Tarefa1

May 2, 2025

[ ]:

1 Tarefa 1 - Algoritmo K-Nearest-Neighbors

1.0.1 Aluno: José Ivo Schwade Araújo

Importando as bibliotecas necessárias:

[ ]:
```
# importar os pandas para armazenar os dados em dataframes
import pandas as pd
# o numpy é utilizado para realizar algumas operações em dados
import numpy as np
# a funcao train_test_split é usada para dividir o conjunto de dados em treino␣
 ↪e teste
import seaborn as sns

from sklearn.model_selection import train_test_split
# essas duas funcoes sao usadas para analisar os resultados
from sklearn.metrics import classification_report, accuracy_score
# esse pacote contém um conjunto de datasets prontos para serem utilizados
from sklearn import datasets
# Esse pacote contém métodos de normalização de atributos
from sklearn.preprocessing import MinMaxScaler, StandardScaler
# importa o knn
from sklearn.neighbors import KNeighborsClassifier
# para visualizar dados
import matplotlib.pyplot as plt
```

Carregando e visualizando nossa base de dados:

[9]:
```
wine = load_wine()

df = pd.DataFrame(data=wine.data, columns=wine.feature_names)
df["target"] = wine.target

df.head(7)
```

[9]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	\
0	14.23	1.71	2.43	15.6	127.0	2.80	
1	13.20	1.78	2.14	11.2	100.0	2.65	
2	13.16	2.36	2.67	18.6	101.0	2.80	
3	14.37	1.95	2.50	16.8	113.0	3.85	
4	13.24	2.59	2.87	21.0	118.0	2.80	
5	14.20	1.76	2.45	15.2	112.0	3.27	
6	14.39	1.87	2.45	14.6	96.0	2.50	

	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	\
0	3.06	0.28	2.29	5.64	1.04	
1	2.76	0.26	1.28	4.38	1.05	
2	3.24	0.30	2.81	5.68	1.03	
3	3.49	0.24	2.18	7.80	0.86	
4	2.69	0.39	1.82	4.32	1.04	
5	3.39	0.34	1.97	6.75	1.05	
6	2.52	0.30	1.98	5.25	1.02	

	od280/od315_of_diluted_wines	proline	target
0	3.92	1065.0	0
1	3.40	1050.0	0
2	3.17	1185.0	0
3	3.45	1480.0	0
4	2.93	735.0	0
5	2.85	1450.0	0
6	3.58	1290.0	0

Visualizando as estatísticas descritivas:

[10]:
```
wine.DESCR
```

[10]:
```
'.. _wine_dataset:\n\nWine recognition
dataset\n------------------------\n\n**Data Set Characteristics:**\n\n:Number of
Instances: 178\n:Number of Attributes: 13 numeric, predictive attributes and the
class\n:Attribute Information:\n    - Alcohol\n    - Malic acid\n    - Ash\n
- Alcalinity of ash\n    - Magnesium\n    - Total phenols\n    - Flavanoids\n
- Nonflavanoid phenols\n    - Proanthocyanins\n    - Color intensity\n    -
Hue\n    - OD280/OD315 of diluted wines\n    - Proline\n    - class:\n        -
class_0\n        - class_1\n        - class_2\n\n:Summary
Statistics:\n\n============================== ==== ===== ======= =====\n
Min   Max   Mean     SD\n============================== ==== ===== =======
=====\nAlcohol:                     11.0  14.8    13.0   0.8\nMalic Acid:
0.74  5.80    2.34  1.12\nAsh:                           1.36  3.23    2.36
0.27\nAlcalinity of Ash:            10.6  30.0    19.5   3.3\nMagnesium:
70.0 162.0    99.7  14.3\nTotal Phenols:                 0.98  3.88    2.29
0.63\nFlavanoids:                   0.34  5.08    2.03  1.00\nNonflavanoid
Phenols:         0.13  0.66    0.36  0.12\nProanthocyanins:               0.41
3.58    1.59  0.57\nColour Intensity:              1.3  13.0     5.1   2.3\nHue:
```

```
0.48  1.71    0.96  0.23\nOD280/OD315 of diluted wines: 1.27  4.00    2.61
0.71\nProline:                       278  1680     746
315\n============================== ==== ===== ======= =====\n\n:Missing
Attribute Values: None\n:Class Distribution: class_0 (59), class_1 (71), class_2
(48)\n:Creator: R.A. Fisher\n:Donor: Michael Marshall
(MARSHALL%PLU@io.arc.nasa.gov)\n:Date: July, 1988\n\nThis is a copy of UCI ML
Wine recognition datasets.\nhttps://archive.ics.uci.edu/ml/machine-learning-
databases/wine/wine.data\n\nThe data is the results of a chemical analysis of
wines grown in the same\nregion in Italy by three different cultivators. There
are thirteen different\nmeasurements taken for different constituents found in
the three types of\nwine.\n\nOriginal Owners:\n\nForina, M. et al, PARVUS -\nAn
Extendible Package for Data Exploration, Classification and
Correlation.\nInstitute of Pharmaceutical and Food Analysis and
Technologies,\nVia Brigata Salerno, 16147 Genoa, Italy.\n\nCitation:\n\nLichman,
M. (2013). UCI Machine Learning Repository\n[https://archive.ics.uci.edu/ml].
Irvine, CA: University of California,\nSchool of Information and Computer
Science.\n\n.. dropdown:: References\n\n    (1) S. Aeberhard, D. Coomans and O.
de Vel,\n    Comparison of Classifiers in High Dimensional Settings,\n    Tech.
Rep. no. 92-02, (1992), Dept. of Computer Science and Dept. of\n    Mathematics
and Statistics, James Cook University of North Queensland.\n    (Also submitted
to Technometrics).\n\n    The data was used with many others for comparing
various\n    classifiers. The classes are separable, though only RDA\n    has
achieved 100% correct classification.\n    (RDA : 100%, QDA 99.4%, LDA 98.9%,
1NN 96.1% (z-transformed data))\n    (All results using the leave-one-out
technique)\n\n    (2) S. Aeberhard, D. Coomans and O. de Vel,\n    "THE
CLASSIFICATION PERFORMANCE OF RDA"\n    Tech. Rep. no. 92-01, (1992), Dept. of
Computer Science and Dept. of\n    Mathematics and Statistics, James Cook
University of North Queensland.\n    (Also submitted to Journal of
Chemometrics).\n'
```

Separando a base de dados em um conjunto de treinamento e um conjunto de teste:

[74]:
```
X, y = datasets.load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,␣
 ↪random_state=0)
print(f'X_train: {X_train.shape}')
print(f'X_test: {X_test.shape}')
print(f'y_train: {y_train.shape}')
print(f'y_test: {y_test.shape}')
```

```
X_train: (90, 4)
X_test: (60, 4)
y_train: (90,)
y_test: (60,)
```

Vamos calcular agora as acurácias para valores k de 1 a 11 usando as distâncias *euclidian*, *cityblock* e *chebyshev*

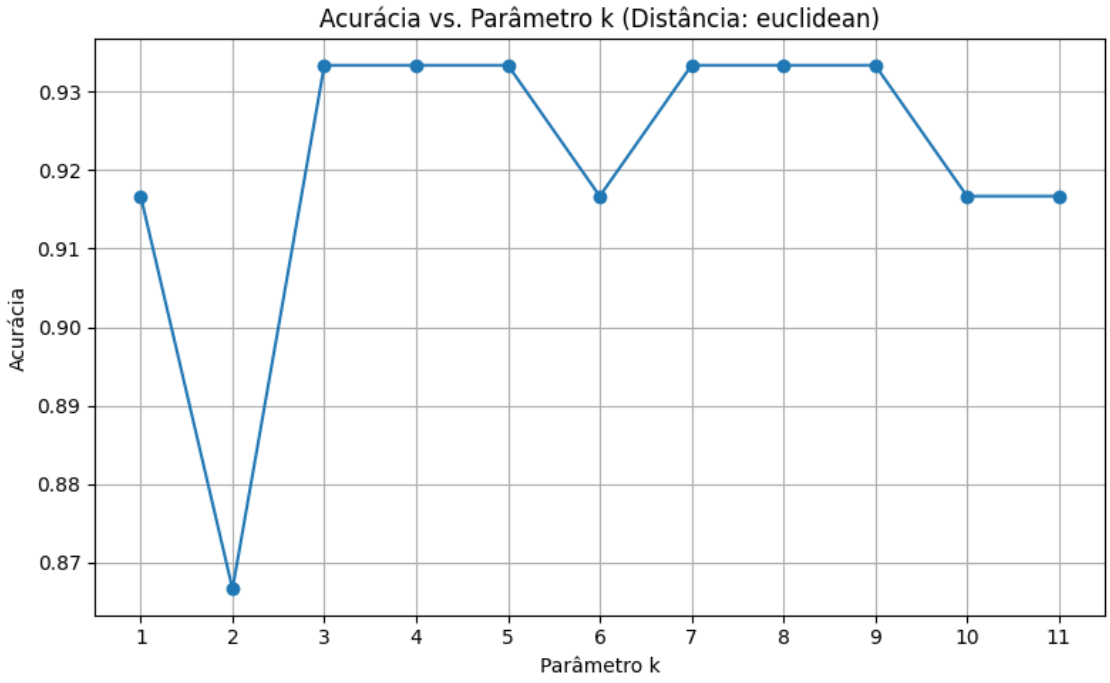
[75]:
```
# Normalizamos os atributos de X
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

# Calculando para cada K e cada distância:

resultados = []

for distance in ['euclidean', 'cityblock', 'chebyshev']:
    for k in range(1,12):
        knn = KNeighborsClassifier(n_neighbors=k, metric=distance)
        knn.fit(X_train, y_train)
        Y_pred = knn.predict(X_test)
        accuracy = accuracy_score(y_test, Y_pred)
        resultados.append({'k': k, 'distance': distance, 'accuracy': accuracy})

display(resultados)
```

