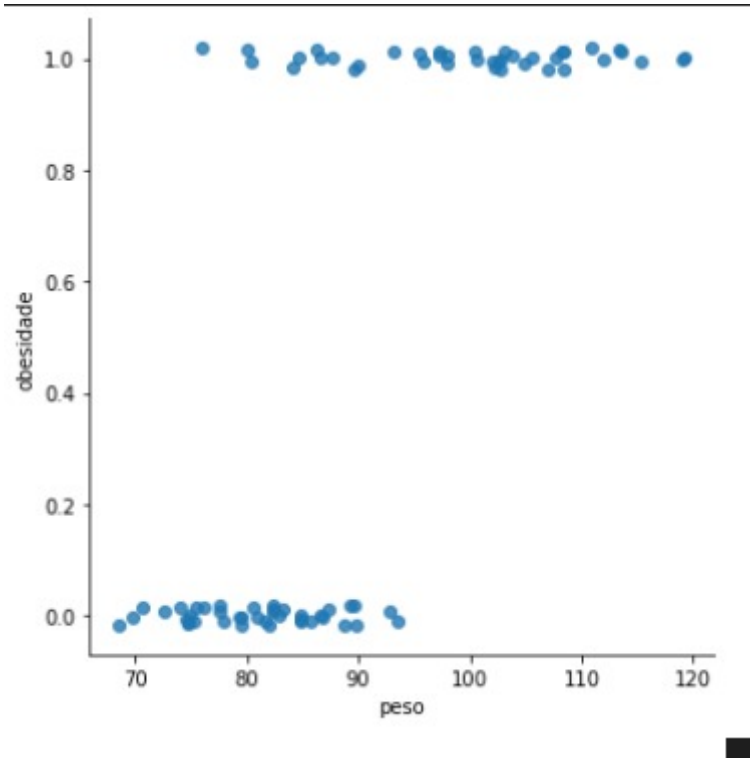
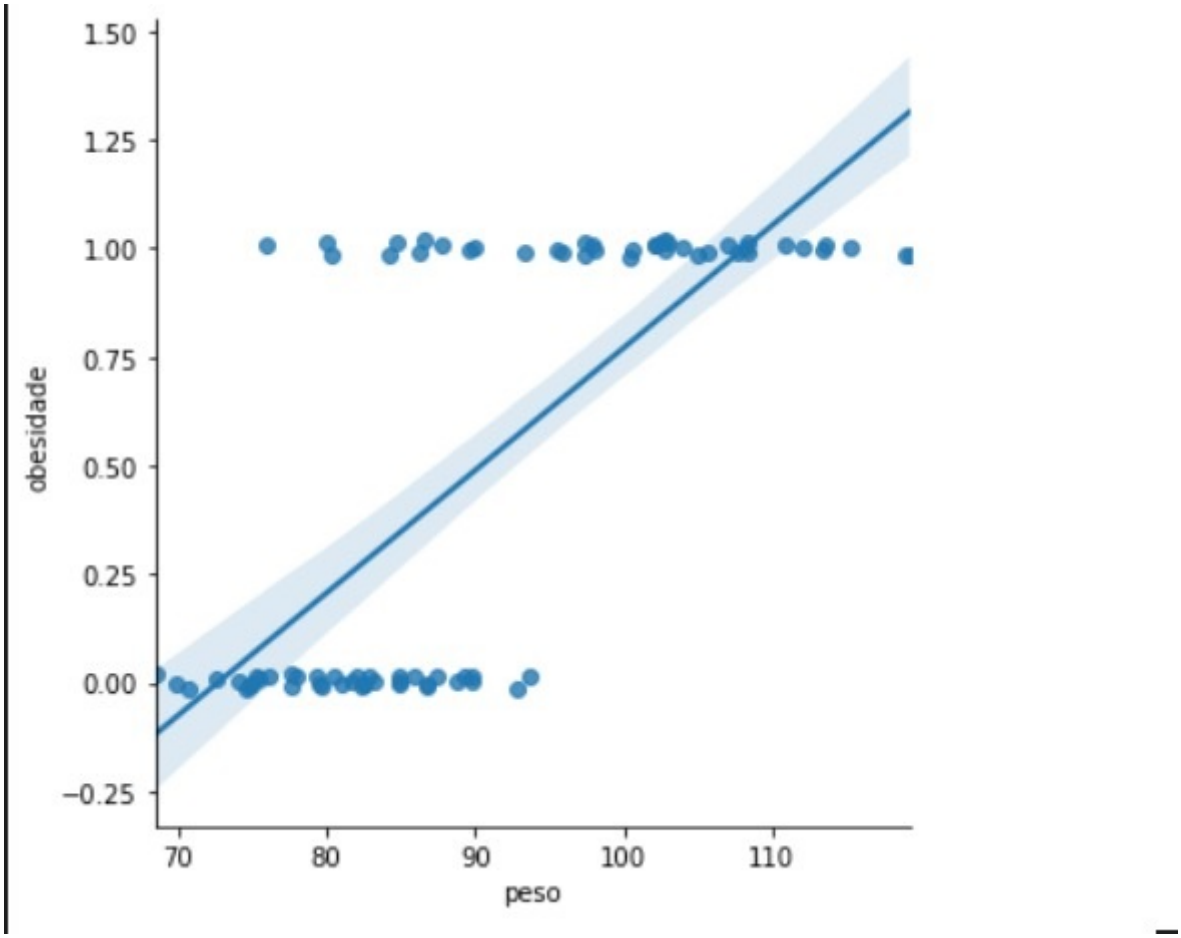
Introdução

