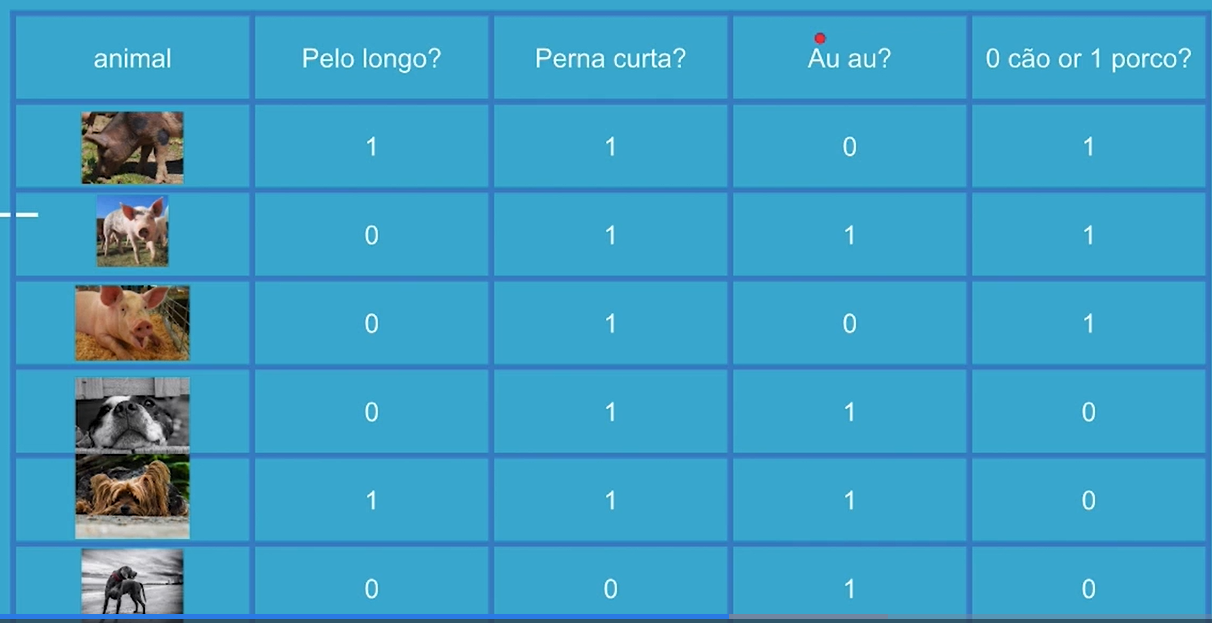
**Machine Learning Completo**

**Machine Learning**

Os notebooks dessa formação estão no seguinte link: <https://drive.google.com/drive/u/0/folders/1XG2od1spsl4OnJLrIs7pztL4c1UmvFDt>

**Classificação com SKLearn**

1. **Aula 1 – Introdução à Classificação:**
   1. Faremos uma classificação para saber se os dados que estamos utilizando é um porco ou um cachorro, por tanto, é uma classificação binária, ou seja, serão classificados em 0 e 1.
      1. Faremos a classificação a partir de características que ensinaremos para o algoritmo, isto é, se o porco tem pelo longo e perna curta, colocaremos 1, se faz se faz “au au”, 0. Essas são as features que serão utilizadas para gerar a classificação de 1 para porco e 0 para cachorro:



* + 1. Através de testes veremos se o nosso algoritmo estimador é bom ou não a partir da taxa de acertos.
  1. A biblioteca que utilizaremos para criar e treinar um modelo é a sklearn.
     1. Dentro dela temos a skleanr.svm, utilizado para importar o LinearSVC, método para criação do modelo.

from sklearn.svm import LinearSVC

* + - 1. Para criar o modelo basta atribuir o método LinearSVC em uma variável.
      2. Para treinar o modelo utilizamos o método .fit não modelo, passando como parâmetro os dados, isto é, uma lista com todos os itens com a lista de suas features dentro, e as classes, dizendo se cada item dos dados é 0 ou 1 (cachorro ou porco, no caso):

# Features: pelo longo, perna curta e faz "au au". (1 == sim, 0 == não)

porco1 = [0, 1, 0]

porco2 = [0, 1, 1]

porco3 = [1, 1, 0]

cachorro1 = [0, 1, 1]

cachorro2 = [1, 0, 1]

cachorro3 = [1, 1, 1]

