DecisionTreesMAC0459

November 18, 2020

1 Classificação e regressão usando árvores de decisão

1.0.1 Um classificador, ou regressor, muito fácil de interpretar e implementar é a árvore de decisão (DT - Decision Tree). Neste notebook você vai aprender, ou relembrar, o conceito básico de uma árvore de decisão, como ela é induzida e como aplicá-la.

Em Computação, uma árvore é uma estrutura de dados, ou uma forma de estruturar dados, que tem um nó raiz, galhos e nós folhas, como uma árvore de verdade. A forma de representar uma árvore em Computação é desenhando-a de cabeça para baixo, como ilustrado na figura abaixo, onde apresentamos a estrutura organizacional da universidade. Nela, a reitoria é a raiz e as pró-reitorias são as folhas.

[2]:
```
from treelib import Node,Tree

arvore = Tree()
arvore.create_node("Reitoria", "reitor")  # nó raiz
arvore.create_node("Pró-reitoria de Graduação", "prg", parent="reitor")
arvore.create_node("Pró-reitoria de Pós-Graduação", "prpg", parent="reitor")
arvore.create_node("Pró-reitoria de Pesquisa", "prp", parent="reitor")
arvore.create_node("Pró-reitoria de Cultura e Extensão Universitária",␣
 ↪"proceu", parent="reitor")
arvore.show()
```

```
Reitoria
   Pró-reitoria de Cultura e Extensão Universitária
   Pró-reitoria de Graduação
   Pró-reitoria de Pesquisa
   Pró-reitoria de Pós-Graduação
```

1.0.2 Árvore binária

Árvore binária é um tipo de árvore em que cada nó, que não é folha, pode ter apenas dois "galhos" para outros nós, ou para folhas. A figura abaixo ilustra uma árvore binária. Uma grande vantagem de armazenar elementos em uma árvore é a rapidez com que elas podem ser buscadas depois. Por exemplo, para procurar pelos mamíferos a partir da raiz, perguntamos na raiz se o vertebrado é de Sangue frio. Se for, seguimos para o nó dos vertebrados de sangue frio. Se não for, seguimos para outro nó (pois há no máximo dois nós), que é o caso. Estando no nó dos vertebrados de sangue

quente, perguntamos se o vertebrado é da classe das Aves e Peixes. Como não é, seguimos para o nó dos mamíferos e pronto.

[3]:
```
arvore = Tree()
arvore.create_node("Vertebrados", "vert")  # nó raiz
arvore.create_node("Sangue quente", "hotb", parent="vert")
arvore.create_node("Sangue frio", "coldb", parent="vert")
arvore.create_node("Mamíferos", "mamm", parent="hotb")
arvore.create_node("Aves/Peixes", "birdfish", parent="hotb")
arvore.create_node("Anfíbios", "anf", parent="coldb")
arvore.create_node("Répteis", "rept", parent="coldb")
arvore.show()
```

```
Vertebrados
  Sangue frio
      Anfíbios
      Répteis
  Sangue quente
       Aves/Peixes
       Mamíferos
```

1.0.3 Quero saber mais sobre árvores

Caso você queira saber mais sobre essa estrutura de dados, procure nas páginas e aulas do prof. Dr. Paulo Feofiloff:

https://www.ime.usp.br/~pf/

Caso você queira saber mais sobre esta biblioteca de árvores do Python, a treelib, a documentação está aqui:

https://treelib.readthedocs.io/en/latest/

1.0.4 Árvore de decisão binária

Árvore de decisão binária é uma árvore binária onde cada nó, que não é folha, tem associada uma "pergunta", ou um valor, sobre um determinado atributo/medida do objeto que queremos classificar. Por exemplo, na árvore binária do exemplo anterior, a primeira pergunta estaria associada ao sangue (o sangue é frio?).

Em outras palavras, usar uma árvore de decisão para classificar um objeto é muito simples, basta percorrermos a árvore da raiz para as folhas, respondendo às perguntas associadas a cada nó interno (ou nó que não é folha) da árvore.

Por outro lado, induzir (que é a palavra que usasse para a tarefa de construir uma árvore binária é o problema real que os algoritmos de aprendizado de máquina precisam resolver.

Para ilustrar um algoritmo de indução de árvores de decisão, vamos primeiramente apresentar o conjunto de dados que vamos usar.

Comecemos apresentando um conjunto de dados (dataset) muito conhecido da comunidade, o IRIS.

[5]:
```
import os
import pydotplus

from sklearn.datasets import load_iris
from IPython.display import Image
```

IRIS é um dataset com 150 medidas de comprimento e largura de pétalas (petal) e sépalas (sepal). A sépala é um tipo de folha que protege a pétala. A figura abaixo mostra um exemplo de uma flor do tipo iris virgínica (fonte: Wikipedia).

[6]:
```
from IPython.display import Image
from IPython.core.display import HTML
Image(url= "https://upload.wikimedia.org/wikipedia/commons/thumb/f/f8/
 ↪Iris_virginica_2.jpg/240px-Iris_virginica_2.jpg")
```

[6]:
```
<IPython.core.display.Image object>
```

A figura abaixo mostra um exemplo de uma flor do tipo iris setosa (fonte: Wikipedia).

