Lista01_aprendizado_máquina_Angelo_del_rey_Raphael_Brito

February 23, 2025

Nome: ANGELO ORLETTI DEL REY, e-mail: Angelo.rey@aluno.ufabc.edu.br

Nome: Raphael Moisés Pereira Brito, e-mail: Raphael.brito@aluno.ufabc.edu.br

Exercício 1

A) Problema de classificação Exemplo: Diagnostico de Doenças Vetores de Características: Idade, Gênero, Sintomas, Exames Rótulos: Doente ou Não Doente

b) Problema de Regressão Exemplo: Previsão de Preço de casas Vetores de Características: Localização, Área, quartos Reposta: Preço da casa

c) Problema de Agrupamento Exemplo: Segmentação de Clientes

Vetores de Características: Idade, Gasto Mensal, Frequência de Compras Grupos: Categoria de Clientes

Exercício 2

A maldição da dimensionalidade refere-se a uma série de fenômenos que surgem ao lidar com dados em espaços de alta dimensão, dificultando a análise, visualização e modelagem. Esses efeitos não ocorrem em espaços de baixa dimensão, como o tridimensional que experimentamos no dia a dia. Entre os principais problemas associados estão o aumento da dispersão dos dados, a redução da eficácia de métodos estatísticos e de aprendizado de máquina, e a dificuldade em medir distâncias de forma significativa, impactando algoritmos de classificação, agrupamento e otimização.

[50]:
```
import pandas as pd
import numpy as np
from collections import Counter
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn import datasets
import random
```

[60]:
```
#Exercício 3

def kNN(k, x, D):
    #calculando a distância euclidiana
    D["dist"] = ((x[0] - D["x_1"] )**2 + (x[1]- D["x_2"] )**2)

    #ordenando pela menor distância e seleciona os k vizinhos mais próximos
    neighbors = D.nsmallest (k, "dist") ["y"]
```

```
    #retorna a classe mais comum entre os vizinhos
    return Counter (neighbors).most_common(1)[0][0]

D = pd.DataFrame({
    "x_1": [-0.0615459, 1.0131817, 0.2965129, 1.5458330, -0.6332987, 2.0871559],
    "x_2": [-0.3463786, -2.7884388, -4.2492883, -2.2266978, -0.4540896, 2.
 ↪1692931],
    "y": ["three", "one", "one", "one", "three", "one"]
})
x = (1, 2)
k = 10
classe_predita = kNN( k, x, D)
print(f"Classe predita pra x = {x}: {classe_predita}")
```

```
Classe predita pra x = (1, 2): one
```

[66]:
```
#Exercício 4

# carregando o dataset iris

data = datasets.load_iris()
df_iris= pd.DataFrame(data.data, columns = data.feature_names)
df_iris["y"] = data.target_names[data.target]

#seleciona as colunas Petal.length e sepal.length
df_iris = df_iris.rename(columns ={"petal length (cm)": "x_1", "sepal length␣
 ↪(cm)": "x_2"})

#separando o conjunto de treino e teste

train, test = train_test_split(df_iris, test_size = 0.3, random_state = 42)

def evaluate_knn(k):
    prediction = test.apply (lambda row: kNN(k, (row["x_1"], row ["x_2"]),␣
 ↪train), axis = 1)
    return accuracy_score(test["y"], prediction)

# Avaliando para k=10 e k=1
acc_k1 = evaluate_knn(1)
acc_k10 = evaluate_knn(10)

print (f"Acurácia com k = 1: {acc_k1}")
print (f"Acurácia com k = 10: {acc_k10}")
```

```
Acurácia com k = 1: 0.9333333333333333
Acurácia com k = 10: 1.0
```

[74]:
```
#Exercício 5

# Mostrando que a função ótima para a perda do erro absoluto é a Mediana(Y | X␣
 ↪= x)
def mediana_erro_absoluto(Y):
    return np.median(Y)

dados_Y = np.array([1,3,3,6,7,8,9])

print(f"Mediana erro absoluto: {mediana_erro_absoluto(dados_Y)}")
```

```
Mediana erro absoluto: 6.0
```

[76]:
```
# Mediana da distância do ponto mais próximo à origem em uma hiperesfera

def mediana_distancia_hiperesfera (m,d):
   return (1 - 0.5 **(1/m))**(1/d)

m, d = 100, 3
print (f"Mediana da distância para m = {m}, d = {d}:␣
 ↪{mediana_distancia_hiperesfera(m,d)}")
```

```
Mediana da distância para m = 100, d = 3: 0.19044682062800528
```

[ ]:
```

```