Preparando os dados para treinamento

A base de dados completa está, à partir da raiz desse repositório, em `/iris/iris.data`. São ao todo 150 exemplares de flores, divididos em 50 de Iris-setosa, 50 de Iris-versicolor e 50 de Iris-virginica. O documento da especificação do trabalho diz que a base deve ser dividida em três partições (A,B,C) de tamanho igual, levando em consideração que cada partição precisa ter aproximadamente a mesma proporção de espécies de flor. Isso será feito logo baixo. Mas antes, vamos visualizar algumas informações sobre essa base de dados.

Visualização dos dados

Temos 5 colunas ao todo nessa base de dados, sem valores faltantes, e queremos criar modelos de aprendizado de máquina capazes de prever a quinta coluna, que é a espécie da flor. Para ser mais específico, cada coluna da amostra segue o padrão abaixo:

Atributo	Tipo	Medida	Valores Faltantes
Comprimento da Sépala	Contínuo	cm	Não
Largura da Sépala	Contínuo	cm	Não
Comprimento da Pétala	Contínuo	cm	Não
Largura da Pétala	Contínuo	cm	Não
Espécie	Alvo	categórica	Não

Vamos primeiro visualizar um gráfico de dispersão da largura e comprimento das sépalas.

In [419…

```
# Data setup
import pandas as pd

# Assuming the Iris dataset CSV has column headers 'sepal_length', 'sepal_width', 'petal
iris = pd.read_csv('../Iris/iris.data', header=None)
iris.columns = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'species']
features = iris[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
target = iris['species']

iris
```

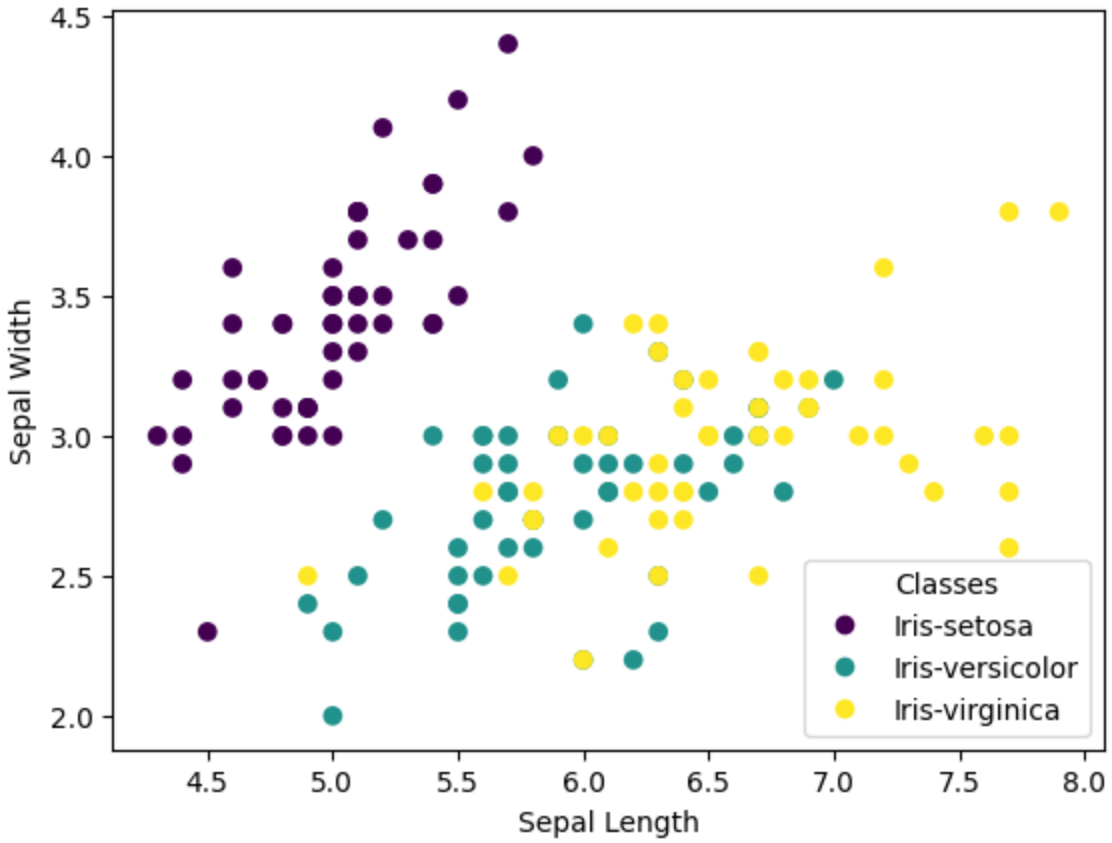
Out[419]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

In [420…

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
iris['species_encoded'] = le.fit_transform(iris['species'])

target_encoded = iris['species_encoded']

import matplotlib.pyplot as plt

# Create a scatter plot
_, ax = plt.subplots()
# just to remember, iloc selects the integer location of that feature collumn.
# for example, features.iloc[:, 0] selects all rows (exemplars) of the collumn 0
# So, in this scenario, scatter plot is basically scattering the sepal length and the se
scatter = ax.scatter(features.iloc[:, 0], features.iloc[:, 1], c=target_encoded)

# Set the labels for the axes
ax.set(xlabel='Sepal Length', ylabel='Sepal Width')

# Automatically generate the legend from the scatter
legend1 = ax.legend(*scatter.legend_elements(), loc="lower right", title="Classes")

# If you want to replace the labels with the species names from LabelEncoder
for i, text in enumerate(legend1.get_texts()):
    text.set_text(le.classes_[i])
```

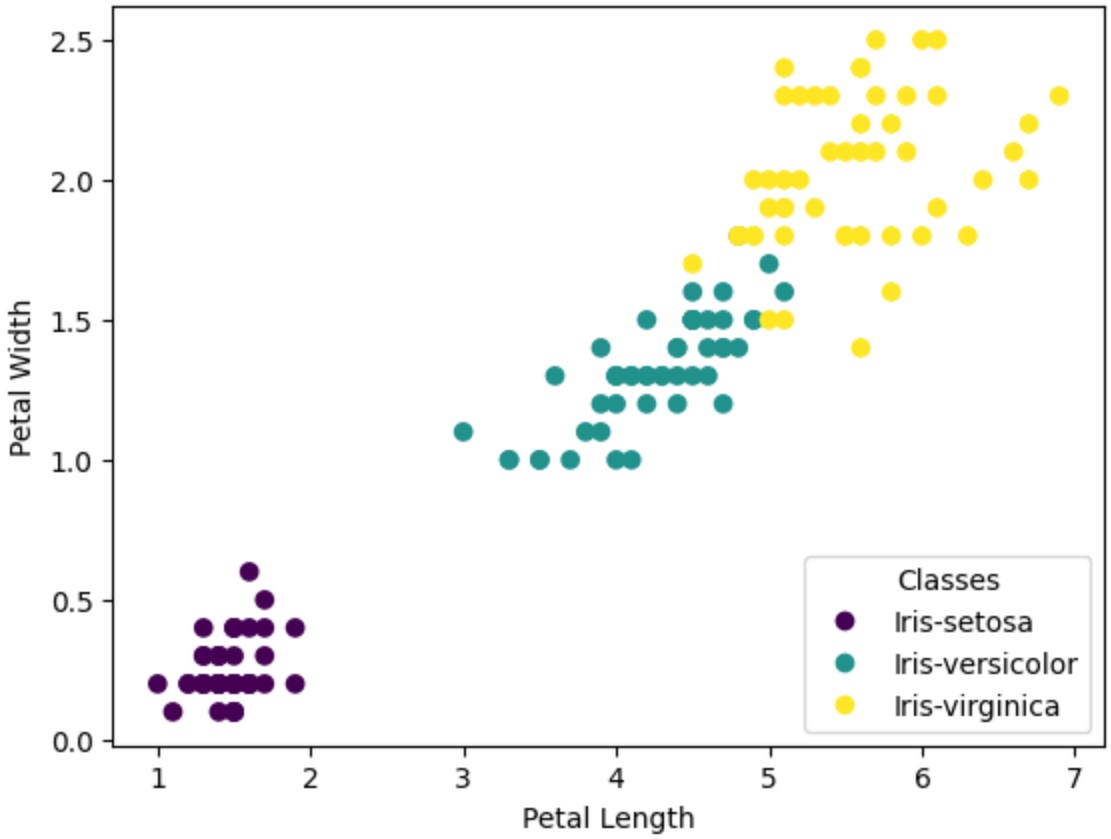


Cada ponto no gráfico acima é uma amostra das flores no dataset, e as cores representam seus tipos. Dá para ver claramente que tem um padrão de sépalas mais curtas e largas para as do tipo Iris-setosa. Mas isso já não é tão verdade para os outros dois tipos: as flores das espécies Iris-versicolor e Iris-virginica se sobrepõem frequentemente. Talvez seja porque está sendo considerado apenas duas dimensões. Vamos ver outros gráficos de dispersão que usam as informações de outras colunas para ver os dados em outra perspectiva.

No gráfico abaixo, um gráfico de dispersão do comprimento e largura de pétalas:

In [421…

