**RELATÓRIO - TRABALHO PRÁTICO**

**APRENDIZAGEM AUTOMÁTICA**

***Python 3 - Árvores de decisão***

***com “pruning”***





**Trabalho realizado por:**

Diogo Castanho, 42496

Pedro Grilo, 43012

**1.1 – Introdução e Objetivo**

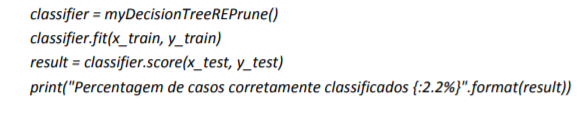
* Uma **árvore de decisão** é uma ferramenta de suporte à tomada de decisão que usa um gráfico no formato de árvore e demonstra visualmente as condições e as probabilidades para se chegar a resultados. É um método de aprendizado de máquina **supervisionado**.
* O método consiste na contínua subdivisão de um espaço amostral (**chamado de raiz**) em classes menores por meio de testes (**chamados de nós**) feitos para subdividir – em dois subespaços para manter uma maior homogeneidade na divisão - esse espaço amostral em classes até que se tenha um subconjunto homogêneo o suficiente para ser classificado como uma mesma classe, criando, assim, um nó terminal (**chamado de folha**).
* Quando árvores de decisão são construídas, muitos ramos ou subárvores podem conter ruídos ou erros. O aprendizado é muito específico ao conjunto de treinamento, não permitindo generalizar para o conjunto de teste (**overfitting**). Para melhorar o modelo, utilizam-se métodos de poda (**pruning**) na árvore, cujo objetivo é melhorar a taxa de acerto do modelo para novas amostras que não foram utilizadas no treinamento. Existem diversas formas de realizar uma poda, e todas elas são classificadas como **pré-poda** ou **pós-poda**.
* Dentre os métodos de poda existentes, destacam-se: **Cost Complexity Pruning, Reduced Error Pruning, Minimum Error Pruning (MEP), Pessimistic Pruning, Error-Based Pruning (EBP), Minimum Description Length (MDL) Pruning, Minimum Message Length (MML) Pruning, Critical Value Pruning (CVP), OPT e OPT-2**.
* Neste trabalho o objetivo é utilizar o método **Reduced Error Pruning** e, dentro dos algoritmos para construção de árvores de decisão será implementado o algoritmo **ID3**.

**1.1 – Introdução e Objetivo**

**(Continuação)**

Assim, considerando os temas e objetivos referidos na página anterior, pretende-se com implementar em Python 3 uma classe que:

* Seja parcialmente compatível com as classes dos classificadores do sklearn;
* Gere uma árvore de decisão de acordo com o algoritmo apresentado nos slides das aulas teóricas (algoritmo ID3) seguida de pruning com o método REP (Reduced Error Pruning);
* Ter uma parametrização para decidir a medida de pureza (opção entre gini, entropia e erro) e indicação se terá ou não pruning;
* Contenha o método fit(x,y) onde seja gerada uma árvore de decisão em função dos dados de treino x e y, em que X é um array bidimensional com conjunto de dados para o treino com a dimensão (n\_samples, n\_features) e y é um array unidimensional com as classes de cada exemplo, com a dimensão (n\_samples);
* Na utilização do método fit, a geração da árvore seja feita com 75% dos dados de treino e o pruning (Reduced Error Pruning) feito com 25%.;
* Apresente o método score(x, y) onde este devolve o valor da exatidão (accuracy) com o conjunto de dados de teste x e y;



**Legenda:** *Exemplo do teste que deverá ser feito à classe no programa;*

**1.2 - Decisões tomadas na realização do trabalho**

Inicialmente, tentou perceber-se o problema em causa (neste caso, o funcionamento das árvores de decisão e a sua construção seguindo os passos e respeitando as regras impostas pelo algoritmo) lendo por isso, mais que uma vez o enunciado e os slides disponibilizados.