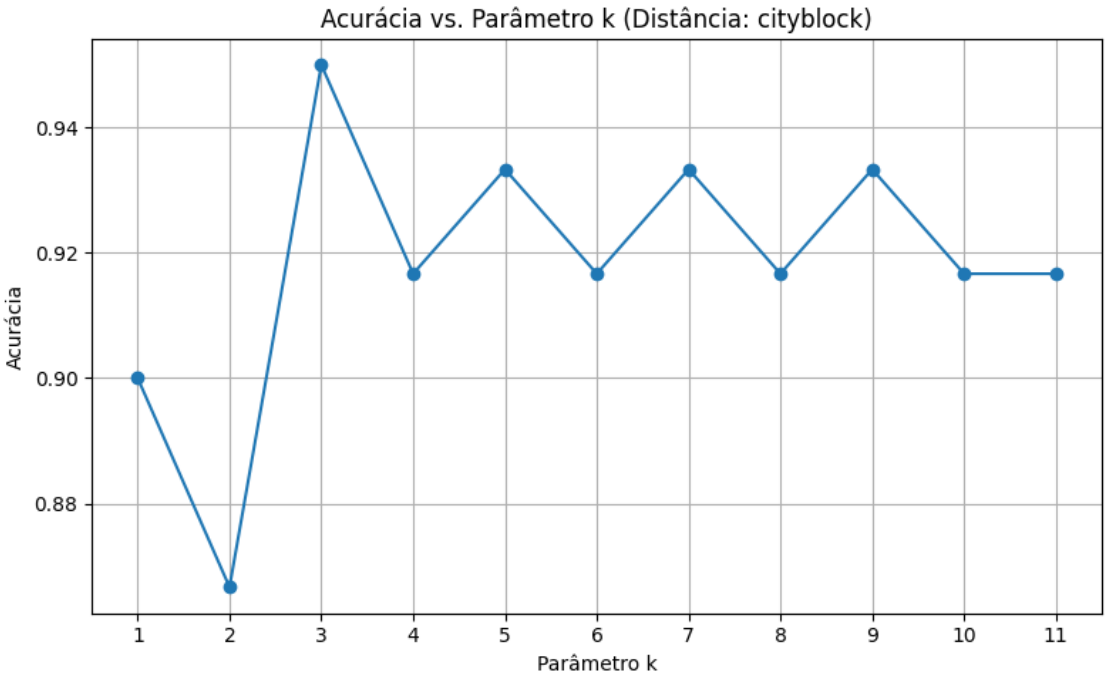
```
[{'k': 1, 'distance': 'euclidean', 'accuracy': 0.9166666666666666},
 {'k': 2, 'distance': 'euclidean', 'accuracy': 0.8666666666666667},
 {'k': 3, 'distance': 'euclidean', 'accuracy': 0.9333333333333333},
 {'k': 4, 'distance': 'euclidean', 'accuracy': 0.9333333333333333},
 {'k': 5, 'distance': 'euclidean', 'accuracy': 0.9333333333333333},
 {'k': 6, 'distance': 'euclidean', 'accuracy': 0.9166666666666666},
 {'k': 7, 'distance': 'euclidean', 'accuracy': 0.9333333333333333},
 {'k': 8, 'distance': 'euclidean', 'accuracy': 0.9333333333333333},
 {'k': 9, 'distance': 'euclidean', 'accuracy': 0.9333333333333333},
 {'k': 10, 'distance': 'euclidean', 'accuracy': 0.9166666666666666},
 {'k': 11, 'distance': 'euclidean', 'accuracy': 0.9166666666666666},
 {'k': 1, 'distance': 'cityblock', 'accuracy': 0.9},
 {'k': 2, 'distance': 'cityblock', 'accuracy': 0.8666666666666667},
 {'k': 3, 'distance': 'cityblock', 'accuracy': 0.95},
 {'k': 4, 'distance': 'cityblock', 'accuracy': 0.9166666666666666},
 {'k': 5, 'distance': 'cityblock', 'accuracy': 0.9333333333333333},
 {'k': 6, 'distance': 'cityblock', 'accuracy': 0.9166666666666666},
 {'k': 7, 'distance': 'cityblock', 'accuracy': 0.9333333333333333},
 {'k': 8, 'distance': 'cityblock', 'accuracy': 0.9166666666666666},
 {'k': 9, 'distance': 'cityblock', 'accuracy': 0.9333333333333333},
 {'k': 10, 'distance': 'cityblock', 'accuracy': 0.9166666666666666},
 {'k': 11, 'distance': 'cityblock', 'accuracy': 0.9166666666666666},
 {'k': 1, 'distance': 'chebyshev', 'accuracy': 0.9166666666666666},
 {'k': 2, 'distance': 'chebyshev', 'accuracy': 0.85},
 {'k': 3, 'distance': 'chebyshev', 'accuracy': 0.9},
 {'k': 4, 'distance': 'chebyshev', 'accuracy': 0.8833333333333333},
 {'k': 5, 'distance': 'chebyshev', 'accuracy': 0.9166666666666666},
 {'k': 6, 'distance': 'chebyshev', 'accuracy': 0.9},
```

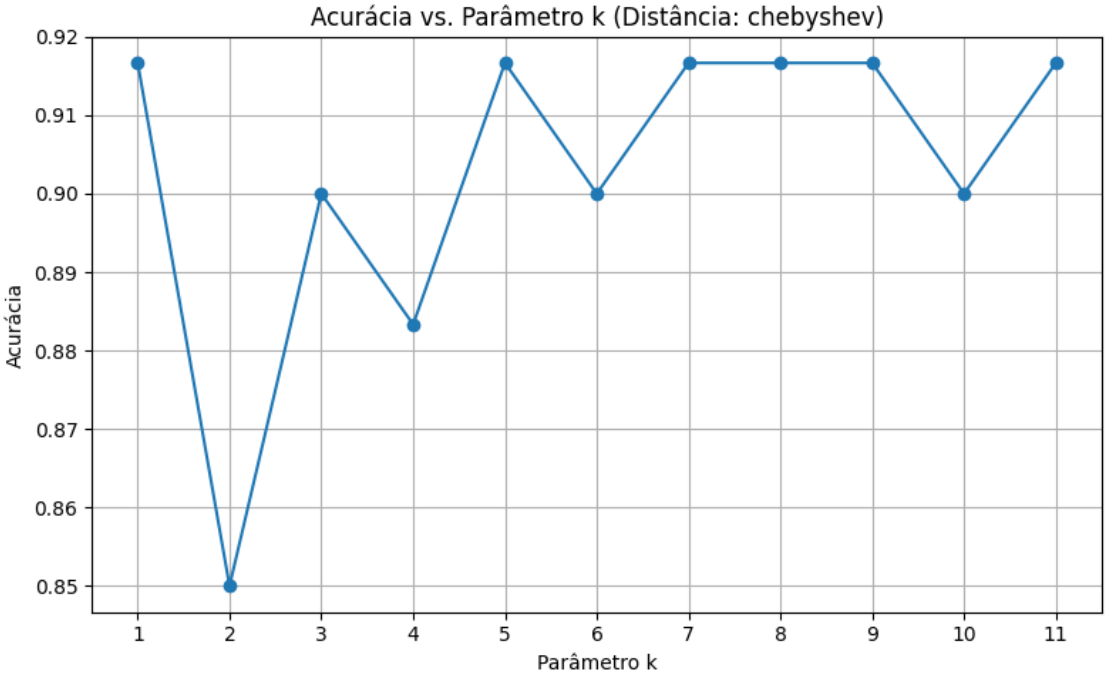
```
{'k': 7, 'distance': 'chebyshev', 'accuracy': 0.9166666666666666},
{'k': 8, 'distance': 'chebyshev', 'accuracy': 0.9166666666666666},
{'k': 9, 'distance': 'chebyshev', 'accuracy': 0.9166666666666666},
{'k': 10, 'distance': 'chebyshev', 'accuracy': 0.9},
{'k': 11, 'distance': 'chebyshev', 'accuracy': 0.9166666666666666}]
```

Verificando a variação da acurácia com o parâmetro K:

[76]:
```
df_resultados = pd.DataFrame(resultados)

# Plotando
distancias = df_resultados['distance'].unique()

for distance in distancias:
    subset = df_resultados[df_resultados['distance'] == distance]

    plt.figure(figsize=(8, 5))
    plt.plot(subset['k'], subset['accuracy'], marker='o')
    plt.xticks(np.arange(1, 12))
    plt.xlabel('Parâmetro k')
    plt.ylabel('Acurácia')
    plt.title(f'Acurácia vs. Parâmetro k (Distância: {distance})')
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```







O melhor modelo é, aparentemente, utilizando a distância *cityblock* e parâmetro k = 3.

[77]:
```
melhor_modelo = df_resultados.loc[df_resultados['accuracy'].idxmax()]
print("Melhor modelo encontrado:")
print(melhor_modelo)
```

```
Melhor modelo encontrado:
k                   3
distance    cityblock
accuracy         0.95
Name: 13, dtype: object
```

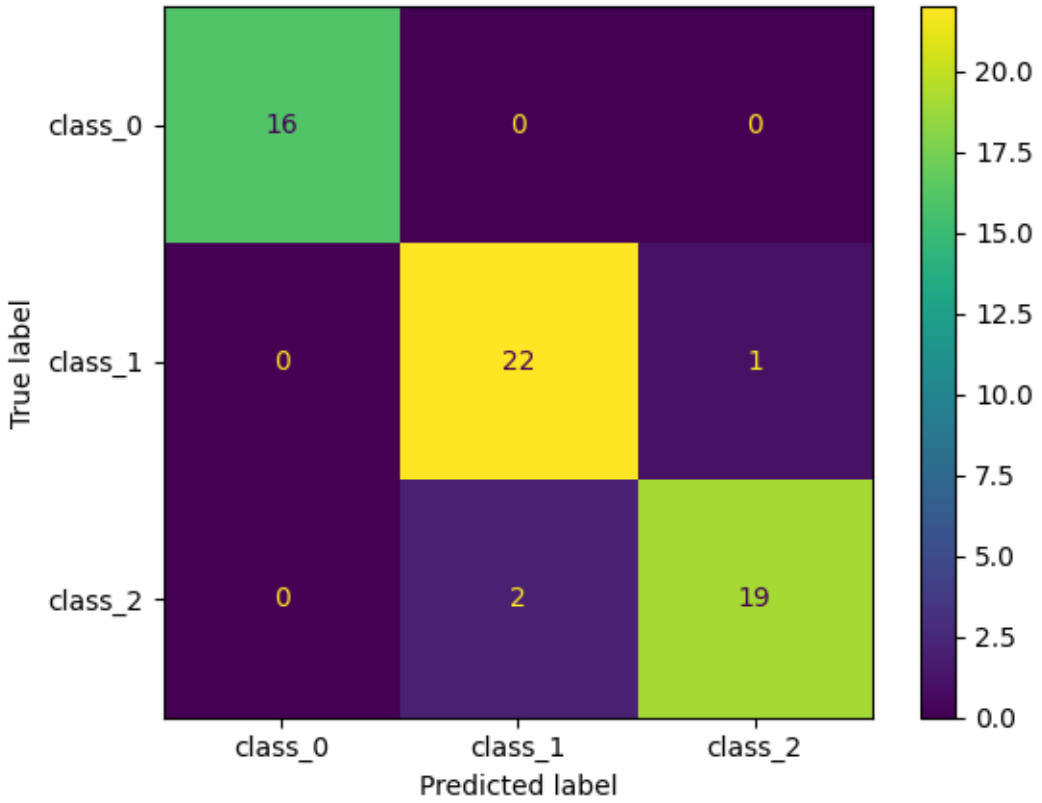
Treinando o modelo:

[78]:
```
melhor_k = melhor_modelo["k"]
melhor_distancia = melhor_modelo["distance"]
knn = KNeighborsClassifier(n_neighbors=melhor_k, metric=melhor_distancia)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
clr = classification_report(y_test, y_pred)
print(clr)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       0.92      0.96      0.94        23
           2       0.95      0.90      0.93        21

    accuracy                           0.95        60
   macro avg       0.96      0.95      0.95        60
weighted avg       0.95      0.95      0.95        60
```

Plotando a matriz de confusão:

[79]:
```
# importa uma função para melhor visualizar a matriz de confusão
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
# calcula a matriz de confusão
cm = confusion_matrix(y_test,y_pred)
# constroi um versão mais otimizada para visualização da matriz de confusão
cmd = ConfusionMatrixDisplay(cm, display_labels=list(wine.target_names))
# plota a matriz de confusão
cmd.plot()
plt.show()
```



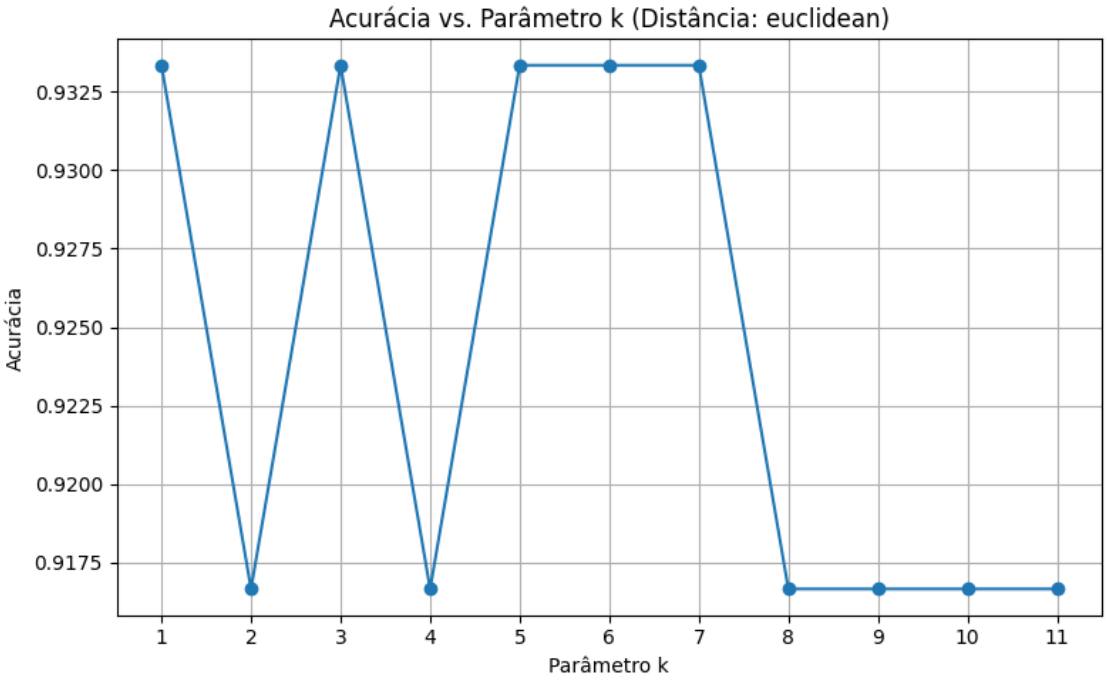
Aplicando ambas normalizações e treinando o modelo novamente:

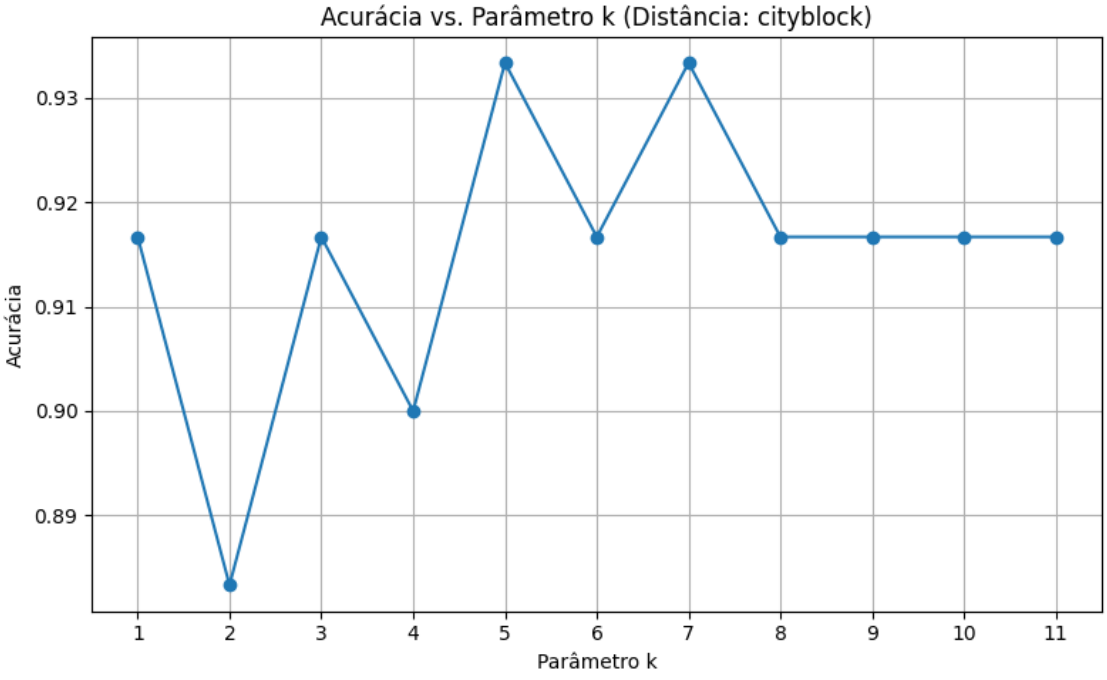
[80]:
```
# Normalizamos os atributos de X
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

