# EXERCÍCIOS

## 1. Abra o arquivo Iris no WordPad ou Bloco de Notas e:

### a. Explique esta base de dados conforme descrição no arquivo .arff.

* Quantos registros? Resposta: 150 registros (instâncias)
* Quantos atributos? Resposta: 5 atributos no total, sendo 4 numéricos e 1 nominal (classe)
* Qual o significado dos atributos? Os atributos são:

1. sepallength - Comprimento da sépala (em cm)
2. sepalwidth - Largura da sépala (em cm)
3. petallength - Comprimento da pétala (em cm)
4. petalwidth - Largura da pétala (em cm)
5. class - A espécie da flor Iris, que pode ser:
   * Iris-setosa
   * Iris-versicolor
   * Iris-virginica

### b. O arquito .arff possui palavras reservadas? Quantas e quais são e quais suas funções?

Sim, o arquivo .arff possui 3 palavras reservadas principais: - (**RELATION?**) - Define o nome da relação/conjunto de dados (no caso, “iris”) - (**ATTRIBUTE?**) - Define os atributos e seus tipos (usado para cada característica do conjunto de dados) - (**DATA?**) - Indica o início da seção de dados/registros

### c. Descreva a estrutura de um arquivo .arff e exemplifique com o arquivo Iris.

A estrutura de um arquivo .arff é composta por três seções principais, que podemos exemplificar com o arquivo Iris:

**Cabeçalho/Header ((RELATION?))** - Primeira linha não comentada do arquivo - No arquivo Iris: @RELATION iris

**Declaração de Atributos ((ATTRIBUTE?))** - Lista todos os atributos e seus tipos - No arquivo Iris temos 5 declarações: @ATTRIBUTE sepallength REAL @ATTRIBUTE sepalwidth REAL @ATTRIBUTE petallength REAL @ATTRIBUTE petalwidth REAL @ATTRIBUTE class {Iris-setosa,Iris-versicolor,Iris-virginica}

**Seção de Dados ((DATA?))** - Começa com (**DATA?**) - Contém os registros, um por linha - Os valores são separados por vírgula - Exemplo do primeiro registro no Iris: 5.1,3.5,1.4,0.2,Iris-setosa

O arquivo também pode conter comentários, que começam com o símbolo ‘%’. No caso do Iris.arff, há vários comentários no início do arquivo que fornecem informações detalhadas sobre o conjunto de dados, incluindo sua origem, histórico de uso e estatísticas.

## 2. Abra o arquivo Iris no Weka e explique a base agora no ambiente gráfico.

Abrindo a funcao “Explorer”, a aba “Preprocess” contem os dados.

* Quantos registros? Resposta: 150 instantes
* Quantos atributos? Resposta: 5 attibutes
* Qual o significado dos atributos? Resposta: Os mesmos do item anterior
* Na sequência utilize o método de rede neural MultilayerPerceptron sem alterar qualquer parâmetro. Explique o que você entendeu do resultado.

Na configuracao padrao, o classificador MultilayerPerceptron assume: - Learning rate: 0.3 - Momentum: 0.2 - Epochs: 500 - Test options: cross-validation Folds :10

Definimos a variavel “class” como alvo.

Em 0.36 segundos o modelo foi capaz de classificar corretamente 146 (97,3333%) registros, errando 4 (2,6667%) Por classe, foi totalmente preciso para Iris-setosa (1.000) e menos para Iris-versicolor e Iris-vriginica (0.960)

## 3. Ainda com o arquivo Iris, utilize o método RandomTree e avalie o resultado obtido.

Compare com o resultado da rede neural!

O modelo apresentou uma acuracia ligeiramente menor, classificando corretamente apenas 138 (92%) registros. Por classe, foi igualmente preciso para Iris-setosa (1.000), mas variou a precisao para Iris-versicolor (0.896) e para Iris-virginica (0.865) Contudo, o modelo foi executado em apenas 0.01 segundos, uma reducao de 97% no tempo de execucao.

## 4. Ainda com o arquivo Iris, utilize o método SimpleCart e avalie o resultado obtido.

Compare com o resultado da questão 3!

Em relacao ao modelo 03, houve um aumento no tempo de execucao (0.13 segundos), mas a acuraria melhorou ligeiramente, acertando 143 registros (95.333%. Por classe, igualmente precisao integral para Iris-setosa, mas uma melhora para Iris-verginica (0.939) e Iris-versicolor (0.922) Alem disto, a arvore do modelo reduziu de tamanho (17 para 9)

### a. Faça um print da explicação da origem deste método (artigo) e a explicação de seus parâmetros.

O método CART (Classification And Regression Trees) foi originalmente apresentado no livro “Classification and Regression Trees”, publicado em 1984 por Leo Breiman, Jerome H. Friedman, Richard A. Olshen e Charles J. Stone. Esta obra é considerada fundamental no desenvolvimento de árvores de decisão e estabeleceu as bases metodológicas para muitos algoritmos modernos de machine learning.

O CART foi inovador por introduzir uma abordagem sistemática para construção de árvores de decisão que não necessitava de pressupostos sobre distribuições estatísticas subjacentes, diferentemente dos métodos de análise discriminante da época. O método pode lidar tanto com problemas de classificação quanto de regressão, e trabalha com variáveis numéricas e categóricas.

Quanto aos parâmetros específicos da implementação do SimpleCart no WEKA:

-S (Random number seed): Define a semente para o gerador de números aleatórios, permitindo reprodutibilidade dos resultados. Valor padrão é 1.

-M (Minimal number of instances): Especifica o número mínimo de instâncias que devem estar presentes em um nó terminal. O valor padrão é 2. Este parâmetro ajuda a controlar o crescimento da árvore e evitar overfitting.

-N (Number of folds): Define o número de partições usadas na poda por complexidade-custo mínima. O valor padrão é 5. Este parâmetro é usado durante o processo de poda para estimar o erro de generalização.

-U (Use minimal cost-complexity pruning): Controla se a poda por complexidade-custo mínima será utilizada. Por padrão, a poda é ativada. Esta é uma das principais contribuições do CART original - uma abordagem em duas etapas onde primeiro se cria uma árvore grande e depois se poda.

-H (Heuristic method for binary split): Controla o uso do método heurístico para divisões binárias. Por padrão está ativado. O CART original sempre faz divisões binárias, uma característica que o distingue de outros métodos de árvore.

-A (1 SE rule): Permite usar a regra de 1 Erro Padrão para decisões de poda. Por padrão está desativado. Esta regra foi proposta no livro original como uma forma de escolher árvores mais simples dentro de uma margem estatisticamente aceitável de desempenho.

-C (Percentage of training data): Define a proporção dos dados de treino a ser utilizada, variando de 0 a 1. O valor padrão é 1 (100% dos dados).

Esta implementação no WEKA segue os princípios fundamentais do CART original, mas com uma modificação importante no tratamento de valores ausentes: usa o método de “instâncias fracionárias” em vez do método de divisão substituta (surrogate split) descrito no livro original. Esta adaptação simplifica a implementação mantendo a eficácia no tratamento de dados faltantes.

O CART se tornou uma referência fundamental na área de árvores de decisão e influenciou o desenvolvimento de outros algoritmos importantes, como o Random Forest, também desenvolvido posteriormente por Breiman. A implementação no WEKA mantém a essência do método original enquanto oferece flexibilidade para ajustes através de seus parâmetros.

### b. Altere o parâmetro UsePrune para False. Execute e compare com o item a desta questão.

Com o UsePrune desligado, o tamanho da arvore aumentou para 11 e o modelo perdeu acuracia, caindo para 142 acertos (94.667%) por cento.

## 5. Abra o arquivo contact-lenses.arff e relate:

* quantos atributos Resposta: 5 atributos.
* quantos registros e Resposta: 24 registros.
* qual a distribuição de cada atributo. A distribuição dos atributos no arquivo contact-lenses.arff, podemos observar uma distribuição bastante equilibrada. O atributo age possui 8 instâncias para cada uma de suas categorias: young, pre-presbyopic e presbyopic. O atributo spectacle-prescrip está igualmente distribuído entre myope e hypermetrope, com 12 instâncias cada. Da mesma forma, astigmatism apresenta 12 instâncias tanto para “yes” quanto para “no”, e tear-prod-rate tem 12 instâncias para cada valor (reduced e normal). A classe contact-lenses apresenta um desbalanceamento, com 15 instâncias para “none”, 5 para “soft” e 4 para “hard”
* Relate o que você entendeu desta base de dados. Esta é uma base de dados médica projetada para auxiliar na decisão sobre prescrição de lentes de contato. O conjunto é completo e livre de ruído, contendo todas as combinações possíveis de pares atributo-valor, totalizando 24 instâncias. Os atributos descrevem características do paciente: sua idade (categorizada em jovem, pré-presbíope ou presbíope), prescrição dos óculos (míope ou hipermetrope), presença ou ausência de astigmatismo e taxa de produção de lágrimas (normal ou reduzida). A classe indica se o paciente deve usar lentes rígidas, macias ou não deve usar lentes de contato.

Na sequência utilize o método C4.5 (J48) com a opção de teste: Use training set.

