Universidade Federal do Maranhão
Programa de Pós Graduação Em Ciência da Computação

Aluno
CARLOS EDUARDO NASCIMENTO CAJADO

Professor
Prof. Dr. LUCIANO REIS COUTINHO

CCOM0002 - APRENDIZAGEM DE MÁQUINA

Lista de Atividades 2 - Questão 1
Algoritmo K-nn

São Luís, MA
Julho- 2024

1. Introdução

1.1 Contexto

O algoritmo k-NN, método dos k-Vizinhos mais Próximos (k-Nearest Neighbor), é amplamente utilizado na Aprendizagem de Máquina para promover a classificação de um padrão cuja classe é desconhecida, avaliando sua vizinhança com base nos k padrões de treinamento mais próximos. O voto majoritário na vizinhança é utilizado para definir a classe atribuída a cada instância [1].

Ademais, é importante salientar a problemática do algoritmo k-NN tradicional, pois todos os k padrões de treinamento na vizinhança do padrão cuja classe se deseja inferir são ponderados igualmente na etapa de determinação da classe majoritária. Isso pode gerar problemas de classificação à medida que o valor de k aumenta.

1.2 Problema

Neste estudo, será utilizado o algoritmo K-nn para investigar como a acurácia varia em relação a diferentes fatores: o número de instâncias (tamanho da base de dados), o número de vizinhos (valor de k) e a medida de distância utilizada. Além disso, será considerado o uso de pesos, onde cada vizinho contribuirá para a classificação com base na sua distância. Desta forma, em vez de utilizar apenas o número de vizinhos para a classificação, atribuir-se-ão pesos aos vizinhos dependendo da sua proximidade, buscando avaliar se isso melhora a acurácia do modelo.

1.3 Objetivos

- **Investigar a Dependência da Acurácia em Relação ao Número de Instâncias**: Avaliar como o desempenho do algoritmo K-nn varia com diferentes tamanhos de base de dados.
- **Analisar o Impacto do Número de Vizinhos (k)**: Como a escolha do valor de k afeta a acurácia do modelo.
- **Examinar a Influência das Medidas de Distância**: Comparar como diferentes medidas de distância (Euclidiana, Manhattan) e seu impacto na acurácia do K-nn.
- **Avaliar o Uso de Pesos Baseados na Distância**: Implementar um esquema de pesos onde cada vizinho contribui para a classificação de acordo com sua proximidade, e verificar se isso melhora a acurácia do modelo.
- **Realizar Análises de Casos de Classificação Incorreta**: Identificar e discutir casos onde a classificação foi incorreta.

1.4 Configuração do Ambiente

Para desenvolvimento do projeto, temos as seguintes configurações:

1.3.1 Software:

- Windows 11
- Linguagem python versão 3.12.2
- Jupyter Notebook
- Bibliotecas scikitlearn, pandas, numpy, matplotlib, seaborn.

1.3.2 Hardware:

- Processador: Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz   1.80 GHz
- Memória RAM: 16,0 GB (utilizável: 15,9 GB)

2. Bases de Dados

2.1 Descrição

- Título da base : Diagnóstico de Câncer de Mama em Wisconsin (WDBC)

A base de dados foi criada por Dr. William H. Wolberg, W. Nick Street e Olvi L. Mangasarian, em novembro de 1995, na Universidade de Wisconsin, o conjunto se baseia em 30 recursos de entrada calculados a partir de imagens digitalizadas de aspirados por agulha fina (PAAF) de massas mamárias, os quais descrevem as características dos núcleos celulares presentes nas imagens.Com o objetivo de desenvolvimento e aprimoramento de métodos de diagnóstico de câncer de mama [2].

2.2 Estatísticas

O conjunto de dados possui 569 instâncias e 32 atributos, incluindo um identificador, um diagnóstico (M = maligno, B = benigno) e 30 características de entrada de valor real. As dez características calculadas para cada núcleo celular incluem raio, textura, perímetro, área, suavidade, compacidade, concavidade, pontos côncavos, simetria e dimensão fractal.