[7]:
```
Image(url= "https://upload.wikimedia.org/wikipedia/commons/thumb/5/56/
 ↪Kosaciec_szczecinkowaty_Iris_setosa.jpg/
 ↪180px-Kosaciec_szczecinkowaty_Iris_setosa.jpg")
```

[7]:
```
<IPython.core.display.Image object>
```

A figura abaixo mostra um exemplo de uma flor do tipo iris versicolor (fonte: Wikipedia).

[8]:
```
Image(url= "https://upload.wikimedia.org/wikipedia/commons/thumb/2/27/
 ↪Blue_Flag%2C_Ottawa.jpg/240px-Blue_Flag%2C_Ottawa.jpg")
```

[8]:
```
<IPython.core.display.Image object>
```

A figura abaixo mostra a diferença entre a sépala e a pétala em uma flor (fonte: Wikipedia). Note que, na flor iris, a sépala é, em geral, maior que a pétala e parece-se muito com uma pétala.

[9]:
```
Image(url= "https://upload.wikimedia.org/wikipedia/commons/thumb/7/78/
 ↪Petal-sepal.jpg/226px-Petal-sepal.jpg")
```

[9]:
```
<IPython.core.display.Image object>
```

[22]:
```
# O conjunto iris pode facilmente ser carregado usando o seguinte comando.

iris = load_iris()
```

[11]:
```
# Estes são as primeiras dez linhas da matriz de dados de quatro colunas do␣
 ↪iris.

print(iris.data[1:10])
```

```
[[4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]]
```

[12]:
```
# Estes são as primeiras dez linhas da matriz de rótulos do iris.

print(iris.target[1:10])
```

```
[0 0 0 0 0 0 0 0 0]
```

[13]:
```
# Estes são os nomes dos rótulos (que são os nomes dos tipos das flores).

print(iris.target_names)
```

```
['setosa' 'versicolor' 'virginica']
```

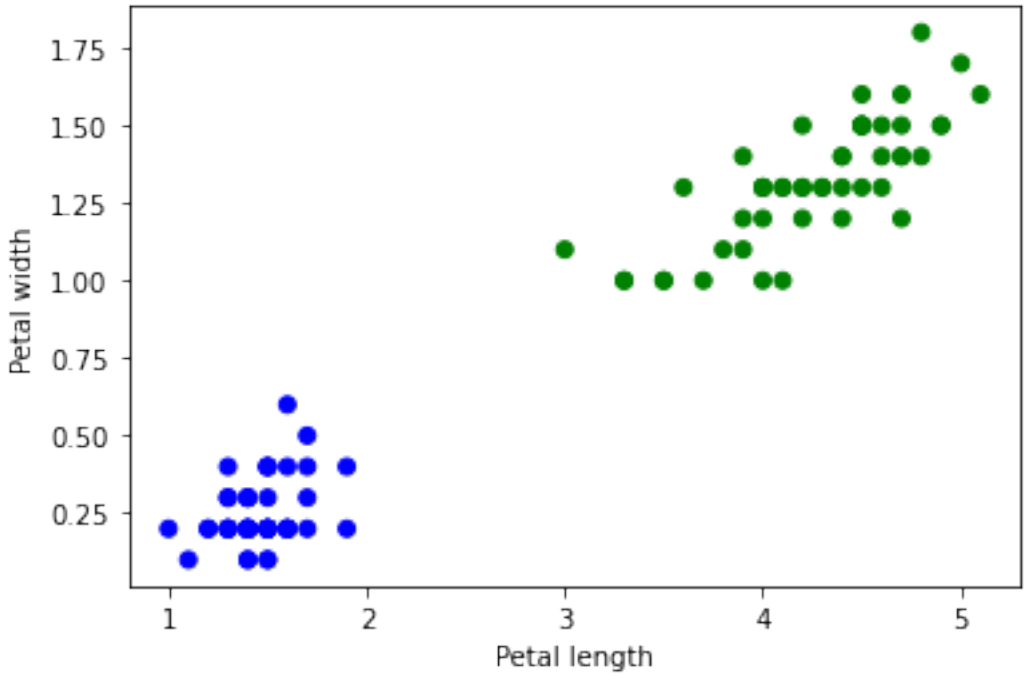
[14]:
```
# Estes são os nomes dos atributos (e as respectivas unidades de medida) de␣
↪cada uma das colunas do iris.

print(iris.feature_names)
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width
(cm)']
```

Para facilitar a explicação, de um algoritmo de indução de árvores de decisão, vamos começar com o problema de separar apenas duas classes de iris, setosa e versicolor. Para isso, vamos tomar apenas os valores do conjunto de dados cujos targets são 0 e 1.