# Calculando para cada K e cada distância:

resultados = []

for distance in ['euclidean', 'cityblock', 'chebyshev']:
    for k in range(1,12):
        knn = KNeighborsClassifier(n_neighbors=k, metric=distance)
        knn.fit(X_train, y_train)
        Y_pred = knn.predict(X_test)
        accuracy = accuracy_score(y_test, Y_pred)
        resultados.append({'k': k, 'distance': distance, 'accuracy': accuracy})

display(resultados)
```

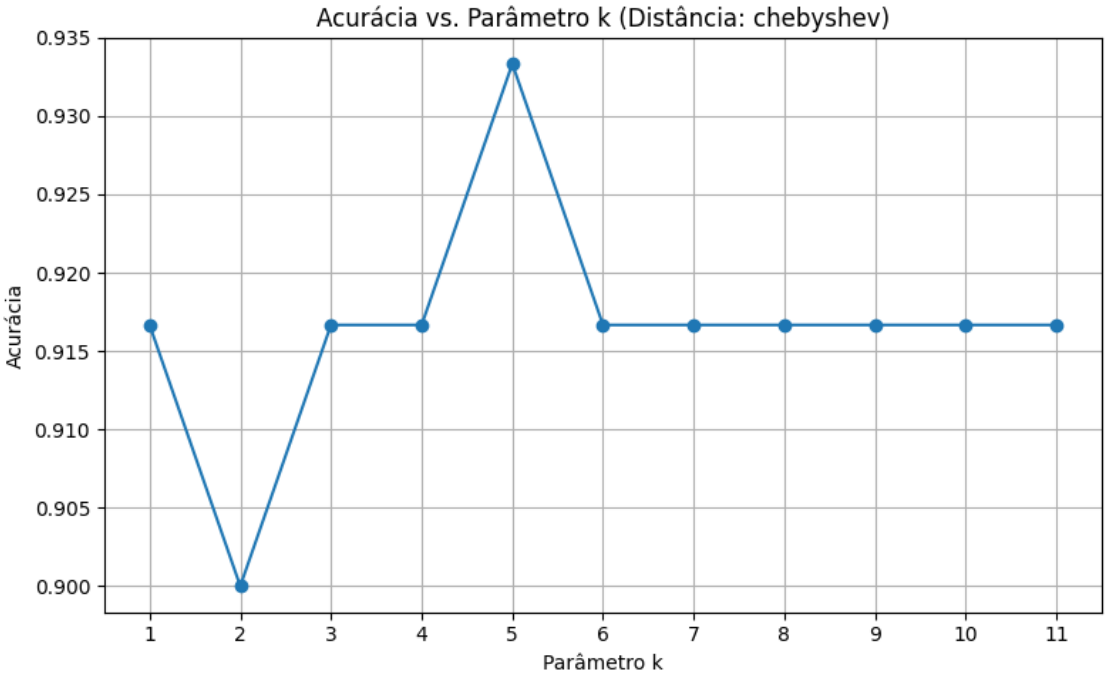
```
[{'k': 1, 'distance': 'euclidean', 'accuracy': 0.9333333333333333},
 {'k': 2, 'distance': 'euclidean', 'accuracy': 0.9166666666666666},
 {'k': 3, 'distance': 'euclidean', 'accuracy': 0.9333333333333333},
 {'k': 4, 'distance': 'euclidean', 'accuracy': 0.9166666666666666},
 {'k': 5, 'distance': 'euclidean', 'accuracy': 0.9333333333333333},
 {'k': 6, 'distance': 'euclidean', 'accuracy': 0.9333333333333333},
 {'k': 7, 'distance': 'euclidean', 'accuracy': 0.9333333333333333},
 {'k': 8, 'distance': 'euclidean', 'accuracy': 0.9166666666666666},
 {'k': 9, 'distance': 'euclidean', 'accuracy': 0.9166666666666666},
 {'k': 10, 'distance': 'euclidean', 'accuracy': 0.9166666666666666},
 {'k': 11, 'distance': 'euclidean', 'accuracy': 0.9166666666666666},
 {'k': 1, 'distance': 'cityblock', 'accuracy': 0.9166666666666666},
 {'k': 2, 'distance': 'cityblock', 'accuracy': 0.8833333333333333},
 {'k': 3, 'distance': 'cityblock', 'accuracy': 0.9166666666666666},
 {'k': 4, 'distance': 'cityblock', 'accuracy': 0.9},
 {'k': 5, 'distance': 'cityblock', 'accuracy': 0.9333333333333333},
 {'k': 6, 'distance': 'cityblock', 'accuracy': 0.9166666666666666},
 {'k': 7, 'distance': 'cityblock', 'accuracy': 0.9333333333333333},
 {'k': 8, 'distance': 'cityblock', 'accuracy': 0.9166666666666666},
 {'k': 9, 'distance': 'cityblock', 'accuracy': 0.9166666666666666},
 {'k': 10, 'distance': 'cityblock', 'accuracy': 0.9166666666666666},
 {'k': 11, 'distance': 'cityblock', 'accuracy': 0.9166666666666666},
 {'k': 1, 'distance': 'chebyshev', 'accuracy': 0.9166666666666666},
 {'k': 2, 'distance': 'chebyshev', 'accuracy': 0.9},
 {'k': 3, 'distance': 'chebyshev', 'accuracy': 0.9166666666666666},
 {'k': 4, 'distance': 'chebyshev', 'accuracy': 0.9166666666666666},
 {'k': 5, 'distance': 'chebyshev', 'accuracy': 0.9333333333333333},
 {'k': 6, 'distance': 'chebyshev', 'accuracy': 0.9166666666666666},
 {'k': 7, 'distance': 'chebyshev', 'accuracy': 0.9166666666666666},
 {'k': 8, 'distance': 'chebyshev', 'accuracy': 0.9166666666666666},
 {'k': 9, 'distance': 'chebyshev', 'accuracy': 0.9166666666666666},
 {'k': 10, 'distance': 'chebyshev', 'accuracy': 0.9166666666666666},
 {'k': 11, 'distance': 'chebyshev', 'accuracy': 0.9166666666666666}]
```

[81]:
```python
df_resultados = pd.DataFrame(resultados)

# Plotando
distancias = df_resultados['distance'].unique()

for distance in distancias:
    subset = df_resultados[df_resultados['distance'] == distance]

    plt.figure(figsize=(8, 5))
    plt.plot(subset['k'], subset['accuracy'], marker='o')
    plt.xticks(np.arange(1, 12))
    plt.xlabel('Parâmetro k')
    plt.ylabel('Acurácia')
```

```
    plt.title(f'Acurácia vs. Parâmetro k (Distância: {distance})')
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```







[82]:
```
melhor_modelo = df_resultados.loc[df_resultados['accuracy'].idxmax()]
print("Melhor modelo encontrado:")
print(melhor_modelo)
```

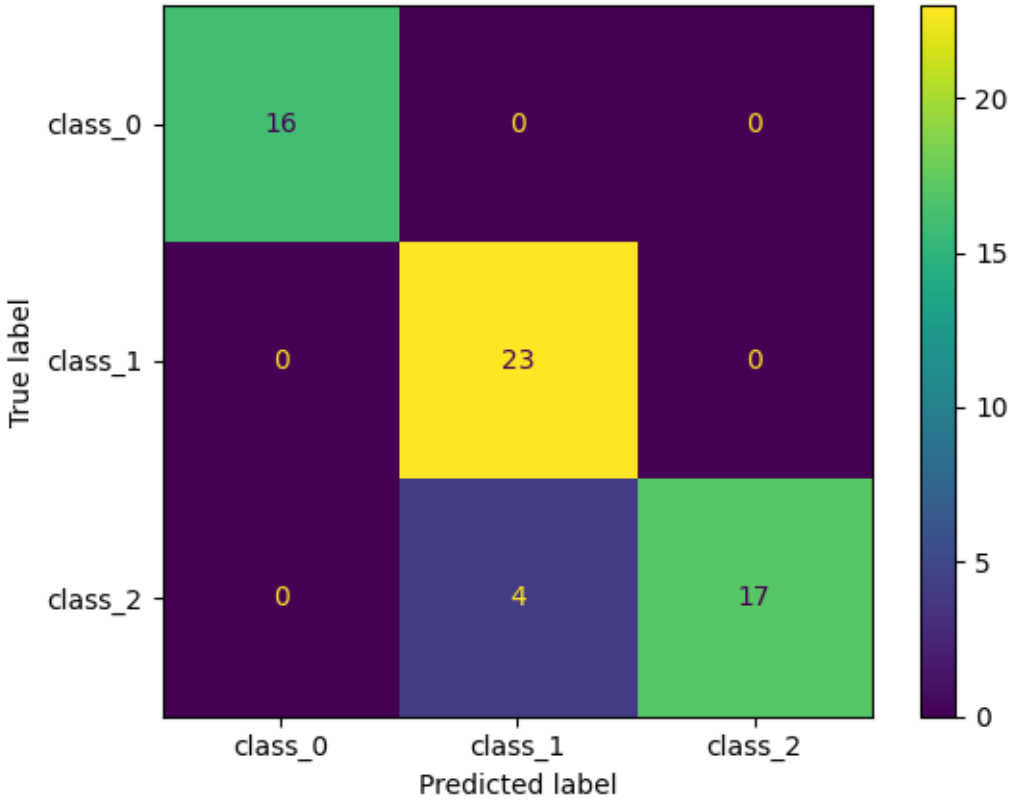
```
Melhor modelo encontrado:
k                   1
distance    euclidean
accuracy     0.933333
Name: 0, dtype: object
```