### a. Faça um print-screen da árvore resultante.

=== Run information ===  
  
Scheme: weka.classifiers.trees.J48 -C 0.25 -M 2  
Relation: contact-lenses  
Instances: 24  
Attributes: 5  
 age  
 spectacle-prescrip  
 astigmatism  
 tear-prod-rate  
 contact-lenses  
Test mode: evaluate on training data  
  
=== Classifier model (full training set) ===  
  
J48 pruned tree  
------------------  
  
tear-prod-rate = reduced: none (12.0)  
tear-prod-rate = normal  
| astigmatism = no: soft (6.0/1.0)  
| astigmatism = yes  
| | spectacle-prescrip = myope: hard (3.0)  
| | spectacle-prescrip = hypermetrope: none (3.0/1.0)  
  
Number of Leaves : 4  
  
Size of the tree : 7  
  
  
Time taken to build model: 0.01 seconds  
  
=== Evaluation on training set ===  
  
Time taken to test model on training data: 0 seconds  
  
=== Summary ===  
  
Correctly Classified Instances 22 91.6667 %  
Incorrectly Classified Instances 2 8.3333 %  
Kappa statistic 0.8447  
Mean absolute error 0.0833  
Root mean squared error 0.2041  
Relative absolute error 22.6257 %  
Root relative squared error 48.1223 %  
Total Number of Instances 24   
  
=== Detailed Accuracy By Class ===  
  
 TP Rate FP Rate Precision Recall F-Measure MCC ROC Area PRC Area Class  
 1.000 0.053 0.833 1.000 0.909 0.889 0.974 0.833 soft  
 0.750 0.000 1.000 0.750 0.857 0.845 0.988 0.917 hard  
 0.933 0.111 0.933 0.933 0.933 0.822 0.967 0.972 none  
Weighted Avg. 0.917 0.080 0.924 0.917 0.916 0.840 0.972 0.934   
  
=== Confusion Matrix ===  
  
 a b c <-- classified as  
 5 0 0 | a = soft  
 0 3 1 | b = hard  
 1 0 14 | c = none

### b. Explique o parâmetro: minNumObj. Compare a opção default (2) com 3. Quais as suas considerações?

O parametro “minNumObj” significa o numero minimo de registros por folha. Com o parametro “minNumObj” ajustado para “3” nao houve qualquer mudanca no modelo.

### c. Compare os resultados obtidos com as opções de teste use training set, 10-fold cross-validation e percentage Split 33%. Explique o que aconteceu na interpretação das matrizes de confusão.

Conforme o criterio de divisao da base de dados passou de “training set” para “10-fold cross-validation” houve uma reducao na acuracia (de 91.6667% para 87.5%). A matriz de confusao mostra que para a variavel “soft” nao houve modificacao, ja para a “hard” houve um incremento (zero erro), mas para a “none” o modelo piorou, apresentando dois erros a mais. Ja para o split de 33% o modelo perdeu totalmente a precisao, indo para 25% de acuracia. A quantidade de registros (8) foi muito reduzida, impedindo que fosse prossivel devolver um modelo adequado.

### d. Utilize o método Prism (em rules) para a classificação. Apresente e interprete alguns resultados. Compare com método J48.

Resultados com divisao da base em “cross-validarion Folds 10”

=== Run information ===  
  
Scheme: weka.classifiers.rules.Prism   
Relation: contact-lenses  
Instances: 24  
Attributes: 5  
 age  
 spectacle-prescrip  
 astigmatism  
 tear-prod-rate  
 contact-lenses  
Test mode: 10-fold cross-validation  
  
=== Classifier model (full training set) ===  
  
Prism rules  
----------  
If astigmatism = no  
 and tear-prod-rate = normal  
 and spectacle-prescrip = hypermetrope then soft  
If astigmatism = no  
 and tear-prod-rate = normal  
 and age = young then soft  
If age = pre-presbyopic  
 and astigmatism = no  
 and tear-prod-rate = normal then soft  
If astigmatism = yes  
 and tear-prod-rate = normal  
 and spectacle-prescrip = myope then hard  
If age = young  
 and astigmatism = yes  
 and tear-prod-rate = normal then hard  
If tear-prod-rate = reduced then none  
If age = presbyopic  
 and tear-prod-rate = normal  
 and spectacle-prescrip = myope  
 and astigmatism = no then none  
If spectacle-prescrip = hypermetrope  
 and astigmatism = yes  
 and age = pre-presbyopic then none  
If age = presbyopic  
 and spectacle-prescrip = hypermetrope  
 and astigmatism = yes then none  
  
  
Time taken to build model: 0.01 seconds  
  
=== Stratified cross-validation ===  
=== Summary ===  
  
Correctly Classified Instances 13 54.1667 %  
Incorrectly Classified Instances 7 29.1667 %  
Kappa statistic 0.3204  
Mean absolute error 0.2333  
Root mean squared error 0.483   
Relative absolute error 75.7098 %  
Root relative squared error 123.7086 %  
UnClassified Instances 4 16.6667 %  
Total Number of Instances 24   
  
=== Detailed Accuracy By Class ===  
  
 TP Rate FP Rate Precision Recall F-Measure MCC ROC Area PRC Area Class  
 0.500 0.125 0.500 0.500 0.500 0.375 0.647 0.325 soft  
 0.333 0.118 0.333 0.333 0.333 0.216 0.575 0.208 hard  
 0.769 0.429 0.769 0.769 0.769 0.341 0.667 0.721 none  
Weighted Avg. 0.650 0.321 0.650 0.650 0.650 0.329 0.649 0.565   
  
=== Confusion Matrix ===  
  
 a b c <-- classified as  
 2 1 1 | a = soft  
 0 1 2 | b = hard  
 2 1 10 | c = none

ANALISE O modelo apresenta baixa acuracia, com 13 acertos em 24 registros.

## 6. Abra o arquivo weather.numeric.arff e relate:

* quantos atributos, 5 atributos: outlook (perspectiva climática), temperature (temperatura), humidity (umidade), windy (ventoso) e play (jogar), sendo este último a classe a ser predita. Os atributos “outlook” e “windy” são categóricos, enquanto “temperature” e “humidity” são numéricos (real).
* quantos registros 14 registros
* qual a distribuição de cada atributo. O atributo outlook possui três possíveis valores: sunny (ensolarado), overcast (nublado) e rainy (chuvoso). A temperatura varia entre 64 e 85 graus (possivelmente Fahrenheit), e a umidade varia entre 65% e 96%. O atributo windy é booleano (TRUE/FALSE) indicando a presença ou não de vento. A classe “play” indica se a atividade foi realizada (yes) ou não (no).
* Relate o que você entendeu desta base de dados. Compare com o arquivo da questão 5. Esta base de dados e um conjunto de dados relacionado à prática de alguma atividade ao ar livre com base em condições climáticas. Analisando os dados, podemos perceber alguns padrões: dias nublados (overcast) sempre resultam em “play=yes”, independentemente das outras condições. Já em dias ensolarados (sunny), a umidade parece ter uma forte influência na decisão, com umidade alta geralmente resultando em “play=no”. Em dias chuvosos (rainy), o vento parece ser um fator determinante, com “windy=TRUE” geralmente levando a “play=no”.

### a. Utilize o método Prism (em rules) para a classificação com Use training set como opção de teste.

* O que ocorre? Justifique.

O modelo nao pode ser executado, pois o metodo Prism so pode ser executado com valores nominais. Por isto ha uma base similar chamada “weather.nominal.arff”

Abra o arquivo weather.nominal.arff para continuar respondendo os itens b e c.

### b. Qual(is) o(s) parâmetro(s) deste método? Explique-o(s).

O algoritmo PRISM, conforme implementado no WEKA, possui quatro parâmetros principais:

* numDecimalPlaces - Define o número de casas decimais a serem usadas na saída de números no modelo. Este parâmetro afeta apenas a forma de apresentação dos resultados, não influenciando o funcionamento do algoritmo.
* batchSize - Especifica o número preferido de instâncias a serem processadas durante a predição em lote. Este parâmetro é uma sugestão de tamanho de lote, podendo ser fornecido mais ou menos instâncias, permitindo que as implementações especifiquem um tamanho de lote preferencial.
* debug - Quando definido como verdadeiro, permite que o classificador exiba informações adicionais no console durante sua execução. É útil para fins de depuração e análise detalhada do comportamento do algoritmo.
* doNotCheckCapabilities - Se ativado, as capacidades do classificador não são verificadas antes da construção do modelo. Este parâmetro deve ser usado com cautela pois, embora possa reduzir o tempo de execução, pode levar a problemas se o conjunto de dados não for compatível com as limitações do algoritmo (como a incapacidade de lidar com atributos não nominais ou valores ausentes).

### c. Procure o artigo que originou este método e explique alguns pontos de convergência e divergência deste método com o ID3 e C4.5 estudados em sala.

Com base no artigo original de Jadzia Cendrowska de 1987, e possivel identificar os principais pontos de convergência e divergência entre o PRISM e os algoritmos ID3/C4.5.

Em termos de convergência, tanto o PRISM quanto ID3/C4.5 utilizam uma abordagem baseada em teoria da informação para descobrir regras disjuntivas, agrupando instâncias com características similares. São algoritmos de aprendizado supervisionado que trabalham com atributos categóricos e produzem regras de classificação. Ambos lidam melhor com conjuntos de treinamento sem ruído e trabalham melhor com dados completos.

Quanto às divergências, a primeira e mais fundamental está na estratégia de indução. Enquanto ID3/C4.5 minimizam a entropia média do conjunto de instâncias buscando o atributo mais relevante globalmente, o PRISM maximiza a informação contribuída por um par atributo-valor para uma classificação específica, focando em valores relevantes para cada classe.

A estrutura de saída também difere significativamente: ID3/C4.5 produzem árvores de decisão, enquanto o PRISM produz regras modulares diretas, evitando a estrutura hierárquica. No tratamento de atributos, ID3/C4.5 consideram todos os valores de um atributo ao fazer divisões, enquanto o PRISM seleciona apenas valores de atributos relevantes para cada classe específica.

Em termos de eficiência computacional, o PRISM requer mais esforço por precisar identificar subconjuntos para classes específicas, enquanto ID3/C4.5 são mais eficientes por dividirem o conjunto sem referência a classes específicas. Na relevância de atributos, o PRISM evita termos redundantes nas regras, usando apenas atributos realmente relevantes, enquanto ID3/C4.5 podem incluir atributos irrelevantes devido à estrutura da árvore.

