SEGUNDO TRABALHO DE INTELIGÊNCIA ARTIFICIAL

Faculdade de Computação (FACOM)

Uberlândia, 20 de Novembro de 2023

Alunos

Bruno Sinhoroto

Lucas Pellegrini

Silvano Junior

Importação das Bibliotecas

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.tree import DecisionTreeClassifier
#from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_iris

from sklearn.tree import export_graphviz
from six import StringIO
from IPython.display import Image
import pydotplus
```

Instanciação e divisão da base de dados

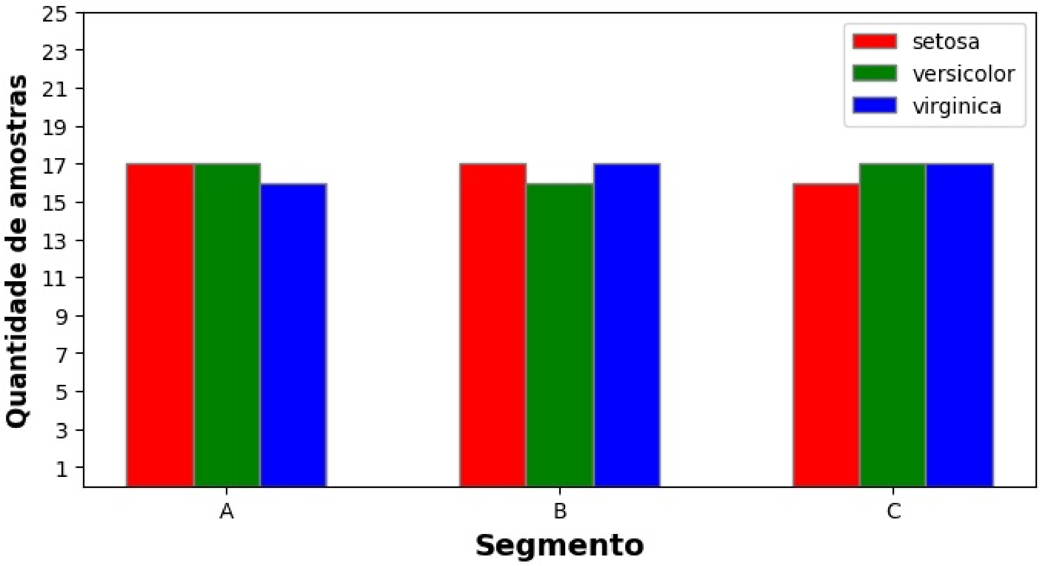
In [2]:
```
class IrisDB():
    XA : list[list[float]] = []
    YA : list[str] = []
    XB : list[list[float]] = []
    YB : list[str] = []
    XC : list[list[float]] = []
    YC : list[str] = []

    def __init__(self) -> None:
        self.db = load_iris()

        # Divisão A/B/C
        for k in range(0, 149, 3):
            self.XA.append(self.db['data'][k])
            self.YA.append(self.db['target'][k])

            self.XB.append(self.db['data'][k+1])
            self.YB.append(self.db['target'][k+1])

            self.XC.append(self.db['data'][k+2])
            self.YC.append(self.db['target'][k+2])


        self.indice = self.db['feature_names']
        self.metas = self.db['target_names']


# Classe para resultados da classificacao
class Classificacao():
    Y_real : list[float]
    Y_pred : list[float]
    acur   : float
    sens   : list[float]
    espe   : list[float]
    prec   : list[float]
```

```python
    def __init__(self, Y_real,  Y_pred) -> None:
        self.Y_pred = Y_pred
        self.Y_real = Y_real

        self.acur = metrics.accuracy_score(self.Y_real, self.Y_pred)
        mcm = metrics.multilabel_confusion_matrix(self.Y_real, self.Y_pred)
        tn = mcm[:, 0, 0]
        tp = mcm[:, 1, 1]
        fn = mcm[:, 1, 0]
        fp = mcm[:, 0, 1]
        self.sens = tp / (tp + fn) # (true positive) / (true positive + false negative)
        self.espe = tn / (tn + fp) # (true negative) / (true negative + false positive)
        self.prec = metrics.precision_score(self.Y_real, self.Y_pred, average=None)

    def print_metrics(self):
        print("\n|‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾|")
        print("|----------------------Acuracia--------------------|")
        print(f'|                        {self.acur}                       |')
        print("|___________________________________________________|")

        print("\n|‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾|")
        print("|-------------------Sensitividade------------------|")
        print(f'|   Setosa    | Versicolor | Virginica  |   Media    |')
        print(f'| {self.sens[0]:.8f} | {self.sens[1]:.8f} | {self.sens[2]:.8f} | {np.mean(self.sens):.8f} |')
        print("|___________________________________________________|")

        print("\n|‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾|")
        print("|-------------------Especificidade------------------|")
        print(f'|   Setosa    | Versicolor | Virginica  |   Media    |')
        print(f'| {self.espe[0]:.8f} | {self.espe[1]:.8f} | {self.espe[2]:.8f} | {np.mean(self.espe):.8f} |')
        print("|___________________________________________________|")

        print("\n|‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾|")
        print("|---------------------Precisao----------------------|")
        print(f'|   Setosa    | Versicolor | Virginica  |   Media    |')
        print(f'| {self.prec[0]:.8f} | {self.prec[1]:.8f} | {self.prec[2]:.8f} | {np.mean(self.prec):.8f} |')
        print("|___________________________________________________|")


iris = IrisDB()
```

Verificação da uniformidade da distribuição A/B/C

In [3]:

