Regressão Logística e Redes Neurais

Rodrigo Rusa
208592
rodrigorusa@gmail.com

Thales Eduardo Nazatto
074388
tenazatto@gmail.com

I. Introdução

Regressão Logística em Aprendizado de Máquina é uma técnica de aprendizado supervisionado que consiste na regressão de um modelo matemático que relaciona variáveis de entrada $X_i(i = 1, 2, ..., n)$ a diferentes grupos de classificação. Para isso, é usada a função *Sigmoid* para determinar a probabilidade de um determinado conjunto de variáveis a pertencerem a determinado grupo:

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}} \tag{1}$$

Como já visto anteriormente em Regressões Lineares, na Regressão Logística o melhor modelo de classificação é encontrado através da utilização do algoritmo de Gradiente Descendente, atualizando os valores de θ_j até encontrar o minímo da função custo J:

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

Quando o melhor modelo de classificação é encontrado, tal classificação está relacionado apenas a uma classe, sendo considerado um modelo de classificação binária por apenas determinar se um dado pode ser considerado da classe em questão ou não. Para problemas de multiclassificação, ou seja, quando temos que classificar modelos com 3 ou mais classes, podemos usar duas abordagens diferentes para resolver essa limitação:

- **One-vs-All/One-vs-Rest:** Esta abordagem consiste em realizar um modelo específico da Regressão Logística para cada classe, totalizando *m* modelos diferentes. Após a descoberta de todos os modelos, a função *Sigmoid* é aplicada para determinar as probabilidades de cada classe, e a com maior valor é a determinada pela predição.
- **Regressão Multinomial:** Esta abordagem generaliza a Regressão Logística para tratar todas as *m* classes de uma única vez. Para isso, ele usa a função *Softmax* no lugar da *Sigmoid*:

$$h_\theta(x) = \frac{e^{\theta_j^T x^{(i)}}}{\sum_{i=1}^{m} e^{\theta_j^T x^{(i)}}} \tag{3}$$

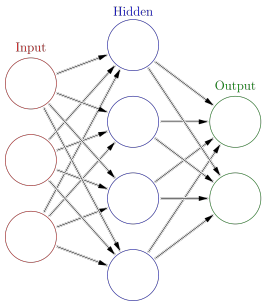
  Dessa forma, é necessário apenas um único modelo pois a soma das probabilidades de todas as classes é sempre 1 para todos os itens, e a com maior valor é a determinada pela predição. Nessa regressão, o cálculo de custo e o cálculo do gradiente mudam sutilmente (mostrados em (4) e (5), respectivamente), diferentemente da Regressão Logística onde o cálculo do gradiente é o mesmo da Regressão linear.

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m} log(h_\theta(x_{y_j}^{(i)})) \tag{4}$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \begin{cases} x^{(i)}(h_\theta(x) - 1), & \text{se } i = y \\ x^{(i)} h_\theta(x), & \text{se } i \neq y \end{cases} \tag{5}$$

Assim como a Regressão Logística, as Redes Neurais são técnicas de aprendizado supervisionado em Aprendizado de Máquina. Uma Rede Neural usualmente é composta por vários neurônios organizados em camadas como ilustra a Figura 1. Cada neurônio representa uma unidade de computação que recebe sinais de entrada de outros neurônios vizinhos com uma certa intensidade (peso) e sua saída é definida pela aplicação de uma função chamada de ativação que é aplicada nas entradas do neurônio. Assim, esse pesos são propagados atá a última camada, a camada de saída.

Figure 1: Rede Neural com uma camada escondida



As Redes Neurais podem ter diversas camadas de neurônios, com tamanhos diferentes, chamadas de camadas intermediárias, quanto mais camadas intermediárias a rede possuir mais complexo o modelo se torna, pois mais paramêtros são usados para descrever o modelo. A primeira camada denomina-se camada de entrada, o número de neurônios dessa camada depende do número de *features* do problema. A última camada denomina-se camada de saída, para problemas de classificação com múltiplas classes, por exemplo, o número de neurônios utilizado é o número de classes do problema. Nesses casos é aplicado uma classificação *Softmax* após a última camada que faz a conversão das saídas para valores de probabilidade de cada classe.

