Universidade Estadual de Campinas

Institute of Mathematics, Statistics and Scientific Computing

(IMECC - Unicamp)

Projeto Computacional 1:

Análise de do comportamento do COVID-19 por ajuste de curvas não-lineares e Classificação de dígitos do MNIST utilizando regressão logística

Aluno : Gabriel Borin Macedo || RA : 197201
Aluno : Matheus Araujo Souza || RA : 184145
Aluno : Raul Augusto Teixeira || RA : 205177

Agosto 2020

1 Resumo

Com a grande disponibilidade de dados que temos atualmente, assim como a relativa popularização de *hardwares* de alto desempenho, contribuíram para o uso de mais técnicas de *Machine Learning* tais como *regressão* e *classificação*. Mesmo que existam técnicas de predição ou classificação de dados com maior precisão (por exemplo a utilização de *Deep learning [2]* nessas tarefas), ainda essa técnicas de *Machine Learning* se demonstram bem eficazes.

Baseado nisso, esse trabalho visa em estudar a técnica de regressão e regressão logística (classificação) utilizando as metodologias do *Machine Learning*. Por fins prático, esse trabalho foi dividido em duas partes.

Na primeira parte desse trabalho, foi utilizado as aproximações polinomial, exponencial e a analítica para criar um modelo que prevê a quantidade de infectados pela doença de acordo com o dia. Pelos resultados, foi observado que a curva polinomial, mais precisamente para o polinômio de grau 10, é a que melhor se aproxima a previsão correta. Além disso, a curva polinomial se aproximou de forma satisfatória

Já na segunda parte, foi criado um modelo de regressão multi-classe para classificar dígitos à mão dos números 1 à 10 e verificar a taxa de acerto do modelo durante a classificação.

Em ambas as partes, também foi estudado o comportamento das funções quando se variava o parâmetro α do gradiente descente (esse valor será descrito com mais detalhes ao longo do relatório). Um outro adendo é que todos os *plots* e resultados serão colocados em uma folha de anexo. No geral, ambos os modelos foram bem satisfatórios e se percebeu que a mudança do valor de α determina a taxa de convergência e até mesmo a taxa de convergência dos modelos. Esses valores de α variam para cada tipo de abordagem e o recomendado é se utilizar valores pequenos e aumentar caso a convergência seja lenta.

Com esses resultados, percebeu-se que essas técnicas de *Machine Learning* ainda são úteis, tanto para as tarefas de regressão e classificação, pois apresentaram um excelente desempenho e ainda são alternativas viáveis para se utilizar nesses problemas.

2 Estrutura matemática

2.1 Regressão não - linear

Esse método visa em aproximar os dados de correlação de elementos através de uma curva não linear e utiliza os dados dos parâmetros do vetor para fazer isso. Usualmente, é definido essa função sendo da seguinte forma

$$h_\theta(x) = \theta_0 + \sum_{i=1}^{n} \theta_i x_i$$

Onde x é o vetor de entrada para o problema com dimensão 1 x m e θ_j com $j \in [0, n]$ são os respectivos pesos que melhor aproximam essa curva linear. Para melhor desempenho computacional, podemos deixar a função $h_\theta(x)$ vetorizada. Isso é feito adotando que

$$\Theta = \begin{vmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{vmatrix}$$

e que

$$X = \begin{vmatrix} 1 \\ x_1 \\ \vdots \\ x_n \end{vmatrix}$$

Com isso, temos que a equação é da forma $h_\theta(x) = \Theta^T X$. Note que podemos também fazer ajuste por curvas que também **não lineares**. Por exemplo, podemos adotar o vetor Θ com a componente $x_2 := (x_2)^2$.

Nesse trabalho, iremos fazer um ajuste de três tipos de curvas. Para todos os casos, iremos adotar um vetor X sendo da seguinte forma

$$X = \begin{vmatrix} 1 \\ x_1 := (x_1)^1 \\ \vdots \\ x_n := (x_n)^n \end{vmatrix}$$

ou seja, iremos fazer a aproximação das curvas utilizando polinômios de grau n. Além disso, foi adotado nesse trabalho que $n = 3, 5, 10$.

2.1.1 Função perda e atualização dos parâmetros θ_j utilizando o gradiente descendente

Para atualizar o valor dos θ_j, é adotado a seguinte função J

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

com isso, aplicando o operador gradiente e visto o resultado durante a aula. Obtemos a uma relação para atualização dos pesos sendo da forma

$$\theta_j := \theta_j - \frac{\alpha}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 x_j \ \forall \ j \geq 0$$

onde α é um parâmetro arbitrário adotado para acelerar o processo do gradiente descendente.

Entretanto, não é recomendando utilizar valores elevados para α, pois o gradiente pode acabar se distanciando dos melhores pesos devido a essa escolha. Por outro lado, valores menores para α deixam o gradiente mais estável para encontrar os valores de θ_j. Porém, com uma taxa de convergência menor.

Dessa forma, a escolha do valor de α é uma parte importante do modelo para encontrar os modelo, podendo gerar melhores ou piores resultados com essa convergência. Nesse trabalho, foram adotado os valores $\alpha = 10, 3, 1, 0.3, 0.1, 0.03, 0.01, 0.003, 0.001$

2.1.2 Normalização dos valores utilizando *scaling*

Um método que é possível adotar para acelerar o método do gradiente descendente é normalizar os dados utilizando o *scaling*. Essa técnica consiste em pegar os valores x_j e restringir eles em uma mesma faixa de valor.

Para cada x_j, teremos que a nova relação será da forma

$$x_j^{(i)} := \frac{x_j^{(i)} - \min(x_j)}{\max(x_j) - \min(x_j)}$$

2.1.3 Aproximando a função por uma exponencial

Queremos estudar o comportamento da função $h_\theta(x) = \theta_0 e^{\theta_1 x}$. Para isso, iremos aproximar a curva aplicando ln na equação.

$$\ln(h_\theta(x)) = \ln(\theta_0 e^{\theta_1 x}) = \ln(\theta_0) + \ln(e^{\theta_1 x}) = \ln(\theta_0) + \theta_1 x \ln(e)$$

com isso, temos a equação $\ln(h_\theta(x)) = \theta_0 + \theta_1 x$. Dessa forma, tomando que $y_\theta(x) = \ln(h_\theta(x))$, temos a equação que será utilizada como base para encontrar os pesos θ_1 e θ_2 sendo

$$y_\theta(x) = \theta_1 + \theta_2 x$$

Note que com isso, temos que refazer duas adaptações bem simples na função de perda e na função atualização dos pesos. Assim, temos a função de perda sendo

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

e a atualização dos pesos sendo

$$\theta_j := \theta_j - \frac{\alpha}{m} \sum_{i=1}^{m} (y_\theta(x^{(i)}) - y^{(i)})^2 x_j \ \forall \ j = 0, 1$$

Outra readaptação que será necessária é atualizar os antigos valores de $y^{(i)}$ para a escala ln. Ou seja, $y^{(i)} := \ln(y^{(i)})$

