Inteligência Artificial

Luís A. Alexandre

UBI

Ano lectivo 2018-19

Conteúdo

Introdução

- Diz-se que um agente **aprende** quando o seu desempenho melhora após ter efetuado alguma observação do mundo.
- Iremos focar um tipo de aprendizagem que, partindo de um conjunto de pares entrada-saída, permite aprender uma função que consegue prever o valor das saídas a partir de novas entradas.
- Porque é que aprender é importante para um agente?
  - Quem programa o agente não consegue prever todas as possíveis situações que ele pode encontrar em problemas complexos.
  - É muito difícil programar o agente para lidar com alterações aos problemas ao longo do tempo.
  - Por vezes, o programador não sabe qual é a solução para o problema.

Conteúdo

Tipos de aprendizagem

- **Não supervisionada**: o agente aprende padrões das entradas, sem receber informação de feedback. Um exemplo é o chamado clustering ou agrupamento de dados.
- **Supervisionada**: o agente recebe pares de entrada-saída e aprende a relação existente entre eles podendo depois usar a relação aprendida para estimar os valores de saída de novos dados quando recebe apenas as entradas.
- **Semi-supervisionada**: o conjunto de dados que o agente recebe tem apenas um pequeno subconjunto com os valores de saída: o restante tem apenas valores de entrada.
- **Por reforço**: o agente aprende ao receber reforços após realizar ações. Estes reforços podem ser positivos ou negativos. Ex.: o facto de não receber gorjeta no fim duma viagem pode dizer ao agente que guia um taxi que o seu comportamento não terá sido bom.

Aprendizagem Supervisionada

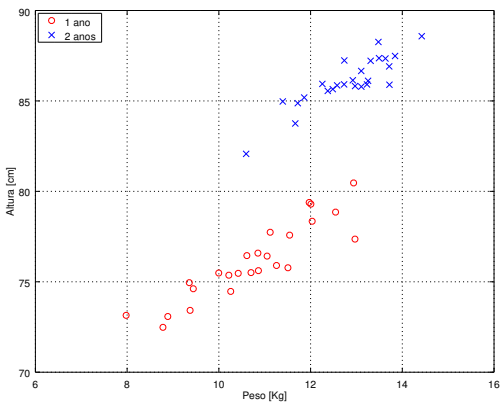
- O agente recebe um **conjunto de treino** com N pares entrada-saída:

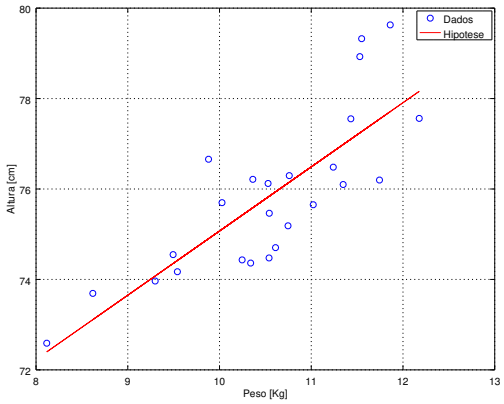
$$(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$$

- Existe naturalmente uma relação entre os x_i e os respetivos y_i, mas é normalmente desconhecida.
- A **aprendizagem** consiste em estimar essa relação $y = f(x)$.
- Dizemos que a máquina de aprendizagem ou o agente, estima a relação entre as entradas e as saídas criando uma **hipótese** h tal que h **aproxima** a verdadeira relação f.
- A aprendizagem passa então a ser uma pesquisa no **espaço de hipóteses**, $\mathcal{H}$: procuramos a hipótese $h \in \mathcal{H}$ que mais se aproxima da verdadeira relação.
- Os valores de x e y podem ser quaisquer: inteiros, reais, letras, palavras, etc.

Classificação e regressão

- Quando os valores de y pertencem a um conjunto finito, dizemos que o problema é de **classificação**.
- Quando os valores de y pertencem a um conjunto infinito, estamos perante um problema de **regressão**.





