

**Fundação Getulio Vargas
Escola de Matemática Aplicada
Curso de Graduação em Ciência de
Dados**

Regressão Logística

**Bianca Dias de Carvalho
Luis Fernando Laguardia**

Rio de Janeiro - Brasil
2021

**Fundação Getulio Vargas
Escola de Matemática Aplicada
Curso de Graduação em Ciência de
Dados**

Regressão Logística

**Bianca Dias de Carvalho
Luis Fernando Laguardia**

Rio de Janeiro - Brasil
2021

Sumário

1	Introdução	3
2	Desenvolvimento	4
3	Considerações Finais	16
4	Referências	17

1 Introdução

Este trabalho aborda a regressão logística, contendo notebooks em python, onde a regressão logística é aplicada em diversas bases de dados, com o fito de realizar previsões a partir de variáveis categóricas.

A regressão logística é uma técnica de mineração de dados, que consiste no processo de encontrar padrões e correlações em grandes conjuntos de dados para prever resultados, além disso, através dela também é possível obter a probabilidade de ocorrência de cada evento, assim como a influência de cada variável independente.

A principal diferença entre a regressão logística e a regressão linear é que a variável dependente/resposta, atributo que se quer prever, é categórica, frequentemente binária.

2 Desenvolvimento

Inicialmente, foi criada uma classe de regressão logística em python.

```
1 class LogisticRegression:
2     def __init__(self, learning_rate=0.001, n_iters=1000):
3         self.lr = learning_rate
4         self.n_iters = n_iters
5         self.weights = None
6         self.bias = None
7
8     def fit(self, X, y):
9         n_samples, n_features = X.shape
10        self.weights = np.zeros(n_features) #parametro inicial
11        self.bias = 0 #parametro inicial
12
13        #gradiente descendente
14        for _ in range(self.n_iters):
15            linear_model = np.dot(X, self.weights) + self.bias #
16                aproxima y com a combinacao linear dos pesos e x
17                somada a constante
18            y_predicted = self._sigmoid(linear_model) #aplica a
19                funcao sigmoide
20
21            #computa os gradientes
22            dw = (1 / n_samples) * np.dot(X.T, (y_predicted - y)
23                )
24            db = (1 / n_samples) * np.sum(y_predicted - y)
25            #atualiza os parametros
26            self.weights -= self.lr * dw
27            self.bias -= self.lr * db
28
29        def predict(self, X):
30            linear_model = np.dot(X, self.weights) + self.bias
31            y_predicted = self._sigmoid(linear_model)
32            y_predicted_cls = [1 if i > 0.5 else 0 for i in
33                y_predicted]
34
35            return np.array(y_predicted_cls)
36
37        def _sigmoid(self, x):
38            return 1 / (1 + np.exp(-x))
```

O gradiente descendente é um algoritmo de otimização, seu objetivo é minimizar algumas funções movendo-se iterativamente na direção de descida mais íngreme. O parâmetro *learning rate* nos diz qual distância será percorrida em cada iteração, ele não pode ser muito pequeno devido ao custo computacional, nem muito grande por falhar em convergir no mínimo local. A função *sigmoide* é aplicada à soma ponderada, essa função transforma varia de 0 a 1 e tem um formato S (não é linear) e basicamente tenta empurrar os valores de y para os extremos.

```
1 def plot(self, X, y, legend):
2     # essa funcao plota o resultado apenas se X se referir a
3     # exatamente 2 variaveis
4     if X.shape[1] != 2:
5         raise ValueError("Can plot only for X's that refers
6                             to exactly 2 vars.")
7
8     slope = -(self.weights[0]/self.weights[1])
9     intercept = -(self.bias/self.weights[1])
10    predictions = self.predict(X)
11
12    sns.set_style('white')
13    sns.scatterplot(x = X[:,0], y= X[:,1], hue=y.reshape(-1)
14                    , style=predictions.reshape(-1));
15
16    ax = plt.gca()
17    ax.autoscale(False)
18    x_vals = np.array(ax.get_xlim())
19    y_vals = intercept + (slope * x_vals)
20    plt.plot(x_vals, y_vals, c="k");
21
22    plt.xlabel(legend[0])
23    plt.ylabel(legend[1])
```

A função acima plota o resultado da regressão linear em um gráfico de dispersão onde nos eixos estão as variáveis independentes.