2.1.4 Utilização de equações normais

Dado um problema que temos m exemplos de treinamento da forma $(x^{(i)}, y^{(i)})$, onde cada vetor de treinamento possui n elementos da forma

$$x^{(i)} = \begin{vmatrix} x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_m^{(i)} \end{vmatrix}$$

sendo o vetor de cada dado de treinamento. Com isso, iremos definir uma matriz denominada de *matriz de design* que é definida da forma

$$X = \begin{vmatrix} -(x^{(1)})^T- \\ -(x^{(2)})^T- \\ \vdots \\ -(x^{(n)})^T- \end{vmatrix} = \begin{vmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \dots & \vdots \\ x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{vmatrix}$$

e com o vetor y

$$y = \begin{vmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{vmatrix}$$

sendo o vetor das respostas. Com isso, queremos encontrar a melhor valor de Θ de tal forma que melhor aproxima a solução $X\Theta \approx y$. Dessa forma, para encontrar o melhor valor de Θ, iremos usar a função de perda sendo

$$J(\Theta) = \|y - X\Theta\|^2 = y^T y - 2\theta^T X^T y + \Theta^T X^T X\Theta$$

Agora, minimizando a função $J(\Theta)$, temos que

$$\frac{\partial J(\Theta)}{\partial \Theta} = 2X^T X\Theta - 2X^T y = 0 \Longrightarrow X^T X\Theta = X^T y$$

sendo a solução sendo $\Theta = X'y$, onde $X' = (X^T X)^{-1} X^T$.

Por fim, iremos melhorar a resolução do sistema fazendo a multiplicação :

$$X^T(X\Theta) = X^T y \Longrightarrow X^T X\overline{\Theta} = X^T y$$

essa equação é denominada como *equação normal* e a solução $\overline{\Theta}$ também é uma solução ótima para o problema e ainda minimiza a função $J(\Theta)$

2.1.5 Utilização da função normal por uma exponencial

Em nosso problema, vamos adotar que $n = 2$ sendo a *matriz de correlação*

$$X = \begin{vmatrix} 1 & x_1^{(1)} \\ 1 & x_1^{(2)} \\ \vdots & \vdots \\ 1 & x_1^{(m)} \end{vmatrix}$$

a matriz y sendo

$$y = \begin{vmatrix} \ln(y^{(1)}) \\ \ln(y^{(2)}) \\ \vdots \\ \ln(y^{(m)}) \end{vmatrix}$$

e a matriz Θ sendo

$$\Theta = \begin{vmatrix} \theta_0 \\ \theta_1 \end{vmatrix}$$

e com essas definições, iremos calcular os valores de θ_0 e θ_1 através da relação $X^T X\overline{\Theta} = X^T y$.

2.2 Regressão logística

Esse tipo de técnica é utilizada quando é necessário classificar objetos em um conjunto de $k-$classes, apenas utilizando uma faixa de valor entre 0 e 1. Um aspecto interessante sobre essa técnica é que toda essa teoria foi baseada utilizando a metodologia da regressão linear e não linear e baseando em alguns teoremas estatísticos. Dessa forma, é definido a função g sendo

$$g(\Theta^T X) = \frac{1}{1 + e^{-(\Theta^T X)}}$$

onde Θ é a matriz dos pesos e X é a matriz de *design* e é da forma $X_{n \text{ x } m}$. Além disso, é definido a função de perda utilizando o teorema do *Maximum likelihood estimation*, temos a expressão

$$P(y = a|X; \theta) = (g(\Theta^T X))^a (1 - g(\Theta^T X))^{1-a}$$

na qual será útil para realizarmos a classificação das multi-classes.

2.2.1 Função de perda e gradiente descendente para multi-classes

Visto em sala de aula, sabemos que a função de perda é definida da seguinte forma

$$J(\Theta) = \frac{-1}{m}[\sum_{i=1}^{m} y^{(i)} \log\left(g(\Theta^T X^{(i)})\right)(1 - y^{(i)}) \log\left(1 - g(\Theta^T X^{(i)})\right)]$$

e a atualização de pesos sendo simultânea para todos os θ_j sendo da forma matricial

$$\Theta := \Theta - \frac{\alpha}{m} X^T (g(X\Theta) - y)$$

e com os valores de $\alpha = 0.01, 0.2, 0.4, 0.8, 1.0, 1.5$.

2.2.2 Predição da classe dos dados

Se tivermos $k-$classes para classificarmos, iremos tomar $k-$classificadores que tentam melhor se aproxima para aquela classe do objeto. Com isso, temos a função para cada classe será da forma

$$h_\theta^{(i)}(X) = P(y = i|X; \theta);\ \forall\ i \in [1, k] \text{ e } i \in N.$$

Além disso, é criado um vetor onde cada índice dele corresponde a classe do objeto classificado. Ou seja, seja o vetor $v_{classe} = [1, 2, 3, \ldots, k]$ e o vetor das probabilidade de cada classificador sendo $v_{prob} = [h_\theta^{(1)}(X), h_\theta^{(2)}(X), h_\theta^{(3)}(X), \ldots, h_\theta^{(k)}(X)]$.

Dessa forma, temos que a classe prevista é da forma $\arg\max(v_{prob})$ que corresponde ao índice do vetor da classe procurada.

3 Resultados Práticos

3.1 Regressão Linear e não Linear

Nessa parte, foi utilizado a parte de regressão para tentar prever o comportamento da COVID-19 através de uma função polinomial e de uma função exponencial. O *dataset* utilizado foi os casos de COVID no Brasil nos dia 25 de fevereiro e que se estendia até 100 dias depois. Para analisar melhor o comportamento, foi variado os valores de α e a aproximação por polinômios distintos para ver qual seria o melhor comportamento da função de predição.

Mantendo $\alpha = 0.5$, obtivemos os seguintes custos finais de $J(\Theta)$ para $n = 3, n = 5$ e $n = 10$, respectivamente: $1.77 * 10^8, 1.69 * 10^8, 7.01 * 10^7$. Assim, o polinômio de ordem 10 forneceu a melhor aproximação em relação aos dados de treinamento.

Analisando os gráficos, vemos que, nos três casos, o custo ao longo das iterações sempre diminui, logo a regressão linear está correta. Além disso, quanto menor o grau do polinômio, pior é a aproximação para a fronteira à esquerda de x dos dados de treinamento.

Portanto, nesse caso, a melhor aproximação para $\alpha = 0.2$ foi utilizar um polinômio de grau igual a 10.

Nesse exercício, aproximamos $\log(y)$ por $\log(h_\theta(x))$. Mantendo $alfa = 0.5$, obtivemos um custo final de $J(\Theta) = 1.2$, o que é um ótimo resultado.

Plotamos dois tipos de gráficos: $\log(y)$ versus $\log(h_\theta(x))$ e y versus $h_\theta(x)$, além do custo $J(\Theta)$ ao longo das iterações. Analisando os gráficos, vemos que a aproximação polinomial foi contra-intuitivamente melhor que a exponencial, apesar de a exponencial ter dado um custo final menor. Ficamos surpresos com o resultado. O gráfico de $J(\Theta)$ é decrescente, o que significa que a implementação está correta.

Para $\alpha = 10$ e $\alpha = 3$, o programa não rodou. O custo estoura e vai a infinito em algumas iterações, o que significa que o método não convergiu.