Por fim, quanto à interpretabilidade, o PRISM produz regras mais compreensíveis e modulares, enquanto ID3/C4.5 podem gerar árvores complexas que são mais difíceis de interpretar. O PRISM foi desenvolvido especificamente para abordar estas limitações do ID3, principalmente a questão da interpretabilidade e modularidade das regras geradas, mesmo que isso resulte em maior custo computacional.

## 7. Ainda com o arquivo weather.nominal.arff realize a análise com o método decision table, com opção de teste Use training set e sem alterar qualquer parâmetro.

Explique de forma resumida o resultado.

O método Decision Table aplicado ao conjunto de dados weather.nominal produziu uma tabela de decisão bastante simples com apenas uma regra. Segundo os resultados o modelo classificou corretamente 64,29% das instâncias (9 de 14) e incorretamente 35,71% (5 instâncias).

Analisando a matriz de confusão, podemos observar que o modelo classificou todas as instâncias como “yes” (jogou), acertando as 9 instâncias que realmente eram “yes”, mas errando todas as 5 instâncias que eram “no” (não jogou). Isto explica a taxa de verdadeiros positivos (TP Rate) de 1.0 para a classe “yes” e 0.0 para a classe “no”.

O desempenho geral do modelo não foi muito bom, como indicado pelo valor Kappa de 0 e a área ROC de 0.5, que sugere um desempenho equivalente a uma classificação aleatória. O modelo parece ter se especializado em uma única classe (yes), indicando um possível overfitting ou uma simplificação excessiva das regras de decisão.

O fato de ter gerado apenas uma regra sugere que o modelo pode estar sendo muito simplista na sua abordagem para este conjunto de dados.

### a. Altere o parâmetro displayrules para true, realize a mineração e explique o resultado, conforme o método.

Com a ativação do parâmetro displayrules (opção -R), podemos agora ver a única regra que o modelo Decision Table gerou para o conjunto de dados weather.

Na seção “Rules”, vemos uma tabela extremamente simples com apenas uma coluna “play” e uma única regra que atribui “yes” como classe para todas as instâncias. Isso explica por que o modelo estava classificando todas as instâncias como “yes” no teste anterior.

Este resultado mostra que o algoritmo optou pela estratégia mais simples possível: classificar tudo como a classe majoritária (yes). Isto ocorre porque, no conjunto de dados, existem mais exemplos de “yes” (9) do que “no” (5). O modelo não encontrou nenhuma combinação de atributos que melhorasse significativamente a precisão além dessa regra simples.

A avaliação mostra que esta estratégia resultou em 64,29% de acurácia (9/14 instâncias), mas é claramente uma solução sub-ótima, pois ignora completamente os padrões nos atributos (outlook, temperature, humidity, windy) que poderiam ajudar a distinguir entre as classes “yes” e “no”.

### b. Compare o resultado obtido com o método ID3, com a mesma opção de teste e sem alterar parâmetros.

Comparando os resultados do ID3 com o Decision Table, podemos observar diferenças significativas:

O ID3 gerou uma árvore de decisão muito mais informativa e precisa que utiliza efetivamente os atributos para fazer as classificações. A árvore considera principalmente ‘outlook’ como atributo raiz, e depois usa ‘humidity’ para casos ‘sunny’ e ‘windy’ para casos ‘rainy’, enquanto ‘overcast’ leva diretamente a ‘yes’.

Em termos de desempenho, o ID3 alcançou 100% de acurácia no conjunto de treinamento, classificando corretamente todas as 14 instâncias. Isto é muito superior ao Decision Table, que obteve apenas 64,29% de acurácia com sua regra única de classificar tudo como ‘yes’.

A matriz de confusão do ID3 mostra classificação perfeita: 9 instâncias ‘yes’ e 5 ‘no’ corretamente classificadas. Em contraste, o Decision Table acertou apenas as 9 instâncias ‘yes’ e errou todas as 5 ‘no’.

Todas as métricas de avaliação do ID3 são perfeitas (1.0 ou 100%): Kappa, TP Rate, Precision, Recall, F-Measure, MCC e ROC Area. O Decision Table teve desempenho muito inferior, com Kappa 0 e várias métricas indefinidas devido à sua classificação unilateral.

Esta comparação demonstra claramente a superioridade do ID3 para este conjunto de dados, pois ele conseguiu identificar e utilizar os padrões nos dados para criar regras de classificação mais precisas e informativas.

## 8. Ainda com o arquivo weather.nominal.arff realize a análise com o método ID3 e C4.5 e validação cruzada de 10 partições.

Compare e explique os resultados obtidos.

Ambos os algoritmos geraram estruturas de árvore muito similares, usando ‘outlook’ como nó raiz e os mesmos critérios de divisão. No entanto, o C4.5 fornece informações adicionais como o número de instâncias em cada folha e métricas da árvore (5 folhas e tamanho 8).

O ID3 teve um desempenho significativamente melhor, com 85,71% de acurácia (12 de 14 instâncias corretas), enquanto o C4.5 obteve apenas 50% de acurácia (7 de 14 instâncias corretas).

O Kappa statistic também mostra esta diferença: ID3 obteve 0,6889 (concordância substancial), enquanto C4.5 obteve -0,0426 (concordância pior que aleatória).

Matrizes de Confusão: - ID3: acertou 8 ‘yes’ e 4 ‘no’, com apenas 2 erros totais - C4.5: acertou 5 ‘yes’ e 2 ‘no’, com 7 erros totais

Esta diferença de desempenho é interessante pois o C4.5 é teoricamente uma evolução do ID3. O resultado inferior do C4.5 pode ser devido ao pequeno conjunto de dados, onde suas características adicionais (como poda) podem estar causando overfitting, ou ao próprio processo de validação cruzada com um conjunto tão pequeno de dados.

## 9. Abra o arquivo soybean.arff e relate:

* quantos atributos, A base possui 36 atributos no total, sendo 35 atributos preditivos que descrevem características da plantação de soja e 1 atributo meta (class) que indica o diagnóstico da doença.
* quantos registros e A base contém 683 registros no total.
* qual a distribuição do atributo meta. Em relação ao atributo meta, a base contém 19 classes diferentes de doenças: diaporthe-stem-canker, charcoal-rot, rhizoctonia-root-rot, phytophthora-rot, brown-stem-rot, powdery-mildew, downy-mildew, brown-spot, bacterial-blight, bacterial-pustule, purple-seed-stain, anthracnose, phyllosticta-leaf-spot, alternarialeaf-spot, frog-eye-leaf-spot, diaporthe-pod-&-stem-blight, cyst-nematode, 2-4-d-injury e herbicide-injury.
* Relate o que você entendeu desta base de dados. Esta é uma base de dados voltada para o diagnóstico de doenças em plantações de soja. Os atributos descrevem diferentes características como condições climáticas (data, precipitação, temperatura), histórico da plantação, características das folhas, caule e raízes, estado da planta (normal/anormal), características da doença (lesões, manchas, etc), condições de crescimento e características dos frutos e sementes.

O objetivo desta base é, a partir dessas características observadas, diagnosticar qual doença está afetando a plantação de soja. Trata-se de um problema de classificação multiclasse, onde cada registro representa uma observação de uma planta doente e seus sintomas, sendo classificada em uma das 19 possíveis doenças.

Na sequência utilize o método c4.5 com a opção de teste use Training Set.

### a. Qual o resultado obtido? Avaliando a matriz de confusão, onde o método errou?

Obteve-se uma acurácia de 96.34% (658 instâncias classificadas corretamente e 25 incorretamente).

Observando a matriz de confusão, os erros ocorreram nos seguintes casos:

1. 1 caso de rhizoctonia-root-rot foi classificado erroneamente como diaporthe-stem-canker
2. 1 caso de anthracnose foi classificado como phytophthora-rot
3. 3 casos de phyllosticta-leaf-spot foram classificados como brown-spot
4. 2 casos de brown-spot foram classificados como frog-eye-leaf-spot
5. 10 casos de frog-eye-leaf-spot foram classificados como alternarialeaf-spot
6. 3 casos de alternarialeaf-spot foram classificados como frog-eye-leaf-spot
7. 1 caso de bacterial-pustule foi classificado como bacterial-blight
8. 4 casos de herbicide-injury foram classificados como phytophthora-rot

Os erros mais significativos foram na distinção entre frog-eye-leaf-spot e alternarialeaf-spot (13 casos no total), sugerindo que estas duas doenças compartilham características similares que dificultam sua diferenciação pelo modelo. Também houve uma dificuldade notável em identificar corretamente casos de herbicide-injury, onde metade dos casos foram confundidos com phytophthora-rot.

### b. Utilize 10-fold cross-validation e compare com os resultados obtidos no item a desta questão. Explique o motivo das diferenças. Avaliando esta matriz de confusão, onde o método errou?

Comparando os resultados entre o teste no conjunto de treinamento e o 10-fold cross-validation:

* Training Set: 96.34% de acurácia (658 corretas, 25 incorretas)
* 10-fold CV: 91.51% de acurácia (625 corretas, 58 incorretas)

A diferença na acurácia (aproximadamente 5%) ocorre porque o teste no conjunto de treinamento tende a ser otimista, já que avalia o modelo nos mesmos dados usados para seu treinamento. O 10-fold cross-validation fornece uma estimativa mais realista do desempenho do modelo em dados novos, pois avalia o modelo em dados não vistos durante o treinamento.