Após se perceber bem os objetivos do trabalho, os valores gerados para **x\_test, x\_train**, **y\_test** e **y\_train** pela função utilizada **train\_test\_split()** do sklearn e a manutenção dos dados, começou-se por inicializar a classe principal e outras classes necessárias ao funcionamento do programa e as suas parametrizações;

**1.2.1 – Ficheiros criados e utilizados**

Para organização do trabalho foram criados e utilizados os seguintes ficheiros:

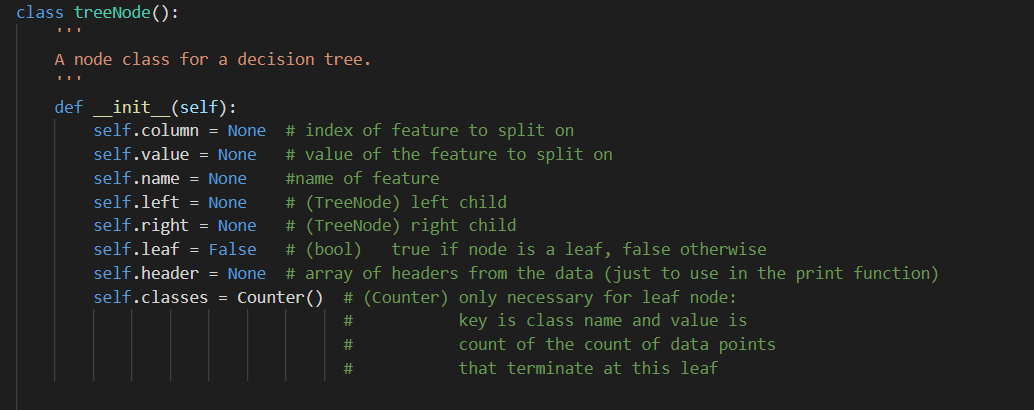
* ***DecisionTreeREPrune.py***
* ***main.py***
* ***treeNode.py***