```
# Create a scatter plot
_, ax = plt.subplots()
scatter = ax.scatter(features.iloc[:, 2], features.iloc[:, 3], c=target_encoded)

ax.set(xlabel='Petal Length', ylabel='Petal Width')

legend1 = ax.legend(*scatter.legend_elements(), loc="lower right", title="Classes")

for i, text in enumerate(legend1.get_texts()):
    text.set_text(le.classes_[i])
```



Usando as medidas das pétalas, temos clusters mais concentrados de Iris-versicolor e Iris-virgniica. Nessas medidas, o tipo Iris-setosa está bem isolada do restante, sendo um outlier. Mas ainda assim existem sobreposições das amostras.

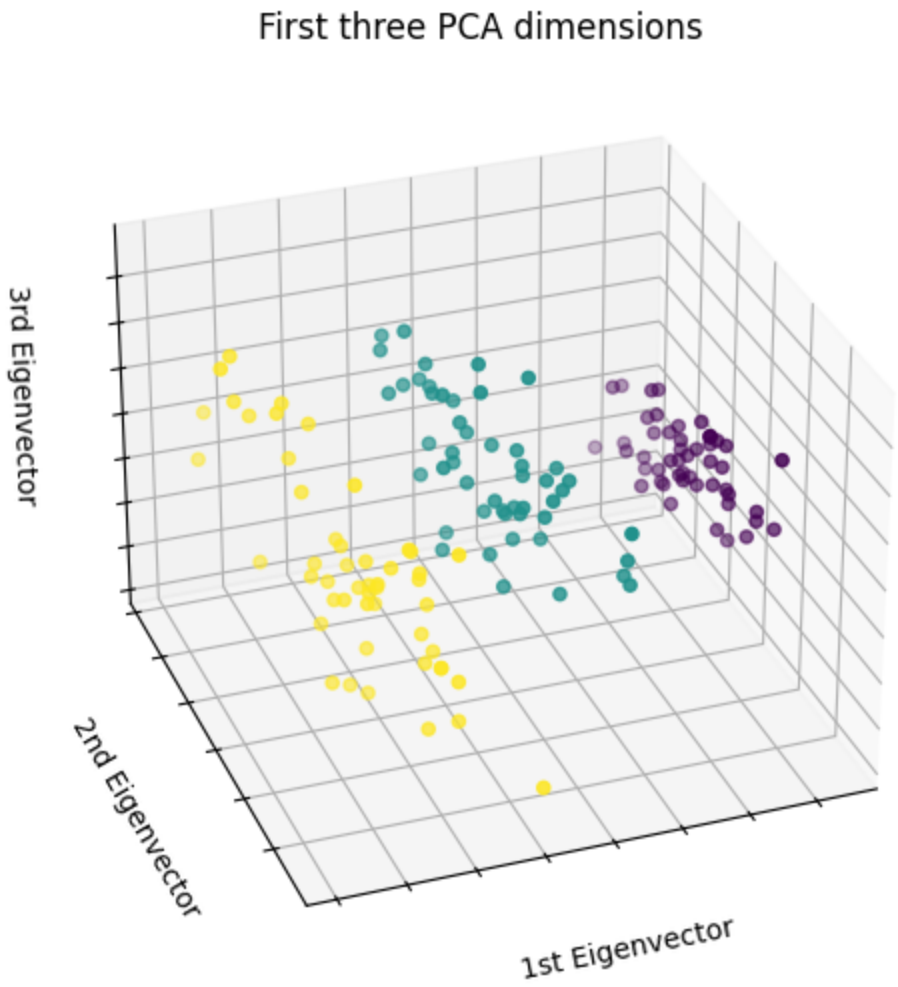
Como temos 4 dimensões de dados contínuos (largura e altura de pétalas e sépalas), podemos usar a Análise de Componentes Principais para reduzir a dimensão e remover ruídos, enquanto retemos o máximo de informação que conseguimos.

In [422…

```
from sklearn.decomposition import PCA

fig = plt.figure(1, figsize=(8, 6))
ax = fig.add_subplot(111, projection="3d", elev=-150, azim=110)

# Assuming 'iris' is your DataFrame and it has the species encoded numerically
features = iris.drop('species', axis=1)  # or whichever column is the target
target = iris['species_encoded']  # make sure this is the numeric target
iris['species_encoded'] = le.fit_transform(iris['species'])

# Apply PCA
X_reduced = PCA(n_components=3).fit_transform(features)

ax.scatter(
    X_reduced[:, 0],
    X_reduced[:, 1],
    X_reduced[:, 2],
    c=target,
)

ax.set_title("First three PCA dimensions")
ax.set_xlabel("1st Eigenvector")
ax.xaxis.set_ticklabels([])
ax.set_ylabel("2nd Eigenvector")
ax.yaxis.set_ticklabels([])
```

```
ax.set_zlabel("3rd Eigenvector")
ax.zaxis.set_ticklabels([])
```

Out[422]:
```
[Text(-0.8, 0, ''),
 Text(-0.6000000000000001, 0, ''),
 Text(-0.4, 0, ''),
 Text(-0.19999999999999996, 0, ''),
 Text(0.0, 0, ''),
 Text(0.19999999999999996, 0, ''),
 Text(0.40000000000000013, 0, ''),
 Text(0.6000000000000001, 0, ''),
 Text(0.8, 0, ''),
 Text(1.0, 0, '')]
```



O método ACP cria 3 novas características (Eigenvectors) que são uma combinação linear das 4 características originais. Além disso, essa transformação maximiza a variância. Nesse gráfico, podemos ver que cada espécie pode ser praticamente identificada usando o primeiro Eigenvector.

Partição dos dados

É possível termos 3 partições de 50 amostras cada. Mas elas serão exatamente iguais, pois temos que ter 1/3 das amostras de cada espécie em cada partição. Mas dá para chegar bem perto disso. O código abaixo extrai uma amostra de cada espécie, até que a partição fique cheia.

In [423…
```
import pandas as pd

# Carregar o dataset
iris = pd.read_csv('../Iris/iris.data', header=None)
```

```python
# Definir os nomes das colunas se o CSV não tiver cabeçalho
iris.columns = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'species']

# Criar uma lista de DataFrames, um para cada espécie
groups = [df for _, df in iris.groupby('species')]

# Inicializar as partições como listas vazias de DataFrames
partitions = [[], [], []]

# Distribuir as amostras igualmente
while any(not group.empty for group in groups):
    for i in range(3):
        if not groups[i].empty:
            for j in range(3):
                # Adicionar a primeira linha do grupo como um DataFrame na partição corr
                partitions[i].append(groups[j].iloc[[0]])
                # Remover a primeira linha do grupo
                groups[j] = groups[j].iloc[1:]

# Concatenar todas as listas de DataFrames para formar as partições finais
A = pd.concat(partitions[0], ignore_index=True)
B = pd.concat(partitions[1], ignore_index=True)
C = pd.concat(partitions[2], ignore_index=True)
```

Vamos checar se a quantidade de dados está correto e se não tem repetição nas partições entre si:

In [424…
```python
# Supondo que 'species' seja o nome da coluna com os tipos de flor
for i, df in enumerate([A,B,C]):
    print(f"Contagem de espécies na Partição {i+1}:")
    print(df['species'].value_counts())
    print()  # Apenas para adicionar uma linha vazia entre as partiçõe
```

```
Contagem de espécies na Partição 1:
Iris-setosa        17
Iris-versicolor    17
Iris-virginica     17
Name: species, dtype: int64

Contagem de espécies na Partição 2:
Iris-setosa        17
Iris-versicolor    17
Iris-virginica     17
Name: species, dtype: int64

Contagem de espécies na Partição 3:
Iris-setosa        16
Iris-versicolor    16
Iris-virginica     16
Name: species, dtype: int64
```

In [425…
```python
for i, df in enumerate([A, B, C]):
    print(f"Partição {i+1} tem linhas duplicadas? {'Sim' if df.duplicated().any() else '
```

```
Partição 1 tem linhas duplicadas? Não
Partição 2 tem linhas duplicadas? Sim
Partição 3 tem linhas duplicadas? Não
```

Uma coisa qu eme incomodou foram essas duplicadas na partição B, mas elas estão no dataset principal, inclusive com algumas outras:

In [426…
```
duplicadas = B.duplicated(keep=False)
linhas_duplicadas = B[duplicadas]
print(linhas_duplicadas)
```

```
    sepal_length  sepal_width  petal_length  petal_width      species
33           4.9          3.1           1.5          0.1  Iris-setosa
36           4.9          3.1           1.5          0.1  Iris-setosa
```

In [427…
```
duplicadas = iris.duplicated(keep=False)
linhas_duplicadas = iris[duplicadas]
print(linhas_duplicadas)
```

```
     sepal_length  sepal_width  petal_length  petal_width         species
9             4.9          3.1           1.5          0.1     Iris-setosa
34            4.9          3.1           1.5          0.1     Iris-setosa
37            4.9          3.1           1.5          0.1     Iris-setosa
101           5.8          2.7           5.1          1.9  Iris-virginica
142           5.8          2.7           5.1          1.9  Iris-virginica
```

Faz sentido eu remover essas duplicada da partição em que estão e colocar em uma outra:

In [428…
```
duplicadas = B.duplicated(keep=False)
index_da_duplicada = duplicadas[duplicadas].index[0]