Aprendizagem Supervisionada

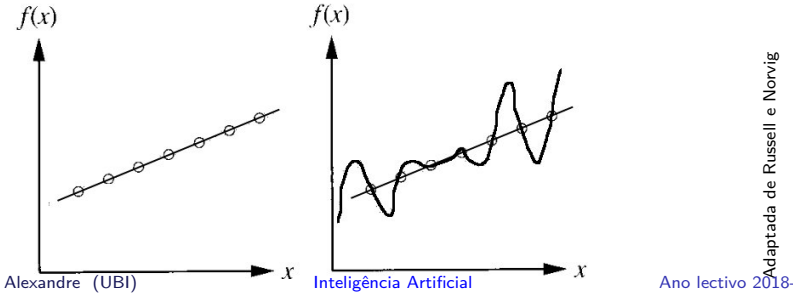
- É importante perceber que nos interessa ser bem sucedidos ao estimar as saídas, principalmente para valores que não estão no conjunto de treino: queremos **generalizar**.
- Para estimarmos quão bom será o desempenho da hipótese criada pelo nosso agente, usamos um outro conjunto de dados, chamado **conjunto de teste**, que **não** terá valores em comum com o conjunto de treino.
- Dizemos que uma hipótese **generaliza** bem se conseguir prever corretamente os valores y_i para x_i do conjunto de teste.
- Muitas vezes a relação entre as entradas e saídas não é determinística, e em vez de uma função $y_i = f(x_i)$ na realidade temos uma probabilidade $P(y_i|x_i)$ que tem que ser estimada pela nossa hipótese.

Escolha de hipóteses

- O que nós queremos é escolher uma hipótese que seja a mais provável em face dos dados que conhecemos do problema.
- Assim, para resolvermos um problema de aprendizagem supervisionada, escolhemos a hipótese

$$h^* = \arg\max_{h \in \mathcal{H}} P(h|dados) \quad (1)$$

onde $\mathcal{H}$ é o espaço de hipóteses.
- Uma hipótese diz-se **consistente** se estiver de acordo com todos os dados do conjunto de treino.



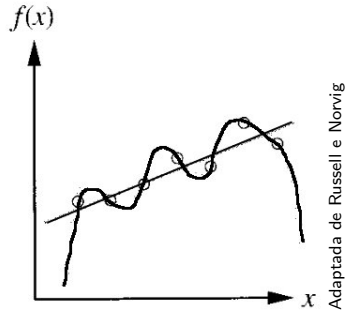
Adaptada de Russell e Norvig

Escolha de hipóteses

- Quando temos várias hipóteses consistentes, qual escolher? A mais simples! (Navalha de Ockham)
- Precisamos de definir "simplicidade", o que nem sempre é fácil, mas para o exemplo acima podemos concordar que um polinómio de grau 1 é mais simples que um de grau 9.
- Há normalmente um equilibrio que deve ser encontrado entre hipóteses que se ajustam muito bem aos dados de treino e hipóteses que generalizam bem.

Escolha do espaço de hipóteses

- Um segundo tipo de problema surge quando observamos a seguinte figura:



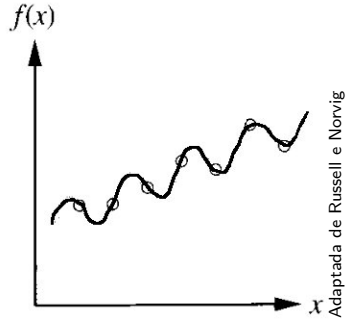
Adaptada de Russell e Norvig

Escolha do espaço de hipóteses

- A figura mostra um conjunto de pontos diferente do usado antes em que já não é possível obter uma hipótese consistente usando um polinómio de grau 1. Neste caso é necessário usar um polinómio de grau 6 (com 7 parâmetros portanto) para obter uma hipótese consistente.
- Como o conjunto de dados só tem 7 pontos, estamos a usar tantos parâmetros quantos os pontos disponíveis o que nos pode indicar que o nosso modelo não estará a conseguir abstrair uma regra que justifique as observações: em princípio não irá generalizar bem.
- Talvez seja melhor usar uma recta pois, embora já não seja uma hipótese consistente, em princípio conseguirá generalizar melhor que o polinómio de grau 6.