A Regressão Linear é muito útil, porém não funciona para problemas de classificação. Um problema de classificação é quando se deseja classificar um objeto em uma de várias categorias através de uma ou mais características que se sabe sobre o objeto.

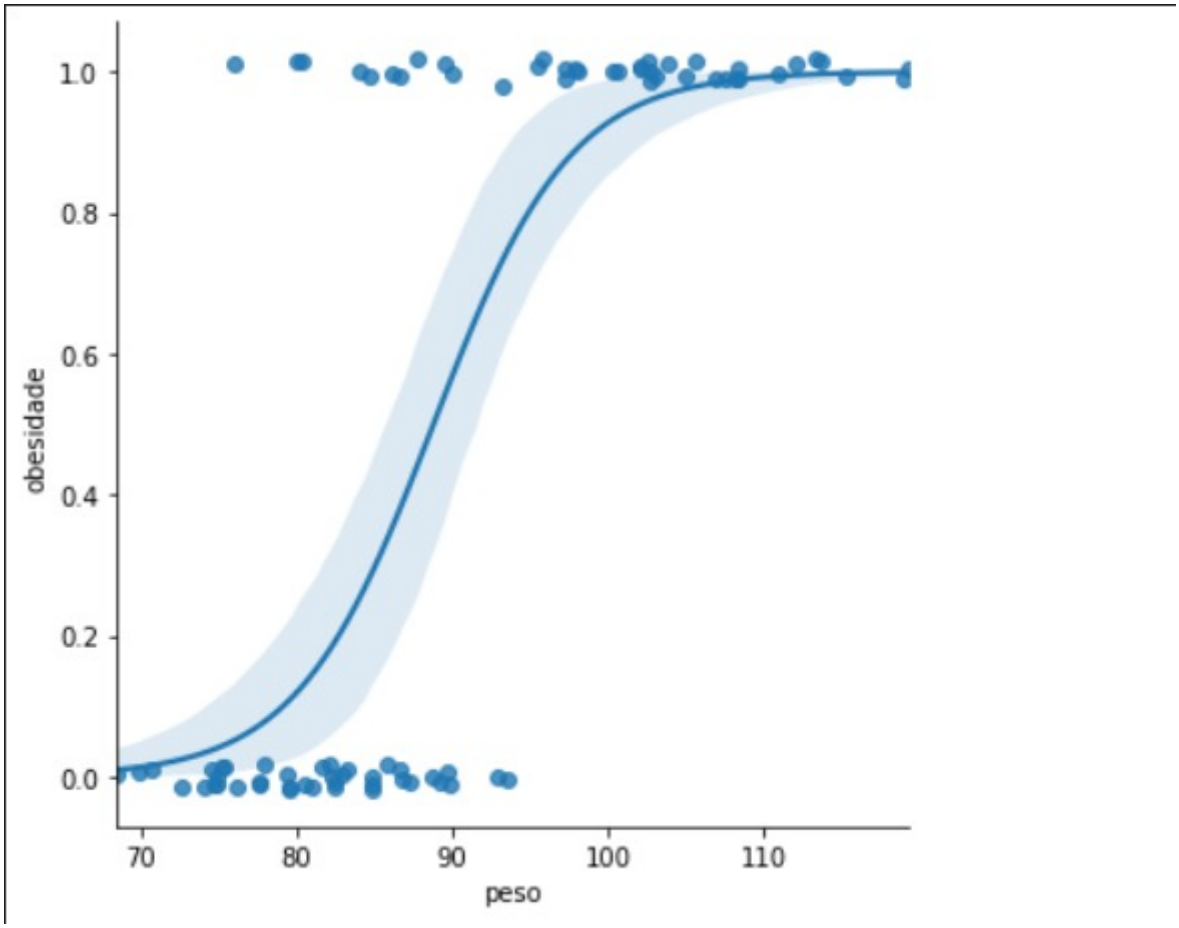
Um exemplo simples deste tipo de problema é quando se deseja prever se uma pessoa é ou não obesa apenas sabendo seu peso. O gráfico abaixo contém as ocorrências de 80 pessoas que possuem um determinado peso e podem ou não ser obesas.