# Copie a linha duplicada para o DataFrame C
duplicated_row = B.loc[[index_da_duplicada]]
C = pd.concat([C, duplicated_row], ignore_index=True)

# Remova a linha duplicada do DataFrame original
B = B.drop(index_da_duplicada)
```

Pronto, agora vemos que as partições, não possuem mais duplicadas, e temos os dados um pouco melhor distribuídos:

In [429…
```
for i, df in enumerate([A, B, C]):
    print(f"Partição {i+1} tem linhas duplicadas? {'Sim' if df.duplicated().any() else '
```

```
Partição 1 tem linhas duplicadas? Não
Partição 2 tem linhas duplicadas? Não
Partição 3 tem linhas duplicadas? Não
```

In [430…
```
# Supondo que 'species' seja o nome da coluna com os tipos de flor
for i, df in enumerate([A,B,C]):
    print(f"Contagem de espécies na Partição {i+1}:")
    print(df['species'].value_counts())
    print()  # Apenas para adicionar uma linha vazia entre as partiçõe
```

```
Contagem de espécies na Partição 1:
Iris-setosa        17
Iris-versicolor    17
Iris-virginica     17
Name: species, dtype: int64

Contagem de espécies na Partição 2:
Iris-versicolor    17
Iris-virginica     17
Iris-setosa        16
Name: species, dtype: int64

Contagem de espécies na Partição 3:
Iris-setosa        17
Iris-versicolor    16
Iris-virginica     16
Name: species, dtype: int64
```

As partições estão salvas em `/Iris`:

In [431…

```python
# Save DataFrame A to CSV
A.to_csv('../Iris//df_A.csv', index=False)

# Save DataFrame B to CSV
B.to_csv('../Iris//df_B.csv', index=False)

# Save DataFrame C to CSV
C.to_csv('../Iris//df_C.csv', index=False)
```

Árvore de decisão aplicada na base Íris

Agora com as partições A, B e C, podemos aplicar o algoritmo de árvore de decisão com a métrica de entropia.

Teoricamente, a equação de entropia é dada por:

$$\text{Entropia}(S) = -\sum_{i=1}^{n} p_i \log_2 p_i$$

Na própria documentação do scikit learn aqui podemos ver que para o critério de entropia o mesmo cálculo é adotado.

Usando-se a entropia, podemos calcular o ganho de cada variável. A que tiver o maior ganho, será usada para o nó inicial de decisão. O processo de calcular o ganho precisa ser repetido para a geração de cada novo nó, isolando-se as amostras que são filtradas pelo pai desses nós.

Vamos carregar os nossos datasets abaixo e construir a árvore de decisão, para os três experimentos, assim como pede a especificação do segundo trabalho.

Os resultados de acurácia, sensitividade, especificidade e outras métricas serão armazenada aqui, para uso posterior em comparação:

In [49]:
```
import pandas as pd

metrics_df = pd.DataFrame(columns=["treinamento", "acurácia", "sensitividade", "especifi

metrics_df
```

Out[49]:

treinamento	acurácia	sensitividade	especificidade	precision

Primeiro: Treinamento (A+B) e Teste (C)

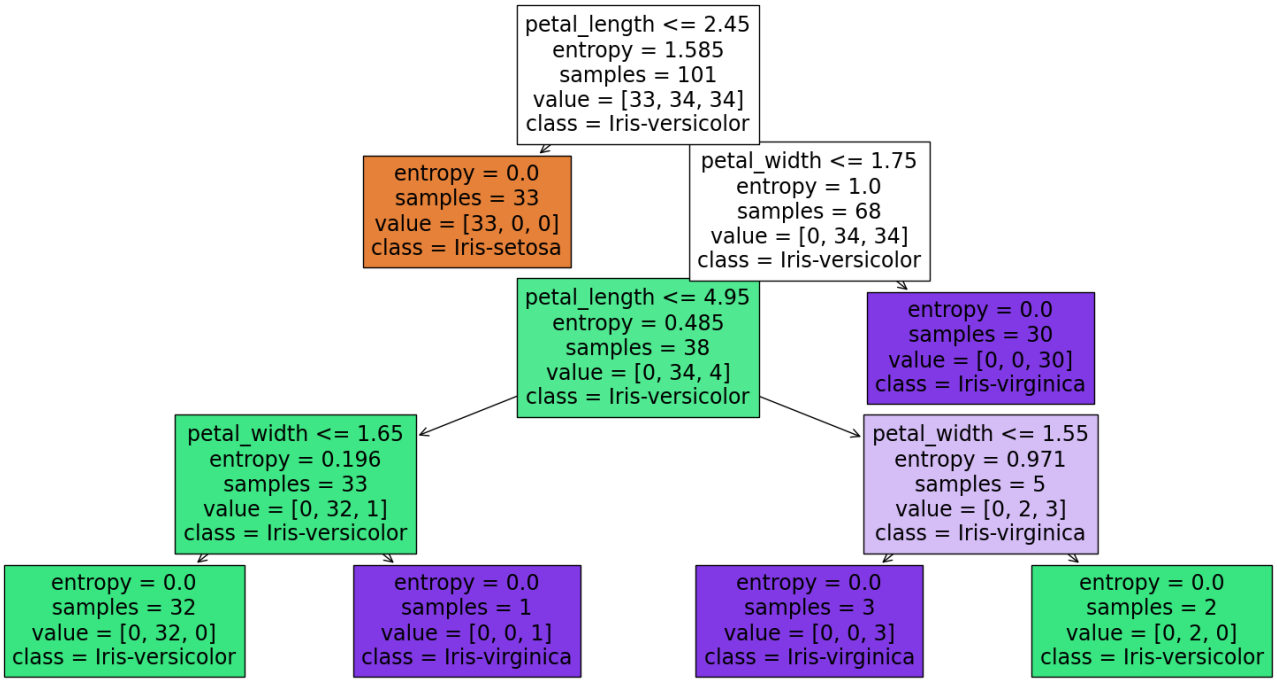
carregar todos os datasets primeiro:

In [50]:
```
A = pd.read_csv('../Iris/df_A.csv', header=None)
B = pd.read_csv('../Iris/df_B.csv', header=None)
C = pd.read_csv('../Iris/df_C.csv', header=None)

# Set the first row as the header
A.columns = A.iloc[0]
B.columns = B.iloc[0]
C.columns = C.iloc[0]

# Drop the first row now that the headers are set
A = A.drop(A.index[0])
B = B.drop(B.index[0])
C = C.drop(C.index[0])

# Reset the index if needed
A.reset_index(drop=True, inplace=True)
```

```
B.reset_index(drop=True, inplace=True)
C.reset_index(drop=True, inplace=True)
```

Abaixo, os cálculos de acurácia, sensitividade, especificidade e precisão, seguindo essa ordem.

In [51]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Concatenate datasets A and B to form the training set
# the X_train contains
X_train = pd.concat([A.iloc[:, :-1], B.iloc[:, :-1]], ignore_index=True)
y_train = pd.concat([A.iloc[:, -1], B.iloc[:, -1]], ignore_index=True)

# Use dataset C as the test set
X_test = C.iloc[:, :-1]
y_test = C.iloc[:, -1]


# Create and fit the decision tree classifier
clf = DecisionTreeClassifier(criterion='entropy')
clf.fit(X_train, y_train)

# Predict the labels for the test set
y_pred = clf.predict(X_test)

from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

# Assuming clf is your trained DecisionTreeClassifier from above code
plt.figure(figsize=(20,10))
plot_tree(clf, filled=True, feature_names=X_train.columns, class_names=y_train.unique())
plt.show()
```



In [52]:
```
# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# Output the accuracy
print(f"Acurácia: {accuracy:.2f}")
```

```
Acurácia: 0.96
```

Temos alguns jeitos de gerar a métrica de precisão. Os dados dessa base são balanceados, com o mesmo número de instâncias de cada espécie. Nesses casos, o tipo "macro" é o mais adequado. Essa métrica trata todas as classes igualmente, dando um peso igual para cada uma, sem considerar as frequências.

In [53]:
```
from sklearn.metrics import precision_score

precision = precision_score(y_test, y_pred, average='macro')

print(f"Precisão: {precision:.2f}")
```

```
Precisão: 0.96
```

Sensitividade (ou recall) é a medida do quão bom um modelo é em identificar os positivos verdadeiros para os casos positivos de fato das amostras. Ele é calculado como o número de positivos verdadeiros divididos pelo número de todas as amostras relevantes. Ou seja, todas as amostras que deviam ser identificadas como positivas.

