	 2. Malic acid 3. Ash 4. Alcalinity of ash 5. Magnesium 6. Total phenols 7. Flavanoids 8. Nonflavanoid phenols 9. Proanthocyanins 10. Color intensity 11. Hue 12. OD280/OD315 of diluted wines 13. Proline Mais informaçoes sobre os dados estão disponíveis em (https://archive.ics.uci.edu/ml/datasets/wine):
0 0	Mais informações sobre os dados estão disponíveis em (https://archive.ics.uci.edu/ml/datasets/wine): import numpy as np import pandas as pd #Lendo a base de dados : o primeiro valor é a classe do vinho (1,2 ou 3), os outros são as caracteristic dt = pd.read_csv(r"G:\Meu Drive\Arquivos\UFPR\Disciplinas\2 - Intro Mineração de Dados\Python\Datasets\dt 0 1 2 3 4 5 6 7 8 9 10 11 12 13 0 1 14.23 1.71 2.43 15.6 127 2.80 3.06 0.28 2.29 5.64 1.04 3.92 1065 1 1 13.20 1.78 2.14 11.2 100 2.65 2.76 0.26 1.28 4.38 1.05 3.40 1050 2 1 13.16 2.36 2.67 18.6 101 2.80 3.24 0.30 2.81 5.68 1.03 3.17 1185
	3 1 14.37 1.95 2.50 16.8 113 3.85 3.49 0.24 2.18 7.80 0.86 3.45 1480 4 1 13.24 2.59 2.87 21.0 118 2.80 2.69 0.39 1.82 4.32 1.04 2.93 735 173 3 13.71 5.65 2.45 20.5 95 1.68 0.61 0.52 1.06 7.70 0.64 1.74 740 174 3 13.40 3.91 2.48 23.0 102 1.80 0.75 0.43 1.41 7.30 0.70 1.56 750 175 3 13.27 4.28 2.26 20.0 1.29 1.69 0.69 0.43 1.35 10.20 0.59 1.56 835 176 3 13.17 2.59 2.37 20.0 1.69 0.50 0.56 1.35 9.20 0.61 1.60 560
	7.3 Ajustando uma árvore Para fazer o ajuste usando as árvores de decisão, precisamos ter um conjunto com os atributos, sendo um array-like (matriz em que registro é uma linha, com n-colunas de atributos), e o vetor das classes, que deve ter o mesmo número de elementos que as linhas observações. Por isso usamos o pandas para separar os dados em X(atributos) e Y(classes). Para estimarmos um modelo de árvore basta instanciarmos um classificador com DecisionTreeClassifier() e usar o método fit , passando X e Y como argumento.
	<pre>#Devido a estrutura do pacote, não podemos importar dessa forma, temos que importar os recursos individo #import sklearn as sk #https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html from sklearn import tree X = dt.iloc[:,1:] Y = dt.iloc[:,0] clf = tree.DecisionTreeClassifier() clf = clf.fit(X, Y)</pre>
	7.4 Visualizando Podemos exportar a árvore em formato de texto e imagem. # Exportando a arvore como texto texto = tree.export_text(clf) print(texto) feature_12 <= 755.00 feature_11 <= 2.11 feature_10 <= 0.94 feature_6 <= 1.58 feature_6 <> 1.58
	<pre> feature_6 > 1.58 class: 2 feature_10 > 0.94 feature_6 <= 0.56 class: 3 feature_6 > 0.56 class: 2 feature_11 > 2.11 feature_6 <= 0.80 class: 3 feature_6 > 0.80 feature_0 <= 13.17 feature_0 <= 13.17 feature_0 > 13.17 feature_9 <= 4.06</pre>
	Para exportarmos a árvore de forma visual, é necessário ter o pacote matplotlib instalado. O código abaixo também exporta a á em formato pdf. # Exportando visualmente (precisa do matplotlib) import matplotlib.pyplot as plt fig = plt.figure() fig.set_size_inches(25,12) im = tree.plot_tree(clf, filled = True) fig.savefig("arvore2.pdf")
	gini = 0.0 samples = 39 value = [0, 0, 39] gini = 0.0 samples = 1 value = [0, 1, 0] gini = 0.0 samples = 5 value = [0, 5, 0] gini = 0.0 samples = 5 value = [0, 5, 0] gini = 0.0 samples = 5 value = [0, 58, 0] gini = 0.0 samples = 5 value = [0, 58, 0] gini = 0.0 samples = 5 value = [0, 3, 0] gini = 0.0 samples = 3 value = [0, 3, 0] 7.5 Como interpretar a árvore Considere a seguinte imagem:
	X[12] <= 755.0 gini = 0.658 samples = 178 value = [59, 71, 48] True False
6	X[11] <= 2.115 gini = 0.492 samples = 111 value = [2,67] 42] I - X[12] <= 755.0: Essa é a condição do nó, ou seja, se o atributo X[12] (como não passamos uma lista de nomes ele indica como elemento do vetor), for menor ou igual a 755.0 La gini = 0.658: A medicão gini se refere a pureza de um pó Quanto mais baixo o valor gini menos beterogeneidade existe pas
	 II - gini=0.658: A medição <i>gini</i> se refere a pureza de um nó. Quanto mais baixo o valor gini, menos heterogeneidade existe nas separações das classes (mostrada em value). Um nó com valor de gini = 0 implica que toda a amostra está somente em uma classe. III - True: Indica o caminho da condição, ou seja, se X[12] <= 755.0, siga para a esquerda, senão para a direita. IV - Samples = 111: Indica o número de elementos da amostra que estão separados no nó. Considerando o primeiro nó, temos que samples = 178, o que é a conjunto total. Seguindo o caminho True, ou seja, se X[12] <= 755.0 a amostra cai para 111. V - Value = [2,67,42]: Esse número indica o número de elementos (da amostra do nó), em cada classe. Ou seja, considerando uma separação dos dados somente pela condição X[12] <= 755.0, a amostra fica com 111 elementos, e destas 2 são da classe 1, 67 da ce 42 da classe 3. Se quisessemos usar somente essa regra para realizar a classificação, usariamos a classe com a maior das frequêncidades, ou seja, usando somente a regra X[12] <= 755.0, o classificador considera que estamos falando do vinho do tipo 2 (devido a solution de considera que estamos falando do vinho do tipo 2 (devido a solution de considera que estamos falando do vinho do tipo 2 (devido a solution de considera que estamos falando do vinho do tipo 2 (devido a solution de considera que estamos falando do vinho do tipo 2 (devido a solution de considera que estamos falando do vinho do tipo 2 (devido a solution de considera que estamos falando do vinho do tipo 2 (devido a solution de considera que estamos falando do vinho do tipo 2 (devido a solution de considera que estamos falando do vinho do tipo 2 (devido a solution de considera que estamos falando do vinho do tipo 2 (devido a solution de considera que estamos falando do vinho do tipo 2 (devido a solution de considera que estamos falando do vinho do tipo 2 (devido a solution de considera que estamos falando do vinho do tipo 2 (devido a solution de considera que estamo
	<pre># Podemos passar uma lista com o nome dos atributos e das classes ao se plotar a árvore (https://scikit v_nomes = ["Vinho 1","Vinho 2","Vinho 3"] v_atrib = ["Alcohol","Malic acid","Ash","Alcalinity of ash","Magnesium","Total phenols","Flavanoids","No</pre>
	Protect = 7.75.0
	Han <= 0.935 gm = 0.27 gm = 0.17 gm = 0.17 gm = 0.15 gm = 0.25 g
	gin = 0.0 gin =
	# Também é possível determinar o criterio de separação a ser usado (o default é o gini) clf = tree.DecisionTreeClassifier(criterion = "entropy") clf = clf.fit(X, Y) 7.6 Usando o modelo para classificação Uma vez ajustado, podemos usar o modelo para classificar novos registros usando a função predict, bastando passar um array- (matriz). O retorno é um array com uma classificação para cada linhas dos dados de entrada. # Considere o conjunto de atributos (como uma matriz: linhas = numero de instancias a classificar, colu- novo_vinho = [[14,2,2,14.56,120,2.08,3,0.25,3,5.5,1,4,1048]] classe_novo_vinho = clf.predict(novo_vinho)