Aplicando uma regressão linear, prevê-se para algumas pessoas uma probabilidade maior que 1, oque é impossível.



Para evitar isso, projeta-se uma regressão logística, que produz um resultado mais adequado.



Como funciona

A regressão logística é versão da regressão linear adequada a problemas de classificação. Para se adequar à este tipo de problema ao invés de prever quanto algo vai ser em uma escala contínua, a regressão logística prevê a probabilidade de algo se encaixar em uma categoria. Esta probabilidade, como qualquer outra, varia entre 0 e 1. para se manter dentro do contradomínio, a regressão logística usa a função Sigmóide:

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

A função Sigmóide tem o seu formato definido pela função $f(x)$, ou seja, para adequar a função logística do jeito desejado é necessário alterar a função $f(x)$. Uma função $f(x)$ simples utilizada na regressão logística é a seguinte função linear:

$f(x) = w_0 + w_1x_1 + w_2x_2 + \ldots + w_nx_n,$

onde cada x_i é uma característica sendo observada e cada w_i é um peso atribuido à característica x_i. Também são definidos $X_{n\times1} = [1, x_1, \ldots, x_n]$ e $W_{n\times1} = [w_0, w_1, \ldots, w_n]$. Assim sendo, tem-se que $f(x) = W^TX$

Para já acostumar o leitor, vamos escrever a função Sigmóide da forma como ela será usada mais à frente:

$p(X) = \frac{1}{1+e^{-W^TX}}$

Dessa forma, o objetivo se torna encontrar a matriz W que gera os resultados mais próximos do esperado.

Fica claro que há casos onde $p(X)$ (valor previsto) é igual ao y (valor real). Uma das formas de calcular o acerto é através da fórmula de INSERT (Máxima Verosemelhança?) NOME. Ela funciona da seguinte forma:

1 Separa os objetos em objetos com $y = 1$ e objetos com $y = 0$. O X de cada determinado objeto será aqui chamado apenas de X 2 Para objetos com $y = 1$, busca-se W tal que $p(X)$ fique o mais próximo possível de 1 3 Para objetos com $y = 0$, busca-se W tal que $p(X)$ fique o mais próximo possível de 0 3* Em outras palavras, para objetos com $y = 0$ busca-se W tal que $1 - p(X)$ fique o mais próximo possível de 1 4 Ao se acumular as probabilidades, temos o produtório de ambos objetos com $y = 1$ ou $y = 0$ deve ser igual a 1 da seguinte forma:

$\prod p(x)$ para $y = 1$ e $\prod(1 - p(x))$ para $y = 0$

5 É possível juntar os dois produtórios, aplicando um expoente y ou $y - 1$ à cada um. O papel deste expoente é garantir que quando cada produtório somente tenha impacto quando for aplicável:

$L(W) = \prod p(x)^y(1 - p(x))^{(1-y)}$

note que $y = 1 \implies (1 - p(x))^{(1-y)} = 1$ e $y = 0 \implies p(x)^y = 1$

Esta é função permite avaliar quão boa é a matriz W. Portanto, ao se maximizar esta função se otimiza a regressão logística!

6 É possível trabalhar esta função aplicando $\log$, que então transforma o produtório em somatório:

$\log L(W) = \log(\prod p(X)^y(1-p(X))^{(1-y)})$

$l(W) = \sum y \log p(X) + (1-y)\log(1-p(X))$

7 Para continuar, é necessário expandir $p(X)$:

$l(W) = \sum y \log \frac{1}{1+e^{-W^TX}} + (1-y)\log(1-\frac{1}{1+e^{-W^TX}}) = \sum y \log \frac{1}{1+e^{-W^TX}} + (1-y)\log(\frac{e^{-W^TX}}{1+e^{-W^TX}}$