Para $\alpha = 1$, $\alpha = 0.3$, $\alpha = 0.1$, $\alpha = 0.03$, $\alpha = 0.01$ e $\alpha = 0.003$, o custo final foi basicamente o mesmo: $J(\Theta) = 1.2$. Para $\alpha = 0.001$, o custo final foi de $J(\Theta) = 1.55$.

Analisando os custos finais e os gráficos para cada valor de alfa, todos os valores até $\alpha = 0.01$ aproximaram bem os dados de treinamento até $x = 100$, a partir daí a aproximação desprega dos dados de treinamento e cresce mais do que esses dados. A partir de $\alpha = 0.003$, quando alfa começa a decrescer, a aproximação piora em x mediano, mas melhora em valores maiores de x: $h_\theta(X)$ demora mais para crescer exponencialmente em relação aos dados de treinamento, assim para $x = 134$ a aproximação é um pouco melhor, mas ainda se desprega muito dos dados de treinamento.

Nesta implementação, utilizamos a pseudo-inversa de uma matriz mencionada nas aulas.

A implementação por equações normais devolveu basicamente a mesma aproximação de $\alpha = 1$, $\alpha = 0.3$, $\alpha = 0.1$, $\alpha = 0.03$ e $\alpha = 0.01$ do item anterior, com gráficos similares e custo final exatamente igual. A diferença foi no tempo de execução, que foi levemente menor que 15000 iterações do Gradiente Descendente.

3.2 Classificação multi-classe

Para esta parte, foi utilizado o *dataset MNIST*, que consiste em dígitos feito a mão para o computador reconhecer e classificar dígitos entre 0 à 9. Além disso, os dados já foram transformados em suas formas de vetores para melhor ajuste na regressão logística.

Depois do desenvolvimento de toda a parte estrutural do nosso código, começamos os nossos testes com $\alpha = 0.01$ e 50 iterações, temos como resultado dessa primeira implementação na figura 15, podemos ver que os dados não apresentam no todo uma configuração exata da nossa classificação, olhando para a linha 5 no qual os dados deveriam estar mais acumulados na coluna 4, é pela ordem de inicio das matrizes do *python*, podemos então perceber que pela dispersão ele não seria um α adequado para a nossas verificações. Não vamos entrar na questão do tempo para o processamento dessa iteração, mas ele foi classificado como alto, esse foi um dos motivos pelo qual mudamos o

número de passos até encontrar algum que nos permitisse fazer verificações no α em tempos consideráveis. Para o próximo teste foi escolhido um $\alpha = 0.2$ e 50 iterações, mesmo com o comentário feito acima mantemos o número de iterações até o momento, podemos perceber a melhora na distribuição de resultados como é possível ver na figura 13. Nos próximos testes vamos variar um pouco o número de iterações para encontrar um novo α e assim continuar verificando a distribuição de dados na nossa matriz de confusão. Para o terceiro teste aumentamos o valor de $\alpha = 0.4$ e colocamos 25 iterações, podemos ver os resultados dessa nova configuração na figura 17, observando mais atentamente podemos perceber uma grande semelhança entre esse último teste e o último. Para fim de testar ao extremo as modificações no nosso α, mantemos agora nosso numero de iterações em 10 e aumentamos ele para 0.8 e 1.0 nas figuras 18 e 14, podemos notar uma leve piora na distribuição dos dados para o $\alpha = 0.8$ 0.8 em comparação com o 0.4 o que por sua vez pode ser explicado pela diminuição de iterações mas também vemos que eles melhoram um pouco para o $\alpha = 1.0$. Como último teste aumentamos o α para $\alpha = 1.5$ e 10 iterações, aqui obtemos uma piora expressiva na distribuição de resultados na figura 16 Como foi possível verificar nos dados apresentados com o α inicial pequeno não conseguimos chegar a resultados satisfatórios, logo encontramos uma faixa na qual ele conseguisse descrever melhor e seus resultados ficassem dentro de uma faixa de normalidade, no fim com o grande aumento dele mantendo essa mesma faixa percebemos que ele acaba se desviando dos valores reais, a correção do ultimo efeito pode vir com aumento do número de iterações.

4 Conclusão

Com isso, percebemos que na parte 1 a função polinomial se saiu melhor para aproximar os casos de COVID-19 do que a função exponencial. Isso se deve pois no caso exponencial, encontramos pesos que minimizavam a função $\log(h_\theta(X))$ e não da função $h_\theta(X)$. Principalmente por esse fator, os valores obtidos de Θ não foram os melhores para aproximar a função. Além disso, percebeu-se que polinômios de grau pequeno não geravam uma boa aproximação para o modelo. Porém, para $n = 10$ e $\alpha = 0.2$, a função de aproximação $h_\theta(x)$ obteve resultados excelentes. No gráfico dos resultados, essa argumentação fica bem clara.

Já na segunda parte, Quando analisamos os resultados obtidos, podemos perceber uma melhora no aumento dos α fixando uma certa quantidade de iterações, mas não podemos comparar α de diferentes passos, logo quando fixamos para 10 iterações e observamos as variações dos resultados para $\alpha = 0.8, 1.0, 1.5$. Com isso, podemos notar que aconteceu uma melhora para $\alpha = 1.0$, mas logo depois os dados se dispersaram, principalmente no valor de $\alpha = 1.5$. Nosso modelo conseguiu classificar os dados com êxito e ainda foi possível perceber um valor de barreira para os α para certos números de iterações.

Com esses resultados, podemos concluir que a aplicação de modelos de *machine learning* ainda são úteis em aplicações desses problemas.

References

[1] Normal equations in machine learning. http://mlwiki.org/index.php/Normal_Equation.

[2] Ludovic Arnold, Sébastien Rebecchi, Sylvain Chevallier, and Hélène Paugam-Moisy. An introduction to deep learning. volume 1, pages 477–488, 01 2011.

[3] Joao B. Florindo. Ms960 : Tópicos especiais em processamento de imagens. https://www.youtube.com/playlist?list=PLGwGFVrptiyRmFoDWxruNGgTu2cSPnTLX, 2020.

[4] Aurelien Geron. Hands-on machine learning with scikit-learn and tensorflow: Concepts, tools, and techniques to build intelligent systems, 2017.

[5] Naveen Venkatesan. Using logistic regression to create a binary and multiclass classifier from basics. https://towardsdatascience.com/using-logistic-regression-to-create-a-binary-and-multiclass-classifier-from-basics-26f5e

Anexo

Problema 1 : Aproximação linear e não linear

Regressão polinomial

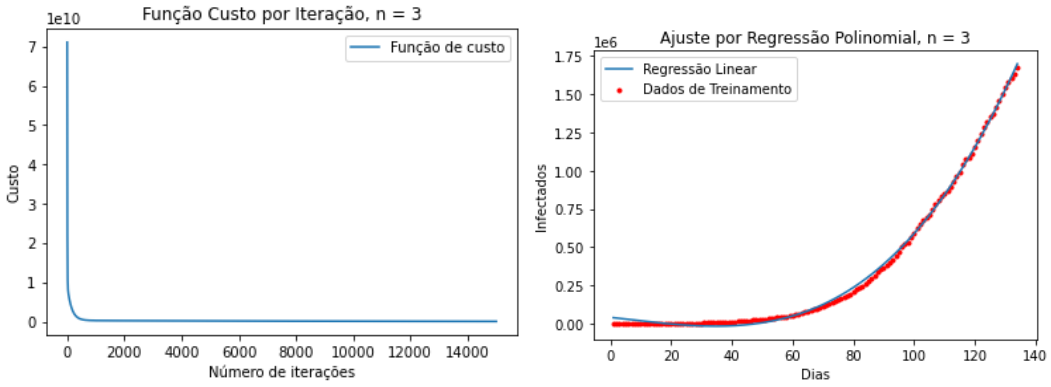


Figure 1: Resultado dos *plots*, respectivamente, das funções de perda e da estimativa da função polinomial $h_\theta(x)$

