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
```

Nesse exemplo, foram comparadas as estrelas brancas (A ou 1) com as estrelas branco-amareladas (F ou 0), com o intuito de prever a qual dessas classes espectrais as estrelas pertencem a partir de sua temperatura e sua magnitude absoluta (magnitude aparente de um objeto celeste a uma distância padrão de 10 parsecs do observador).

```
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)
```

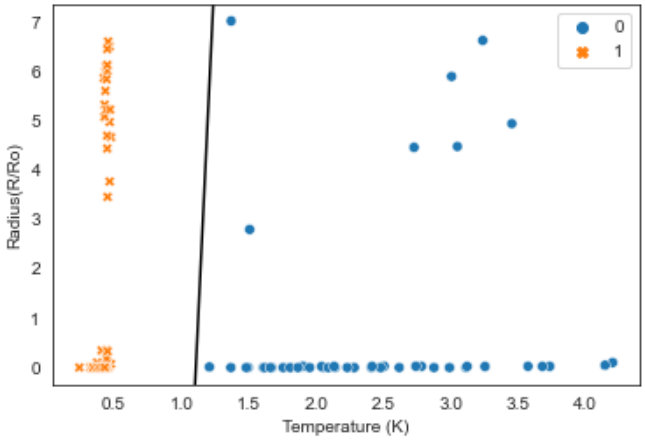
Após isso, a regressão logística foi aplicada, a precisão da previsão foi avaliada e os pesos de influência de cada variável numérica foram calculados.

```
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)}")

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
```

Por fim, foi feita uma visualização dessa regressão.

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

Figura 1: Regressão linear - Resultado Final



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