Analisando a matriz de confusão do 10-fold CV, os principais erros foram:

1. Confusão entre alternarialeaf-spot e frog-eye-leaf-spot:
   * 20 casos de frog-eye-leaf-spot classificados como alternarialeaf-spot
   * 5 casos de alternarialeaf-spot classificados como frog-eye-leaf-spot
2. Erros na classificação de herbicide-injury:
   * Apenas 3 casos classificados corretamente de 8 total
   * Confundido com phytophthora-rot, anthracnose, diaporthe-pod-&-stem-blight e 2-4-d-injury
3. Outros erros significativos:
   * 4 casos de anthracnose classificados como phytophthora-rot
   * 3 casos de brown-spot classificados como frog-eye-leaf-spot
   * 3 casos de phyllosticta-leaf-spot classificados como brown-spot

O modelo mostrou maior dificuldade especialmente em distinguir entre doenças com características similares, como alternarialeaf-spot e frog-eye-leaf-spot, e teve particular dificuldade com casos de herbicide-injury, sugerindo que esta classe tem características que se sobrepõem com várias outras doenças.

A avaliação com cross-validation revelou mais fraquezas do modelo do que eram aparentes no teste com conjunto de treinamento, demonstrando a importância de usar métodos de validação mais robustos para avaliar o desempenho real do modelo.

## 10. Abra o arquivo labbor.arff e relate:

* quantos atributos A base possui 17 atributos no total, sendo 16 atributos preditivos e 1 atributo meta chamado ‘class’.
* quantos registros e A base contém 57 registros no total
* qual a distribuição do atributo meta. O atributo meta se divide em duas classes possíveis: ‘good’ (bom acordo) e ‘bad’ (acordo ruim).
* Relate o que você entendeu desta base de dados. Esta é uma base de dados que contém informações sobre acordos trabalhistas no Canadá, especificamente relacionados ao setor de serviços pessoais e empresariais para sindicatos com pelo menos 500 membros (como professores, enfermeiros, funcionários universitários, policiais) durante o ano de 1987 e primeiro trimestre de 1988.

Os atributos descrevem diversas características dos acordos trabalhistas, incluindo duração do contrato, aumentos salariais para os três primeiros anos, ajuste do custo de vida, horas de trabalho semanais, plano de pensão, pagamento em sobreaviso, diferencial de turno, auxílio educação, feriados, férias, assistência por invalidez de longo prazo, contribuição para plano odontológico, assistência por luto e contribuição para plano de saúde.

O objetivo desta base é classificar se um acordo trabalhista é considerado bom ou ruim com base nessas características. É um conjunto de dados voltado para análise de negociações coletivas e condições de trabalho, que pode ser útil para sindicatos e empresas avaliarem propostas de acordos trabalhistas, servindo como uma ferramenta de suporte à decisão nas negociações coletivas.

Utilize o método de rede neural Multilayer Perceptron e compare o resultado com o obtido com o C4.5, sempre com a opção 10-fold cross-validation. \* Qual o “melhor método”? \* Quais os critérios que você utilizou para escolher o “melhor”?

Comparando os resultados dos dois métodos com 10-fold cross-validation, o Multilayer Perceptron alcançou uma acurácia de 85,96% (49 instâncias corretas e 8 incorretas), taxa de Verdadeiros Positivos de 0,800 para a classe “bad” e 0,892 para “good”, área ROC de 0,923 e estatística Kappa de 0,6919.

Já o J48 (C4.5) obteve uma acurácia de 73,68% (42 instâncias corretas e 15 incorretas), taxa de Verdadeiros Positivos de 0,700 para “bad” e 0,757 para “good”, área ROC de 0,695 e estatística Kappa de 0,4415.

O método Multilayer Perceptron demonstrou melhor desempenho considerando diversos critérios. Primeiramente, apresentou maior acurácia geral com uma diferença de mais de 12 pontos percentuais. Também mostrou melhor equilíbrio entre as classes, com taxas de Verdadeiros Positivos mais próximas entre si e mais elevadas. A área sob a curva ROC superior indica melhor capacidade de discriminação entre as classes, enquanto o maior valor Kappa demonstra maior concordância além do acaso. Os erros (absoluto médio e quadrático médio) também foram menores para a rede neural.

Embora o Multilayer Perceptron tenha demandado mais tempo computacional para construir o modelo (0,95s contra 0,01s do J48), esta desvantagem é compensada pelo ganho significativo em desempenho. O modelo conseguiu capturar melhor os padrões nos dados, especialmente considerando que existem várias características categóricas e numéricas que podem ter relações não-lineares entre si.

Vale notar que o modelo gerado pelo J48 é mais simples e interpretável, usando apenas dois atributos (wage-increase-first-year e statutory-holidays) em sua árvore de decisão, porém essa simplicidade resultou em menor poder preditivo.

## 11. Resolver a atividade abaixo no Weka.

| # | SEXO | IDADE | Salário | ESPORTE | LAZER | ESTAÇÃO | VIAGEM |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | F | 36 | 5 | Mergulho | Leitura | Verão | Fazenda |
| 2 | M | 21 | 6 | Ciclismo | Cinema | Verão | Praia |
| 3 | F | 28 | 3 | Corrida | Viagem | Inverno | Montanha |
| 4 | M | 27 | 2 | Mergulho | Cinema | Inverno | Fazenda |
| 5 | F | 18 | 9 | Mergulho | Viagem | Verão | Praia |
| 6 | M | 49 | 7 | Ciclismo | Cinema | Inverno | Montanha |
| 7 | F | 42 | 3 | Mergulho | Viagem | Inverno | Montanha |
| 8 | M | 40 | 11 | Ciclismo | Leitura | Inverno | Praia |
| 9 | F | 39 | 21 | Mergulho | Leitura | Verão | Montanha |
| 10 | M | 43 | 12 | Ciclismo | Viagem | Outono | Montanha |
| 11 | F | 44 | 18 | Mergulho | Cinema | Inverno | Praia |
| 12 | M | 59 | 6 | Ciclismo | Viagem | Verão | Montanha |
| 13 | M | 69 | 22 | Corrida | Leitura | Verão | Fazenda |
| 14 | F | 36 | 28 | Corrida | Leitura | Verão | Fazenda |
| 15 | M | 21 | 29 | Ciclismo | Cinema | Outono | Praia |
| 16 | F | 28 | 12 | Corrida | Viagem | Inverno | Montanha |
| 17 | M | 27 | 4 | Mergulho | Cinema | Inverno | Fazenda |
| 18 | F | 18 | 8 | Mergulho | Viagem | Verão | Praia |
| 19 | M | 49 | 7 | Ciclismo | Cinema | Inverno | Montanha |
| 20 | F | 42 | 5 | Mergulho | Viagem | Inverno | Montanha |

Utilizar o método J48 (WEKA) no exemplo acima com validação cruzada de 10 partições e apresentar: \* a árvore gerada no protocolo de experimentos, \* taxas de acertos e \* erros e matriz de confusão.

=== Run information ===  
  
Scheme: weka.classifiers.trees.J48 -C 0.25 -M 2  
Relation: perfil\_viagem  
Instances: 20  
Attributes: 8  
 id  
 sexo  
 idade  
 salario  
 esporte  
 lazer  
 estacao  
 viagem  
Test mode: 10-fold cross-validation  
  
=== Classifier model (full training set) ===  
  
J48 pruned tree  
------------------  
  
idade <= 21: Praia (4.0)  
idade > 21  
| lazer = Leitura: Fazenda (5.0/2.0)  
| lazer = Cinema  
| | salario <= 5: Fazenda (2.0)  
| | salario > 5: Montanha (3.0/1.0)  
| lazer = Viagem: Montanha (6.0)  
  
Number of Leaves : 5  
  
Size of the tree : 8  
  
  
Time taken to build model: 0 seconds  
  
=== Stratified cross-validation ===  
=== Summary ===  
  
Correctly Classified Instances 14 70 %  
Incorrectly Classified Instances 6 30 %  
Kappa statistic 0.5472  
Mean absolute error 0.2278  
Root mean squared error 0.3932  
Relative absolute error 51.6187 %  
Root relative squared error 82.9384 %  
Total Number of Instances 20  
  
=== Detailed Accuracy By Class ===  
  
 TP Rate FP Rate Precision Recall F-Measure MCC ROC Area PRC Area Class  
 0.800 0.133 0.667 0.800 0.727 0.630 0.787 0.517 Fazenda  
 0.667 0.214 0.571 0.667 0.615 0.435 0.750 0.767 Praia  
 0.667 0.091 0.857 0.667 0.750 0.601 0.859 0.799 Montanha  
Weighted Avg. 0.700 0.139 0.724 0.700 0.704 0.558 0.808 0.719  
  
=== Confusion Matrix ===  
  
 a b c <-- classified as  
 4 1 0 | a = Fazenda  
 1 4 1 | b = Praia  
 1 2 6 | c = Montanha

Interprete os resultados.

Com a árvore gerada, classifique os exemplos abaixo:

* 1. Sexo = M, Idade = 33, Salário = 15, Esporte = Mergulho, Lazer = Leitura, Estação = Inverno
  2. Sexo = F, Idade = 45, Salário = 6, Esporte = Ciclismo, Lazer = Leitura, Estação = Verão

Vou analisar os resultados do experimento e realizar a classificação dos novos exemplos.

Interpretação dos Resultados:

A árvore de decisão J48 gerada apresenta uma estrutura relativamente simples, usando principalmente os atributos idade, lazer e salário para tomar decisões. A árvore tem 5 folhas e tamanho 8, indicando uma estrutura compacta.

O modelo alcançou uma taxa de acerto de 70% (14 instâncias classificadas corretamente) e erro de 30% (6 instâncias classificadas incorretamente). O coeficiente Kappa de 0,5472 indica uma concordância moderada entre as previsões e os valores reais.