```python
# set width of bar
barWidth = 0.20
fig = plt.subplots(figsize =(8, 4))

# Set position of bar on X axis
br1 = np.arange(3)
br2 = [x + barWidth for x in br1]
br3 = [x + barWidth for x in br2]

# Make the plot
plt.bar(br1, [iris.YA.count(0), iris.YB.count(0), iris.YC.count(0)], color ='r', width = barWidth,
        edgecolor ='grey', label ='setosa')
plt.bar(br2, [iris.YA.count(1), iris.YB.count(1), iris.YC.count(1)], color ='g', width = barWidth,
        edgecolor ='grey', label ='versicolor')
plt.bar(br3, [iris.YA.count(2), iris.YB.count(2), iris.YC.count(2)], color ='b', width = barWidth,
        edgecolor ='grey', label ='virginica')

# Adding Xticks
plt.xlabel('Segmento', fontweight ='bold', fontsize = 14)
plt.ylabel('Quantidade de amostras', fontweight ='bold', fontsize = 12)
plt.xticks([r + barWidth for r in range(3)],
        ['A', 'B', 'C'])
plt.yticks([1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25])

plt.legend()
plt.show()
```



Árvore de Decisão

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

In [4]:
```
clf1 = DecisionTreeClassifier()

clf1 = clf1.fit(iris.XA+iris.XB, iris.YA+iris.YB)

YC_pred = clf1.predict(iris.XC)
```

Métricas

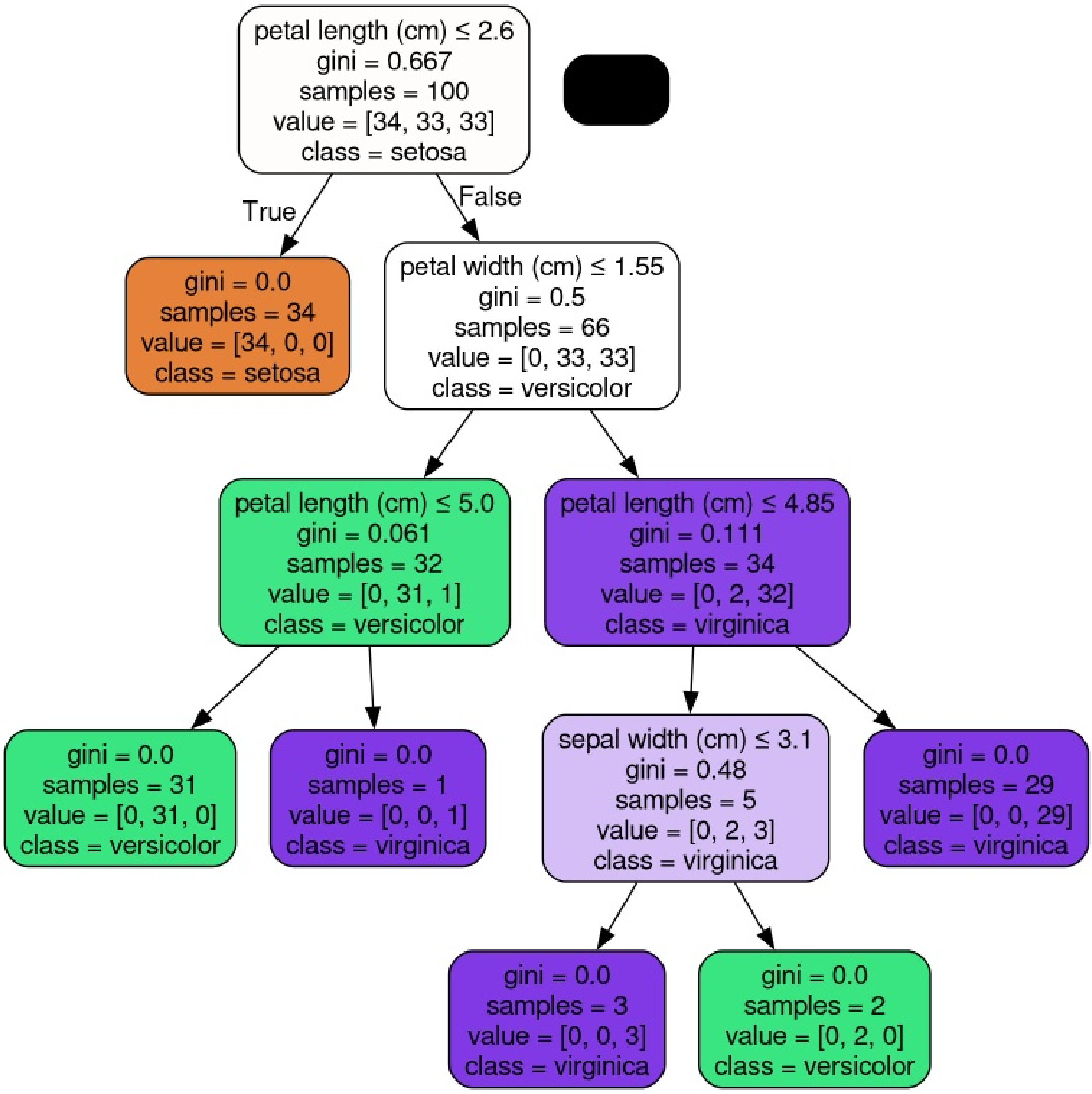
In [5]:
```
c1 = Classificacao(iris.YC, YC_pred)