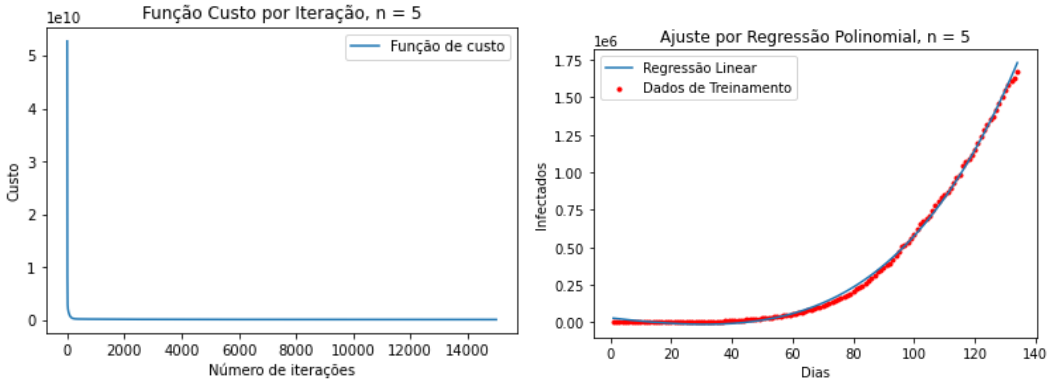


Figure 2: Resultado dos *plots*, respectivamente, das funções de perda e da estimativa da função polinomial $h_\theta(x)$

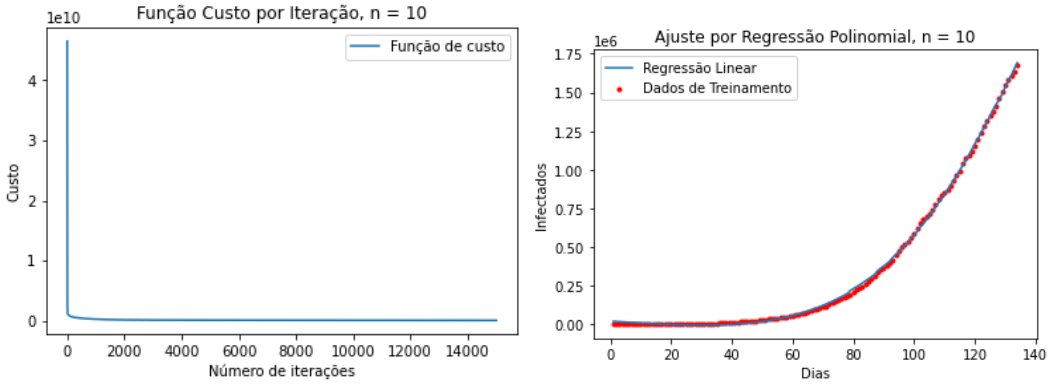


Figure 3: Resultado dos *plots*, respectivamente, das funções de perda e da estimativa da função polinomial $h_\theta(x)$

Ajuste por curvas exponenciais com diferentes valores de α

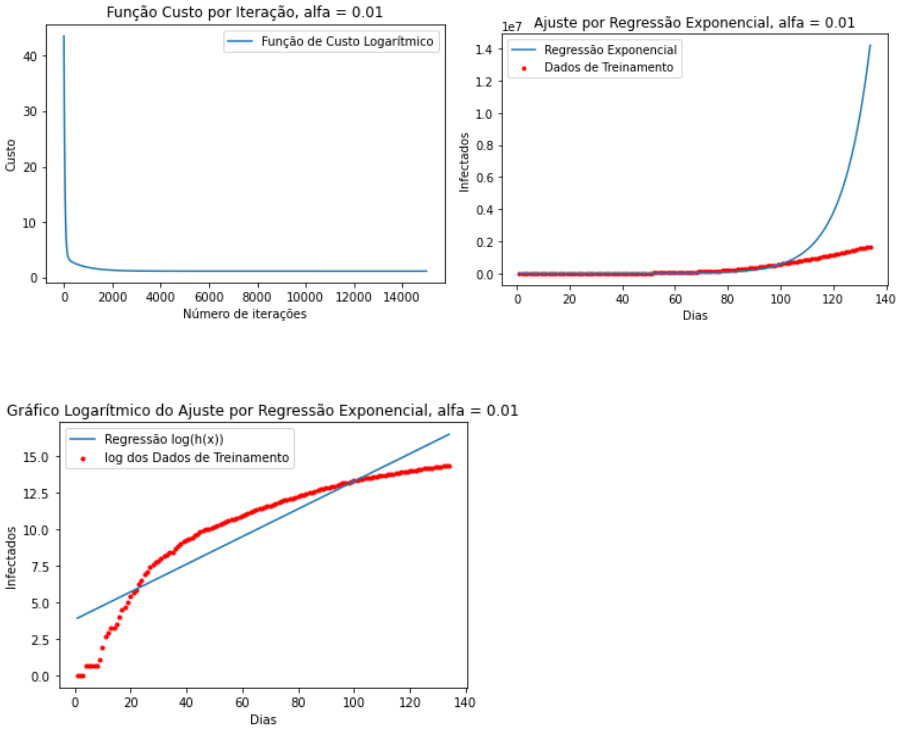


Figure 4: Resultado dos *plots*, respectivamente, das funções de perda, da aproximação pela exponencial e da aproximação $\log(h_\theta(x))$