Analisando a matriz de confusão: - Para classe Fazenda: 4 classificações corretas, com 2 erros - Para classe Praia: 4 classificações corretas, com 2 erros - Para classe Montanha: 6 classificações corretas, com 3 erros

As medidas por classe mostram: - Fazenda: F-Measure de 0,727 - Praia: F-Measure de 0,615 - Montanha: F-Measure de 0,750, sendo a classe com melhor desempenho

Classificação dos novos exemplos:

Para classificar, seguirei o caminho da árvore:

1. Sexo = M, Idade = 33, Salário = 15, Esporte = Mergulho, Lazer = Leitura, Estação = Inverno

* idade > 21: sim
* lazer = Leitura Classificação: FAZENDA

1. Sexo = F, Idade = 45, Salário = 6, Esporte = Ciclismo, Lazer = Leitura, Estação = Verão

* idade > 21: sim
* lazer = Leitura Classificação: FAZENDA

Em ambos os casos, por terem Lazer = Leitura e idade > 21, são classificados como FAZENDA segundo a árvore de decisão gerada.

## 12. Com a mesma base de dados da questão anterior, utilize o OptimizedForest (que utiliza algoritmos genéticos) (se não estiver disponível, instale o pacote), utilize validação cruzada de 10 partições e compare com os resultados obtidos anteriormente. Qual deles demora mais para processar?

Comparando os resultados do J48 e OptimizedForest, notamos diferenças significativas no desempenho. O J48 alcançou uma taxa de acerto de 70% com 14 instâncias classificadas corretamente, enquanto o OptimizedForest obteve 55% com apenas 11 instâncias corretas.

O índice Kappa também demonstra superioridade do J48, que apresentou 0,5472 indicando concordância moderada, contra 0,2713 do OptimizedForest que indica concordância fraca.

Quanto ao tempo de processamento, o J48 foi mais rápido, construindo o modelo instantaneamente (0 segundos), enquanto o OptimizedForest necessitou de 0,83 segundos. Embora ambos sejam tempos relativamente curtos, o OptimizedForest demandou mais recursos computacionais.

A complexidade dos modelos também difere consideravelmente. O J48 gerou uma árvore simples com 5 folhas e tamanho 8, já o OptimizedForest criou uma subestrutura com 86 árvores, resultando em um modelo muito mais complexo.

Na matriz de confusão, o J48 mostrou melhor distribuição de acertos entre as classes: 4 acertos para Fazenda, 4 para Praia e 6 para Montanha. O OptimizedForest teve distribuição menos balanceada: 1 acerto para Fazenda, 3 para Praia e 7 para Montanha.

Para este conjunto de dados específico, o J48 demonstrou ser superior em todos os aspectos analisados: maior acurácia, melhor equilíbrio entre classes, menor tempo de processamento e modelo mais simples. O OptimizedForest, apesar de sua maior complexidade e tempo de processamento mais longo, não conseguiu superar o desempenho do J48.

## 13. Abra o arquivo weather.numeric.arff e utilize os dois métodos: J48 e RepTree.

Compare as duas árvores e os resultados. O que você acha que aconteceu?

Analisando as duas saídas, observa-se um comportamento interessante. O J48 gerou uma árvore de decisão com 5 folhas e 8 nós, usando os atributos outlook, humidity e windy para tomar decisões. Por outro lado, o RepTree gerou uma árvore mínima com apenas um nó, que sempre classifica como “yes”.

Embora ambos os métodos tenham atingido a mesma acurácia de 64,29% (9 instâncias corretas e 5 incorretas), a forma como chegaram a esse resultado é muito diferente:

O J48 tentou criar um modelo mais elaborado, considerando vários atributos e conseguindo identificar tanto casos “yes” quanto “no”, com uma taxa de verdadeiros positivos de 0,778 para “yes” e 0,400 para “no”.

Já o RepTree optou por um modelo extremamente simplificado, classificando todas as instâncias como “yes”. Isso resultou em uma taxa de verdadeiros positivos de 1,000 para “yes” e 0,000 para “no”, pois nunca prevê a classe “no”.

Esta diferença ocorreu provavelmente devido ao tamanho muito pequeno do conjunto de dados (apenas 14 instâncias) e ao desbalanceamento das classes (9 “yes” e 5 “no”). O RepTree, sendo mais conservador em sua construção de árvore, optou por um modelo que simplesmente prevê a classe majoritária, enquanto o J48 tentou criar um modelo mais detalhado, mesmo com poucos dados.

Esta situação ilustra como diferentes algoritmos de árvore de decisão podem tomar abordagens distintas quando confrontados com conjuntos de dados muito pequenos, onde o RepTree preferiu evitar overfitting com uma solução minimalista, enquanto o J48 tentou encontrar padrões mais específicos nos dados.

## 14. Carregue a base diabetes.arff e utilize método de lógica fuzzy, o MultiObjectiveEvolutionaryFuzzyClassifier: MultiObjectiveEvolutionaryFuzzyClassifier (se não estiver disponível, instale o pacote) e compare com os resultados do FURIA. Utilize percentagem split de 66% para este experimento.

Analisando a acurácia geral, o MOEFC alcançou 80,84% (211/261 instâncias) enquanto o FURIA atingiu 80,46% (210/261 instâncias), demonstrando uma melhoria marginal de 0,38% para o MOEFC. As estatísticas Kappa também refletem essa ligeira vantagem, com o MOEFC em 0,5406 comparado a 0,5267 do FURIA, indicando melhor concordância além do acaso.

Ao examinar o desempenho específico por classe, ambos os classificadores tiveram desempenho semelhante para os casos negativos. O FURIA alcançou uma Taxa VP de 0,899, Precisão de 0,829 e Medida-F de 0,863, enquanto o MOEFC mostrou métricas comparáveis com Taxa VP de 0,893, Precisão de 0,837 e Medida-F de 0,864. Para casos positivos, o MOEFC demonstrou recall ligeiramente melhor com Taxa VP de 0,627 comparado a 0,602 do FURIA, embora seus valores de precisão fossem similares (0,732 e 0,735 respectivamente).

A complexidade dos modelos difere notavelmente. O MOEFC gerou 7 regras comparado às 5 regras do FURIA, sugerindo que o MOEFC alcança seu desempenho ligeiramente superior através de uma estrutura de regras mais complexa. No entanto, isso vem com um custo computacional, com o MOEFC levando 5,72 segundos para construir o modelo versus 0,59 segundos do FURIA.

Em relação às taxas de erro, o FURIA mostrou desempenho ligeiramente melhor com RMSE de 0,4144 comparado a 0,4377 do MOEFC. Ambos os classificadores lidaram bem com o desbalanceamento de classes, como evidenciado por seu desempenho similar em ambas as classes positivas e negativas.

## 15. Resolva as questões abaixo no Excel (valor: 30 pontos)

1. Resolver utilizando o ID3 com recursividade no Excel e apresentar as árvores finais:

| ESPORTE PREFERIDO? | PAÍS? | ALIMENTO PREFERIDO | COMPROU NOVO LIVRO DE CULINÁRIA VEGETARIANA? |
| --- | --- | --- | --- |
| Futebol | Brasil | Feijoada | Sim |
| Natação | EUA | Salada | Sim |
| Futebol | França | Macarronada | Não |
| Natação | Brasil | Salada | Não |
| Futebol | Brasil | Macarronada | Não |
| Triatlo | França | Macarronada | Não |
| Triatlo | EUA | Salada | Sim |
| Futebol | EUA | Feijoada | Sim |
| Triatlo | Brasil | Feijoada | Não |
| Triatlo | França | Feijoada | Sim |
| Futebol | Brasil | Macarronada | Não |
| Natação | França | Salada | Sim |
| Futebol | Brasil | Feijoada | Não |

import math  
from collections import Counter  
  
def calculate\_entropy(data, target\_attribute):  
 target\_values = [row[target\_attribute] for row in data]  
 value\_counts = Counter(target\_values)  
 total\_instances = len(data)  
  
 entropy = 0  
 for count in value\_counts.values():  
 probability = count / total\_instances  
 entropy -= probability \* math.log2(probability)  
  
 return entropy  
  
def calculate\_information\_gain(data, split\_attribute, target\_attribute):  
 total\_entropy = calculate\_entropy(data, target\_attribute)  
   
 # Group data by the values of the split\_attribute  
 grouped\_data = {}  
 for row in data:  
 grouped\_data.setdefault(row[split\_attribute], []).append(row)  
  
 split\_entropy = 0  
 total\_instances = len(data)  
  
 for subset in grouped\_data.values():  
 subset\_probability = len(subset) / total\_instances  
 split\_entropy += subset\_probability \* calculate\_entropy(subset, target\_attribute)  
  
 information\_gain = total\_entropy - split\_entropy  
 return information\_gain  
  
def find\_best\_attribute(data, attributes, target\_attribute):  
 best\_attribute = None  
 best\_gain = -float('inf')  
  
 for attribute in attributes:  
 gain = calculate\_information\_gain(data, attribute, target\_attribute)  
 if gain > best\_gain:  
 best\_gain = gain  
 best\_attribute = attribute  
  
 return best\_attribute  
  
def id3(data, attributes, target\_attribute):  
 # Base cases  
 target\_values = [row[target\_attribute] for row in data]  
 if len(set(target\_values)) == 1:  
 return target\_values[0] # Pure node  
  
 if not attributes:  
 # Return the majority class if no attributes are left  
 return Counter(target\_values).most\_common(1)[0][0]  
  
 # Recursive case  
 best\_attribute = find\_best\_attribute(data, attributes, target\_attribute)  
 tree = {best\_attribute: {}}  
  
 grouped\_data = {}  
 for row in data:  
 grouped\_data.setdefault(row[best\_attribute], []).append(row)  
  
 for value, subset in grouped\_data.items():  
 if not subset:  
 # If subset is empty, use majority class of current node  
 tree[best\_attribute][value] = Counter(target\_values).most\_common(1)[0][0]  
 else:  
 remaining\_attributes = [attr for attr in attributes if attr != best\_attribute]  
 tree[best\_attribute][value] = id3(subset, remaining\_attributes, target\_attribute)  
  
 return tree  
  
def print\_tree(tree, depth=0):  
 if not isinstance(tree, dict):  
 print("| " \* depth + f"-> {tree}")  
 return  
   
 for key, subtree in tree.items():  
 if isinstance(subtree, dict):  
 print("| " \* depth + f"[{key}]")  
 print\_tree(subtree, depth + 1)  
 else:  
 print("| " \* depth + f"[{key}] -> {subtree}")  
  