Após isso, as bases foram importadas e os dados foram normalizados. Como exemplo, será mostrado a importação e normalização de uma das bases.

```
df = pd.read_csv("db_estrelas.csv")
2
df = df[(df['Spectral Class'] == 'B') | (df['Spectral Class'] ==
    'M')]
4
df['Spectral Class'].replace(to_replace='B', value=1, inplace=
    True)
df['Spectral Class'].replace(to_replace='M', value=0, inplace=
    True)
6
# Seleção de Dados
8
dados = {
    'X' : ['Temperature (K)', 'Absolute magnitude(Mv)'],
10
    'y' : 'Spectral Class',
    'normalizada' : False
12
}

14
df = df[ dados['X']+[dados['y']] ]
df = df.dropna()
16
if not dados['normalizada']:
18
    for col in dados['X']:
        df[[col]] = df[[col]]/df[[col]].mean()
20
X = df[ dados['X'] ].to_numpy()
22
y = df[[ dados['y'] ]].to_numpy()
y = np.hstack((y)).T
24
df.sample(5)
```

Também foi criada uma tabela que mostra os pesos de cada variável independente.

```

1 norma_pesos = pd.DataFrame(regressor.weights)/pd.DataFrame(
    regressor.weights).abs().sum()
2 norma_pesos = norma_pesos[0].values.tolist()
3
4 dfpesos = pd.DataFrame({'Pesos':norma_pesos}, index=dados['X'])
5 dfpesos

```

Figura 1: Pesos de cada variável

Out[7]:	Pesos
Temperature (K)	-0.98281
Radius(R/Ro)	0.01719

Nessa base, aplicamos a regressão logística para tentar prever a classe espectral das estrelas. Sabendo que a Classe Espectral de uma estrela é principalmente baseada em sua temperatura, mas geralmente mostrada em um gráfico de correlação temperatura x magnitude absoluta, utilizamos essas duas propriedades numéricas como colunas para a análise da regressão logística.

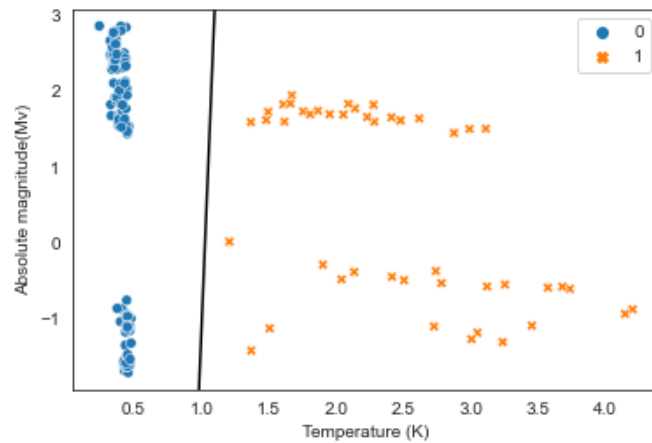
No primeiro caso, comparando as Classes Espectrais B (muito quente) e M (muito “fria”), conseguimos atingir 100% de precisão na aplicação do algoritmo. Porém, algo estranho de se observar é que as estrelas de Classe B estão cobrindo uma região muito dispersa de temperatura, algo incomum segundo as informações na Wikipedia.

```

1 df = df[(df['Spectral Class'] == 'A') | (df['Spectral Class'] ==
    'F')]
2 df['Spectral Class'].replace(to_replace='A', value=1, inplace=
    True)

```


Figura 2: Estrelas - Segundo Caso



```
df['Spectral Class'].replace(to_replace='F', value=0, inplace=True)

4
dados = {'X': ['Temperature (K)', 'Absolute magnitude(Mv)'],
6        'y': 'Spectral Class',
        'normalizada': False}

8
df = df[ dados['X']+[dados['y']] ]
10 df = df.dropna()

12 if not dados['normalizada']:
    for col in dados['X']:
14         df[[col]] = df[[col]]/df[[col]].mean()

16 X = df[ dados['X'] ].to_numpy()
y = df[[ dados['y'] ]].to_numpy()
18 y = np.hstack((y)).T

20 df.sample(5)
```

Após isso, a regressão logística foi aplicada e a precisão da previsão foi avaliada. A precisão desse modelo em específico foi de 100%.

Figura 3: Dados selecionados

Out[4]:

	Temperature (K)	Radius(R/R _o)	Spectral Class
111	0.448486	4.445558	1
43	0.398101	0.067237	1
193	0.399594	0.001495	1
84	1.754134	0.000035	0
175	0.449606	6.142306	1

```

regressor = LogisticRegression(learning_rate=0.000001, n_iters
    =2000)
2 regressor.fit(X, y)
predictions = regressor.predict(X)
4
def accuracy(y_true, y_pred):
6     accuracy = np.sum(y_true == y_pred) / len(y_true)
    return accuracy
8
print(f"A precisão do modelo : {accuracy(y, predictions)}")
10
norma_pesos = pd.DataFrame(regressor.weights)/pd.DataFrame(
    regressor.weights).abs().sum()
12 norma_pesos = norma_pesos[0].values.tolist()
14 dfpesos = pd.DataFrame({'Pesos':norma_pesos}, index=dados['X'])
16 dfpesos

```

Por fim, foi feita uma visualização dessa regressão.