Para as funções de ativação dos neurônios, diversas funções podem ser utilizadas, as mais comuns são a *Sigmoid*, *Tanh* e *ReLU (Rectified Linear Unit)*.

O treinamento de uma Rede Neural consiste, basicamente, em dois passos principais:

- **FeedForward:** Nesse passo as entradas são apresentadas a rede e as saídas de cada neurônio são computadas e propagadas ao longo das camadas intermediárias até a última camada, de saída. Na primeira execução desse passo, os pesos das ligações entre os neurônios são definidas aleatoriamente.
- **BackPropagation:** Com a saída calculada no passo anterior, o erro entre a saída da rede e a saída esperada é calculada e propagada pelas camadas da rede até a camada de entrada. Isso faz com que a rede ajuste os paramêtros (pesos) de modo a minimizar a função de erro, função custo J como em (6), semelhante ao Gradiente Descendente utilizado nas Regressões.

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} log(h_\theta(x^{(i)}))_k + (1 - y_k^{(i)}) log(1 - (h_\theta(x^{(i)}))_k) + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{S_l}\sum_{j=1}^{S_{l+1}}(\theta_{ji})^2 \quad (6)$$

Essas técnicas de aprendizado supervisionado, Regressão Logística e Redes Neurais, são muito utilizadas em problemas de classificação. Nesse trabalho exploramos essas duas técnicas para encontrar o melhor modelo matemático que é capaz de classificar peças de roupas, utilizando a base de dados *Fashion-MNIST* disponibilizado em [1]

Essa base de dados possui um total de 70,000 exemplos, 60,000 exemplos de treinamento e 10,000 de teste. Cada exemplo consiste em uma imagem de 28x28 pixels em escala de cinza, ou seja, com valores de 0 a 255 e um rótulo de acordo com as classes: T-shirt/top (0), Trouser (1), Pullover (2), Dress (3), Coat (4), Sandal (5), Shirt (6), Sneaker (7), Bag (8), Ankle boot (9).

II. Metodologia

Para o uso da Regressão Logística foi implementado na linguagem *Python* um classificador utilizando as abordagens *One-vs-All* e Regressão Multinomial realizando o Gradiente Descendente no modo *Batch*, ou seja, a cada passo do algoritmo todo os dados de treinamento são utilizados na descida do gradiente. Para otimizar o desempenho dos classificadores foi desenvolvida uma alternativa vetorial em detrimento da iterativa, aproveitando as qualidades da biblioteca *NumPy* e inspirada em abordagens vistas em [2] e [3]. Também foram implementados os métodos existentes na biblioteca *Scikit Learn*. Antes de ser realizadas as regressões, foi realizada a normalização dos dados subtraindo os valores pela média e dividindo pelo desvio padrão, e após os classificadores foram calculadas as estatísticas dos modelos em questão de tempo de execução para obtenção dos mesmos.

A fim de explorar os modelos de aprendizado supervisionado utilizando Redes Neurais foi implementado na linguagem *Python* um classificador utlizando Rede Neural capaz de criar redes com múltiplas camadas intermediárias com diferentes quantidades de neurônios. Para esse classificador implementado, também foi utilizado o aprendizado no modo *Batch*. Além disso, foram implementadas diferentes funções de ativações: *Sigmoid*, *Tanh* e *ReLU*.

A camada de entrada da rede consiste em 784 neurônios, um para cada pixel das imagens de entrada de tamanho 28x28. A camada de saída utilizada possui 10 neurônios, um para cada classe da base *Fashion-MNIST*. Ao final da camada de saída foi adicionado um classificador *Softmax* para converter as saídas em probabilidades, de modo que a classe de saída esperada seja a de maior probabilidade.

Como os dados de entrada correspondem a valores de pixel (0 - 255), foi também realizado a normalização dos dados dividindo cada pixel pelo valor máximo de 255.

Para validar o modelo de Rede Neural implementado com múltiplas camadas, foi adicionado ao framework desenvolvido a implementação do MLP pelo pacote *Scikit Learn* [4].

Assim, com esse framework desenvolvido realizou-se alguns experimentos variando o número de camadas intermediárias, o número de neurônios das camadas intermediárias e as funções de ativação dos neurônios com o objetivo de encontrar o modelo de classificação de melhor acurácia para a base *Fashion-MNIST*.