# 1 => porco, 0 => cachorro

dados = [porco1, porco2, porco3, cachorro1, cachorro2, cachorro3]

classes = [1,1,1,0,0,0]

model = LinearSVC()

model.fit(dados, classes)

* + - * 1. Esse é o método de treinamento assistido, onde dizemos para o algoritmo o que cada item dos dados é.
      1. A partir disso podemos utilizar o predict() no modelo, passando como parâmetro uma lista, para que ele nos diga se é um porco ou charro. Ele irá nos retornar um Array com o número 0 ou 1, a depender da previsão que ele fez:

animal\_misterioso = [1,1,1]

model.predict([animal\_misterioso])



* + - 1. Com o modelo treinado podemos fazer o teste com outros animais misteriosos:

misterio1 = [1,1,1]

misterio2 = [1,1,0]

misterio3 = [0,1,1]

testes = [misterio1, misterio2, misterio3]

previsoes = model.predict(testes)

previsoes



* + - 1. Entretanto, sabemos que o último animal misterioso era na verdade um porco, e não um cachorro, ou seja, o nosso modelo errou. Precisamos testar a taxa de acerto dele, ou a acurácia.
    1. A sklearn.metrics é utilizada para importar o accuracy\_score, método para saber a taxa de acertos que o nosso modelo teve a partir da previsão que ele fez quando comparado com os valores reais, isto é, com os dados que sabemos qual é a resposta.

from sklearn.metrics import accuracy\_score

* + - 1. Utilizamos o accuracy\_score(), passando como parâmetro os dados reais e depois a previsão, isto é, os dados que sabemos qual é a resposta e em seguida a previsão que nosso modelo fez. Esse método retorna a porcentagem de acerto:

testes\_classes = [0,1,1]

taxa\_de\_acerto = accuracy\_score(testes\_classes, previsoes)

print("Taxa de acerto de:", taxa\_de\_acerto \* 100)



* 1. O que aprendemos:
     1. Treinar algoritmos;
     2. Definir características (features) do que desejamos classificar;
     3. Classificar em categorias;
     4. Utilizar o módulo linearSVC e accuracy\_score;
     5. Utilizar o método fit;
     6. Prever dados com a função predict;
     7. Calcular a taxa de acerto do modelo;
     8. Comparar testes e previsões;
     9. Utilizar a função soma;
     10. Padronizar variáveis.

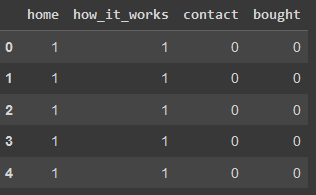
1. **Aula 2 – Testes Replicáveis, Estratificação e Lendo Dados da Internet:**
   1. Nessa parte pegamos os dados fornecidos em um repositório do github no qual se trata de uma tabela com 3 features e uma classificação, assim como fizemos na primeira parte, mas ao invés de animais, foi utilizado os dados de acesso a uma página web, isto é, a primeira coluna dizia se a pessoa acessou a página principal do site, a segunda se a pessoa acessou a página de como o produto funciona e a terceira se a pessoa entrou na página de contato.
      1. A última coluna era a classificação que dizia se a pessoa comprou o produto ou não.
      2. Esses dados foram fornecidos em formato csv e utilizamos o pandas para obtê-los direto da URI:

import pandas as pd

dados = pd.read\_csv("https://gist.githubusercontent.com/guilhermesilveira/2d2efa37d66b6c84a722ea627a897ced/raw/10968b997d885cbded1c92938c7a9912ba41c615/tracking.csv")

dados.head()

* + - 1. A tabela consistia em 4 colunas, como já citado, e 99 elementos:



dados.shape



* + 1. Em ML nós nunca utilizamos 100% dos dados para fazer o treinamento do modelo, sempre deixamos uma porcentagem de fora para que possamos fazer os testes. Nesse caso específico utilizamos 75% para treinar o molde e os outros 25% para fazer o teste de acurácia e verificar se o modelo está sendo eficiente em predizer se o cliente comprou ou não o produto a partir das páginas que ele acessou.
       1. É sempre importante traduzir os dados que estamos trabalhando para melhor se adequar ao nosso trabalho, portanto fizemos um rename das colunas através de um mapa de dicionário:

mapa = {

    "home" : "principal",

    "how\_it\_works" : "como\_funciona",

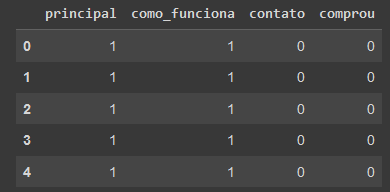
    "contact" : "contato",

    "bought" : "comprou"