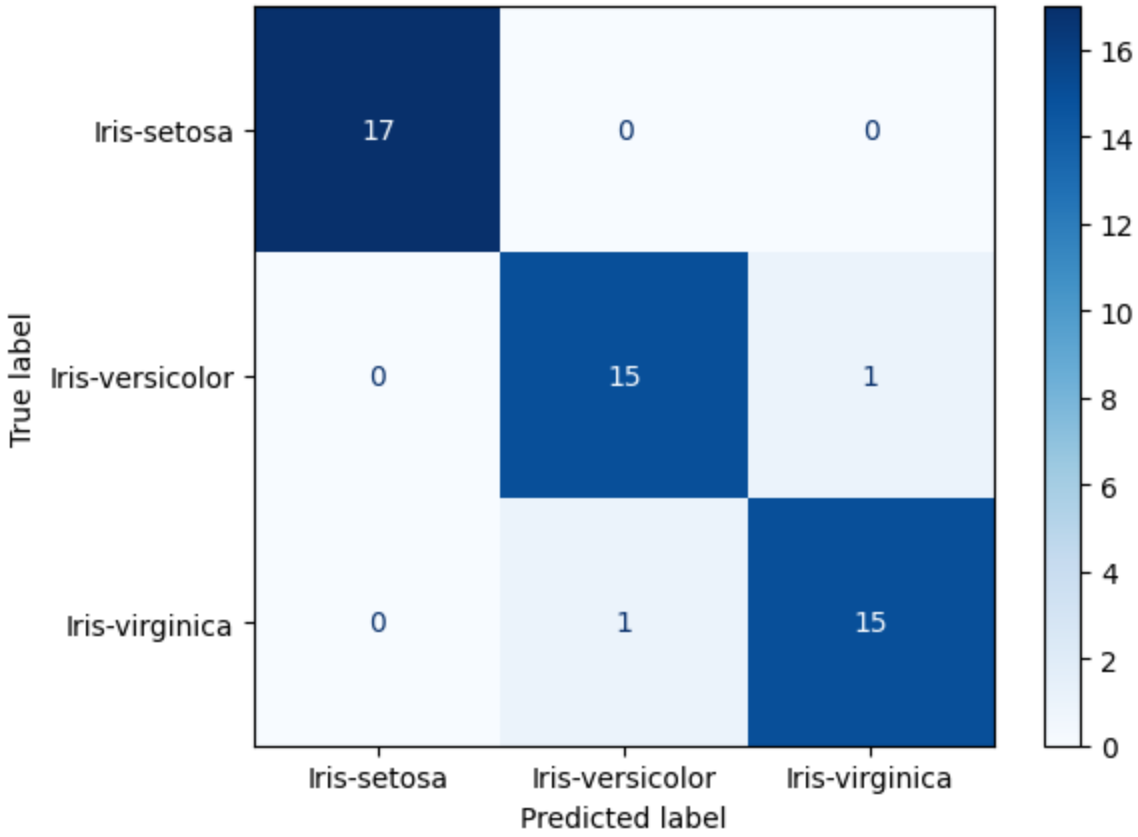
No contexto da base, sensitividade trata da capacidade de identificar corretamente Iris-setosa, por exemplo, quando ela o é de fato.

In [54]:
```
from sklearn.metrics import recall_score

sensitivity = recall_score(y_test, y_pred, average='macro')

print(f"Sensitividade: {sensitivity:.2f}")
```

```
Sensitividade: 0.96
```

A especificidade mede o quão bem o modelo identifica as outras espécies (que seriam os casos negativos quando estamos focados em detectar Iris-setosa) corretamente. Ou seja, quando o modelo prevê que uma flor não é Iris-setosa, a especificidade nos diz quantas vezes essa previsão está correta, indicando que a flor é realmente uma Iris-versicolor ou Iris-virginica.

In [55]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, y_pred)

# Plotting the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_)
disp.plot(cmap=plt.cm.Blues)  # You can specify other colormaps like 'viridis', 'plasma'
plt.show()
```



In [56]:
```
specificity_per_class = []
for i in range(len(cm)):
    tn = cm[0, 0] + cm[1, 1] - cm[i, i]
    fp = cm[i, :].sum() - cm[i, i]
    specificity_i = tn / (tn + fp)
    specificity_per_class.append(specificity_i)

print(f"Especificidade por classe: {specificity_per_class}")
```

```
Especificidade por classe: [1.0, 0.9444444444444444, 0.9444444444444444]
```

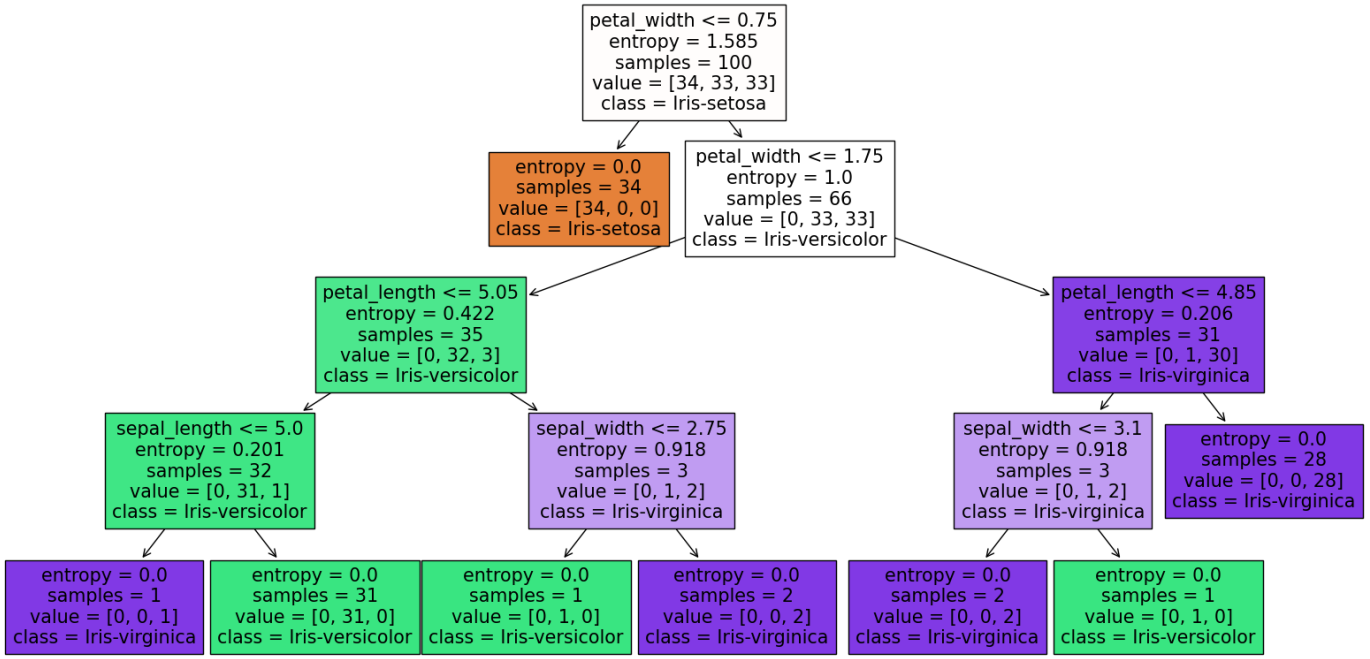
In [57]:
```
# Using loc to add a new row
metrics_df.loc[len(metrics_df)] = {
    "treinamento": "A+B e teste C",
    "acurácia": accuracy,
    "sensitividade": sensitivity,
    "especificidade": specificity_per_class,
    "precision": precision
}
```

Segundo: Treinamento (A+C) e Teste (B)

Os textos explicativos à partir de agora não existirão, porque a sequência à seguir é basicamente uma repetição anterior, com exceção de um pequeno trecho de código.

In [58]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Concatenate datasets A and B to form the training set
# the X_train contains
X_train = pd.concat([A.iloc[:, :-1], C.iloc[:, :-1]], ignore_index=True)
y_train = pd.concat([A.iloc[:, -1], C.iloc[:, -1]], ignore_index=True)
```

```
# Use dataset C as the test set
X_test = B.iloc[:, :-1]
y_test = B.iloc[:, -1]


# Create and fit the decision tree classifier
clf = DecisionTreeClassifier(criterion='entropy')
clf.fit(X_train, y_train)

# Predict the labels for the test set
y_pred = clf.predict(X_test)

from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

# Assuming clf is your trained DecisionTreeClassifier from above code
plt.figure(figsize=(20,10))
plot_tree(clf, filled=True, feature_names=X_train.columns, class_names=y_train.unique())
plt.show()
```



In [59]:
```
# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# Output the accuracy
print(f"Acurácia: {accuracy:.2f}")
```