Divisão dos valores alvos de cada instância temos:

357 - Benigno.

212- Maligno.

3. Preparação (Pré- processamento)

3.1 Preparando as valores alvos

Primeiramente, é essencial substituir os identificadores de diagnóstico (M = maligno, B = benigno) por números inteiros. Para essa finalidade, definimos M = 1 e B = 0. Utilizarmos os métodos, da figura 1, para mapear o conjunto de dados e realizar as substituições.

Figura 1: convertendo as classes *strings* em números.

```python
def substituir_strings_por_numeros(linha, mapeamento):
    """
    Substitui as strings por números em uma
    linha de acordo com o mapeamento fornecido.
    """
    for chave, valor in mapeamento.items():
        linha = linha.replace(chave, valor)
    return linha

def converter_arquivo(arquivo_entrada, arquivo_saida, mapeamento):
    """
    Converte o arquivo de entrada substituindo as
    strings por números de acordo com o mapeamento fornecido.
    """
    with open(arquivo_entrada, 'r') as arquivo_entrada, open(arquivo_saida, 'w') as arquivo_saida:
        for linha in arquivo_entrada:
            linha_numerica = substituir_strings_por_numeros(linha, mapeamento)
            arquivo_saida.write(linha_numerica)

#conversão
mapeamento_linha = {
    'M': '1',
    'B': '0'
}
```

Autor: próprio

3.2 Limpeza dos dados

Através da documentação fornecida pelos criadores do *dataset* observamos a presença de uma coluna "id" destinada a identificar a amostra de cada instância. Essa coluna é considerada desnecessária para o treinamento do modelo. Portanto, procederemos à remoção da coluna "id", figura 2 , garantindo assim a eficiência e a simplificação do conjunto de dados para análises posteriores.

Figura 2: removendo a coluna “id”

```python
def remove_ids(input_file, output_file):
    with open(input_file, 'r') as infile, open(output_file, 'w') as outfile:
        for line in infile:
            # Split ID
            parts = line.split(',', 1)
            if len(parts) > 1:
                outfile.write(parts[1])

input_file = 'wdbc_pre'
output_file = 'wdbc_normalizacao'

remove_ids(input_file, output_file)
```

Autor: próprio

3.3 Distribuição das classes

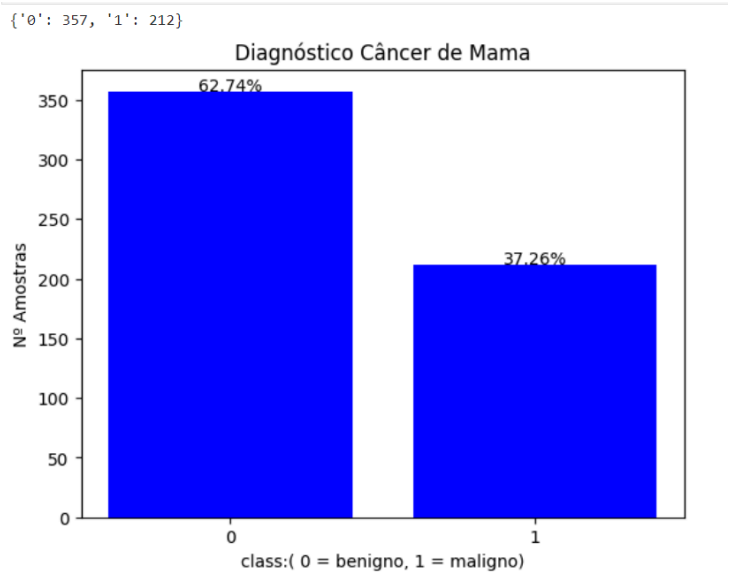
Através do método presente na figura 3 , podemos validar a distribuição dos dados, figura 4 :

