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
```

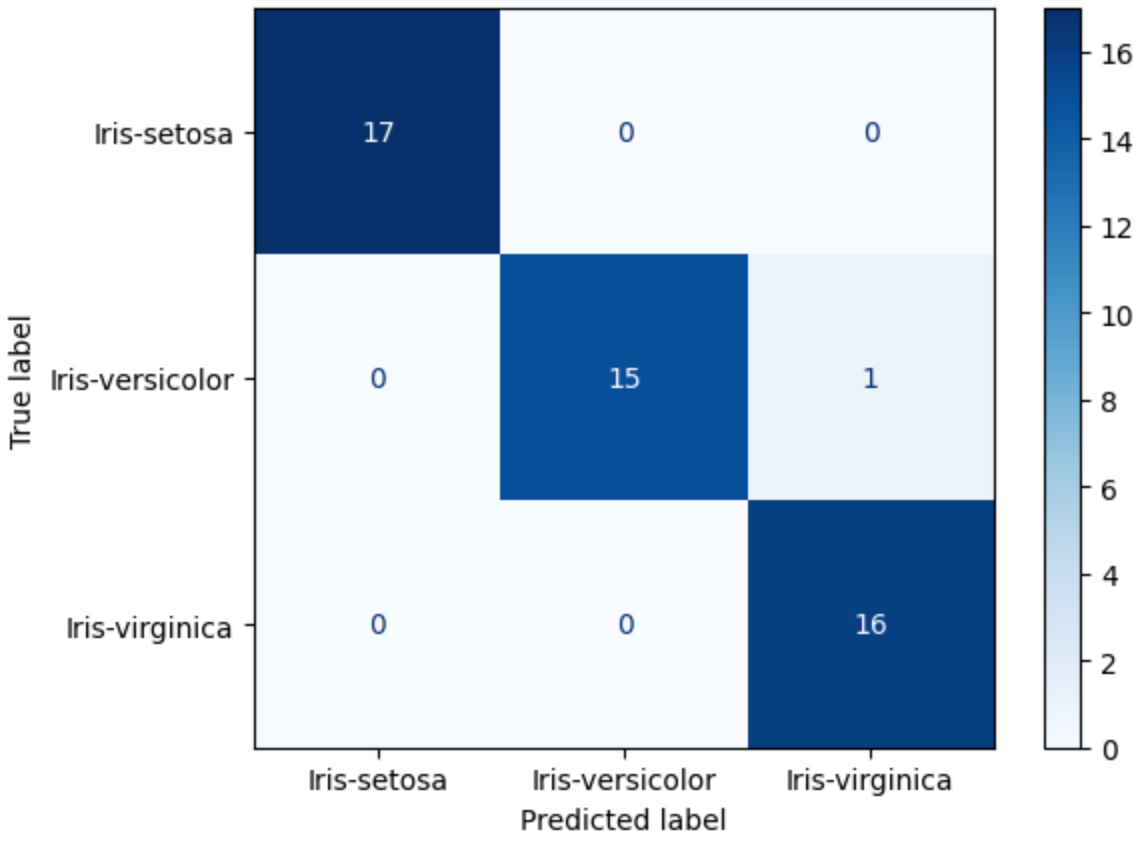
Primeiro: Treinamento (A+B) e Teste (C)

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)

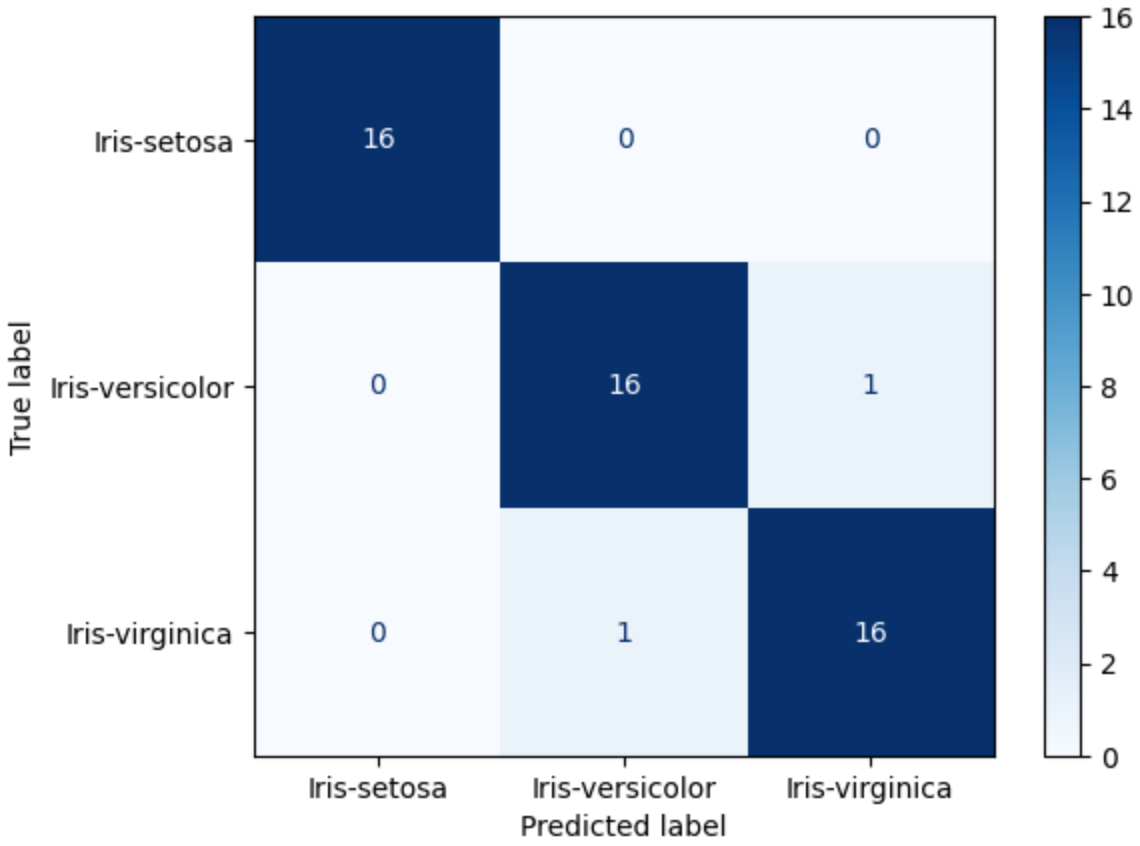