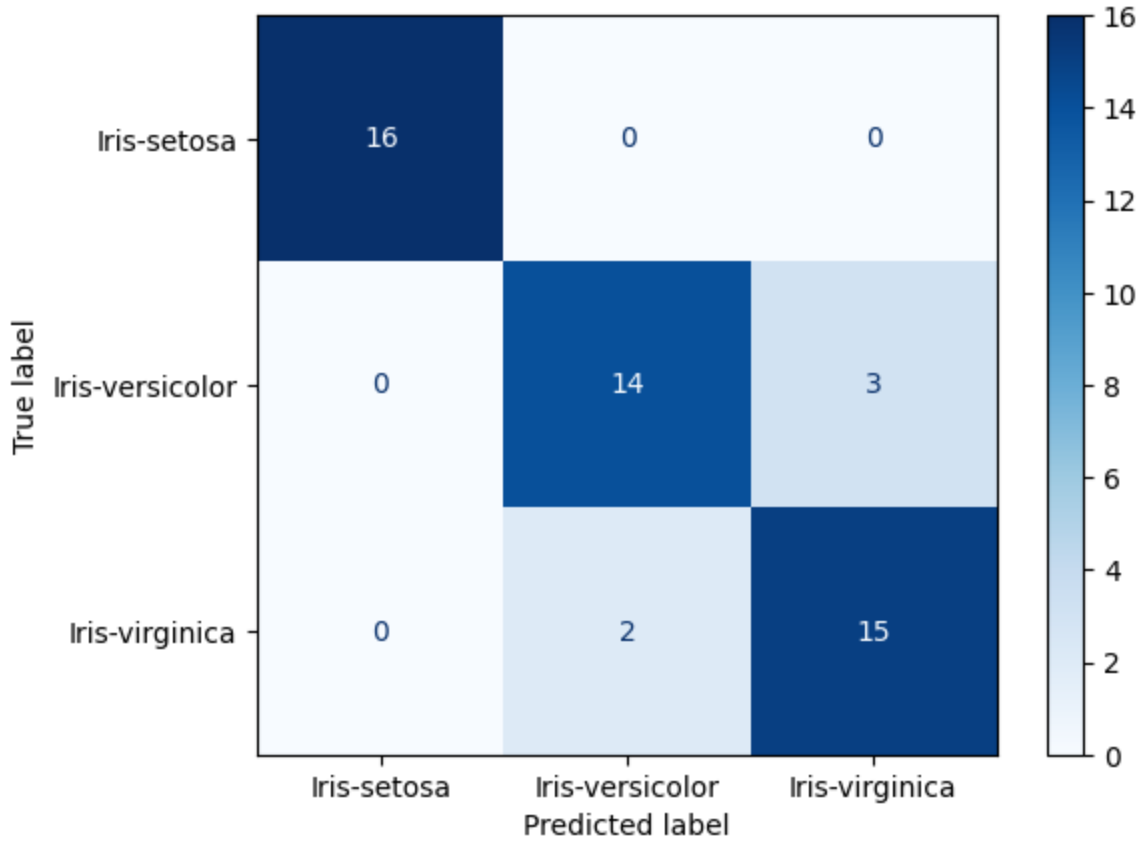
```
Acurácia: 0.90
```

In [60]:
```
from sklearn.metrics import recall_score

sensitivity = recall_score(y_test, y_pred, average='macro')

print(f"Sensitividade: {sensitivity:.2f}")
```

```
Sensitividade: 0.90
```

In [61]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, y_pred)
```

```
# Plotting the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_)
disp.plot(cmap=plt.cm.Blues)  # You can specify other colormaps like 'viridis', 'plasma'
plt.show()
```



In [62]:
```
specificity_per_class = []
for i in range(len(cm)):
    tn = cm[0, 0] + cm[1, 1] - cm[i, i]
    fp = cm[i, :].sum() - cm[i, i]
    specificity_i = tn / (tn + fp)
    specificity_per_class.append(specificity_i)

print(f"Especificidade por classe: {specificity_per_class}")
```

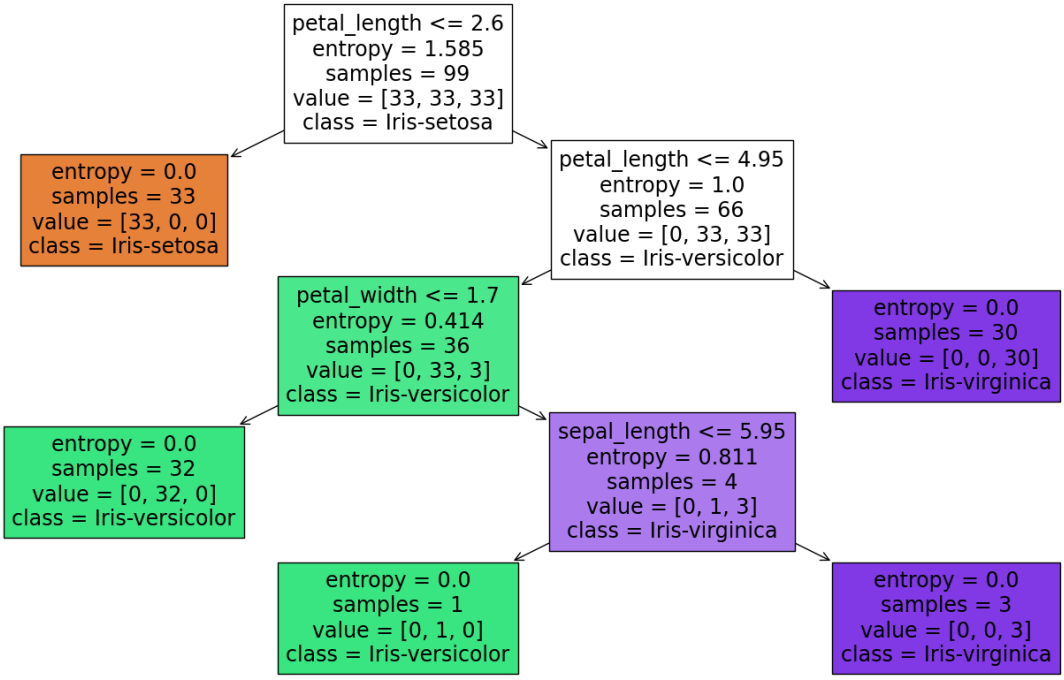
```
Especificidade por classe: [1.0, 0.8421052631578947, 0.8823529411764706]
```

In [63]:
```
# Using loc to add a new row
metrics_df.loc[len(metrics_df)] = {
    "treinamento": "A+C e teste B",
    "acurácia": accuracy,
    "sensitividade": sensitivity,
    "especificidade": specificity_per_class,
    "precision": precision
}
```

Terceiro: Treinamento (C+B) e Teste (A)

In [64]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Concatenate datasets A and B to form the training set
# the X_train contains
X_train = pd.concat([C.iloc[:, :-1], B.iloc[:, :-1]], ignore_index=True)
y_train = pd.concat([C.iloc[:, -1], B.iloc[:, -1]], ignore_index=True)
```

```
# Use dataset C as the test set
X_test = A.iloc[:, :-1]
y_test = A.iloc[:, -1]


# Create and fit the decision tree classifier
clf = DecisionTreeClassifier(criterion='entropy')
clf.fit(X_train, y_train)

# Predict the labels for the test set
y_pred = clf.predict(X_test)

from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

# Assuming clf is your trained DecisionTreeClassifier from above code
plt.figure(figsize=(20,10))
plot_tree(clf, filled=True, feature_names=X_train.columns, class_names=y_train.unique())
plt.show()
```



In [65]:
```
# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# Output the accuracy
print(f"Acurácia: {accuracy:.2f}")
```

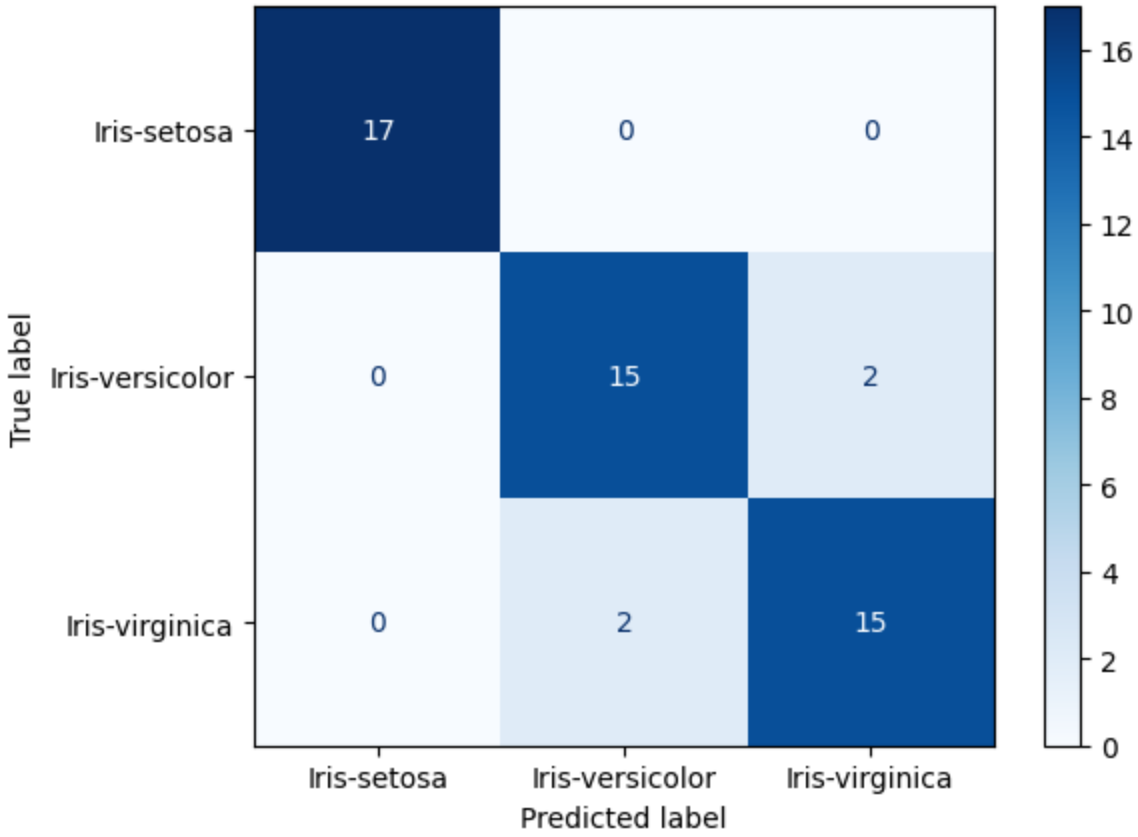
```
Acurácia: 0.92
```

In [66]:
```
from sklearn.metrics import recall_score

sensitivity = recall_score(y_test, y_pred, average='macro')

print(f"Sensitividade: {sensitivity:.2f}")
```

```
Sensitividade: 0.92
```

