

```
Score : 0.5626822157434402
Out[23]: array([[ 0,  0,  1,  1,  0,  0],
        [ 0,  0,  4,  5,  0,  0],
        [ 0,  0,  91, 54,  0,  0],
        [ 0,  0, 36, 102,  0,  0],
        [ 0,  0,  1, 42,  0,  0],
        [ 0,  0,  1,  5,  0,  0]], dtype=int64)
```

Se os vinhos com as notas 3 e 8 fossem os mais importantes na correta classificação, pois eles são de alguma forma venenosos? Pela classificação acima (na matriz de confusão), nenhum dos dois foi corretamente classificado. Isso pode ser causa do não-balanceamento dos dados. Podemos tentar corrigir isso estimando um modelo com o argumento `class_weight`, como "balanced", de forma que um peso é dado a cada classe, inversamente proporcional à sua frequência relativa no banco.

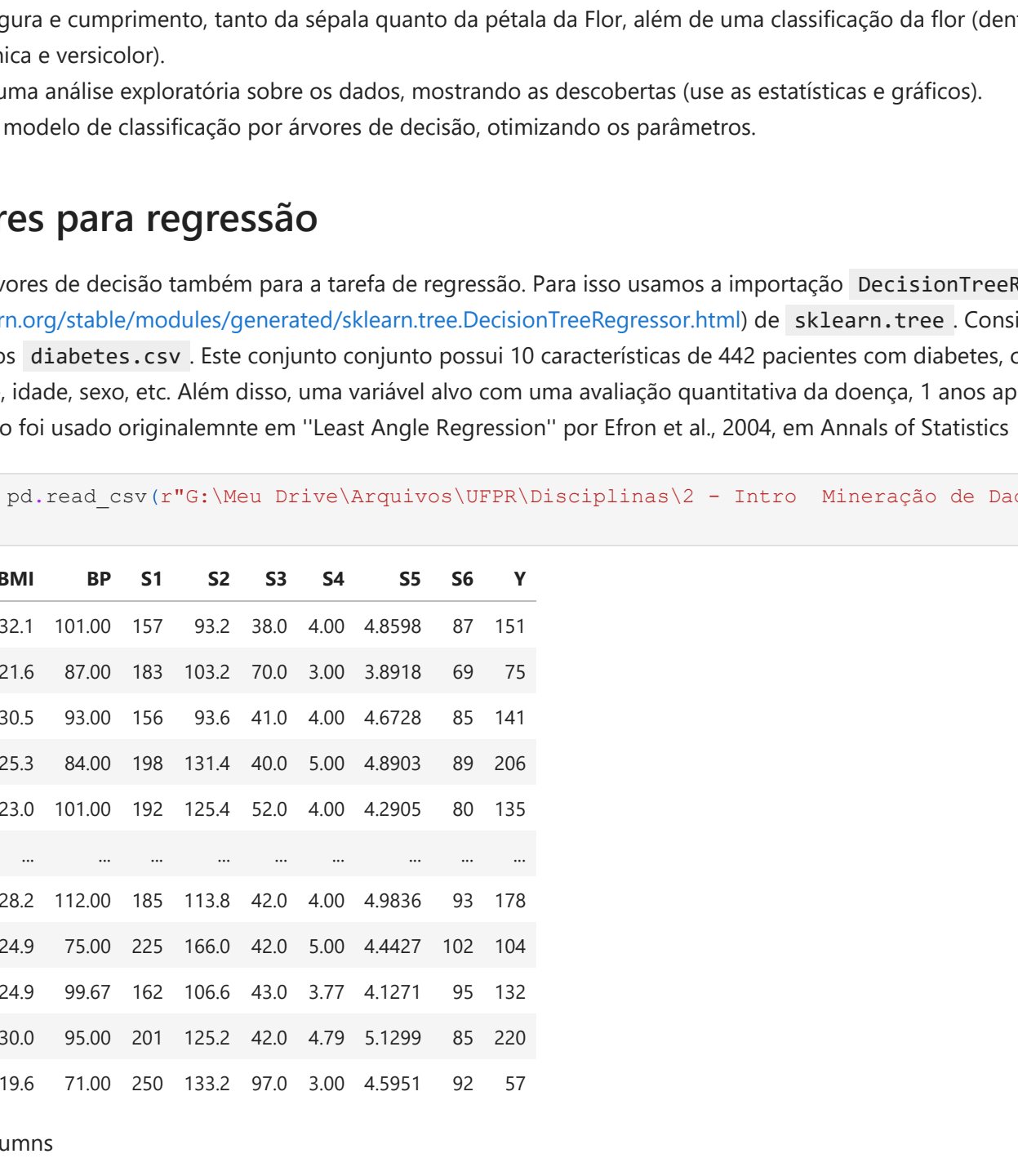
```
In [24]: # Podemos alterar os parâmetros da árvore para tentar deixar ela mais genérica (podar)

# Criando um conjunto de treino/testes - holdout 0.3
clf = tree.DecisionTreeClassifier(max_depth = 2, criterion = "entropy", random_state = 1, class_weight = "balanced",
                                clf.fit(x_treino, y_treino))

fig = plt.figure()
fig.set_size_inches(15,10)
fig = tree.plot_tree(clf)
plt.show()

# Verificando o score e a matriz de confusão para os dados de teste:
score = clf.score(x_teste, y_teste)
print("Score : ", score)
y_pred = clf.predict(x_teste)
y_pred

confusion_matrix(y_teste, y_pred)
```



```
Score : 0.14868804664723032
Out[24]: array([[ 1,  1,  0,  0,  0,  0],
        [ 1,  3,  4,  0,  0,  2],
        [22, 56, 43,  0,  0, 24],
        [23, 27, 20,  0,  0, 68],
        [ 1,  6,  2,  0,  0, 24],
        [ 0,  1,  0,  0,  0, 4]], dtype=int64)
```