# Dataset  
columns = ["ESPORTE PREFERIDO", "PAÍS", "ALIMENTO PREFERIDO", "COMPROU NOVO LIVRO DE CULINÁRIA VEGETARIANA"]  
data = [  
 {"ESPORTE PREFERIDO": "Futebol", "PAÍS": "Brasil", "ALIMENTO PREFERIDO": "Feijoada", "COMPROU NOVO LIVRO DE CULINÁRIA VEGETARIANA": "Sim"},  
 {"ESPORTE PREFERIDO": "Natação", "PAÍS": "EUA", "ALIMENTO PREFERIDO": "Salada", "COMPROU NOVO LIVRO DE CULINÁRIA VEGETARIANA": "Sim"},  
 {"ESPORTE PREFERIDO": "Futebol", "PAÍS": "França", "ALIMENTO PREFERIDO": "Macarronada", "COMPROU NOVO LIVRO DE CULINÁRIA VEGETARIANA": "Não"},  
 {"ESPORTE PREFERIDO": "Natação", "PAÍS": "Brasil", "ALIMENTO PREFERIDO": "Salada", "COMPROU NOVO LIVRO DE CULINÁRIA VEGETARIANA": "Não"},  
 {"ESPORTE PREFERIDO": "Futebol", "PAÍS": "Brasil", "ALIMENTO PREFERIDO": "Macarronada", "COMPROU NOVO LIVRO DE CULINÁRIA VEGETARIANA": "Não"},  
 {"ESPORTE PREFERIDO": "Triatlo", "PAÍS": "França", "ALIMENTO PREFERIDO": "Macarronada", "COMPROU NOVO LIVRO DE CULINÁRIA VEGETARIANA": "Não"},  
 {"ESPORTE PREFERIDO": "Triatlo", "PAÍS": "EUA", "ALIMENTO PREFERIDO": "Salada", "COMPROU NOVO LIVRO DE CULINÁRIA VEGETARIANA": "Sim"},  
 {"ESPORTE PREFERIDO": "Futebol", "PAÍS": "EUA", "ALIMENTO PREFERIDO": "Feijoada", "COMPROU NOVO LIVRO DE CULINÁRIA VEGETARIANA": "Sim"},  
 {"ESPORTE PREFERIDO": "Triatlo", "PAÍS": "Brasil", "ALIMENTO PREFERIDO": "Feijoada", "COMPROU NOVO LIVRO DE CULINÁRIA VEGETARIANA": "Não"},  
 {"ESPORTE PREFERIDO": "Triatlo", "PAÍS": "França", "ALIMENTO PREFERIDO": "Feijoada", "COMPROU NOVO LIVRO DE CULINÁRIA VEGETARIANA": "Sim"},  
 {"ESPORTE PREFERIDO": "Futebol", "PAÍS": "Brasil", "ALIMENTO PREFERIDO": "Macarronada", "COMPROU NOVO LIVRO DE CULINÁRIA VEGETARIANA": "Não"},  
 {"ESPORTE PREFERIDO": "Natação", "PAÍS": "França", "ALIMENTO PREFERIDO": "Salada", "COMPROU NOVO LIVRO DE CULINÁRIA VEGETARIANA": "Sim"},  
 {"ESPORTE PREFERIDO": "Futebol", "PAÍS": "Brasil", "ALIMENTO PREFERIDO": "Feijoada", "COMPROU NOVO LIVRO DE CULINÁRIA VEGETARIANA": "Não"}  
]  
  
# Attributes and target attribute  
attributes = ["ESPORTE PREFERIDO", "PAÍS", "ALIMENTO PREFERIDO"]  
target\_attribute = "COMPROU NOVO LIVRO DE CULINÁRIA VEGETARIANA"  
  
# Build the decision tree  
decision\_tree = id3(data, attributes, target\_attribute)  
  
# Print the decision tree  
print("Decision Tree:")  
print\_tree(decision\_tree)  
  
# Save the decision tree to a .txt file  
def save\_tree\_to\_file(tree, file\_path):  
 with open(file\_path, "w") as file:  
 def write\_tree(subtree, depth=0):  
 if not isinstance(subtree, dict):  
 file.write("| " \* depth + f"-> {subtree}\n")  
 return  
   
 for key, branch in subtree.items():  
 if isinstance(branch, dict):  
 file.write("| " \* depth + f"[{key}]\n")  
 write\_tree(branch, depth + 1)  
 else:  
 file.write("| " \* depth + f"[{key}] -> {branch}\n")  
   
 write\_tree(tree)  
  
save\_tree\_to\_file(decision\_tree, "./mineracao\_dados/at3/decision\_tree.txt")

Solução:

[PAÍS] | [Brasil] | | [ALIMENTO PREFERIDO] | | | [Feijoada] | | | | [ESPORTE PREFERIDO] | | | | | [Futebol] -> Sim | | | | | [Triatlo] -> Não | | | [Salada] -> Não | | | [Macarronada] -> Não | [EUA] -> Sim | [França] | | [ALIMENTO PREFERIDO] | | | [Macarronada] -> Não | | | [Feijoada] -> Sim | | | [Salada] -> Sim

1. Resolver utilizando o ID3 com recursividade no Excel e apresentar as árvores finais:

| ID | PAÍS | DOCE | FRUTA | IDADE | Compra? |
| --- | --- | --- | --- | --- | --- |
| 1 | Argentina | Bala | Banana | A | Sim |
| 2 | Brasil | Chocolate | Banana | A | Sim |
| 3 | Brasil | Bala | Laranja | C | Não |
| 4 | Brasil | Sorvete | Banana | C | Não |
| 5 | Brasil | Sorvete | Laranja | B | Não |
| 6 | Brasil | Chocolate | Banana | A | Não |
| 7 | EUA | Chocolate | Laranja | A | Sim |
| 8 | EUA | Sorvete | Laranja | B | Sim |
| 9 | EUA | Sorvete | Laranja | B | Sim |
| 10 | Suíça | Chocolate | Laranja | A | Sim |
| 11 | Suíça | Chocolate | Banana | A | Não |
| 12 | Suíça | Bala | Banana | C | Sim |

import math  
from collections import Counter  
  
def calculate\_entropy(data, target\_attribute):  
 target\_values = [row[target\_attribute] for row in data]  
 value\_counts = Counter(target\_values)  
 total\_instances = len(data)  
  
 entropy = 0  
 for count in value\_counts.values():  
 probability = count / total\_instances  
 entropy -= probability \* math.log2(probability)  
  
 return entropy  
  
def calculate\_information\_gain(data, split\_attribute, target\_attribute):  
 total\_entropy = calculate\_entropy(data, target\_attribute)  
   
 # Group data by the values of the split\_attribute  
 grouped\_data = {}  
 for row in data:  
 grouped\_data.setdefault(row[split\_attribute], []).append(row)  
  
 split\_entropy = 0  
 total\_instances = len(data)  
  
 for subset in grouped\_data.values():  
 subset\_probability = len(subset) / total\_instances  
 split\_entropy += subset\_probability \* calculate\_entropy(subset, target\_attribute)  
  
 information\_gain = total\_entropy - split\_entropy  
 return information\_gain  
  
def find\_best\_attribute(data, attributes, target\_attribute):  
 best\_attribute = None  
 best\_gain = -float('inf')  
  
 for attribute in attributes:  
 gain = calculate\_information\_gain(data, attribute, target\_attribute)  
 if gain > best\_gain:  
 best\_gain = gain  
 best\_attribute = attribute  
  
 return best\_attribute  
  
def id3(data, attributes, target\_attribute):  
 # Base cases  
 target\_values = [row[target\_attribute] for row in data]  
 if len(set(target\_values)) == 1:  
 return target\_values[0] # Pure node  
  
 if not attributes:  
 # Return the majority class if no attributes are left  
 return Counter(target\_values).most\_common(1)[0][0]  
  
 # Recursive case  
 best\_attribute = find\_best\_attribute(data, attributes, target\_attribute)  
 tree = {best\_attribute: {}}  
  
 grouped\_data = {}  
 for row in data:  
 grouped\_data.setdefault(row[best\_attribute], []).append(row)  
  
 for value, subset in grouped\_data.items():  
 if not subset:  
 # If subset is empty, use majority class of current node  
 tree[best\_attribute][value] = Counter(target\_values).most\_common(1)[0][0]  
 else:  
 remaining\_attributes = [attr for attr in attributes if attr != best\_attribute]  
 tree[best\_attribute][value] = id3(subset, remaining\_attributes, target\_attribute)  
  
 return tree  
  
def print\_tree(tree, depth=0):  
 if not isinstance(tree, dict):  
 print("| " \* depth + f"-> {tree}")  
 return  
   
 for key, subtree in tree.items():  
 if isinstance(subtree, dict):  
 print("| " \* depth + f"[{key}]")  
 print\_tree(subtree, depth + 1)  
 else:  
 print("| " \* depth + f"[{key}] -> {subtree}")  
  