	# O resultado é um array com o tamanho das linhas da matriz de previsão, com os elementos classificados print("Classe novo vinho : ",classe_novo_vinho) # Se passarmos 2 valores a serem classificados: novos_vinhos = [[14,2,2,14.56,120,2.08,3,0.25,3,5.5,1,4,1048],
	 7.8 Separação dos dados Quando fazemos a estimação de um modelo, usamos uma parte dos dados para a estimação, e outra parte para os testes. Existem diversas formas de se realizar essa separaçai, como o método holdout e o cross-validation 7.8.1 Método Holdout Separar os dados em duas partes: uma de treinamento e testes, com uma definição da porcentagem de quantos registros devem es em cada uma. O método holdout é exemplificado pela figura abaixo: Teste
	Treino
	Gerar o modelo a partir dos dados de treinamento para em seguida usar o conjunto de testes para aferir a sua acurácia. Para isso u o método train_test_split de sklearn.cross_validation (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html#sklearn.model_selection.train_test_split). Nesse ca passamos os arrays X e Y, o tamanho do teste. random_state serve para reproducibilidade dos resultados, enquanto stratify faz uma amostragem estratificada, tentando balancear as classes presentes no conjunto de treino e de testes. # EXEMPLO USANDO HOLDOUT # Holdout -> dividindo a base em treinamento (70%) e teste (30%), estratificada from sklearn.model selection import train test split
	<pre>from sklearn.model_selection import train_test_split X_treino, X_teste, Y_treino, Y_teste = train_test_split(X, Y, test_size = 0.3, random_state = 0, stratic print("Elementos no treino : ", X_treino.shape[0]) print("Elementos no teste : ", X_teste.shape[0]) # Declara o classificador clf = tree.DecisionTreeClassifier() clf.fit(X_treino, Y_treino) predicted = clf.predict(X_teste) print(predicted) Elementos no treino : 124 Elementos no treino : 124</pre>
1	Elementos no teste : 54 [1 1 1 1 2 2 2 3 1 3 1 2 3 1 3 3 1 2 3 1 1 3 2 1 1 2 1 2
	Treino Treino
,	A eficácia do modelo é medida pelo erro médio de todos os modelos (veja os erros na próxima Seção). Dessa forma, o <i>k-fold</i> não é usado para estimar e usar o modelo , mas sim para verificar a acurácia de um modelo (ou mesmo comparar a forma de geração de vários modelos). Por esse motivo a saída dó método é um conjunto de erros (um para cada k). Para isso usamos a função cross_val_score importada de sklearn.model_selection. Podemos escolher qual tipo de medida queremos que ele calcula o argumento scoring , sendo elas: 1. 'accuracy' 2. 'balanced_accuracy'
	3. 'roc_auc' 4. 'f1' 5. 'neg_mean_absolute_error' 6. 'neg_root_mean_squared_error' 7. 'r2' Usaremos a accuracy , que diz respeito a porcentagem de instancias classificadas de forma correta com o modelo. from sklearn.model_selection import cross_val_score cl_cross = tree.DecisionTreeClassifier(criterion='entropy') folds = 10 scores = cross_val_score(cl_cross, X, Y, cv=5, scoring='accuracy')
