## revisao

Exported 26/11/2025, 00:05:58

# Revisão Machine Learning

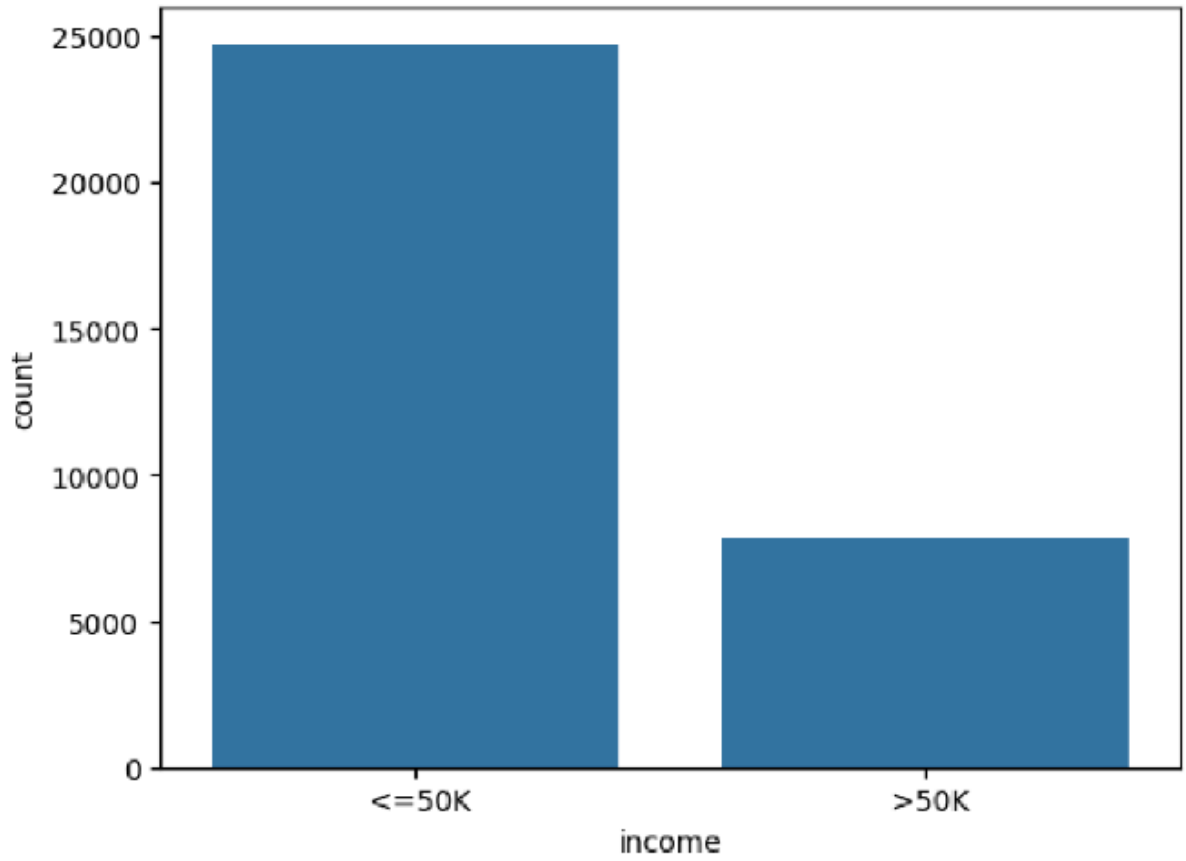
## Importação de arquivos

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv('datasetaqui.csv')
```

# Gráficos

## Countplot & Histograma




In [ ]:

```
sns.countplot(x = base['coluna']

plt.hist(x = base['coluna']
```

## TreeMap

In [ ]:
```
grafico = px.parallel_categories(base_census, dimensions=
['occupation', 'relationship']
grafico.show()
```

### Heatmap

Nota: Precisa incluir o corr, heatmap lida com a **correlação** entre features

In [ ]:
```
plt.figure(figsize=(10,5))
sns.heatmap(base_plano.corr(), annot=True, cmap='coolwarm', fmt=".4f")
plt.title('Correlation between Features')
plt.show()
```

### Box Plot

In [ ]:
```
plt.boxplot(data)

plt.title("Multiple Box Plots")
plt.xlabel("Data Sets")
plt.ylabel("Values")

plt.show()
```

## Dados de Treino e Teste

In [ ]:
```
x = base_census.iloc[:, 0:14].values
y = base_census.iloc[:, 14].values

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(x, y,
test_size=0.2)
```

## ☆ Padronização dos Dados

### Standard Scaler

Responsável por deixar todos os dados em uma mesma escala.

- Centraliza a média em 0
- Ajusta o desvio padrão para 1

É necessário usá-lo em modelos que usem **regularização**, visto que eles podem interpretar dados maiores como mais importantes, o que nem sempre é verdade.

#### Fórmula

$$\frac{x_i - \mu(x)}{\sigma} \qquad \frac{x_i - \mu(x)}{\sigma}$$

#### Código

In [ ]:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
x_train = scaler.fit_transform(x_train) # aprende média e desvio nos
dados
x_test = scaler.transform(x_test) # aplica
```

# Encoder

## Label Encoding

Cada classe terá seu rótulo transformado para um valor numérico

- É comumente utilizado em algoritmos de **Classificação**
- É utilizado em dados ordenáveis, como:
  - Bronze, prata e ouro;
  - Criança, adulto e idoso;
  - Baixo, médio e alto.