# Dataset  
columns = ["PAÍS", "DOCE", "FRUTA", "IDADE", "Compra?"]  
data = [  
 {"PAÍS": "Argentina", "DOCE": "Bala", "FRUTA": "Banana", "IDADE": "A", "Compra?": "Sim"},  
 {"PAÍS": "Brasil", "DOCE": "Chocolate", "FRUTA": "Banana", "IDADE": "A", "Compra?": "Sim"},  
 {"PAÍS": "Brasil", "DOCE": "Bala", "FRUTA": "Laranja", "IDADE": "C", "Compra?": "Não"},  
 {"PAÍS": "Brasil", "DOCE": "Sorvete", "FRUTA": "Banana", "IDADE": "C", "Compra?": "Não"},  
 {"PAÍS": "Brasil", "DOCE": "Sorvete", "FRUTA": "Laranja", "IDADE": "B", "Compra?": "Não"},  
 {"PAÍS": "Brasil", "DOCE": "Chocolate", "FRUTA": "Banana", "IDADE": "A", "Compra?": "Não"},  
 {"PAÍS": "EUA", "DOCE": "Chocolate", "FRUTA": "Laranja", "IDADE": "A", "Compra?": "Sim"},  
 {"PAÍS": "EUA", "DOCE": "Sorvete", "FRUTA": "Laranja", "IDADE": "B", "Compra?": "Sim"},  
 {"PAÍS": "EUA", "DOCE": "Sorvete", "FRUTA": "Laranja", "IDADE": "B", "Compra?": "Sim"},  
 {"PAÍS": "Suíça", "DOCE": "Chocolate", "FRUTA": "Laranja", "IDADE": "A", "Compra?": "Sim"},  
 {"PAÍS": "Suíça", "DOCE": "Chocolate", "FRUTA": "Banana", "IDADE": "A", "Compra?": "Não"},  
 {"PAÍS": "Suíça", "DOCE": "Bala", "FRUTA": "Banana", "IDADE": "C", "Compra?": "Sim"}  
]  
  
# Attributes and target attribute  
attributes = ["PAÍS", "DOCE", "FRUTA", "IDADE"]  
target\_attribute = "Compra?"  
  
# Build the decision tree  
decision\_tree = id3(data, attributes, target\_attribute)  
  
# Print the decision tree  
print("Decision Tree:")  
print\_tree(decision\_tree)  
  
# Save the decision tree to a .txt file  
def save\_tree\_to\_file(tree, file\_path):  
 with open(file\_path, "w") as file:  
 def write\_tree(subtree, depth=0):  
 if not isinstance(subtree, dict):  
 file.write("| " \* depth + f"-> {subtree}\n")  
 return  
   
 for key, branch in subtree.items():  
 if isinstance(branch, dict):  
 file.write("| " \* depth + f"[{key}]\n")  
 write\_tree(branch, depth + 1)  
 else:  
 file.write("| " \* depth + f"[{key}] -> {branch}\n")  
   
 write\_tree(tree)  
  
save\_tree\_to\_file(decision\_tree, "./mineracao\_dados/at3/decision\_tree2.txt")

SOLUCAO

[PAÍS] | [Argentina] -> Sim | [Brasil] | | [DOCE] | | | [Chocolate] | | | | [FRUTA] | | | | | [Banana] | | | | | | [IDADE] | | | | | | | [A] -> Sim | | | [Bala] -> Não | | | [Sorvete] -> Não | [EUA] -> Sim | [Suíça] | | [DOCE] | | | [Chocolate] | | | | [FRUTA] | | | | | [Laranja] -> Sim | | | | | [Banana] -> Não | | | [Bala] -> Sim

Agora utilize o Weka com a opção de teste Dados de Treinamento, método C4.5 (J48), execute, visualize a árvore, insira aqui e compare os resultados obtidos.

=== Run information ===  
  
Scheme: weka.classifiers.trees.J48 -C 0.25 -M 2  
Relation: compras  
Instances: 12  
Attributes: 6  
 ID  
 PAIS  
 DOCE  
 FRUTA  
 IDADE  
 COMPRA  
Test mode: 10-fold cross-validation  
  
=== Classifier model (full training set) ===  
  
J48 pruned tree  
------------------  
  
PAIS = Argentina: Sim (1.0)  
PAIS = Brasil: Nao (5.0/1.0)  
PAIS = EUA: Sim (3.0)  
PAIS = Suica: Sim (3.0/1.0)  
  
Number of Leaves : 4  
  
Size of the tree : 5  
  
  
Time taken to build model: 0 seconds  
  
=== Stratified cross-validation ===  
=== Summary ===  
  
Correctly Classified Instances 6 50 %  
Incorrectly Classified Instances 6 50 %  
Kappa statistic -0.1613  
Mean absolute error 0.5189  
Root mean squared error 0.5991  
Relative absolute error 101.1932 %  
Root relative squared error 115.4868 %  
Total Number of Instances 12   
  
=== Detailed Accuracy By Class ===  
  
 TP Rate FP Rate Precision Recall F-Measure MCC ROC Area PRC Area Class  
 0.857 1.000 0.545 0.857 0.667 -0.255 0.314 0.550 Sim  
 0.000 0.143 0.000 0.000 0.000 -0.255 0.314 0.375 Nao  
Weighted Avg. 0.500 0.643 0.318 0.500 0.389 -0.255 0.314 0.477   
  
=== Confusion Matrix ===  
  
 a b <-- classified as  
 6 1 | a = Sim  
 5 0 | b = Nao

1. Resolver pelo C4.5 no Excel: Três faixas para temperatura e duas para umidade, apresentar as árvores sem recursividade com e sem poda. Utilize o pelo método de corte fechando o intervalo no <=

| ID | Tempo | Temperatura | Umidade | Vento | Joga? |
| --- | --- | --- | --- | --- | --- |
| 1 | Ensolarado | 85 | 85 | Não | Não |
| 2 | Ensolarado | 80 | 90 | Sim | Não |
| 3 | Nublado | 83 | 86 | Não | Sim |
| 4 | Chuvoso | 70 | 96 | Não | Sim |
| 5 | Chuvoso | 68 | 80 | Não | Sim |
| 6 | Chuvoso | 65 | 70 | Sim | Não |
| 7 | Nublado | 64 | 65 | Sim | Sim |
| 8 | Ensolarado | 72 | 95 | Não | Não |
| 9 | Ensolarado | 69 | 70 | Não | Sim |
| 10 | Chuvoso | 75 | 80 | Não | Sim |
| 11 | Ensolarado | 75 | 70 | Sim | Sim |
| 12 | Nublado | 72 | 90 | Sim | Sim |
| 13 | Nublado | 81 | 75 | Não | Sim |
| 14 | Chuvoso | 71 | 91 | Sim | Não |

import math  
from collections import Counter  
  
def calculate\_entropy(data, target\_attribute):  
 target\_values = [row[target\_attribute] for row in data]  
 value\_counts = Counter(target\_values)  
 total\_instances = len(data)  
  
 entropy = 0  
 for count in value\_counts.values():  
 probability = count / total\_instances  
 entropy -= probability \* math.log2(probability)  
  
 return entropy  
  
def calculate\_information\_gain(data, split\_attribute, target\_attribute):  
 total\_entropy = calculate\_entropy(data, target\_attribute)  
  
 grouped\_data = {}  
 for row in data:  
 grouped\_data.setdefault(row[split\_attribute], []).append(row)  
  
 split\_entropy = 0  
 total\_instances = len(data)  
  
 for subset in grouped\_data.values():  
 subset\_probability = len(subset) / total\_instances  
 split\_entropy += subset\_probability \* calculate\_entropy(subset, target\_attribute)  
  
 information\_gain = total\_entropy - split\_entropy  
 return information\_gain  
  
def find\_best\_attribute(data, attributes, target\_attribute, thresholds):  
 best\_attribute = None  
 best\_gain = -float('inf')  
 best\_threshold = None  
  
 for attribute in attributes:  
 if attribute in thresholds:  
 for threshold in thresholds[attribute]:  
 data\_split = [{\*\*row, attribute: f"<= {threshold}" if row[attribute] <= threshold else f"> {threshold}"} for row in data]  
 gain = calculate\_information\_gain(data\_split, attribute, target\_attribute)  
 if gain > best\_gain:  
 best\_gain = gain  
 best\_attribute = attribute  
 best\_threshold = threshold  
  
 return best\_attribute, best\_threshold  
  
def c45(data, attributes, target\_attribute, thresholds):  
 target\_values = [row[target\_attribute] for row in data]  
 if len(set(target\_values)) == 1:  
 return target\_values[0] # Pure node  
  
 if not attributes:  
 return Counter(target\_values).most\_common(1)[0][0] # Majority class  
  
 best\_attribute, best\_threshold = find\_best\_attribute(data, attributes, target\_attribute, thresholds)  
  
 if best\_attribute is None:  
 return Counter(target\_values).most\_common(1)[0][0] # Fallback to majority class  
  
 if best\_attribute in thresholds:  
 data = [{\*\*row, best\_attribute: f"<= {best\_threshold}" if row[best\_attribute] <= best\_threshold else f"> {best\_threshold}"} for row in data]  
  
 tree = {best\_attribute: {}}  
  
 grouped\_data = {}  
 for row in data:  
 grouped\_data.setdefault(row[best\_attribute], []).append(row)  
  
 for value, subset in grouped\_data.items():  
 remaining\_attributes = [attr for attr in attributes if attr != best\_attribute]  
 tree[best\_attribute][value] = c45(subset, remaining\_attributes, target\_attribute, thresholds)  
  
 return tree  
  
def print\_tree(tree, depth=0):  
 if not isinstance(tree, dict):  
 print("| " \* depth + f"-> {tree}")  
 return  
  
 for key, subtree in tree.items():  
 if isinstance(subtree, dict):  
 print("| " \* depth + f"[{key}]")  
 print\_tree(subtree, depth + 1)  
 else:  
 print("| " \* depth + f"[{key}] -> {subtree}")  
  