8 Efetua-se a multiplicação $(1-y)\log(\frac{e^{-W^TX}}{1+e^{-W^TX}})$:

$\sum y \log \frac{1}{1+e^{-W^TX}} - y\log(\frac{e^{-W^TX}}{1+e^{-W^TX}}) + \log(\frac{e^{-W^TX}}{1+e^{-W^TX}})$

9 Agrupa-se onde há coeficiente y:

$\sum y[\log \frac{1}{1+e^{-W^TX}} - \log(\frac{e^{-W^TX}}{1+e^{-W^TX}})] + \log(\frac{e^{-W^TX}}{1+e^{-W^TX}})$

10 Utiliza-se a propriedade da subtração de dois logarítmos:

$\sum y \log(\frac{1}{1+e^{-W^TX}} * \frac{1+e^{-W^TX}}{e^{-W^TX}}) + \log(\frac{e^{-W^TX}}{1+e^{-W^TX}}) = \sum y \log(\frac{1}{e^{-W^TX}}) + \log(\frac{e^{-W^TX}}{1+e^{-W^TX}})$

11 Altera-se as funções dentro dos logarítmos mantendo a igualdade:

$\sum y \log(\frac{1}{e^{W^TX}}^{-1}) + \log(\frac{e^{-W^TX}}{1+e^{-W^TX}\prod \frac{e^{W^TX}}{e^{W^TX}}}) = \sum y \log(e^{W^TX}) + \log(\frac{1}{1+e^{W^TX}})$

12 Finalizando:

$l(W) = \sum yW^TX - \log(1+e^{W^TX})$

Assim, busca-se encontrar W que maximiza $l(W)$

Algoritmo

Idealmente seria possível encontrar o máximo de $l(W)$ de maneira "simples", porém a função em questão não é algébrica e sim transcendental. Isto ocorre por causa da função $\log$, e significa que não é possível calcular seu máximo de maneira exata e "instantânea". Contudo existem métodos de aproximação, e o método aqui mostrado será o Método de Newton Raphson.

O Método de Newton Raphson atua começando com uma variável independente arbitrária e então encontrando um novo valor para esta variável independente através da divisão do valor da função sendo analisada pela sua derivada:

$x_{n+1} = x_n - \frac{\partial f(x_n)}{\partial f(x_n)}$

Ou seja, dado um W_n, haverá um $W_{n+1} = W_n + gradiente$. Assim sendo, a equação que precisa ser iterada para encontrar o valor desejado para W é:

$W_{n+1} = (X^TW_nX)^{-1}X(Y-\hat{Y}_n)$,

onde W_n são os parâmetros na iteração n, X são as características sendo observadas, Y é o valor de y e $\hat{Y}_n$ é o valor previsto para y.

Assim, se começa com um W arbitrário (comumente $W = 1$) e então se caminha até chegar em um resultado satisfatório

Rodando o algoritmo:

In [ ]:
```
# importando bibliotecas
import pandas as pd
import numpy as np
import plotly.express as px
from sklearn.linear_model import LogisticRegression
```

In [ ]:
```
# abrindo base de dados
df = pd.read_csv("diabetes.csv")
```

In [ ]:
```
# Escolhendo quais serão as variáveis independentes e a variável dependente sendo ob

X = df.iloc[:, df.columns!="Outcome"]
y = df["Outcome"]
```

In [ ]:
```
# Rodando o algorítmo
lr = LogisticRegression(random_state=0, max_iter=1000).fit(df.loc[:, df.columns!="Ou
```

In [ ]:
```
# Calculando o y esperado
y_esperado = lr.predict(X.loc[:, :])
```

In [ ]:
```
# Calculando a probabilidade de y
y_prob = lr.predict_proba(X.loc[:, :])
```

In [ ]:
```
# Precisão do algoritmo
lr.score(X,y)
```

Out[ ]:
```
0.78125
```

In [ ]:
```
n_df = pd.concat([X,y], axis=1)
n_df["y_esperado"] = y_esperado
n_df["y_prob"] = [y[0] for y in y_prob]
n_df
```

Out[ ]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	A
0	6	148	72	35	0	33.6	0.627	
1	1	85	66	29	0	26.6	0.351	
2	8	183	64	0	0	23.3	0.672	
3	1	89	66	23	94	28.1	0.167	
4	0	137	40	35	168	43.1	2.288	
...	...	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	0.171	
764	2	122	70	27	0	36.8	0.340	
765	5	121	72	23	112	26.2	0.245	

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	A
766	1	126	60	0	0	30.1	0.349	
767	1	93	70	31	0	30.4	0.315	

768 rows × 11 columns

Conclusão

O algoritmo teve uma precisão de 78%