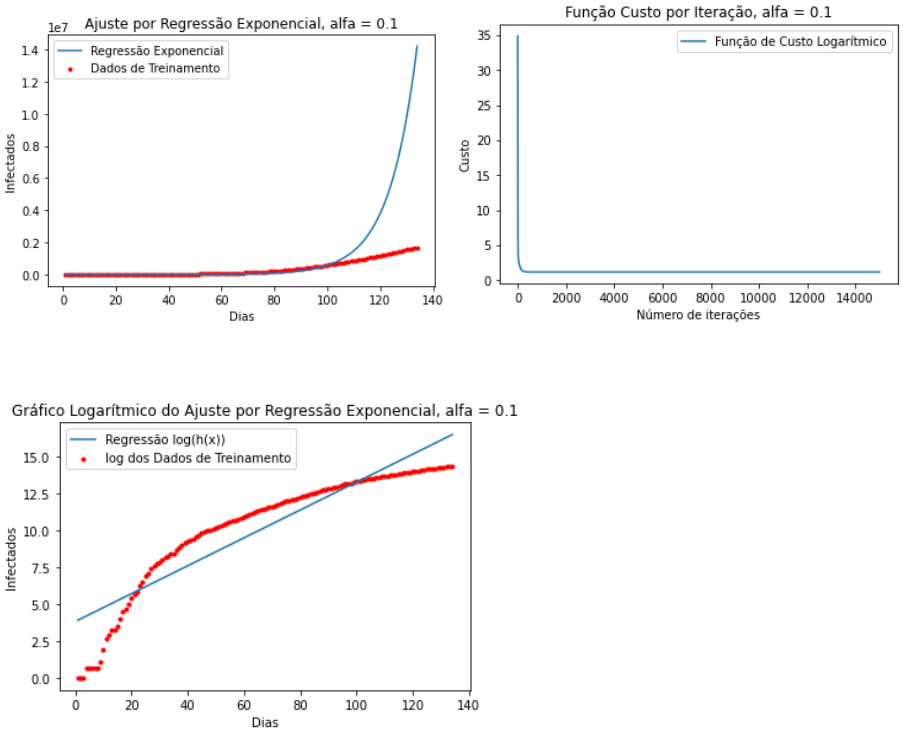
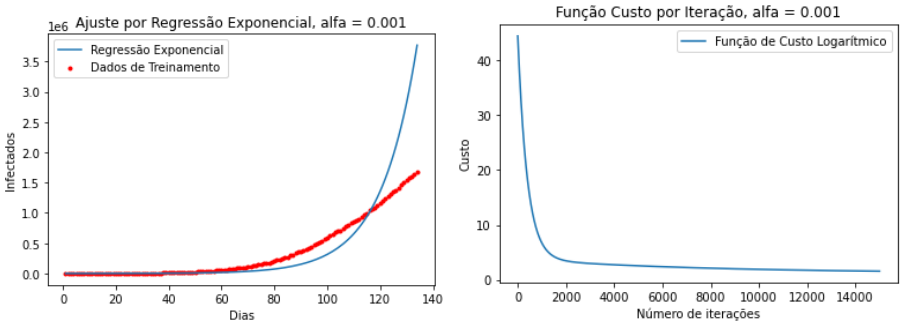


Figure 5: Resultado dos *plots*, respectivamente, das funções de perda, da aproximação pela exponencial e da aproximação $\log(h_\theta(x))$



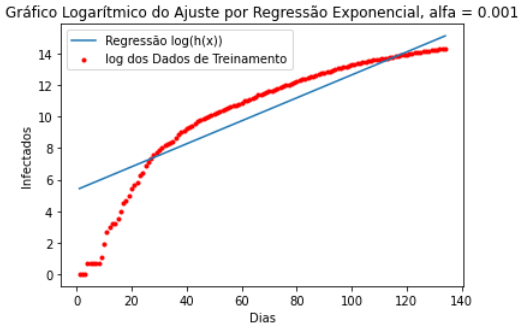
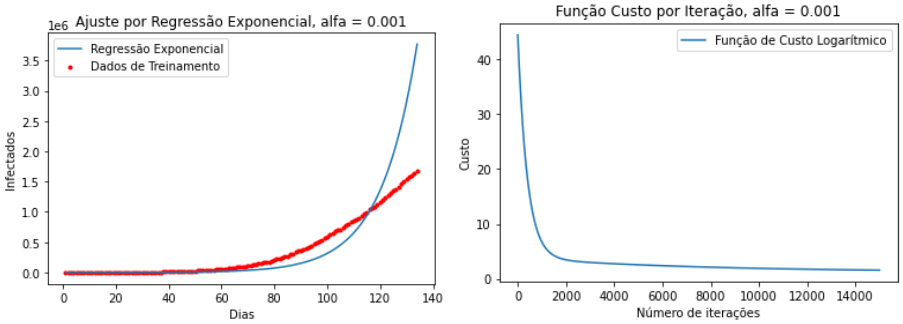


Figure 6: Resultado dos *plots*, respectivamente, das funções de perda, da aproximação pela exponencial e da aproximação $\log(h_\theta(x))$



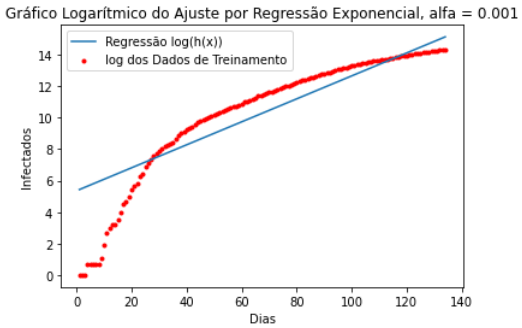
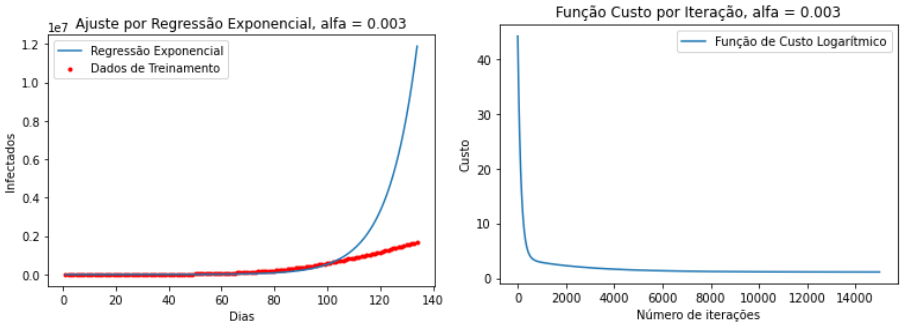


Figure 7: Resultado dos *plots*, respectivamente, das funções de perda, da aproximação pela exponencial e da aproximação $\log(h_\theta(x))$



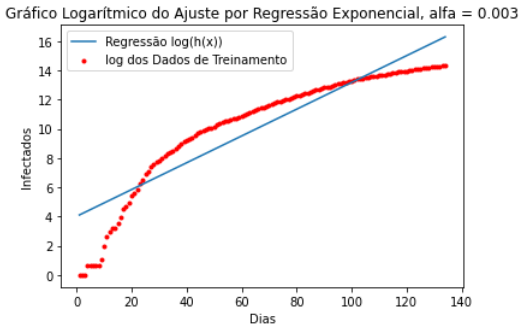
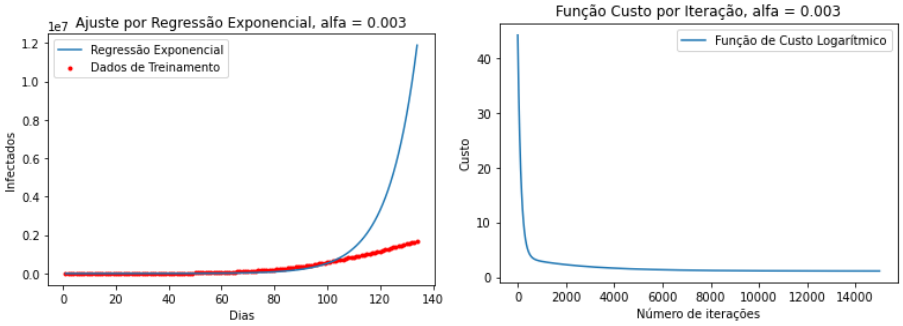


Figure 8: Resultado dos *plots*, respectivamente, das funções de perda, da aproximação pela exponencial e da aproximação $\log(h_\theta(x))$



