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

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 criado uma classe de regressão logística em python.

```
class LogisticRegression:
    def __init__(self, learning_rate=0.001, n_iters=1000):
        self.lr = learning_rate
        self.n_iters = n_iters
        self.weights = None
        self.bias = None

    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features) #parametro inicial
        self.bias = 0 #parametro inicial

        #gradiente descendente
        for _ in range(self.n_iters):
            linear_model = np.dot(X, self.weights) + self.bias #
                aproxima y com a combinacao linear dos pesos e x
                somada a constante
            y_predicted = self._sigmoid(linear_model) #aplica a
                funcao sigmoide

            #computa os gradientes
            dw = (1 / n_samples) * np.dot(X.T, (y_predicted - y)
                )
            db = (1 / n_samples) * np.sum(y_predicted - y)
            #atualiza os parametros
            self.weights -= self.lr * dw
            self.bias -= self.lr * db

    def predict(self, X):
        linear_model = np.dot(X, self.weights) + self.bias
        y_predicted = self._sigmoid(linear_model)
        y_predicted_cls = [1 if i > 0.5 else 0 for i in
            y_predicted]

        return np.array(y_predicted_cls)

    def _sigmoid(self, x):
        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.

```
def plot(self, X, y, legend):
        # essa funcao plota o resultado apenas se X se referir a
            exatamente 2 variaveis
        if X.shape[1] != 2:
            raise ValueError("Can plot only for X's that refers
                to exactly 2 vars.")

        slope = -(self.weights[0]/self.weights[1])
        intercept = -(self.bias/self.weights[1])
        predictions = self.predict(X)

        sns.set_style('white')
        sns.scatterplot(x = X[:,0], y= X[:,1], hue=y.reshape(-1)
            , style=predictions.reshape(-1));

        ax = plt.gca()
        ax.autoscale(False)
        x_vals = np.array(ax.get_xlim())
        y_vals = intercept + (slope * x_vals)
        plt.plot(x_vals, y_vals, c="k");

        plt.xlabel(legend[0])
        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")

df = df[(df['Spectral Class'] == 'B') | (df['Spectral Class'] == 'M')]
df['Spectral Class'].replace(to_replace='B', value=1, inplace=True)
df['Spectral Class'].replace(to_replace='M', value=0, inplace=True)

# Sele  o de Dados
dados = {
    'X' : ['Temperature (K)', 'Absolute magnitude(Mv)'],
    'y' : 'Spectral Class',
    'normalizada' : False
}

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

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

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

df.sample(5)
```

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

```
norma_pesos = pd.DataFrame(regressor.weights)/pd.DataFrame(
    regressor.weights).abs().sum()
norma_pesos = norma_pesos[0].values.tolist()

dfpesos = pd.DataFrame({'Pesos':norma_pesos}, index=dados['X'])

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.

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

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

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

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

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

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

df.sample(5)
```

Figura 2: Dados selecionados

Out[4]:

	Temperature (K)	Radius(R/Ro)	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

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

```
regressor = LogisticRegression(learning_rate=0.000001, n_iters
    =2000)
regressor.fit(X, y)
predictions = regressor.predict(X)

```

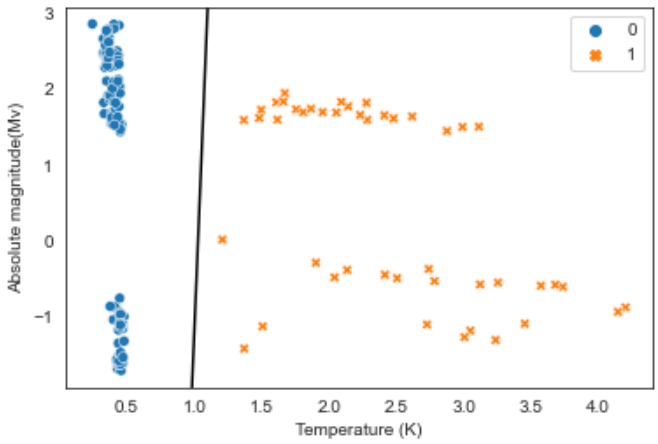
```
def accuracy(y_true, y_pred):
    accuracy = np.sum(y_true == y_pred) / len(y_true)
    return accuracy

print(f"A precis o do modelo   : {accuracy(y, predictions)}")
```

Por fim, foi feita uma visualização dessa regressão.

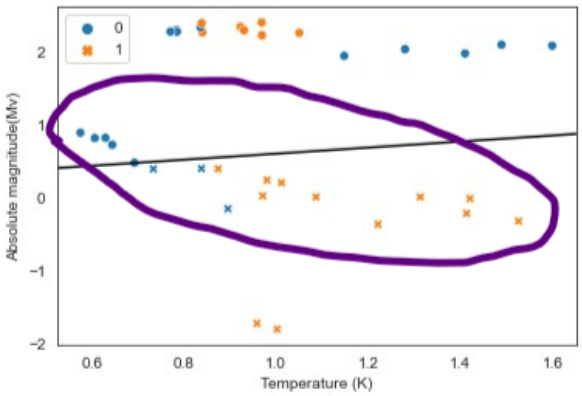
```
try:
    regressor.plot(X, y, dados['X'])
except:
    print("Sem visualiza  o dispon vel.")
```

Figura 3: Estrelas - Primeiro Caso



No segundo caso, percebemos o que acontece. Nessa base, apenas 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 4: Estrelas - Segundo Caso



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 5: 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, já que o resultado foi 72.22% preciso, 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 6: Gráfico sobre os estudantes

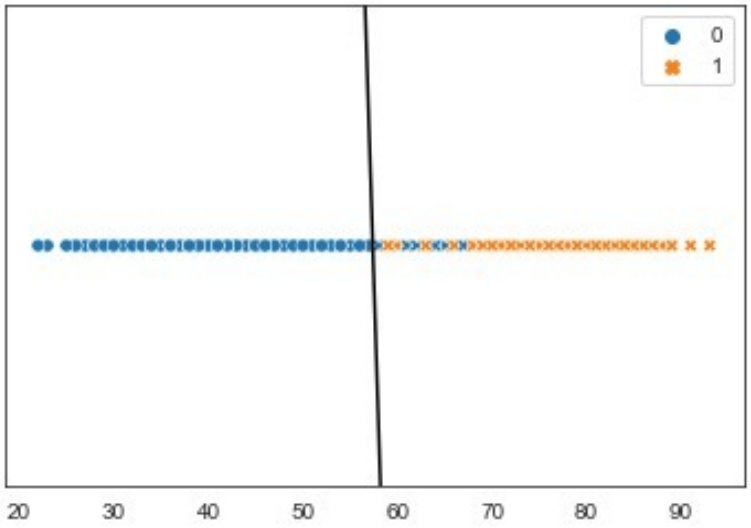


Figura 7: Resultado final - estudantes

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 8: 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 9: Resultado final - câncer de mama

Out[6]:

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

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