[15]:
```
# Criamos um conjunto X com apenas as linhas correspondentes às classes setosa
# e versicolor e um conjunto Y com os respectivos rótulos.

X = iris.data[(iris.target==0)|(iris.target==1),:]
Y = iris.target[(iris.target==0)|(iris.target==1)]

# Vamos fazer um gráfico do comprimento vs largura das pétalas
import matplotlib as mpl
import matplotlib.pyplot as plt

colors = ['blue','green']
plt.scatter(X[:,2],X[:,3],c=Y,cmap=mpl.colors.ListedColormap(colors))

plt.xlabel('Petal length')
```

```
plt.ylabel('Petal width')
plt.show()
```



É fácil ver que basta olhar para o comprimento da pétala para separar as duas classes com apenas uma pergunta que poderia ser (escolheremos um valor arbitrário para o comprimento da pétala): If X[i,2] $>$ 2.5: classe = 0 else: classe = 1

Uma outra forma seria examinar a largura da pétala e escolher o valor, digamos 0.75:

Em ambos os casos temos uma forma heurística de induzir uma árvore de decisão e que resultam em uma árvore com apenas um nó raiz e nesse nó temos uma pergunta sobre apenas uma característica que é o comprimento, ou a largura, da pétala. Todos os dados do conjunto seriam separados perfeitamente em dois nós folha.

A forma heurística não é interessante e precisamos de uma outra estratégia que induza a árvore sem necessidade de auxílio humano, ou gráfico.

Algoritmo para indução de uma árvore de decisão binária Existem muitas formas de induzir uma árvore de decisão e a forma mais intuitiva é um algoritmo recursivo que, a cada nó, que não um nó folha, particiona (em duas partes/subconjuntos) o conjunto de treinamento de acordo com algum critério que minimiza a impureza de cada subconjunto.

Um algoritmo recursivo é formado por duas partes importantes: a base e o passo da recursão. A base da recursão é fundamental, senão o algoritmo não pára. O passo da recursão faz as vezes do laço ("loop") num algoritmo não recursivo.

Para facilitar o entendimento, vamos chamar de X o conjunto de objetos $x \in \mathcal{R}^d$ que desejamos

classificar. A cada $x \in \mathcal{X}$, está associado um rótulo $y \in Y$, $Y \in \mathcal{Z}$.

Vamos denominar por $x_i \in R$ a cada uma das características (medidas) de x na dimensão i. Dessa forma, $x = \{x_1, x_2, \ldots, x_d\}$.

Desta forma, dado um conjunto de objetos $X \in R^4$, como no caso do exemplo do iris reduzido a duas classes acima, podemos particionar X em dois subconjuntos:

$X_R = \{x \in X | x_2 <= 2.5\}$ e $X_L = \{x \in X | x_2 > 2.5\}$

Neste notebook trataremos apenas das árvores binárias transversais, isto é, cada nó está associado a uma pergunta sobre apenas uma característica.

Medida de impureza Um outro elemento fundamental no algoritmos de indução é a medida de impureza que será usada para decidir que valor será usado como limiar para o particionamento do conjunto. Essa medida tem que refletir, de alguma forma, a mistura de rótulos num determinado subconjunto. O objetivo, então, é achar uma característica x_i e um valor para ser aplicado a x_i que particione o conjunto em dois subconjuntos X_R e X_L, de forma a tornar os subconjuntos mais puros e ter um ganho na medida de impureza. Matematicamente, se denotarmos por I essa medida, queremos otimizá-la de tal forma que:

$\Delta I(X) = I(X) - P_L I(X_L) - P_R I(X_R) = I(X) - P_L I(X_L) - (1 - P_L) I(X_R)$

Onde P_L (P_R) é a proporção de elementos de X que estão no conjunto X_L (X_R). Note que $P_R = 1 - P_L$.

Quanto maior o $\Delta I(X)$, melhor o particionamento. Pense sobre isso!

Algoritmo DT em alto nível de abstração entrada: X, # conjunto de dados Y, # conjunto de rótulos I, # medida de impureza

saída: Árvore de decisão

DT(X,Y,I):

```
cria novo node # nó
if X é puro:
    retorna o node (nó folha) que representa X
else:
    encontre a partição X = X_L \cup X_R que dê o maior ganho de pureza
    node.left = DT(X_L)
    node.right = DT(X_R)
retorna node
```

A entropia como medida de impureza Antes de demonstrar o algoritmo, vamos apresentar uma medida de impureza e fazer a conta que o algoritmo faria na mão.

Entropia é uma medida cuja história é, por si, muito interessante. No contexto de Teoria da Informação, ela foi inicialmente proposta por Claude Shanon no artigo "A Mathematical Theory of Communication":

https://ieeexplore.ieee.org/document/6773024

Em teoria da informação, há duas atoras importantes: a fonte e a receptora da mensagem. A ideia de Shannon era criar uma medida de informação que refletisse o quão surpreendente era uma mensagem para a receptora. Quanto mais surpreendente, ou menos provável, for a mensagem, mais informação ela carrega. Por outro lado, quanto mais provável for a mensagem, menos surpreendende ela será e menos informação ela carrega. Matematicamente, se denotarmos por H a medida de informação e por p a medida de probabilidade (note que estamos tratando de eventos discretos), matematicamente teríamos:

$H(m) \propto \frac{1}{p(m)}$

Uma outra propriedade que Shannon queria dessa medida é que, se a fonte mandasse duas mensagens diferentes, digamos m_1 e m_2, a medida da informação das duas mensagens conjuntas, isto é, $m_1.m_2$ fosse a soma das medidas de informação m_1 e m_2. Matematicamente:

$H(m_1.m_2) \propto H(m_1) + H(m_2)$

Como é sabido, o logaritmo é o funcional que leva uma função produto a uma função soma, assim, a medida de entropia de um conjunto $M = \{m_1, m_2, \ldots, m_n\}$ de mensagens, usualmente denotada por H, é definida como uma média (por isso o produto pela probabilidade) da medida de informação de cada mensagem do conjunto M:

$$H(M) = \sum_{i=1}^{n} -p(m_i) \log p(m_i)$$

Uma propriedade importante da entropia é que ela é um valor entre 0 e 1.

Exemplo usando os dados Vamos fazer a conta da entropia para três valores de largura de pétala e três de comprimento de pétala.

Para a largura da pétala, vamos escolher arbitrariamente os valores $0.4, 0.8$ e 1.2.

[23]:
```
from math import log2

print(X[:,3])

# Entropia do conjunto X[:,3]
H = -50/100 * log2(50/100) - 50/100 * log2(50/100)
print("Entropia do conjunto X[:,3] =", H)

# Vamos particionar X[:,3] em 0.4 :
index04 = (X[:,3]<=0.4)
print(index04)
nSetosa04 = sum(Y[index04]==0)
n04R = sum(index04)
nVersicolor04 = sum(Y[~index04])
n04L = sum(~index04)

print(Y[index04])
print(Y[~index04])
print("Número de flores iris setosa com largura de pétala menor, ou igual a 0.4␣
 ↪=", nSetosa04)
```

```
print("Número de flores com largura de pétala menor, ou igual a 0.4 =", n04R)
print("Número de flores iris versicolor com largura de pétala maior que 0.4=",␣
 ↪nVersicolor04)
print("Número de flores com largura de pétala maior que 0.4 =", n04L)

# A entropia do lado direito (valores menores, ou iguais a 0.4)
H04R = -nSetosa04/n04R * log2(nSetosa04/n04R)
print("H04R =" , H04R)

# A entropia do lado esquerdo (valores maiores que 0.4)
H04L = -(n04L-nVersicolor04)/n04L * log2((n04L-nVersicolor04)/n04L) -␣
 ↪nVersicolor04/n04L * log2(nVersicolor04/n04L)
print("H04L =" , H04L)

# O delta de entropia
deltaH04 = H - n04L/(n04L+n04R) * H04L - n04R/(n04L+n04R) * H04R
print("DeltaH04 =", deltaH04)
```

```
[0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 0.2 0.2 0.1 0.1 0.2 0.4 0.4 0.3
 0.3 0.3 0.2 0.4 0.2 0.5 0.2 0.2 0.4 0.2 0.2 0.2 0.2 0.4 0.1 0.2 0.2 0.2
 0.2 0.1 0.2 0.2 0.3 0.3 0.2 0.6 0.4 0.3 0.2 0.2 0.2 0.2 1.4 1.5 1.5 1.3
 1.5 1.3 1.6 1.  1.3 1.4 1.  1.5 1.  1.4 1.3 1.4 1.5 1.  1.5 1.1 1.8 1.3
 1.5 1.2 1.3 1.4 1.4 1.7 1.5 1.  1.1 1.  1.2 1.6 1.5 1.6 1.5 1.3 1.3 1.3
 1.2 1.4 1.2 1.  1.3 1.2 1.3 1.3 1.1 1.3]
100
Entropia do conjunto X[:,3] = 1.0
[ True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True False
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True False  True  True  True  True
  True  True False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
Número de flores iris setosa com largura de pétala menor, ou igual a 0.4 = 48
Número de flores com largura de pétala menor, ou igual a 0.4 = 48
Número de flores iris versicolor com largura de pétala maior que 0.4= 50
Número de flores com largura de pétala maior que 0.4 = 52
H04R = -0.0
H04L = 0.23519338181924143
DeltaH04 = 0.8776994414539945
```

