Relatório do Projeto 2 de IA

Grupo 11: Afonso Carvalho 93681, Mónica Jin 92532

Problemas do algoritmo base descrito na figura 18.5 do capítulo 18 do AIMA:

1. Encontrar a melhor maneira para classificar a importância de um atributo.
2. Encontrar a solução mais curta, *i.e.*, devolver a árvore de decisão mais curta.
3. Lidar com o ruído e evitar situações de *overfitting* e *underfitting*.

Soluções para os Problemas:

1. De modo a escolher o melhor atributo de teste para dividir as classificações dos exemplos de treino, usou-se a Entropia como quantidade de medição de informação, o que mede a incerteza de dado atributo. Para obtermos a relevância de cada atributo, foi aplicado o método *Information Gain* que usa o conceito de entropia, de maneira a quantificar o ganho de informação de cada atributo.

   Expressão da entropia para valores binários: $B(q) = -(q \log_2 q + (1-q)\log_2(1-q))$ onde *q* é a probabilidade de a classificação ser verdadeira ou falsa.

   Expressão do ganho de informação de um atributo:

   $$Gain(A) = B(\frac{p}{p+n}) - \sum_{k=1}^{d} \frac{p_k+n_k}{p+n} B(\frac{p_k}{p_k+n_k})$$

   p e n são os números de classificações positivas e negativas, respetivamente.

   pk e nk são os números de classificações positivas e negativas para um dado subconjunto, em que o atributo A toma um valor do seu domínio com d valores.

   Em cada iteração do algoritmo *decisionTreeLearning*, escolhe-se o atributo com maior ganho de informação e em caso de empate escolhe-se o primeiro da lista de atributos.

2. Para devolver a árvore de decisão mais curta, foi elaborado a função *cutTree* que procura recursivamente as similaridades entre os ramos da árvore e compacta-a se possível. Se for detetada igualdade entre os ramos da esquerda e direita do mesmo nó, então um destes ramos passa a ser a raiz da subárvore em análise.

3. Para lidar com os dados ruidosos, foram testados métodos de *pre-pruning*, de *cross-validation* e híbrido.
   O método de *pre-pruning* usado consistiu em limitar a expansão da árvore de decisão nos casos onde o ganho de informação de todos os atributos é inferior ou igual a 0.05. Nestes casos, o algoritmo *decisionTreeLearning* apenas devolve a classificação maioritária dos exemplos fornecidos. Este método foi gerado de modo a evitar *overfitting* nas situações onde não há qualquer ganho de informação e a constante não é 0 para evitar *underfitting*. Assim, a constante 0.05 foi decidida de forma empírica através de execuções de testes com diferentes valores de limite.

O método de *Cross-validation*, consiste em criar árvores diferentes a partir de subconjuntos de treino diferentes. Para isto, cada conjunto de dados fornecido foi partido em 2 subconjuntos, um de treino e outro de teste. O critério de seleção da árvore de decisão foi a que tinha menor erro entre as classificações teste e as classificações obtidas a partir da árvore gerada.
Foram efetuados testes com este método com diferentes percentagens de partição dos dados: 50% set treino, 50% set de teste; 80% set treino com 5 combinações diferentes, 20% set de teste; 90% set treino com 10 combinações diferentes, 10% set de teste.
Para além de testarmos cada método separadamente, também testámos a combinação dos dois (método híbrido).

Análise e conclusão dos resultados obtidos

Método *pre-pruning*:

Ao analisar os gráficos e tabelas 1 e 2 verificámos que para valores inferiores a 0.01 nota-se um aumento da árvore gerada e um aumento ligeiro do erro, devido a *overfitting* (apresentado nas entradas a azul). Para valores superiores a 0.08 verificámos o oposto, onde alguns testes geravam árvores significativamente mais pequenas, e consequentemente um incremento substancial do erro, devido a *underfitting* (apresentado nas entradas a vermelho). Para valores dentro do intervalo [0.02, 0.08] o erro e o comprimento mantêm-se. Portanto, optámos por escolher um valor limite que se enquadrasse no meio deste intervalo, 0.05.

Método cross-validation:

Com este método, passávamos apenas a 2 teste com ruído e os erros obtidos eram relativamente grandes em relação ao método mencionado anteriormente.

Método híbrido:

Todos os erros obtidos com este método foram iguais a zero (tabela 3, 4 e 5). O tempo de execução e o comprimento da árvore variam dependendo do set de treino.

Método híbrido *vs* Método *pre-pruning*:

Ambos os métodos diminuíram o erro de cada teste do script, e a margem de erro do método híbrido era bastante similar, no entanto os tempos de execução eram mais extensos (tabela 6). Portanto, seguindo princípio *Ockham's Razor* optámos pela solução mais simples e mais eficiente em termos de tempo de execução, o método de *pre-pruning*.

Anexos

Limit	Teste 1	Teste 2	Teste 3	Teste 4
0	0	0,011	0,0094	0,0025
0,01	0	0,0001	0	0
0,02	0	0	0	0
0,03	0	0	0	0
0,04	0	0	0	0
0,05	0	0	0	0
0,06	0	0	0	0
0,07	0	0	0	0
0,08	0	0	0	0
0,09	0	0,38	0	0
0,1	0,4367	0,38	0	0

Tabela 1

Limit	Teste 1	Teste 2	Teste 3	Teste 4
0	3145	9578	8404	6083
0,01	49	287	12	11
0,02	49	90	57	57
0,03	49	90	57	57
0,04	49	90	57	57
0,05	49	90	57	57
0,06	49	90	57	57
0,07	49	90	57	57
0,08	49	90	57	57
0,09	49	9	57	57
0,1	9	9	57	57

