Introdução

No âmbito da UC de Aprendizagem Automática foi-nos proposta a realização de um trabalho que consistiu na implementação de um algoritmo de árvore de decisão, bem como a implementação de funções de cálculo da impureza dos nós da árvore e estratégias de pré e pós poda da árvore criada.

Foi pedida a implementação das seguintes componentes:

- Implementação do algoritmo ID3 (Iterative Dichotomiser 3);
- Implementação de duas funções de impureza;
- Implementação de dois métodos de pré-poda;
- Implementação de um método de pós-poda.

Todas estas componentes foram implementadas em Python3.

Desenvolvimento e Solução Encontrada

De modo a criar um programa que seja fiel ao algoritmo e que contenha e cumpra tudo o que foi pedido, foram criados os seguintes pontos:

- O algoritmo que representa a criação da árvore e crescimento da mesma;
- As duas funções de impureza (índice de Gini e entropia);
- Os dois métodos de pré-poda (profundidade máxima e número mínimo de instâncias por nó);
- O método de pós-poda (reduced-error pruning).

Todos estes aspetos encontram-se divididos em três ficheiros:

- **main.py**: ficheiro principal que permite a execução do programa e onde é definido o ficheiro de dados e as configurações do algoritmo;
- **decision_tree.py**: ficheiro que alberga todo o código da árvore de decisão e dos seus nós;
- **tree_utils**.py: ficheiro auxiliar que contém todas as funções secundárias da árvore de decisão.

Processamento dos dados

Relativamente ao funcionamento do programa, ao iniciar o ficheiro (depois de ser especificado um ficheiro .csv que contenha os dados, estes são divididos em x e y (onde x são os atributos e valores das instâncias, e y a classificação por classes das várias instâncias). Existe depois outra divisão dos dados em dois conjuntos, que são os seguintes:

- **Conjunto de treino** (utilizado para criar e treinar o modelo);
- **Conjunto de teste** (utilizado para testar o modelo criado).

Criação da árvore

Depois das divisões, é criada uma instância da classe DecisionTree, que cria uma árvore de decisão com os seguintes parâmetros (com valores por defeito caso não sejam passados certos argumentos):

- **impurity_function** (indicador da função utilizada para calcular a impureza, sendo o índice de Gini a função utilizada por defeito);
- **max_depth** (profundidade máxima da árvore, sendo infinito o valor escolhido por defeito);

- **min_number_of_instances_per_node** (número mínimo de instancias por nó, sendo 1 o valor escolhido por defeito);
- **post_pruning** (decisão de fazer ou não pós-poda à árvore, sendo “false” o valor escolhido por defeito).

Construção da árvore e pré-poda

Depois de serem definidos todos os parâmetros da árvore, é chamado o método **fit()** com os dados de treino da árvore. Neste método, é calculado o número de instâncias do nó raíz e é chamado o método **grow_tree()** (com a união das componentes x e y de treino, bem como a raíz da árvore).

No método **grow_tree()**, são primeiro verificadas as duas condições de pré-poda. Caso as condições de pré-poda forem verdadeiras, a construção da árvore está completa, senão o crescimento da árvore continua. Caso o crescimento continue, é verificado se os dados a acrescentar são homogéneos (isto é, todos os elementos do conjunto passado como argumento tenham a mesma classe). Se o conjunto for homogéneo, é definida a classe do nó como a classe do conjunto e a construção termina. Caso o conjunto não for homogéneo, são calculados o atributo do nó e a melhor partição, calculando a impureza para todos os atributos do conjunto de dados, e escolhido o subconjunto que tenha o maior grau de pureza. Caso haja melhor partição (e esta não seja vazia), é criado um nó filho, e é adicionado ao dicionário que contém os nós que são filhos da raíz da árvore. Depois disto o método grow_tree() é chamado novamente, mas desta vez com o nó filho e os dados que possam chegar a esse nó.

Após todo este processo estar concluído para todos os dados de treino, a árvore está construída, e o modelo está construído e treinado.

Pós-poda

Após a construção do modelo, é feita a pós-poda, caso o argumento **post_pruning** tenha sido passado como ”true”. Caso haja pós-poda, é feita uma divisão do conjunto de treino dois conjuntos:

- **Conjunto de treino;**
- **Conjunto de poda.**

Esta divisão é feita quando é criada a instância da **DecisionTree**.

Para a pós-poda ser feita, é chamado o método **reduced_error_pruning()**, que recebe como argumento o **conjunto de poda**. Neste algoritmo, são criadas várias sub árvores da árvore de decisão onde cada uma tem um nó removido (os nós removidos são apenas os nós que têm apenas folhas como filhos). É depois calculado o número de instâncias bem classificadas (calculado com o método **pruning_score()**) para cada uma das árvores. Por fim, é calculado o número de instâncias bem classificadas da árvore principal. Caso alguma das árvores podadas tenha um score melhor ou igual ao da árvore principal, essa árvore é escolhida e o método **reduced_error_pruning()** é chamado novamente (este processo repete-se até que deixem de existir árvores com melhores scores que a árvore principal).

No final de ser selecionada a melhor árvor, é chamado o método **predict()** para classificar os dados de teste.

Exatidão e Score

É depois calculado o score final. O score final compara os valores previstos com os reais, e calcula a exatidão com a seguinte fórmula:

$$Exatidão = \frac{True\ Positives\ +\ True\ Negatives}{Total}$$

O **score final**, calculado pela média da exatidão de cada uma das classes, é depois apresentado ao utilizador.

Conclusões Finais e Problemas Encontrados

Com a realização deste trabalho foi possível concluir os seguintes pontos:

- É necessário treinar um algoritmo com um bom conjunto de treino de modo a construir um modelo fiável;
- As funções de impureza são importantes para escolher a criação de uma árvore de decisão;
- Os métodos de poda (tanto pré como pós) reduzem o tamanho da árvore e reduzem assim o sobreajustamento do modelo.

Foram encontrados os seguintes problemas durante a criação do algoritmo:

- Problemas com conjuntos de dados muito pequenos, pois nem sempre o conjunto de treino possuía todos os valores possíveis para cada atributo, gerando uma exceção no programa;
- Conjuntos de dados, tais como o conjunto de dados do ficheiro “dropout-nominal.csv”, necessitam de uma melhor atenção pois estes possuem atributos cujo valor é único, não podendo ser tirada nenhuma conclusão por isso (uma solução possível seria remover essa coluna e não a considerar para a criação do modelo).