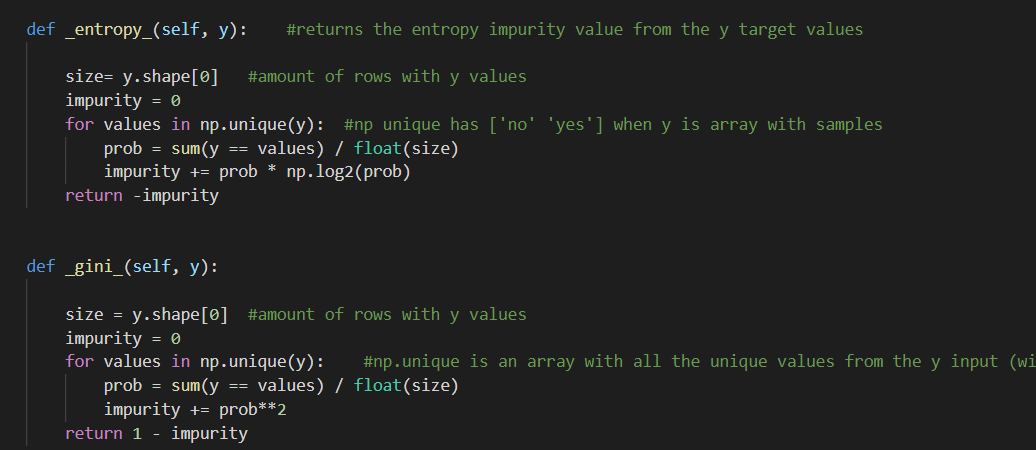
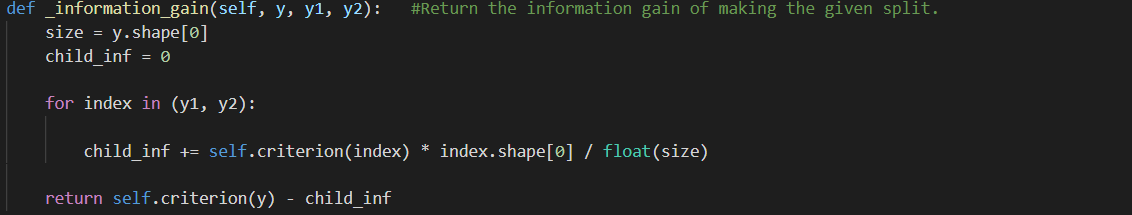
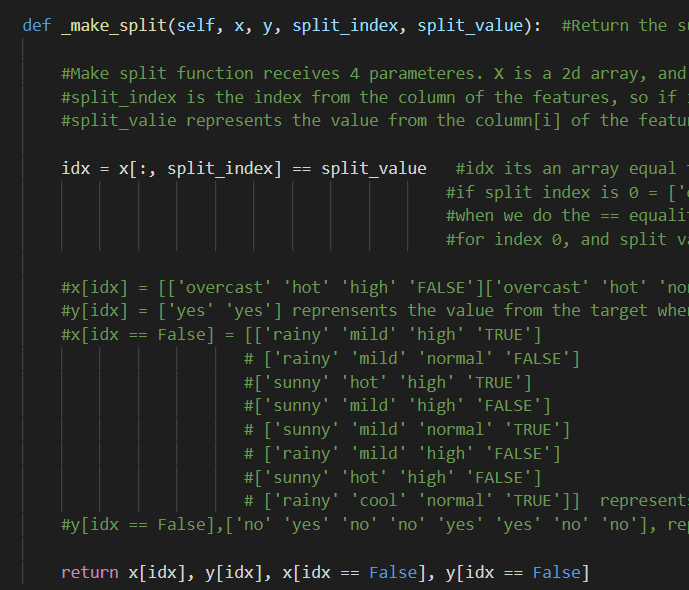
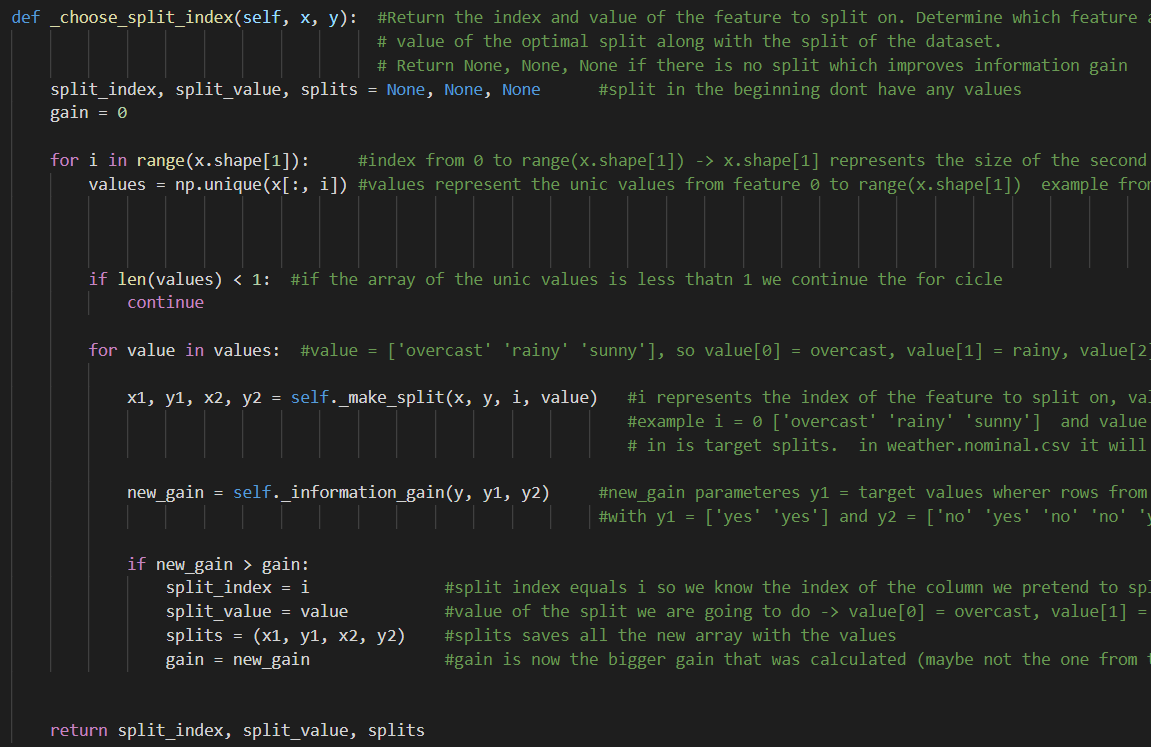
