Prática de modelagem com regressão logística para previsões de classificação binária

Breno Pedrosa Gil Araújo João Victor Russo

2023

Resumo

A utilização de métodos de estimativas para entender o comportamento de uma variável dependente é bastante útil em diversas áreas. Este artigo demonstra o funcionamento do método de Regressão Logística para Categorização Binária. Foi utilizado um conjunto de dados médicos e demográficos públicos que tem como variável dependente a presença da diabetes no paciente. Foi gerado um modelo cujo objetivo é prever se um paciente pode ou não ser diabético, com base nas informações disponíveis.

Palavras-chave: Regressão Logística, Estimativa de Resultado, Classificação Binária, Previsão de Dados.

Abstract

The use of estimation methods to understand the behavior of a given datapoint is very useful in various fields, including the field of Health. This article demonstrates the use of Logistic Regression method for Binary Categorization. A publicly available dataset of medical and demographic data was used, with the dependent variable being the presence of diabetes in the patient. A logit regression model was generated with the objective of predicting whether a patient may or may not be diabetic based on the available information.

Keywords: Logistic Regression, Outcome Estimation, Binary Classification, Data Prediction.

Introdução

O objetivo deste artigo é descrever o passo a passo de como construir, treinar e utilizar um modelo de regressão logística para prever respostas à questões de classificação binária. Fazendo uso de uma base de dados médicos e demográficos de 100 mil pacientes, composta de 9 variáveis, para construir e treinar um modelo de regressão logística capaz de prever a presença ou não de diabetes com 95,98% de precisão.

Base de dados

A base de dados usada encontra-se no repositório público de datasets *Kaggle*. A base de previsão da diabetes é uma coleção de dados médicos e demográficos de pacientes, bem como seu status de diabetes (positivo ou negativo). Os dados incluem características como idade, gênero, índice de massa corporal (IMC), hipertensão, doença cardíaca, histórico de tabagismo e níveis de HbA1c e glicose no sangue. Esta base foi escolhida por sua facilidade de manuseio, sendo necessário apenas mínimos ajustes para encaixe do modelo.

Para garantir o funcionamento do código, precisamos nos certificar que o dataset não possui dados faltantes, para isso usamos a biblioteca Pandas, do Python, carregamos o arquivo .csv e atribuímos estes dados a um DataFrame do Pandas:

```
df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/diabetes_study/diabetes_prediction_dataset.csv")
df.head()
```

	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_glucose_level	diabetes
0	Female	80.0	0	1	never	25.19	6.6	140	0
1	Female	54.0	0	0	No Info	27.32	6.6	80	0
2	Male	28.0	0	0	never	27.32	5.7	158	0
3	Female	36.0	0	0	current	23.45	5.0	155	0
4	Male	76.0	1	1	current	20.14	4.8	155	0

Depois, checamos se há algum dado faltando em cada característica utilizando a função isnull():

```
df.isnull().sum()
```

```
gender                0
age                   0
hypertension          0
heart_disease         0
smoking_history       0
bmi                   0
HbA1c_level           0
blood_glucose_level   0
diabetes              0
dtype: int64
```

O *Output* nos diz que nenhuma característica tem valores nulos, então não será necessário lidar com dados faltantes para que o treinamento do modelo funcione.

A função *describe()* nos dá um panorama estatístico da base de dados

df.describe()