Ambos os experimentos foram conduzidos usando apenas o conjunto de treinamento que foi dividido em treinamento e validação, sendo 80% para treinamento e 20% para validação. O conjunto de teste foi usado apenas uma vez para testar o melhor modelo encontrado.

III. Experimentos e Discussão

Para validar a corretude da implementação das abordagens relativas a Regressão Logística, o experimento realizado visava verificar a convergência dos cálculos de erro, o aumento da acurácia normalizada conforme as iterações e o Tempo de execução de cada um em segundos. Para isso, foram realizados testes com 100, 1000 e 10000 iterações, *learning rate* de 0.01 e tolerância de 10^{-6}. Para a estratégia *One-Vs-All*, o valor de $J(\theta)$ é equivalente a média de todas as funções de custo de todas as classes. Os resultados foram colocados na Tabela abaixo:

Algoritmo			Treinamento		Validação	
Abordagem	Iterações	$T.Exec.$	$J(\theta)$	$Acur.$	$J(\theta)$	$Acur.$
One-Vs-All	100	173.47	0.566894	0.35	0.566895	0.34
	1000	1738.59	0.340638	0.41	0.340649	0.40
	10000	5047.69	0.323920	0.40	0.323915	0.41
Multinomial	100	50.88	2.302117	0.23	2.302134	0.23
	1000	530.11	2.297837	0.36	2.297969	0.36
	10000	4953.39	2.256434	0.54	2.256726	0.54

Podemos ver que os erros de ambos os conjuntos ficaram muito próximos, caracterizando assim que não ocorre *overfitting* no modelo. Nos casos de 100 e 1000 iterações, a *One-Vs-All* leva mais tempo para ser executado, uma vez que precisa calcular 10 modelos de classes diferentes contra 1 da Regressão Multinomial e inicialmente possui a maior acurácia.

Conforme o número de iterações vai aumentando, a função de custo dos modelos da *One-Vs-All* converge para um valor mínimo e o valor de sua acurácia é estacionado, validando sua implementação. No caso da Regressão Multinomial, o valor da função de custo não converge, mas a acurácia de seu modelo aumenta conforme o número de iterações também aumenta, o que também valida sua implementação.

Quanto a abordagem vetorial dos algoritmos, pode-se dizer, mesmo sem dados de uma abordagem iterativa, que ela aumenta bastante o desempenho, uma vez que para seu treinamento são utilizados 48000 amostras com 785 coeficientes para serem calculados a cada iteração. Em média, cada iteração levou aproximadamente 0.173 segundo na *One-Vs-All* e 0.5 segundo na Regressão Multinomial, o que pode ser considerado baixo devido ao tamanho da base de dados utilizada.

Figure 2: Gráfico da função custo J ao longo das iterações da abordagem *One-Vs-All*

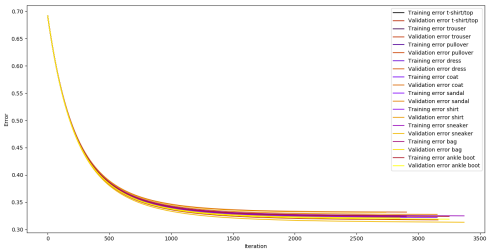
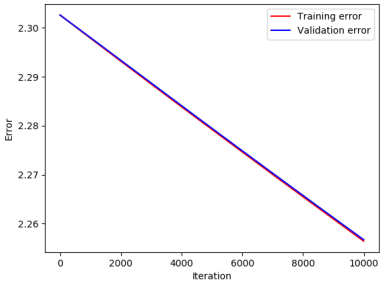


Figure 3: Gráfico da função custo J ao longo das iterações da abordagem Multinomial



Para validar os modelos finais das abordagens relativas a Regressão Logística, um novo experimento foi realizado para verificar a eficácia do modelo no conjunto de teste. Para isso, foram realizadas execuções de 10000 iterações, *learning rate* de 0.01 e tolerância de 10^{-6} e foram calculadas novas estatísticas, uma vez que a acurácia é apenas uma das métricas para verificar sua eficácia e não é definitiva. Os resultados foram colocados na tabela abaixo:

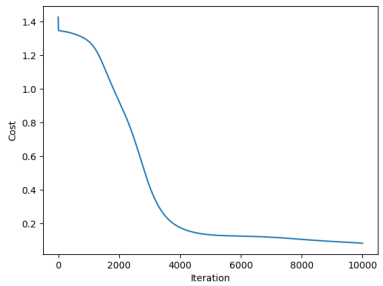
Abordagem	$T.Exec.$	$Acur.$	$Prec.$	Recall	F_1
One-Vs-All	5947.05	0.41	0.49	0.41	0.44
Multinomial	9955.36	0.54	0.52	0.54	0.53

Os valores dos tempos de execução estão diferentes pois os experimentos foram realizados em máquinas diferentes. Apesar de ter um menor desempenho, a Regressão Multinomial apresentou valores melhores de acurácia, precisão e recall. Consequentemente, seu *score* F_1, que determina a harmonia entre precisão e recall, também é superior. Esse fator junto com a ainda não-convergência de seu modelo fazem da Regressão Multinomial a melhor abordagem de Regressão Logística para esta base de dados.

Realizados os experimentos utilizando Regressão Logística, iniciou-se os experimentos utilizando Redes Neurais. O primeiro experimento realizado teve como objetivo validar a corretude da implementação antes de submetê-la a base de dados proposta no trabalho. Para isso, usou-se dados de entrada da operação lógica *XOR*.

Foram testadas duas redes diferentes, com uma e com duas camadas intermediárias, ambas com 5 neurônios. Além disso, o resultado foi comparado com a implementação do *Scikit Learn*. Foram utilizados nesse experimento a função de ativação *Sigmoid*, *learning rate* de 0.1 e 10.000 iterações. A Figura 4 ilustra a função custo J da rede com uma camada intermediária ao longo das iterações.

Figure 4: Gráfico da função custo J ao longo das iterações para treinar o modelo *XOR*



Como podemos ver, a convergência foi alcançada. A Tabela abaixo ilustra os resultados obtidos nesse experimento:

Algoritmo	Camadas	$Acur.$
Implementado	1	1.0
	2	1.0
Scikit Learn	1	1.0
	2	1.0

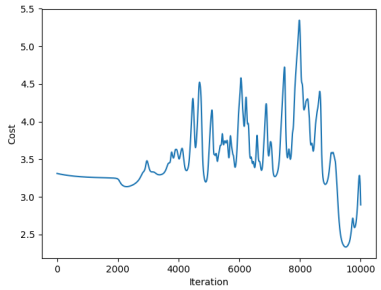
Com base nos resultados obtidos podemos concluir que a Rede Neural implementada apresentou boa acurácia para a classificação da operação *XOR* e resultado similar ao da biblioteca *Scikit Learn*, validando assim a implementação do algoritmo.

A fim de encontrar o melhor classificador baseado em Redes Neurais para a base de dados *Fashion-MNIST*, foram realizados alguns experimentos partindo de modelos mais simples, com apenas uma camada intermediária, para modelos mais complexos com duas camadas intermediárias e diferentes funções de ativação.

No primeiro modelo experimentado foi utilizado apenas uma camada intermediária com 512 neurônios, função de ativação *Sigmoid*, *learning rate* de 0.1 e 10.000 iterações. A função custo J ao longo das iterações é ilustrada na Figura 5.

Podemos observar que a função custo apresentou oscilação ao longo das iterações não alcançando a convergência esper-

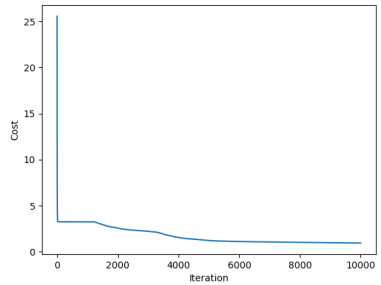
Figure 5: Gráfico da função custo J ao longo das iterações para rede com uma camada de 512 neurônios



ada, o que resultou em uma acurácia de 0.57 para o conjunto de validação. Esse resultado indica que o número de parâmetros, ou seja, o número de neurônios e camadas intermediárias são insuficientes para modelar os dados com boa acuŕacia.

