LEIC-A 2020 *Duarte Bento 92456*
Grupo 89 *Susana Monteiro 92560*

Inteligência Artificial - Projeto 2

O objetivo do 2º projeto de Inteligência Artificial consiste em construir e analisar árvores de decisão, começando pela implementação de um algoritmo base e, a partir deste, tentar obter árvores mais simples (mais curtas) e árvores que produzam resultados pouco erráticos na presença de ruído.

Algoritmo Base:

Deste modo, começou-se por implementar o algoritmo base, o qual produz árvores corretas para todos os exemplos testados, na ausência de ruído. Resumidamente, este algoritmo consiste em inferir uma árvore de decisão a partir de um conjunto de exemplos e o respetivo resultado. Em cada nível de profundidade da árvore, é feita a escolha do melhor atributo para usar como nó com recurso a um conceito chamado ganho de informação (maximizar a quantidade de informação aprendida com a escolha de um atributo).

Ruído:

O próximo passo foi criar uma forma de lidar com testes que incluam ruído. Notou-se que usando um critério de desempate na classificação das folhas, para os casos onde a contagem de casos positivos e negativos era igual, que atribuísse uma classificação positiva, então os teste de ruído disponibilizados eram resolvidos. Sabendo que este critério não resolveria todos os casos, decidiu-se implementar um algoritmo para redução de ruído - validação cruzada com k iterações.
Este algoritmo consiste em, para um valor arbitrário k, subdividir o conjunto total de exemplos de treino em k subconjuntos e deles escolher um para servir de conjunto de testes enquanto os restantes (k-1)/k exemplos são usados para treinar o modelo. Gera-se uma hipótese de árvore para este conjunto de treino e mede-se a percentagem de exemplos erradamente classificados (erro). Itera-se k vezes sobre o processo anterior, escolhendo um subconjunto diferente e seleciona-se a árvore que resultar no menor erro, a árvore pretendida.
É de notar que a solução implementada depende de uma escolha aleatória dos atributos a adicionar a cada subconjunto. Optou-se por seguir esta estratégia com o objetivo de distribuir uniformemente os exemplos incoerentes por subconjunto de forma a evitar um potencial agrupamento de exemplos com ruído no conjunto de treino. Além disso, de forma a mitigar alguma inconsistência causada pela aleatoriedade, os subconjuntos são gerados de forma a serem disjuntos, isto é, cada exemplo pertence apenas a um subconjunto. Portanto, ao iterar por todos os subconjuntos, percorre-se a totalidade dos exemplos de treino fornecidos.

Árvores mais curtas:

Por fim, faltava encontrar árvores suficientemente pequenas. A solução para este problema passa por duas partes. Inicialmente observou-se que, por vezes, existem vários atributos com um ganho de informação máximo e, com o algoritmo base, a escolha do próximo atributo nestes casos não era criteriosa. Assim, a primeira parte da solução consiste em alterar a ordem dos atributos, consequentemente alterando aquele que é escolhido quando existem vários atributos com o ganho de informação máximo. A primeira tentativa passou por criar uma lista com todas as permutações possíveis das ordens dos atributos, inferir a árvore para cada uma destas ordens e devolver a mais curta. Esta estratégia funcionava bem para testes com poucos atributos, mas para um número superior de atributos, o número de árvores a calcular tornava-se absurdo (por exemplo, para um teste com 12 atributos é necessário calcular 12! = 479001600 árvores de decisão). Assim, esta solução era suficiente para passar um dos testes feitos às “shortest trees”, mas não chegava sequer a devolver uma solução para o segundo destes testes (por exceder a quantidade de memória disponível). A segunda estratégia, aquela que se escolheu usar na entrega, implica também calcular várias árvores, mas apenas para aqueles atributos que a determinado ponto da execução do algoritmo apresentaram simultaneamente o mesmo ganho de informação máximo (isto é, para aqueles atributos que foram preteridos em relação a outro ainda que tenham tido o mesmo ganho que eles). Assim evita-se que um atributo seja escolhido em relação a outro apenas por ter um menor índice.

Mesmo com esta melhoria de eficiência, a primeira parte da solução não foi suficiente para passar ao segundo teste das “shortest trees”. Por esta razão, esboçou-se a árvore devolvida pelo algoritmo base para tentar compreender como seria possível reduzir o seu tamanho. Verificou-se que a árvore desenhada tinha 2 ramos iguais e que eliminando um destes ramos era possível obter uma árvore mais compacta (de menor dimensão) - Figura 1. Para tal, criou-se uma função que recursivamente procura ramos iguais (começando nas folhas) e, se os encontrar, simplifica esse nível da árvore de acordo com o exemplo da figura (truncando os ramos iguais num só, concatenando os dois diferentes e alterando a ordem dos nós).

Analisando o resultado obtido com esta solução de duas partes, conclui-se que em alguns casos é possível obter árvores bastante mais simples do que aquela obtida com o algoritmo inicial, apenas por tornar a escolha de atributos mais completa e por retirar ramos repetidos da árvore. É de notar que se garante que a árvore devolvida com esta solução é sempre equivalente (produz os mesmos resultados) àquela devolvida pelo algoritmo inicial, mas não se garante que devolva sempre a árvore mais simples possível. No entanto, é suficiente para provar que o algoritmo base, ainda que tenha em conta o ganho de informação de cada atributo (e mesmo que tenha também em conta a primeira parte da solução explicada), não devolve necessariamente a árvore mais simples possível.

Duarte Bento 92456
Susana Monteiro 92560

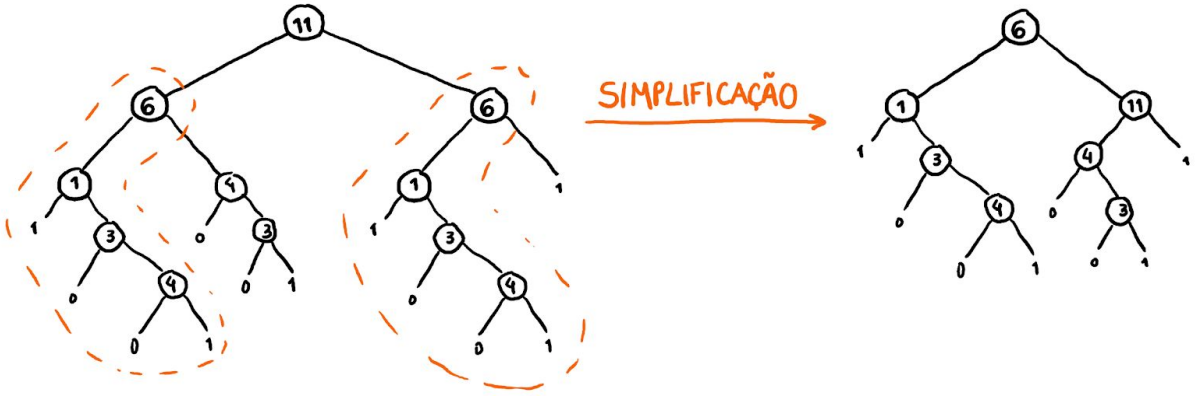

Figura 1