[24]:
```
# Isto aqui pode ser deixado como exercício

# Vamos particionar X[:,3] em 0.8 :
index08 = (X[:,3]<=0.8)
print(index08)
nSetosa08 = sum(Y[index08]==0)
n08R = sum(index08)
nVersicolor08 = sum(Y[~index08])
n08L = sum(~index08)

print(Y[index08])
print(Y[~index08])
print("Número de flores iris setosa com largura de pétala menor, ou igual a 0.8␣
 ↪=", nSetosa08)
print("Número de flores com largura de pétala menor, ou igual a 0.8 =", n08R)
print("Número de flores iris versicolor com largura de pétala maior que 0.8=",␣
 ↪nVersicolor08)
print("Número de flores com largura de pétala maior que 0.8 =", n08L)

# A entropia do lado direito (valores menores, ou iguais a 0.8)
H08R = -nSetosa08/n08R * log2(nSetosa08/n08R)
print("H08R =" , H08R)

# A entropia do lado esquerdo (valores maiores que 0.8)
H08L = -nVersicolor08/n08L * log2(nVersicolor08/n08L)
print("H08L =" , H08L)

# O delta de entropia
deltaH08 = H - n08L/(n08L+n08R) * H08L - n08R/(n08L+n08R) * H08R
print("DeltaH08 =", deltaH08)
```

```
[ True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0]
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1]
Número de flores iris setosa com largura de pétala menor, ou igual a 0.8 = 50
Número de flores com largura de pétala menor, ou igual a 0.8 = 50
Número de flores iris versicolor com largura de pétala maior que 0.8= 50
```

```
Número de flores com largura de pétala maior que 0.8 = 50
H08R = -0.0
H08L = -0.0
DeltaH08 = 1.0
```

[25]:
```
# Vamos particionar X[:,3] em 1.2 :
index12 = (X[:,3]<=1.2)
print(index12)
nSetosa12 = sum(Y[index12]==0)
n12R = sum(index12)
nVersicolor12 = sum(Y[~index12])
n12L = sum(~index12)

print(Y[index12])
print(Y[~index12])
print("Número de flores iris setosa com largura de pétala menor, ou igual a 1.2␣
 ↪=", nSetosa12)
print("Número de flores com largura de pétala menor, ou igual a 1.2 =", n12R)
print("Número de flores iris versicolor com largura de pétala maior que 1.2=",␣
 ↪nVersicolor12)
print("Número de flores com largura de pétala maior que 1.2 =", n12L)

# A entropia do lado direito (valores menores, ou iguais a 1.2)
H12R = -nSetosa12/n12R * log2(nSetosa12/n12R) - (n12R - nSetosa12)/n12R *␣
 ↪log2((n12R - nSetosa12)/n12R)
print("H12R =" , H12R)

# A entropia do lado esquerdo (valores maiores que 1.2)
H12L = - nVersicolor12/n12L * log2(nVersicolor12/n12L)
print("H12L =" , H12L)

# O delta de entropia
deltaH12 = H - n12L/(n12L+n12R) * H12L - n12R/(n12L+n12R) * H12R
print("DeltaH12 =", deltaH12)
```

```
[ True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True False False False False False False False  True False False
  True False  True False False False False  True False  True False False
 False  True False False False False False  True  True  True  True False
 False False False False False False  True False  True  True False  True
 False False  True False]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

```
Número de flores iris setosa com largura de pétala menor, ou igual a 1.2 = 50
Número de flores com largura de pétala menor, ou igual a 1.2 = 65
Número de flores iris versicolor com largura de pétala maior que 1.2= 35
Número de flores com largura de pétala maior que 1.2 = 35
H12R = 0.7793498372920851
H12L = -0.0
DeltaH12 = 0.49342260576014474
```

Desta maneira, comparando os três deltas, o melhor particionamento dentre esses três seria o 0.8.

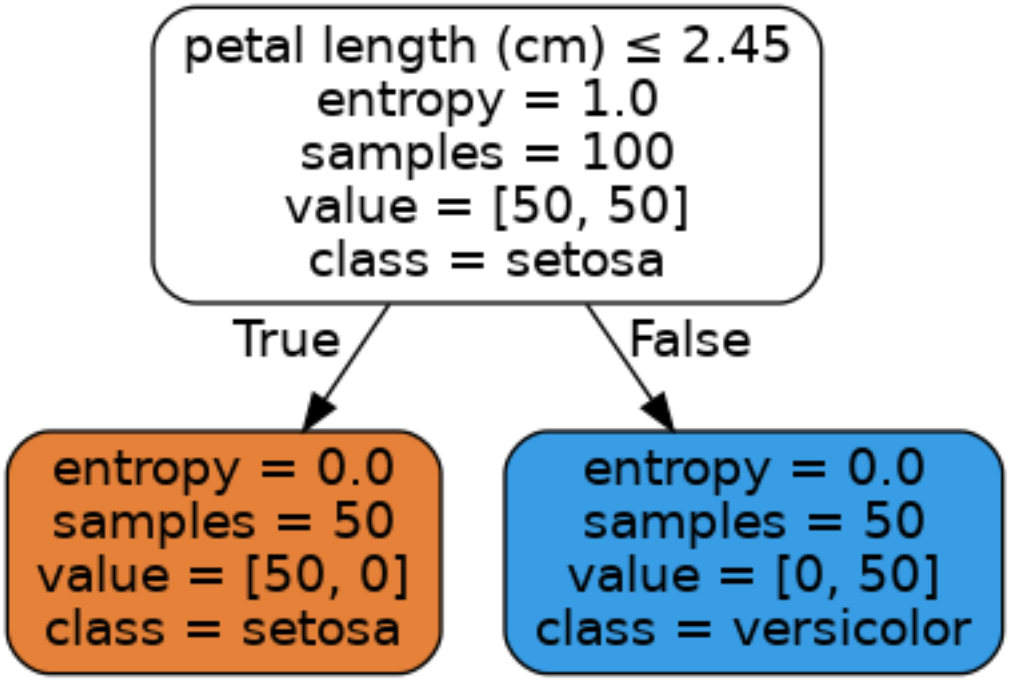
Exercício, faça o mesmo para o comprimento da pétala. Para isso, escolha "Insert" no menu e "Insert Cell Bellow". Depois, copie o conteúdo da célula anterior na célula criada e refaça as contas para o valor do comprimento da pétala.

1.0.5 Vamos agora usar o pacote sklearn e a classe tree para fazer o mesmo problema.

[27]:
```
# Importa a classe tree
from sklearn import tree
from sklearn.metrics import accuracy_score