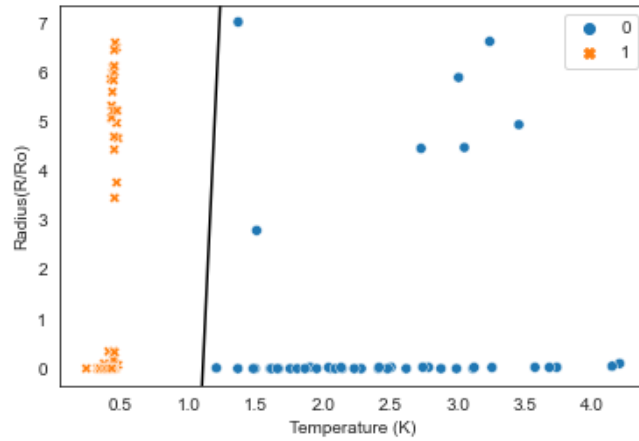
```

try:
2     regressor.plot(X, y, dados['X'])
except:
4     print("Sem visualização disponível.")

```

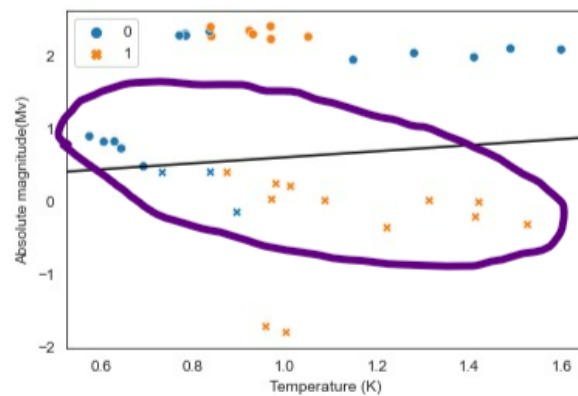
No segundo caso, percebemos o que acontece. Nessa base, ape-

Figura 4: Regressão linear - Resultado Final



nas as estrelas de classe M e as estrelas situadas na chamada Faixa Principal (região circulada) estão corretamente classificadas, enquanto as outras estão seguindo outro padrão. Por isso, ao aplicarmos a regressão logística em duas classes estelares diferentes de M e que saem da Faixa Principal (nesse caso, A e F), o algoritmo encontra outro padrão (fisicamente incorreto) e dá mais peso para a Magnitude Absoluta.

Figura 5: Classes Espectrais - Estrelas



Como as classes A e F estão muito próximas em temperatura, mesmo essa pequena inclinação em relação a temperatura significa um peso relativamente grande (30%), mas nem por isso a Magnitude Absoluta deixa de ser mais importante (absurdos 70%).

Figura 6: Pesos Variáveis Independentes - Segundo Caso

Out[10]:

Pesos	
Temperature (K)	0.293145
Absolute magnitude(Mv)	-0.706855

Apesar da inadequação física, esse fica sendo um ótimo exemplo de como o algoritmo funciona (além de um grande aprendizado sobre averiguar as bases antes de iniciar as análises).

A partir disso, segue o resultado final obtido com as demais bases.

- **Estudantes**

Neste exemplo, a nota do pré-teste foi utilizada para prever se a nota do pós-teste será maior ou igual a 7. A precisão deste modelo foi de 89% e conseguimos ver que essa variável independente tem grande influência na nota do pós-teste. A partir dessa análise, foi possível concluir que alunos com a nota igual ou superior à 58 no pré-teste teriam uma nota igual ou superior à 7 no pós-teste.

Figura 7: Gráfico sobre os estudantes

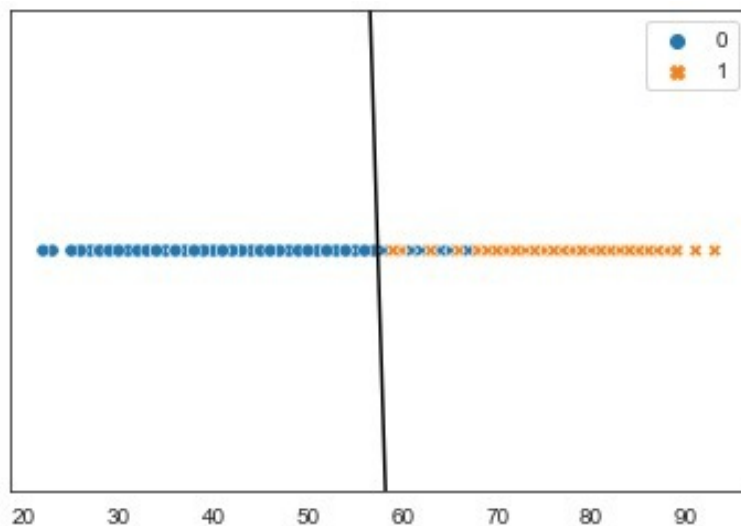


Figura 8: Resultado final - estudantes

```
Out[111...  
Pesos(%)  
-----  
pretest      1.0
```