# Dataset  
columns = ["Tempo", "Temperatura", "Umidade", "Vento", "Joga?"]  
data = [  
 {"Tempo": "Ensolarado", "Temperatura": 85, "Umidade": 85, "Vento": "Não", "Joga?": "Não"},  
 {"Tempo": "Ensolarado", "Temperatura": 80, "Umidade": 90, "Vento": "Sim", "Joga?": "Não"},  
 {"Tempo": "Nublado", "Temperatura": 83, "Umidade": 86, "Vento": "Não", "Joga?": "Sim"},  
 {"Tempo": "Chuvoso", "Temperatura": 70, "Umidade": 96, "Vento": "Não", "Joga?": "Sim"},  
 {"Tempo": "Chuvoso", "Temperatura": 68, "Umidade": 80, "Vento": "Não", "Joga?": "Sim"},  
 {"Tempo": "Chuvoso", "Temperatura": 65, "Umidade": 70, "Vento": "Sim", "Joga?": "Não"},  
 {"Tempo": "Nublado", "Temperatura": 64, "Umidade": 65, "Vento": "Sim", "Joga?": "Sim"},  
 {"Tempo": "Ensolarado", "Temperatura": 72, "Umidade": 95, "Vento": "Não", "Joga?": "Não"},  
 {"Tempo": "Ensolarado", "Temperatura": 69, "Umidade": 70, "Vento": "Não", "Joga?": "Sim"},  
 {"Tempo": "Chuvoso", "Temperatura": 75, "Umidade": 80, "Vento": "Não", "Joga?": "Sim"},  
 {"Tempo": "Ensolarado", "Temperatura": 75, "Umidade": 70, "Vento": "Sim", "Joga?": "Sim"},  
 {"Tempo": "Nublado", "Temperatura": 72, "Umidade": 90, "Vento": "Sim", "Joga?": "Sim"},  
 {"Tempo": "Nublado", "Temperatura": 81, "Umidade": 75, "Vento": "Não", "Joga?": "Sim"},  
 {"Tempo": "Chuvoso", "Temperatura": 71, "Umidade": 91, "Vento": "Sim", "Joga?": "Não"}  
]  
  
# Attributes and target attribute  
attributes = ["Tempo", "Temperatura", "Umidade", "Vento"]  
target\_attribute = "Joga?"  
thresholds = {  
 "Temperatura": [65, 75, 85],  
 "Umidade": [70, 80, 90]  
}  
  
# Build the decision tree  
decision\_tree = c45(data, attributes, target\_attribute, thresholds)  
  
# Print the decision tree  
print("Decision Tree:")  
print\_tree(decision\_tree)

SOLUCAO

Decision Tree: [Umidade] | [> 80] | | [Temperatura] | | | [> 75] -> Não | | | [<= 75] -> Sim | [<= 80] | | [Temperatura] | | | [> 65] -> Sim | | | [<= 65] -> Nã

Classificar, utilizando a árvore de decisão após a poda e os seguintes exemplos:

| ID | Tempo | Temperatura | Umidade | Vento | Joga? | Classificação prevista? |
| --- | --- | --- | --- | --- | --- | --- |
| 15 | Ensolarado | 52 | 70 | Sim | Não |  |
| 16 | Ensolarado | 84 | 52 | Não | Sim |  |
| 17 | Chuvoso | 68 | 87 | Não | Sim |  |
| 18 | Ensolarado | 77 | 84 | Não | Não |  |
| 19 | Chuvoso | 86 | 65 | Sim | Sim |  |

def classify(tree, instance):  
 if not isinstance(tree, dict):  
 return tree  
 attribute = next(iter(tree))  
 value = instance[attribute]  
 for condition, subtree in tree[attribute].items():  
 threshold = float(condition.split()[1])  
 if ("<=" in condition and value <= threshold) or (">" in condition and value > threshold):  
 return classify(subtree, instance)  
  
# Novos exemplos  
new\_data = [  
 {"Tempo": "Ensolarado", "Temperatura": 52, "Umidade": 70, "Vento": "Sim", "Joga?": "Não"},  
 {"Tempo": "Ensolarado", "Temperatura": 84, "Umidade": 52, "Vento": "Não", "Joga?": "Sim"},  
 {"Tempo": "Chuvoso", "Temperatura": 68, "Umidade": 87, "Vento": "Não", "Joga?": "Sim"},  
 {"Tempo": "Ensolarado", "Temperatura": 77, "Umidade": 84, "Vento": "Não", "Joga?": "Não"},  
 {"Tempo": "Chuvoso", "Temperatura": 86, "Umidade": 65, "Vento": "Sim", "Joga?": "Sim"}  
]  
  
# Classificar os novos exemplos  
correct = 0  
for instance in new\_data:  
 predicted = classify(decision\_tree, instance)  
 print(f"Instance: {instance} => Predicted: {predicted}, Actual: {instance['Joga?']}")  
 if predicted == instance["Joga?"]:  
 correct += 1  
  
# Taxa de acerto  
accuracy = correct / len(new\_data)  
print(f"Accuracy: {accuracy \* 100:.2f}%")

SOLUCAO

Instance: {'Tempo': 'Ensolarado', 'Temperatura': 52, 'Umidade': 70, 'Vento': 'Sim', 'Joga?': 'Não'} => Predicted: Não, Actual: Não  
Instance: {'Tempo': 'Ensolarado', 'Temperatura': 84, 'Umidade': 52, 'Vento': 'Não', 'Joga?': 'Sim'} => Predicted: Sim, Actual: Sim  
Instance: {'Tempo': 'Chuvoso', 'Temperatura': 68, 'Umidade': 87, 'Vento': 'Não', 'Joga?': 'Sim'} => Predicted: Sim, Actual: Sim  
Instance: {'Tempo': 'Ensolarado', 'Temperatura': 77, 'Umidade': 84, 'Vento': 'Não', 'Joga?': 'Não'} => Predicted: Não, Actual: Não  
Instance: {'Tempo': 'Chuvoso', 'Temperatura': 86, 'Umidade': 65, 'Vento': 'Sim', 'Joga?': 'Sim'} => Predicted: Sim, Actual: Sim  
Accuracy: 100.00%

Qual a taxa de acerto encontrada? 100%

Classificar, utilizando a árvore de decisão após a poda e os seguintes exemplos:

| ID | Tempo | Temperatura | Umidade | Vento | Classificação prevista? |
| --- | --- | --- | --- | --- | --- |
| 20 | Nublado | 84 | 65 | Sim |  |
| 21 | Nublado | 90 | 58 | Sim |  |
| 22 | Ensolarado | 86 | 86 | Sim |  |
| 23 | Ensolarado | 59 | 90 | Sim |  |
| 24 | Chuvoso | 69 | 89 | Sim |  |
| 25 | Chuvoso | 80 | 87 | Sim |  |

# New examples  
new\_data = [  
 {"Tempo": "Nublado", "Temperatura": 84, "Umidade": 65, "Vento": "Sim"},  
 {"Tempo": "Nublado", "Temperatura": 90, "Umidade": 58, "Vento": "Sim"},  
 {"Tempo": "Ensolarado", "Temperatura": 86, "Umidade": 86, "Vento": "Sim"},  
 {"Tempo": "Ensolarado", "Temperatura": 59, "Umidade": 90, "Vento": "Sim"},  
 {"Tempo": "Chuvoso", "Temperatura": 69, "Umidade": 89, "Vento": "Sim"},  
 {"Tempo": "Chuvoso", "Temperatura": 80, "Umidade": 87, "Vento": "Sim"}  
]  
  
# Classify new examples  
for idx, instance in enumerate(new\_data, start=20):  
 predicted = classify(decision\_tree, instance)  
 print(f"ID {idx}: Predicted = {predicted}")

SOLUCAO

ID 20: Predicted = Sim ID 21: Predicted = Sim ID 22: Predicted = Não ID 23: Predicted = Sim ID 24: Predicted = Sim ID 25: Predicted = Não

Agora com a tabela principal utilize o Weka com a opção de teste Dados de Treinamento, método C4.5 (J48), execute, visualize a árvore, insira aqui e compare os resultados obtidos. Na sequência, forneça a base de testes (linhas 20 a 25) e discuta os resultados obtidos.

=== Run information ===  
  
Scheme: weka.classifiers.trees.J48 -C 0.25 -M 2  
Relation: clima\_jogar  
Instances: 14  
Attributes: 6  
 id  
 tempo  
 temperatura  
 umidade  
 vento  
 joga  
Test mode: 10-fold cross-validation  
  
=== Classifier model (full training set) ===  
  
J48 pruned tree  
------------------  
  
tempo = Ensolarado  
| id <= 8: Não (3.0)  
| id > 8: Sim (2.0)  
tempo = Nublado: Sim (4.0)  
tempo = Chuvoso  
| vento = Sim: Não (2.0)  
| vento = Não: Sim (3.0)  
  
Number of Leaves : 5  
  
Size of the tree : 8  
  
  
Time taken to build model: 0 seconds  
  
=== Stratified cross-validation ===  
=== Summary ===  
  
Correctly Classified Instances 7 50 %  
Incorrectly Classified Instances 7 50 %  
Kappa statistic -0.0426  
Mean absolute error 0.4184  
Root mean squared error 0.5967  
Relative absolute error 87.8571 %  
Root relative squared error 120.945 %  
Total Number of Instances 14   
  
=== Detailed Accuracy By Class ===  
  
 TP Rate FP Rate Precision Recall F-Measure MCC ROC Area PRC Area Class  
 0.556 0.600 0.625 0.556 0.588 -0.043 0.689 0.854 Sim  
 0.400 0.444 0.333 0.400 0.364 -0.043 0.689 0.511 Não  
Weighted Avg. 0.500 0.544 0.521 0.500 0.508 -0.043 0.689 0.732   
  
=== Confusion Matrix ===  
  
 a b <-- classified as  
 5 4 | a = Sim  
 3 2 | b = Não