# Cria um objeto decision tree com a medida de impuridade entropy
clf = tree.DecisionTreeClassifier("entropy")

# Ajusta o classificador sobre o conjunto de dados rotulados
clf.fit(X, Y)

# Calcula o erro do classificador nos dados de treinamento
Ypredicted = clf.predict(X)

print("Acurácia =",accuracy_score(Y,Ypredicted)*100," porcento, como esperado.")

# Apresenta a árvore
# Código copiado de um exemplo da biblioteca

dot_data = tree.export_graphviz(clf, out_file=None,
                                feature_names=iris.feature_names,
                                class_names=iris.target_names,
                                filled=True, rounded=True,
                                special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```

```
/usr/local/anaconda3/lib/python3.8/site-packages/sklearn/utils/validation.py:67:
FutureWarning: Pass criterion=entropy as keyword args. From version 0.25 passing
these as positional arguments will result in an error
  warnings.warn("Pass {} as keyword args. From version 0.25 "

Acurácia = 100.0  porcento, como esperado.
```

[27]:

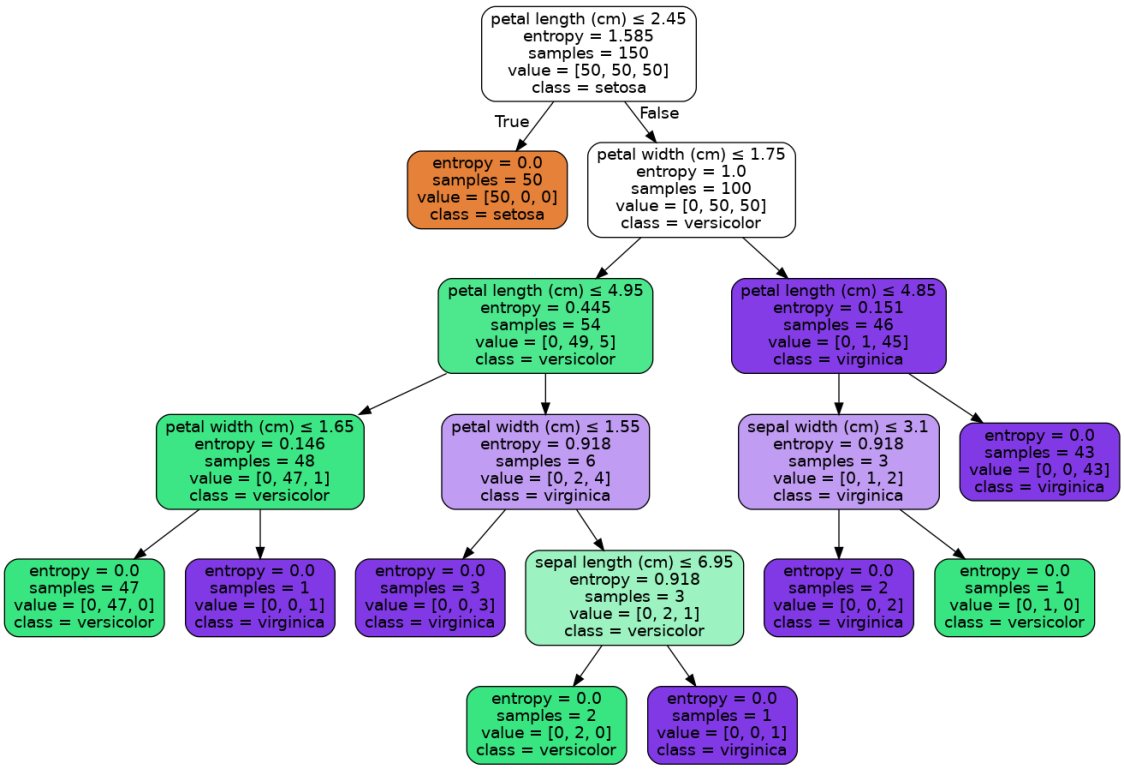


[28]:
```
# Vamos fazer o mesmo para o conjunto todo do Iris. Note como a árvore fica␣
 ↪mais profunda
# por conta da mistura de classes.

X = iris.data
Y = iris.target

# Cria um objeto decision tree com a medida de impuridade entropy
clfAll = tree.DecisionTreeClassifier("entropy")

# Ajusta o classificador sobre o conjunto de dados rotulados
clfAll.fit(X, Y)

# Apresenta a árvore
# Código copiado de um exemplo da biblioteca

dot_data = tree.export_graphviz(clfAll, out_file=None,
                         feature_names=iris.feature_names,
                         class_names=iris.target_names,
                         filled=True, rounded=True,
                         special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```

```
/usr/local/anaconda3/lib/python3.8/site-packages/sklearn/utils/validation.py:67:
```

```
FutureWarning: Pass criterion=entropy as keyword args. From version 0.25 passing
these as positional arguments will result in an error
  warnings.warn("Pass {} as keyword args. From version 0.25 "
```

[28]:



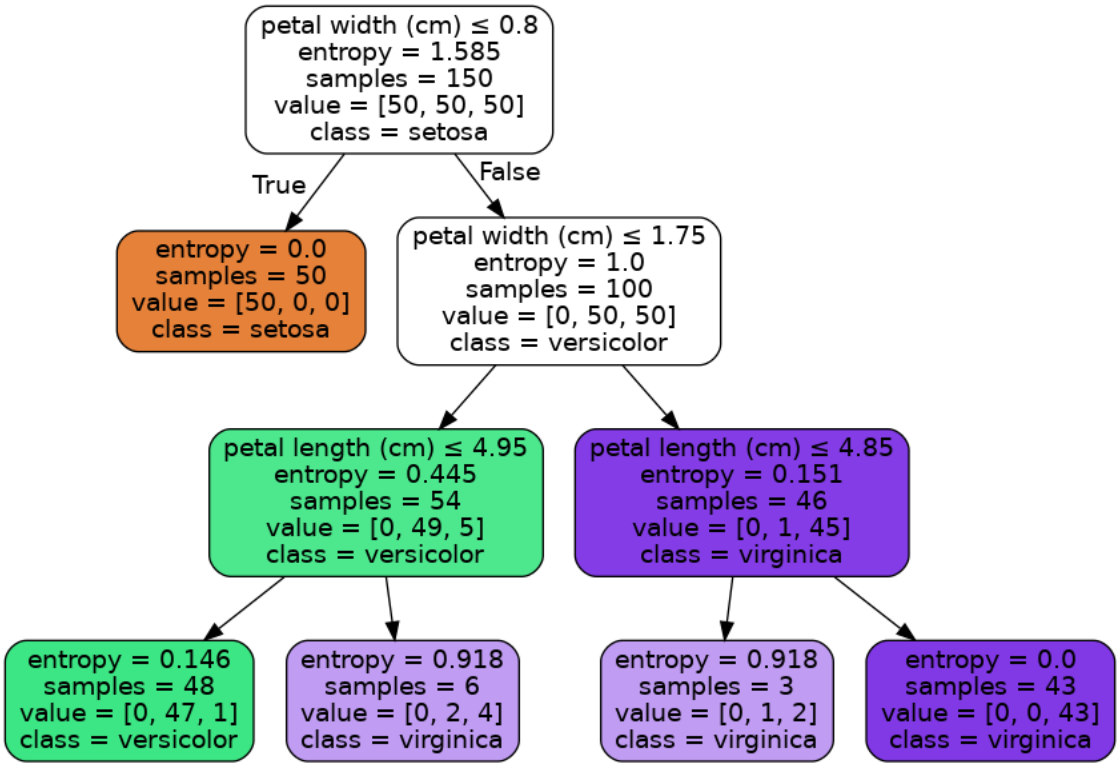
[30]:

```
# Vamos refazer o mesmo exemplo anterior, mas agora vamos limitar
# a profundidade da árvore e calcular o erro nos dados de treinamento.
# Esta é uma forma usual para regularizar o problema da indução da árvore.

X = iris.data
Y = iris.target

# Cria um objeto decision tree com a medida de impuridade entropy
clfAll = tree.DecisionTreeClassifier("entropy",max_depth=3)

# Ajusta o classificador sobre o conjunto de dados rotulados
clfAll.fit(X, Y)

# Calcula o erro do classificador nos dados de treinamento
Ypredicted = clfAll.predict(X)

print(Y,Ypredicted)
```

```
print("Acurácia =",accuracy_score(Y,Ypredicted)*100," porcento. Note que␣
 ↪diminuiu a acurácia.")

# Apresenta a árvore
# Código copiado de um exemplo da biblioteca

dot_data = tree.export_graphviz(clfAll, out_file=None,
                                feature_names=iris.feature_names,
                                class_names=iris.target_names,
                                filled=True, rounded=True,
                                special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2] [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1
 1 1 1 2 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 1 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
Acurácia = 97.33333333333334  porcento. Note que diminuiu a acurácia.