Tabela 2

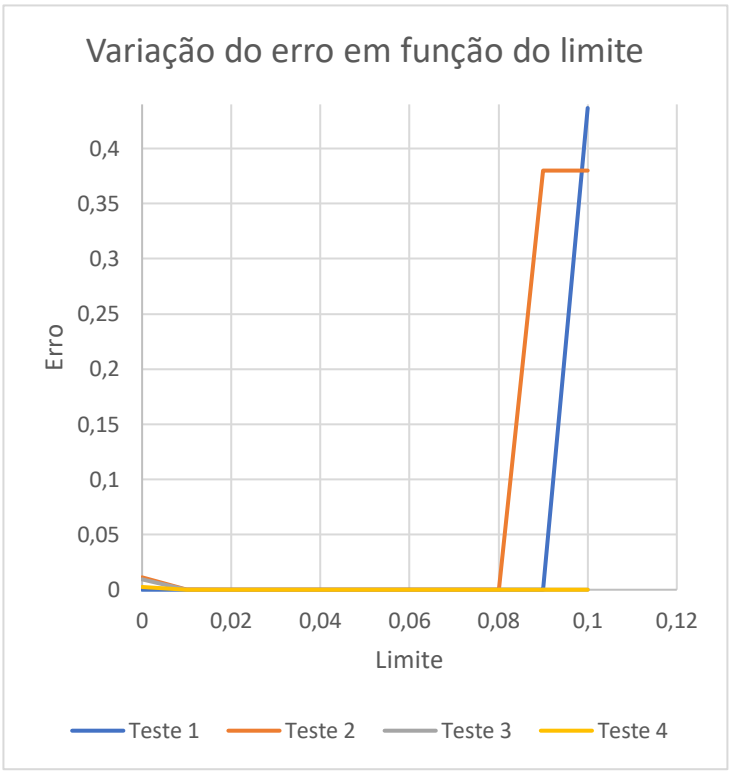

Gráfico 1

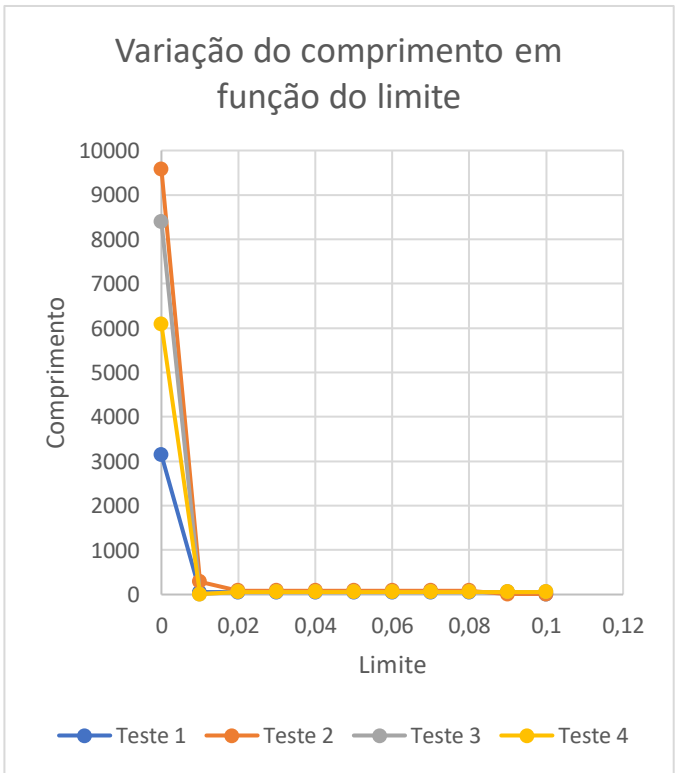

Gráfico 2

Lado esquerdo (Tabela 1 e Gráfico 1): Tabela e gráfico com os erros dos 4 testes do script em função da constante limite definida.

Lado direito (Tabela 2 e Gráfico 2): Tabela e gráfico com o comprimento de cada árvore gerada nos 4 testes do script em função da constante limite definida.

	Teste 1	Teste 2	Teste 3	Teste 4
erro	0	0	0	0
tempo	0,3611	0,4138	0,3631	0,3645
comp	49	58	57	57

Tabela 3: Erro, tempo de execução e comprimento da árvore gerada para cada teste do script com o método híbrido de 2 partições (50% set treino, 50% set de teste).

	Teste 1	Teste 2	Teste 3	Teste 4
erro	0	0	0	0
tempo	1,4232	1,347	1,0765	1,1257
comp	49	90	57	57

Tabela 4: Erro, tempo de execução e comprimento da árvore gerada para cada teste do script com o método híbrido de 5 partições (80% set treino com 5 combinações diferentes, 20% set de teste).

	Teste 1	Teste 2	Teste 3	Teste 4
erro	0	0	0	0
tempo	2,1653	3,0533	2,5158	2,4924
comp	49	90	57	57

Tabela 5: Erro, tempo de execução e comprimento da árvore gerada para cada teste do script com o método híbrido de 10 partições (90% set treino com 10 combinações diferentes, 10% set de teste).

	Teste 1	Teste 2	Teste 3	Teste 4
pre-pru	0,3246	0,38	0,3331	0,3271
hibrido	2,1653	3,0533	2,5158	2,4924

Tabela 6: Tempo de execução dos métodos *pre-pruning* e híbrido (com 10 partições) testados em segundos.

Referências

Russel, S., Norvig, P., (2009). Artificial Intelligence: A Modern Approach. Prentice Hall.

https://towardsdatascience.com/do-not-use-decision-tree-like-this-369769d6104d

https://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15381-s06/www/DTs2.pdf