Estudos em Regressão Linear

Matheus Mortatti Diamantino
RA 156740
matheusmortatti@gmail.com

José Renato Vicente
RA 155984
joserenatovi@gmail.com

I. Introdução

Este projeto teve como intuito o estudo prático do método de regressão linear em Machine Learning. Foram utilizado os algoritmos de *Gradient Descent* conhecidos como *Stochastic, Batch, Mini Batch e Equação Normal*, de modo a compara-los em termos de complexidade e acurácia. [1]

Foi feito um estudo de predição de preço de diamantes, utilizando uma base de dados com 54000 exemplos, em que são apresentados seus preços e nove features como tamanho, cor e número de quilates.

II. Atividades

A. Regressão Linear

Regressão Linear é um método muito conhecido de Machine Learning, utilizado para predizer o valor de uma variável dependente baseado em valores de variáveis independentes. Essa regressão é chamada linear porque se considera que a relação da resposta às variáveis é uma função linear de alguns parâmetros. Desta forma, dado um vetor Theta de tamanho igual ao número de features, cujo valor queremos determinar, temos que:

$$PreçoAlvoEsperado = \sum_{i=1}^{m} \theta_i X_i = h_\theta(x) \tag{1}$$

Em que X é um vetor com os valores das features para um dado diamante, cujo preço queremos determinar.Para encontrar esse valor de Theta, utilizaremos alguns algoritmos e compararemos os resultados obtidos com cada um.

Cada algoritmo utilizado é baseado no método de *Descrida de Gradiente (ou Gradient Descent)*. Este é um método utilizado para achar o ponto mínimo de uma função, aproximando gradativamente seu valor até um ponto quando não é possível ser diminuido mais (i.e. a derivada da função neste ponto é zero). Este método consegue apenas achar mínimos locais e, com isso, não é garantido que o resultado obtido é o melhor para o dado problema.

Para medirmos a eficácia do algoritmo, utilizamos uma *Função de Custo* que nos diz o quão perto do resultado desejado estamos, dado um conjunto de dados. Esta função é definida por:

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

Como queremos minimizar a função de custo, queremos que cada passo da nossa descida de gradiente se aproxime mais do mínimo local. Para extrairmos a direção que temos que ir, utilizamos a derivada da função de custo:

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

Logo, para aproximarmos os valores de θ de modo a nos aproximar do mínimo local, utilizamos a fórmula

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

, onde $0 \leq j \leq m$, $x_0 = 1$ e α é o que chamamos de *Learning Rate* que define o quão agressivamente tentaremos nos aproximar do mínimo. Este método de descida de gradiente é chamada de *Batch Gradient Descent*. A seguir, veremos três outras variações deste algoritmo

a) Stochastic Gradient Descent: Neste método, utiliza-se apenas um exemplo de treino para cada passo da descida de gradiente. Deste modo, a equação 4 se transforma em

$$\theta_j := \theta_j - \alpha (h_\theta(x^i) - y^i) x^i \tag{5}$$

Utiliza-se este método quando procura-se rapidez de execução. Contudo, um ponto negativo deste método é que, como usamos como amostra apenas um exemplo de treino, um passo pode nos levar a um custo mais alto. Como o número de iterações é alta, porém, o método converge ao mínimo e mais eficientemente do que o *Batch* até um certo limite.

b) Mini Batch Gradient Descent: Para obtermos um resultado balanceado, utiliza-se uma mistura dos métodos *Batch* e *Stochastic*, em que define-se um tamanho para o lote de exemplos de treino que serão utilizados para atualizar cara θ.

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=k}^{k+b-1} (h_\theta(x^i) - y^i) x^i \tag{6}$$

, onde b é o tamanho do lote, $k = 0, b, 2b, ..., m-1$.

c) Equação Normal: Para a regressão linear, é possível derivar uma fórmula direta para o ponto de mínimo local que desejamos. Para isso, utilizamos manipulações matriciais na forma [2]

$$\theta = (X^T X)^{-1} X^T y \tag{7}$$

, onde X é a matriz de features, y é a matriz dos dados que queremos prever e θ é a matriz dos coeficientes de $h(X)$.

B. Normalização de Features

Como cada feature tem seu valor em uma escala diferente (i.e. algumas estão na ordem de milhares e outras na ordem de centenas), o processo de descida do gradiente poderá acontecer de forma lenta. Isso se dá pelo fato de que a atualização dos θs não ocorrerá de forma uniforme entre as features, já que a distância de um dado θ_i a seu valor esperado pode ser maior do que de outro $\theta_j, j \neq i$. Assim, realizamos o que é chamado de *Normalização de Features*, onde colocamos todas as features x_i em um valor entre $0.5 \leq x_i \leq 0.5$. Isso é feito através da fórmula:

$$x_i = \frac{x_i - \frac{size(x_i)}{2}}{size(x_i)} \tag{8}$$

C. Transformação de Features

Em alguns casos, podemos ter features que não possuem um valor no domínio dos números reais. Por exemplo, uma feature pode representar a cor de um dado elemento. Deste modo, é necessário realizar uma transformação de tais features para o domínio dos reais. Para realizar tal transformação, criamos uma nova feature para cada possível valor da feature que queremos transformar, de modo que, se para o exemplo de dado e_i temos que esta feature possui um valor x_j, a nova feature correspondente a x_j terá o valor 1 e as demais features criadas terão o valor 0. Tomemos como exemplo uma feature de Cor que pode receber os valores *Azul, Amarelo, Vermelho e Verde*. Assim, o resultado da transformação acontecerá da seguinte forma:

Feature x_i	Azul	Amarelo	Vermelho	Verde
Azul	1	0	0	0
Amarelo	0	1	0	0
Vermelho	0	0	1	0
Verde	0	0	0	1

Onde cada coluna representa a nova feature criada e cada linha representa o valor original da feature x_i.