Repare que o score foi reduzido, porém as classificações corretas do vinho 3 e 8 aumentaram. Dessa forma, nem sempre a acurácia (medida pelo `score`) é a melhor medida de desempenho em uma árvore. Tudo vai depender do que se deseja realizar.

7.12 Rotina para criação de um modelo

De forma geral, alguns passos podem ser tomados ao se induzir um modelo de classificação usando árvores de decisão:

1. Realizar uma análise exploratória dos dados, verificar se existe classes não balanceadas e se são importantes.
2. Separar os dados em treino e testes.
3. Ajustar uma árvore com os dados de treino.
4. (se factível) imprimir visualmente a árvore.
5. Verificar a acurácia da classificação, tanto no treino quanto nos teste (evitar overfitting). Escolher uma medida de desempenho adequada ao que se deseja (lembre do desbalanceamento das classes).
6. Tentar otimizar os parâmetros do modelo.
7. Com os parâmetros encontrados, estimar uma nova árvore com todo o conjunto de dados, e usar o modelo no negócio.

Exercícios

1. Considere o conjunto de dados `WineQV.csv` utilizado neste Notebook. Finalize a análise dos dados otimizando os parâmetros do modelo. Pense em qual medida será usada para avaliar o desempenho (considere o cenário em que os vinhos com notas 3 e 8 são venenosos.).
2. Considerando o conjunto de dados `IrisDataset.csv`, Este conjunto contém dados sobre três tipos de flores Iris. Cada registro contém dados de largura e comprimento, tanto da sépala quanto da pétala da Flor, além de uma classificação da flor (dentre uma das três - setosa, virginica e versicolour).

- A. Realize uma análise exploratória sobre os dados, mostrando as descobertas (use as estatísticas e gráficos).
- B. Crie um modelo de classificação por árvores de decisão, otimizando os parâmetros.

7.13 Árvores para regressão

Podemos usar árvores de decisão também para a tarefa de regressão. Para isso usamos a importação `DecisionTreeRegressor` (<https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html>) de `sklearn.tree`. Considerando o conjunto de dados `diabetes.csv`, Este conjunto possui 10 características de 442 pacientes com diabetes, como pressão média do sangue, idade, sexo, etc. Além disso, uma variável alvo com uma avaliação quantitativa da doença, 1 anos após a coleta dos dados. O conjunto foi usado originalmente em "Least Angle Regression" por Efron et al., 2004, em *Annals of Statistics*

```
In [25]: pd_diabetes = pd.read_csv("Q0:\Meu Drive\Arquivos\UFFPR\Disciplinas\2 - Intro - Mineração de Dados\Python\Dataset\diabetes
pd_diabetes

Out[25]:
```