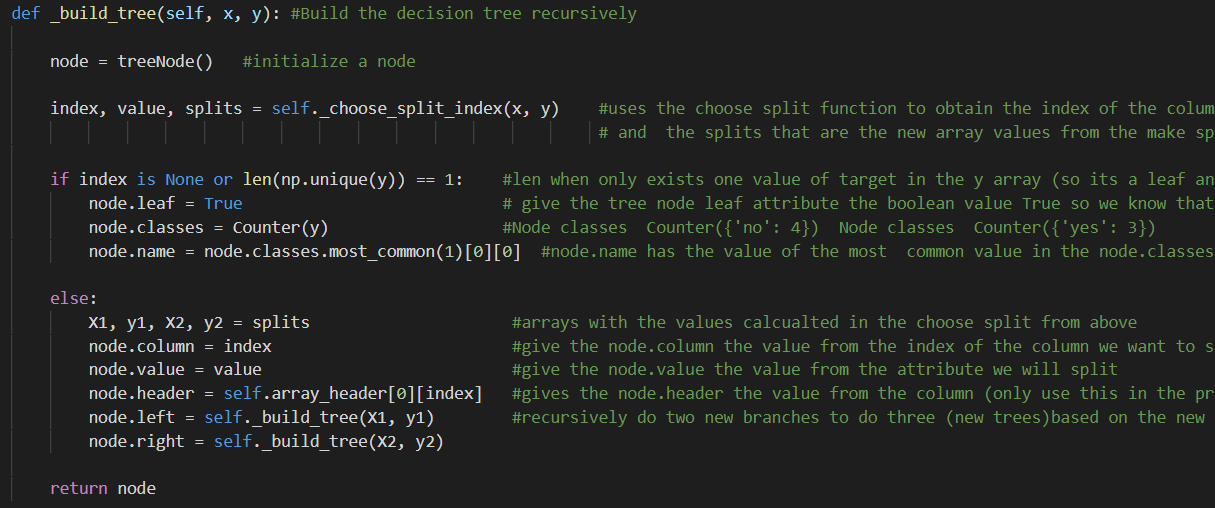
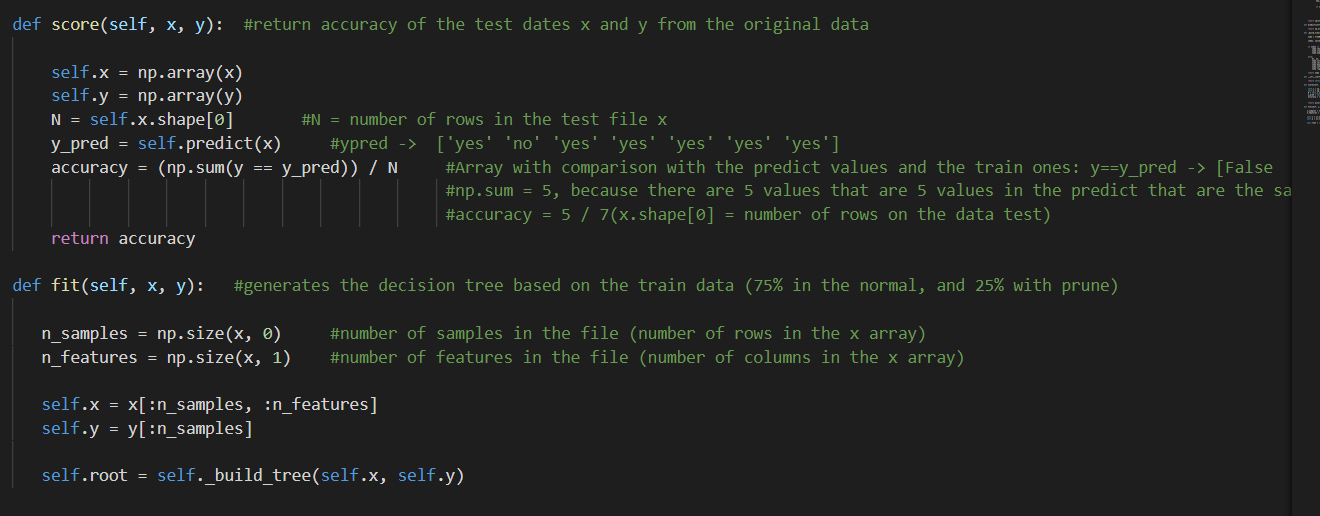
Para testes, foram utilizados os seguintes ficheiros disponibilizados no moodle:

* **“weather.nominal.csv”**
* **“soybean.csv”**
* **“vote.csv”**
* **“contact-lenses.csv”**

**1.3 – Explicação das funções utilizadas**

* Dentro da função main.py apenas temos código para ler os dados, dividi-los usando o train\_test\_split do scikit-learn que nos foi aconselhado pelos professores.
* Dentro da treeNode.py temos:

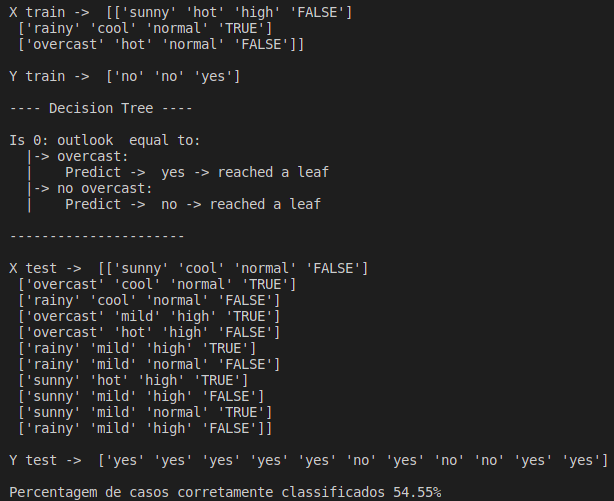


* (cont) onde damos ao nosso Node todos os atributos que vão ser necessários durante a realização do trabalho. Para além disso, temos funções que vão ajudar a dar print na nossa árvore posteriormente, e uma função **predict\_one** que retorna o valor da previsão da classificação de um atributo.
* Onde está a maior parte do código é no ficheiro DecisionTreeREPrune.py (o nosso classifier).
* Temos as funções **gini** e **entropy**, que calculam a pureza do conjunto de dados associados a estas
* Depois temos uma função **information gain** para determinar o gain do atributo que vamos dar “split” (fazer a decisão na árvore):
* A função ***make\_split*** recebe os parâmetros x e y, mas também o split\_index que indica o um valor int da coluna da feature que queremos dar o split, mas também um split\_value com o valor dessa feature. Retorna os novos valores de x,y e y1,y2, que serão usados depois noutras funções.
* A função **choose\_split\_index** retorna os valores do split\_index que vamos querer dividir, o valor desse mesmo index, e os novos valores dos arrays x,y,y1,y2 (usámos as função make\_split para fazer essa divisão, esta função apenas vê dentro do valores que temos, qual o que tem maior gain e é esse mesmo que irá ser dividido com a função).
* De seguida temos a função **build\_tree** que retorna um node, com os atributos column, value, array\_header (apenas usado para dar print posteriormente), isto se não for uma leaf, se for, retorna um valor booleano node.leaf = True e a maior classe existente nessa leaf (ou yes ou no, por isso é que é uma leaf). Recursivamente chama-se novamente para que se continuem a subdividir em outras sub-árvores até que não hajam mais index para dividir.
* Por último, temos as funções fit, que gera a árvore baseado nos valores de x e y que serão dados na main.py (deverão ser os valores x\_train e y\_train, os valores de treino), chamando a self.root que será a construção da nossa árvore.
* A função score calcula a exatidão para um conjunto de dados dado (são os x\_test e y\_test dados da divisão feita na main.py). Usa a função predict que retorna um array com os valores expectáveis.