In [67]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, y_pred)
```

```
# Plotting the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_)
disp.plot(cmap=plt.cm.Blues)  # You can specify other colormaps like 'viridis', 'plasma'
plt.show()
```



In [68]:
```
specificity_per_class = []
for i in range(len(cm)):
    tn = cm[0, 0] + cm[1, 1] - cm[i, i]
    fp = cm[i, :].sum() - cm[i, i]
    specificity_i = tn / (tn + fp)
    specificity_per_class.append(specificity_i)

print(f"Especificidade por classe: {specificity_per_class}")
```

```
Especificidade por classe: [1.0, 0.8947368421052632, 0.8947368421052632]
```

In [69]:
```
# Using loc to add a new row
metrics_df.loc[len(metrics_df)] = {
    "treinamento": "C+B e teste A",
    "acurácia": accuracy,
    "sensitividade": sensitivity,
    "especificidade": specificity_per_class,
    "precision": precision
}
```

Resultados

Veja abaixo as métricas resultantes.

In [70]:
```
metrics_df
```

Out[70]:

	treinamento	acurácia	sensitividade	especificidade	precision
0	A+B e teste C	0.959184	0.958333	[1.0, 0.9444444444444444, 0.9444444444444444]	0.958333
1	A+C e teste B	0.900000	0.901961	[1.0, 0.8421052631578947, 0.8823529411764706]	0.958333
2	C+B e teste A	0.921569	0.921569	[1.0, 0.8947368421052632, 0.8947368421052632]	0.958333

Foi aberto o array de especificidade onde acada array é sua própria coluna agora. Cada uma delas representa respesctivamente, a especificidade de Iris-setosa, Iris-versicolor e Iris-virginica.

In [71]:
```
# Create individual columns for each value in the 'especificidade' list
especificidades = metrics_df['especificidade'].apply(pd.Series)

# Rename each new column to reflect what it represents
especificidades.columns = [f'especificidade_{i+1}' for i in range(especificidades.shape[

# Concatenate these new columns to the original DataFrame
metrics_df = pd.concat([metrics_df.drop('especificidade', axis=1), especificidades], axi

print(metrics_df)
```

```
     treinamento  acurácia  sensitividade  precision  especificidade_1  \
0  A+B e teste C  0.959184       0.958333   0.958333               1.0
1  A+C e teste B  0.900000       0.901961   0.958333               1.0
2  C+B e teste A  0.921569       0.921569   0.958333               1.0

   especificidade_2  especificidade_3
0          0.944444          0.944444
1          0.842105          0.882353
2          0.894737          0.894737
```

In [72]:
```
# Calculate the mean of each numerical column
mean_values = metrics_df.mean()

# Print the mean values
print(mean_values)
```

```
acurácia            0.926917
sensitividade       0.927288
precision           0.958333
especificidade_1    1.000000
especificidade_2    0.893762
especificidade_3    0.907178
dtype: float64
/tmp/ipykernel_19750/737132350.py:2: FutureWarning: The default value of numeric_only in
DataFrame.mean is deprecated. In a future version, it will default to False. In additio
n, specifying 'numeric_only=None' is deprecated. Select only valid columns or specify th
e value of numeric_only to silence this warning.
  mean_values = metrics_df.mean()
```

K Nearest Neighbours aplicado à base IRIS

Como solicitado na especificação do trabalho, a base de dados foi dividida em três partes: A, B e C - cada uma com a mesma proporção.

Inicialmente, devemos carregar o dataset previamente divido:

In [20]:
```
import pandas as pd
import seaborn as sns
# Try resetting it
sns.reset_orig()

A = pd.read_csv('../Iris/df_A.csv', header=None)
B = pd.read_csv('../Iris/df_B.csv', header=None)
C = pd.read_csv('../Iris/df_C.csv', header=None)

# Set the first row as the header
A.columns = A.iloc[0]
B.columns = B.iloc[0]
C.columns = C.iloc[0]

# Drop the first row now that the headers are set
A = A.drop(A.index[0])
B = B.drop(B.index[0])
C = C.drop(C.index[0])

# Reset the index if needed
A.reset_index(drop=True, inplace=True)
B.reset_index(drop=True, inplace=True)
C.reset_index(drop=True, inplace=True)
```

Com a base divida de acordo com a especificação, podemos iniciar os experimentos. A variável abaixo será utilizada posteriormente para compararmos os resultados e tirar uma conclusão baseado nos experimentos.

In [21]:
```
import pandas as pd

metrics_df = pd.DataFrame(columns=["treinamento", "acuracia", "sensitividade", "especifi
```

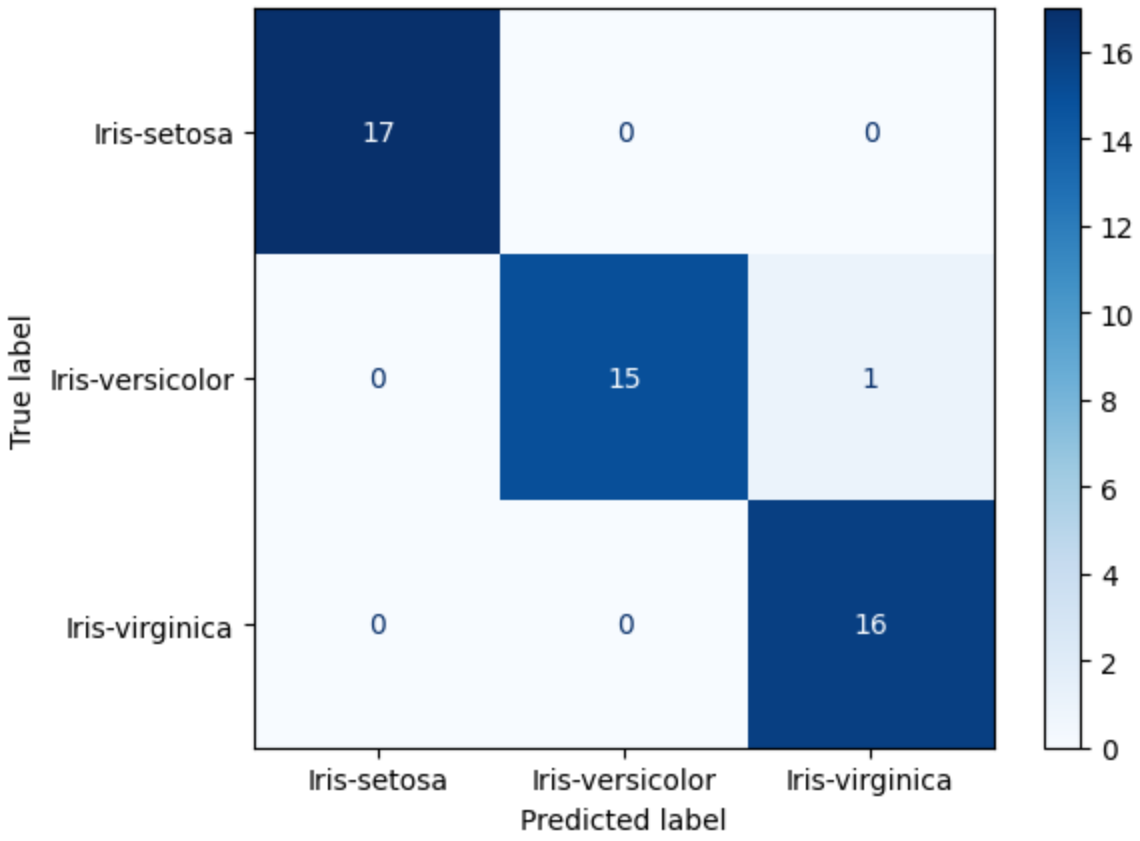
Primeiro: Treinamento (A+B) e Teste (C)

In [22]:
```
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import precision_score, accuracy_score, confusion_matrix, recall_sc

train = pd.concat([A, B])
test = pd.concat([C])

feature_columns = ['sepal_length', 'sepal_width', 'petal_length','petal_width']


x_train = train[feature_columns].values
y_train = train['species'].values
```

```
x_test = test[feature_columns].values
y_test = test['species'].values

classifier = KNeighborsClassifier(n_neighbors=3)
classifier.fit(x_train, y_train)

y_pred = classifier.predict(x_test)

cm = confusion_matrix(y_test, y_pred)
specificity_per_class = []
for j in range(len(cm)):
    tn = cm[0, 0] + cm[1, 1] - cm[j, j]
    fp = cm[j, :].sum() - cm[j, j]
    specificity_j = (tn / (tn + fp))
    specificity_per_class.append(specificity_j)

accuracy = accuracy_score(y_test, y_pred)*100
sensitivity = recall_score(y_test, y_pred, average='macro')
precision = precision_score(y_test, y_pred, average='macro')
```

In [23]:

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, y_pred)

# Plotting the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=classifier.classes_)
disp.plot(cmap=plt.cm.Blues)
plt.show()
```



