UNIVERSIDADE
FEDERAL DO CEARÁ

Trabalho 5

kNN com mecanismo de laplace

Equipe:
Pedro Joás Freitas Lima - 548292
Marcos Antônio Alencar da Rocha Junior - 496563

Objetivo

O trabalho consiste em implementar mecanismo de laplace, que introduz privacidade diferencial em consultas realizadas sobre bases de dados. Deve-se incluir esse mecanismo no modelo kNN (k-Nearest Neighbors), a fim de comparar as respostas geradas pelo classificador sem a introdução de ruído e as respostas geradas pelo kNN diferencialmente privado.

Pseudocódigo do kNN tradicional

Entrada:
    X_train, y_train
    X_test
    k

Para cada x em X_test:
    calcular a distância euclidiana de x para todos os pontos em X_train
    ordenar as distâncias em ordem crescente
    selecionar os k vizinhos mais próximos
    obter os rótulos desses k vizinhos
    escolher o rótulo mais frequente
    armazenar a predição

Saída:
    lista de predições

Codificação das variáveis não numéricas

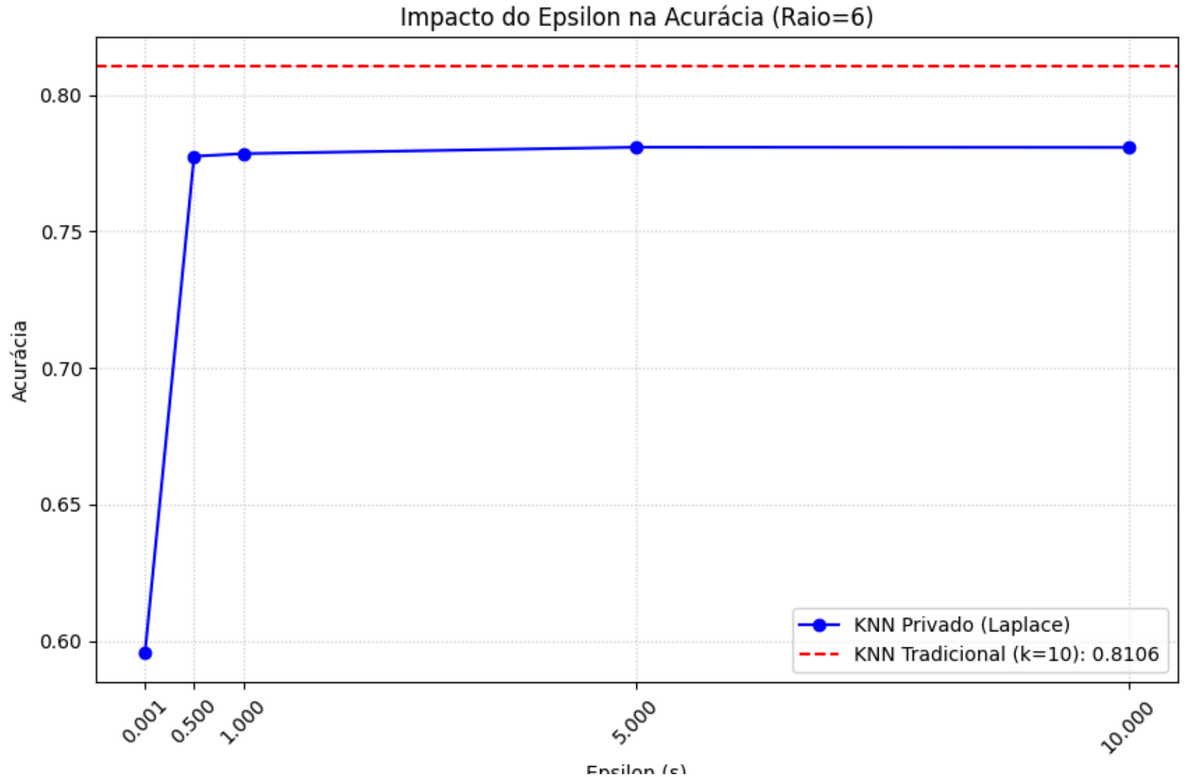
```python
def codificar_dados(df: pd.DataFrame) -> Tuple[Dict, pd.DataFrame]:
    """Codifica colunas categóricas."""
    df_enc = df.copy()

    cols = ['workclass','marital-status','occupation','relationship','race','gender','native-country','income']

    mapeamento = {}
    for c in cols:
        unique_vals = df_enc[c].unique()
        mapa = {val: i for i, val in enumerate(unique_vals)}
        mapeamento[c] = mapa
        df_enc[c] = df_enc[c].map(mapa)

    return mapeamento, df_enc