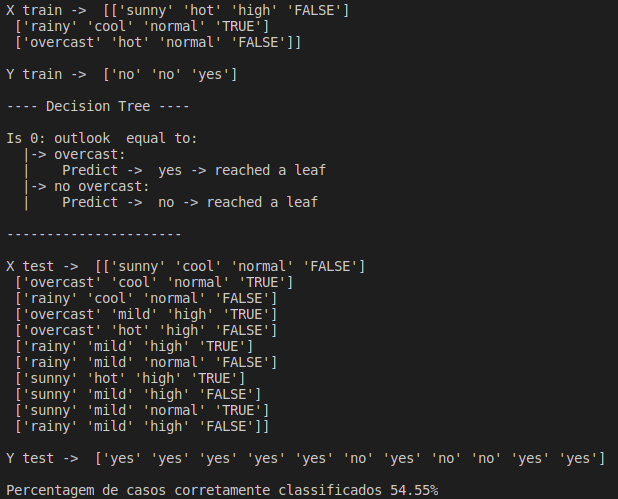
**1.4 – Análise de Dados e Desempenho do Programa**

1.4.1 Análise do desempenho para:

* Weather.nominal.csv com random\_state = 0 e usando “gini” como factor de impuridade



* Weather.nominal.csv com random\_state = 0 e usando “entropy” como factor de impuridade

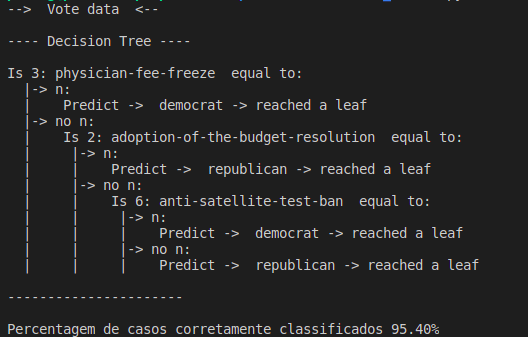


**1.4 - Análise de Dados e Desempenho do Programa**

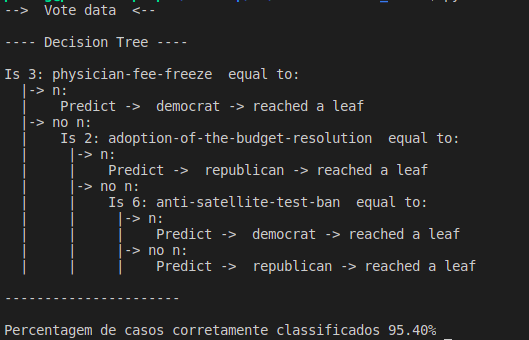
**(Continuação)**

1.4.2 Análise do desempenho para:

* vote.csv com random\_state = 0 e usando “gini” como factor de impuridade



* vote.csv com random\_state = 0 e usando “entropy” como factor de impuridade

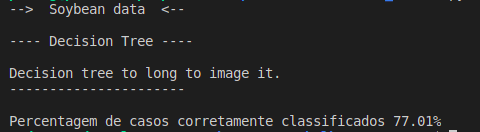


**1.4 - Análise de Dados e Desempenho do Programa**

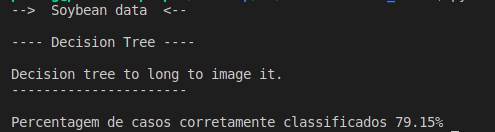
**(Continuação)**

1.4.3 Análise do desempenho para:

* soybean.csv com random\_state = 0 e usando “gini” como factor de impuridade (não mostramos a decision tree e os valores de treino e de teste porque seriam demasiado grandes para caberem numa imagem)



