

Mineração de Dados

06 - Implementação de Árvore de Decisão em Python

Marcos Roberto Ribeiro



Instituto Federal Minas Gerais - Campus Bambuí

2018

- Nesta aula vamos implementar uma árvore de decisão em Python, usando sklearn e pandas
- Serão abordados os seguintes tópicos:
 - Importação de arquivo csv usando pandas
 - Utilização do pandas para preparar os dados
 - Criar a árvore de decisão usando o sklearn
 - Plotar a árvore de decisão
 - Extração das regras da árvore
- Vamos considerar um ambiente com Ubuntu Linux 18.04

Instalação dos Pacotes

- No Linux devem ser instalados os seguintes pacotes:
 - python3-numpy
 - python3-scipy
 - python3-sklearn
 - python3-pandas
 - python3-graphviz¹
 - ipython3
- Recomendo instalar também: python-numpy-doc python-scipy-doc python-sklearn-doc python-pandas-doc

¹No caso de do Ubuntu 16.04 trocar instalar o pacote python3-pydot

Primeiros Passos

Importação das Bibliotecas

```
# Biblioteca para desenhar a árvore  
# No caso do Ubuntu 16.04 usar a biblioteca pydot  
import graphviz  
# Bibliotecas para trabalhar com os dados  
import pandas as pd  
import numpy as np  
# Bibliotecas para criar a árvore e exportar para graphviz  
from sklearn.tree import DecisionTreeClassifier,  
↪ export_graphviz
```

Lendo os Dados

```
# Leitura dos dados de um arquivo CSV  
df = pd.read_csv('iris.csv')
```

Informações sobre os Dados

Cabeçalho

```
print(df.head())
```

	SepalLength	SepalWidth	PetalLength	PetalWidth	Class
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

Classes

```
df['Class'].value_counts()
```

Iris-virginica	50
Iris-setosa	50
Iris-versicolor	50

Criando a Árvore de Decisão

```
# Colunas de dados
cols = list(df.columns[:4])
data_cols = df[cols]
# Coluna da classe
class_col = df['Class']
# Cria a árvore de decisão com parâmetros padrões
dt = DecisionTreeClassifier(random_state=0)
# Treina a árvore de decisão
dt.fit(data_cols, class_col)
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                        max_features=None, max_leaf_nodes=None, min_samples_leaf=1,
                        min_samples_split=2, min_weight_fraction_leaf=0.0,
                        presort=False, random_state=None, splitter='best')
```

Observação

O parâmetro **random_state** deve ter um valor constante para que seja gerada sempre a mesma árvore

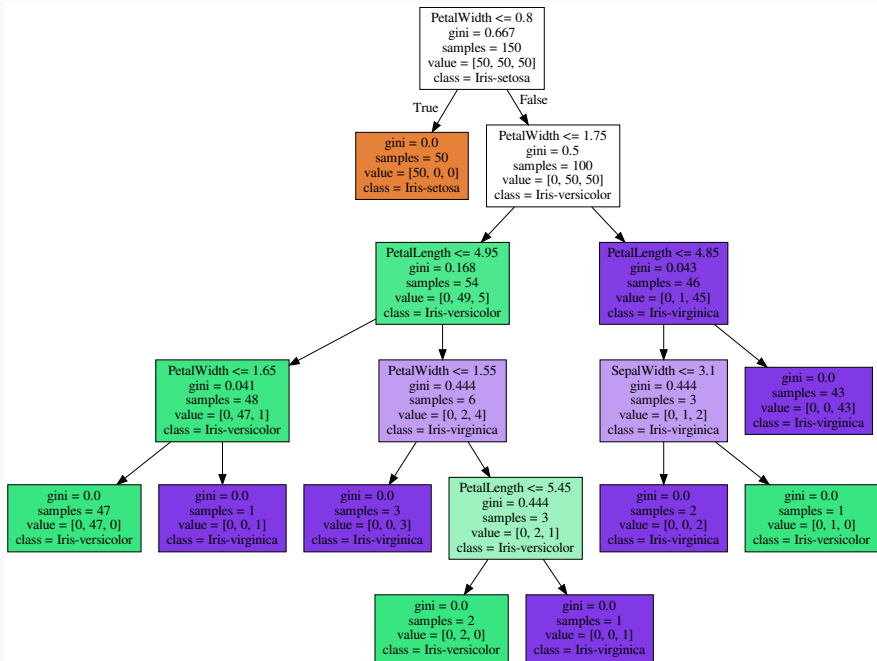
Desenhando a Árvore

Ubuntu 18.04

```
graph_data = export_graphviz(dt, out_file=None, feature_names=cols,  
                             class_names=df["Class"], filled=True)  
graph = graphviz.Source(graph_data)  
graph.render('dt.pdf')
```