c1.print_metrics()
```

```
|____________________________________________________|
|---------------------Acuracia--------------------|
|                        0.94                        |
|____________________________________________________|

|____________________________________________________|
|------------------Sensitividade------------------|
|   Setosa    | Versicolor | Virginica  |    Media    |
| 1.00000000 | 0.88235294 | 0.94117647 | 0.94117647 |
|____________________________________________________|

|____________________________________________________|
|------------------Especificidade------------------|
|   Setosa    | Versicolor | Virginica  |    Media    |
| 1.00000000 | 0.96969697 | 0.93939394 | 0.96969697 |
|____________________________________________________|

|____________________________________________________|
|--------------------Precisao---------------------|
|   Setosa    | Versicolor | Virginica  |    Media    |
| 1.00000000 | 0.93750000 | 0.88888889 | 0.94212963 |
|____________________________________________________|
```

Estrutura da Árvore

In [6]:
```
dot_data = StringIO()
export_graphviz(clf1, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names = iris.indice,class_names=iris.metas)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Out[6]:



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

In [7]:
```
clf2 = DecisionTreeClassifier()

clf2 = clf2.fit(iris.XA+iris.XC, iris.YA+iris.YC)

YB_pred = clf2.predict(iris.XB)
```

Métricas

In [8]:
```
c2 = Classificacao(iris.YB, YB_pred)

c2.print_metrics()
```

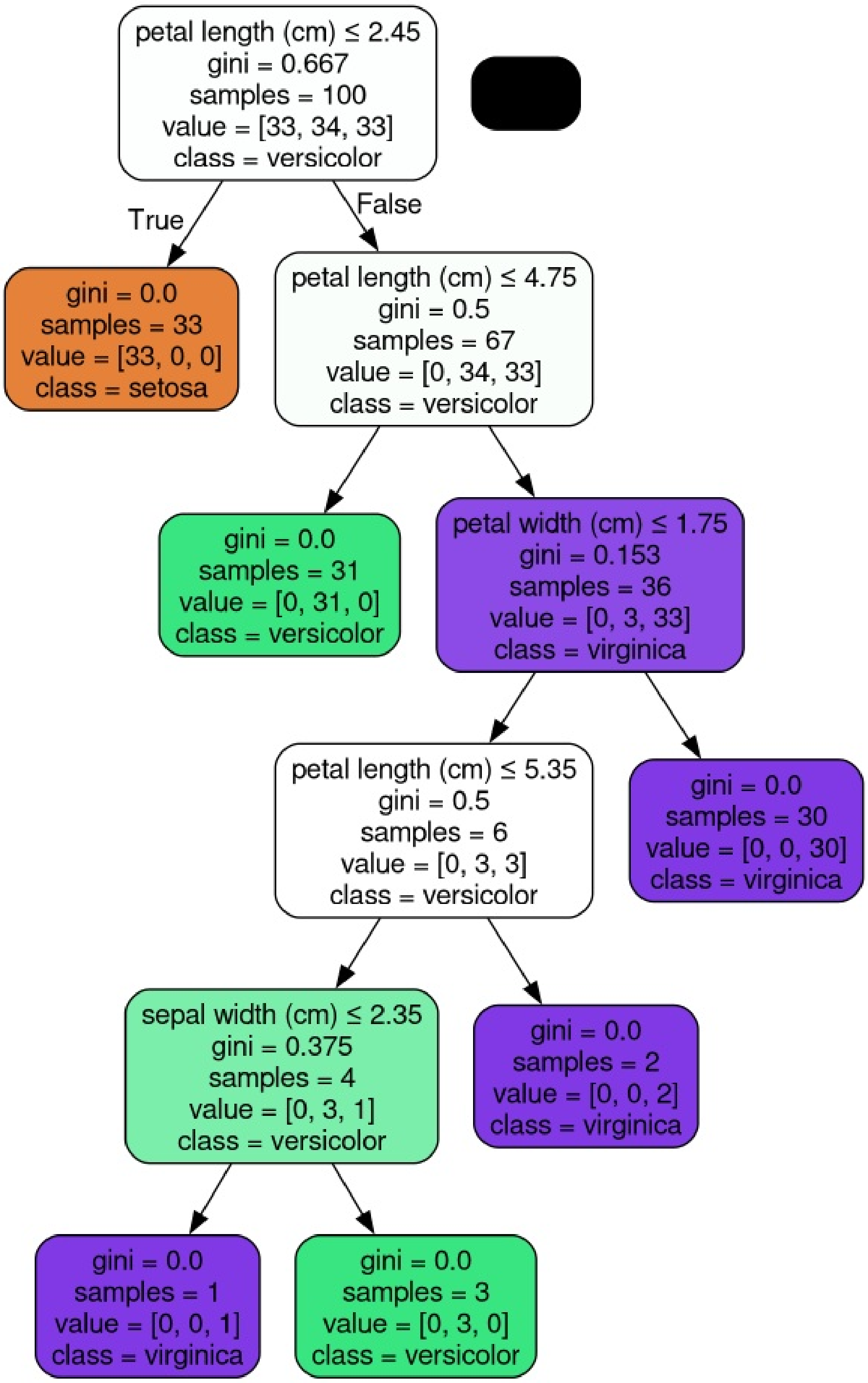
```
 ____________________________________________________
|                                                    |
|----------------------Acuracia---------------------|
|                        0.94                        |
|____________________________________________________|

 ____________________________________________________
|                                                    |
|-------------------Sensitividade------------------|
|   Setosa    | Versicolor | Virginica  |   Media    |
| 1.00000000 | 0.93750000 | 0.88235294 | 0.93995098 |
|____________________________________________________|

 ____________________________________________________
|                                                    |
|-------------------Especificidade-------------------|
|   Setosa    | Versicolor | Virginica  |   Media    |
| 1.00000000 | 0.94117647 | 0.96969697 | 0.97029115 |
|____________________________________________________|

 ____________________________________________________
|                                                    |
|---------------------Precisao---------------------|
|   Setosa    | Versicolor | Virginica  |   Media    |
| 1.00000000 | 0.88235294 | 0.93750000 | 0.93995098 |
|____________________________________________________|
```