Figura 3: Plota a distribuição dos dados após normalização.

```python
def plotar_grafico_contagem(arquivo):
    class_distribution_tumor = {'0': 0, '1': 0}

    total_linhas = 0
    with open(arquivo, 'r') as file:
        for linha in file:
            # Pegando a classe
            classes = linha.strip()[0]
            if classes in class_distribution_tumor:
                class_distribution_tumor[classes] += 1
                total_linhas += 1

    porcentagens = {k: (v / total_linhas) * 100 for k, v in class_distribution_tumor.items()}

    plt.bar(class_distribution_tumor.keys(), class_distribution_tumor.values(), color='blue')

    plt.xlabel('class:( 0 = benigno, 1 = maligno)')
    plt.ylabel('Nº Amostras')
    plt.title('Diagnóstico Câncer de Mama')
    print(class_distribution_tumor)
    # Exibindo as porcentagens
    for key, value in class_distribution_tumor.items():
        plt.text(key, value + 0.5, f'{porcentagens[key]:.2f}%', ha='center')
    plt.show()
```

Autor: próprio

Figura 4: Distribuição das classes



Autor: próprio

4. Divisão dados

O *Database* será dividido nas partes:

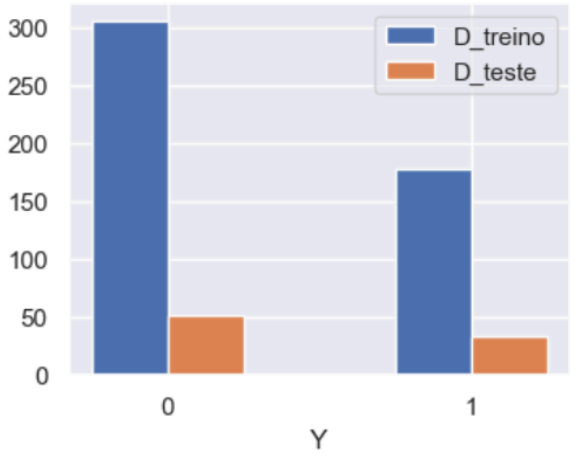
- Treino: 85%
- Teste: 15%

Definindo as colunas dos Atributos “X” e Classes “Y” para treino e teste.

```
# Selecionado os atributos e classes de (treino + validação) e teste.
X_treino = D_treino.iloc[:, 1:31]
Y_treino = D_treino.iloc[:, 0]
print("D_treino: ", (X_treino.shape, Y_treino.shape))

X_teste = D_teste.iloc[:, 1:31]
Y_teste = D_teste.iloc[:, 0]
print("D_teste: ", (X_teste.shape, Y_teste.shape))
```

```
D_treino:  ((483, 30), (483,))
D_teste:  ((86, 30), (86,))
```

Figura 6: Plot treino e teste.



Autor: próprio

Visando combater a sensibilidade na escala dos atributos, utilizaremos a normalização dos dados para o intervalo [-1, +1].

5. Experimentos e Resultados

5.1 Usando o algoritmo K-nn para base de dados.

Definindo o modelo KNeighborsClassifier disponível através da biblioteca Scikit-learn. O classificador é configurado para considerar os 2 vizinhos mais próximos (n_neighbors=2) ao realizar a classificação de novos dados, figura 7. Ou seja, para cada instância a ser classificada, o algoritmo irá buscar os dois pontos de dados mais próximos no conjunto de treinamento e utilizará a classe majoritária desses vizinhos para determinar a classe da nova instância.

Figura 7: Modelo algoritmo K-nn

```
[10]: # Importando o KNN
      from sklearn.neighbors import KNeighborsClassifier

[11]: # Criando o classificador
      clf = KNeighborsClassifier(n_neighbors=2)

[12]: # Fazendo o fit com os dados de treino
      clf = clf.fit(X_treino_Normalizado,Y_treino)

[13]: # Fazendo a previsão para os dados de teste
      y_pred = clf.predict(X_teste_Normalizado)
```

Autor: próprio

Utilizando os métodos de validação e treino onde o método 'teste' avalia um modelo treinado em um conjunto de dados de teste e retorna uma métrica de desempenho. já o método 'validacao' treina um modelo com dados de treinamento, avalia o desempenho tanto nos dados de treinamento quanto nos de validação, e retorna essas métricas de desempenho. Tens-se presente na figura 8:

Figura 8: Função teste e validação

```
def teste(g, D_tst, metrica, **kwds):
    X_tst, y_tst = D_tst
    y_true = y_tst
    y_pred = g.predict(X_tst)
    E_tst = metrica(y_true, y_pred, **kwds)

    return E_tst


def validacao(Modelo, D_trn, D_val, metrica, **kwds):
    # Treinamento:
    g = Modelo.fit(*D_trn)

    # Teste:
    E_trn = teste(g, D_trn, metrica, **kwds)
    E_val = teste(g, D_val, metrica, **kwds)

    return E_trn, E_val
```

Autor: próprio

5.2 Desempenho do modelo

Resultado da avaliação do modelo, na figura 9:

```
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(Y_teste, y_pred)
print("Acurácia ", accuracy)
```

```
Acurácia  0.9651162790697675
```

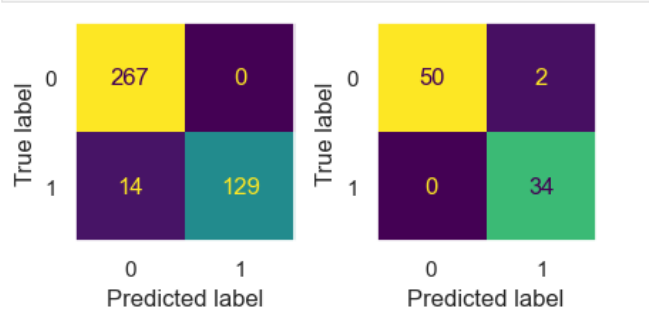
Para o modelo com 2 vizinhos é encontrado uma acurácia de teste de 96,51%

5.3 Matriz de confusão

Aspirando uma representação visual das previsões do modelo em relação aos valores reais e uma análise mais rápida do desempenho em cada classe será gerado a matriz de confusão. Plotamos matriz de confusão na figura 10:

```
# Plot matriz de confusão
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm_trn, cm_tst = validacao(clf, (X_train, y_train), (X_teste_Normalizado, Y_teste), confusion_matrix)

with sns.plotting_context(rc={'grid.linewidth': 0}):
    fig, ax = plt.subplots(ncols=2, figsize=(5,2))
    ConfusionMatrixDisplay(confusion_matrix=cm_trn, display_labels=[0,1]).plot(ax=ax[0], colorbar=False)
    ConfusionMatrixDisplay(confusion_matrix=cm_tst, display_labels=[0,1]).plot(ax=ax[1], colorbar=False)
```

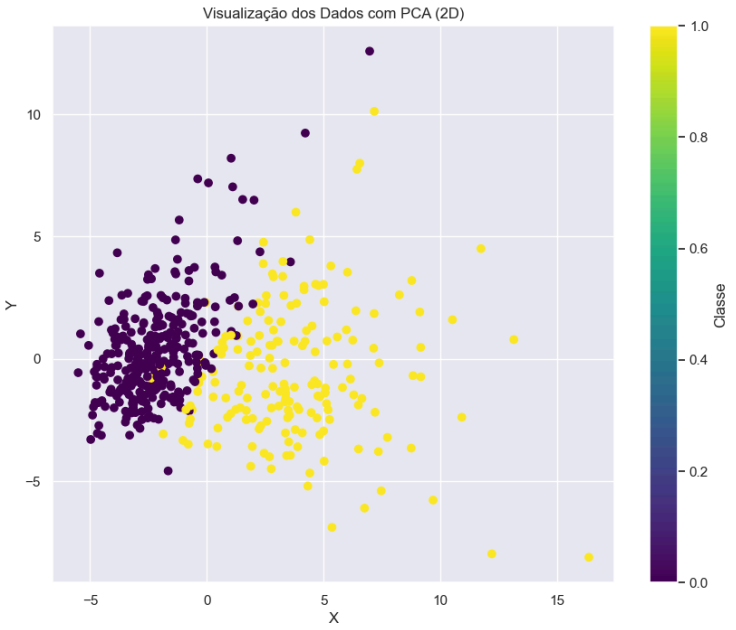
Figura 10: Plot Matriz de confusão



Autor: próprio

Para visualizar os dados em um espaço bidimensional é utilizado a técnica de Análise de Componentes Principais (PCA). Primeiramente, criando uma instância de um objeto que normaliza os dados, garantindo que cada característica tenha média 0 e desvio padrão 1. Em seguida, aplica-se essa normalização ao conjunto de treinamento. Após a normalização, a dimensionalidade dos dados é reduzida para dois componentes principais, visando uma visualização mais simples através de um gráfico de dispersão dos dados reduzidos abaixo:

Figura 11: Gráfico de dispersão dos dados



Autor: próprio

A classificação entre as duas categorias é claramente perceptível, demonstrando como cada instância é distribuída e classificada dentro da dispersão dos dados.

5.4 Acurácia em relação ao número de instâncias no algoritmo K-nn

Visando analisar a dependência da acurácia em relação à variação no número de instâncias para o algoritmo K-nn, será realizado experimentos com diferentes tamanhos de amostras. No Experimento 1, o número de instâncias foi reduzido para 60, enquanto no Experimento 2 foi reduzido para 200, a partir do total original de 569 na base de dados. A relação entre o número de instâncias e a precisão, recall e acurácia é detalhada na Tabela 1.

Tabela 1: Experimentos Instâncias

	Nº Instância	**precisão Treinamento**	**recall Treinamento**	**precisão Teste**	**recall Teste**	**Acurácia Teste**
Experimento 1	**60**	1.00	1.00	0.31	0.56	55%
Experimento 2	**200**	0.97	0.97	0.92	0.90	86%
Padrão	**569**	0.97	0.97	0.98	0.98	96.5%

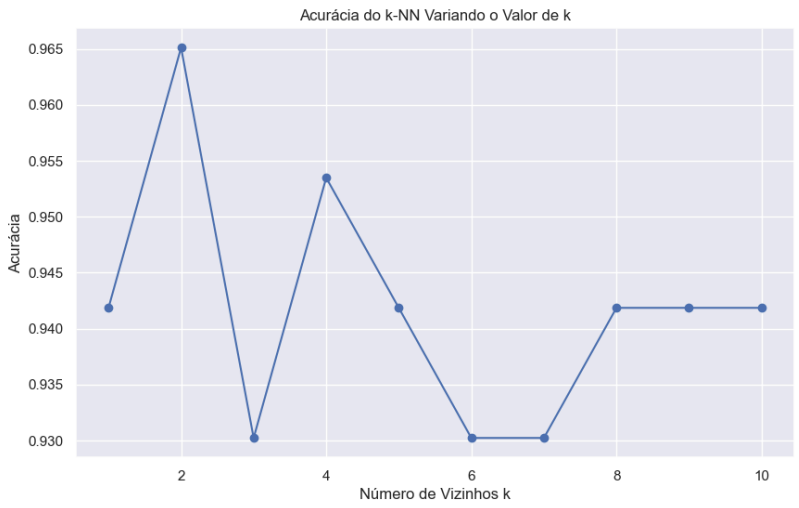
Autor: próprio

Analisando os resultados dos experimentos temos que que, com 60 instâncias, o modelo K-nn sofre de overfitting, com alta precisão e recall no treinamento, mas baixa performance no teste, resultando em 55% de acurácia. Ao aumentar o número de instâncias para 200, a performance melhora significativamente, com uma acurácia de 86%, indicando melhor generalização. Com 569 instâncias, é indubitável que o modelo atinge a melhor configuração, com uma acurácia de 96.5%, demonstrando excelente capacidade de generalização. Assim, aumentar o número de instâncias leva a uma melhor performance e estabilidade do modelo K-nn.

5.5 Acurácia em relação ao número de Vizinhos (k)

Com o objetivo de avaliar como a acurácia do algoritmo K-nn varia conforme o valor de k, que representa o número de vizinhos considerados para a classificação, neste experimento variamos k de 1 a 10 para medir a performance do modelo em diferentes configurações de vizinhança. A relação entre a acurácia e os valores de k é ilustrada na Figura 12.

