desta aplicação foi de 836s, o maior dentre os métodos testados, isso pode ter sido consequência do fato de que a implementação não ter tirou proveito de técnicas de vetorização, como ocorreu por exemplo na implementação do GD. O terceiro menor custo ocorreu para o SGD da biblioteca *scikit learn*, notando-se que a aplicação executou em 0.02s. Essa diferença de tempo é uma decorrência do fato de que a biblioteca implementa uma série de otimizações,uma delas é o uso da linguagem Cython, que busca oferecer uma performance de tempo semelhante à linguagem C. [5] Dentre as técnicas de descida de gradiente implementadas neste estudo, aquela que atingiu melhor desempenho (menor custo) no conjunto de validação foi o MBGD, com um tamanho de *mini batch* igual a 64, uma explicação possível para esse resultado em comparação ao SGD é que a quantidade de dados analisados antes de se gerar uma atualização foi maior, o que pode gerar um valor de gradiente mais próximo do gradiente conjunto como um todo e menos influenciado por eventuais "pontos fora da curva". Como o esperado, apesar de ter uma tendência de diminuição do erro, o GD ainda apresenta um alto custo.

Experimento 2: regularização

Para estudar o efeito da regularização, foram adicionados uma penalização nos parâmetros de acordo com a seguinte equação:

$$\theta_j := \theta_j(1 - \alpha\frac{\lambda}{m}) - \alpha\frac{1}{m}\sum_{i}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2.x_j^{(i)} \quad (7)$$

Para cada um dos $\theta_j, j \neq 0$ Os resultados obtidos estão na tabela III. Utilizando-se um parâmetro de regularização $\lambda = 1000$, pode-se perceber que houve um aumento do erro, comparando-se com a implementação sem o termo de regularização para ajustar os parâmetros. Esse aumento é proporcional ao aumento de λ. A regressão linear busca acomodar os dados utilizando uma função muito simples, com baixa capacidade de *overfitting*, nesse sentido utilizar técnicas para evitar este problema não se fazem críticas. Em modelos mais complexos, como redes neurais, no entanto, que são capazes de aprender funções extremamente complexas, usar regularização é essencial para preservar a habilidade de generalização do modelo.

Experimento 3: o maior learning rate

No experimento 3, testou-se o maior *learning rate* que poderia ser utilizado sem que algoritmo divergisse ou que ocorresse *overflow*. Com isso foi possível perceber a vantagem do SGD em relação aos demais algoritmos implementados, com um *learning rate* mais alto, o modelo conseguiu um ajuste melhor no conjunto de validação com o mesmo número de iterações (1000) dos experimentos anteriores. De acordo com a tabela do experimento 3, pode-se perceber que algoritmo que exeige o maior *learning rate* é o *Batch Gradient Descent*, qualquer ordem de grandeza maior que 10^-8 resultará em divergência. Neste experimento percebeu-se que, uma vez escolhido um *learnng rate* que garanta a convergência, quanto maior o valor desse hiperparâmetro mais rápida é a queda do erro.

Table II
EXPERIMENTO 1: ALGORITMOS COM ERRO RESPECTIVO NO CONJUNTO DE VALIDAÇÃO E TEMPO DE TREINAMENTO. PARA OS ALGORITMOS BASEADOS EM *Gradient Descent*, FORAM UTILIZADAS 1000 ITERAÇÕES (ÉPOCAS)

Algoritmo	**Erro no conjunto de validação**	**Tempo de treinamento**
GD	15861300.76	5.01s
SGD	6883087.04	836.40s
MBGD	7658958.30	9.42s
Equação Normal	940316.90	0.02s
SGD scikit learn	7658936.79	0.04s

Table III
EXPERIMENTO 2: UTILIZANDO REGULARIZAÇÃO

Algoritmo	**Erro com regularização**	**Erro sem regularização**
GD	15861315.72	15861240.2
SGD	7880755.40	6883087.0
MBGD	7659115.79	7658958.3
Equação Normal	940316.9	940316.9

Table IV
EXPERIMENTO 3: O MAIOR *learning rate* SUPORTADO

Algoritmo	***learning rate***	**Erro**
GD	0.000000001	15861355.2
SGD	0.0001	6399805.4
MBGD	0.00001	7662774.7

Experimento 4: escolha do melhor modelo

No conjunto de testes, escolhendo a equação normal sem regularização como modelo final, o erro obtido foi de 928633.22. Ao contrário do esperado, houve uma diminuição do erro quadrático comparado com o erro na validação.