	print ("Acuracia : ", scores) print ("Acuracia média : ", scores.mean()) Acuracia : [0.91666667 0.83333333 0.88888889 0.97142857 0.88571429] Acurácia média : 0.8992063492063492 7.9 Avaliando o desempenho do modelo Os erros cometidos por um modelo de classificação são geralmente divididos em dois grupos: erro de treinamento e erro de generalização (ou testes). Erros de treinamento se referem aos erros de classificação equivocada do modelo cometido no registro de treinamento, enquanto os erros de generalização são os erros do modelo em registros não vistos anteriormente. Um bom modelo deve ter baixa quantidade de erros de treinamento assim como de erros de generalização. Isso é importante, pois
	modelo com baixo erro de treinamento pode muito bem possuir um alto erro de testes. Isso é conhecido como <i>overfitting</i> (o mode está muito ajustado aos dados de treino, e não generaliza bem para instâncias não vistas). 7.9.1 Erro de treinamento Podemos aferir o erro de treinamento pelo próprio modelo, usando o método score. Ele mede a acurácia do modelo, ou seja, a porcentagem de classificações corretas. from sklearn.model_selection import train_test_split X_treino, X_teste, Y_treino, Y_teste = train_test_split(X, Y, test_size = 0.3, random_state = 0) print("Elementos no treino: ", X treino.shape[0])
	<pre>print("Elementos no teste : ", X_teste.shape[0]) # Declara o classificador clf = tree.DecisionTreeClassifier() clf.fit(X_treino, Y_treino) print(clf.score(X_treino, Y_treino)) fig = plt.figure() fig.set_size_inches(25,12) im = tree.plot_tree(clf, filled = True) Elementos no treino : 124 Elementos no teste : 54 1.0 X[9] <= 3.82 gini = 0.66</pre>
	value = [0, 44, 0] value = [1, 0, 0] gini = 0.0 samples = 1 value = [1, 0, 0] gini = 0.0 samples = 5 value = [0, 5, 0] 7.9.2 Erro de generalização (de testes) Usando o mesmo método score, porém com o conjunto de testes, temos a porcentagem de classes corretamente classificadas. Usando deve ter um score "equiparável", considerando o erro de testes e de treino.
	print (clf.score (X_teste, Y_teste)) 0.9259259259259259 OBS: Note que ao se alterar o parâmetro test_size do split, também se alteram os valores de acurácia. Por esse motivo a validad fold é mais confiável ao se escolher um modelo. O código abaixo estima um modelo com alguns parâmetros e calcula o erro de generalização usando holdout comum e k-fold: X_treino, X_teste, Y_treino, Y_teste = train_test_split(X, Y, test_size = 0.4, random_state = 42) model = tree.DecisionTreeClassifier(criterion = "entropy", random_state = 42) # Ajustando o modelo model.fit(X_treino, Y_treino)
	# Calculando o score por holdout score_holdout = model.score(X_teste, Y_teste) # Calculando por k-fold, k = 10 cl_cross = tree.DecisionTreeClassifier(criterion='entropy', random_state = 42) scores_k_fold = cross_val_score(cl_cross, X, Y, cv = 10, scoring = 'accuracy') print("Acuracia holdout: ", score_holdout) print("Acuracia média k-fold: ", scores_k_fold.mean()) Acuracia holdout: 0.847222222222222 Acuracia média k-fold: 0.9153594771241831
	<pre>Exemplo "toy" para o cálculo do score (treino e testes) X_treino = [[17,1,2], [15,1,2], [5,1,2], [5,1,2]] Y_treino = [0,0,1,1,0] X_testes = [[17,1,2], [15,1,2], [5,1,2], [5,1,2]] Y_testes = [0,0,1,1] mod_pred = tree.DecisionTreeClassifier(criterion = "entropy", max_depth = 3) mod_pred.fit(X_treino, Y_treino) fig = plt.figure() fig.set_size_inches(10,5) fig = tree.plot_tree(mod_pred, filled = True)</pre>