Estrutura da Árvore

In [9]:

```python
dot_data = StringIO()
export_graphviz(clf2, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names = iris.indice,class_names=iris.metas)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Out[9]:



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

In [10]:
```
clf3 = DecisionTreeClassifier()

clf3 = clf3.fit(iris.XC+iris.XB, iris.YC+iris.YB)

YA_pred = clf3.predict(iris.XA)
```

Métricas

In [11]:
```
c3 = Classificacao(iris.YA, YA_pred)

c3.print_metrics()
```

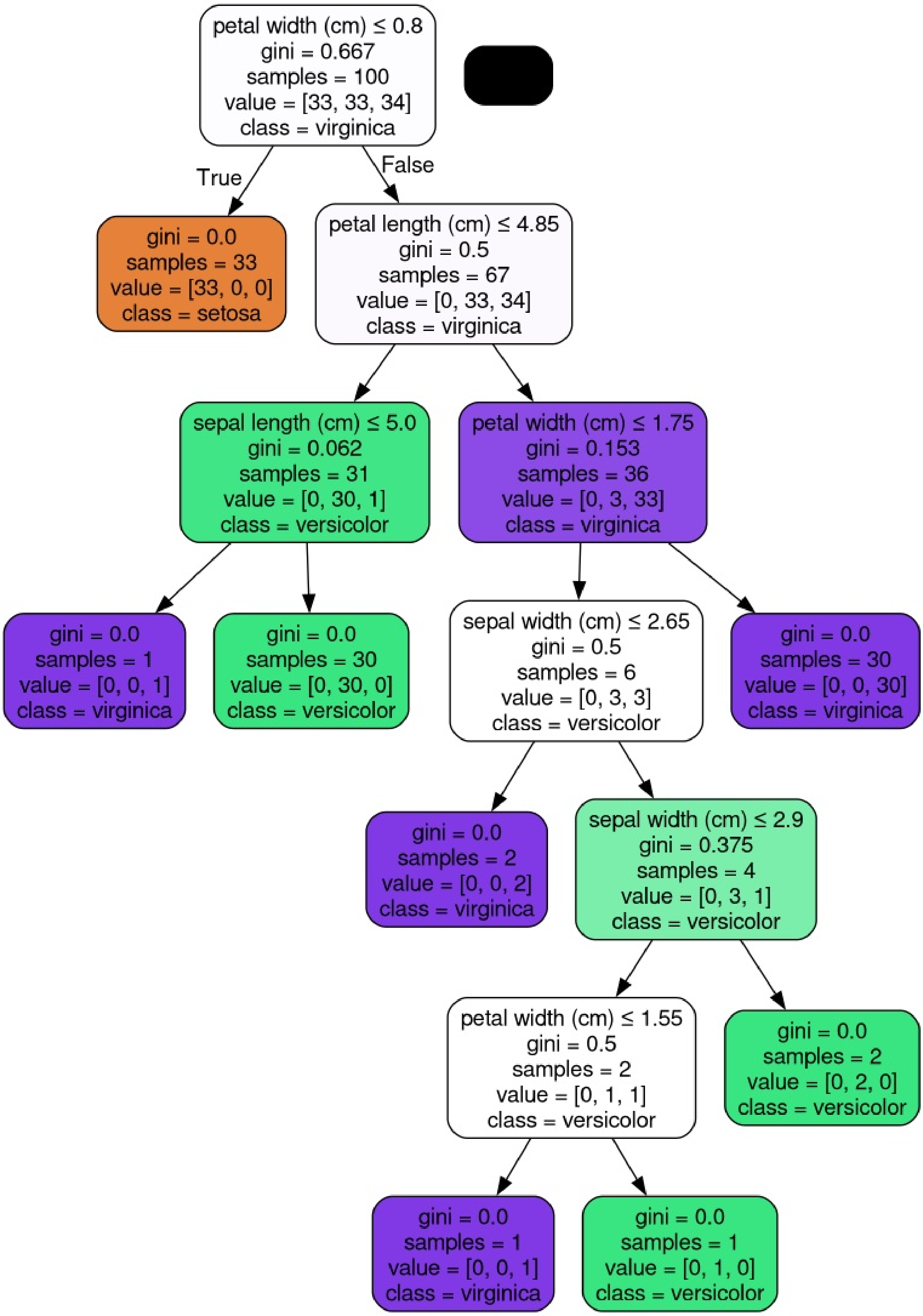
```
|________________________________________________________|
|----------------------Acuracia---------------------|
|                         0.86                       |
|________________________________________________________|