É preciso criar um objeto LabelEncoder **para cada coluna**

| Color | Size | Price |
|---|---|---|
| Red | Small | 10 |
| Green | Medium | 20 |
| Blue | Large | 30 |
| Red | Large | 25 |
| Green | Small | 15 |

| Category | Encoded Value |
|---|---|
| Color: Red | 0 |
| Color: Green | 1 |
| Color: Blue | 2 |

| Category | Encoded Value |
|---|---|
| Size: Small | 0 |
| Size: Medium | 1 |
| Size: Large | 2 |

| Color_Encoded | Size_Encoded | Price |
|---|---|---|
| 0 | 0 | 10 |
| 1 | 1 | 20 |
| 2 | 2 | 30 |
| 0 | 2 | 25 |
| 1 | 0 | 15 |

### Código

In [ ]:

```
from sklearn.preprocessing import LabelEncoder

encoding_col1 = LabelEncoder()
endoding_col2 = LabelEncoder()

x[:,1] = encoding_col1.fit_transform(x[:,1])
x[:,2] = encoding_col2.fit_transform(x[:,2])
```

## One-hot Encoding

Cada categoria é transformada em um atributo: dummy variable, um valor binário que informa a ocorrência

### Quando utilizar?

- Quando a variável categórica **não tem ordem** (nominal);
- Quando o número de categorias não é muito **grande**;

### Observações

- Muitas colunas podem gerar um espaço de características de alta dimensão, que pode causar super ajuste e ter um custo computacional muito alto.
- Maldição da Dimensionaldiade: Dados esparços, muitas colunas com valor zero, tornando difícil encontrar valores nos dados
- Dummy Variable Trap: valores de colunas binárias podem ser previstos a partir dos valores de outras colunas.

| Color | Size | Price |
|---|---|---|
| Red | Small | 10 |
| Green | Medium | 20 |
| Blue | Large | 30 |
| Red | Large | 25 |
| Green | Small | 15 |

| Color_Red | Color_Green | Color_Blue | Size_Small | Size_Medium | Size_Large | Price |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | 0 | 10 |
| 0 | 1 | 0 | 0 | 1 | 0 | 20 |
| 0 | 0 | 1 | 0 | 0 | 1 | 30 |
| 1 | 0 | 0 | 0 | 0 | 1 | 25 |
| 0 | 1 | 0 | 1 | 0 | 0 | 15 |

### Código

In [ ]:

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose mport ColumnTransformer

onehotX = ColumnTransformer(transformers=[('OneHot',
OneHotEncoder(handle_unknown='ignore', [6,7], remainder='passthrough')
x = onehotX.fit_transform(x)
```

# ☆ Regressão

- **Não** possui hiperparâmetro
- **Métrica de erro:** MAE
- **Métrica de desempenho:** Score

### MAE

Mean Absolute Error

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

mae = mean_absolute_error(y_test, prev)
```

### Score

In [ ]:
```
regressor.score(x_train, y_train)
regressor.score(x_test, y_test)
```

## Regressão Linear Simples

Modelagem da relação entre variáveis numéricas (variável dependente y e variáveis explanatórias x)

### Intersecção

O ponto de encontro da linha no eixo Y, onde X = 0

### Inclinação

Fator que determina a inclinação da linha onde a cada unidade que aumenta a variável **independente**(x), a variável de **resposta**(y) sobe o valor da inclinação

**Fórmula**

**P = b + m * v**

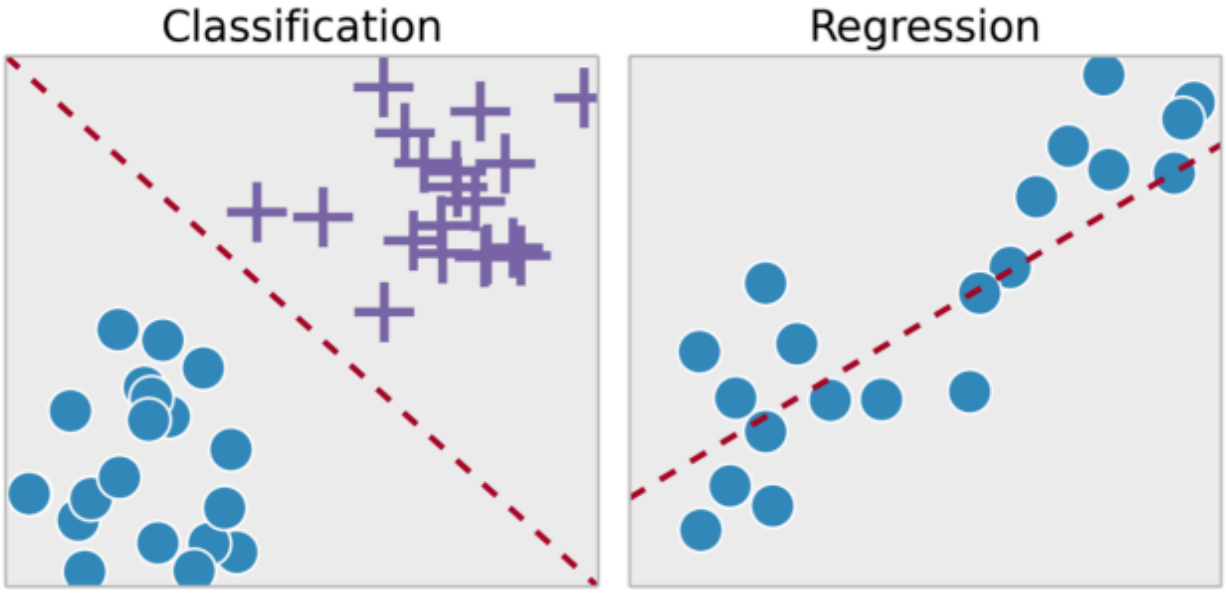
**onde:**

**p:** previsão

**b(constante):** intersecção

**m(coeficiente):** inclinação

**v:** valor a ser previsto em x




In [ ]:

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(x, y)
```

## Regressão Linear Múltipla

É similar a Regressão Linear **Simples**, porém mais complexa.

- Possui **duas ou mais** variáveis exploratórias

**Fórmula**

P = b + m1 * v1 + m2 * v2 + ... + mn * vm

**Código**

In [ ]:

```
from sklearn.linear_model import LinearRegression

regressor = LinearRegression()
regressor.fit(x_train, y_train)
prev = regressor.predict(x_test)
```

## Regressão Linear Polinomial

- Usada quando a relação entre x e y é **curva**, e não pode ser representada apenas por uma linha reta.
- Quando falamos em código, falamos em algo basicamente **igual** a uma regressão linear simples.
- É preciso adequar os dados e criar novas features (elevadas a n) para finalmente treiná-lo

### Fórmula

P = C + m1 * v1 + m2 * v2 + ... + mn * v1^n

### Código

In [ ]:

```
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree = 2)
x_train = poly.fit_transform(x_train)
x_test = poly.transform(x_test)

regressor = LinearRegression()
regressor.fit(x_train, y_train)
```

## Elastic Net

Utilizado quando temos duas ou mais variáveis exploratórias.

Quando temos muitas variáveis, ou quando elas possuem valores muito parecidos, o modelo pode ficar "confuso" e distribuir pesos muito grandes, ocasionando em **overfitting**.

O Elastic usa:

- Ridge para remover exageros
- Lasso para remover variáveis que não são fortemente relacionadas com y.

$$\min_{w} \sum_{i=1}^{N} (\vec{w} \cdot \vec{x}_i + \beta - y_i)^2 + \alpha\rho|w| + \alpha(1-\rho)w^2$$

### Ridge

Penaliza colocando um viés que reduz os grandes pesos o máximo possível, mas sem zerar, assim fazendo com que a variável contribua menos para a predição

$$\min_{w} \sum_{i=1}^{N} (\vec{w} \cdot \vec{x}_i + \beta - y_i)^2 + \alpha w^2$$

### Lasso

- Penaliza o valor, assim como o Ridge, mas, ao invés de penalizar apenas os pesos de grande valor, ele penaliza os de baixo valor também.
- A penalização ocorre até que o valor seja zero.
- Os atributos zerados são descartados da predição.

$$\min_{w} \sum_{i=1}^{N} (\vec{w} \cdot \vec{x}_i + \beta - y_i)^2 + \alpha|w|$$

### Código

In [ ]:

```
from sklearn.linear_model import ElasticNet

regressor_en = ElasticNet(alpha=0.1, l1_ratio=0.5, random_state=0)
regressor_en.fit(x_train, y_train)

prev = regressor_en.predict(x_test)
```

# ☆ Classificação

- **Métrica de erro:** Matriz de confusão
- **Métrica de desempenho:** accuracy_score

## Matriz de confusão (Confusion Matrix)

In [ ]:

```
from yellowbrick.classifier import ConfusionMatrix

plt.figure(figsize=(4,4))
cm = ConfusionMatrix(svm)
cm.fit(x_train, y_train)
cm.score(x_test, y_test)
```

## Accuracy Score

In [ ]:

```
from sklearn.metrics import accuracy_score, classification_report
previsao = svm.predict(x_test)
accuracy_score(y_test, previsao)
```

## Naive Bayes

É um algoritmo de classificação que usa o teorema de Bayes para calcular a probabilidade de um dado pertencer a uma classe específica.

Ele é muito usado em classificação de textos, como detecção de spam, análise de sentimentos e categorização de documentos.

- Esse e-mail é spam ou não spam?
- Esse comentário é positivo, neutro ou negativo?
- Esse cliente vai comprar ou não comprar?

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)}$$

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)}$$

Onde A é cada classe e B a quantidade de dados Na prática, temos uma base de dados original:

| História do crédito | Dívida | Garantias | Renda anual | Risco |
|---|---|---|---|---|
| Ruim | Alta | Nenhuma | < 15.000 | Alto |
| Desconhecida | Alta | Nenhuma | >= 15.000 a <= 35.000 | Alto |
| Desconhecida | Baixa | Nenhuma | >= 15.000 a <= 35.000 | Moderado |
| Desconhecida | Baixa | Nenhuma | > 35.000 | Alto |
| Desconhecida | Baixa | Nenhuma | > 35.000 | Baixo |
| Desconhecida | Baixa | Adequada | > 35.000 | Baixo |
| Ruim | Baixa | Nenhuma | < 15.000 | Alto |
| Ruim | Baixa | Adequada | > 35.000 | Moderado |
| Boa | Baixa | Nenhuma | > 35.000 | Baixo |
| Boa | Alta | Adequada | > 35.000 | Baixo |
| Boa | Alta | Nenhuma | < 15.000 | Alto |
| Boa | Alta | Nenhuma | >= 15.000 a <= 35.000 | Moderado |
| Boa | Alta | Nenhuma | > 35.0000 | Baixo |
| Ruim | Alta | Nenhuma | >= 15.000 a <= 35.000 | Alto |

E para cada feature faz a divisão relacionada com a classe que queremos prever

| | História do crédito | | | Dívida | | Garantias | | Renda anual | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Risco de crédito | Boa 5 | Desconhecida 5 | Ruim 4 | Alta 7 | Baixa 7 | Nenhuma 11 | Adequada 3 | < 15 3 | >= 15 <= 35 4 | > 35 7 |
| Alto 6/14 | 1/6 | 2/6 | 3/6 | 4/6 | 2/6 | 6/6 | 0 | 3/6 | 2/6 | 1/6 |
| Moderado 3/14 | 1/3 | 1/3 | 1/3 | 1/3 | 2/3 | 2/3 | 1/3 | 0 | 2/3 | 1/3 |
| Baixo 5/14 | 3/5 | 2/5 | 0 | 2/5 | 3/5 | 3/5 | 2/5 | 0 | 0 | 5/5 |

| Renda anual | Risco |
|---|---|
| < 15.000 | Alto |
| >= 15.000 a <= 35.000 | Alto |
| >= 15.000 a <= 35.000 | Moderado |
| > 35.000 | Alto |
| > 35.000 | Baixo |
| > 35.000 | Baixo |
| < 15.000 | Alto |
| > 35.000 | Moderado |
| > 35.000 | Baixo |
| > 35.000 | Baixo |
| < 15.000 | Alto |
| >= 15.000 a <= 35.000 | Moderado |
| > 35.0000 | Baixo |
| >= 15.000 a <= 35.000 | Alto |

Essa tabela pronta é o que vai gerar o aprendizado do algoritmo, a partir disso podemos prever novos valores com os cálculos de probabilidade de cada classe

| | História do crédito | | | Dívida | | Garantias | | Renda anual | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Risco de crédito | Boa 5 | Desconhecida 5 | Ruim 4 | Alta 7 | Baixa 7 | Nenhuma 11 | Adequada 3 | < 15 3 | >= 15 <= 35 4 | > 35 7 |
| Alto 6/14 | 1/6 | 2/6 | 3/6 | 4/6 | 2/6 | 6/6 | 0 | 3/6 | 2/6 | 1/6 |
| Moderado 3/14 | 1/3 | 1/3 | 1/3 | 1/3 | 2/3 | 2/3 | 1/3 | 0 | 2/3 | 1/3 |
| Baixo 5/14 | 3/5 | 2/5 | 0 | 2/5 | 3/5 | 3/5 | 2/5 | 0 | 0 | 5/5 |

História = Boa
Dívida = Alta
Garantias = Nenhuma
Renda = > 35

Soma: 0,0079 + 0,0052 + 0,0514 = **0,0645**

P(Alto) = 6/14 * 1/6 * 4/6 * 6/6 * 1/6
P(Alto) = 0,0079
P(Alto) = 0,0079 / 0,0645 * 100 = **12,24%**

P(Moderado) = 3/14 * 1/3 * 1/3 * 2/3 * 1/3
P(Moderado) = 0,0052
P(Moderado) = 0,0052 / 0,0645 * 100 = **8,06%**

P(Baixo) = 5/14 * 3/5 * 2/5 * 3/5 * 5/5
P(Baixo) = 0,0514
P(Baixo) = 0,0514 / 0,0645 * 100 = **79,68%**

## Métricas de desempenho para classificação

**TP - True Positive:** Quantidade de dados que **eram** de uma classe e foram preditas **corretamente**

**TN - True Negative:** Quantidade de dados que **não** eram de uma classe e foram preditas **corretamente**

**FP - False Positive:** Quantidade de dados que **eram** de uma classe e foram preditas **incorretamente**

**FN - False Negative:** Quantidade de dados que **não** eram de uma classe e foram preditas **incorretamente**


**Acurácia**

Mede a porcentagem/proporção de valores que foram preditos de forma correta. ((tp + tn) / (tp + tn + fp + fn))

**Precisão**

Retorna a proporção de todos os dados que foram predições corretas apenas para quais seu modelo apontou como verdadeiras. (tp / (tp + fp)

**Recall**

Retorna a proporção de todos os dados que eram de fato verdadeiras e quantas foram corretamente preditas como positiva. (tp / (tp + fn)

**F1_SCORE**

Ele é a média harmônica entre a precisão e o recall. 2(precisao*recall/precisao+recall)

In [ ]:

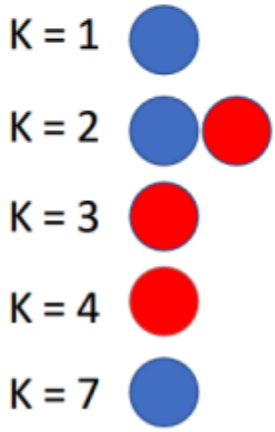
```
from sklearn.naive_bayes import GaussianNB
import pickle
with open('census.pkl', 'rb') as f:
 X_census_treinamento, y_census_treinamento, X_census_teste,
y_census_teste = pick
naive_census = GaussianNB()
naive_census.fit(X_census_treinamento.toarray(), y_census_treinamento)
previsoes = naive_census.predict(X_census_teste.toarray()) #to array
não é necessário
previsoes
```

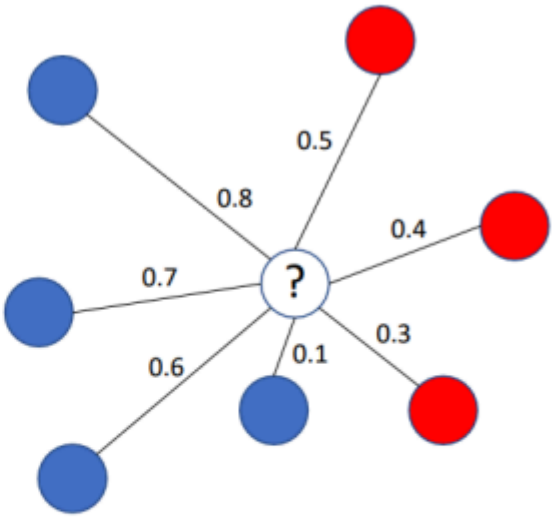
In [ ]:

```
print(classification_report(y_census_teste, previsoes))
```

## KNN (K-Nearest Neighbors)

"K Vizinhos Mais Próximos" — é um algoritmo simples de classificação, que classifica um novo dado olhando quem são os vizinhos mais próximos dele. Onde o valor de K determina o número de vizinhos que o modelo vai considerar

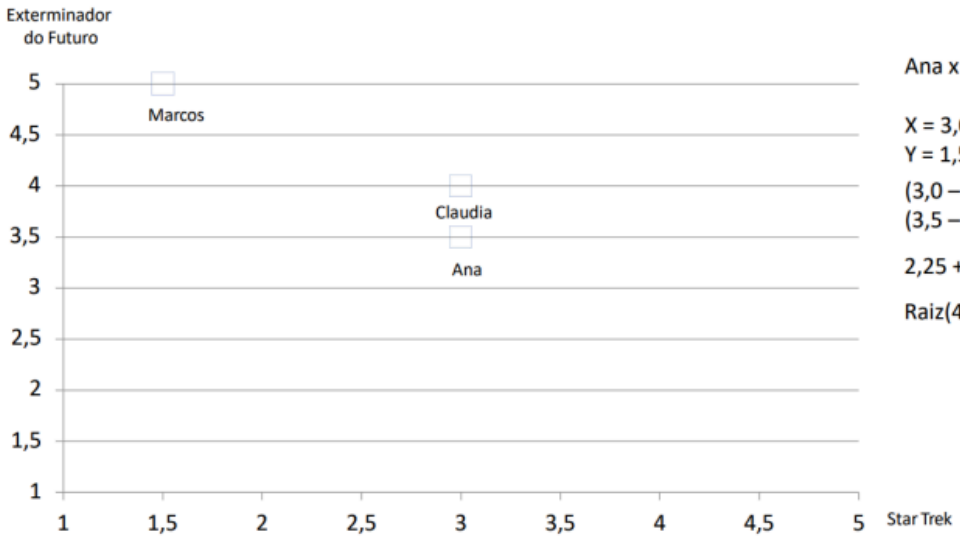



Calcula a distância entre o novo ponto e todos os pontos do conjunto de treinamento (geralmente usa-se a distância Euclidiana).

$$DE(x,y) = \sqrt{\sum_{i}^{p}(x_i - y_i)^2}$$

Pega os K vizinhos mais próximos (os que têm menor distância)




Ana x Marcos

X = 3,0 3,5
Y = 1,5 5,0
$(3,0 - 1,5)^2 = 2,25$
$(3,5 - 5,0)^2 = 2,25$

$2,25 + 2,25 = 4,5$

Raiz(4,5) = **2,12**

Ana x Cláudia

X = 3,0 3,5
Y = 3,0 4,0
$(3,0 - 3,5)^2 = 0,25$
$(3,0 - 4,0)^2 = 1,00$

$0,25 + 1,00 = 1,25$

Raiz(1,25) = **1,11**

O modelo escolhe a classe mais frequente entre os vizinhos

| Filme | Violência | Romance | Ação | Comédia | Classe |
|---|---|---|---|---|---|
| Invocação do Mal | 0,6 | 0,0 | 0,3 | 0,0 | Terror |
| Floresta Maldita | 0,9 | 0,0 | 0,5 | 0,1 | Terror |
| Meu Passado me Condena | 0,1 | 0,2 | 0,1 | 0,9 | Comédia |
| Tirando o atraso | 0,0 | 0,2 | 0,2 | 0,8 | Comédia |

Violência = 0.8
Romance = 0.1
Ação = 0.5
Comédia = 0.0

A Hora do Pesadelo

**Pesadelo x Invocação**
0,8 0,1 0,5 0,0
0,6 0,0 0,3 0,0

$0,2^2 + 0,1^2 + 0,2^2 + 0$
0,04 + 0,01 + 0,04 = 0,09
Raiz(0,09) = **0,30**

**Pesadelo x Floresta**
0,8 0,1 0,5 0,0
0,9 0,0 0,5 0,1

$0,1^2 + 0,1^2 + 0 + 0,1^2$
0,01 + 0,01 + 0,01 = 0,03
Raiz(0,03) = **0,17**

**Pesadelo x Passado**
0,8 0,1 0,5 0,0
0,1 0,2 0,1 0,9

$0,7^2 + 0,1^2 + 0,4^2 + 0,9^2$
0,49 + 0,01 + 0,16 + 0,8 = 1,46
Raiz(1,46) = **1,20**

**Pesadelo x Atraso**
0,8 0,1 0,5 0,0
0,0 0,2 0,2 0,8

$0,8^2 + 0,1^2 + 0,4^2 + 0,8^2$
0,64 + 0,01 + 0,16 + 0,64 = 1,45
Raiz(1,45) = **1,20**

Nesse algoritmo, **não** é criado um modelo, mas sim uma memorização dos dados vizinhos e o cálculo das distâncias para determinar a classe.

**Códi:.**

In [ ]:

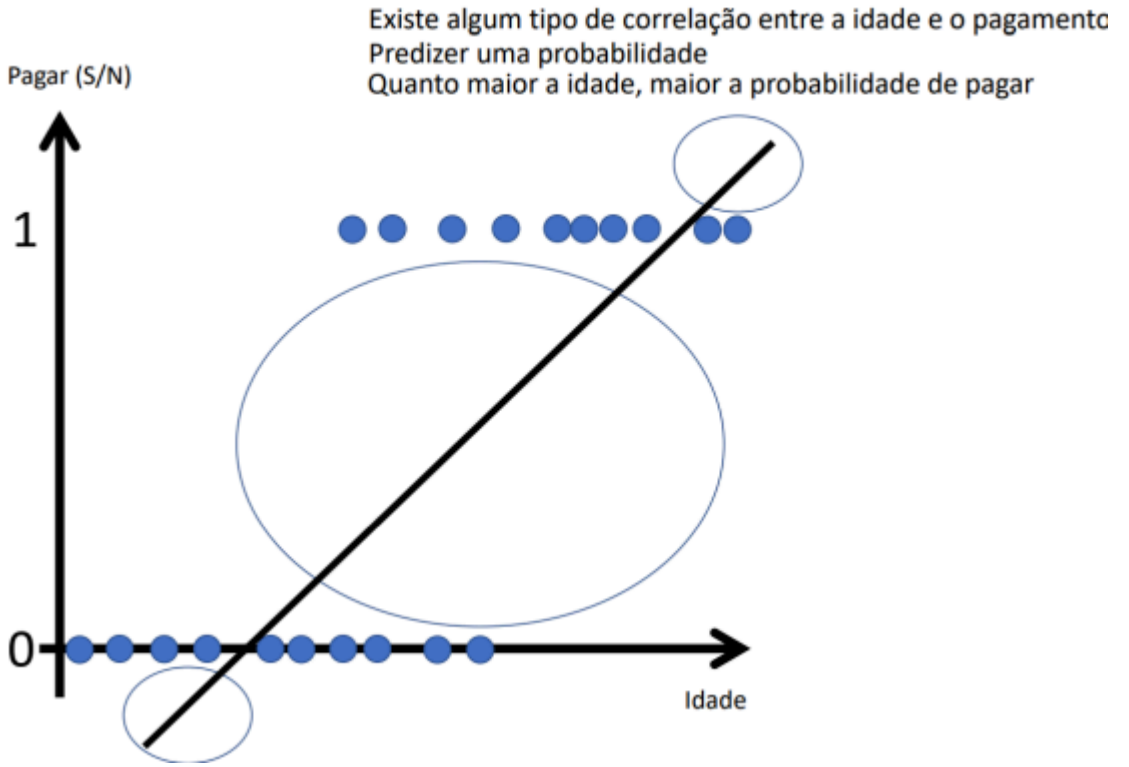
```
from sklearn.neighbors import KNeighborsClassifier