/usr/local/anaconda3/lib/python3.8/site-packages/sklearn/utils/validation.py:67:
FutureWarning: Pass criterion=entropy as keyword args. From version 0.25 passing
these as positional arguments will result in an error
  warnings.warn("Pass {} as keyword args. From version 0.25 "
```

[30]:

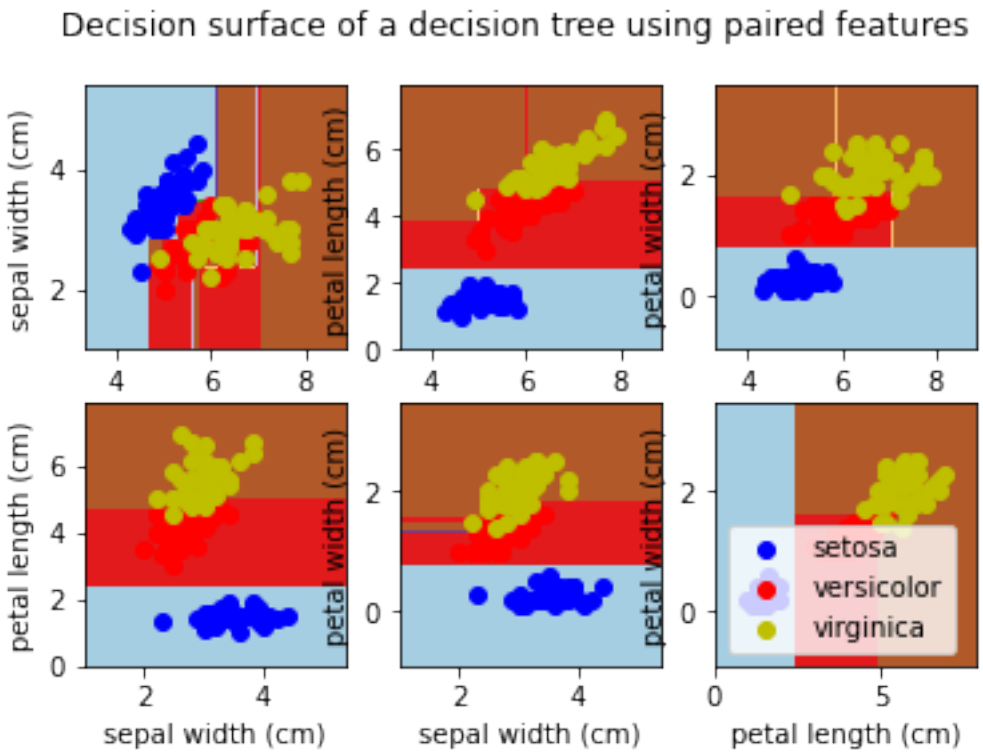


Exercício: Experimente refazer o exercício, agora limitando o número de folhas. Para isso, escolha "Insert" no menu e "Insert Cell Bellow". Depois, copie o conteúdo da célula anterior na célula criada e troque max_depth, por max_leaf_nodes.

Visualizando as superfícies de decisão No exemplo abaixo, o notebook apresenta as superficíes de decisão encontradas pelo classificador, induzindo-o par a par de características e, depois, classificando pontos do espaço (2D) formado pelo par escolhido. A escolha dos pontos a serem classificados é feita usando o método meshgrid do numpy, que gera um conjunto de pontos no plano, que não necessariamente correspondem a um objeto real, e classifica-o. Esse mapa de pontos é, então, pintado para a visualização das regiões onde um objeto com aqueles valores de características seria classificado.

[31]:
```
# PRECISA ACERTAR A VISUALIZAÇÃO DESTES GRÁFICOS QUE A SEPARAÇÃO NÃO FICOU BOA

import numpy as np

n_classes = 3
plot_colors = "bry"
plot_step = 0.02

# Induz o classificador para cada par de características.
```

```
for pairidx, pair in enumerate([[0, 1], [0, 2], [0, 3],
                                [1, 2], [1, 3], [2, 3]]):

    # X agora é um conjunto com apenas as características do par.
    X = iris.data[:, pair]
    Y = iris.target

    # Induz o classificador com X e Y como dados e rótulos.
    clf = tree.DecisionTreeClassifier().fit(X, Y)

    # Plot the decision boundary
    plt.subplot(2, 3, pairidx + 1)

    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, plot_step),
                         np.arange(y_min, y_max, plot_step))

    # Classifica os pontos da grade gerada e apresenta-os no gráfico
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    cs = plt.contourf(xx, yy, Z, cmap=plt.cm.Paired)

    # Escolhe o lugar da figura onde vai ficar o gráfico
    plt.xlabel(iris.feature_names[pair[0]])
    plt.ylabel(iris.feature_names[pair[1]])
    plt.axis("tight")

    # Coloca os pontos de treinamento no gráfico
    for i, color in zip(range(n_classes), plot_colors):
        idx = np.where(Y == i)
        plt.scatter(X[idx, 0], X[idx, 1], c=color, label=iris.target_names[i],
                    cmap=plt.cm.Paired)
    plt.axis("tight")

plt.suptitle("Decision surface of a decision tree using paired features")
plt.legend()
plt.show()
```


Decision surface of a decision tree using paired features
sepal width (cm)
petal length (cm)
petal width (cm)
petal length (cm)
petal width (cm)
petal width (cm)
sepal width (cm)
sepal width (cm)
petal length (cm)
setosa
versicolor
virginica