Os textos explicativos à partir de agora não existirão, porque a sequência à seguir é basicamente uma repetição anterior, com exceção de um pequeno trecho de código.

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]:

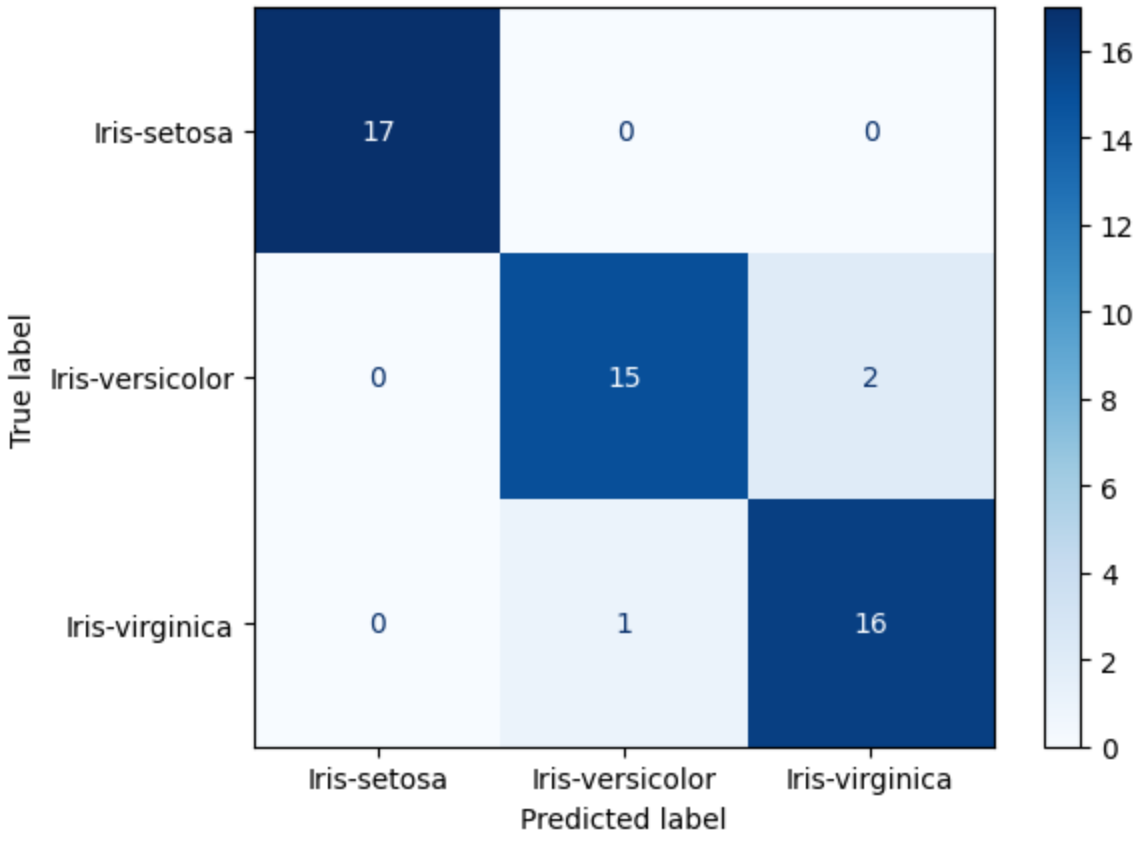
	treinamento	acuracia	sensitividade	especificidade	precisao