In [24]:

```
metrics_df.loc[0] = {
    "treinamento": "A+B e teste C",
    "acuracia": accuracy,
    "sensitividade": sensitivity,
    "especificidade": specificity_per_class,
```

```
    "precisao": precision
}

metrics_df.loc[[0]]
```

Out[24]:

	treinamento	acuracia	sensitividade	especificidade	precisao
0	A+B e teste C	97.959184	0.979167	[1.0, 0.9444444444444444, 1.0]	0.980392

Segundo: Treinamento (A+C) e Teste (B)

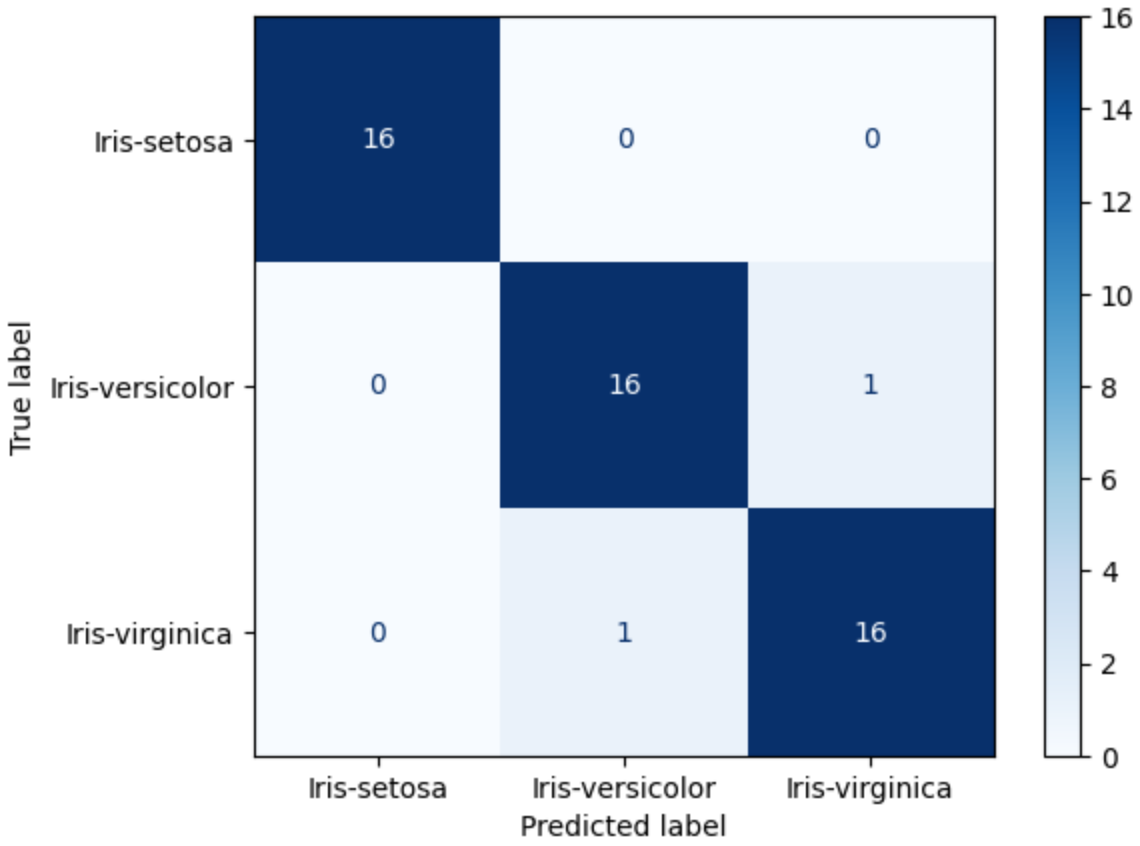
Os textos explicativos à partir de agora não existirão, porque a sequência à seguir é basicamente uma repetição anterior, com exceção de um pequeno trecho de código.

In [25]:

```
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import precision_score, accuracy_score, confusion_matrix, recall_sc

train = pd.concat([A, C])
test = pd.concat([B])

feature_columns = ['sepal_length', 'sepal_width', 'petal_length','petal_width']


x_train = train[feature_columns].values
y_train = train['species'].values

x_test = test[feature_columns].values
y_test = test['species'].values

classifier = KNeighborsClassifier(n_neighbors=3)
classifier.fit(x_train, y_train)

y_pred = classifier.predict(x_test)

cm = confusion_matrix(y_test, y_pred)
specificity_per_class = []
for j in range(len(cm)):
    tn = cm[0, 0] + cm[1, 1] - cm[j, j]
    fp = cm[j, :].sum() - cm[j, j]
    specificity_j = (tn / (tn + fp))
    specificity_per_class.append(specificity_j)

accuracy = accuracy_score(y_test, y_pred)*100
sensitivity = recall_score(y_test, y_pred, average='macro')
precision = precision_score(y_test, y_pred, average='macro')
```

In [26]:

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, y_pred)

# Plotting the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=classifier.classes_)
disp.plot(cmap=plt.cm.Blues)
plt.show()
```



In [27]:
```
metrics_df.loc[1] = {
    "treinamento": "A+C e teste B",
    "acuracia": accuracy,
    "sensitividade": sensitivity,
    "especificidade": specificity_per_class,
    "precisao": precision
}

metrics_df.loc[[1]]
```

Out[27]:

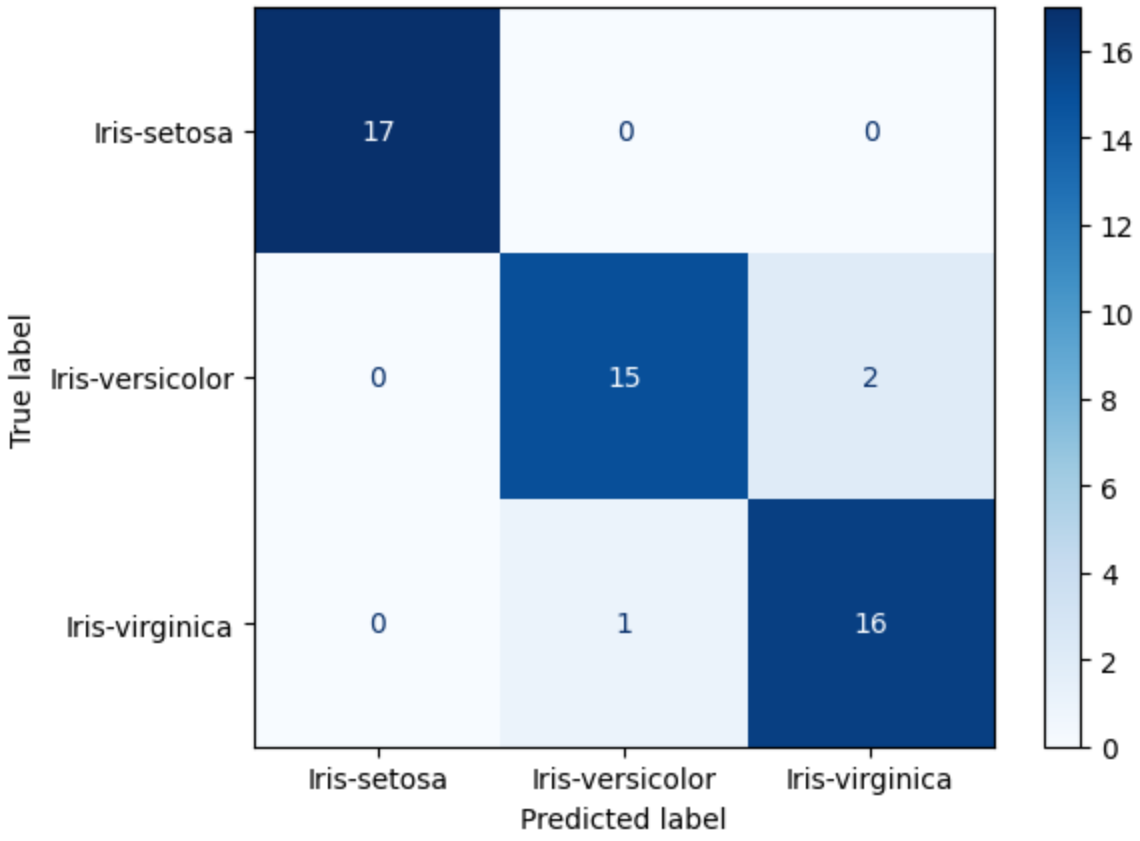
	treinamento	acuracia	sensitividade	especificidade	precisao
1	A+C e teste B	96.0	0.960784	[1.0, 0.9411764705882353, 0.9411764705882353]	0.960784

Terceiro: Treinamento (C+B) e Teste (A)

In [28]:
```
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import precision_score, accuracy_score, confusion_matrix, recall_sc