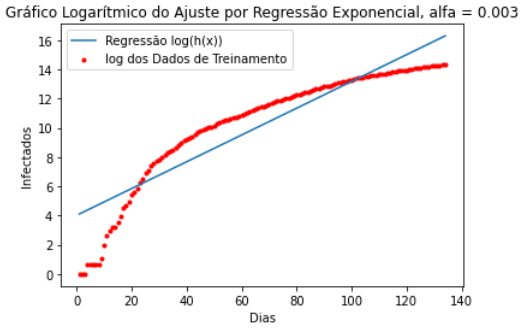
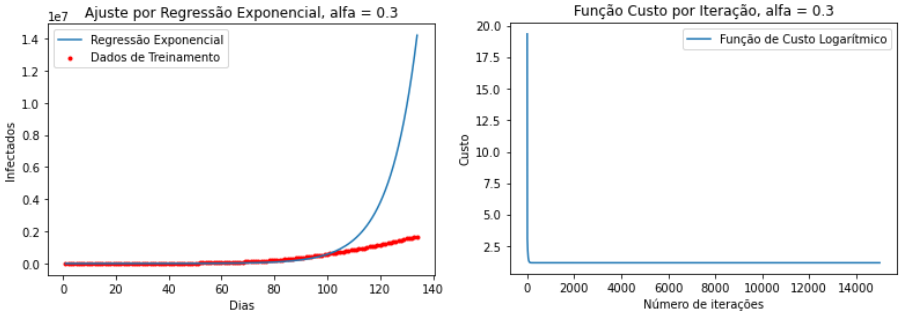


Figure 9: Resultado dos *plots*, respectivamente, das funções de perda, da aproximação pela exponencial e da aproximação $\log(h_\theta(x))$



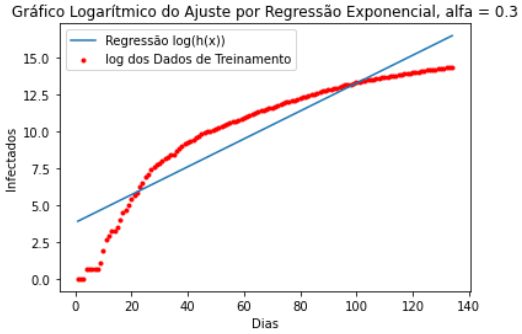


Figure 10: Resultado dos *plots*, respectivamente, das funções de perda, da aproximação pela exponencial e da aproximação $\log(h_\theta(x))$

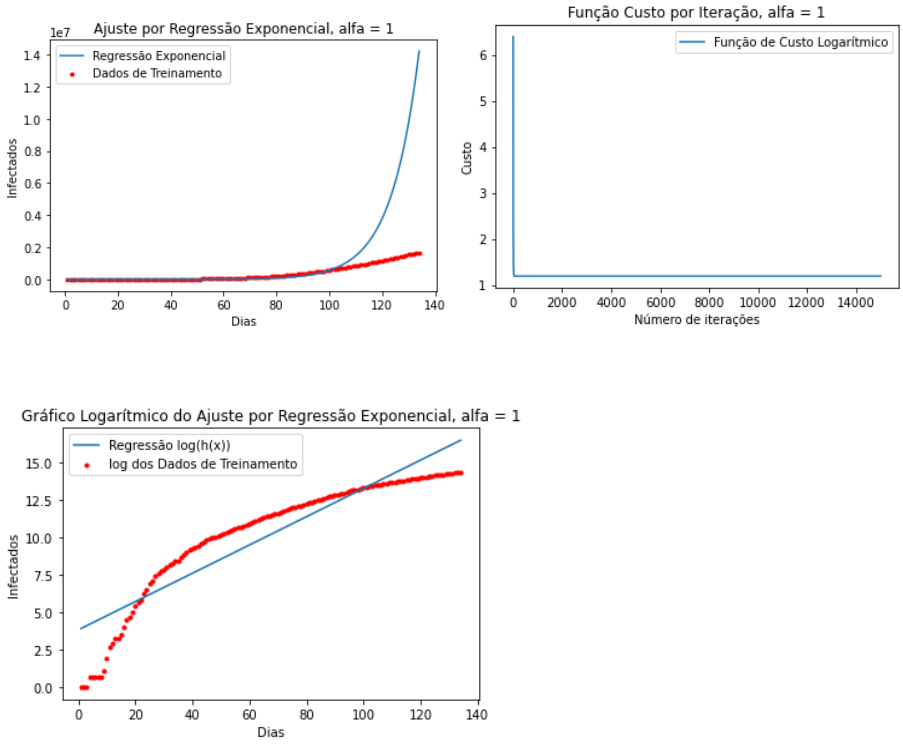


Figure 11: Resultado dos *plots*, respectivamente, das funções de perda, da aproximação pela exponencial e da aproximação $\log(h_\theta(x))$

Aproximação utilizando equações normais

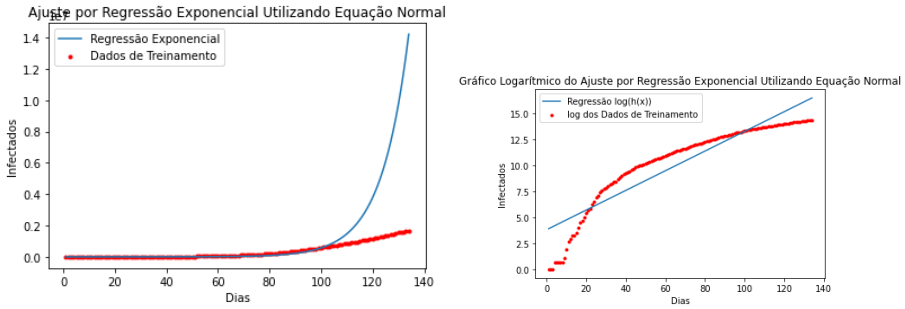


Figure 12: Resultado dos *plots*, respectivamente, da aproximação pela exponencial e da aproximação $\log(h_\theta(x))$ utilizando equações normais

Regressão Logística

$\alpha = 0.2$ **e** 50 **iterações**

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	481	2	1	0	3	2	1	10	0	0	0
2	20	396	16	10	1	18	6	20	3	10	0
3	7	19	412	1	13	5	9	23	9	2	0
4	8	5	1	430	0	13	2	7	32	2	0
5	15	7	71	16	310	17	1	28	16	19	0
6	12	10	0	4	3	457	2	3	0	9	0
7	22	6	1	13	0	0	433	2	13	10	0
8	13	10	32	3	7	7	2	409	15	2	0
9	8	5	7	32	4	2	25	8	398	10	0
10	1	1	2	1	0	13	0	6	0	476	0