1	A+C e teste B	96.0	0.960784	[1.0, 0.9411764705882353, 0.9411764705882353]	0.960784

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
```

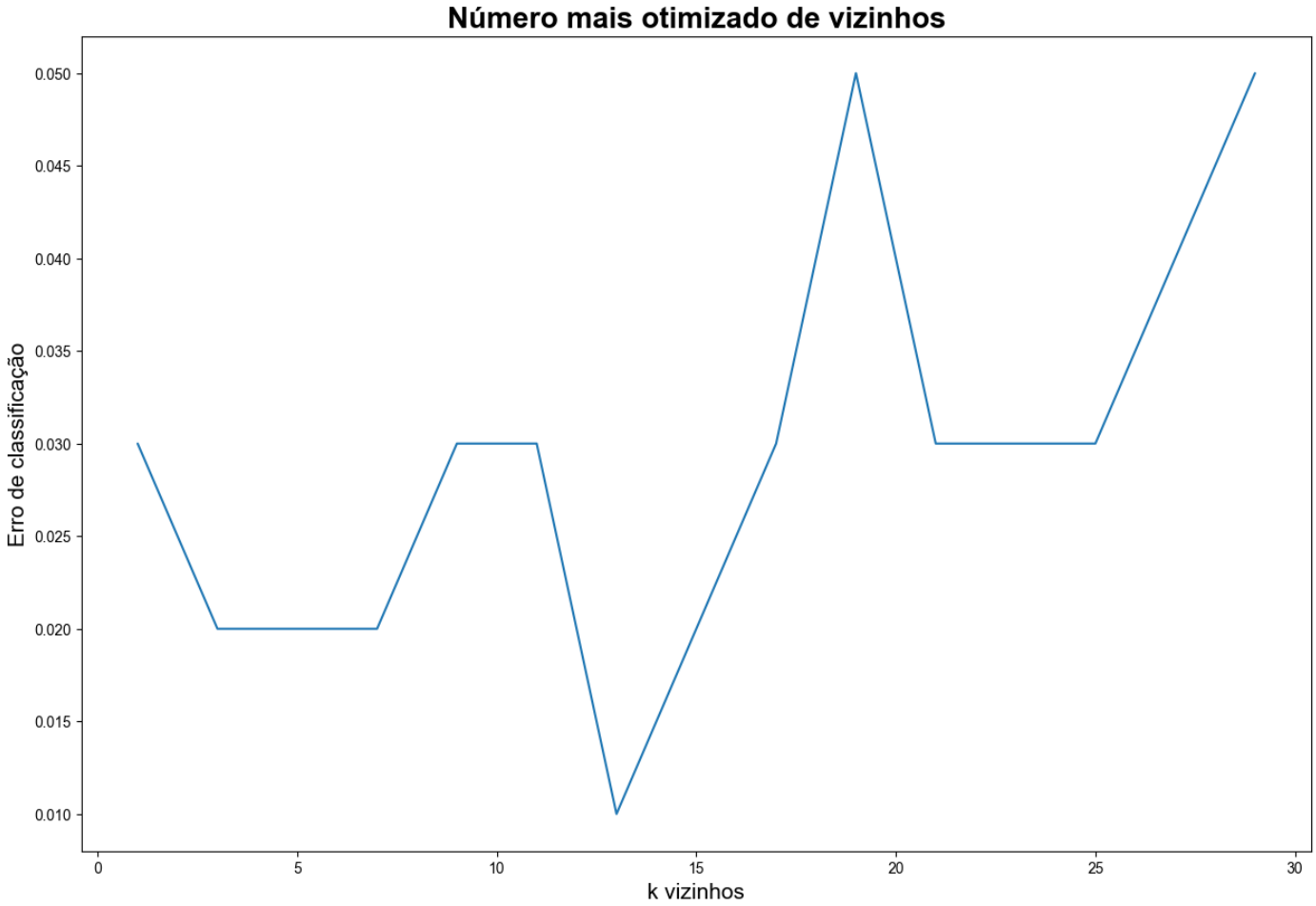
Qual o melhor N?

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.