[83]:
```
melhor_k = melhor_modelo["k"]
melhor_distancia = melhor_modelo["distance"]
knn = KNeighborsClassifier(n_neighbors=melhor_k, metric=melhor_distancia)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
clr = classification_report(y_test, y_pred)
print(clr)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       0.85      1.00      0.92        23
           2       1.00      0.81      0.89        21
```

```
    accuracy                           0.93        60
   macro avg       0.95      0.94      0.94        60
weighted avg       0.94      0.93      0.93        60
```

[84]:
```
# importa uma função para melhor visualizar a matriz de confusão
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
# calcula a matriz de confusão
cm = confusion_matrix(y_test,y_pred)
# constroi um versão mais otimizada para visualização da matriz de confusão
cmd = ConfusionMatrixDisplay(cm, display_labels=list(wine.target_names))
# plota a matriz de confusão
cmd.plot()
plt.show()
```



1.1 Conclusão

Podemos concluir que o melhor modelo é o primeiro, utilizando a normalização StandardScaler, a distância *cityblock* e o parâmetro k = 3, devido a sua menor quantidade de erros analisada na matriz de confusão.

Tarefa2

May 2, 2025

1 Tarefa 2: Algoritmo Naive Bayes

1.0.1 Aluno: José Ivo Schwade Araújo

1.0.2 0. Importando as bibliotecas necessárias e carregando a base de dados

[56]:
```
# Importação das bibliotecas necessárias
import numpy as np                                          # Biblioteca para␣
 ↪operações numéricas
import pandas as pd                                         # Biblioteca para␣
 ↪manipulação de dados em DataFrames
from sklearn.model_selection import train_test_split        # Função para␣
 ↪dividir os dados em treino e teste
from sklearn.feature_extraction.text import CountVectorizer   # Transforma␣
 ↪texto em representações vetoriais
from sklearn.naive_bayes import BernoulliNB                 # Classificador␣
 ↪Naive Bayes com distribuição de Bernoulli
from sklearn.neighbors import KNeighborsClassifier          # Classificador␣
 ↪k-NN
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report  # Métricas de avaliação
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_digits
import matplotlib.pyplot as plt                             # Biblioteca de␣
 ↪visualização gráfica
import seaborn as sns                                       # Biblioteca de␣
 ↪visualização baseada em matplotlib, com estilo melhorado
# Para figuras geradas pelo matplotlib serem exibidas diretamente no notebook
%matplotlib inline
```

[47]:
```
# Carregamento do dataset de spam
url = "https://raw.githubusercontent.com/justmarkham/pycon-2016-tutorial/master/
 ↪data/sms.tsv"
df = pd.read_csv(url, sep='\t', header=None, names=['label', 'message'])

# Visualização dos dados
print("Visualização das primeiras linhas do dataset:")
display(df.head(11))
```

Visualização das primeiras linhas do dataset:

```
   label                                            message
0    ham  Go until jurong point, crazy.. Available only …
1    ham                      Ok lar… Joking wif u oni…
2   spam  Free entry in 2 a wkly comp to win FA Cup fina…
3    ham  U dun say so early hor… U c already then say…
4    ham  Nah I don't think he goes to usf, he lives aro…
5   spam  FreeMsg Hey there darling it's been 3 week's n…
6    ham  Even my brother is not like to speak with me. …
7    ham  As per your request 'Melle Melle (Oru Minnamin…
8   spam  WINNER!! As a valued network customer you have…
9   spam  Had your mobile 11 months or more? U R entitle…
10   ham  I'm gonna be home soon and i don't want to tal…
```

1.0.3 1. Testando outras métricas do KNN na base de dados SMS Spam Collection

Pré-processamento dos dados

[48]:
```
# Conversão das labels para valores binários
df['label'] = df['label'].map({'ham': 0, 'spam': 1})

# Divisão dos dados em treino e teste
X_train_spam, X_test_spam, y_train_spam, y_test_spam = train_test_split(  # Divide dados em treino (70%) e teste (30%)
    df['message'],  # Variável independente: textos das mensagens
    df['label'],    # Variável dependente: rótulo binário
    test_size=0.3,  # Proporção de teste: 30%
    random_state=42 # Semente para reprodutibilidade dos resultados
)

# Transformação dos textos em vetores de contagem de palavras (binário)
vectorizer = CountVectorizer(binary=True)          # Inicializa o vetor com contagem binária (presença/ausência da palavra)
X_train_vec = vectorizer.fit_transform(X_train_spam)   # Ajusta e transforma os dados de treino em vetores binários
X_test_vec = vectorizer.transform(X_test_spam)         # Transforma os dados de teste com o mesmo vocabulário aprendido no treino
```

Normalização dos dados para o KNN

[49]:
```
scaler = StandardScaler(with_mean=False)
X_train_scaled = scaler.fit_transform(X_train_vec)
X_test_scaled = scaler.transform(X_test_vec)
```

Testando, finalmente, outras métricas para o KNN

[50]:
```
resultadosKNN = []
# Métrica Cosine sendo testada no lugar da Chebyshev pois estamos tratando dados esparsos.
```

```
distances = ['euclidean', 'cityblock', 'cosine']
for distance in distances:
  KNN = KNeighborsClassifier(metric=distance)
  KNN.fit(X_train_scaled, y_train_spam)
  y_pred = KNN.predict(X_test_scaled)
  accuracy = accuracy_score(y_test_spam, y_pred)
  resultadosKNN.append({
      'distance' : distance,
      'accuracy' : accuracy
  })

df_resultadosKNN = pd.DataFrame(resultadosKNN)
print(df_resultadosKNN.sort_values(by='accuracy', ascending=False))
```

```
    distance  accuracy
2     cosine  0.955742
1  cityblock  0.895933
0  euclidean  0.879785
```

Portanto, temos que a melhor métrica para o modelo é a `cosine`, com um total de 0.955742 de precisão.

1.0.4 2.1 Carregando a base `digits` e pré-processando seus dados

[66]:
```
from ucimlrepo import fetch_ucirepo
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix
import numpy as np

ORHD = fetch_ucirepo(id=80)

features = ORHD.data.features.to_numpy().astype(np.float32) / 16.0
features = (features >= 0.5).astype(np.int64)  # binariza para BernoulliNB
targets = ORHD.data.targets.to_numpy().astype(np.int64).ravel()

X_train_digits, X_test_digits, y_train_digits, y_test_digits =␣
 ↪train_test_split(features, targets, test_size=0.3, random_state=42)
```

1.0.5 2.2 Treinando um modelo Naive Bayes (Bernoulli)

[77]:
```
model = BernoulliNB()
model.fit(X_train_digits, y_train_digits)

y_pred_NB = model.predict(X_test_digits)
accuracy_NB = accuracy_score(y_test_digits, y_pred_NB)
classification_report_NB = classification_report(y_test_digits, y_pred_NB)
```

```
confusion_matrix_NB = confusion_matrix(y_test_digits, y_pred_NB)

print("Acurácia:", accuracy_NB)
print("Relatório:\n", classification_report_NB)
print("Matriz de Confusão:\n", confusion_matrix_NB)
```

```
Acurácia: 0.8855278766310795
Relatório:
               precision    recall  f1-score   support

           0       0.99      0.97      0.98       170
           1       0.83      0.79      0.81       173
           2       0.92      0.87      0.90       154
           3       0.95      0.83      0.88       173
           4       0.94      0.88      0.91       182
           5       0.91      0.90      0.90       153
           6       0.94      0.98      0.96       168
           7       0.86      0.96      0.91       186
           8       0.75      0.84      0.79       153
           9       0.79      0.83      0.81       174

    accuracy                           0.89      1686
   macro avg       0.89      0.88      0.89      1686
weighted avg       0.89      0.89      0.89      1686

Matriz de Confusão:
 [[165   0   1   0   2   0   1   0   0   1]
 [  0 137   6   1   0   1   6   0  16   6]
 [  0   3 134   1   0   0   0   3  10   3]
 [  0   1   1 143   0   6   0   4   6  12]
 [  0   1   0   0 161   1   2  10   4   3]
 [  0   1   1   3   1 138   0   0   0   9]
 [  1   3   0   0   0   0 164   0   0   0]
 [  0   4   0   0   3   0   0 179   0   0]
 [  0  12   2   1   1   3   1   1 128   4]
 [  0   3   0   2   4   3   0  12   6 144]]
```