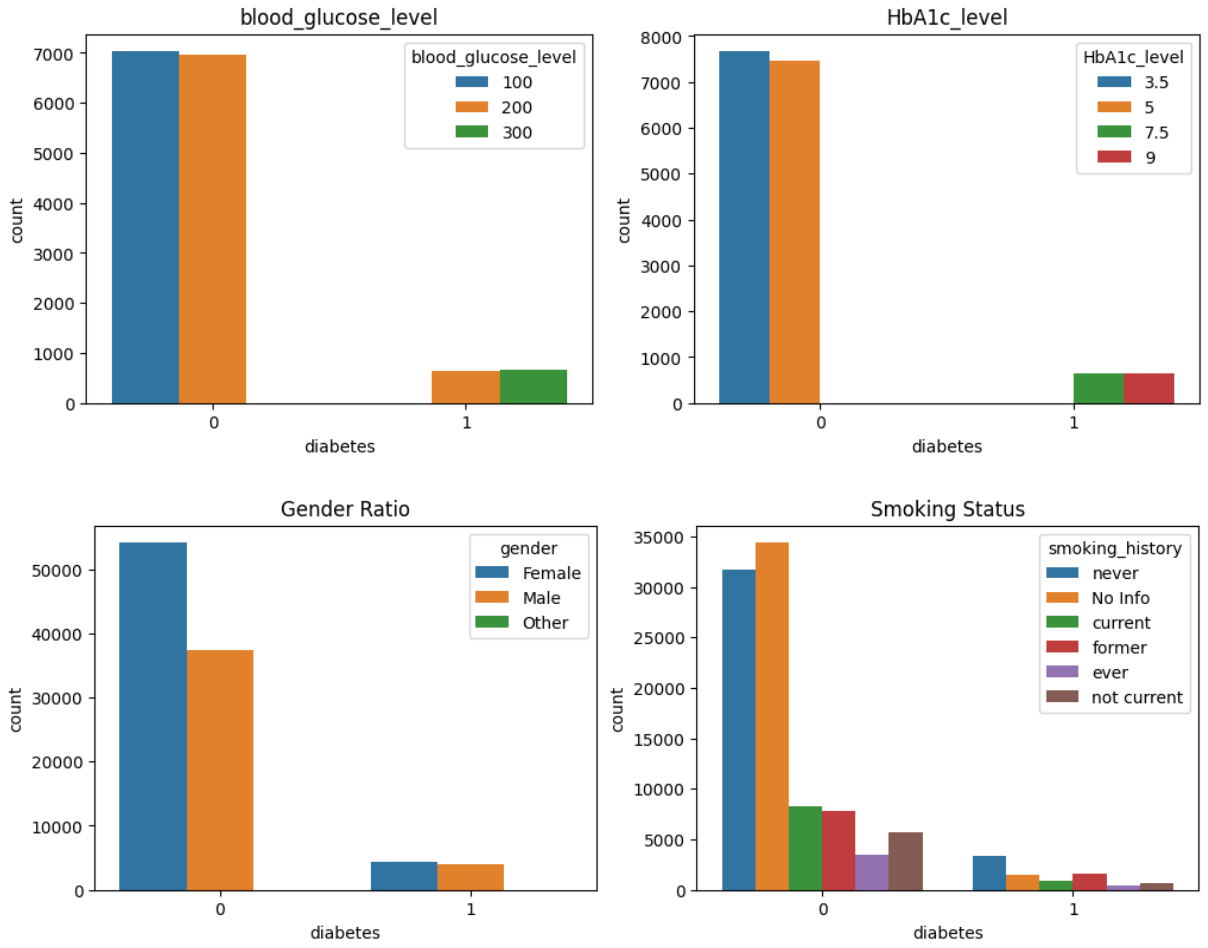
	age	hypertension	heart_disease	bmi	HbA1c_level	blood_glucose_level	diabetes
count	100000.000000	100000.00000	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000
mean	41.885856	0.07485	0.039420	27.320767	5.527507	138.058060	0.085000
std	22.516840	0.26315	0.194593	6.636783	1.070672	40.708136	0.278883
min	0.080000	0.00000	0.000000	10.010000	3.500000	80.000000	0.000000
25%	24.000000	0.00000	0.000000	23.630000	4.800000	100.000000	0.000000
50%	43.000000	0.00000	0.000000	27.320000	5.800000	140.000000	0.000000
75%	60.000000	0.00000	0.000000	29.580000	6.200000	159.000000	0.000000
max	80.000000	1.00000	1.000000	95.690000	9.000000	300.000000	1.000000

Rodando a função *value_counts()* na característica 'diabetes' temos que 91500 dos 100 mil pacientes têm o diagnóstico da diabetes negativo e 8500 deles positivo

```
df['diabetes'].value_counts()
```

```
0    91500
1     8500
Name: diabetes, dtype: int64
```

Gráficos são uma boa forma de visualizar quais características poderão ter mais peso no cálculo da regressão. Eis os Gráficos *countplot* das características utilizando as bibliotecas matplotlib e seaborn, que nos dão mais informações sobre cada coluna



Para que o modelo consiga ser utilizado no código, é preciso associar os dados de variáveis qualitativas a números. As colunas 'gender' e 'smoking_history' do dataset representam dados de gênero e histórico de fumante, que precisam ser convertidos.

```
#substitute string data to int
df['gender'].replace({'Female':0, 'Male':1, 'Other':2},
inplace=True)
df['smoking_history'].replace({'No Info':0, 'current':1, 'ever':2,
'former':3, 'never':4, 'not current':5}, inplace=True)
#divide X and y features
y = df.pop('diabetes')
X = df
```

também é preciso dividir a base de dados em amostras de treinamento e de testes, bem como separar a variável dependente que queremos prever - neste caso, a coluna 'diabetes' - dos demais.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.20, random_state=0) #split data into training and
testing datasets
```

Parte importante do pré-processamento dos dados é saber se estes estão normalizados. Dados devidamente padronizados não foram essenciais para o funcionamento do modelo, mas o uso da função StandardScaler() da biblioteca ScikitLearn nas amostras de treinamento promoveu um aumento de quase 5% na precisão do nosso modelo - de 91,5% de precisão nas previsões para 96%.

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Modelo

A regressão logística é um método estatístico utilizado para problemas de classificação binária, onde o objetivo é prever a probabilidade de uma dada característica estar em alguma classe (tipicamente denotado por 1 ou 0). Isto é obtido fazendo uma regressão linear e aplicando seu resultado numa função sigmóide.

A regressão linear, pressupõe uma relação linear entre as características de entrada e a variável dependente. O objetivo da regressão linear é encontrar a reta de melhor ajuste que minimize a diferença entre os valores previstos e os valores reais da variável dependente. Matematicamente, a regressão linear pode ser representada por:

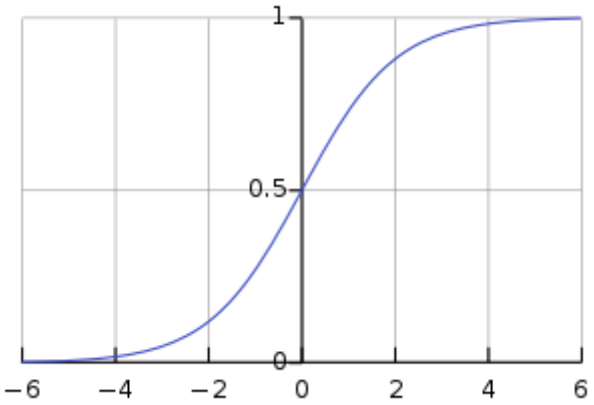
$$\hat{y} = b + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$$

onde $\hat{y}$ é a variável alvo (a variável a ser prevista), b é o *bias* ou viés, $w1, w2, \ldots, wn$ são os coeficientes (pesos) associados a cada característica de entrada $x1, x2, \ldots, xn$, que são as características de entrada

isto é então aplicado à função sigmóide, ou função logística,

$$f(x) = \frac{1}{1+e^{-x}}$$

onde x é a equação da reta $\hat{y} = w * x + b$ encontrada pelo cálculo da regressão linear. Esta função transforma a função linear em um valor de probabilidade entre 0 e 1, demonstrada pelo gráfico abaixo:



onde podemos inferir uma classificação binária a partir do resultado da função, obtendo 0 caso $f(x) < 0.5$, e 1 caso contrário.

Para a regressão logística é preciso chegar a uma função onde os pesos e viés estejam o mais próximo possível do resultado real. Para isso precisamos calcular os erros dessas variáveis, calculando a entropia cruzada ou log loss, que pode ser simplificada como

$dw = \frac{1}{N}\sum 2x_i(\hat{y} - y_i)$ para o erro das características x_i

$db = \frac{1}{N}\sum 2(\hat{y} - y_i)$ para o erro relacionado ao viés

onde N = número de amostras e y_i é o valor obtido pela função sigmóide

Com as funções erro, podemos atualizar as variáveis de pesos e viés com a técnica de gradiente descendente, que conjuntamente com a taxa de aprendizado - se aplicada corretamente -, irá convergir aos erros minimizados dos pesos(w) e viés(b), completando o treinamento do modelo de regressão logística.

O gradiente descendente, aplicado aos pesos e viés, é denotado como

$w = w - \alpha \cdot dw$

$b = b - \alpha \cdot db$

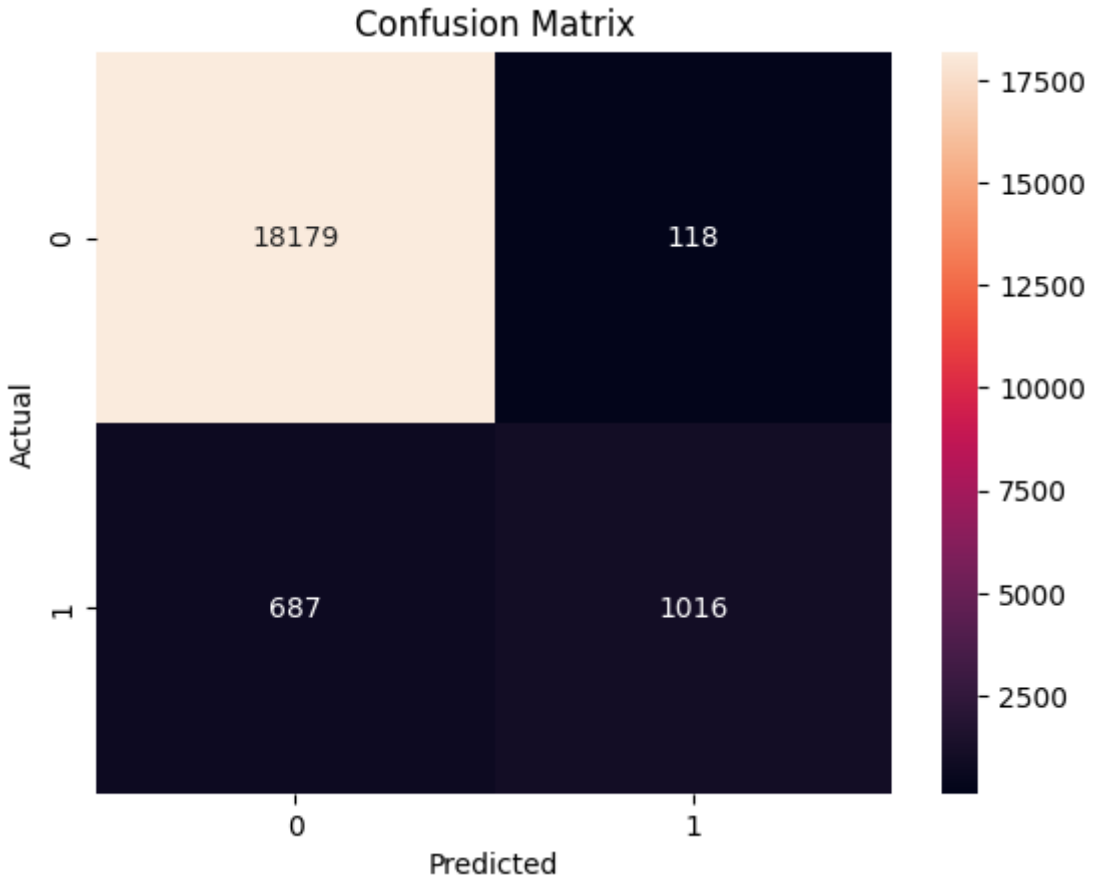
onde α é a taxa de aprendizado que o modelador irá escolher para o seu treinamento. Estas equações vão atualizando os pesos e viés, convergindo estes valores para seus mínimos globais.

Todo o processo é descrito pelo seguinte código:

```
def sigmoid(x):
  return 1/(1+np.exp(-x))

#create model class
class LogisticRegression():
    #class constructor
    def __init__(self, learning_rate=0.01, iterations=1000):
        self.learning_rate = learning_rate
        self.iterations = iterations
        self.weights = None
        self.bias = None

    def fit(self, X, y):
        #initializing regression model
        samples, features = X.shape
        self.weights = np.zeros(features)
        self.bias = 0

        for _ in range(self.iterations):

            #prediction update
            linear_prediction = np.dot(X, self.weights) + self.bias
            predictions = sigmoid(linear_prediction)

            #binary cross-entropy loss functions (error functions)
            error_weight = (1/samples) * np.dot(X.T, (predictions -
y))
            error_bias = (1/samples) * np.sum(predictions-y)

            #gradient descent updates
            self.weights = self.weights -
self.learning_rate*error_weight
            self.bias = self.bias - self.learning_rate*error_bias

    def predict(self, X):
        linear_prediction = np.dot(X, self.weights) + self.bias
        y_prediction = sigmoid(linear_prediction)
        binary_classifier = [0 if y<=0.5 else 1 for y in
y_prediction]
        return binary_classifier
```

Resultados

Rodando a classe do modelo no numa amostra de testes, com 20 mil amostras, obtemos os seguintes resultados, demonstrados pela matriz de confusão abaixo:



Conseguimos observar que o modelo obteve 18179 verdadeiros negativos e 1016 verdadeiros positivos, com 687 falsos negativos e 118 falsos positivos, alcançando 96% de precisão para previsão da diabetes baseado nas variáveis médicas e demográficas descritas na base de dados.

Conclusão

A prática do modelo de regressão logística é importante pois a experiência do modelador nas etapas do processo influenciam no resultado final da função de previsão, como observado no uso da normalização da base de dados para melhorar a precisão do modelo. Um modelador experiente pode observar mais otimizações ao longo do processo inteiro, que podem influenciar positivamente na confiabilidade do modelo treinado.

Referências

pandas.Series.value_counts. pandas, 2023. Disponível em: https://pandas.pydata.org/docs/reference/api/pandas.Series.value_counts.html . Acesso em: 6 de jun. de 2023.

pandas.read_csv. pandas, 2023. Disponível em: https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html . Acesso em: 6 de jun. de 2023.

sklearn.preprocessing.StandardScaler. scikit-learn, 2023. Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html . Acesso em: 6 de jun. de 2023.

pandas.DataFrame. pandas, 2023. Disponível em: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html . Acesso em: 8 de jun. de 2023

numpy.zeros. numpy, 2023. Disponível em: https://numpy.org/doc/stable/reference/generated/numpy.zeros.html . Acesso em: 8 de jun. de 2023

numpy.dot. numpy, 2023. Disponível em: https://numpy.org/doc/stable/reference/generated/numpy.dot.html . Acesso em: 8 de jun. de 2023

numpy.sum. numpy, 2023. Disponível em: https://numpy.org/doc/stable/reference/generated/numpy.sum.html . Acesso em: 8 de jun. de 2023

pandas.DataFrame.head. pandas, 2023. Disponível em: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.head.html . Acesso em: 6 de jun. de 2023.