Figura 12: Acurácia k-nn variando nº de vizinhos



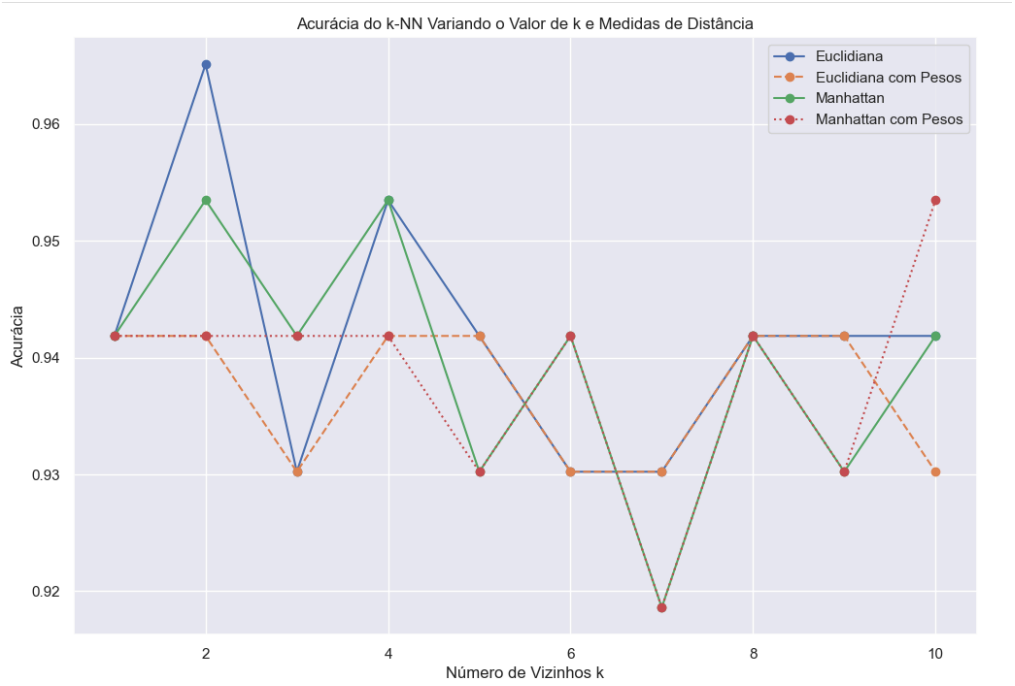
Autor: próprio

Isso nos permite entender qual valor de k proporciona a melhor acurácia para o conjunto de dados específico utilizado, ajudando a determinar um valor ótimo que equilibre o viés e a variância do modelo. Neste estudo, observamos que o algoritmo K-nn alcançou a maior acurácia quando configurado com 2 vizinhos.

5.6 Acurácia relacionada aos pesos baseados em medidas de distância

Nesse tópico, será repetido o experimento anterior que explora a relação entre a acurácia e o número de vizinhos no k-NN adicionados de pesos baseados na distância, utilizando métricas Euclidiana e Manhattan. Figura 13;

Figura 13: Acurácia k-nn variando nº de vizinhos e pesos



Autor: próprio

5.7 Casos de classificação incorreta

Nas figuras 14, 15, 16 e 17 são apresentados os testes alterando a métrica de distâncias, com ou sem pesos na ponderação dos vizinhos:

Figura 14: Distância euclidiana e sem ponderação dos vizinhos

```
k=2, Acurácia=0.9651, Metrica=euclidean, use_weights=False
Classificações incorretas para k=2:
Índice: 23, Classe Real: 0, Predição: 1
Índice: 26, Classe Real: 0, Predição: 1
Índice: 56, Classe Real: 0, Predição: 1

Matriz de Confusão para k=2:
[[49  3]
 [ 0 34]]
```

Autor: próprio

Analisando com base no experimento anterior, figura 14, é mostrado que, utilizando k=2, uma acurácia de 0.9651 foi alcançada com a métrica de distância euclidiana e sem ponderação dos vizinhos.Com as classificações erradas; índices números: 23,26,56. Comparativamente é a melhor configuração encontrada.