	AGE	SEX	BMI	BP	S1	S2	S3	S4	S5	S6	Y
0	59	2	321	101.00	157	93.2	38.0	4.00	4.858	87	151
1	48	1	216	87.00	183	103.2	70.0	3.00	3.8918	69	75
2	72	2	30.5	93.00	156	93.6	41.0	4.00	4.6728	85	141
3	24	1	25.3	84.00	198	131.4	40.0	5.00	4.8903	89	206
4	50	1	23.0	101.00	192	125.4	52.0	4.00	4.2305	80	135
...
437	60	2	28.2	112.00	185	113.8	42.0	4.00	4.9836	93	178
438	47	2	24.9	75.00	225	166.0	42.0	5.00	4.4427	102	104
439	60	2	24.9	99.07	162	106.6	43.0	3.77	4.1271	95	132
440	36	1	30.0	95.00	201	125.2	42.0	4.79	5.1299	85	220
441	36	1	19.6	71.00	250	133.2	97.0	3.00	4.5951	92	57

442 rows × 11 columns

Separando os atributos e o valor alvo em X e Y.

```
In [26]: X = pd_diabetes.iloc[:,0:10]
X = pd_diabetes.iloc[:,0:11]
```

Separando os dados em treino e teste por holdout 0.3 teste:

```
In [27]: x_treino, y_teste, y_treino, y_teste = train_test_split(X, Y, test_size = 0.3, random_state = 1)
print(x_treino.shape, x_teste.shape)

(309, 10) (133, 10)
```

Declarando e estimando o regressor:

```
In [308]: from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor()
regressor.fit(x_treino, y_treino)

Out[308]: DecisionTreeRegressor()
```

Plotando a árvore:

```
In [302]: fig = plt.figure()
fig.set_size_inches(20,15)
fig = tree.plot_tree(regressor, filled = True, feature_names = X.columns)
plt.show()
```



Como não existem classes, a verificação do erro de predições pelo método `score`, é dada pelo coeficiente de determinação R^2

(https://pt.wikipedia.org/wiki/Coefficiente_de_determina%C3%A7%C3%A3o, no Python <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html#sklearn.tree.DecisionTreeRegressor.fit>). Um coeficiente de 1 para os testes indica um overfitting do modelo.

```
In [301]: print("R2 testes : ", regressor.score(x_treino, y_treino))

R2 testes : 1.0

O coeficiente está muito alto (=1), o que pode indicar um overfitting aos dados de treino. Verificando o valor nos dados de teste:
```

```
In [305]: print(regressor.score(x_teste, y_teste))

-0.48217911494509824
-0.1655013093100642

Out[305]:
```

Realmente os dados se ajustaram muito bem ao treino porém não conseguem explicar os dados de teste. Criando um pequeno algoritmo para otimizar os parâmetros de regressão: usando a validação k-fold com k = 4, vamos alterar os parâmetros `max_depth`, `criterion`, `splitter`, `min_samples_split`.

```
In [306]: # Parâmetros variáveis:
criterio = ["squared_error", "friedman_mse", "absolute_error", "poisson"]
split = ["best", "random"]
min_samples = range(2,5)
param_otimos = {}

best_s = 0
for i in range(1,20):
    for c in criterio:
        for s in split:
            for m in min_samples:
                regressor = DecisionTreeRegressor( random_state = 42, max_depth = i, criterion = c, splitter = s,
                mean_score = cross_val_score(regressor, X, y, cv = 4).mean()
                if mean_score > best_s:
                    best_s = mean_score
                    param_otimos = i,c,s,m
                    print("Melhor current score : ", best_s)
                    print("Melhor current param : ", param_otimos)

print("Melhor score : ", best_s)
print("Melhor param : ", param_otimos)

Melhor current score : 0.21894383383110033
Melhor current param : (1, 'squared_error', 'best', 2)
Melhor current score : 0.3345696710614511
Melhor current param : (2, 'squared_error', 'best', 2)
Melhor current score : 0.344610077434291
Melhor current param : (2, 'poisson', 'best', 2)
Melhor current score : 0.35375360294288943
Melhor current param : (3, 'poisson', 'best', 2)
Melhor score : 0.35375360294288943
Melhor param : (3, 'poisson', 'best', 2)
```