```

Implementação do mecanismo de laplace

```python
class LaplaceKNN:

    def __init__(self, dataset, labels, privacy_budget, radius_r):
        """ Inicializa o classificador Laplace KNN.
            Parâmetros:
            dataset: Conjunto de dados de treinamento (features).
            labels: Rótulos correspondentes ao conjunto de dados de treinamento.
            privacy_budget: Orçamento de privacidade (epsilon).
            radius_r: Raio r para considerar vizinhos.
        """
        self.dataset = dataset
        self.labels = labels
        self.privacy_budget = privacy_budget
        self.radius_r = radius_r


    def countXT(self, x_teste):
        """ Conta os rótulos dos vizinhos dentro do raio r para a amostra x_teste. """

        distancias = distancia_euclidiana(self.dataset, x_teste)

        # C_r_x(I) < r
        # np.where retorna uma tupla, onde o primeiro elemento é o array de índices
        indices_vizinhos = np.where(distancias <= self.radius_r)[0]

        # Se não houver vizinhos dentro do raio, retorna dicionário vazio
        if len(indices_vizinhos) == 0:
            return {}

        # Pegando os rótulos dos vizinhos dentro do raio
        rotulos_vizinhos = self.labels[indices_vizinhos]

        # Contagem dos votos dos rótulos dos vizinhos
        contagem_votos = {}
        for rotulo in rotulos_vizinhos:
            if rotulo in contagem_votos:
                contagem_votos[rotulo] += 1
            else:
                contagem_votos[rotulo] = 1

        return contagem_votos

```

```python
def gerar_ruido_laplace(self, epsilon: float) -> float:
        """
        Gera um ruído de Laplace usando Amostragem por Transformada Inversa.
        Fórmula: x = mu - b * sgn(u) * ln(1 - 2|u|)
        """
        escala = 1 / epsilon

        u = np.random.rand() - 0.5
        sinal = np.sign(u)
        termo_log = np.log(1 - 2 * np.abs(u))

        ruido = -escala * sinal * termo_log
        return ruido

    def calcular_contagens_ruidosas(self, x_teste, labelsSet):
        """ Calcula as contagens ruidosas para cada rótulo possível. """
        # Obter as contagens
        contagens = self.countXT(x_teste)

        # Se contagens estiver vazio (nenhum vizinho), para tudo, e retorna None
        if not contagens:
            return None

        # Ajuste obrigatório: Composição Sequencial (Epsilon / L)
        L = len(labelsSet)
        epsilon_dividido = self.privacy_budget / L

        # Loop sobre todos os rótulos possíveis
        # Adiciona ruído de Laplace a cada contagem
        contagens_ruidosas = {}
        for I in labelsSet:
            contagem_I = contagens.get(I, 0)
            nc_I = contagem_I + self.gerar_ruido_laplace(epsilon_dividido)
            contagens_ruidosas[I] = nc_I

        return contagens_ruidosas


    def decidir_vencedor_ruidoso(self, contagens_ruidosas):
        """ Decide o rótulo vencedor com base nas contagens ruidosas. """
        label_previsto = max(contagens_ruidosas.items(), key=lambda item: item[1])[0]
        return label_previsto

    def atualizar_epsilon(self, epsilon):
        """ Atualiza o orçamento de privacidade (epsilon). """
        self.privacy_budget = epsilon
```

Gráfico de acurácia



- **Sensibilidade igual a 1 para todos os casos**
- **Valores de epsilons usados [0.001,0.5,1,5,10]**