train = pd.concat([B, C])
test = pd.concat([A])

feature_columns = ['sepal_length', 'sepal_width', 'petal_length','petal_width']


x_train = train[feature_columns].values
y_train = train['species'].values

x_test = test[feature_columns].values
y_test = test['species'].values

classifier = KNeighborsClassifier(n_neighbors=3)
```

```
classifier.fit(x_train, y_train)

y_pred = classifier.predict(x_test)

cm = confusion_matrix(y_test, y_pred)
specificity_per_class = []
for j in range(len(cm)):
    tn = cm[0, 0] + cm[1, 1] - cm[j, j]
    fp = cm[j, :].sum() - cm[j, j]
    specificity_j = (tn / (tn + fp))
    specificity_per_class.append(specificity_j)

accuracy = accuracy_score(y_test, y_pred)*100
sensitivity = recall_score(y_test, y_pred, average='macro')
precision = precision_score(y_test, y_pred, average='macro')
```

In [29]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, y_pred)

# Plotting the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=classifier.classes_)
disp.plot(cmap=plt.cm.Blues)
plt.show()
```



In [30]:
```
metrics_df.loc[2] = {
    "treinamento": "B+C e teste A",
    "acuracia": accuracy,
    "sensitividade": sensitivity,
    "especificidade": specificity_per_class,
    "precisao": precision
}

metrics_df.loc[[2]]
```

Out[30]:

	treinamento	acuracia	sensitividade	especificidade	precisao
2	B+C e teste A	94.117647	0.941176	[1.0, 0.8947368421052632, 0.9411764705882353]	0.94213

Resultados

In [31]:
```
metrics_df
```

Out[31]:

	treinamento	acuracia	sensitividade	especificidade	precisao
0	A+B e teste C	97.959184	0.979167	[1.0, 0.9444444444444444, 1.0]	0.980392
1	A+C e teste B	96.000000	0.960784	[1.0, 0.9411764705882353, 0.9411764705882353]	0.960784
2	B+C e teste A	94.117647	0.941176	[1.0, 0.8947368421052632, 0.9411764705882353]	0.942130

Foi aberto o array de especificidade onde acada array é sua própria coluna agora. Cada uma delas representa respesctivamente, a especificidade de Iris-setosa, Iris-versicolor e Iris-virginica.

In [32]:
```
# Create individual columns for each value in the 'especificidade' list
especificidades = metrics_df['especificidade'].apply(pd.Series)

# Rename each new column to reflect what it represents
especificidades.columns = [f'especificidade_{i+1}' for i in range(especificidades.shape[

# Concatenate these new columns to the original DataFrame
metrics_df = pd.concat([metrics_df.drop('especificidade', axis=1), especificidades], axi

print(metrics_df)
```

```
     treinamento   acuracia  sensitividade  precisao  especificidade_1  \
0  A+B e teste C  97.959184       0.979167  0.980392               1.0
1  A+C e teste B  96.000000       0.960784  0.960784               1.0
2  B+C e teste A  94.117647       0.941176  0.942130               1.0

   especificidade_2  especificidade_3
0          0.944444          1.000000
1          0.941176          0.941176
2          0.894737          0.941176
```

In [33]:
```
# Calculate the mean of each numerical column
mean_values = metrics_df.mean(numeric_only=True)

# Print the mean values
print(mean_values)
```

```
acuracia            96.025610
sensitividade        0.960376
precisao             0.961102
especificidade_1     1.000000
especificidade_2     0.926786
especificidade_3     0.960784
dtype: float64
```

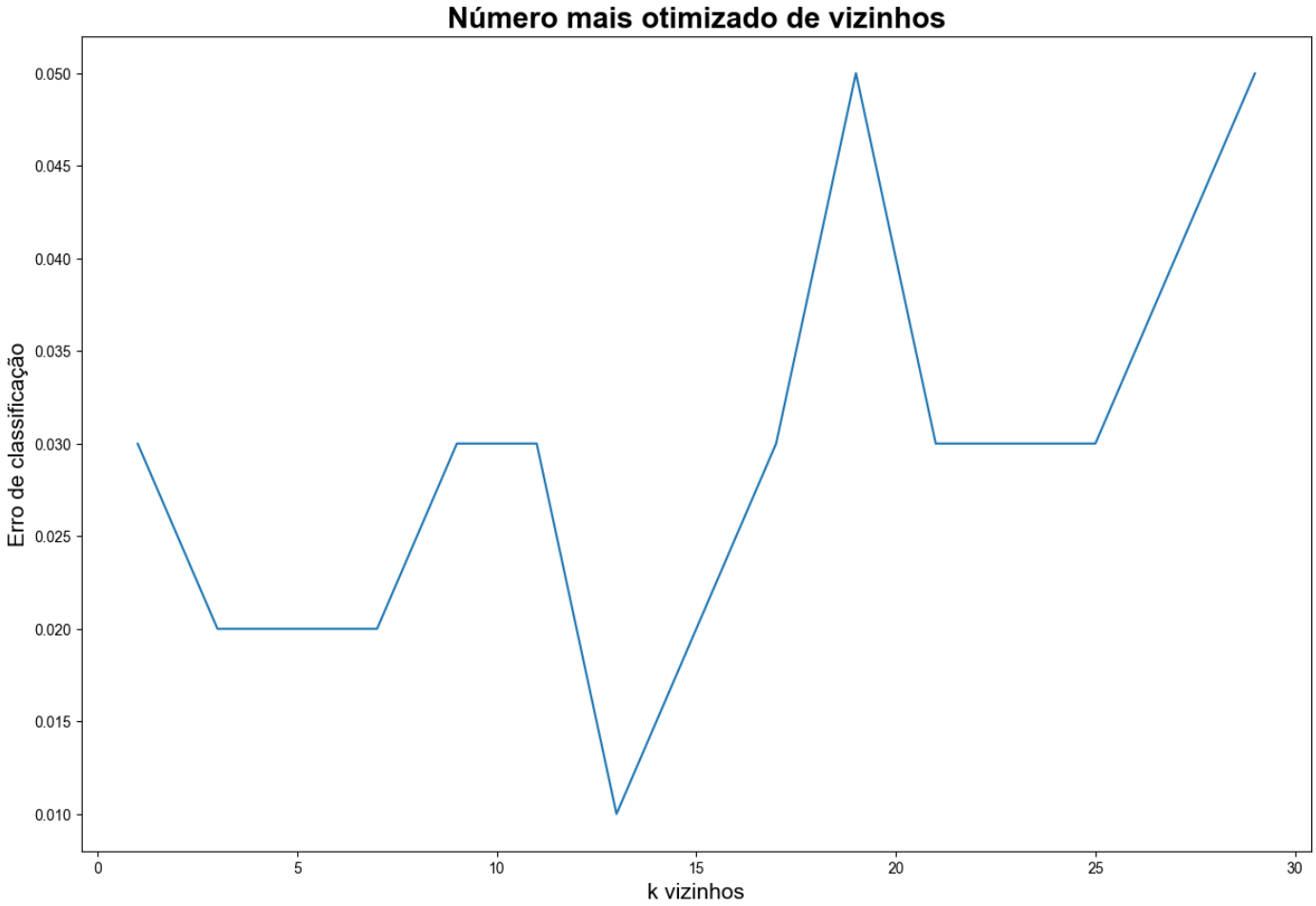
Qual o melhor N?

Apenas para fim de curiosidade, é possível calcular o *n* que melhor se adapta à esse conjunto de dados baseado na acurácia de cada *n*.

Utilizando o experimento 3 como exemplo, podemos encontrar o melhor n:

In [34]:
```
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
import seaborn as sns

k_list = list(range(1, 30,2))
cv_scores = []

# 10-fold cross validation
for k in k_list:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, x_train, y_train, cv=10, scoring='accuracy')
    cv_scores.append(scores.mean())


MSE = [1 - x for x in cv_scores]

plt.figure()
plt.figure(figsize=(15,10))
plt.title('Número mais otimizado de vizinhos', fontsize=20, fontweight='bold')
plt.xlabel('k vizinhos', fontsize=15)
plt.ylabel('Erro de classificação', fontsize=15)
sns.set_style("whitegrid")
plt.plot(k_list, MSE)

plt.show()


best_k = k_list[MSE.index(min(MSE))]
print("O melhor n para este experimento é %d." % best_k)
```

```
<Figure size 640x480 with 0 Axes>
```



O melhor n para este experimento é 13.