|________________________________________________________|
|-------------------Sensitividade------------------|
|   Setosa   | Versicolor | Virginica  |   Media    |
| 1.00000000 | 0.76470588 | 0.81250000 | 0.85906863 |
|________________________________________________________|

|________________________________________________________|
|-------------------Especificidade-------------------|
|   Setosa   | Versicolor | Virginica  |   Media    |
| 1.00000000 | 0.90909091 | 0.88235294 | 0.93048128 |
|________________________________________________________|

|________________________________________________________|
|---------------------Precisao---------------------|
|   Setosa   | Versicolor | Virginica  |   Media    |
| 1.00000000 | 0.81250000 | 0.76470588 | 0.85906863 |
|________________________________________________________|
```

Estrutura da Árvore

In [12]:

```
dot_data = StringIO()
export_graphviz(clf3, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names = iris.indice,class_names=iris.metas)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Out[12]:



K-Nearest Neighbours

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

In [13]:
```
knn1 = KNeighborsClassifier(n_neighbors=9)

knn1.fit(iris.XA+iris.XB, iris.YA+iris.YB)
YC_pred = knn1.predict(iris.XC)
```

Métricas

In [14]:
```
c4 = Classificacao(iris.YC, YC_pred)
```

```
c4.print_metrics()
```

```
|_____________________________________________________|
|---------------------Acuracia--------------------|
|                       0.96                      |
|_________________________________________________|

|_________________________________________________|
|------------------Sensitividade------------------|
|   Setosa   | Versicolor | Virginica  |   Media    |
| 1.00000000 | 0.88235294 | 1.00000000 | 0.96078431 |
|_________________________________________________|

|_________________________________________________|
|------------------Especificidade------------------|
|   Setosa   | Versicolor | Virginica  |   Media    |
| 1.00000000 | 1.00000000 | 0.93939394 | 0.97979798 |
|_________________________________________________|

|_________________________________________________|
|--------------------Precisao---------------------|
|   Setosa   | Versicolor | Virginica  |   Media    |
| 1.00000000 | 1.00000000 | 0.89473684 | 0.96491228 |
|_________________________________________________|
```

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

In [15]:
```
knn2 = KNeighborsClassifier(n_neighbors=9)

knn2.fit(iris.XA+iris.XC, iris.YA+iris.YC)

YB_pred = knn2.predict(iris.XB)
```

Métricas

In [16]:
```
c5 = Classificacao(iris.YB, YB_pred)

c5.print_metrics()
```

```
|_________________________________________________|
|---------------------Acuracia--------------------|
|                       0.98                      |
|_________________________________________________|

|_________________________________________________|
|------------------Sensitividade------------------|
|   Setosa   | Versicolor | Virginica  |   Media    |
| 1.00000000 | 1.00000000 | 0.94117647 | 0.98039216 |
|_________________________________________________|

|_________________________________________________|
|------------------Especificidade------------------|
|   Setosa   | Versicolor | Virginica  |   Media    |
| 1.00000000 | 0.97058824 | 1.00000000 | 0.99019608 |
|_________________________________________________|

|_________________________________________________|
|--------------------Precisao---------------------|
|   Setosa   | Versicolor | Virginica  |   Media    |
| 1.00000000 | 0.94117647 | 1.00000000 | 0.98039216 |
|_________________________________________________|
```