Utilizando a métrica de distância euclidiana e com ponderação dos vizinhos, tens-se a acurácia de 0.9419. Com as classificações erradas; índices números: 23,26,35, 56 e 58. Como mostra a figura 15.

Figura 15: Distância euclidiana e com ponderação dos vizinhos

```
k=2, Acurácia=0.9419, Metrica=euclidean, use_weights=True
Classificações incorretas para k=2:
Índice: 23, Classe Real: 0, Predição: 1
Índice: 26, Classe Real: 0, Predição: 1
Índice: 35, Classe Real: 0, Predição: 1
Índice: 56, Classe Real: 0, Predição: 1
Índice: 58, Classe Real: 0, Predição: 1

Matriz de Confusão para k=2:
[[47  5]
 [ 0 34]]
```

Autor: próprio

Para a métrica de distância Manhattan e sem ponderação dos vizinhos, tens-se a acurácia de 0.9535. Com as classificações erradas; índices números: 23,35, 56 e 57. Como mostra a figura 16.

Figura 16: distância Manhattan sem ponderação dos vizinhos

```
k=2, Acurácia=0.9535, Metrica=manhattan, use_weights=False
Classificações incorretas para k=2:
Índice: 23, Classe Real: 0, Predição: 1
Índice: 35, Classe Real: 0, Predição: 1
Índice: 56, Classe Real: 0, Predição: 1
Índice: 57, Classe Real: 1, Predição: 0

Matriz de Confusão para k=2:
[[49  3]
 [ 1 33]]
```

Autor: próprio

Por fim, para a métrica de distância Manhattan com ponderação dos vizinhos, tens-se a acurácia de 0.9419. Com as classificações erradas; índices números: 23,26, 35, 56 e 58. Como mostra a figura 17.

Figura 17: Distância Manhattan com ponderação dos vizinhos

```
k=2, Acurácia=0.9419, Metrica=manhattan, use_weights=True
Classificações incorretas para k=2:
Índice: 23, Classe Real: 0, Predição: 1
Índice: 26, Classe Real: 0, Predição: 1
Índice: 35, Classe Real: 0, Predição: 1
Índice: 56, Classe Real: 0, Predição: 1
Índice: 58, Classe Real: 0, Predição: 1

Matriz de Confusão para k=2:
[[47  5]
 [ 0 34]]
```

Autor: próprio

6. Conclusão

Neste trabalho, foi investigado como diferentes fatores influenciam a acurácia do algoritmo K-nn, incluindo o número de instâncias, o número de vizinhos (valor de k), as medidas de distância utilizadas e o uso de pesos baseados na proximidade dos vizinhos. A análise revelou que a acurácia do K-nn melhora com o aumento do tamanho da base de dados. Com apenas 60 instâncias, o modelo sofreu de overfitting, resultando em baixa performance no teste . Ao aumentar o número de instâncias foi encontrado uma melhor generalização. Esses resultados indicam que aumentar o número de instâncias leva a uma melhor performance e estabilidade do modelo K-nn.

Além disso, variamos o valor de k de 1 a 10 para avaliar seu impacto na acurácia do modelo. Observamos que a escolha de k tem um impacto significativo na performance, com o melhor desempenho obtido com k=2, equilibrando bem o viés e a variância do modelo. A comparação entre diferentes medidas de distância, como Euclidiana e Manhattan, revelou que a métrica Euclidiana sem pesos dos vizinhos, nesse caso, obteve o melhor desempenho no conjunto de dados utilizado.

7. Referências

[1] BRITTO, Larissa FS; PACIFICO, Luciano DS. Plant classification using weighted k-nn variants. In: **Anais do XV Encontro Nacional de Inteligência Artificial e Computacional**. SBC, 2018. p. 58-69.

[2] Wolberg,William, Mangasarian,Olvi, Street,Nick, and Street,W.. (1995). Breast Cancer Wisconsin (Diagnostic). UCI Machine Learning Repository. https://doi.org/10.24432/C5DW2B.