> Na aula inicial, examinamos brevemente o problema de tentar prever quais usuários da *Facedata* pagam para ter contas *premium*. Vamos revisitar esse problema, entender rapidamente o conceito de probabilidade e decidir quando aplicar a regressão logística.

Introdução

Temos um conjunto de dados anônimo de 200 usuários, contendo seus anos de experiência como cientista de dados e se pagam por uma conta *premium*. Como é típico com variáveis categóricas, representaremos o *output* como 0 (conta gratuita) ou 1 (conta paga). Vamos preparar nossos dados:

In [ ]:
```python
# Carregando bibliotecas
import pandas as pd
import matplotlib.pyplot as plt

# Dados
experiencia = [6.8, 3.9, 0.1, 3.4, ..., 10.6,  10.3, 11.5,  8.9]
tipo_conta = [0, 0, 0, 0, ..., 1, 1, 1, 1]

# Transformando para pd.DataFrame
conta = pd.DataFrame({
    "experiencia": experiencia,
    "tipo_conta": tipo_conta
})
```

Uma primeira tentativa óbvia é usar a regressão linear e encontrar o melhor modelo:

$$\text{conta paga} = \alpha + \beta \times \text{experiência} + \varepsilon$$

Embora nosso *output* seja um atributo **quantitativo discreto**, não há nada que nos impeça de modelar o problema dessa maneira no `Python`.

> Por essas e outras situações, **a teoria é tão importante quanto a prática**. O que irá diferenciar o trabalho de um cientista de dados é a tomada de decisão melhor do que de uma inteligência artificial.

Recuperando os comandos utilizados anteriormente, vamos aplicar uma regressão linear nestes dados utilizando:

In [ ]:
```
import numpy as np
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm

# Definindo input e output - treinamento
X_treino = sm.add_constant(conta['experiencia'])
y_treino = conta['tipo_conta']

# Definindo o modelo de regressao linear
reg = LinearRegression()

# Estimando o modelo
modelo_estimado = reg.fit(X = X_treino,
                          y = y_treino)
```

Neste exemplo, não nos atentamos a separar a base de dados em `treinamento` e *teste*. Mas é sempre recomendável.

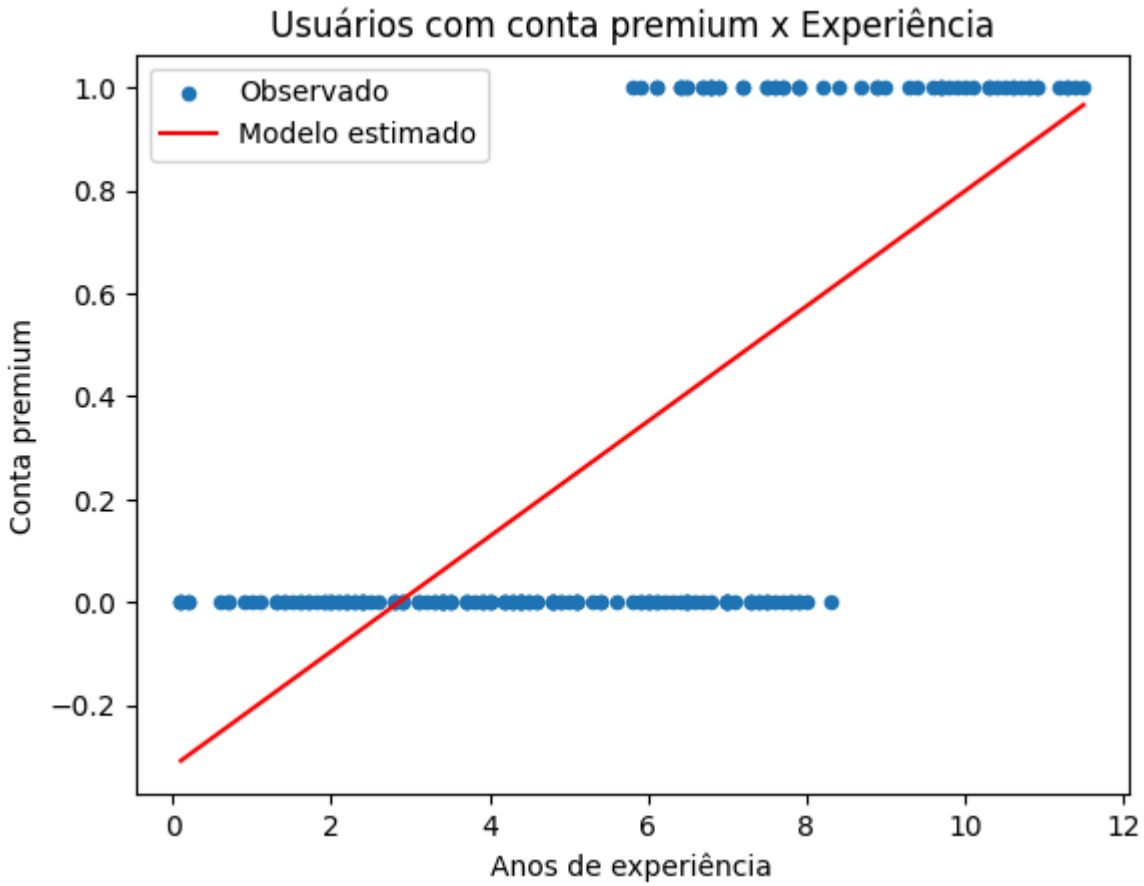
Note que, enquanto nosso *output* é um valor `0` ou `1`, as previsões feitas pelo modelo linear retornam:

In [ ]:
```
# Previsao
previsao = modelo_estimado.predict(X = X_treino)
print(previsao)
# [0.441 0.117 -0.309 ... 0.833 0.967 0.676]
```

De forma geral, vamos analisar o resultado para todo o intervalo utilizando um gráfico:

In [ ]:
```
# Grafico de dispersao - conjunto de teste
conta.plot.scatter(x="experiencia", y= "tipo_conta", label='Observado')

# Desenhando a linha de previsao
x_prev = np.linspace(conta['experiencia'].min(),
```

```
                    conta['experiencia'].max(),
                    100).reshape(-1, 1)
y_prev = modelo_estimado.intercept_ + modelo_estimado.coef_[1] * x_prev
plt.plot(x_prev, y_prev, color='red', label='Modelo estimado')

# Rotulos e titulo
plt.xlabel("Anos de experiência")
plt.ylabel("Conta premium")
plt.title('Usuários com conta premium x Experiência')

# Legenda
plt.legend()

# Exibindo o grafico
plt.show()
```



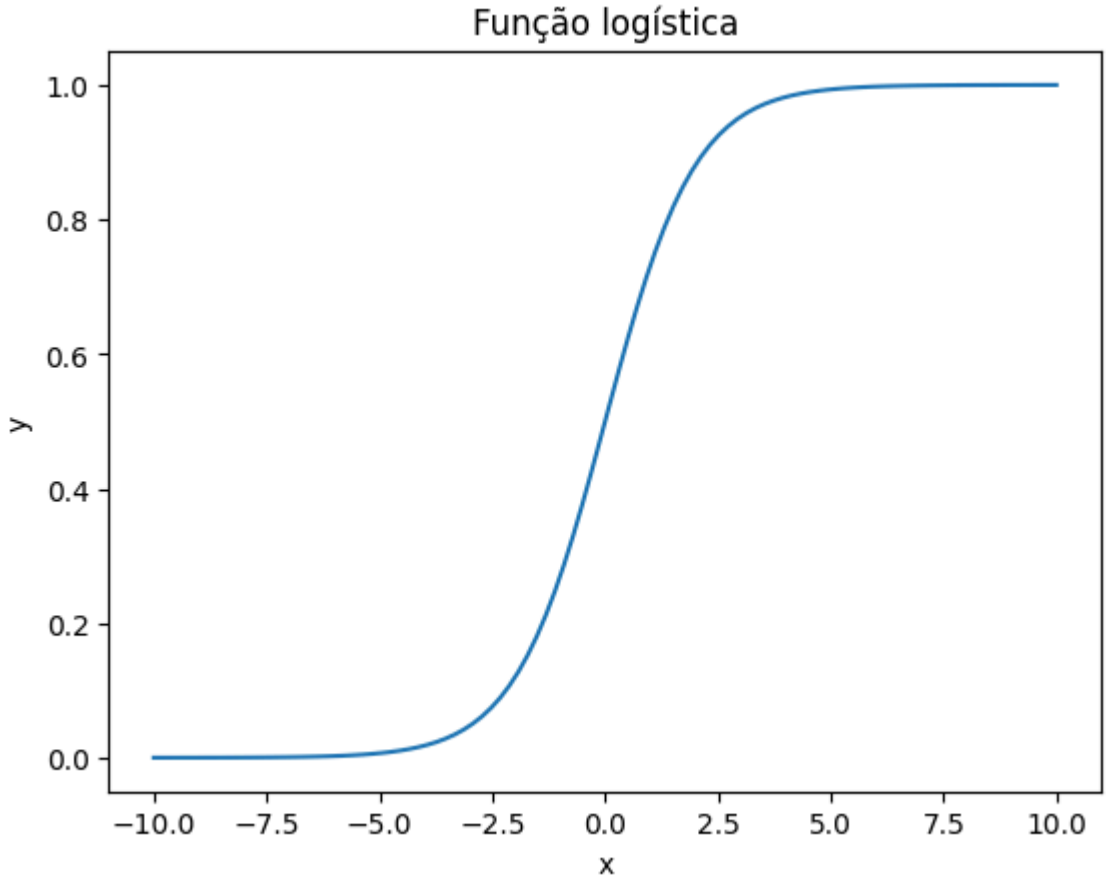
Assim, essa abordagem leva à alguns problemas imediatos:

- Gostaríamos que nossas previsões fossem 0 ou 1, para indicar a associação da classe.
- Tudo bem se eles estiverem entre 0 e 1, pois podemos interpretá-los como probabilidades - uma saída de 0,25 pode representar 25% de chance de ser um membro pago.
- Mas as saídas do modelo linear podem ser grandes números positivos ou até números negativos, que não está claro como interpretar nesse contexto.

Gostaríamos que valores menores de `experiencia` correspondessem à probabilidades próximas de `0`. Por outro lado, valores maiores correspondessem à probabilidades próximas de `1`. Podemos fazer isso aplicando outra função ao resultado.

Função Logística

A regressão logística possui esse nome por conta da função matemática homônima que ela relaciona conforme a figura abaixo:



Graficamente, a curva se aproxima de 1 quando `x` (ou o *input*) cresce. Por outro lado, se aproxima de 0 quando `x` (ou o *input*) diminui. Assim, a curva simboliza a probabilidade de um evento ou resultado.

> A probabilidade é um conceito fundamental na teoria estatística. Ela é uma medida numérica que descreve a chance de um evento ocorrer. A probabilidade varia entre 0 e 1, sendo que 0 indica a impossibilidade do evento ocorrer e 1 indica a certeza absoluta de que o evento ocorrerá.

Aplicação

De forma similar a feita anteriormente, vamos dividir nossos dados em um conjunto de treinamento e um conjunto de teste:

In [ ]:
```
from sklearn.model_selection import train_test_split

# Definindo conjunto de treino e teste simultaneamente
# Aqui, o conjunto de treino sera 80% dos dados
treino, teste = train_test_split(conta,
                                 train_size = 0.80,
                                 random_state = 123)
# random_state = 123 foi escolhido aleatoriamente
# use o numero que achar melhor

print(treino.shape)
print(teste.shape)
```

Utilizaremos 80% dos dados como base de treino. De forma muito similar à regressão linear, aplicamos o modelo logístico da seguinte forma:

In [ ]:
```
import numpy as np
from sklearn.linear_model import LogisticRegression
import statsmodels.api as sm

# Definindo input e output - treinamento
X_treino = sm.add_constant(treino['experiencia'])
y_treino = treino['tipo_conta']

# Definindo o modelo de regressao logistica
reg = LogisticRegression()

# Estimando o modelo
```

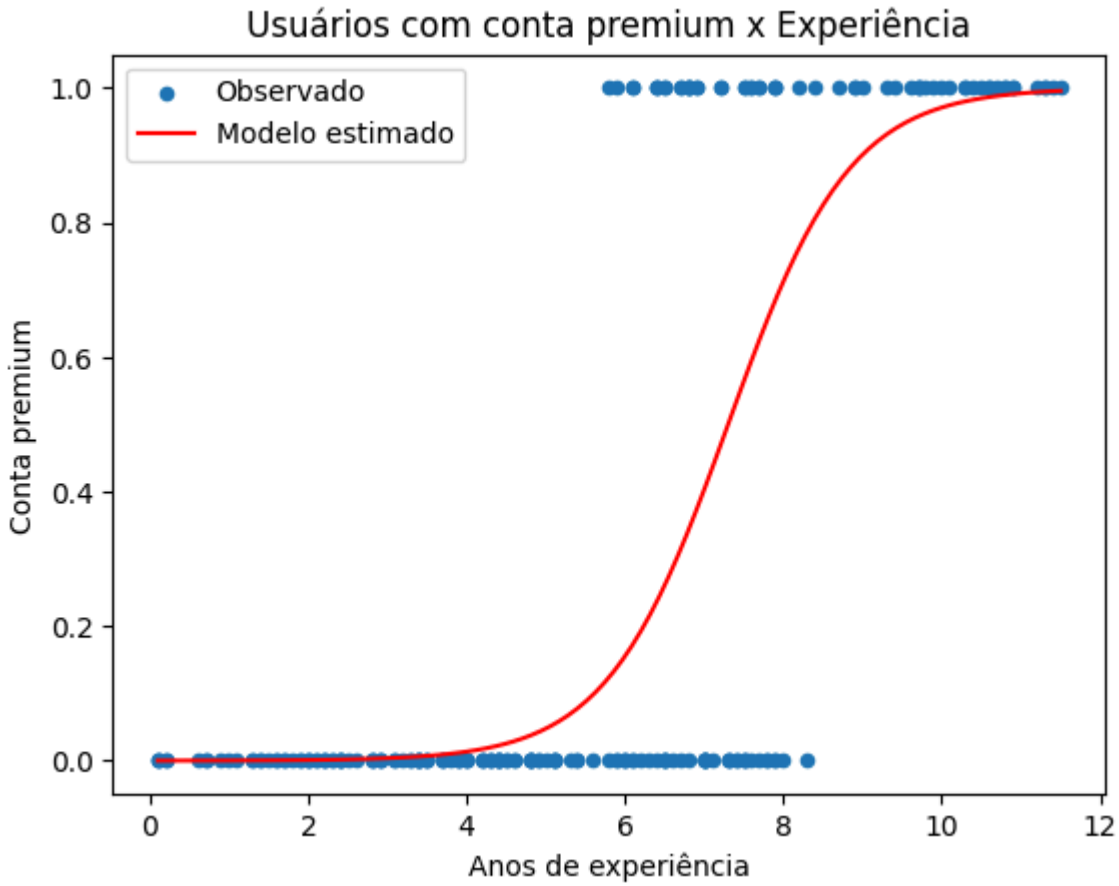
```
modelo_estimado = reg.fit(X = X_treino,
                          y = y_treino)
```

Por fim, podemos fazer previsões da seguinte forma:

In [ ]:
```
# Definindo input e output - teste
X_teste = sm.add_constant(teste['experiencia'])
y_teste = teste['tipo_conta']

# Previsao
previsao = modelo_estimado.predict(X = X_teste)
print(previsao)
```

Interpretação

Infelizmente, este modelo não é tão intuitivo de interpretar quanto a regressão linear. A interpretação geral dos coeficientes α e β são diferentes e não serão abordadas neste contexto. De forma geral, a curva logística irá representa a probabilidade de um usuário ter uma conta *premium* dada sua experiência:



Performance

Anteriormente, introduzimos métricas de *performance* falando sobre 'matriz de confusão'. Neste caso da regressão logística, que faz a classificação de usuários, criamos essa matriz da seguinte forma:

In [ ]:
```
from sklearn.metrics import confusion_matrix

# Matriz de confusao
mat_confusao = confusion_matrix(y_teste, previsao)
```

Interpretaremos os valores da seguinte forma:

	Predito Gratuita	Predito Premium	Total
Gratuita	20	4	24
Premium	6	10	16
Total	26	14	40

Vimos também que um 'bom' modelo envolve um equilíbrio entre as métricas *precisão* e *sensibilidade*. Podemos calculá-las diretamente:

In [ ]:
```
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score

# Precisao
print(precision_score(y_teste, previsao)) # 0.714

# Sensibilidade
print(recall_score(y_teste, previsao)) # 0.625
```

Ou seja, temos uma precisão de 0,714, prevemos corretamente 71% das vezes que o usuário tem uma conta paga. Por outro lado, temos uma sensibilidade de 0,625, dado os usuários com contas pagas, acertamos 62,5%.

Lógico que temos vários outros fatores envolvidos, mas conseguimos construir um bom modelo para classificar um usuário no grupo de contas pagas ou gratuitas.