V. CONCLUSÃO E TRABALHOS FUTUROS

Ao utilizar os parâmetros obtidos no conjunto de treinamento para predizer os preços dos diamantes, tanto no conjunto de validação quanto no próprio conjunto de treinamento, observou-se que a diferença de preços acontecia na casa dos

milhares. Tais experimentos foram feitos sem técnicas de seleção de *features*. Ao escolher o modelo, com base no menor erro na validação(ver tabelas II, III e IV), houve o aumento do erro ao testar com o conjunto de teste, comparando com o conjunto de validação. Como o erro é alto tanto para o treinamento, validação quanto para teste, podemos inferir que houve *underfitting*, indicando que a regressão linear pode não ser o modelo mais adequado para tratar o problema da predição de preços de diamantes. Além disso, a falta de conhecimentos prévios do funcionamento do mercado de diamantes é uma barreira para a limpeza de dados, o qual é muito importante em *Machine Learning*, em especial para tratar os atributos categóricos (*cut, color, clarity*), utilizamos funções pré-programadas da classe *sklearn.preprocessing.StandardScaler* para transformá-los em números. Observando o erro no conjunto de teste, podemos observar que é possível melhorar o resultado com os conhecimentos adquiridos futuramente na disciplina, como a aplicar redes neurais neste problema de predição de preços. Uma vantagem do uso desta técnica é que essas redes são capazes de aprender funções mais complexas, devido ao maior número de parâmetros e também a utilização de funções de ativação não lineares em várias camadas (*layers*).

REFERENCES

[1] GIA Diamond Grading Scales: The Universal Measure of Quality. Disponível em: <*https://4cs.gia.edu/en-us/blog/gia-diamond-grading-scales/*>. Acesso em: 2 de setembro de 2018.

[2] Mitchell, T. (1997). Machine Learning. McGraw Hill. p. 2. ISBN 978-0-07-042807-2

[3] Aurélien Géron. Hands-On Machine Learning with Scikit-Learn and TensorFlow, 2017.

[4] Batch gradient descent versus stochastic gradient descent. Disponível em: < *https://stats.stackexchange.com/questions/49528/batch-gradient-descent-versus-stochastic-gradient-descent*> Acesso em: 02 de setembro de 2018.

[5] scikit-learn: machine learning in Python. Disponível em: <*http://scikit-learn.org/stable/*>. Acesso em: 02 de setembro de 2018.

[6] Deep Learning. (2016). Ian Goodfellow & Yoshua Bengio & Aaron Courville. MIT Press. Disponível em: <*http://www.deeplearningbook.org*>. Acesso em:03 de setembro.

[7] Linear Regression with Multiple Variables. Disponível em: <*https://www.coursera.org/learn/machine-learning*>, Week 3 & 6. Acesso em: 03 de setembro de 2018.

[8] Deriving the Normal Equation using matrix calculus. Disponível em: <*https://ayearofai.com/rohan-3-deriving-the-normal-equation-using-matrix-calculus-1a1b16f65dda*> . Acesso em: 03 de setembro de 2018.

[9] Avila, S. MC886/MO444 — Aprendizado de Máquina e Reconhecimento de Padrões. <*http://www.ic.unicamp.br/ sandra/teaching/2018-2-mc886-mo444/*>. Regressão Linear. Acesso em: 29 de agosto de 2018.

[10] diamonds-dataset. <*https://www.dropbox.com/s/hm2kim0j9gwr0vk/diamonds-dataset.zip*> Acesso em: 03 de setembro de 2018.