Portanto os melhores parâmetros são: `max_depth = 3`, `criterion = 'poisson'`, `splitter = 'best'`, `min_samples_split = 2`, com um R^2 médio de 0.3537 (o que é ainda muito baixo). Ajustando um novo modelo com esses parâmetros e plotando a árvore:

```
In [311]: regressor = DecisionTreeRegressor( random_state = 42, max_depth = 3, criterion = 'poisson', splitter = 'best',
regressor.fit(x_treino, y_treino)
mean_score = cross_val_score(regressor, X, y, cv = 4).mean()
print("Erro no teste (k-fold), mean_score)
print("Erro no treino", regressor.score(x_teste, y_teste))

fig = plt.figure()
fig.set_size_inches(20,15)
fig = tree.plot_tree(regressor, filled = True, feature_names = X.columns)
plt.show()

Erro no teste (k-fold) 0.35375360294288943
Erro no treino 0.14051298340366853
```



Como não existem classes, a verificação do erro de predições pelo método `score`, é dada pelo coeficiente de determinação R^2

(https://pt.wikipedia.org/wiki/Coefficiente_de_determina%C3%A7%C3%A3o, no Python <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html#sklearn.tree.DecisionTreeRegressor.fit>). Um coeficiente de 1 para os testes indica um overfitting do modelo.

```
In [301]: print("R2 testes : ", regressor.score(x_treino, y_treino))

R2 testes : 1.0

O coeficiente está muito alto (=1), o que pode indicar um overfitting aos dados de treino. Verificando o valor nos dados de teste:
```

```
In [305]: print(regressor.score(x_teste, y_teste))

-0.48217911494509824
-0.1655013093100642

Out[305]:
```

Realmente os dados se ajustaram muito bem ao treino porém não conseguem explicar os dados de teste. Criando um pequeno algoritmo para otimizar os parâmetros de regressão: usando a validação k-fold com k = 4, vamos alterar os parâmetros `max_depth`, `criterion`, `splitter`, `min_samples_split`.

```
In [306]: # Parâmetros variáveis:
criterio = ["squared_error", "friedman_mse", "absolute_error", "poisson"]
split = ["best", "random"]
min_samples = range(2,5)
param_otimos = {}

best_s = 0
for i in range(1,20):
    for c in criterio:
        for s in split:
            for m in min_samples:
                regressor = DecisionTreeRegressor( random_state = 42, max_depth = i, criterion = c, splitter = s,
                mean_score = cross_val_score(regressor, X, y, cv = 4).mean()
                if mean_score > best_s:
                    best_s = mean_score
                    param_otimos = i,c,s,m
                    print("Melhor current score : ", best_s)
                    print("Melhor current param : ", param_otimos)

print("Melhor score : ", best_s)
print("Melhor param : ", param_otimos)

Melhor current score : 0.21894383383110033
Melhor current param : (1, 'squared_error', 'best', 2)
Melhor current score : 0.3345696710614511
Melhor current param : (2, 'squared_error', 'best', 2)
Melhor current score : 0.344610077434291
Melhor current param : (2, 'poisson', 'best', 2)
Melhor current score : 0.35375360294288943
Melhor current param : (3, 'poisson', 'best', 2)
Melhor score : 0.35375360294288943
Melhor param : (3, 'poisson', 'best', 2)
```

Portanto os melhores parâmetros são: `max_depth = 3`, `criterion = 'poisson'`, `splitter = 'best'`, `min_samples_split = 2`, com um R^2 médio de 0.3537 (o que é ainda muito baixo). Ajustando um novo modelo com esses parâmetros e plotando a árvore:

```
In [311]: regressor = DecisionTreeRegressor( random_state = 42, max_depth = 3, criterion = 'poisson', splitter = 'best',
regressor.fit(x_treino, y_treino)
mean_score = cross_val_score(regressor, X, y, cv = 4).mean()
print("Erro no teste (k-fold), mean_score)
print("Erro no treino", regressor.score(x_teste, y_teste))

fig = plt.figure()
fig.set_size_inches(20,15)
fig = tree.plot_tree(regressor, filled = True, feature_names = X.columns)
plt.show()

Erro no teste (k-fold) 0.35375360294288943
Erro no treino 0.14051298340366853
```