knn_census = KNeighborsClassifier(n_neighbors=10)
knn_census.fit(X_census_treinamento, y_census_treinamento)
previsoes = knn_census.predict(X_census_teste)
previsoes
accuracy_score(y_census_teste, previsoes)
```

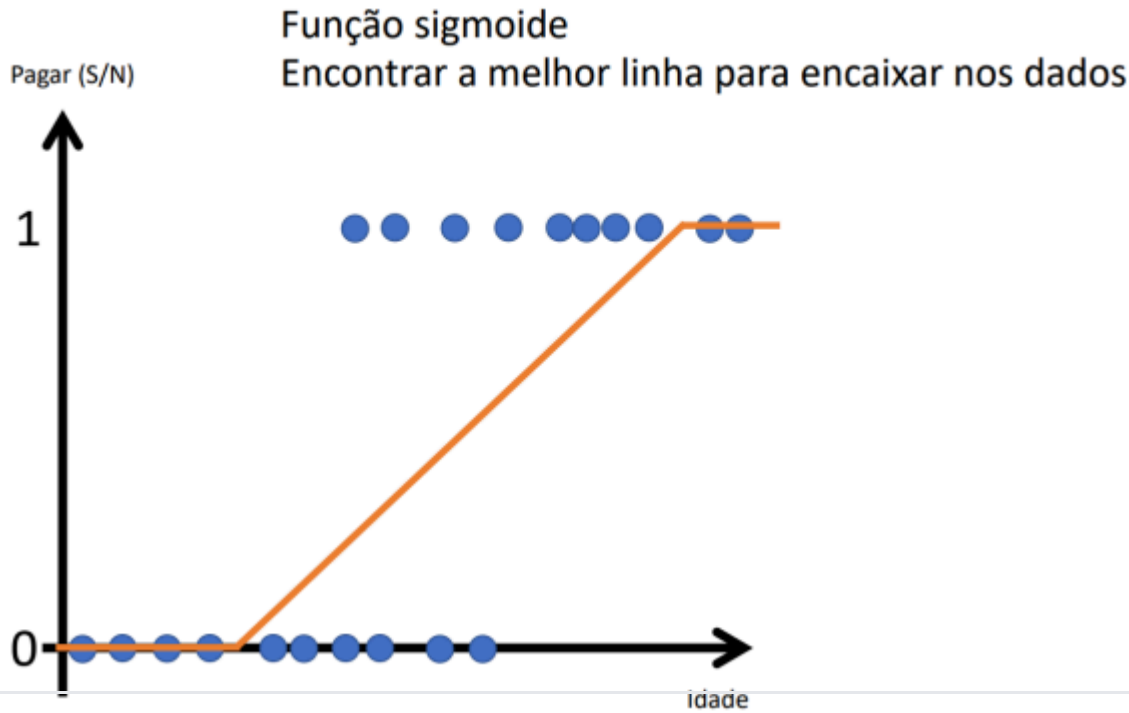
## Regressão Logística

**Hiperparâmetro:** Número de iterações

É um algoritmo para resolver problemas de classificação, ou seja, prevê um resultado categórico. Funciona estimando a probabilidade de um determinado evento ocorrer com base em uma ou mais variáveis independentes.

É mais eficiente quando usado em problemas que terão uma classificação binária, como sim ou não. Esse algoritmo identifica esses valores como 0 e 1 para traçar uma reta entre eles e aplica a função da sigmoide para encontrar a melhor linha e encaixar os dados.



$$y = b_0 + b_1 * x$$

x = idade

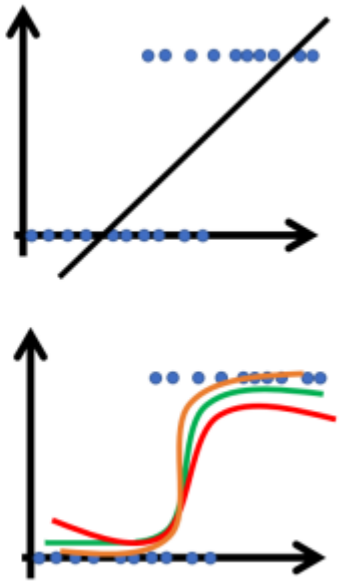
Equação da reta

$$p = \frac{1}{1 + e^{-y}}$$

Função sigmoide

$$\log(\frac{p}{1-p}) = b_0 + b_1 * x$$

Transformação "logit"



In [5]:

```
from sklearn.linear_model import LogisticRegression

logistic_census = LogisticRegression(random_state = 1,max_iter=100)
logistic_census.fit(X_census_treinamento, y_census_treinamento)
previsoes = logistic_census.predict(X_census_teste)
previsoes
```

## SVM

Algoritmo de classificação que cria um hiperplano com base nos vetores de suporte para dividir classes.

Procura forma de separar grandes grupos de dados.

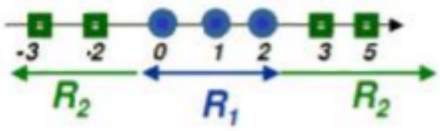
- Tenta encontrar linha (ou plano) que divide os grupos.
- Tenta ao máximo MAXIMIZAR a margem e cria uma "zona de segurança".
- É potente para dados de alta dimensão, mas exige normalização e não escala bem para datasets gigantes.

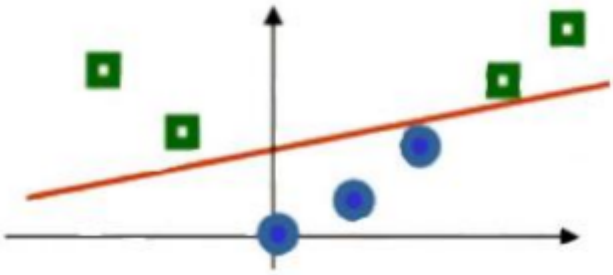
### Hiperparâmetros

- **Kernel Trick:** (Transformação dos Dados) Usado quando os dados não podem ser separados por uma linha reta. Transforma os dados para outra dimensão.

### Principais tipos de kernel:

- Kernel Linear
- Kernel Polinomial
- Kernel Gaussiano (RBF/Radial)



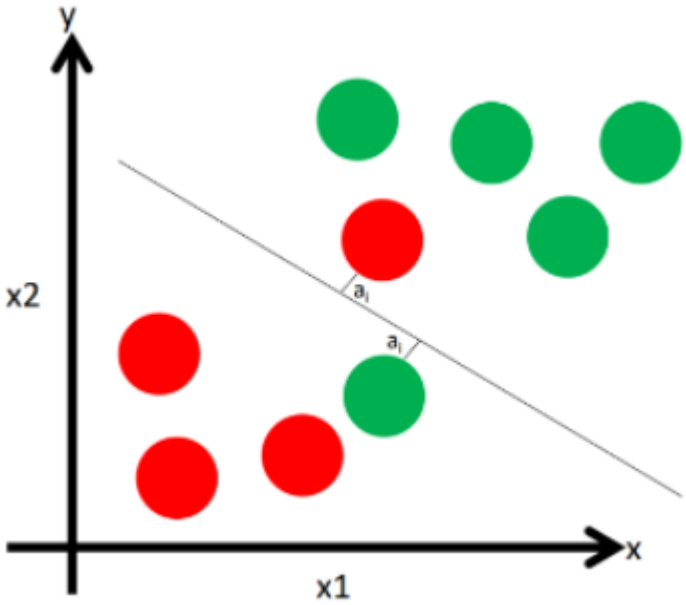
| Característica | Linear | Polinomial | Gaussiano (RBF) |
|---|---|---|---|
| Hiperparâmetros Principais | C | C, grau d, γ, coef0 r | C, γ |
| Risco de Overfitting | Baixo | Médio a Alto | Alto |
| Quando Usar | Dados aproximadamente lineares | R elações não-lineares moderadas | Estrutura muito não-linear ou desconhecida |
| Custo Computacional | Baixo | Médio/Alto | Médio |

- **C:** Penalização dos erros. Controla o quão rígido o modelo será com eles.

$$\frac{1}{2}|w|^2 + c\sum_i a_i$$

Penaliza classificações incorretas.

- C alto → tenta separar completamente, maior risco de overfitting.
- C baixo → permite mais erros, modelo mais simples.




## PCA - Principal Component Analysis

A Análise de Componentes Principais (PCA) é uma técnica utilizada para reduzir o número de variáveis de um conjunto de dados, mantendo o máximo possível da informação essencial. Ela é muito útil quando trabalhamos com bases de dados que possuem muitas colunas e quando percebemos que algumas delas são redundantes ou estão correlacionadas entre si.

**Característica:** técnica utilizada para reduzir o número de variáveis de um conjunto de dados, mantendo o máximo possível da informação essencial, cria atributos artificiais.

**Hiperparâmetro:** n_components (número de componentes é a quantidade de features/colunas que você quer no fim, depois de reduzi-las)

## Select Atributes

A inclusão de muitas características do modelo deteriora sua performance, tornando o modelo super ajustado. Dessa forma a seleção de atributoes tem como objetivo definir quais características, entre as "naturais" e as produzidas, são mais importantes para a performance do modelo.

Para isso existem algumas técnicas:

- **Algoritmo de força bruta:** Testa todas as combinações possíveis de atributos para encontrar o subconjunto que gera o melhor desempenho do modelo, escolhe um modelo (por exemplo, uma árvore de decisão), gera todas as combinações possíveis de atributos (ex: se há 10 atributos, há $2^{10}$ = 1024 combinações), treina e avalia o modelo em cada combinação, seleciona o conjunto de atributos que teve o melhor desempenho.
- **Teste Qui-Quadrado (Chi²):** Mede o grau de dependência entre duas variáveis categóricas — por exemplo, um atributo e a classe alvo. Ele verifica se a distribuição observada dos dados difere da distribuição esperada caso as variáveis fossem independentes. Se uma variável (atributo) é independente da classe, não ajuda a prever, deve ser descartada.
- **ANOVA (Analysis of Variance):** É um teste estatístico usado para verificar se há diferença significativa entre as médias de dois ou mais grupos. O teste calcula uma estatística F:

$$F = \frac{\text{variância entre os grupos}}{\text{variância dentro dos grupos}}$$

  Se F for alto, significa que as médias dos grupos são bem diferentes, variável importante. Se F for baixo, significa que as médias são parecidas, variável irrelevante. O SelectKBest(f_classif) faz exatamente isso para cada feature numérica, comparando-a com as classes da variável alvo

**Tipos de Kernel:**

- linear
- sigmoid
- polynomial
- rbf

**Hiperparâmetros**

- **K:** Número de atributos
- **f_classf:** Para atributos numéricos
- **chi2:** Para atributos categóricos

In [ ]:

```
from sklearn.feature_selection import f_classif, SelectKBest

selecao = SelectKBest(f_classif, k=7)
X_anova = selecao.fit_transform(X, y)
X_treino, X_teste, y_treino, y_teste = train_test_split(X_anova, y,
test_size=0.3, random_state=0)

svm_anova = SVC(kernel='linear', C = 2)
svm_anova.fit(X_treino, y_treino)
previsoes = svm_anova.predict(X_teste)
accuracy_score(y_teste, previsoes)
```

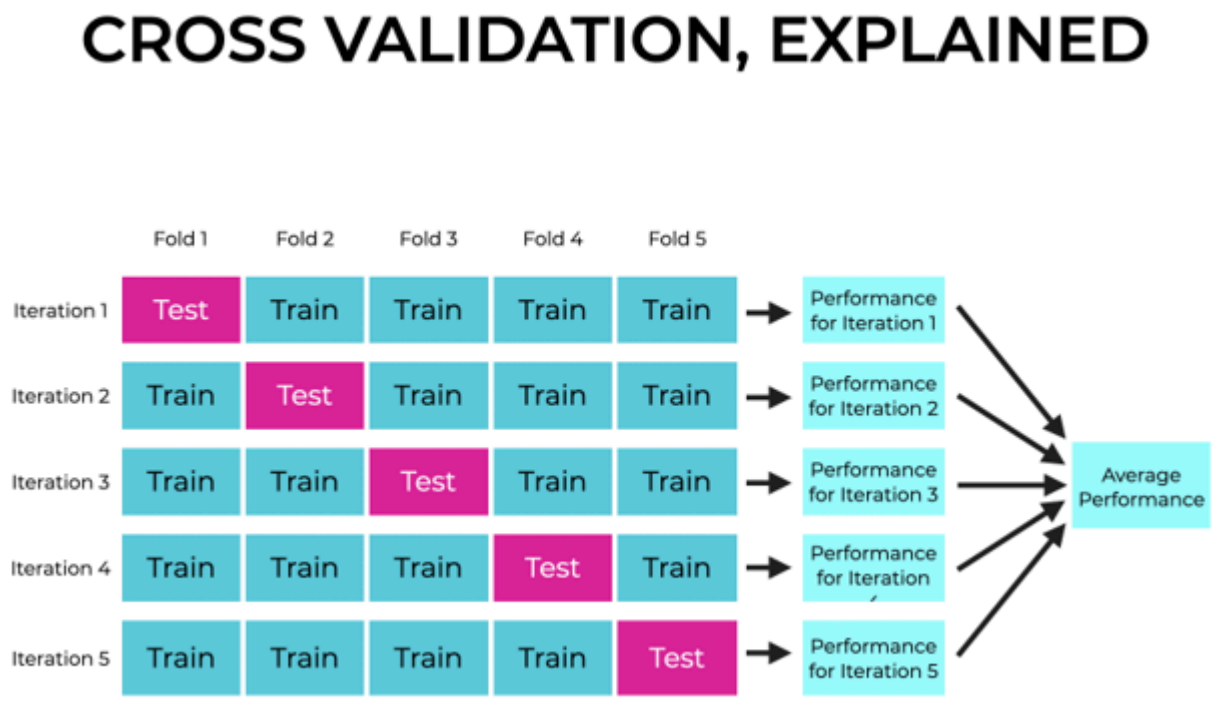
## Cross Validation

- É um método para avaliar o desempenho do modelo de uma forma mais segura.
- Ele repete vários treinos com alguns tipos de divisão de dados, por exemplo: 70% dos dados para treino e 30% para teste, depois 80% treino e 20% teste, com os resultados desses testes é calculado a média geral de todas as performances e assim um desempenho geral do seu modelo de aprendizado.

Benefícios:

- Fornece uma estimativa mais confiável.
- Rápido, evita testar com outras quantidades de dados manualmente.
- Garante que todas as amostras de dados sejam usadas.
- Detecção de OVERFITTING ou UNDERFITTING.

K-Fold Cross Validation (Método mais usado): • Consiste em dividir o conjunto de dados em K partes do mesmo tamanho. O modelo vai repetir o treino K vezes. • Em cada vez 1 das partes será usada para os testes.




Exemplo: Os dados serão divididos em 5 partes. Na primeira vez, a parte 1 parte será utilizada para teste e as partes restantes para treinamento gerando uma métrica de avaliação. Na segunda vez, a parte 2 será utilizada para teste enquanto as demais para treino, assim mostra na imagem acima. Esse processo será repetido 5 vezes até que toda a base passe pelo

processo de treino e teste gerando uma métrica de avaliação média para o modelo.

| Valor de K | Efeito no Modelo | Problema Principal |
|---|---|---|
| **K Pequeno** (Ex.: 3) | Treinamento com **poucos dados**. | Estimativa de desempenho **pessimista** (alto viés). |
| **K Grande** (Ex.: 50) | Modelo treinado com **quase todos os dados**. | Resultados dos testes **variam muito** entre si (alta variância). |
| **K Ideal** (5 ou 10) | **Equilíbrio** entre viés e variância. | Estimativa de desempenho **confiável** e **estável**. |

Hiperparâmetro : O CV é usado para avaliar e selecionar os melhores hiperparâmetros do algoritmo escolhido (Ex: Regressão Linear) Métricas de erro: As métricas são calculadas com base nos testes realizados, então o CV é usado para gerar o melhor desempenho para elas. Métricas de desempenho: As métricas de desempenho são utilizadas em cada iteração da validação cruzada.

In [65]:
```
from sklearn.model_selection import cross_val_score, KFold
```

In [66]:
```
resultados_svm = []

for i in range(30):

  kfold = KFold(n_splits=10, shuffle=True, random_state=i)

  svm = SVC(kernel = 'rbf', C = 1.5)
  scores = cross_val_score(svm, X, y, cv = kfold)
  resultados_svm.append(scores.mean())
```

In [67]:
```
resultados_svm
```

## Grid Search

Grid Search (busca em grade) é um método sistemático para testar várias combinações de parâmetros de um modelo de machine learning e descobrir qual configuração gera o melhor desempenho. Pense nele como um "teste exaustivo" — ele monta uma grade (grid) com todas as combinações possíveis dos valores que você indicar e treina o modelo para cada uma (geralmente usando cross-validation para validar).

In [ ]:

```
from sklearn.model_selection import GridSearchCV

parametros = {'C': [1.0, 1.5, 2.0],
 'kernel': ['rbf', 'linear', 'poly', 'sigmoid']}

grid_search = GridSearchCV(estimator=SVC(), param_grid=parametros)
grid_search.fit(X, y)
melhores_parametros = grid_search.best_params_
melhor_resultado = grid_search.best_score_

print(melhores_parametros)
print(melhor_resultado)
```