1.0.6 3.1 Treinando um modelo KNN

[76]:
```
distances = ['euclidean', 'cityblock', 'cosine']
resultadosKNN = []
for distance in distances:
  KNN = KNeighborsClassifier(metric=distance)
  y_pred_KNN = KNN.fit(X_train_digits, y_train_digits).predict(X_test_digits)
  accuracy_KNN = accuracy_score(y_test_digits, y_pred_KNN)
  classification_report_KNN = classification_report(y_test_digits, y_pred_KNN)
  confusion_matrix_KNN = confusion_matrix(y_test_digits, y_pred_KNN)
```

```
print("Acurácia:", accuracy_KNN)
print("Relatório:\n", classification_report_KNN)
print("Matriz de Confusão:\n", confusion_matrix_KNN)
```

```
Acurácia: 0.9406880189798339
Relatório:
               precision    recall  f1-score   support

           0       0.99      0.99      0.99       170
           1       0.83      0.98      0.90       173
           2       0.99      0.97      0.98       154
           3       0.93      0.88      0.91       173
           4       0.99      0.95      0.97       182
           5       0.90      0.96      0.93       153
           6       0.97      0.99      0.98       168
           7       0.97      0.98      0.97       186
           8       0.92      0.85      0.88       153
           9       0.94      0.85      0.89       174

    accuracy                           0.94      1686
   macro avg       0.94      0.94      0.94      1686
weighted avg       0.94      0.94      0.94      1686

Matriz de Confusão:
 [[169   1   0   0   0   0   0   0   0   0]
 [  0 169   0   0   0   0   0   0   4   0]
 [  0   0 149   1   0   0   0   1   2   1]
 [  0   2   1 153   0   8   0   0   3   6]
 [  0   2   0   0 173   1   2   2   1   1]
 [  0   3   0   2   0 147   0   0   0   1]
 [  1   1   0   0   0   0 166   0   0   0]
 [  0   4   0   0   0   0   0 182   0   0]
 [  1  18   0   0   0   1   3   0 130   0]
 [  0   3   0   9   1   7   1   3   2 148]]
```

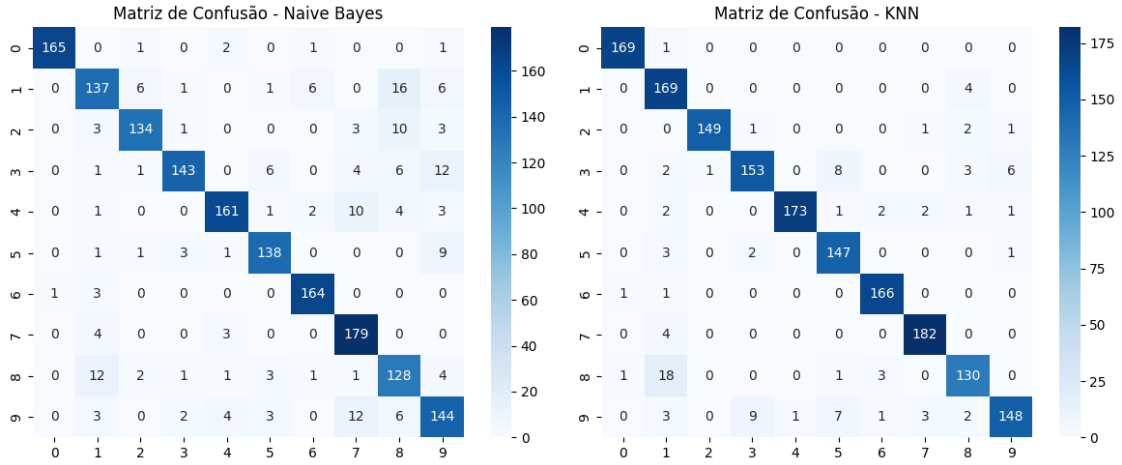
1.0.7 3.2 Comparando os resultados com o item 2

[79]:
```
print("Modelo Naive Bayes")
print("Acurácia:", accuracy_NB)
print("Relatório:\n", classification_report_NB)
print("Modelo KNN")
print("Acurácia:", accuracy_KNN)
print("Relatório:\n", classification_report_KNN)

# Plotar as duas matrizes de confusão em heatmap
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
```

```
sns.heatmap(confusion_matrix(y_test_digits, y_pred_NB), annot=True, fmt='d',␣
 ↪cmap='Blues', ax=axes[0])
axes[0].set_title('Matriz de Confusão - Naive Bayes')
axes[0].set

sns.heatmap(confusion_matrix(y_test_digits, y_pred_KNN), annot=True, fmt='d',␣
 ↪cmap='Blues', ax=axes[1])
axes[1].set_title('Matriz de Confusão - KNN')

plt.tight_layout()
```

```
Modelo Naive Bayes
Acurácia: 0.8855278766310795
Relatório:
              precision    recall  f1-score   support

           0       0.99      0.97      0.98       170
           1       0.83      0.79      0.81       173
           2       0.92      0.87      0.90       154
           3       0.95      0.83      0.88       173
           4       0.94      0.88      0.91       182
           5       0.91      0.90      0.90       153
           6       0.94      0.98      0.96       168
           7       0.86      0.96      0.91       186
           8       0.75      0.84      0.79       153
           9       0.79      0.83      0.81       174

    accuracy                           0.89      1686
   macro avg       0.89      0.88      0.89      1686
weighted avg       0.89      0.89      0.89      1686

Modelo KNN
Acurácia: 0.9406880189798339
Relatório:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       170
           1       0.83      0.98      0.90       173
           2       0.99      0.97      0.98       154
           3       0.93      0.88      0.91       173
           4       0.99      0.95      0.97       182
           5       0.90      0.96      0.93       153
           6       0.97      0.99      0.98       168
           7       0.97      0.98      0.97       186
           8       0.92      0.85      0.88       153
           9       0.94      0.85      0.89       174
```

```
    accuracy                           0.94      1686
   macro avg       0.94      0.94      0.94      1686
weighted avg       0.94      0.94      0.94      1686
```



Portanto, o modelo utilizando o algoritmo KNN se saiu melhor na classificação dos dígitos.

[ ]:

Tarefa3

May 2, 2025

1 Tarefa 3 - Árvores de Decisão

1.1 Aluno: José Ivo Schwade Araújo

1.1.1 0. Preparando o ambiente do colab e baixando a base de dados

[29]:
```
# Importando as bibliotecas necessárias:
!pip install mahotas gdown tqdm
import mahotas
import os
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
import gdown
from tqdm.auto import tqdm
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
import math
```

[2]:
```
# função que extrai as características da imagem
def describe(image):
        # extrai a média e desvio padrão de cada canal do espaço HSV.
        (means, stds) = cv2.meanStdDev(cv2.cvtColor(image, cv2.COLOR_BGR2HSV))
        colorStats = np.concatenate([means, stds]).flatten()
        # converte a imagem para a escala de cinza
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        # extrai Haralick texture features
```

```
        haralick = mahotas.features.haralick(gray).mean(axis=0)
        # retorna um vetor formado por estatísticas básicas e das Haralick␣
 ↪texture features
        return np.hstack([colorStats, haralick])  # colorStats=(6,)␣
 ↪haralick=(13,)
```

[3]:
```
url = 'https://drive.google.com/file/d/1bHHgeZ3H75oigySkcZG0S98unOJNft0b/view?
 ↪usp=sharing'
output = '4scenes.zip'
gdown.download(url=url, output=output, quiet=False, fuzzy=True)

# descompactando o conjunto de dados.
# Se já descompactou antes, descomente a linha abaixo
#!rm -rf '4scenes'
!unzip -q '/content/4scenes.zip'
```

```
Downloading…
From (original):
https://drive.google.com/uc?id=1bHHgeZ3H75oigySkcZG0S98unOJNft0b
From (redirected): https://drive.google.com/uc?id=1bHHgeZ3H75oigySkcZG0S98unOJNf
t0b&confirm=t&uuid=00ff1530-d6a9-4919-a471-761de8a1a552
To: /content/4scenes.zip
100%|      | 54.8M/54.8M [00:00<00:00, 85.5MB/s]
```

[4]:
```
# construindo a base de dados a partir das features extraídas do conjunto␣
 ↪4scenes

imagesPath = './4scenes'
labels = []
data = []

# lista todos os nomes de arquivos presentes no diretório especificado em␣
 ↪imagesPath
# o resultado (files) é uma lista de strings, onde cada string é o nome de um␣
 ↪arquivo
files = os.listdir(imagesPath)

for image_name in tqdm(files, desc="[INFO] Processando imagens"):
  # extraindo o rótulo das imagens (primeira parte do nome do arquivo)
  label = image_name.split("_")[0]
  # carregando a imagem
  image = cv2.imread(imagesPath + '/' + image_name)
  # extraindo as features da imagem
  features = describe(image)
  # adicionando o rótulo da imagem (classe) em uma lista de rótulos
  labels.append(label)
  # adicionando a imagem (features) ao conjunto de dados
```

```
  data.append(features)
```

```
[INFO] Processando imagens:   0%|          | 0/1240 [00:00<?, ?it/s]
```

1.1.2 1. Dividindo a base de dados em subconjuntos de treinamento, validação e teste.