Escolha do espaço de hipóteses

- A figura abaixo mostra o mesmo conjunto de pontos da última figura com uma hipótese consistente obtida com recurso a uma função mais simples (só usa 3 parâmetros) que o polinómio de 6 grau e que é da forma: $ax + b + c\sin(x)$.
- Concluímos então que se mudarmos o espaço de hipóteses podemos obter de novo uma hipótese consistente e mais simples que a que necessitávamos no espaço dos polinómios.



Escolha do espaço de hipóteses

- Como escolher o espaço de hipóteses $\mathcal{H}$ a usar?
- Devemos usar conhecimento a priori para tentar descobrir um espaço que contenha uma hipótese consistente.
- Uma outra abordagem seria usar o espaço de hipóteses mais vasto possível (para termos a certeza de encontrar hipóteses consistentes).
- P. ex., $\mathcal{H}$ poderia ser o espaço de todas as máquinas de Turing. Como todas as funções computáveis podem ser representadas por uma máquina de Turing, isto seria o melhor que poderíamos alcançar.
- Agora o problema que se coloca é a complexidade computacional da aprendizagem.

Escolha do espaço de hipóteses

- Existe um equilíbrio que deve ser alcançado entre a expressividade do espaço de hipóteses e a complexidade de encontrar nesse espaço uma hipótese consistente.
- Ex.: ajustar rectas a dados é muito simples; ajustar polinómios de grau elevado é mais difícil; ajustar máquinas de Turing é muito difícil.
- Uma outra razão para se escolherem espaços de hipóteses simples é que as hipóteses resultantes serão simples e mais fáceis de calcular: é mais fácil achar $h(x)$ quando h é um polinómio de grau 1 do que quando é uma máquina de Turing.

Resumo de algumas questões da aprendizagem indutiva

- Se temos várias hipóteses consistentes, qual escolher? Normalmente a mais simples.
- Pode ser preferível usar uma hipótese simples não consistente a uma consistente mas muito complexa. Como saber qual escolher? Avaliar a sua capacidade de generalizar.
- É muito importante a escolha do espaço de hipóteses: como fazê-la? Devemos usar informação a priori sobre o problema.

Conteúdo

Árvores de decisão

- A indução com Árvores de Decisão (AD) é um dos métodos mais simples e que obtém melhores resultados em termos da aprendizagem.
- Uma AD recebe como entrada um vetor de **atributos** e devolve uma decisão: o valor previsto para a entrada que recebeu.
- As variáveis de entrada podem ser discretas ou contínuas. Por enquanto vamos assumir que têm valores discretos.
- Iremos concentrarmo-nos na classificação binária (ou booleana) onde cada exemplo é classificado como verdadeiro ou falso (ou Sim ou Não ou 0 ou 1).

Árvores de decisão

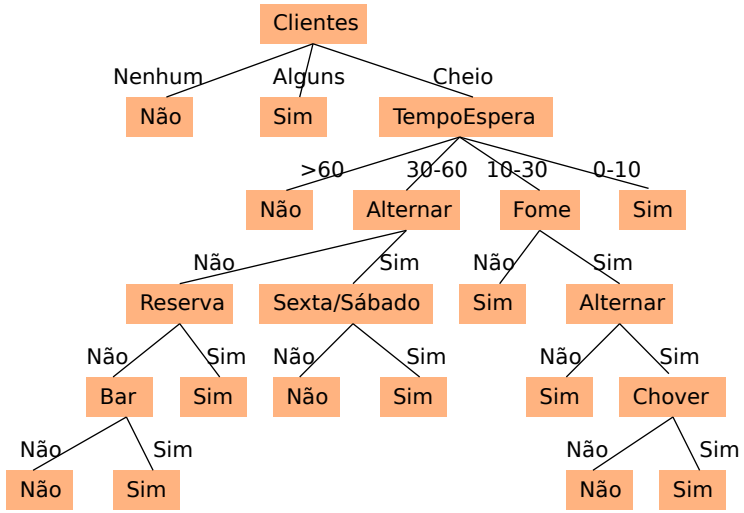
- A AD chega à **decisão** através duma **sequência de testes**.
- Cada **nodo não terminal** da árvore corresponde a um **teste** do valor de um dos atributos e as arestas que saem do nodo são etiquetadas com os valores possíveis do teste.
- Cada **folha** da árvore especifica a **decisão** a devolver se se atingir essa folha.
- Exemplo: decidir se devemos esperar por mesa num restaurante.
- Para formular isto como um problema de aprendizagem teremos que definir quais os atributos disponíveis para descrever os exemplos deste domínio.