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

In [17]:
```
knn3 = KNeighborsClassifier(n_neighbors=9)

knn3.fit(iris.XB+iris.XC, iris.YB+iris.YC)

YA_pred = knn3.predict(iris.XA)
```

Métricas

In [18]:
```
c6 = Classificacao(iris.YA, YA_pred)
```

```
c6.print_metrics()
```

```
|_______________________________________________________|
|----------------------Acuracia---------------------|
|                        1.0                         |
|_______________________________________________________|

|_______________________________________________________|
|-------------------Sensitividade------------------|
|   Setosa    | Versicolor | Virginica  |    Media    |
| 1.00000000 | 1.00000000 | 1.00000000 | 1.00000000 |
|_______________________________________________________|

|_______________________________________________________|
|-------------------Especificidade------------------|
|   Setosa    | Versicolor | Virginica  |    Media    |
| 1.00000000 | 1.00000000 | 1.00000000 | 1.00000000 |
|_______________________________________________________|

|_______________________________________________________|
|---------------------Precisao---------------------|
|   Setosa    | Versicolor | Virginica  |    Media    |
| 1.00000000 | 1.00000000 | 1.00000000 | 1.00000000 |
|_______________________________________________________|
```

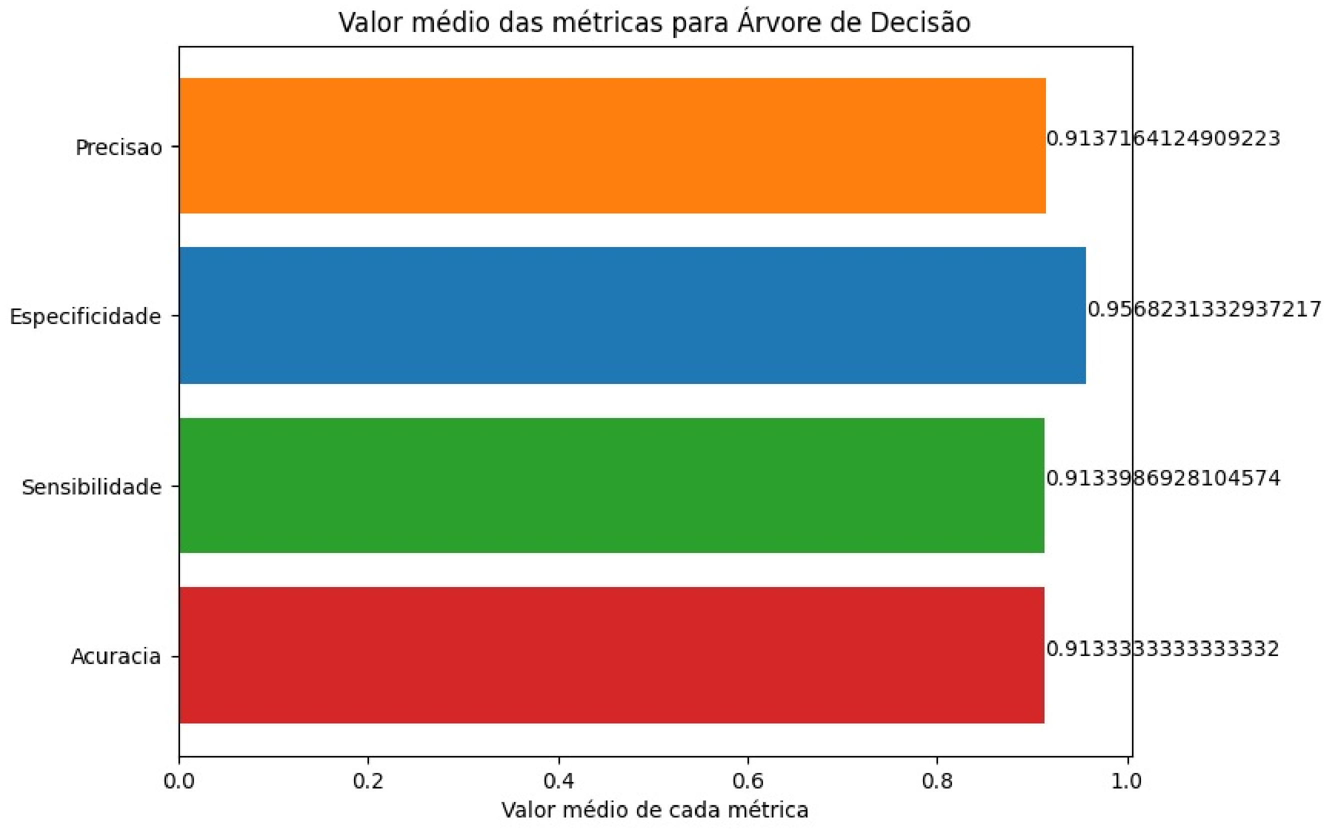
Considerações Finais

Valores médios das métricas

Árvore de Decisão

In [19]:

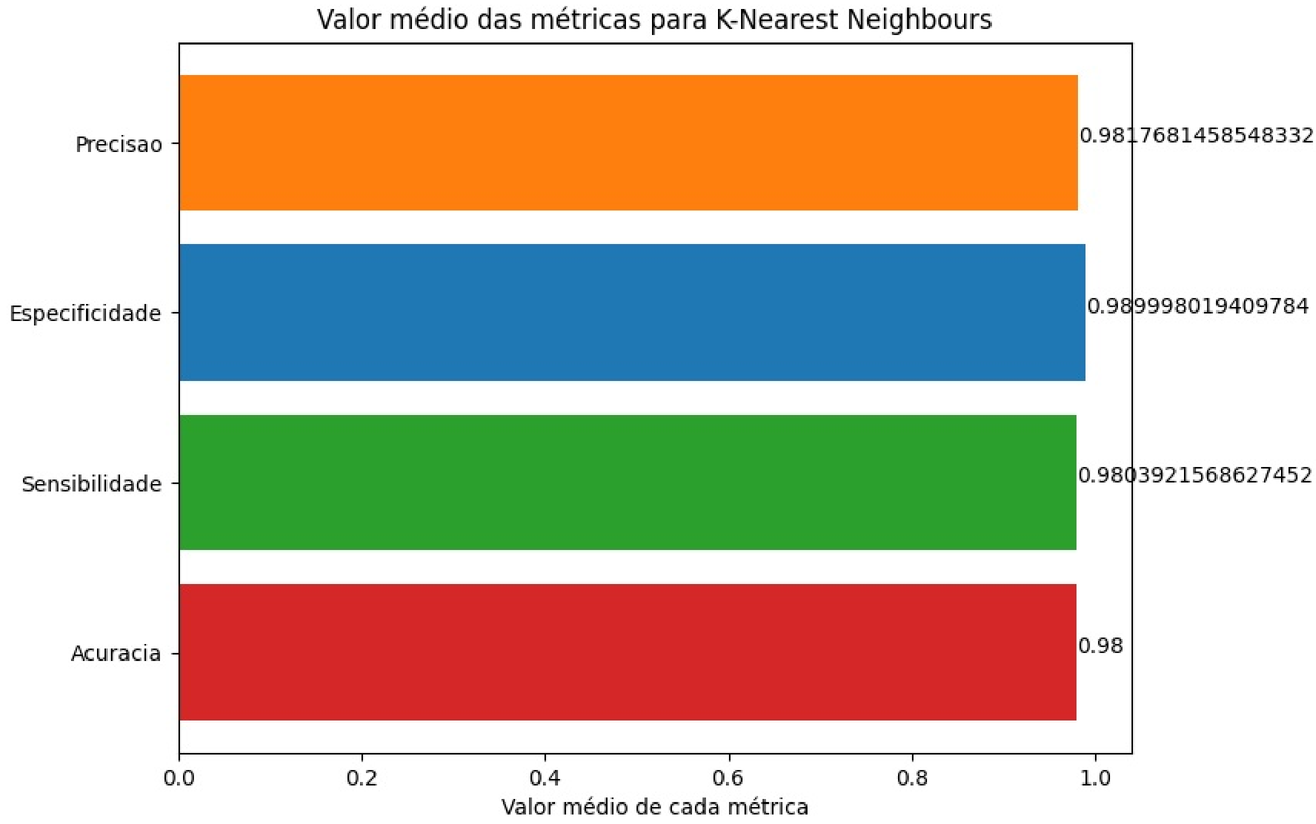
```
fig, ax = plt.subplots(figsize = (8,6))

metricas = ['Acuracia', 'Sensibilidade', 'Especificidade', 'Precisao']
mediasAD = [np.mean([c1.acur, c2.acur, c3.acur]),
          np.mean([np.mean(c1.sens), np.mean(c2.sens), np.mean(c3.sens)]),
          np.mean([np.mean(c1.espe), np.mean(c2.espe), np.mean(c3.espe)]),
          np.mean([np.mean(c1.prec), np.mean(c2.prec), np.mean(c3.prec)])]

bar_colors = ['tab:red', 'tab:green', 'tab:blue', 'tab:orange']

ax.barh(metricas, mediasAD, color=bar_colors)

for index, value in enumerate(mediasAD):
    plt.text(value, index,
             str(value))

ax.set_xlabel('Valor médio de cada métrica')
ax.set_title('Valor médio das métricas para Árvore de Decisão')

plt.show()
```



K-Nearest Neighbours

In [20]:

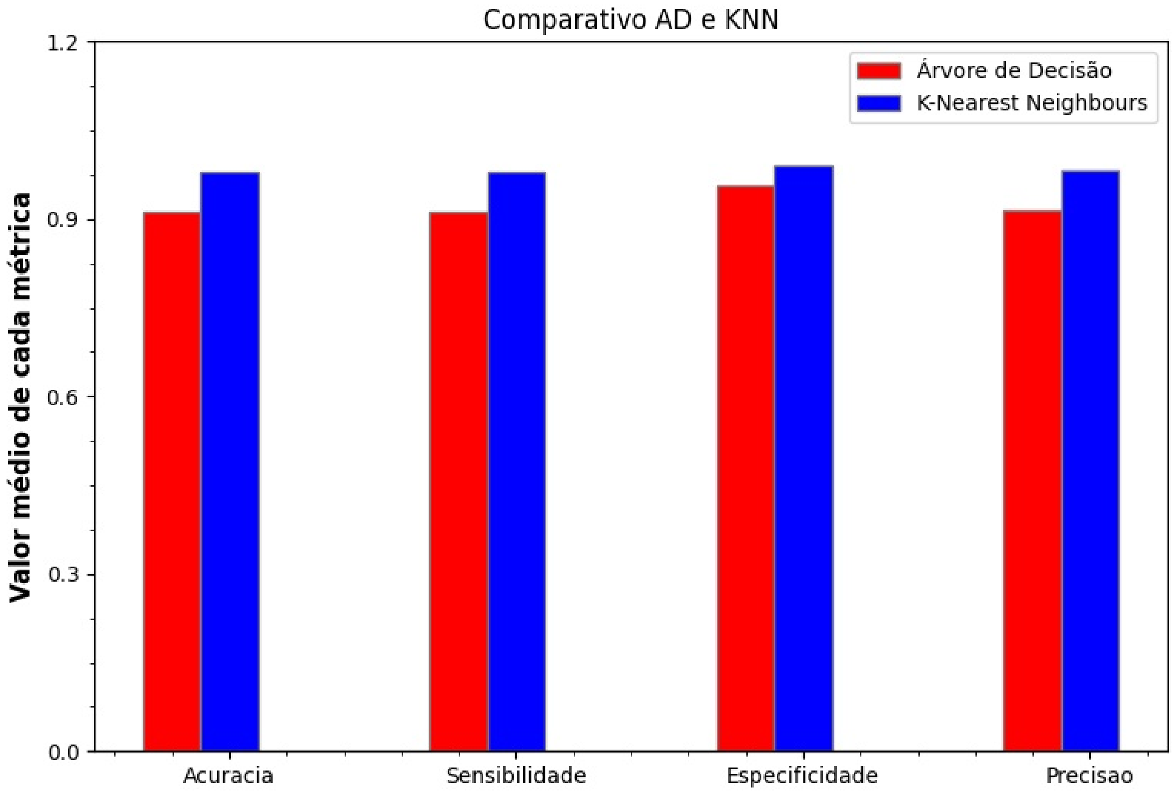
```
fig, ax = plt.subplots(figsize = (8,6))

metricas = ['Acuracia', 'Sensibilidade', 'Especificidade', 'Precisao']
mediasKNN = [np.mean([c4.acur, c5.acur, c6.acur]),
          np.mean([np.mean(c4.sens), np.mean(c5.sens), np.mean(c6.sens)]),
          np.mean([np.mean(c4.espe), np.mean(c5.espe), np.mean(c6.espe)]),
          np.mean([np.mean(c4.prec), np.mean(c5.prec), np.mean(c6.prec)])]

bar_colors = ['tab:red', 'tab:green', 'tab:blue', 'tab:orange']

ax.barh(metricas, mediasKNN, color=bar_colors)

for index, value in enumerate(mediasKNN):
    plt.text(value, index,
             str(value))

ax.set_xlabel('Valor médio de cada métrica')
ax.set_title('Valor médio das métricas para K-Nearest Neighbours')

plt.show()
```



Comparativo AD e KNN

In [21]:

```
# set width of bar
barWidth = 0.20
fig = plt.subplots(figsize =(9, 6))

# Set position of bar on X axis
br1 = np.arange(4)
br2 = [x + barWidth for x in br1]

# Make the plot
plt.bar(br1, mediasAD, color ='r', width = barWidth,
        edgecolor ='grey', label ='Árvore de Decisão')
plt.bar(br2, mediasKNN, color ='b', width = barWidth,
        edgecolor ='grey', label ='K-Nearest Neighbours')

plt.minorticks_on()

plt.ylabel('Valor médio de cada métrica', fontweight ='bold', fontsize = 12)
plt.xticks([r + barWidth for r in range(4)],
        ['Acuracia', 'Sensibilidade', 'Especificidade', 'Precisao'])
plt.yticks([0.0, 0.3, 0.6, 0.9, 1.2])

plt.title('Comparativo AD e KNN')

plt.legend()
plt.show()
```



É possível notar que, para a classificação realizada, a técnica *K-Nearest Neighbours* apresentou resultados médios um pouco superiores que a técnica *Árvore de Decisão* para todas as quatro métricas observadas (Acurácia, Sensibilidade, Especificidade e Precisão).