1. O documentário *Coded Bias* expõe os perigos dos vieses presentes na inteligência artificial. Iniciando com a descoberta da pesquisadora Joy Buolamwini, do MIT, que percebeu que algoritmos de reconhecimento facial não identificavam seu rosto corretamente, por ser uma pessoa negra, mas funcionavam quando ela usava uma máscara branca. Assim, revelando que muitos sistemas de IA são treinados com dados predominantemente de pessoas brancas, tornando-os ineficazes para outros perfis.

Além disso, apresenta outros casos de vieses na tecnologia, como um sistema de recrutamento da *Amazon* que discriminava mulheres, um algoritmo de saúde que priorizava o atendimento de pessoas brancas e o uso de dados para manipular eleições. Ademais, alerta para o uso crescente da IA em vigilância pública e tomada de decisões que afetam diretamente a vida das pessoas.

Portanto, a obra questiona a legalidade da coleta e uso indiscriminado de dados pessoais, destacando como a ausência de regulamentação pode permitir abusos. Além do mais, defende a necessidade de políticas e leis que garantam transparência e justiça no desenvolvimento e aplicação dessas tecnologias.

1. Os algoritmos de aprendizagem de máquina são classificados quanto ao tipo de problemas de aprendizagem:
   1. **Classificação:** o objetivo é **prever ou descrever** uma classe, sendo o atributo de classificação **nominal**.
      1. Ex.: Um sistema de detecção de spam que classifica e-mails como "spam" ou "não spam".
   2. **Regressão:** o objetivo é **prever ou descrever** uma classe, mas o atributo de classificação é **numérico.**
      1. Ex.: Quanto tempo de vida uma pessoa terá.
   3. **Agrupamento (*clustering)*:** o objetivo é agrupar as instâncias de acordo com o atributo de entrada e **não é conhecido** o atributo de classificação.
      1. Ex.: Identificar perfis de usuários em redes sociais.
   4. **Associação:** o objetivo é buscar semelhanças/associações entre os elementos.
      1. Ex.: Regras de associação em supermercados, como “clientes que compram pão também compram manteiga”.
2. Respondendo as questões:
3. Uma árvore de decisão é construída através de um processo recursivo que divide os dados em subconjuntos menores com base em um critério de divisão (como entropia e ganho de informação). O algoritmo segue os seguintes passos:
   * 1. **Escolha do atributo raiz da árvore**: O atributo que melhor separa os dados (com maior ganho de informação ou menor impureza) é escolhido como a raiz da árvore.
     2. **Divisão dos dados**: O conjunto de dados é dividido com base nos valores desse atributo.
     3. **Criação de novos nós**: Para cada subconjunto resultante, o processo se repete recursivamente até que:
        1. Todos os exemplos em um nó pertençam à mesma classe.
        2. Não haja mais atributos relevantes para dividir.
        3. Um critério de parada (como profundidade máxima da árvore) seja atingido.
4. Uma vez gerada, a árvore pode ser usada para classificar novos dados, analisar padrões, gerar regras de decisão e selecionar atributos mais relevantes.
5. Suas vantagens incluem facilidade de interpretação, suporte a dados numéricos e categóricos e rapidez no treinamento. No entanto, pode sofrer *overfitting*, ser sensível a variações nos dados e enviesada para atributos com mais valores distintos.
6. Base de Dados:



* Código para cálculo de ganho de Informação e do nível da árvore

import numpy as np

import pandas as pd

from collections import Counter

class Planilha:

def \_\_init\_\_(self, caminho):

self.dataframe = pd.read\_csv(caminho)

self.atributos = self.dataframe.columns.tolist()

# \_\_init\_\_( )

def entropia(self, coluna):

serie\_temporal = self.dataframe[coluna]

contador = Counter(serie\_temporal)

total = len(serie\_temporal)

entropia = 0.0

for contagem in contador.values():

frequencia = contagem / total

entropia += frequencia \* np.log2(frequencia)

entropia = -entropia

return entropia

# entropia ( )

def ganho(self, coluna):

total = len(self.dataframe)

entropia\_total = self.entropia('Conclusao')

valores = self.dataframe[coluna].value\_counts()

entropia\_condicional = 0

for valor in valores.index:

subconjunto = self.dataframe[self.dataframe[coluna] == valor]

proporcao = len(subconjunto) / total

entropia\_condicional += proporcao \* self.entropia('Conclusao')

return entropia\_total - entropia\_condicional

# ganho ( )

def raiz(self):

ganhos = {coluna: self.ganho(coluna) for coluna in self.atributos}

return max(ganhos, key=ganhos.get)

# raiz ( )

def segundo\_nivel(self):

raiz = self.raiz()

ganhos = {coluna: self.ganho(coluna) for coluna in self.atributos}

ganhos.pop(raiz)

return max(ganhos, key=ganhos.get)

# segundo\_nivel ( )

def arvore(self):

raiz = self.raiz()

segundo = self.segundo\_nivel()

print(f"Raiz: {raiz}")

print(f"Segundo nível: {segundo}")

# arvore ( )

# Planilha

if \_\_name\_\_ == "\_\_main\_\_":

planilha = Planilha('restaurante.csv')

for atributo in planilha.atributos:

print(f"Ganho de informação ({atributo}): {planilha.ganho(atributo)}")

planilha.arvore()

# if \_\_name\_\_ == "\_\_main\_\_"

1. Ao executar:

* Ganho de informação (Alternativo): 0.0
* Ganho de informação (Bar): 0.0
* Ganho de informação (SexSab): 0.0
* Ganho de informação (Fome): 0.0
* Ganho de informação (Cliente): 1.1102230246251565e-16
* Ganho de informação (Preco): 0.0
* Ganho de informação (Chuva): 0.0
* Ganho de informação (Res): 0.0
* Ganho de informação (Tipo): 1.1102230246251565e-16
* Ganho de informação (Tempo): 1.1102230246251565e-16
* Ganho de informação (Conclusao): 0.0

1. Pelo cálculo feito também no código anterior, o segundo atributo da árvore será **tipo**.

* Raiz: Cliente
* Segundo nível: Tipo