[9]:
```
(trainData, x, trainLabels, y) = train_test_split(np.array(data), np.
 ↪array(labels), test_size=0.30, random_state=42)
(testData, valData, testLabels, valLabels) = train_test_split(x, y, test_size=0.
 ↪5, random_state=42)
```

1.1.3 2. Selecionar a melhor configuração de hiperparâmetros.

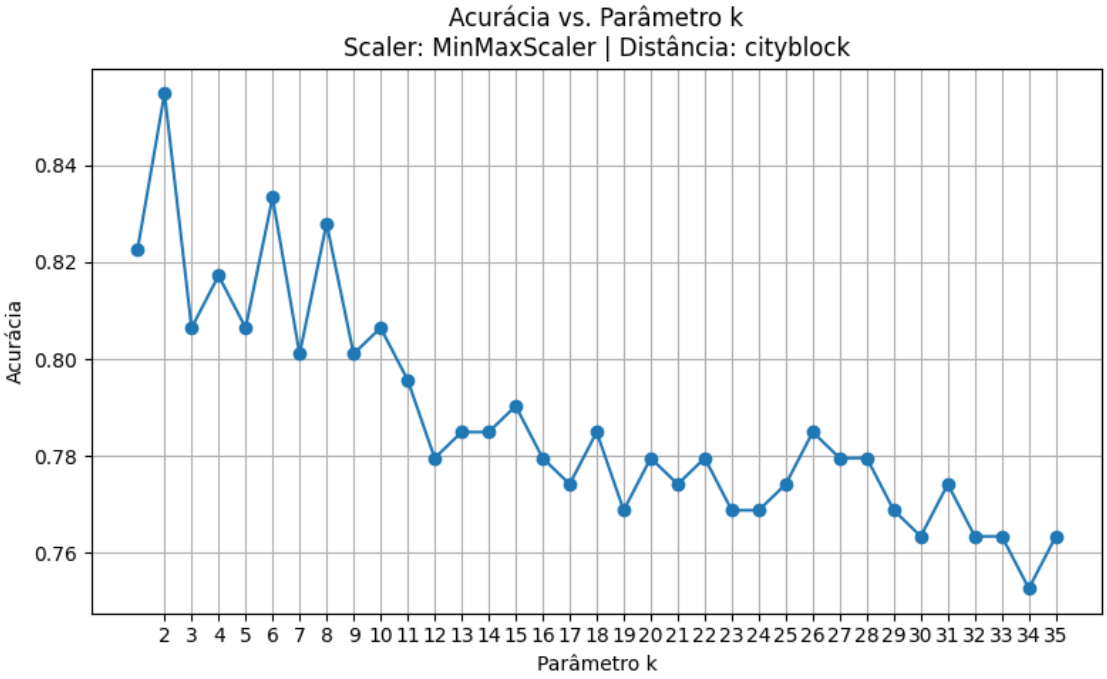
Vamos, então, utilizar o conjunto de treino e o conjunto de validação para encontrar a melhor configuração possível para nossos algoritmos de KNN e de Árvore de Decisão.

1.1.4 KNN:

[10]:
```
# Codificando labels em números
labelEncoder = LabelEncoder()
trainLabels = labelEncoder.fit_transform(trainLabels)
testLabels = labelEncoder.transform(testLabels)

resultadosKNN = []

scalers = {'StandardScaler': StandardScaler(), 'MinMaxScaler': MinMaxScaler()}
distances = ['euclidean', 'cityblock', 'chebyshev']
ks = range(1, int(math.sqrt(len(data))) + 1)
# Normalizando utilizando cada Scaler
for scaler_name, scaler in scalers.items():
    X_train = scaler.fit_transform(trainData)
    X_val = scaler.transform(testData)

    K, D = np.meshgrid(ks, distances, indexing='ij')
    for k, distance in zip(K.flatten(), D.flatten()):
        distance = str(distance)
        knn = KNeighborsClassifier(n_neighbors=k, metric=distance)
        knn.fit(X_train, trainLabels)
        Y_pred = knn.predict(X_val)
        accuracy = accuracy_score(testLabels, Y_pred)
        resultadosKNN.append({
            'scaler': scaler_name,
            'k': k,
            'distance': distance,
            'accuracy': accuracy
        })
```

```
df_resultadosKNN = pd.DataFrame(resultadosKNN)

# Selecionando a melhor combinação de parâmetros:
print(df_resultadosKNN.sort_values(by='accuracy', ascending=False).head(1))

# Plotando o gráfico da melhor combinação de scaler e métrica para melhor␣
 ↪visualização:
scaler_name = df_resultadosKNN.sort_values(by='accuracy',␣
 ↪ascending=False)['scaler'].iloc[0]
distance = df_resultadosKNN.sort_values(by='accuracy',␣
 ↪ascending=False)['distance'].iloc[0]
subset = df_resultadosKNN[(df_resultadosKNN['scaler'] == scaler_name) &␣
 ↪(df_resultadosKNN['distance'] == distance)]

plt.figure(figsize=(8, 5))
plt.plot(subset['k'], subset['accuracy'], marker='o')
plt.xticks(np.arange(1, int(math.sqrt(len(data)))) + 1)
plt.xlabel('Parâmetro k')
plt.ylabel('Acurácia')
plt.title(f'Acurácia vs. Parâmetro k\nScaler: {scaler_name} | Distância:␣
 ↪{distance}')
plt.grid(True)
plt.tight_layout()
plt.show()

# Decodificando as labels para uso posterior
trainLabels = labelEncoder.inverse_transform(trainLabels)
testLabels = labelEncoder.inverse_transform(testLabels)
```

```
          scaler  k   distance  accuracy
109  MinMaxScaler  2  cityblock  0.854839
```



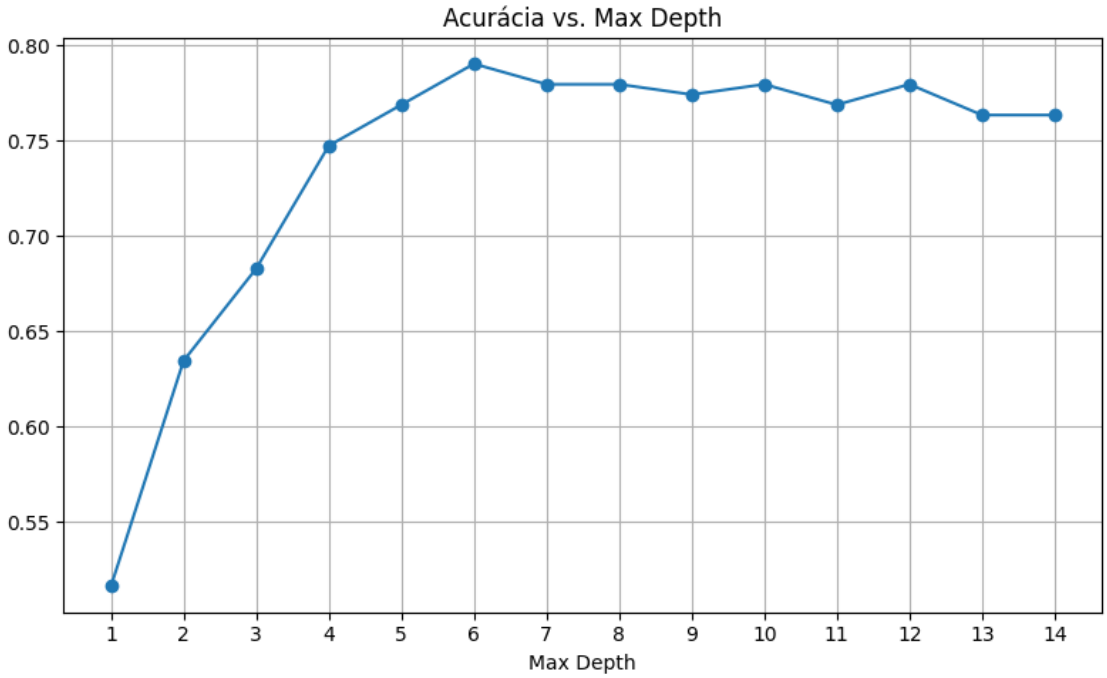
Portanto, para o algoritmo KNN temos que os melhores hiperparâmetros são o scaler `MinMax`, a métrica `cityblock` e o k = 2, com uma acurácia total de 0.854839.