Árvores de decisão

- Consideremos que a lista de atributos disponíveis é:
  - Alternar: se existe um outro restaurante próximo do actual onde possamos ir.
  - Bar: se o restaurante dispõe dum bar para se poder esperar pela mesa.
  - Sexta/Sábado: é verdadeiro quando for um destes dois dias.
  - Fome: verdadeiro se estivermos com fome.
  - Clientes: quantos clientes estão no restaurante (Nenhum, Alguns, Cheio).
  - Preço: a gama de preços do restaurante ($,$$,$$$).
  - Chover: se está a chover.
  - Reserva: se fizemos reserva.
  - Tipo: tipo de restaurante (Francês, Italiano, Tailandês, Hamburguer).
  - TempoEspera: o tempo estimado de espera em minutos (0-10, 10-30, 30-60, >60).

Árvores de decisão

- Uma possível árvore de decisão é a seguinte:



Árvores de decisão

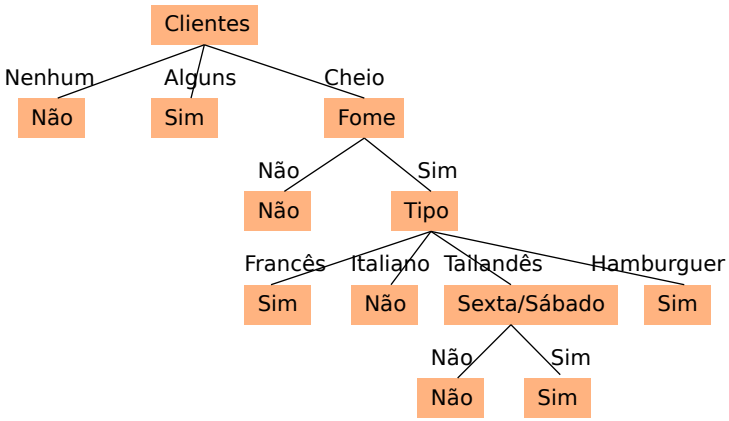
- Note-se que a árvore não usa todos os atributos disponíveis, considerando alguns irrelevantes.
- Os exemplos são processados pela árvore começando pela raiz e seguindo pela árvore até ser atingida uma folha.
- Ex.: Se o restaurante estiver cheio e o tempo de espera for entre 0 e 10 minutos então devemos esperar pela mesa.

Indução das árvores de decisão a partir de dados

- Um ponto do conjunto de dados no caso da AD booleana é um vector de atributos $\mathbf{x}$ e um valor booleano de saída y.
- O conjunto de dados usado para construir (induzir) a AD chama-se o **conjunto de treino**.
- A ideia básica do algoritmo para induzir árvores de decisão é começar por testar os atributos mais importantes primeiro.
- Os atributos mais importantes são aqueles que fazem mais diferença na classificação dum ponto: apresentam maior ganho de informação (ver mais abaixo).
- Depois de usarmos o primeiro atributo, o problema que fica é de novo a construção duma AD mas agora com menos um atributo e considerando apenas os pontos do conjunto de treino que não foram classificados com recurso ao primeiro atributo.