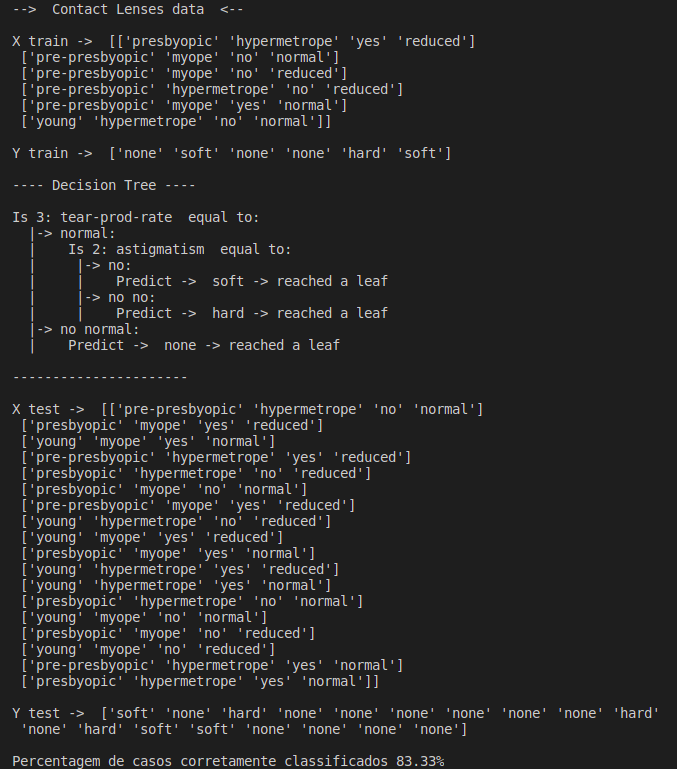
* soybean.csv com random\_state = 0 e usando “entropy” como factor de impuridade (não mostramos a decision tree e os valores de treino e de teste porque seriam demasiado grandes para caberem numa imagem)



**1.4 - Análise de Dados e Desempenho do Programa**

**(Continuação)**

1.4.4 Análise do desempenho para:

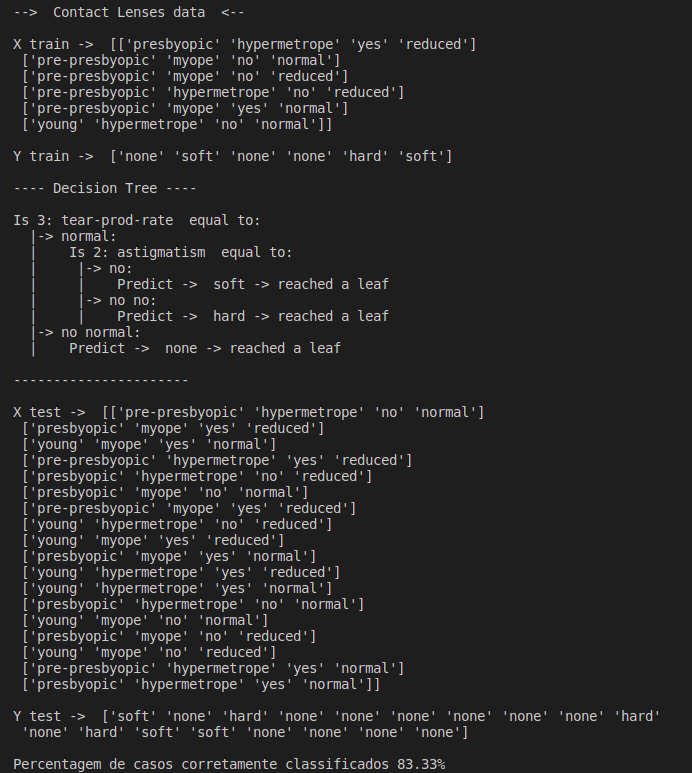
* ****contact-lenses.csv com random\_state = 0 e usando “gini” como factor de impuridade

**1.4 - Análise de Dados e Desempenho do Programa**

**(Continuação)**

1.4.4 Análise do desempenho para:

* contact-lenses.csv com random\_state = 0 e usando “entropy” como factor de impuridade