	<pre># Portanto existe erro: print("Acurácia do treino:",mod_pred.score(X_treino, Y_treino)) print("Acurácia do testes:",mod_pred.score(X_testes, Y_testes)) mod_pred.predict(X_treino) Acurácia do treino: 0.8 Acurácia do testes: 1.0 array([0, 0, 1, 1, 1]) X[0] <= 10.0 entropy = 0.971</pre>
	samples = 5 value = [3, 2] entropy = 0.918 samples = 3 value = [1, 2] entropy = 0.0 samples = 2 value = [2, 0]
	7.9.3 Matriz de confusão A matriz de confusão permite analisar em quais locais o modelo está errando mais (ou acertando mais). Isso é usado para dados desbalanceados, ou em que uma classe tem uma importância maior do que a outra. Passando um vetor com as classes ocorridas e com as estimadas pelo modelo, podemos calcular a matriz de confusão. from sklearn.metrics import confusion_matrix # Sejam os vetores v_ocorrido e v_previsto os tipos de vinhos e as classificações que o modelo fez: v_ocorrido = [1,2,3,3,2,2,1,3] v_previsto = [1,2,1,1,3,2,3,3] confusion_matrix(v_ocorrido, v_previsto)
	array([[1, 0, 1],
	<pre>X_treino, X_teste, Y_treino, Y_teste = train_test_split(X, Y, test_size = 0.3, random_state = 1) cl = tree.DecisionTreeClassifier(random_state = 1) cl.fit(X_treino, Y_treino) predicao = cl.predict(X_teste) from sklearn.metrics import confusion_matrix confusion_matrix(Y_teste, predicao) array([[22, 1, 0],</pre>
	Vimos como calcular o desempenho de uma árvore de decisão, e como o desempenho é afetado pelos parâmetros escolhidos. Des forma surge a pergunta: existe um conjunto de parâmetros capaz de gerar uma árvore melhor do que outra? A resposta é sim. Con alguns parâmetros possível para o DecisionTreeClassifier: 1. criterion{"gini", "entropy", "log_loss"}, default="gini": Método que define a qualidade da separação dos nós. 2. splitter{"best", "random"}, default="best": Método usado para realizar a separação dos nós. 3. max_depth:int, default=None: profundidade máxima da árvore. 4. min_samples_split:int or float, default=2: Número mínimo de amostras necessária para se expandir um nó. 5. min_samples_leaf:int or float, default=1: Numero mínimo de amostras em um nó. Um ponto de separação será considerado somente se deixar min_samples_leaf em cada lado da separação. Uma forma para tentarmos otimizar o modelo seria a seguinte:
,	 Atualizar parâmetros Ajustar modelo Calcular desempenho Guardar melhores parãmetros até o momento Voltar a 1 se critério de parada não for atingido Vamos criar um código que faça isso. Primeiro criaremos uma função que recebe os dados X e Y e os parâmetros, e retorna a acurá conjunto de treino e a média das acurácias do conjunto de testes pelo método k-fold com k=5: from sklearn.model_selection import train_test_split
	<pre>from sklearn.model_selection import train_test_split from sklearn import tree def calcula_z(X, Y, criterio): X_treino, X_teste, Y_treino, Y_teste = train_test_split(X, Y, test_size = 0.3, random_state = 1) cl = tree.DecisionTreeClassifier(random_state = 1, criterion = criterio) cl.fit(X_treino, Y_treino) score_treino = cl.score(X_treino, Y_treino) scores_k_fold = cross_val_score(cl, X, Y, cv = 5, scoring = 'accuracy') return (score_treino, scores_k_fold.mean()) scores = calcula_z(X,Y, "gini") print(scores) (1.0, 0.8876190476190476)</pre>