- **Vinhos**

Nem toda aplicação de regressão logística pode ser perfeitamente transformada em um exemplo visual bidimensional. Nesta análise, por exemplo, de previsão de avaliação de vinhos, isso foi impossível. Isso porque o algoritmo não dava resultados suficientemente precisos utilizando apenas 2 variáveis numéricas, então utilizamos 7 das colunas disponíveis para tentar prever quais vinhos receberam nota superior a 6 (nota considerada muito boa - sommeliers são muito mesquinhos com notas).

Se utilizando 2 variáveis nosso ápice de precisão foi 54%, utilizando essas 7 variáveis escolhidas a dedo conseguimos atingir 74.6% de precisão. A desvantagem nesse caso é que, por usarmos 7 variáveis, não é possível mostrar um belo gráfico em sete dimensões que mostre a melhor decisão da regressão logística, então só podemos nos contentar com a exibição dos pesos escolhidos para cada coluna. Dessa tabela, é interessante perceber, por exemplo, o quanto a quantidade de álcool está positivamente relacionada com a nota.

Figura 9: Resultado final - vinhos

Out[20]:	Pesos(%)
fixed acidity	-0.023601
volatile acidity	-0.130716
total sulfur dioxide	-0.046467
density	-0.136581
pH	-0.088948
sulphates	0.091823
alcohol	0.481864

Além disso, é importante notar que, embora o resultado usando 7 variáveis tenha sido muito mais preciso que o resultado usando 2, nem sempre aumentar o número de variáveis significa um aumento de precisão. Nesse mesmo exemplo, se usássemos todas as 11 colunas numéricas disponíveis na base, nosso ápice de precisão seria um pouco menor (73%). Isso acontecia porque as outras variáveis, como, por exemplo, “chlorides”, tinham correlação real nula com a nota dos vinhos, mas, como o algoritmo sempre busca encontrar um padrão, essa variável recebia um peso (ainda que pequeno) que prejudicava a qualidade da previsão.

Diante dessa situação, nosso método foi utilizar inicialmente todas as colunas na regressão logística e depois selecionar apenas as que tinham um peso de 3% ou mais em relação ao total.

- **Câncer de mama**

Este exemplo é muito famoso no campo da regressão logística, devido a isso, essa base já possui uma *learning rate* ótima conhecida e se tornou substancialmente previsível. Além disso, é um ótimo retrato da importância dos modelos de predição para a sociedade.

Os dados dessa base foram calculados a partir de uma imagem digitalizada de uma massa mamária e descrevem as características dos núcleos celulares presentes na imagem. É importante ressaltar que apesar de nenhuma característica por si só ter uma influência excepcional no resultado final, algumas podem ser destacadas, como as propriedades da área e do perímetro dos núcleos. A precisão desse modelo é de 90%.

Figura 10: Resultado final - câncer de mama

Out[6]:	Pesos		compactness error	-0.000040
	mean radius	0.038284	concavity error	-0.000078
	mean texture	0.054792	concave points error	-0.000008
	mean perimeter	0.223411	symmetry error	0.000060
	mean area	0.101491	fractal dimension error	0.000005
	mean smoothness	0.000358	worst radius	0.040296
	mean compactness	-0.000134	worst texture	0.069340
	mean concavity	-0.000659	worst perimeter	0.225760
	mean concave points	-0.000282	worst area	-0.138987
	mean symmetry	0.000681	worst smoothness	0.000454
	mean fractal dimension	0.000283	worst compactness	-0.000617
	radius error	0.000143	worst concavity	-0.001361
	texture error	0.003648	worst concave points	-0.000310
	perimeter error	-0.000995	worst symmetry	0.000937
	area error	-0.096296	worst fractal dimension	0.000268
	smoothness error	0.000020		

3 Considerações Finais

Este trabalho se propôs, como objetivo geral, mostrar aplicações da regressão logística através de diferentes bases de dados. Desta forma, é possível concluir que é possível utilizar essa técnica em inúmeras áreas, onde ela sempre é destacada como uma importante ferramenta de análise de dados.

4 Referências

- [1] a. [Towards Data Science - Logistic Regression from Scratch with NumPy](#)
- [2] b. [Leando Gonzales - Regressão Logística e suas aplicações](#)