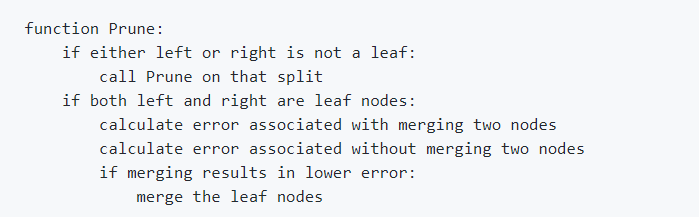


**1.4 - Análise de Dados e Desempenho do Programa**

**(Continuação)**

* Pelas imagens anteriores, conseguimos perceber que os valores da exatidão são iguais quer usando a impuridade gini quer a entropia. Apenas é diferente usando os dados do ficheiro soybean.csv derivado provavelmente de haver uma quantidade muito variada de exemplos.
* Para o ficheiro **weather.nominal.csv** podemos ver que a exatidão é de 54.55%, o que nos diz que provavelmente o modelo usado não está muito bem preparado para os dados de teste usados. É um valor intermediário quando usamos 75% dos dados de teste, o que nos diz que apenas metade destes estão corretamente classificados.
* Já pelo contrário, com o ficheiro **vote.csv**, temos um valor de exatidão de 95.40% para este conjunto de dados de teste, pelo que podemos dizer que o modelo é de facto apropriado a estes dados, pois irá classificar a maioria dos mesmos bem.
* Para o ficheiro **soybean.csv**, usando gini como fator de impuridade temos uma exatidão de 77.01%, o que é relativamente bom, e com entropia temos ainda um valor maior de 79.15%. Podemos dizer que o modelo é apropriado também aos valores de teste dados.
* Por último, para o ficheiro **contact-lenses.csv**, temos um valor de exatidão de 83.33%, conseguindo então prever que o irá também classificar corretamente a maioria dos dados de teste dados.
* Em geral, a árvore de decisão apresentada consegue classificar maioritariamente bem os valores de teste dos nossos dados, pelo que podemos dizer que é bastante apropriada para os ficheiros dados.

**1.5 – Algumas considerações sobre o trabalho**

* Após diversas tentativas de debug e erros no código, o grupo não conseguiu aplicar o “prunning” no algoritmo.
* Apesar de terem sido percebidas as funcionalidades e o objetivo de “prunnning” no algoritmo, não se conseguiu transportar na totalidade para o código o pedido, deixando este um pouco incompleto para uma análise total e correta dos resultados.
* O REP (Reduced error pruning) supostamente dividiria o conjunto de amostras em treinamento e validação. As amostras de treinamento são utilizadas para construir a árvore de decisão. As amostras de validação são utilizadas para verificar os erros de classificação cometidos ao utilizar sub-árvores da árvore gerada.
* Teoricamente, a poda de erro reduzido (REP) tem como objetivo resolver o problema de sobre-ajustamento, minimizando características existentes apenas no conjunto de treinamento.
* Deixou-se ainda assim o código feito e desenvolvido no trabalho para aplicação de “prunning” no ficheiro (mesmo sem este estar funcional, e usando a fórmula seguinte como inspiração para o mesmo).

**1.4 – Breve Conclusão**

* A realização deste trabalho permitiu-nos um maior contacto com a linguagem python, a qual era nova para nós, deixando-nos com uma melhor noção de todas as vantagens (e desvantagens) de usar uma linguagem como esta. Por esta razão, foi-nos um pouco difícil começar o trabalho derivado a isto, mas foi uma dificuldade ultrapassada.
* Conseguimos implementar a árvore de decisão pedida, não conseguindo fazer posteriormente o prune, por termos alguns erros quando tentávamos compilar.
* Posto isto, achamos que o trabalho foi bem conseguindo, e abriu-nos os horizontes para tudo aquilo que é a aprendizagem automática para grandes previsões, e como esta pode ser muito vantajosa em diversas áreas que possam existir.
* Também aprendemos a mexar com a interface Kaggle, que foi aconselhada pela professora.

**1.5 – Bibliografia**

* Slides no moodle
* <https://en.wikipedia.org/wiki/Decision_tree>
* <https://scikit-learn.org/stable/modules/tree.html>
* <https://www.geeksforgeeks.org/decision-tree/>
* Entre outros fóruns, vídeos, etc existentes