

Universidade Federal de Mato Grosso do Sul

Campus Ponta Porã
Inteligência Artificial

Trabalho Prático I

Regressão Linear

Aluno: Daniel de Leon Bailo da Silva
Professor: Daniel Matte Freitas

Setembro
2019

Sumário

Resumo	1
1 Introdução	2
2 Métodos	3
2.1 Notação Algébrica	3
2.2 Notação Matricial	3
2.3 Cálculo da Regressão Linear em Python	4
3 Análise dos Resultados	5
3.1 Treinamento utilizando 1000 iterações	11
4 Interpretação dos Resultados	13

Resumo

Este trabalho consiste em implementar a *Regressão Linear com uma variável* utilizando sua forma matricial. O objetivo do mesmo consiste em plotar os valores de $J(\theta)$ para diferentes valores de α , que seria a Taxa de Aprendizagem (Learning Rate) do algoritmo.

Este trabalho está disponibilizado num repositório do *GitHub* para melhor controle do versionamento do programa, e o mesmo se encontra em um *Jupyter Notebook*.

https://github.com/danbailo/T1-Inteligencia_Artificial

1 Introdução

Em estatística ou econometria, regressão linear é uma equação para se estimar a condicional (valor esperado) de uma variável y , dados os valores de algumas outras variáveis x .

A regressão, em geral, tem como objetivo tratar de um valor que não se consegue estimar inicialmente.

A regressão linear é chamada "linear" porque se considera que a relação da resposta às variáveis é uma função linear de alguns parâmetros. Os modelos de regressão que não são uma função linear dos parâmetros se chamam modelos de regressão não-linear. Sendo uma das primeiras formas de análise regressiva a ser estudada rigorosamente, e usada extensamente em aplicações práticas. Isso acontece porque modelos que dependem de forma linear dos seus parâmetros desconhecidos, são mais fáceis de ajustar que os modelos não-lineares aos seus parâmetros, e porque as propriedades estatísticas dos estimadores resultantes são fáceis de determinar.

Modelos de regressão linear são frequentemente ajustados usando a abordagem dos mínimos quadrados, mas que também pode ser montada de outras maneiras, tal como minimizando a "falta de ajuste" em alguma outra norma (com menos desvios absolutos de regressão), ou através da minimização de uma penalização da versão dos mínimos quadrados. Por outro lado, a abordagem de mínimos quadrados pode ser utilizado para ajustar a modelos que não são modelos lineares. Assim, embora os termos "mínimos quadrados" e "modelo linear" estejam intimamente ligados, eles não são sinônimos.

2 Métodos

2.1 Notação Algébrica

Definições:

- $h(x) = \theta_1 x + \theta_0$
- $J(h) = \frac{1}{2m} \sum_{i=0}^m ((h(x_i) - y_i)^2)$

Algoritmo; sendo i a iteração:

- $\theta_1^i = \theta_1^{i-1} - \alpha \frac{\partial}{\partial \theta_1^{i-1}} J \rightarrow \theta_1^{i-1} - \alpha \frac{1}{m} \sum_{i=0}^m ((h(x_i) - y_i)x)$
- $\theta_0^i = \theta_0^{i-1} - \alpha \frac{\partial}{\partial \theta_0^{i-1}} J \rightarrow \theta_0^{i-1} - \alpha \frac{1}{m} \sum_{i=0}^m ((h(x_i) - y_i)1)$

2.2 Notação Matricial

Definições:

Considere x' a entrada, então:

- $X = \begin{bmatrix} 1 & x'_1 \\ 1 & x'_2 \\ 1 & \dots \\ 1 & x'_m \end{bmatrix}, \Theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$
- $H = X \cdot \Theta$
- $E = H - Y$
- $J = \frac{1}{2m} (E^T \cdot E)$

Algoritmo; sendo i a iteração:

- $\Theta_i = \Theta_{i-1} - \frac{\alpha}{m} (X^T \cdot E)$

Sendo que a cada iteração, o valor de H , E e J devem ser atualizados.

2.3 Cálculo da Regressão Linear em Python

A função abaixo realiza o cálculo da Regressão Linear na sua forma matricial, foi adotado essa forma, pois assim é realizado menos operações e sendo assim, também é paralelizável.

Esta função, recebe como parâmetro o número de iterações que irá treinar o algoritmo, o α , que é a taxa de aprendizagem, e os valores de θ inicial, que são os pesos; é importante observar que, nesse caso, a variável "theta" é um vetor coluna 2×1 .

```
def linear_regression(num_it, alpha, theta):
    err = np.zeros(num_it)
    for i in range(num_it):
        h = np.dot(X, theta)
        e = h - y
        J = (np.dot(e.T, e) / (2 * m))
        err[i] = J
        theta = theta - ((alpha / m) * np.dot(X.T, e))
    return err, theta
```

3 Análise dos Resultados

Considerando os dados de teste que foram disponibilizados pelo professor, com a entrada representando o eixo X, e a saída o eixo Y, o algoritmo tem a função de prever o melhor valor para a saída, e para isso, foi utilizado algumas técnicas dispondo de gráficos para selecionar o melhor valor de α dentre o intervalo conhecido para os testes.

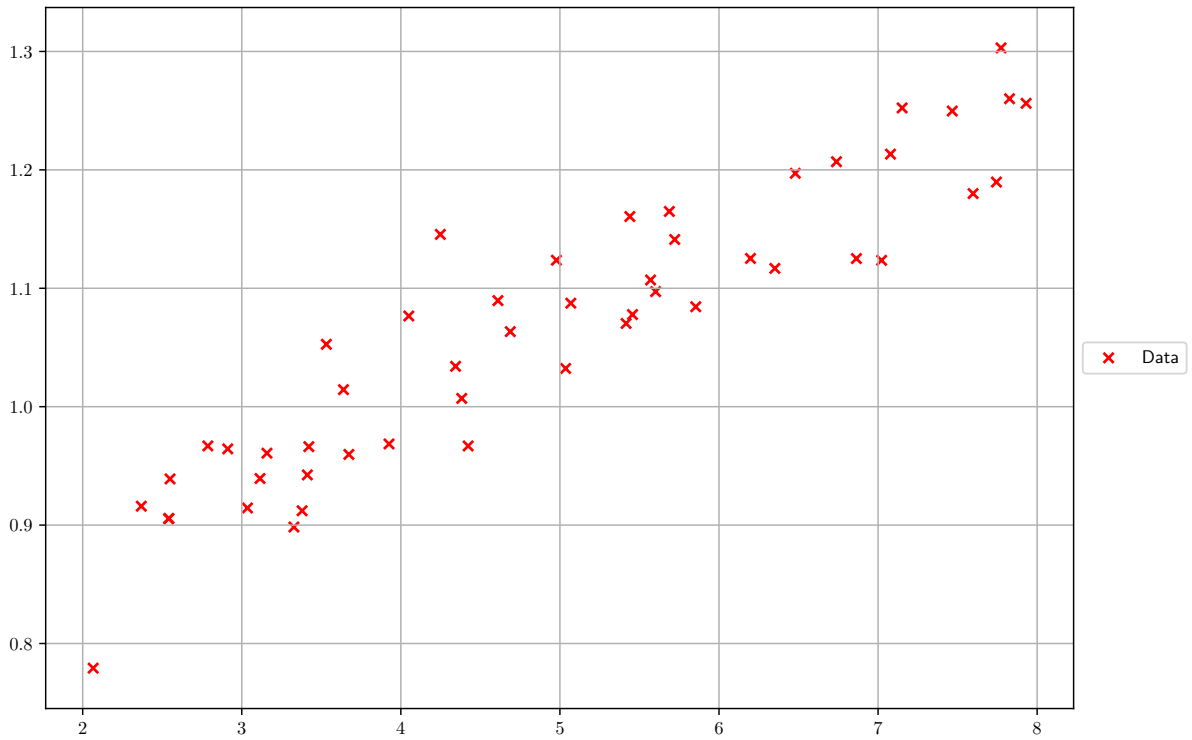


Figura 1: Dados disponibilizados.

Esse intervalo para os valores selecionar os valores de α , foi gerado partindo de uma hipótese de valores desconhecidos para entender como o algoritmo iria se comportar.

Para todos os testes, os pesos (θ) foram inicializados com $(0,0)$ e inicialmente foi-se utilizada 100 iterações para observar o comportamento do algoritmo.

Com isso, o primeiro insight obtido após selecionar um valor para o α , foi de que era necessário diminuir ainda mais seu valor, pois o mesmo estava estourando exponencialmente, ou seja, estava se divergindo da solução, e o que estamos buscando, é um α que se convirja a solução ideal.

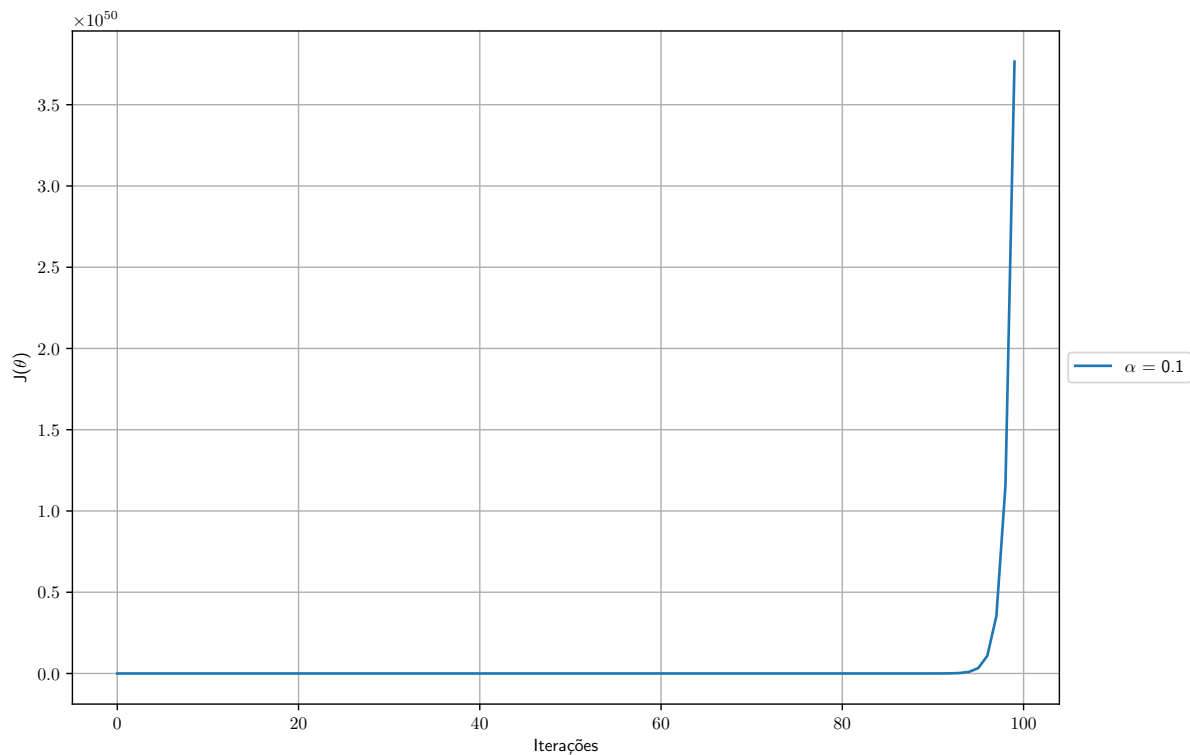


Figura 2: Primeiro insight obtido a partir de um valor de α inicial.

Com os dados dispostos, se realizarmos a aplicação do algoritmo, utilizando o valor de α acima, que foi de 0.1, iremos obter a seguinte predição.

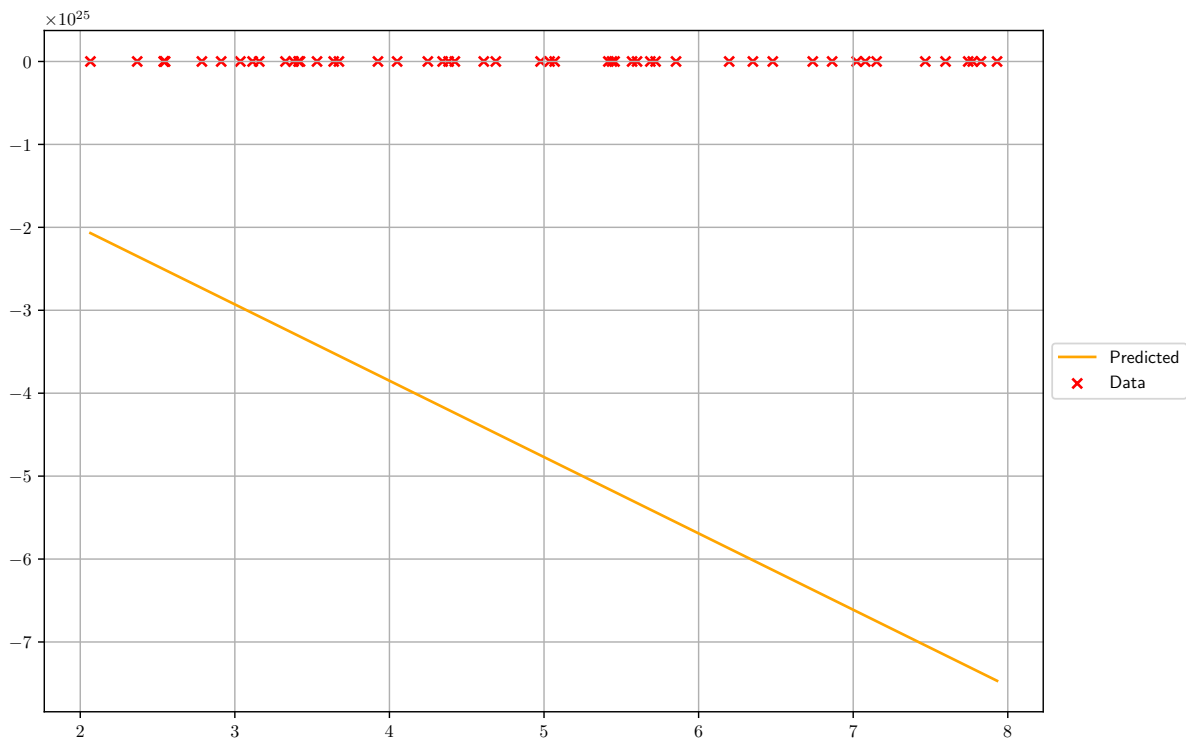


Figura 3: Predição com mau treinamento.

Como podemos ver, numa escala logarítmica, a nossa predição está absurdamente longe dos dados que foram dispostos, aproximadamente 10^{25} se comparado no eixo y.

Então, foi realizado alguns testes para tentar selecionar o melhor valor local para α , logo, é importante notar que, quanto mais próximo de 0, melhor será a solução que o algoritmo irá encontrar.

Neste gráfico, é possível observar que, para alguns valores de α , como o próprio 0.1, que já foi testado anteriormente, e também, 0.08 e 0.09, convergem da solução e "estouram" exponencialmente, portanto esses valores serão descartados para as próximas análises.

É possível também que, para alguns valores temos uma convergência mais rápida, ou seja, esses valores se aproximam mais rápido de 0; isso significa que, na *Descida de Gradiente*, o algoritmo irá realizar um salto bem próximo de uma boa solução (valor próximo de 0), em contrapartida, alguns valores também convergem, mas de forma mais lenta.

Este gráfico representa o que conhecemos por *erro*...

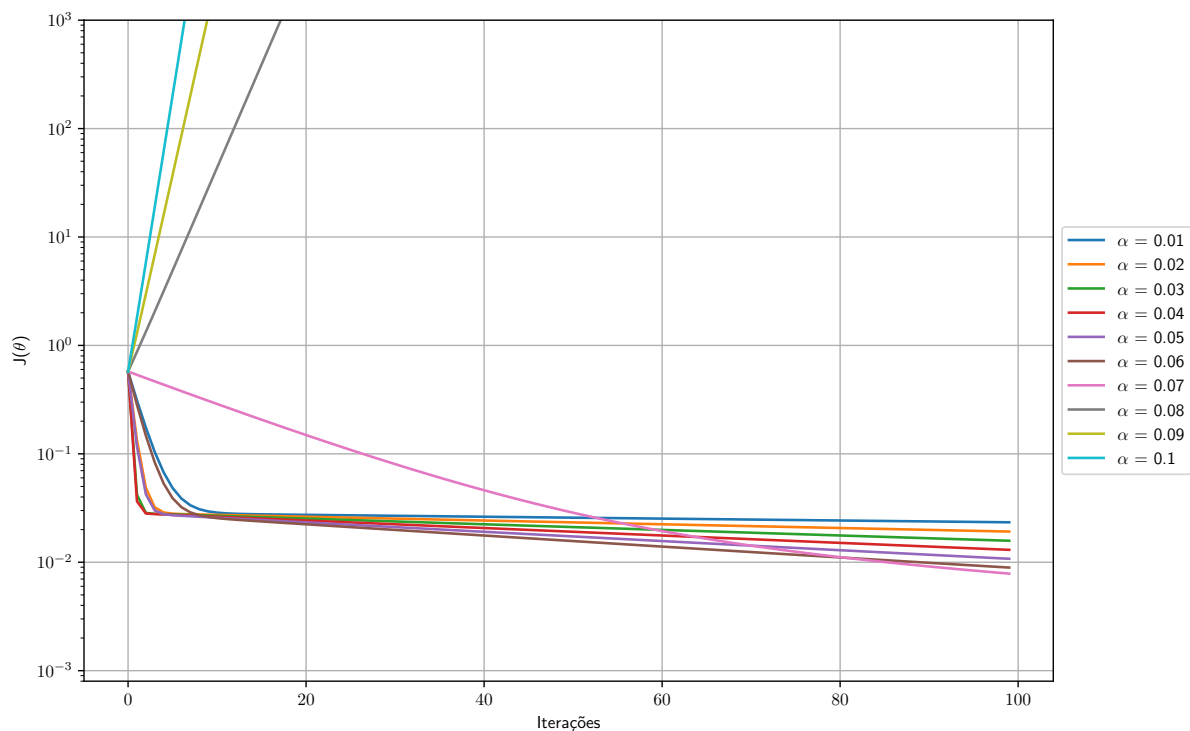


Figura 4: Primeiro insight obtido a partir de um valor de α inicial.

Logo, a seguinte questão é levantada: "Para esses valores dispostos de α , o valor que converge mais rápido, será de fato o melhor valor para utilizar no treinamento do algoritmo?"

No gráfico abaixo, pode-se observar melhor como está atualmente o comportamento do algoritmo para cada valor de α . De primeira impressão, é possível ver que o valor que mais demora a convergir é 0.07, porém, o mesmo é o que está mais próximo do eixo, pode-se afirmar que para 100 iterações, este seria um bom valor.

Entretanto, não é possível saber de fato, qual é o valor que converge mais rápido, pois nessa escala, os valores estão muito próximos um do outro.

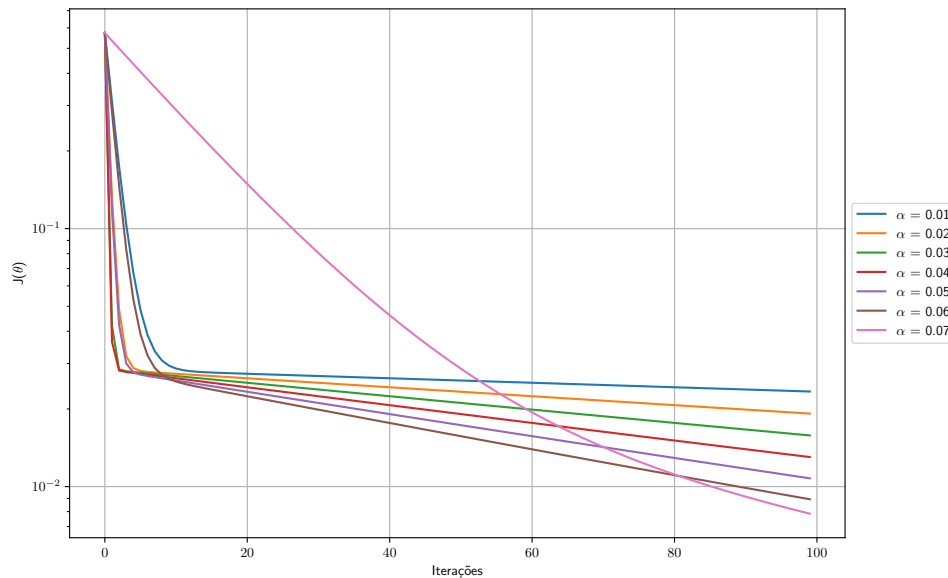


Figura 5: Intervalo de 100 iterações.

Com isso, o gráfico abaixo foi estreito num intervalo menor, relacionado a figura 8. Agora é possível ver com mais clareza que o valor 0.04, referente a linha vermelha, é o valor que converge mais rápido.

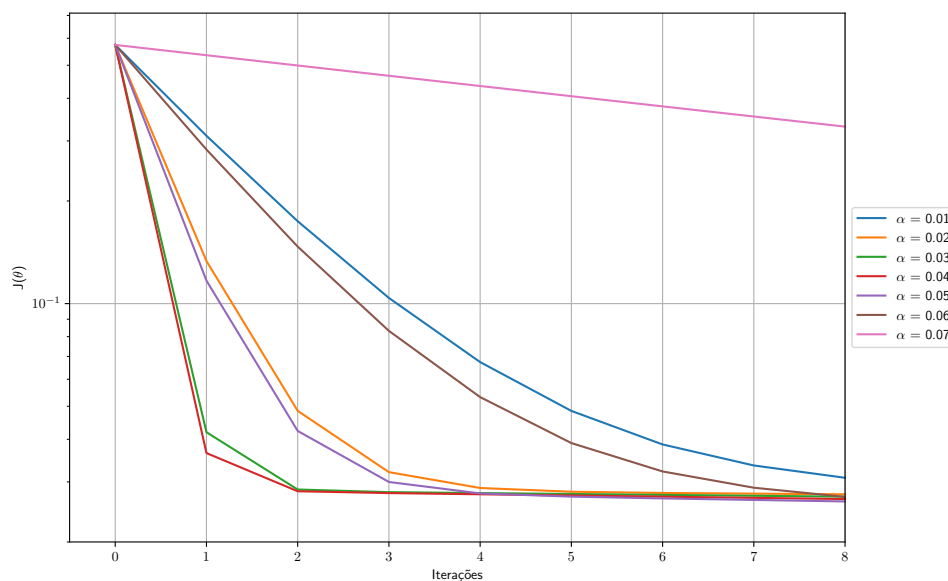


Figura 6: Intervalo estreito mais a esquerda.

Continuando a análise para selecionar o valor de α , agora veremos como está se comportando os valores num intervalo mais estrito ao final do eixo x. Assim como na figura 8, como os valores estão bem diferentes entre si, o comportamento é o mesmo que já foi notado.

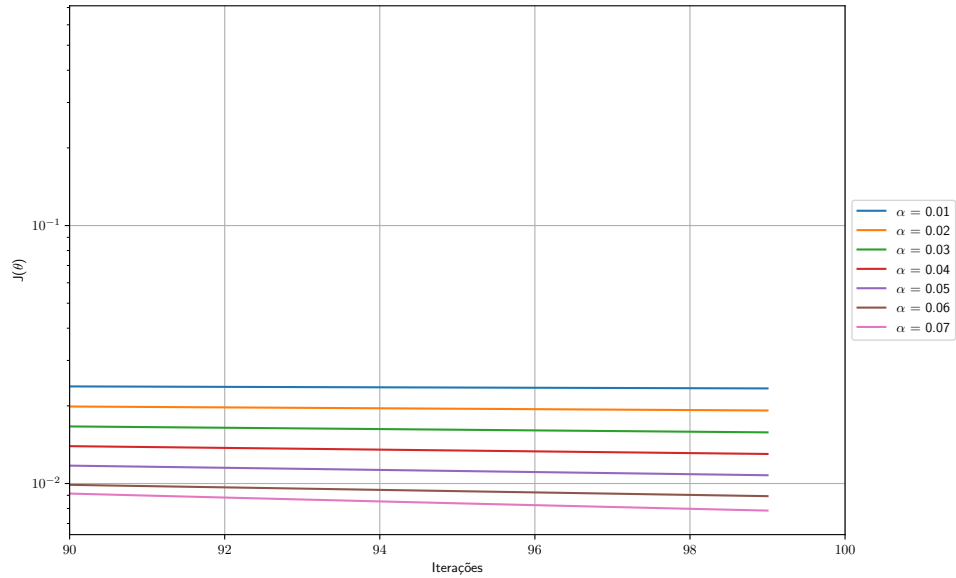


Figura 7: Intervalo estreito mais a direita.

Porém, é importante lembrar que, este treinamento foi utilizando somente **100 iterações**, esse é um numero pequeno de iterações, mesmo sabendo que os dados dispostos não são tão grandes; porém, para obter uma análise mais concreta para selecionar um bom α predizer os valores de y , será realizado outra análise, mas agora utilizando **1000 iterações**, e será observado se o comportamento do modelo continua o mesmo, ou se sofre alguma alteração durante o treinamento.

3.1 Treinamento utilizando 1000 iterações

Ao realizar o treinamento utilizando 1000 iterações para treinar o algoritmo, pode-se ver que alguns valores, para essa quantidade de iterações, temos alguns valores que convergiram a partir das 500 iterações e outros nem com essa quantidade de iterações conseguiram convergir, mesmo que com 100 iterações, pareciam ser bons valores, pelo fato de convergir mais rápido que outros valores.

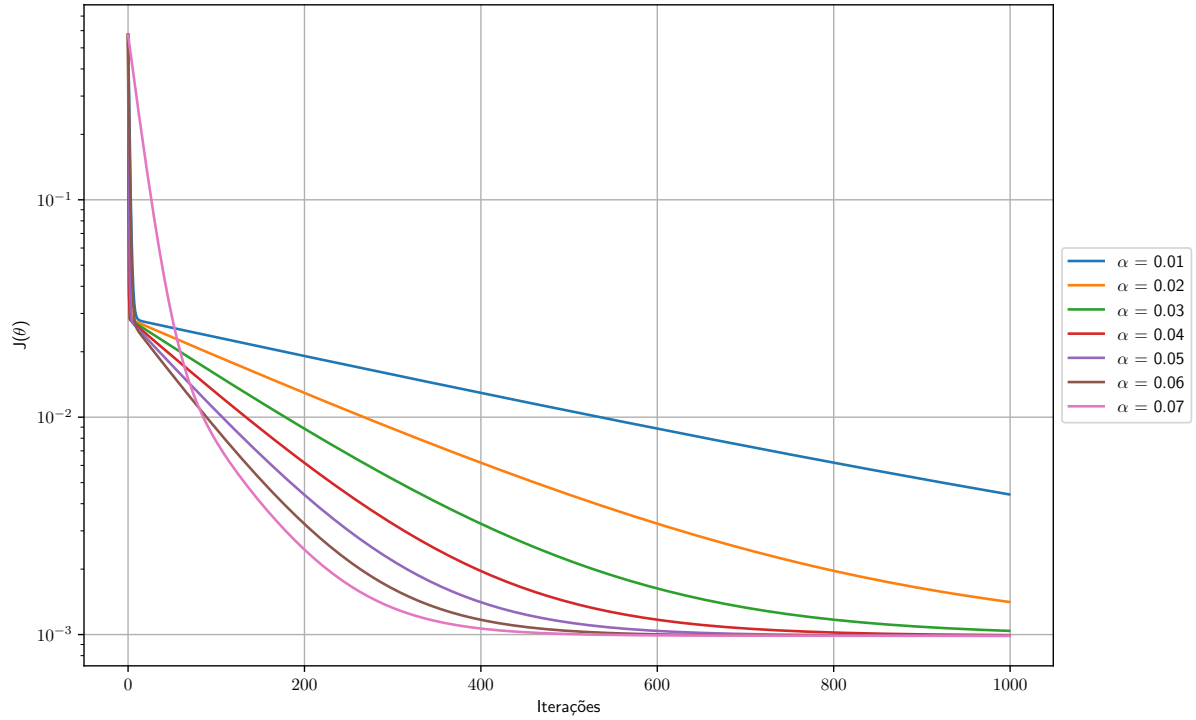


Figura 8: Intervalo de 1000 iterações.

Porém, novamente como agora os valores estão muito próximos um do outro, é necessário ampliar o intervalo ao final do eixo X para saber de fato qual valor está mais abaixo no eixo y. Para isso, foi gerado dois gráficos, a figura 9 e 10, as duas figuras estão estritos no intervalo $[990, 1000]$ para X, e $[10^{-3}, 10^{-1}]$, $[9.8 \times 10^{-4}, 1.06 \times 10^{-3}]$, respectivamente.

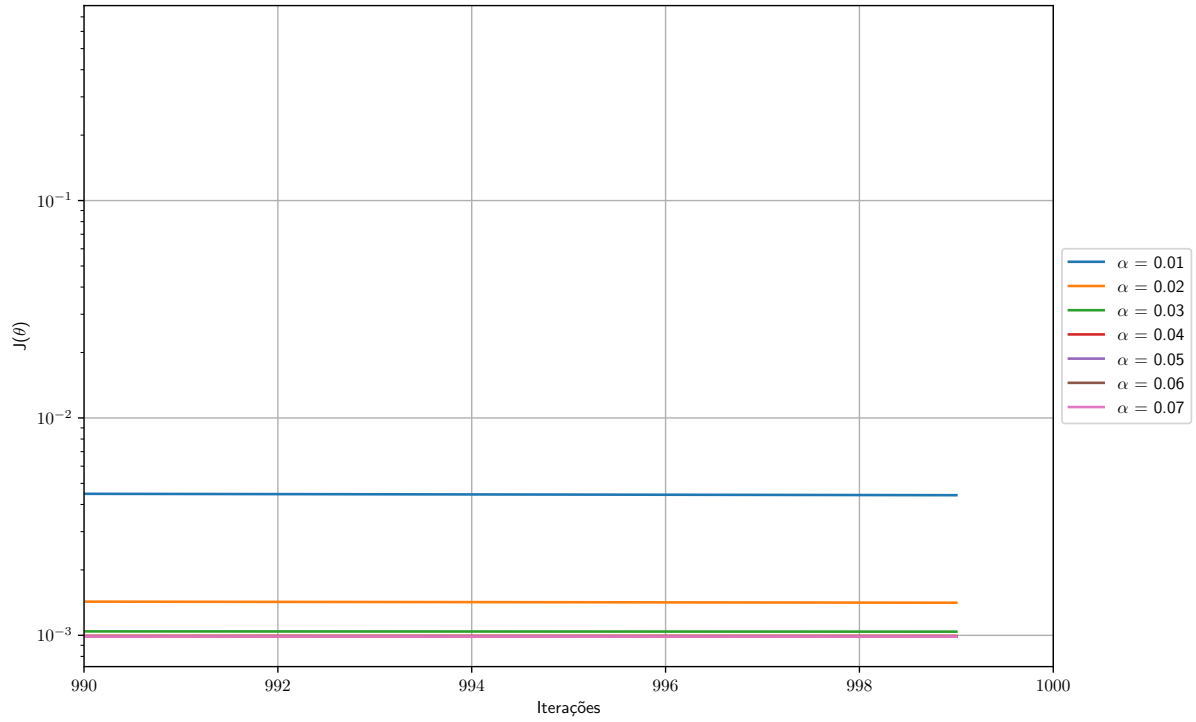


Figura 9: Intervalo estrito mais a direita para 1000 iterações.

No gráfico da figura 9, é impossível notar a presença de mais de uma linha entre as cores *verde* e *rosa*, que representam os valores 0.03 e 0.07, respectivamente. Porém, no gráfico da figura 10, é possível notar que ainda existem mais alguns valores neste intervalo.

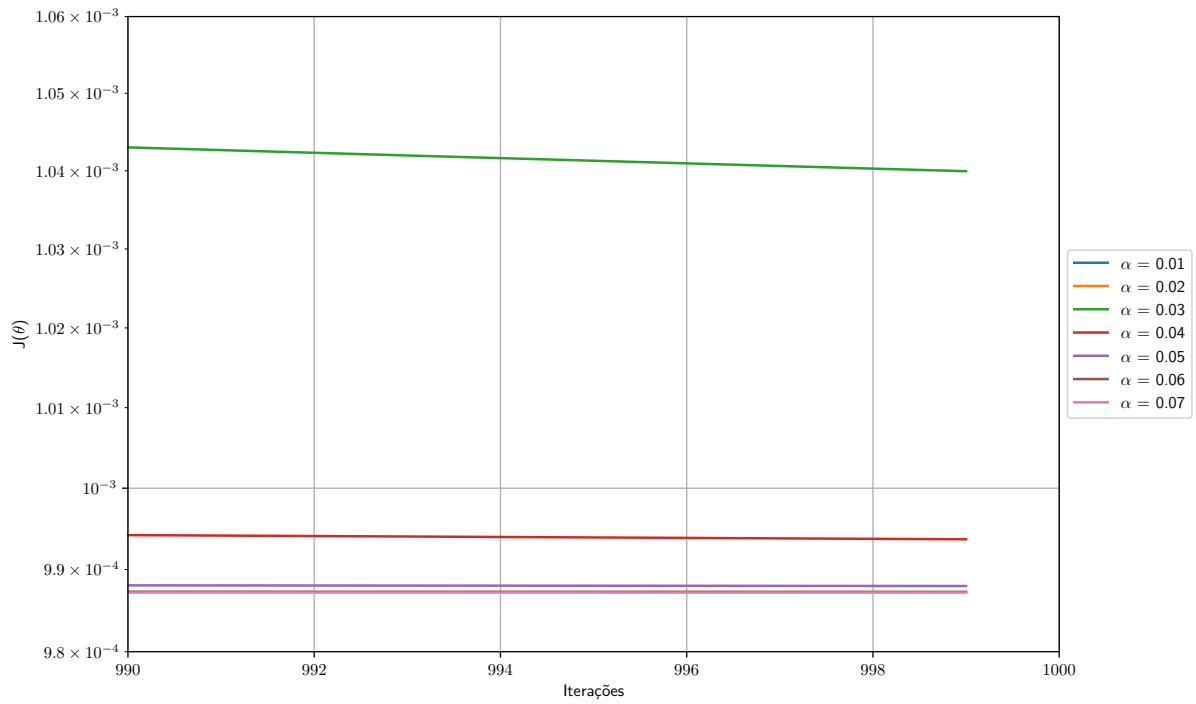


Figura 10: Zoom no intervalo.