1.2 Árvore de Decisão:

[19]:
```
resultadosDecision = []

for max_depth in range(1, 15):
  model = DecisionTreeClassifier(random_state=84, max_depth=max_depth)
  model.fit(trainData,trainLabels)
  Y_pred = model.predict(testData)
  accuracy = accuracy_score(testLabels, Y_pred)
  resultadosDecision.append({
      'max_depth' : max_depth,
      'accuracy' : accuracy
  })
  df_resultadosDecision = pd.DataFrame(resultadosDecision)

# Pegar o resultado com maior precisão
print(df_resultadosDecision.sort_values(by='accuracy', ascending=False).head(1))

#Plotando o gráfico max_depth vs accuracy
plt.figure(figsize=(8, 5))
plt.plot(df_resultadosDecision['max_depth'], df_resultadosDecision['accuracy'],␣
  ↪marker='o')
```

```
plt.xticks(np.arange(1, 15))
plt.xlabel('Max Depth')
plt.title('Acurácia vs. Max Depth')
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
   max_depth  accuracy
5          6  0.790323
```



Portanto, para o modelo com o algoritmo de Árvore de decisão temos que o melhor valor para o hiperparâmetro `max_depth` é 6, com uma acurácia total de 0.790323

1.3 3. Treinando os modelos com a combinação dos conjuntos de treinamento e validação

1.3.1 KNN:

[23]:
```
# Codificando labels em números
labelEncoder = LabelEncoder()
trainLabels = labelEncoder.fit_transform(trainLabels)
valLabels = labelEncoder.transform(valLabels)

resultadosKNN = []

scalers = {'StandardScaler': StandardScaler(), 'MinMaxScaler': MinMaxScaler()}
```

```
distances = ['euclidean', 'cityblock', 'chebyshev']
ks = range(1, int(math.sqrt(len(data))) + 1)

# Normalizando utilizando cada Scaler
for scaler_name, scaler in scalers.items():
    X_train = scaler.fit_transform(trainData)
    X_val = scaler.transform(valData)

    K, D = np.meshgrid(ks, distances, indexing='ij')
    for k, distance in zip(K.flatten(), D.flatten()):
        distance = str(distance)
        knn = KNeighborsClassifier(n_neighbors=k, metric=distance)
        knn.fit(X_train, trainLabels)
        Y_pred = knn.predict(X_val)
        accuracy = accuracy_score(valLabels, Y_pred)


accuracy_KNN = df_resultadosKNN.sort_values(by='accuracy',␣
 ↪ascending=False)['accuracy'].iloc[0]
classification_report_KNN = classification_report(valLabels, Y_pred,␣
 ↪target_names=labelEncoder.classes_)
confusion_matrix_KNN = confusion_matrix(valLabels, Y_pred)

# Decodificando as labels para uso posterior
trainLabels = labelEncoder.inverse_transform(trainLabels)
valLabels = labelEncoder.inverse_transform(valLabels)
```

1.3.2 Naive Bayes:

[24]:
```
model = GaussianNB()
model.fit(X_train, trainLabels)

accuracy_NB = accuracy_score(valLabels, model.predict(X_val))
classification_report_NB = classification_report(valLabels, model.
 ↪predict(X_val))
confusion_matrix_NB = confusion_matrix(valLabels, model.predict(X_val))
```

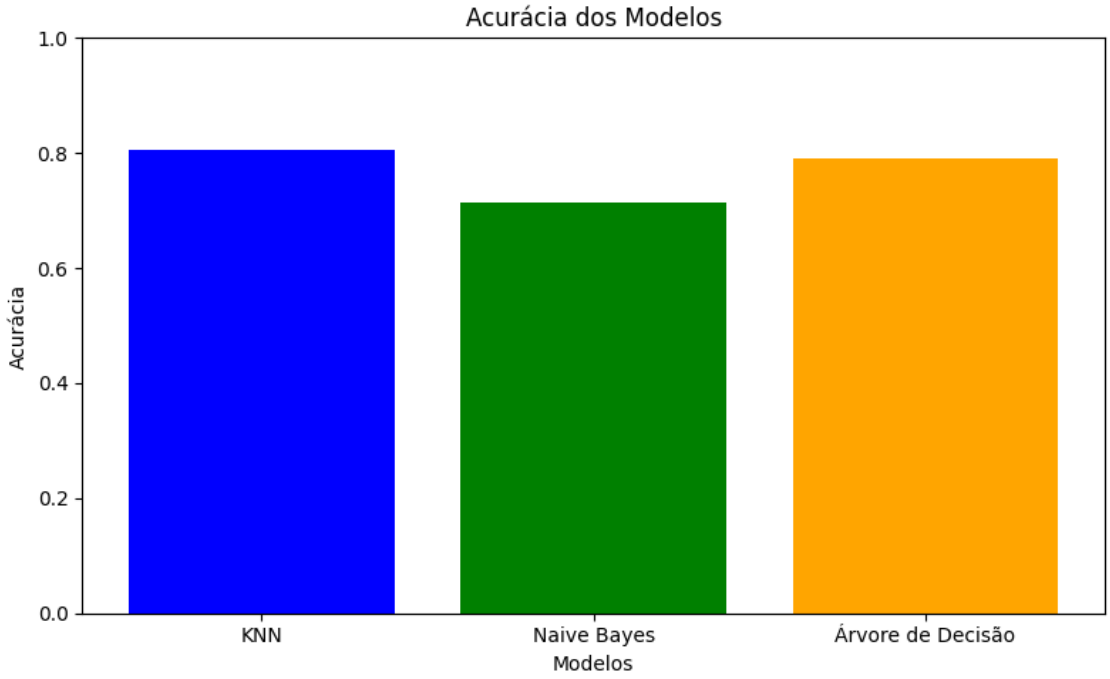
1.3.3 Árvore de decisão:

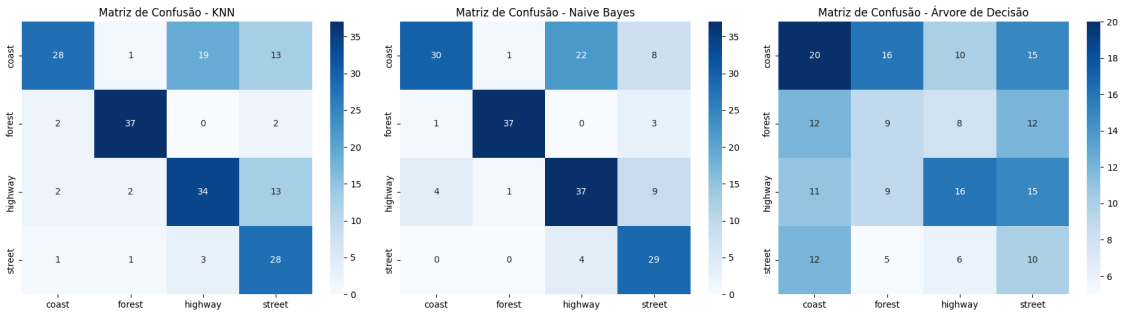
[32]:
```
resultadosDecision = []

for max_depth in range(1, 15):
  model = DecisionTreeClassifier(random_state=84, max_depth=max_depth)
  model.fit(trainData,trainLabels)
  Y_pred = model.predict(testData)
  accuracy = accuracy_score(testLabels, Y_pred)
```

```
accuracy_DT = df_resultadosDecision.sort_values(by='accuracy',␣
 ↪ascending=False)['accuracy'].iloc[0]
classification_report_DT = classification_report(valLabels, Y_pred,␣
 ↪target_names=labelEncoder.classes_)
confusion_matrix_DT = confusion_matrix(valLabels, Y_pred)
```

1.3.4 4. Comparando os 3 modelos:

[35]:
```
model_names = ['KNN', 'Naive Bayes', 'Árvore de Decisão']
accuracies = [accuracy_KNN, accuracy_NB, accuracy_DT]

# Gráfico de barras das acurácias
plt.figure(figsize=(8, 5))
plt.bar(model_names, accuracies, color=['blue', 'green', 'orange'])
plt.title('Acurácia dos Modelos')
plt.xlabel('Modelos')
plt.ylabel('Acurácia')
plt.ylim(0, 1)
plt.tight_layout()
plt.show()

# Matrizes de confusão para os três modelos
conf_matrices = [confusion_matrix_KNN, confusion_matrix_NB, confusion_matrix_DT]
titles = ['Matriz de Confusão - KNN', 'Matriz de Confusão - Naive Bayes',␣
 ↪'Matriz de Confusão - Árvore de Decisão']

# Plots lado a lado
fig, axes = plt.subplots(1, 3, figsize=(18, 5))

for i, ax in enumerate(axes):
    sns.heatmap(conf_matrices[i], annot=True, fmt='d', cmap='Blues',
                xticklabels=labelEncoder.classes_, yticklabels=labelEncoder.
 ↪classes_, ax=ax)
    ax.set_title(titles[i])

plt.tight_layout()
plt.show()
```





Portanto, vemos que, embora que o modelo que utiliza a Árvore de Decisão possua uma acurácia maior que o modelo de Naive Bayes, sua matriz de confusão demonstra uma incorretude maior na classificação das imagens.

[ ]: