Exercício KNN Ponderado

Introdução ao Reconhecimento de Padrões

Turma TB

Gabriel Vaz Cançado Ferreira

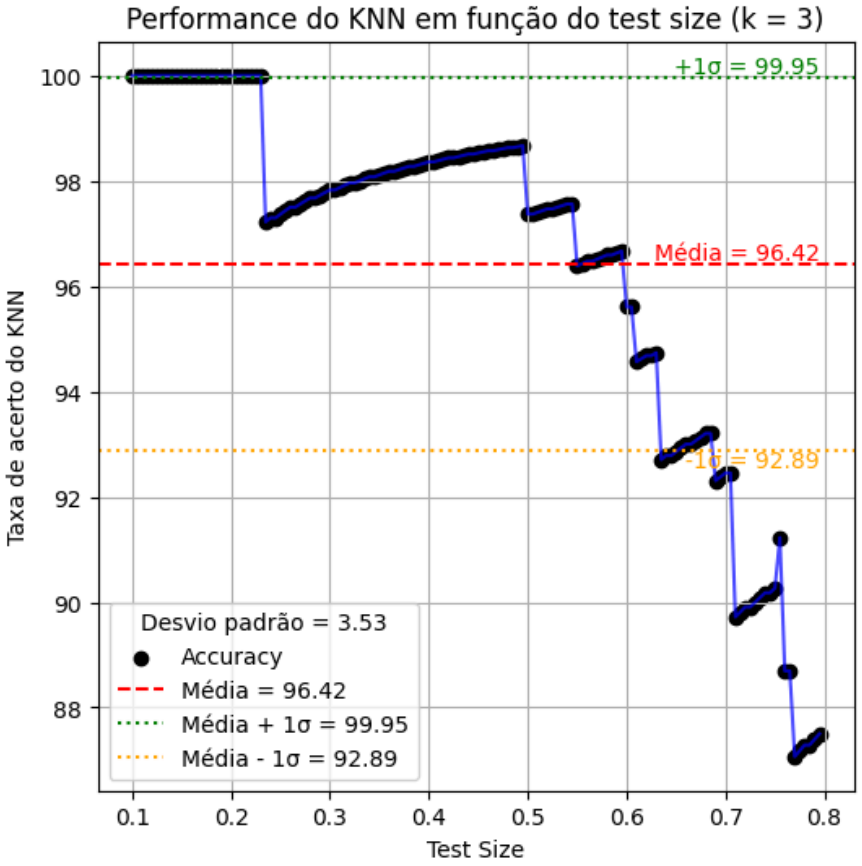
Matrícula: 2021015194

Intermediário 1

- **Objetivos**: Avaliar o desempenho do KNN em função do parâmetro k. Apresentar gráfico de desempenho em função de k.

  Para a realização da atividade usamos o conjunto de dados Iris disponível em `from sklearn.datasets import load_iris`.

1. Inicialmente, escolhemos um valor de k = 3. Em seguida realizamos uma série de experimentos de modo a determinar a performance do KNN quando variamos o tamanho dos conjuntos de dados de treino e teste.

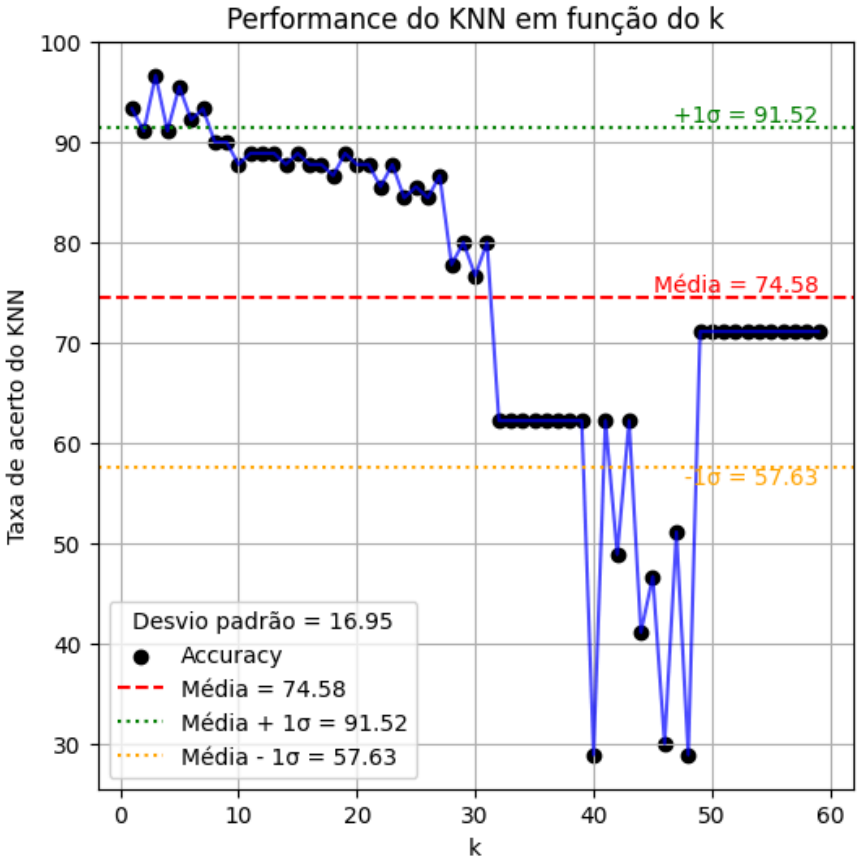


Como é de se esperar, conforme aumentamos o conjunto de testes, temos uma queda na performance do modelo. Ao mesmo tempo, valores pequenos para o conjunto de teste causam um overfitting no modelo.

Isso ocorre pois, no KNN, cada ponto do conjunto de treino tem o mesmo peso na votação da classe. Assim, quando usamos poucos dados de treino, o modelo memoriza excessivamente os vizinhos disponíveis e não generaliza bem para novos pontos (overfitting). Por outro lado, quando usamos muitos dados de treino e aumentamos o conjunto de teste, a quantidade de informação usada na classificação por vizinhança relativa diminui, o que pode reduzir a acurácia observada no teste. Além disso, valores de k muito pequenos tornam o

modelo sensível a ruídos (alta variância), enquanto valores de k muito grandes suavizam demais a fronteira de decisão (alto viés).

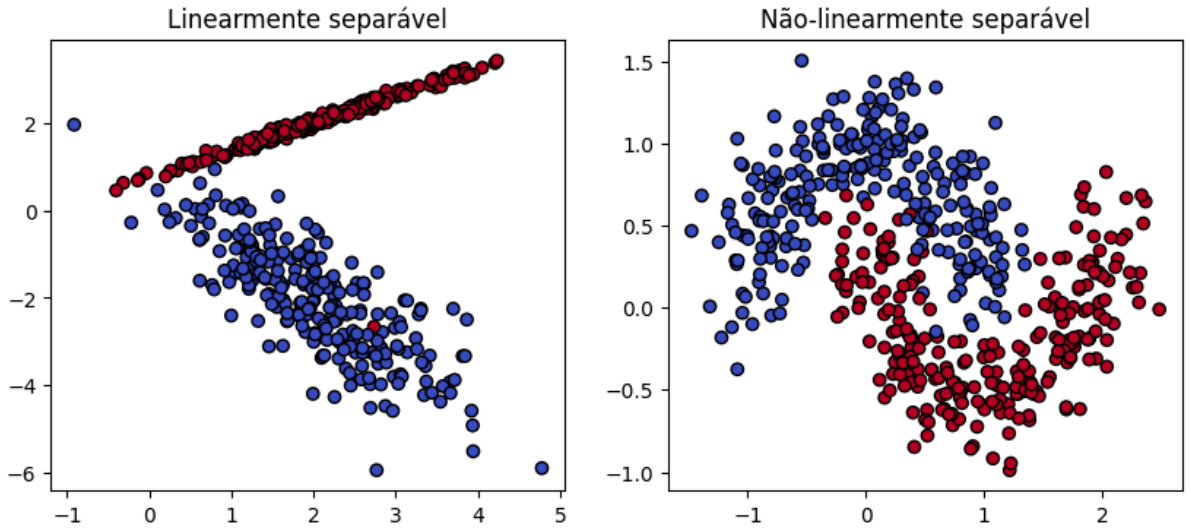
2. Em seguida, mantemos o tamanho dos conjuntos de treino e teste constantes e variamos o valor de k para entender seu comportamento. Escolhemos um valor de test size = 0.6, uma vez que é o valor que proporcionou a precisão média no item 1. Para valores pequenos de k (entre 1 e 10), o modelo apresenta alta acurácia (acima de 90%) mas é mais sensível a ruídos e pontos isolados, caracterizando overfitting, pois se adapta fortemente a casos específicos e pode errar em regiões mais complexas. Para valores intermediários de k (entre 15 e 30), a acurácia cai de forma gradual, estabilizando em torno de 80%, alcançando um equilíbrio entre viés e variância, onde o modelo não é tão sensível a ruídos, mas começa a perder detalhes da fronteira de decisão. Já para valores grandes de k (acima de 35), a performance despenca para cerca de 30%–60%, pois o modelo considera muitos vizinhos e a fronteira de decisão fica excessivamente suavizada, ignorando estruturas locais e tendendo a prever a classe majoritária. Estatisticamente, a média da acurácia é de ~74,6% com desvio padrão de ~16,9%, indicando alta variação conforme k; os melhores resultados ocorrem com k pequeno, enquanto os piores aparecem para k muito grande. Conceitualmente, k pequeno leva a baixa generalização e alto overfitting, k grande gera excesso de suavização e alto viés (underfitting), sendo ideal escolher um k intermediário que balanceie esses efeitos.



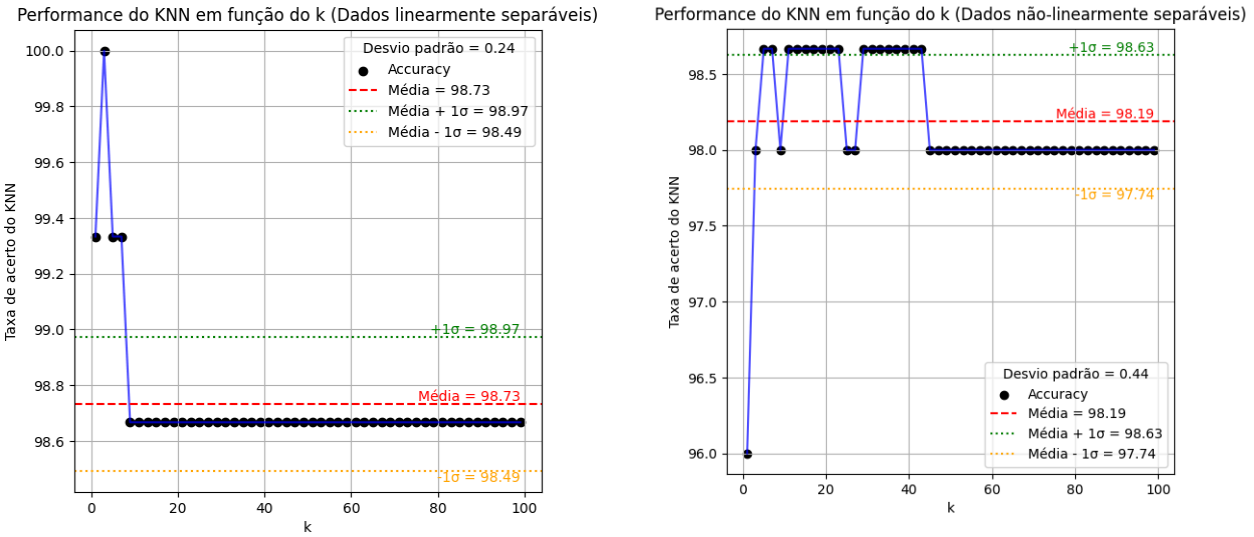
Intermediário 2

- **Objetivos:** Analisar o comportamento do KNN ponderado como uma função de mapeamento em um espaço de verossimilhanças.

  Para gerar dois conjuntos de dados linearmente e não linearmente separáveis usamos `from sklearn.datasets import make_classification, make_moons`, respectivamente.



Em seguida, variamos o valor de k e usamos o algoritmo KNN ponderado.



Notamos que ambos os casos demonstram comportamentos semelhantes. No KNN ponderado, conforme aumenta-se o valor de k, os pontos mais distantes têm peso cada vez menor no mapeamento para o espaço Q1xQ2, assim, o modelo tende a convergir para uma performance constante. Como diferença clara, temos que o valor de k que proporciona convergência da performance é o dobro para os dados não-linearmente separáveis.

- **Como determinar o melhor valor de k?**
  Para problemas em que os conjuntos de dados não são tão grandes e o tempo de solução do modelo não é tão grande, podemos seguir uma abordagem de otimização conhecida como métodos de busca. Ou seja, como k assume apenas valores inteiros, realizamos uma busca entre os extremos de k, avaliando a função e reduzindo o intervalo de busca a cada interação. Apesar deste ser um método de otimização conhecido pela 'força bruta', como não há tantos valores de k a serem avaliados, principalmente pois podemos ignorar os valores em que k converge, este é um método simples, fácil de implementar e com alta precisão.

Anexos

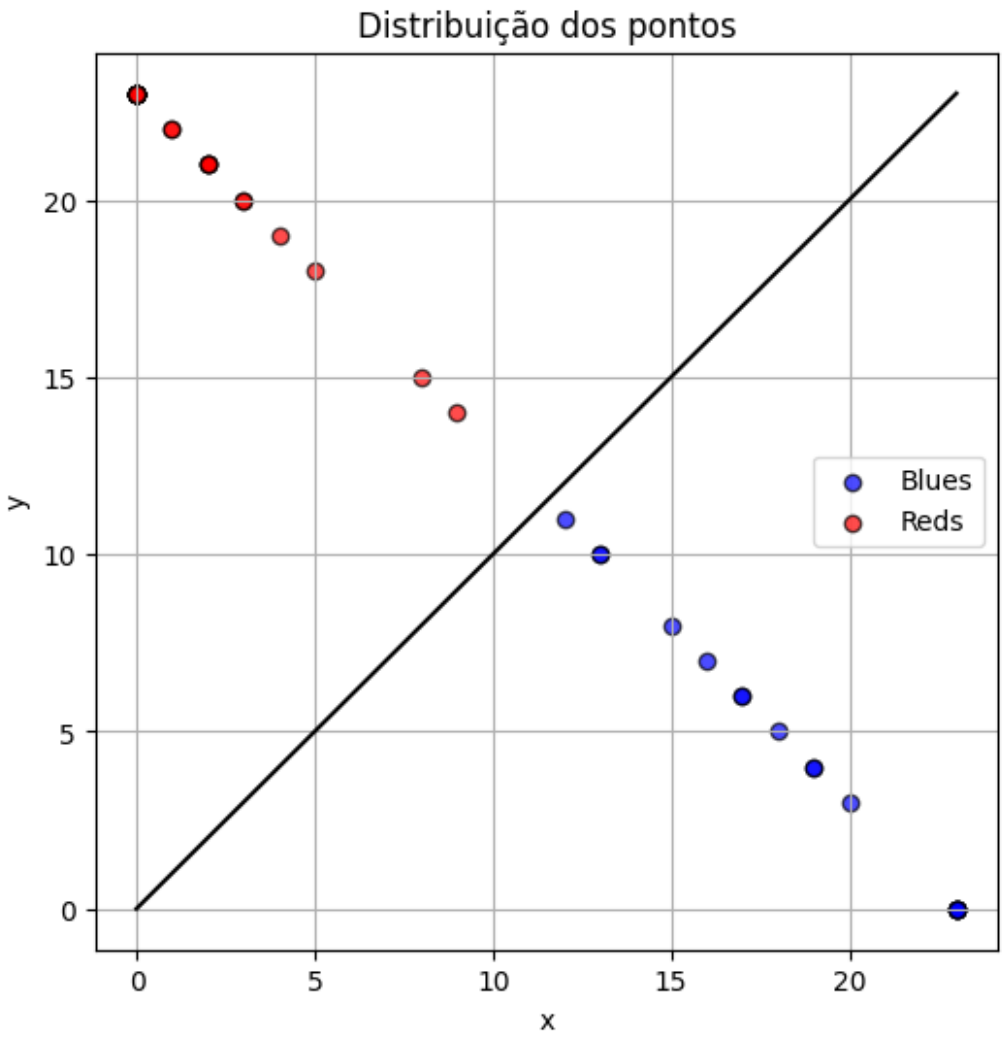
Código desenvolvido e utilizado:

1.3 Exercício Intermediário 1

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm

iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['class'] = iris.target

df['class'] = df['class'].apply(lambda x: -1 if x in [0,2] else 1)

df = df.sample(frac=1, random_state=42).reset_index(drop=True)

n = len(df)
k = 3

X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.4, shuffle=True, random_state=42
)

def euclidean_distance(x, y):
    if len(x) != len(y):
        raise ValueError("x and y must have the same dimension")
    return np.sqrt(sum((xi - yi) ** 2 for xi, yi in zip(x, y)))

def knn(X_train, y_train, X_test, k):

  y_pred = []
  Q = []
  for i in range(len(X_test)):
    distances = []
    for j in range(len(X_train)):
      distances.append((euclidean_distance(X_test[i], X_train[j]),
y_train[j]))
    distances.sort()
    neighbors = distances[:k]
    labels = [neighbor[1] for neighbor in neighbors]
    Q1 = 0
    Q2 = 0

    for label in labels:
      if label == 1:
```

```
        Q1 += 1
      elif label == -1:
        Q2 += 1

    Q.append((Q1,Q2))

    y_pred.append(int(max(set(labels), key=labels.count)))
  # print(Q)
  return Q , y_pred

def get_labels(Q, y_pred):
    blues = []
    reds = []
    for q,pred in zip(Q,y_pred):

        if pred == 1:
            blues.append(q)
        else:
            reds.append(q)

    return blues,reds

def plot_predictions(blues,reds):

    xb,yb = zip(*blues)
    xr,yr = zip(*reds)
    plt.figure(figsize=(8,8))
    plt.scatter(xb,yb, color = 'blue', marker='o')
    plt.scatter(xr,yr, color = 'red', marker='o')
    line = np.linspace(6, 18, 100)
    plt.plot(line, line, color="black", linestyle="-", label="y = x")
    plt.title("Previsões no plano Q1xQ2")
    plt.xlabel('Q2')
    plt.ylabel('Q1')
    plt.grid(True)
    plt.gca().set_aspect('equal', adjustable='box')
    plt.show()

    return None

def accuracy_percentage(y_pred, y_test):
    if len(y_pred) != len(y_test):
        raise ValueError("As listas devem ter o mesmo tamanho.")

    acertos = sum(yp == yt for yp, yt in zip(y_pred, y_test))
    total = len(y_test)
    return (acertos / total) * 100

iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['class'] = iris.target
```

```
df['class'] = df['class'].apply(lambda x: -1 if x in [0,2] else 1)

df = df.sample(frac=1, random_state=42).reset_index(drop=True)

n = len(df)
k = 23

X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, shuffle=True, random_state=42
)
Q, y_pred = knn(X_train, y_train, X_test, k)
blues, reds = get_labels(Q, y_pred)
def plot_blues_and_reds(blues, reds):

    xb, yb = zip(*blues) if blues else ([], [])
    xr, yr = zip(*reds) if reds else ([], [])

    plt.figure(figsize=(6,6))
    plt.scatter(xb, yb, color="blue", label="Blues", alpha=0.7,
edgecolor="k")
    plt.scatter(xr, yr, color="red", label="Reds", alpha=0.7,
edgecolor="k")
    plt.plot(range(0,k+1),range(0,k+1), color = 'black')
    plt.title("Distribuição dos pontos")
    plt.xlabel("x")
    plt.ylabel("y")
    plt.legend()
    plt.grid(True)
    plt.show()

plot_blues_and_reds(blues,reds)
```



Escolha um conjunto de dados real, determine um valor inicial para k, separe os dados em dois conjuntos, sendo um de referência para o KNN (treinamento) e outro de validação e avalie o percentual de acerto do KNN no conjunto de validação para este valor de k e partição atual do conjunto de dados. Repita esta operação pelo menos 10 e armazene a média e o desvio padrão do desempenho para este valor de k.

Escolhi k = 3!

```
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['class'] = iris.target

df['class'] = df['class'].apply(lambda x: -1 if x in [0,2] else 1)

df = df.sample(frac=1, random_state=42).reset_index(drop=True)

n = len(df)
```

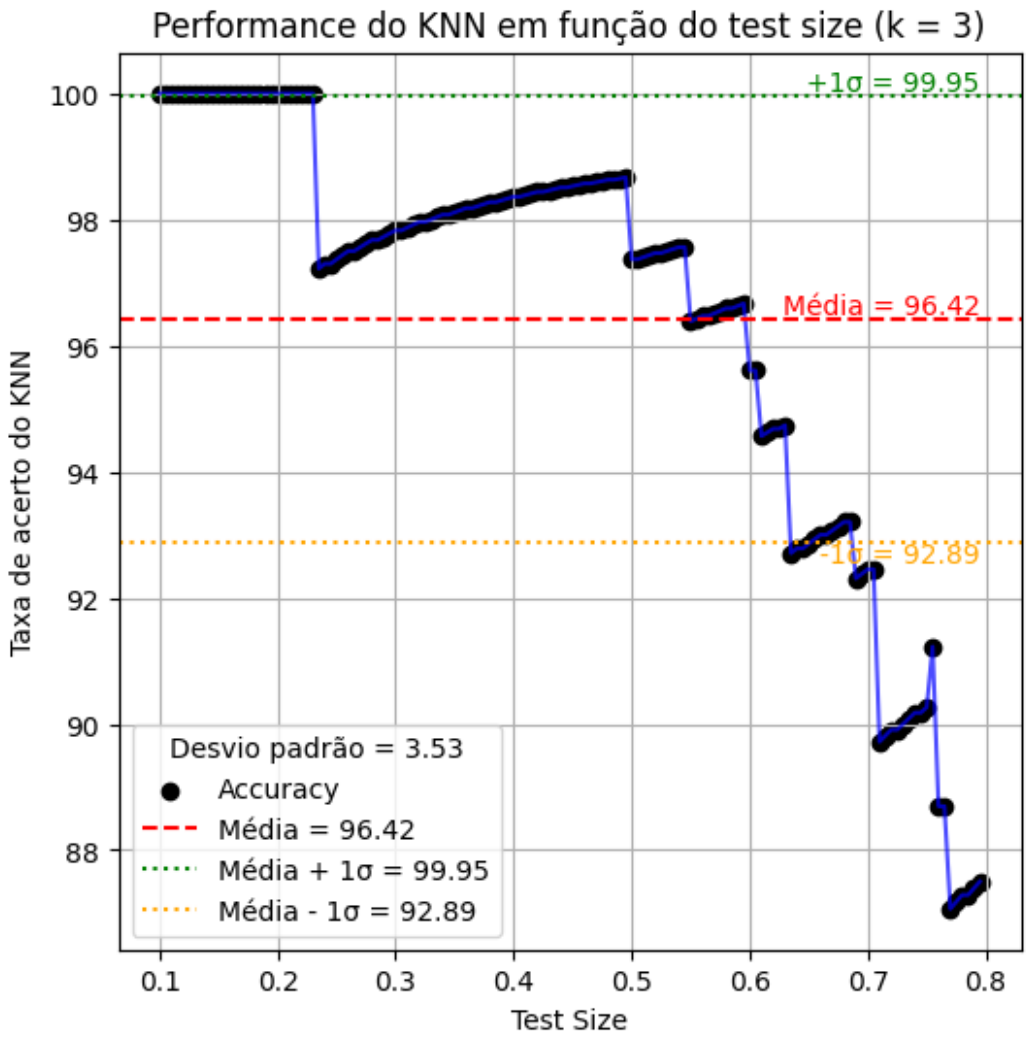
```
k = 3
accuracy = []


for test_size in np.arange(0.1, 0.8, 0.005):
    X = df.iloc[:, :-1].values
    y = df.iloc[:, -1].values

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=test_size, shuffle=True, random_state=42
    )

    Q, y_pred = knn(X_train, y_train, X_test, k)
    blues, reds = get_labels(Q, y_pred)
    acc = accuracy_percentage(y_pred, y_test)
    accuracy.append(acc)

    print(f'Test size = {round(test_size,2)} → Accuracy =
{round(acc,2)}')

xgrid = np.arange(0.1, 0.8, 0.005)
mean = np.mean(accuracy)
std = np.std(accuracy)

plt.figure(figsize=(6,6))
plt.scatter(xgrid, accuracy, color='black', marker='o',
label="Accuracy")
plt.plot(xgrid, accuracy, color="blue", linestyle="-", alpha=0.7)


plt.axhline(mean, color='red', linestyle='--', label=f"Média =
{mean:.2f}")
plt.axhline(mean + std, color='green', linestyle=':', label=f"Média +
1σ = {mean + std:.2f}")
plt.axhline(mean - std, color='orange', linestyle=':', label=f"Média -
1σ = {mean - std:.2f}")

plt.text(xgrid[-1], mean + std, f"+1σ = {mean + std:.2f}",
color='green', va='bottom', ha='right')
plt.text(xgrid[-1], mean - std, f"-1σ = {mean - std:.2f}",
color='orange', va='top', ha='right')
plt.text(xgrid[-1], mean, f"Média = {mean:.2f}", color='red',
va='bottom', ha='right')


plt.title(f'Performance do KNN em função do test size (k = {k})')
plt.xlabel('Test Size')
plt.ylabel("Taxa de acerto do KNN")
plt.legend(title=f"Desvio padrão = {std:.2f}", loc='lower left')
```

```
plt.grid(True)
plt.show()
```

```
Test size = 0.1 → Accuracy = 100.0
Test size = 0.11 → Accuracy = 100.0
Test size = 0.11 → Accuracy = 100.0
Test size = 0.12 → Accuracy = 100.0
Test size = 0.12 → Accuracy = 100.0
Test size = 0.13 → Accuracy = 100.0
Test size = 0.13 → Accuracy = 100.0
Test size = 0.14 → Accuracy = 100.0
Test size = 0.14 → Accuracy = 100.0
Test size = 0.15 → Accuracy = 100.0
Test size = 0.15 → Accuracy = 100.0
Test size = 0.16 → Accuracy = 100.0
Test size = 0.16 → Accuracy = 100.0
Test size = 0.17 → Accuracy = 100.0
Test size = 0.17 → Accuracy = 100.0
Test size = 0.18 → Accuracy = 100.0
Test size = 0.18 → Accuracy = 100.0
Test size = 0.19 → Accuracy = 100.0
Test size = 0.19 → Accuracy = 100.0
Test size = 0.2 → Accuracy = 100.0
Test size = 0.2 → Accuracy = 100.0
Test size = 0.21 → Accuracy = 100.0
Test size = 0.21 → Accuracy = 100.0
Test size = 0.22 → Accuracy = 100.0
Test size = 0.22 → Accuracy = 100.0
Test size = 0.23 → Accuracy = 100.0
Test size = 0.23 → Accuracy = 100.0
Test size = 0.24 → Accuracy = 97.22
Test size = 0.24 → Accuracy = 97.3
Test size = 0.25 → Accuracy = 97.3
Test size = 0.25 → Accuracy = 97.37
Test size = 0.26 → Accuracy = 97.44
Test size = 0.26 → Accuracy = 97.5
Test size = 0.27 → Accuracy = 97.5
Test size = 0.27 → Accuracy = 97.56
Test size = 0.28 → Accuracy = 97.62
Test size = 0.28 → Accuracy = 97.67
Test size = 0.29 → Accuracy = 97.67
Test size = 0.29 → Accuracy = 97.73
Test size = 0.3 → Accuracy = 97.78
Test size = 0.3 → Accuracy = 97.83
Test size = 0.31 → Accuracy = 97.83
Test size = 0.31 → Accuracy = 97.87
Test size = 0.32 → Accuracy = 97.92
Test size = 0.32 → Accuracy = 97.96
Test size = 0.33 → Accuracy = 97.96
Test size = 0.33 → Accuracy = 98.0
```

```
Test size = 0.34 → Accuracy = 98.04
Test size = 0.34 → Accuracy = 98.08
Test size = 0.35 → Accuracy = 98.08
Test size = 0.35 → Accuracy = 98.11
Test size = 0.36 → Accuracy = 98.15
Test size = 0.36 → Accuracy = 98.18
Test size = 0.37 → Accuracy = 98.18
Test size = 0.37 → Accuracy = 98.21
Test size = 0.38 → Accuracy = 98.25
Test size = 0.38 → Accuracy = 98.28
Test size = 0.39 → Accuracy = 98.28
Test size = 0.39 → Accuracy = 98.31
Test size = 0.4 → Accuracy = 98.33
Test size = 0.4 → Accuracy = 98.36
Test size = 0.41 → Accuracy = 98.36
Test size = 0.41 → Accuracy = 98.39
Test size = 0.42 → Accuracy = 98.41
Test size = 0.42 → Accuracy = 98.44
Test size = 0.43 → Accuracy = 98.44
Test size = 0.43 → Accuracy = 98.46
Test size = 0.44 → Accuracy = 98.48
Test size = 0.44 → Accuracy = 98.51
Test size = 0.45 → Accuracy = 98.51
Test size = 0.45 → Accuracy = 98.53
Test size = 0.46 → Accuracy = 98.55
Test size = 0.46 → Accuracy = 98.57
Test size = 0.47 → Accuracy = 98.57
Test size = 0.47 → Accuracy = 98.59
Test size = 0.48 → Accuracy = 98.61
Test size = 0.48 → Accuracy = 98.63
Test size = 0.49 → Accuracy = 98.63
Test size = 0.49 → Accuracy = 98.65
Test size = 0.5 → Accuracy = 98.67
Test size = 0.5 → Accuracy = 97.37
Test size = 0.51 → Accuracy = 97.37
Test size = 0.51 → Accuracy = 97.4
Test size = 0.52 → Accuracy = 97.44
Test size = 0.52 → Accuracy = 97.47
Test size = 0.53 → Accuracy = 97.47
Test size = 0.53 → Accuracy = 97.5
Test size = 0.54 → Accuracy = 97.53
Test size = 0.54 → Accuracy = 97.56
Test size = 0.55 → Accuracy = 97.56
Test size = 0.55 → Accuracy = 96.39
Test size = 0.56 → Accuracy = 96.43
Test size = 0.56 → Accuracy = 96.47
Test size = 0.57 → Accuracy = 96.47
Test size = 0.57 → Accuracy = 96.51
Test size = 0.58 → Accuracy = 96.55
Test size = 0.58 → Accuracy = 96.59
```

```
Test size = 0.59 → Accuracy = 96.59
Test size = 0.59 → Accuracy = 96.63
Test size = 0.6 → Accuracy = 96.67
Test size = 0.6 → Accuracy = 95.6
Test size = 0.61 → Accuracy = 95.6
Test size = 0.61 → Accuracy = 94.57
Test size = 0.62 → Accuracy = 94.62
Test size = 0.62 → Accuracy = 94.68
Test size = 0.63 → Accuracy = 94.68
Test size = 0.63 → Accuracy = 94.74
Test size = 0.64 → Accuracy = 92.71
Test size = 0.64 → Accuracy = 92.78
Test size = 0.65 → Accuracy = 92.78
Test size = 0.65 → Accuracy = 92.86
Test size = 0.66 → Accuracy = 92.93
Test size = 0.66 → Accuracy = 93.0
Test size = 0.67 → Accuracy = 93.0
Test size = 0.67 → Accuracy = 93.07
Test size = 0.68 → Accuracy = 93.14
Test size = 0.68 → Accuracy = 93.2
Test size = 0.69 → Accuracy = 93.2
Test size = 0.69 → Accuracy = 92.31
Test size = 0.7 → Accuracy = 92.38
Test size = 0.7 → Accuracy = 92.45
Test size = 0.71 → Accuracy = 92.45
Test size = 0.71 → Accuracy = 89.72
Test size = 0.72 → Accuracy = 89.81
Test size = 0.72 → Accuracy = 89.91
Test size = 0.73 → Accuracy = 89.91
Test size = 0.73 → Accuracy = 90.0
Test size = 0.74 → Accuracy = 90.09
Test size = 0.74 → Accuracy = 90.18
Test size = 0.75 → Accuracy = 90.18
Test size = 0.75 → Accuracy = 90.27
Test size = 0.76 → Accuracy = 91.23
Test size = 0.76 → Accuracy = 88.7
Test size = 0.77 → Accuracy = 88.7
Test size = 0.77 → Accuracy = 87.07
Test size = 0.78 → Accuracy = 87.18
Test size = 0.78 → Accuracy = 87.29
Test size = 0.79 → Accuracy = 87.29
Test size = 0.79 → Accuracy = 87.39
Test size = 0.8 → Accuracy = 87.5
```



Varie o valor de k de 1 a um valor mais alto que permita a observação da mudança de comportamento do desempenho do modelo, possivelmente passando por um máximo.

Vamos selecionar um test size de 0.6 pois é o valor que está na média para k = 3.

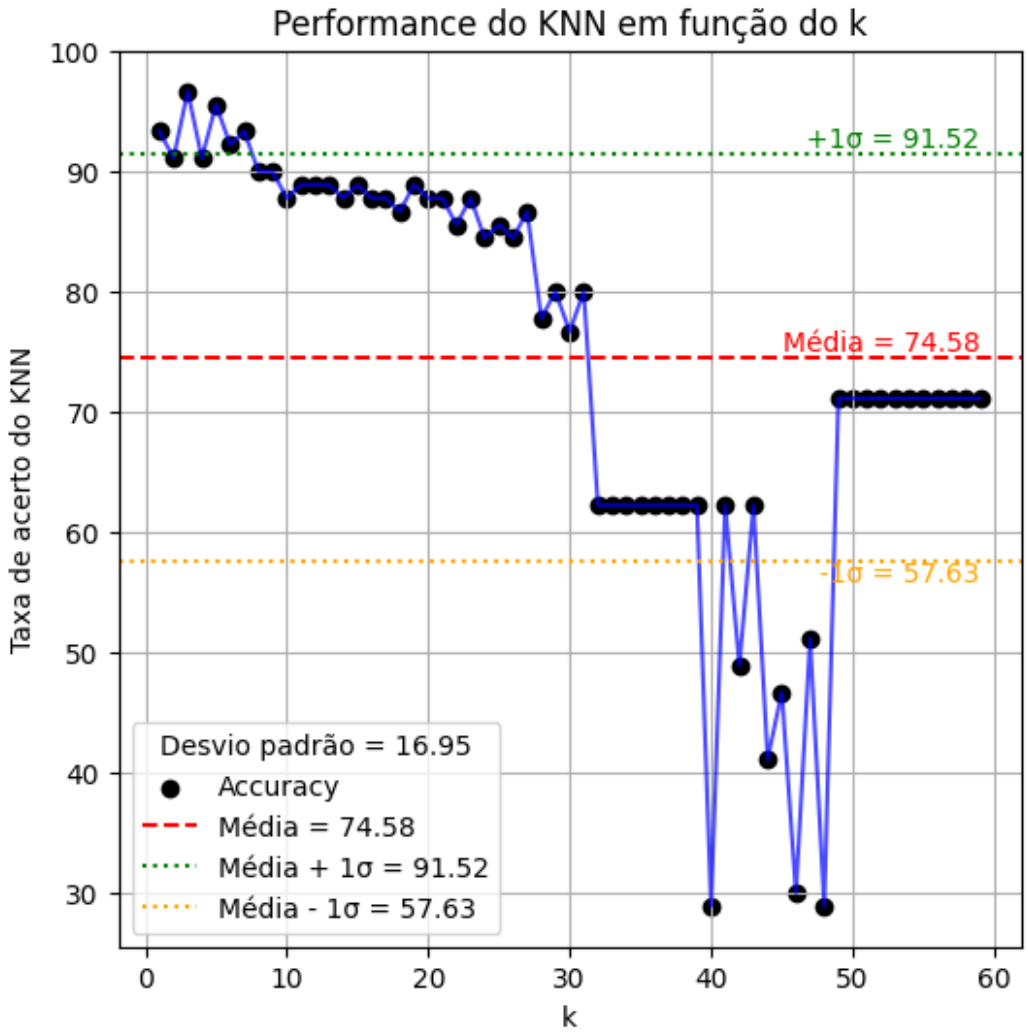
```
n = len(df)
accuracy = []

test_size = 0.6

print(n*test_size)

xgrid = range(1,60,1)

for k in xgrid:
    X = df.iloc[:, :-1].values
    y = df.iloc[:, -1].values
```

```
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=test_size, shuffle=True, random_state=42
    )

    Q, y_pred = knn(X_train, y_train, X_test, k)
    blues, reds = get_labels(Q, y_pred)
    acc = accuracy_percentage(y_pred, y_test)
    accuracy.append(acc)

    print(f'k = {k} → Accuracy = {round(acc,2)}')

mean = np.mean(accuracy)
std = np.std(accuracy)

plt.figure(figsize=(6,6))
plt.scatter(xgrid, accuracy, color='black', marker='o',
label="Accuracy")
plt.plot(xgrid, accuracy, color="blue", linestyle="-", alpha=0.7)


plt.axhline(mean, color='red', linestyle='--', label=f"Média =
{mean:.2f}")
plt.axhline(mean + std, color='green', linestyle=':', label=f"Média +
1σ = {mean + std:.2f}")
plt.axhline(mean - std, color='orange', linestyle=':', label=f"Média -
1σ = {mean - std:.2f}")

plt.text(xgrid[-1], mean + std, f"+1σ = {mean + std:.2f}",
color='green', va='bottom', ha='right')
plt.text(xgrid[-1], mean - std, f"-1σ = {mean - std:.2f}",
color='orange', va='top', ha='right')
plt.text(xgrid[-1], mean, f"Média = {mean:.2f}", color='red',
va='bottom', ha='right')


plt.title('Performance do KNN em função do k')
plt.xlabel('k')
plt.ylabel("Taxa de acerto do KNN")
plt.legend(title=f"Desvio padrão = {std:.2f}", loc='lower left')
plt.grid(True)
plt.show()
```

```
90.0
k = 1 → Accuracy = 93.33
k = 2 → Accuracy = 91.11
k = 3 → Accuracy = 96.67
k = 4 → Accuracy = 91.11
k = 5 → Accuracy = 95.56
k = 6 → Accuracy = 92.22
```

```
k = 7 → Accuracy = 93.33
k = 8 → Accuracy = 90.0
k = 9 → Accuracy = 90.0
k = 10 → Accuracy = 87.78
k = 11 → Accuracy = 88.89
k = 12 → Accuracy = 88.89
k = 13 → Accuracy = 88.89
k = 14 → Accuracy = 87.78
k = 15 → Accuracy = 88.89
k = 16 → Accuracy = 87.78
k = 17 → Accuracy = 87.78
k = 18 → Accuracy = 86.67
k = 19 → Accuracy = 88.89
k = 20 → Accuracy = 87.78
k = 21 → Accuracy = 87.78
k = 22 → Accuracy = 85.56
k = 23 → Accuracy = 87.78
k = 24 → Accuracy = 84.44
k = 25 → Accuracy = 85.56
k = 26 → Accuracy = 84.44
k = 27 → Accuracy = 86.67
k = 28 → Accuracy = 77.78
k = 29 → Accuracy = 80.0
k = 30 → Accuracy = 76.67
k = 31 → Accuracy = 80.0
k = 32 → Accuracy = 62.22
k = 33 → Accuracy = 62.22
k = 34 → Accuracy = 62.22
k = 35 → Accuracy = 62.22
k = 36 → Accuracy = 62.22
k = 37 → Accuracy = 62.22
k = 38 → Accuracy = 62.22
k = 39 → Accuracy = 62.22
k = 40 → Accuracy = 28.89
k = 41 → Accuracy = 62.22
k = 42 → Accuracy = 48.89
k = 43 → Accuracy = 62.22
k = 44 → Accuracy = 41.11
k = 45 → Accuracy = 46.67
k = 46 → Accuracy = 30.0
k = 47 → Accuracy = 51.11
k = 48 → Accuracy = 28.89
k = 49 → Accuracy = 71.11
k = 50 → Accuracy = 71.11
k = 51 → Accuracy = 71.11
k = 52 → Accuracy = 71.11
k = 53 → Accuracy = 71.11
k = 54 → Accuracy = 71.11
k = 55 → Accuracy = 71.11
```

```
k = 56 → Accuracy = 71.11
k = 57 → Accuracy = 71.11
k = 58 → Accuracy = 71.11
k = 59 → Accuracy = 71.11
```

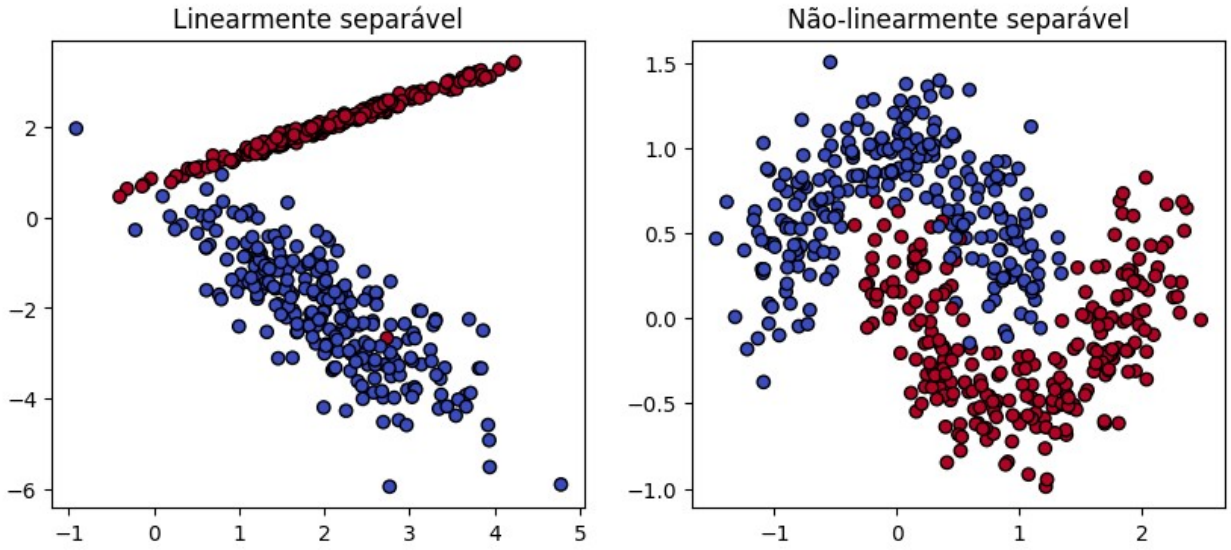


1.4 Exercício Intermediário 2

```python
def gaussian_kernel(x, xi, sigma):
    dist2 = np.linalg.norm(x - xi) ** 2
    return np.exp(-dist2 / (2 * sigma**2))

def knn_ponderado(X_train, y_train, X_test, k, sigma=0.3):
    y_pred = []
    Q = []

    for i in range(len(X_test)):
```

```
        distances = []
        for j in range(len(X_train)):
            dist = euclidean_distance(X_test[i], X_train[j])
            distances.append((dist, y_train[j], X_train[j]))

        distances.sort(key=lambda x: x[0])
        neighbors = distances[:k]

        Q1 = 0.0
        Q2 = 0.0
        for _, label, xj in neighbors:
            weight = gaussian_kernel(X_test[i], xj, sigma)
            # print(weight)
            if label == 1:
                Q1 += weight
            elif label == 0:
                Q2 += weight

        Q.append((Q1, Q2))


        decision_value = Q1 - Q2

        y_hat = 1 if decision_value > 0 else 0
        y_pred.append(y_hat)

    return Q, y_pred

from sklearn.datasets import make_classification, make_moons

X_linear, y_linear = make_classification(
    n_samples=500,
    n_features=2,
    n_redundant=0,
    n_informative=2,
    n_clusters_per_class=1,
    class_sep=2.0,
    random_state=42
)

X_nonlinear, y_nonlinear = make_moons(
    n_samples=500,
    noise=0.2,
    random_state=42
)

X_lin_train, X_lin_test, y_lin_train, y_lin_test = train_test_split(
    X_linear, y_linear, test_size=0.3, random_state=42
)
```

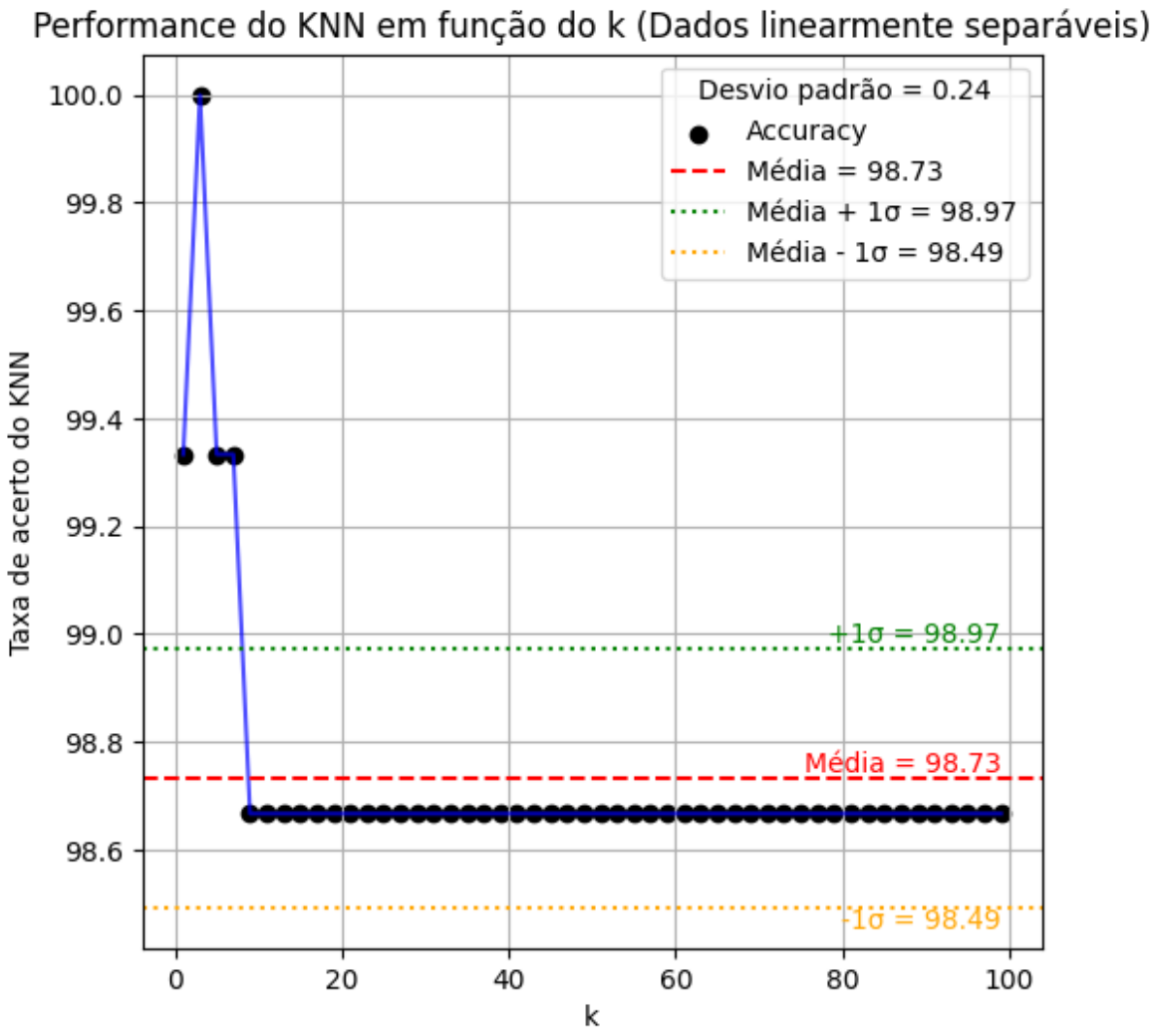
```
X_non_train, X_non_test, y_non_train, y_non_test = train_test_split(
    X_nonlinear, y_nonlinear, test_size=0.3, random_state=42
)

fig, axs = plt.subplots(1, 2, figsize=(10, 4))

axs[0].scatter(X_linear[:, 0], X_linear[:, 1], c=y_linear,
cmap=plt.cm.coolwarm, edgecolor='k')
axs[0].set_title("Linearmente separável")

axs[1].scatter(X_nonlinear[:, 0], X_nonlinear[:, 1], c=y_nonlinear,
cmap=plt.cm.coolwarm, edgecolor='k')
axs[1].set_title("Não-linearmente separável")

plt.show()
```

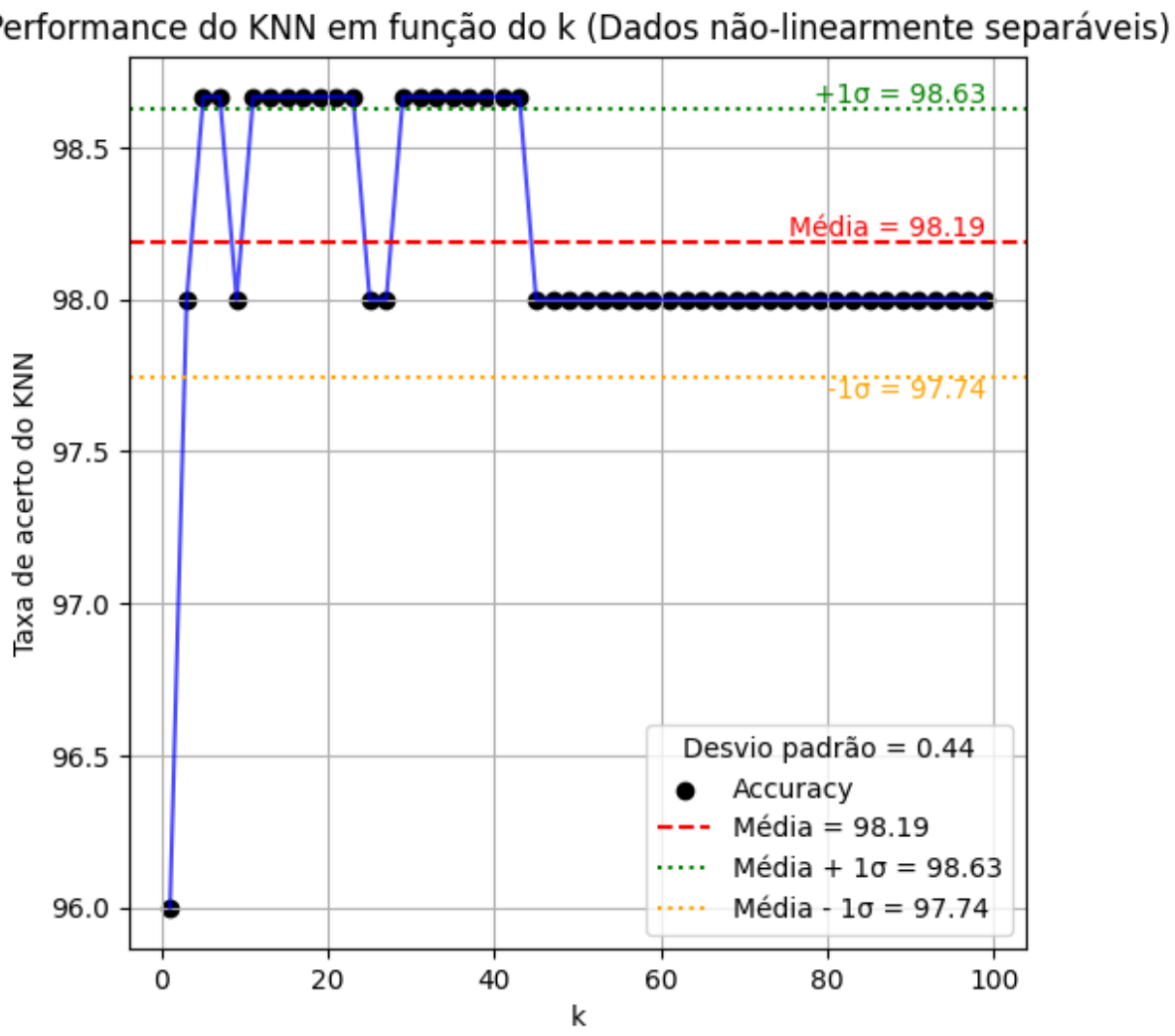



```
def plot_knn_ponderado(X_train, y_train, X_test,y_test, k, linear =
True):
    xgrid = range(1,100,2)

    accuracy = []
    for k in xgrid:

        Q, y_pred = knn_ponderado(X_train, y_train, X_test, k)
        blues, reds = get_labels(Q, y_pred)
        acc = accuracy_percentage(y_pred, y_test)
        accuracy.append(acc)

        # print(f'k = {k} → Accuracy = {round(acc,2)}')

    mean = np.mean(accuracy)
```

```
    std = np.std(accuracy)

    plt.figure(figsize=(6,6))
    plt.scatter(xgrid, accuracy, color='black', marker='o',
label="Accuracy")
    plt.plot(xgrid, accuracy, color="blue", linestyle="-", alpha=0.7)


    plt.axhline(mean, color='red', linestyle='--', label=f"Média =
{mean:.2f}")
    plt.axhline(mean + std, color='green', linestyle=':',
label=f"Média + 1σ = {mean + std:.2f}")
    plt.axhline(mean - std, color='orange', linestyle=':',
label=f"Média - 1σ = {mean - std:.2f}")

    plt.text(xgrid[-1], mean + std, f"+1σ = {mean + std:.2f}",
color='green', va='bottom', ha='right')
    plt.text(xgrid[-1], mean - std, f"-1σ = {mean - std:.2f}",
color='orange', va='top', ha='right')
    plt.text(xgrid[-1], mean, f"Média = {mean:.2f}", color='red',
va='bottom', ha='right')

    if linear == True:
        plt.title('Performance do KNN em função do k (Dados
linearmente separáveis)')
    else:
        plt.title('Performance do KNN em função do k (Dados não-
linearmente separáveis)')
    plt.xlabel('k')
    plt.ylabel("Taxa de acerto do KNN")
    plt.legend(title=f"Desvio padrão = {std:.2f}")
    plt.grid(True)

plot_knn_ponderado(X_lin_train,y_lin_train, X_lin_test,y_lin_test, k,
linear=True)
plot_knn_ponderado(X_non_train,y_non_train, X_non_test,y_non_test, k,
linear=False)
```

```
50
50
50
50
```


Performance do KNN em função do k (Dados linearmente separáveis)
Desvio padrão = 0.24
Accuracy
Média = 98.73
Média + 1σ = 98.97
Média - 1σ = 98.49
+1σ = 98.97
Média = 98.73
-1σ = 98.49
Taxa de acerto do KNN
k
100.0
99.8
99.6
99.4
99.2
99.0
98.8
98.6
0
20
40
60
80
100


Performance do KNN em função do k (Dados não-linearmente separáveis)
+1σ = 98.63
Média = 98.19
-1σ = 97.74
Taxa de acerto do KNN
k
Desvio padrão = 0.44
Accuracy
Média = 98.19
Média + 1σ = 98.63
Média - 1σ = 97.74