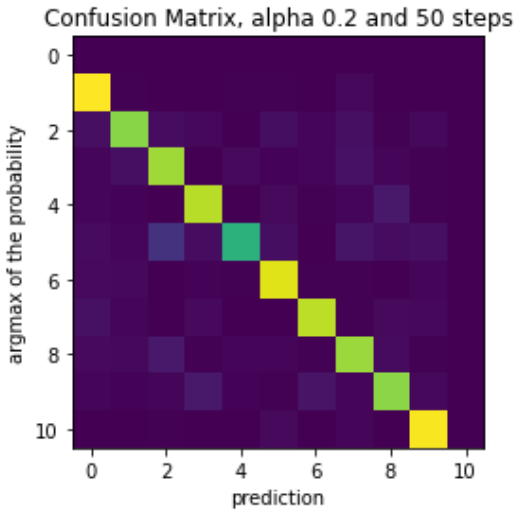
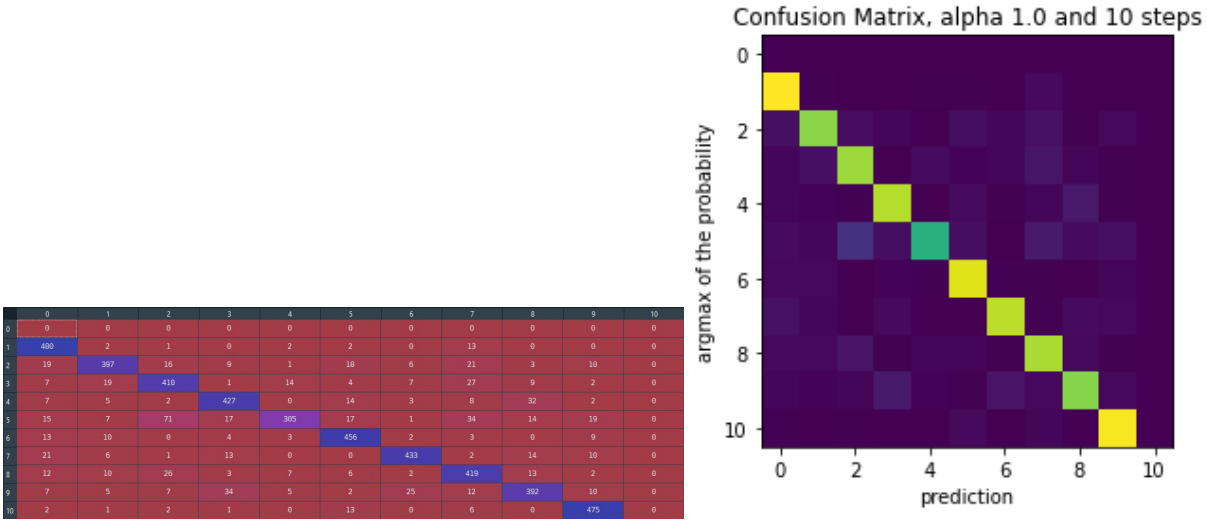


Figure 13: Resultado dos valores obtidos nas iterações e o resultado pela matriz de confusão

$\alpha = 1.0$ **e** 10 **iterações**



	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	480	2	1	0	2	2	0	13	0	0	0
2	19	397	16	9	1	18	6	21	3	10	0
3	7	19	410	1	14	4	7	27	9	2	0
4	7	5	2	427	0	14	3	8	32	2	0
5	15	7	71	17	305	17	1	34	14	19	0
6	13	10	0	4	3	456	2	3	0	9	0
7	21	6	1	13	0	0	433	2	14	10	0
8	12	10	26	3	7	6	2	419	13	2	0
9	7	5	7	34	5	2	25	12	392	10	0
10	2	1	2	1	0	13	0	6	0	475	0

Figure 14: Resultado dos valores obtidos nas iterações e o resultado pela matriz de confusão

$\alpha = 0.01$ **e** 50 **iterações**

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	439	2	4	0	0	2	0	52	0	1	0
2	7	365	26	2	0	25	5	25	1	44	0
3	2	12	404	2	1	3	6	36	3	31	0
4	5	2	6	367	0	21	1	18	58	22	0
5	3	18	89	8	58	31	3	77	12	201	0
6	2	12	0	1	0	444	1	9	0	31	0
7	11	11	2	6	0	1	392	25	14	38	0
8	1	7	45	0	0	5	0	407	7	28	0
9	4	2	10	22	0	2	20	41	366	32	0
10	1	1	2	0	0	10	0	3	0	483	0

Figure 15: Resultado dos valores obtidos nas iterações

$\alpha = 1.5$ **e** 10 **iterações**

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	470	2	1	0	0	3	0	24	0	0	0
2	14	388	15	9	0	12	6	45	2	9	0
3	6	18	399	2	2	6	5	59	1	2	0
4	5	7	3	429	0	11	3	26	15	1	0
5	11	10	87	19	177	14	2	145	12	23	0
6	6	11	0	3	3	450	2	17	0	8	0
7	16	6	1	13	0	0	433	16	5	10	0
8	2	5	18	1	0	3	0	469	1	1	0
9	5	5	9	42	2	1	32	69	325	9	0
10	1	1	1	1	0	10	1	17	0	468	0

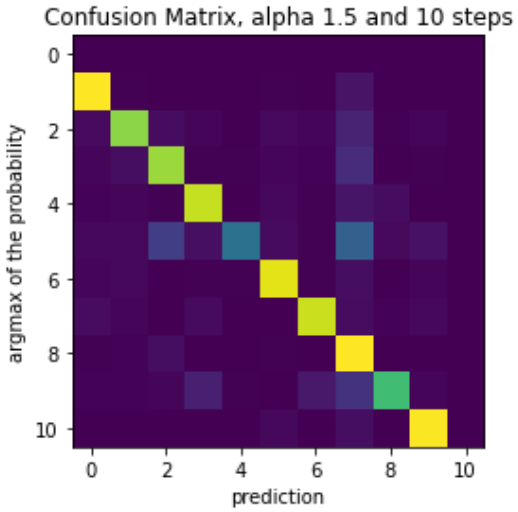


Figure 16: Resultado dos valores obtidos nas iterações e o resultado pela matriz de confusão

$\alpha = 0.4$ **e** 25 **iterações**

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	481	2	1	0	3	2	1	10	0	0	0
2	20	395	16	10	1	18	6	21	3	10	0
3	7	19	412	1	13	5	9	23	9	2	0
4	8	5	1	429	0	13	2	7	33	2	0
5	15	7	71	16	310	17	1	28	16	19	0
6	12	10	0	4	3	457	2	3	0	9	0
7	22	6	1	13	0	0	433	2	13	10	0
8	13	10	32	3	7	7	2	409	15	2	0
9	8	5	7	32	4	2	25	8	398	10	0
10	2	1	2	1	0	12	0	6	0	476	0

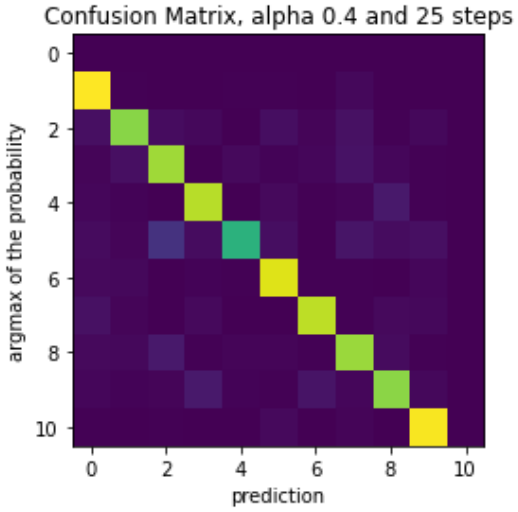
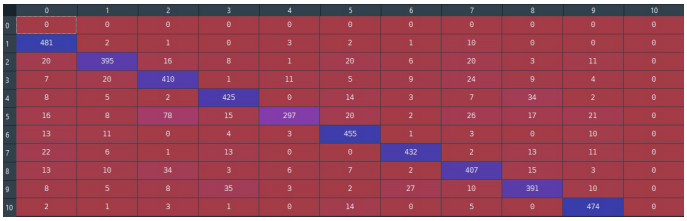