}

dados.rename(columns = mapa, inplace = True)

dados.head()



* + - 1. Para essa divisão fizemos o fatiamento dos dados:

treino\_x = x[:75]

treino\_y = y[:75]

teste\_x = x[75:]

teste\_y = y[75:]

print(f'Treinaremos com {len(treino\_x)} elementos e testaremos com {len(teste\_x)} elementos')



* + 1. O processo executado para esse treinamento e teste foi exatamente igual o da primeira parte, com o diferencial dos nomes das variáveis que começamos a utilizar os nomes reais de quando estamos trabalhando com ML.
       1. É chamado de *x* os dados referentes às features e de *y* os dados referentes às labels, portanto, os dados que utilizaremos para fazer o treinamento do modelo se chamam traino\_x e treino\_y, para as features e as labels, respectivamente. O mesmo vale para os dados de teste, sendo teste\_x e teste\_y.
    2. A partir disso fizemos a atribuição, treinamento e teste de acurácia do modelo como antes:

from sklearn.svm import LinearSVC

from sklearn.metrics import accuracy\_score

model = LinearSVC()

model.fit(treino\_x, treino\_y)

previsoes = model.predict(teste\_x)

acuracia = accuracy\_score(teste\_y, previsoes)

print(f'A acurácia desse modelo é de: {acuracia\*100:.2f}%')



* + - 1. Podemos notar que a acurácia desse modelo é bem maior do que a do anterior, provavelmente devido a quantidade de dados fornecidos, sendo essa a tendência.
  1. Podemos utilizar outra importação para facilitar ainda mais nossa vida no processo de separação de dados. Ao invés fazer o fatiamento e gastar 4 linhas de código, podemos importar o train\_test\_split da sklearn.model\_selection, passar os dados totais de features e classificações como parâmetros bem como o test\_size dele, ou seja, a porcentagem que desejamos que a divisão seja feita.
     1. Esse método retorna os 4 dados que utilizaremos: treino\_x, teste\_x, treino\_y, teste\_y, respectivamente, portanto, precisamos fazer as atribuições nessa ordem:

treino\_x, teste\_x, treino\_y, teste\_y = train\_test\_split(x, y, test\_size = 0.25)

* + 1. Contudo, esse método sempre gera uma separação aleatória dos dados e, para que tenhamos uma replicabilidade desse modelo, devemos criar uma seed e passar esse valor para o parâmetro Random\_state, fixando assim a aleatoriedade desse método gerando o mesmo resultado sempre que for executado:

treino\_x, teste\_x, treino\_y, teste\_y = train\_test\_split(x, y, random\_state = SEED, test\_size = 0.25)

* + - 1. O algoritmo complete fica assim:

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import LinearSVC

from sklearn.metrics import accuracy\_score

SEED = 20

treino\_x, teste\_x, treino\_y, teste\_y = train\_test\_split(x, y, random\_state = SEED, test\_size = 0.25)

print(f'Treinaremos com {len(treino\_x)} elementos e testaremos com {len(teste\_x)} elementos')

model = LinearSVC()

model.fit(treino\_x, treino\_y)

previsoes = model.predict(teste\_x)

acuracia = accuracy\_score(teste\_y, previsoes)

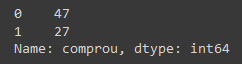
print(f'A acurácia desse modelo é de: {acuracia\*100:.2f}%')



* + 1. Porém, utilizar esse método gera um problema de proporcionalidade, fazendo com que os dados de treino e de teste tenham proporções diferentes, podendo prejudicar e muito na predição:

# proporção de ~2:1

treino\_y.value\_counts()



# proporção de ~3:1

teste\_y.value\_counts()



* + - 1. Para concertar esse ponto utilizamos o parâmetro stratify do método train\_test\_split e passando como valor os dados totais das classificações, ou seja, o *y*:

treino\_x, teste\_x, treino\_y, teste\_y = train\_test\_split(x, y, random\_state = SEED, stratify = y, test\_size = 0.25)

* + - 1. Resultado de proporcionalidade com essa alteração:

# proporção de ~2:1

treino\_y.value\_counts()



# proporção de ~2:1

teste\_y.value\_counts()