Com o objetivo de melhorar a acurácia do classificador encontrado com uma camada intermediária foi realizado outro experimento com duas camadas intermediárias, cada uma com 256 neurônios, aumentado a complexidade do modelo. Para esse experimento foram utilizados, função de ativação *Sigmoid*, *learning rate* de 0.1 e 10.000 iterações. O comportamento da função custo J ao longo das iterações é ilustrado na Figura 6.

Figure 6: Gráfico da função custo J ao longo das iterações para rede com duas camadas de 256 neurônios



Podemos ver pelo gráfico do custo J que a convergência foi alcançada e que o modelo atingiu acurácia de 0.78 no conjunto de validação. Comparado com o modelo com apenas uma camada intermediária, esse apresentou melhor resultado, pois aumentou-se a complexidade do modelo, mais parâmetros foram utilizados.

Além da função de ativação *Sigmoid*, outras duas funções foram implementadas nesse trabalho, a *Tanh* e *ReLU*. Desse modo, a fim de avaliar o impacto da função de ativação na performance da rede foram testadas as funções *Tanh* e *ReLU* com uma rede com duas camadas intermediárias de 256 neurônios cada, *learning rate* de 0.1 e 10.000 iterações. Os resultados desse experimento são ilustrados na Tabela a seguir:

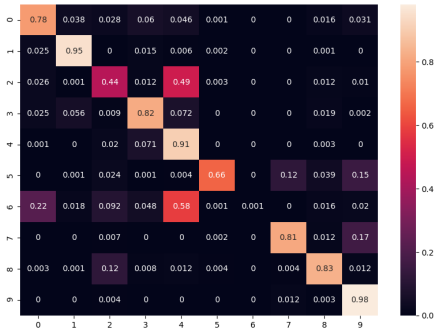
Função de Ativação	*Acur.*	*Prec.*	*Recall*	F_1
Sigmoid	0.78	0.78	0.78	0.77
Tanh	0.10	0.01	0.1	0.01
ReLU	0.09	0.0	0.1	0.01

Como podemos observar, a utilização das funções de ativção *Tanh* e *ReLU* não melhorou a acurácia do modelo. Na utilização da *Tanh* como função de ativação a função custo J oscilou ao longo das iterações e não convergiu e na *ReLU* a função custo convergiu mas com alto valor de custo, não apresentando boa performance.

Por fim, a partir do melhor modelo encontrado nos experimentos realizados com Regressão Logística e Redes Neurais foi apresentado o conjunto de teste e analisado a acurácia do modelo nesse conjunto. O melhor modelo escolhido foi o que apresentou melhor acurácia no conjunto de validação, portanto o modelo de Rede Neural com duas camadas intermediárias de 256 neurônios e função de ativação *Sigmoid* foi o escolhido. O resultado do teste com as métricas de avaliação e a matriz de confusão são ilustrados a seguir:

Acur.	*Prec.*	*Recall*	F_1
0.71	0.78	0.71	0.68

Figure 7: Matriz de confusão para o conjunto de teste



Podemos concluir que o modelo obteve boa acurácia no conjunto de teste, próximo da acurácia obtida no conjunto de validação.

IV. Conclusões

Nesse trabalho foram exploradas as técnicas de Regressão Logística e Redes Neurais com o objetivo de encontrar o melhor modelo de classificação para a base de dados *Fashion-MNIST*. Para isso foi implementado essas duas técnicas e realizado experimentos. Concluímos que as Redes Neurais se mostraram mais eficiente que a Regressão Logística para a classificação dos dados, atingindo maior acurácia nos testes. Porém, encontrar os parâmetros ideias para a Rede Neural, como número de camadas, número de neurônios em cada camada, função de ativação e *learning rate* requer tempo de processamento e vários testes.

References

[1] *Fashion-MNIST* dataset: https://github.com/zalandoresearch/fashion-mnist
[2] Logistic and Softmax Regression: https://houxianxu.github.io/2015/04/23/logistic-softmax-regression/
[3] Gradient descent on a Softmax cross-entropy cost function: https://madalinabuzau.github.io/2016/11/29/gradient-descent-on-a-softmax-cross-entropy-cost-function.html
[4] Scikit Learn: http://scikit-learn.org