4 Interpretação dos Resultados

Após estudarmos o comportamento do algoritmo para os α selecionados, podemos chegar numa conclusão;

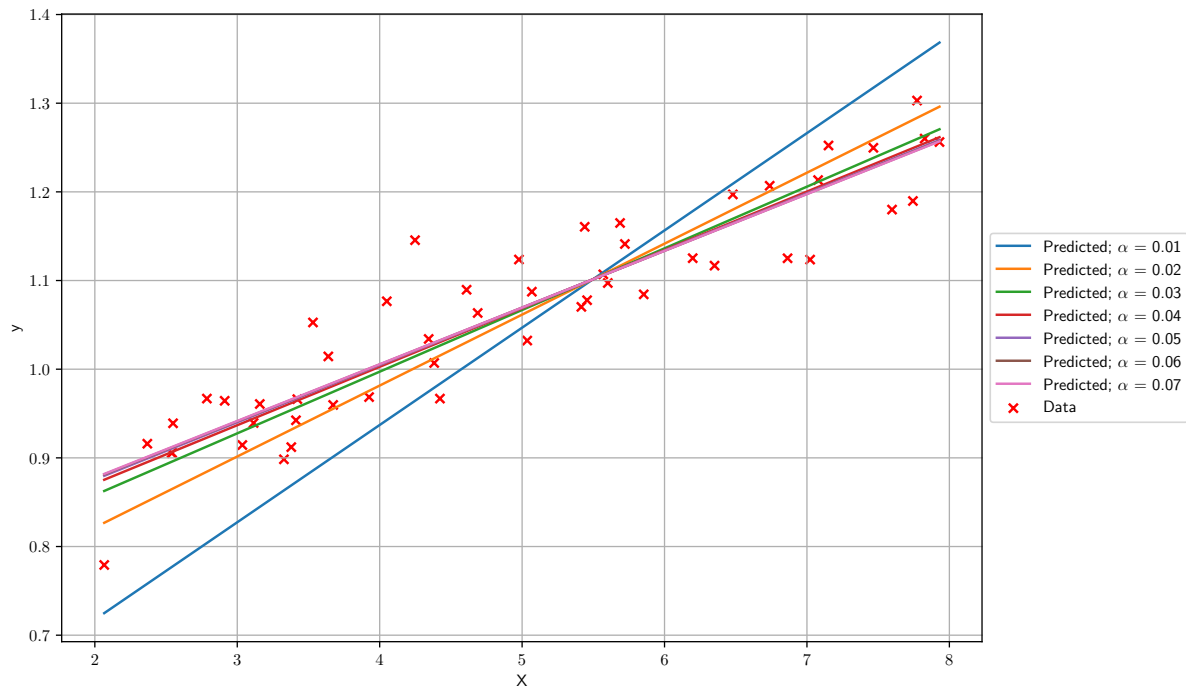


Figura 11: Todas as previsões para os valores de α estudados.

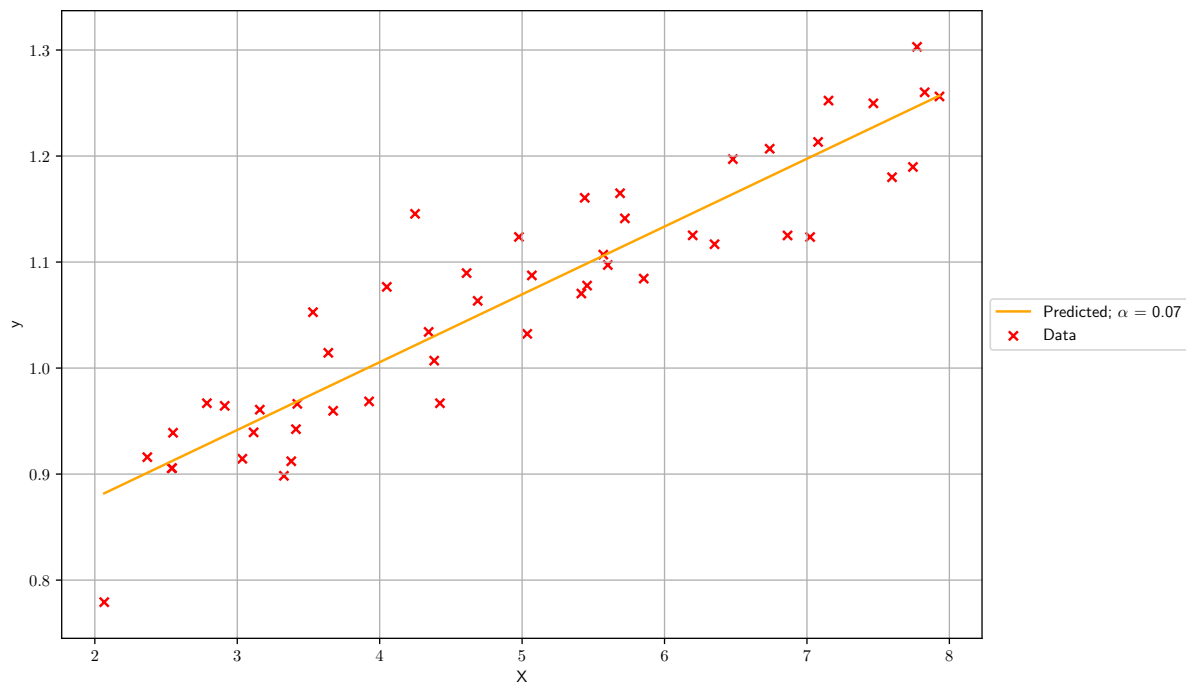


Figura 12: Modelo final.