Figure 17: Resultado dos valores obtidos nas iterações e o resultado pela matriz de confusão

$\alpha = 0.8$ **e** 10 **iterações**



	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	481	2	1	0	3	2	1	10	0	0	0
2	20	395	16	8	1	20	6	20	3	11	0
3	7	20	410	1	11	5	9	24	9	4	0
4	8	5	2	425	0	14	3	7	34	2	0
5	16	8	78	15	297	20	2	26	17	21	0
6	13	11	0	4	3	455	1	3	0	10	0
7	22	6	1	13	0	0	432	2	13	11	0
8	13	10	34	3	6	7	2	407	15	3	0
9	8	5	8	35	3	2	27	10	391	10	0
10	2	1	3	1	0	14	0	5	0	474	0

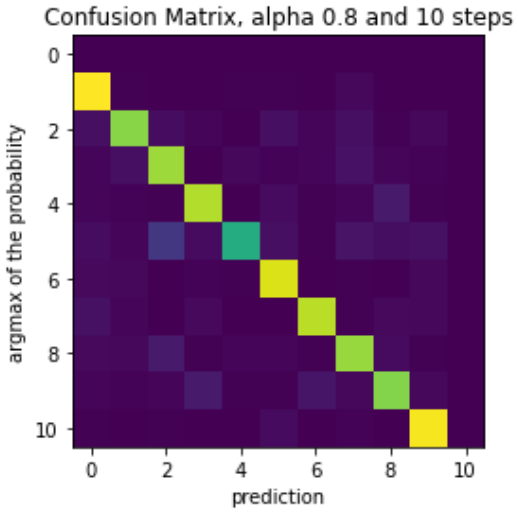
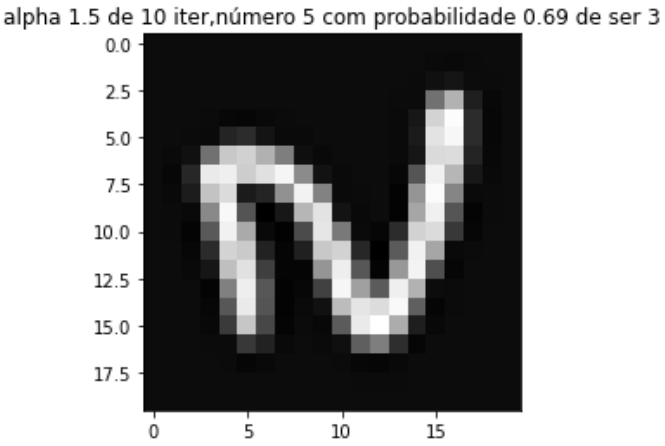
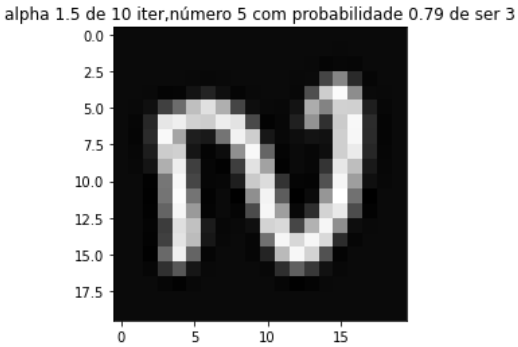


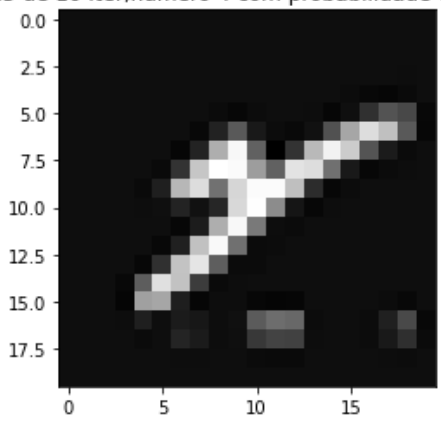
Figure 18: Resultado dos valores obtidos nas iterações e o resultado pela matriz de confusão

Classificações equivocadas do modelo

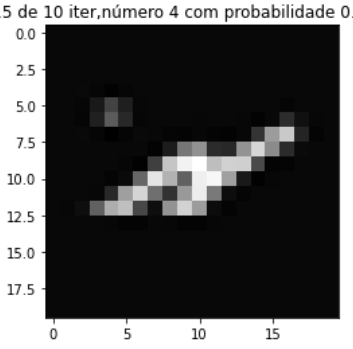




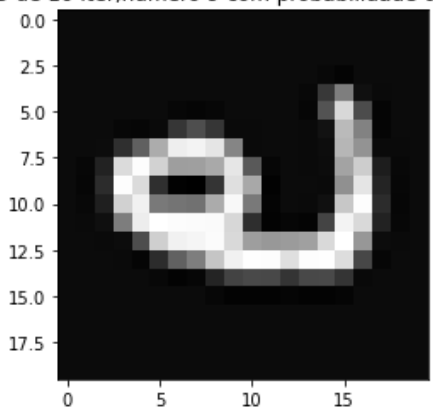
alpha 1.5 de 10 iter,número 4 com probabilidade 0.72 de ser 8



alpha 1.5 de 10 iter,número 4 com probabilidade 0.80 de ser 1

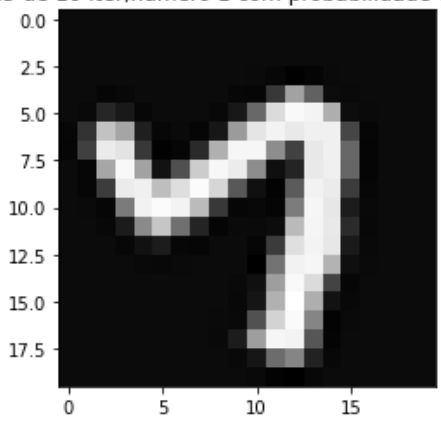


alpha 1.5 de 10 iter,número 9 com probabilidade 0.766 de ser 3

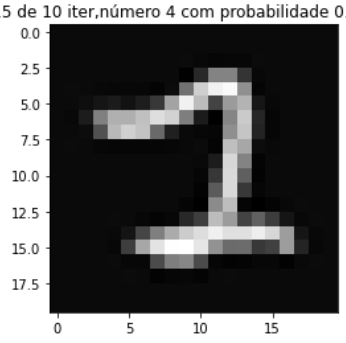


alpha 1.5 de 10 iter,número 9 com probabilidade 0.644 de ser 1

alpha 1.5 de 10 iter,número 1 com probabilidade 0.88 de ser 6



alpha 1.5 de 10 iter,número 4 com probabilidade 0.49 de ser 10



alpha 1.5 de 10 iter,número 2 com probabilidade 0.68 de ser 4

alpha 1.5 de 10 iter,número 2 com probabilidade 0.74 de ser 8