	Note que deixamos um argumento na função referente ao critério de cálculo da qualidade dos nós. Podemos então criar uma lista os 3 valores possíveis ["gini", "entropy", "log_loss"] e verificar qual gera melhores resultados: crit = ["gini", "entropy", "log_loss"] for c in crit: score = calcula_z(X,Y,c) print(score) (1.0, 0.8876190476190476) (1.0, 0.8934920634920633) (1.0, 0.8934920634920633)
	<pre>Ainda, podemos alterar a função para receber o parâmetro de max_depth: def calcula_z2(X, Y, criterio, n_depth): X_treino, X_teste, Y_treino, Y_teste = train_test_split(X, Y, test_size = 0.3, random_state = 1) cl = tree.DecisionTreeClassifier(random_state = 1, criterion = criterio, max_depth = n_depth) cl.fit(X_treino, Y_treino) score_treino = cl.score(X_treino, Y_treino) scores_k_fold = cross_val_score(cl, X, Y, cv = 5, scoring = 'accuracy') return (score_treino, scores_k_fold.mean()) crit = ["gini", "entropy", "log_loss"] max_prof = [1,2,3,4] for c in crit: for n in max_prof:</pre>
	<pre>for n in max_prof: score = calcula_z2(X,Y,c,n) print(score) (0.6612903225806451, 0.6463492063492063) (0.9193548387096774, 0.8261904761904761) (0.9838709677419355, 0.8820634920634921) (0.9919354838709677, 0.916031746031746) (0.6209677419354839, 0.562063492063492) (0.967741935483871, 0.910952380952381) (0.9919354838709677, 0.9046031746031747) (1.0, 0.8934920634920633) (0.6209677419354839, 0.562063492063492) (0.967741935483871, 0.910952380952381) (0.9919354838709677, 0.9046031746031747)</pre>
	(1.0, 0.8934920634920633) E dessa forma podemos criar o algoritmo para otimizar os parâmetros da árvore de decisão. 7.11 Dados não balanceados Considere o banco de dados WineQT.csv, que afere a qualidade de vinhos por uma nota (de 1 a 10), com base em um conjunto de atributos do vinho. As notas 3,4 e 8 são muito menos frequêntes do que as outras. dt_im = pd.read_csv(r"G:\Meu Drive\Arquivos\UFPR\Disciplinas\2 - Intro Mineração de Dados\Python\Datasedt_im
	<pre>dt_grouped = dt_im.groupby("quality").count() fig, ax = plt.subplots(1,1) ax.bar(dt_grouped.index, dt_grouped["Id"]) plt.show()</pre> 500 400
	dt_grouped fixed acidity volatile acidity acid sugar chlorides free sulfur dioxide density pH sulphates alcohologopathy
	<pre>## demora para rodar mesmo, até estimar ## # Gerando um modelo da forma normal X = dt_im.iloc[:,0:11] Y = dt_im.iloc[:,11] # Criando um conjunto de treino/testes - holdout 0.3 x_treino, x_teste, y_treino, y_teste = train_test_split(X, Y, test_size = 0.3, random_state = 1, straticle = tree.becisionTreeClassifier(criterion = "entropy", random_state = 1) clf.fit(x_treino, y_treino) fig = plt.figure()</pre>
	<pre>fig = plt.figure() fig.set_size_inches(20,15) fig = tree.plot_tree(clf) plt.show()</pre>
1	
1	
	<pre># Verificando o score e a matriz de confusão para os dados de teste: score = clf.score(x_teste, y_teste) print("Score: ", score) v_pred = clf.predict(x_teste) v_pred confusion_matrix(y_teste, v_pred) Score: 0.5422740524781341 array([[0, 0, 0, 2, 0, 0],</pre>
	<pre># Verificando o score e a matriz de confusão para os dados de teste: score = clf.score(x_teste, y_teste) print("Score: ", score) v_pred = clf.predict(x_teste) v_pred confusion_matrix(y_teste, v_pred) Score: 0.5422740524781341 array([[0, 0, 0, 2, 0, 0],</pre>