D. Regularização

Regularização é um método utilizado para *evitar* overfitting dos dados. É relizado de forma a penalizar os valores de θ para que estes mantenham valores pequenos, sendo menos propenso ao overfitting. Isto é feito na forma de um novo parâmetro λ que definirá o quanto cada θ será penalizado:

$$\theta_j := \theta_j * (1 - \lambda * \frac{\alpha}{m}) - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i)x^i \tag{9}$$

Este valor deve ser modificado de forma a melhorar o resultado final. Se o valor for muito alto, a influência da regularização pode ser negativa, criando um aumento no erro final. Se muito baixo, não terá o efeito desejado. Vale mencionar que a regularização na equação normal se da pela forma

$$\theta = (X^T X + \lambda I)^{-1} X^T y \tag{10}$$

, onde I é a matriz identidade.

E. Base de Dados

Foi utilizado uma base de dados sobre Diamantes, a qual descreve diferentes aspectos sobre o diamante como cor, claridade e quilate, e apresenta seu preço. O objetivo deste projeto foi de utilizado o método de *Regressão Linear* descrito acima para prever o preço de um dado diamante, dado suas características.

III. Soluções Propostas

A. Tratamento dos Dados

Para obtermos melhores resultados, começamos por realizar o tratamento dos dados. Para isso, foram aplicados os métodos de *Normalização* descritos na seção anterior. Também, foi aplicado uma *Transformação de Features* nas features *Cut, Color e Clarity*, pois seus valores são descritos por uma string.

B. Algoritmos Implementados

Com o objetivo de compara-los, foram implementados os algoritmos *Batch, Mini Batch, Stochastic e Equação Normal* como solução para a Regressão Linear.

a) Algoritmos Iterativos: Algoritmos iterativos para a *Descida de Gradiente* precisam de um critério de parada para sua execução. Tais critérios foram, demonstrados pela Listagem 1:

- Número máximo de iterações.
- Valor máximo da a diferença entre os θs originais e novos.

```
while(iterations < max_iterations and not
    done):
   # Do Gradient Descent
   ...
   iterations = iterations + 1
   if iterations >= max_iterations:
      break
   # If the change in value for new thetas is
       too small, we can stop iterating
   done = True
   for k in range(len(thetas)):
      done = abs(thetas[k] - new_thetas[k]) <
          stopCondition and done
   if done:
      break
```

Listing 1: Criterios de Parada para os algoritmos de Descida do Gradiente

Foi implementado também uma solução para o caso onde a função de custo diverge devido a um α muito alto. Caso o custo corrente seja maior do que o último custo calculado, o valor de α é diminuido por um fator pré definido, de modo que a descida aconteça com menores riscos de divergência, como demonstrado pela Listagen 2.

```
if len(costs) > 0 and cost > costs[-1]:
   learningRate *= alphaFactor
   if retryCount < retryMax:
      retryCount += 1
   else:
      done = true
```

Listing 2: Modo como o fator α é modificado

b) Regularização: Para todas as soluções implementadas, foi utilizado o método de regularização para melhorar os resultados e evitar *Overfitting*. Foi seguido os métodos descritos acima para a implementação tanto na forma da *Equação Normal* quanto para os algoritmos iterativos.

IV. Experimentos e Discussão

Os experimentos foram realizados em uma máquina que possui um processador Intel Core i7-6700HQ com 4 cores rodando a 2.60GHz e 16GB de RAM, com Ubuntu 16.04.

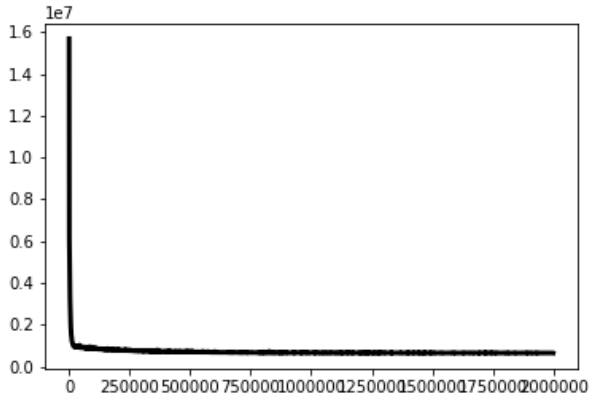
V. Conclusions and Future Work

The main conclusions of the work as well as some future directions for other people interested in continuing this work.

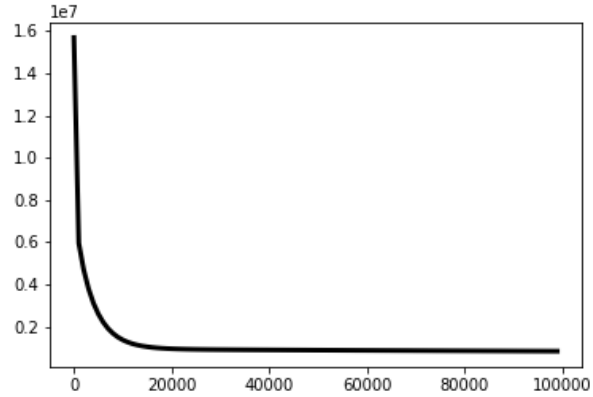
References

[1] S. Avila. [Online]. Available: https://www.ic.unicamp.br/~sandra/

[2] R. Kapur. [Online]. Available: https://ayearofai.com/rohan-3-deriving-the-normal-equation-using-matrix-calculus-1a1b16f65dda



(a) Stochastic with λ = 0.0000005, α = 0.01



(b) Mini Batch with λ = 0.0000005, α = 0.01

Figure 1: Gráficos de Custo x Iterações para cada algoritmo