Ubuntu 16.04

```
export_graphviz(dt, out_file='dt.dot', feature_names=cols,  
                class_names=df["Class"], filled=True)  
tg = pydot.graph_from_dot_file('dt.dot')  
tg.write_png('dt.png')
```



Classificado Novos Dados

```
dt.predict([[5.6, 2.1, 4.9, 1.8]])
```

```
array(['Iris-virginica'], dtype=object)
```

```
dt.predict(data_cols)
```

```
array(['Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',  
      'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',  
      'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',  
      'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',  
      'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',  
      'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',  
      'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',  
      'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',  
      'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',  
      'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',  
      'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',  
      'Iris-setosa', 'Iris-setosa', 'Iris-versicolor', 'Iris-versicolor',  
      'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',  
      ...])
```

Validando com o Todo o Conjunto de treinamento

```
# Bibliotecas com métricas de validação
from sklearn.metrics import classification_report,
    ↪ confusion_matrix
# Resultado da classificação
pred = dt.predict(data_cols)
# Exibindo as métricas
print(confusion_matrix(class_col, pred))
print(classification_report(class_col, pred))
```

```
[[50  0  0]
 [ 0 50  0]
 [ 0  0 50]]
```

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	50
Iris-versicolor	1.00	1.00	1.00	50
Iris-virginica	1.00	1.00	1.00	50
avg / total	1.00	1.00	1.00	150

Validação Cruzada

```
# Biblioteca para validação cruzada  
from sklearn.cross_validation import cross_val_score  
# Cálculo da validação cruzada  
score = cross_val_score(dt, data_cols, class_col, cv=10)  
# Acurácia média  
print(score.mean())
```

0.96

Extraindo as Regras i

```
from sklearn.tree import _tree

def tree_to_code(dtree, feature_names):
    # Estrutura da árvore de decisão
    tree_ = dtree.tree_
    # Considera valor indefinido para nós
    all_features = [
        feature_names[i] if i != _tree.TREE_UNDEFINED else "undefined!"
        for i in tree_.feature
    ]
    # Imprime cabeçalho
    print("def tree({}):".format(", ".join(feature_names)))
```

Extraindo as Regras ii

```
# Função recursiva para percorrer a estrutura da árvore
def recurse(node, depth):
    # Calcula indentação com base na profundidade do nó
    indent = "  " * depth
    # Teste se o nó tem valor diferente de indefinido
    if tree_.feature[node] != _tree.TREE_UNDEFINED:
        # Atributo do nó
        name = all_features[node]
        # Valor do atributo
        threshold = tree_.threshold[node]
        # Recursão a esquerda
        print("{}if {} <= {}:".format(indent, name, threshold))
        recurse(tree_.children_left[node], depth + 1)
        # Recursão a direita
        print("{}else:".format(indent))
        recurse(tree_.children_right[node], depth + 1)
    else:
        print("{}return {}".format(indent, np.argmax(tree_.value[node][0])))
```


Extraindo as Regras iii

```
recurse(0, 1)  
  
tree_to_code(dt, cols)
```

Extraindo as Regras iv

```
def tree(SepalLength, SepalWidth, PetalLength, PetalWidth):  
    if PetalWidth <= 0.800000011920929:  
        return 0  
    else:  
        if PetalWidth <= 1.75:  
            if PetalLength <= 4.949999809265137:  
                if PetalWidth <= 1.6500000953674316:  
                    return 1  
                else:  
                    return 2  
            else:  
                if PetalWidth <= 1.5499999523162842:  
                    return 2  
                else:  
                    if PetalLength <= 5.449999809265137:  
                        return 1  
                    else:  
                        return 2  
        else:  
            if PetalLength <= 4.850000381469727:  
                if SepalWidth <= 3.09999999046325684:  
                    return 2  
                else:  
                    return 1  
            else:  
                return 2
```

- Estudar os demais parâmetros da classe `sklearn.tree.DecisionTreeClassifier` e criar outras árvores de decisão usando estes parâmetros

 Quinlan, J. R. (1986).
Induction of decision trees.
Machine learning, 1(1):81–106.

 TAN, P.-N., STEINBACH, M., and KUMAR, V. (2009).
Introdução ao data mining: mineração de dados.
Ciência Moderna, Rio de Janeiro.