Indução das árvores de decisão a partir de dados

- Ao aplicar o algoritmo de indução de uma AD a um conjunto de treino foi obtida a seguinte árvore:



Indução das árvores de decisão a partir de dados

- Porque é que é diferente da árvore que mostrámos atrás? Porque foi a árvore gerada a partir dos dados recebidos. Os dados podem não representar todas as possibilidades do problema em questão.
- Quanto mais dados forem usados no conjunto de treino, em princípio mais representativa será a árvore do problema em questão.
- Porque é que não usou todos os atributos disponíveis? P. ex. não usou Chover nem Reserva. Porque consegue classificar todos os dados recebidos sem ser preciso usar esses atributos.
- Ache a classe dos seguintes dados usando as duas árvores anteriores:

Ponto	Alt	Bar	Sext	Fome	Clientes	Preço	Chove	Res	Tipo	Tempo
X_1	N	S	N	S	Alg	$$	S	S	Ita	0-10
X_2	N	N	N	N	Cheio	$	N	N	Tai	0-10
X_3	S	S	N	S	Cheio	$	N	N	Tai	30-60

Indução das árvores de decisão a partir de dados

- Os resultados são:

Ponto	Árvore1	Árvore2
X_1	S	S
X_2	S	N
X_3	N	N

- Como foram construídas com conjuntos de treino diferentes, não irão generalizar sempre da mesma forma.
- Se aumentarmos o número de dados em ambos os conjuntos de treino elas tenderão a produzir as mesmas previsões para dados nunca vistos. Voltaremos a falar deste assunto mais à frente.

Conteúdo

Avaliar o desempenho

- Um algoritmo de aprendizagem é bom se produzir hipóteses que conseguem prever a classe de pontos nunca vistos.
- A forma de verificar qual a qualidade da previsão passa por usar um **conjunto de teste**.
- O processo que se usa normalmente é chamado **validação cruzada** (cross validation):
  1. Recolher conjunto de dados, com número de pontos n
  2. Escolher o número de vezes que se irá dividir os dados: k
  3. Dividir os dados em conjuntos disjuntos: de treino com $n - n/k$ pontos e de teste com os restantes n/k pontos
  4. Aplicar o algoritmo de aprendizagem ao conjunto de treino gerando assim uma hipótese h
  5. Aplicar h ao conjunto de teste e calcular a proporção de acertos
  6. Repetir os passos 3 a 5, k vezes, nunca usando para teste pontos já usados em repetições anteriores, e devolver como estimativa da capacidade de generalização da máquina de aprendizagem a média dos k valores obtidos no ponto 5.

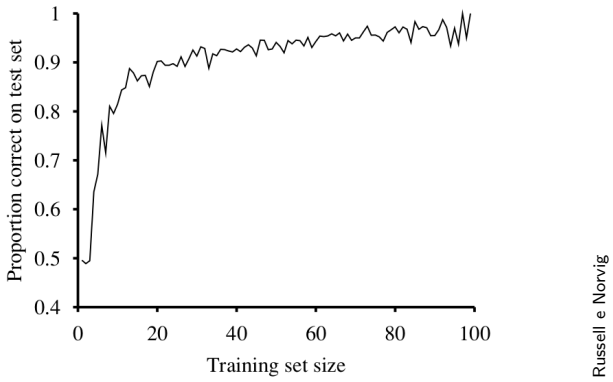
Avaliar o desempenho

$$\boxed{X_1, X_2, X_3}, X_4, X_5, X_6, X_7, X_8, X_9, X_{10}, X_{11}, X_{12}$$
$$X_1, X_2, X_3, \boxed{X_4, X_5, X_6}, X_7, X_8, X_9, X_{10}, X_{11}, X_{12}$$
$$X_1, X_2, X_3, X_4, X_5, X_6, \boxed{X_7, X_8, X_9}, X_{10}, X_{11}, X_{12}$$
$$X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8, X_9, \boxed{X_{10}, X_{11}, X_{12}}$$

- Segundo este processo são criadas k hipóteses.
- São usados todos os pontos tanto no treino como no teste, mas **nunca** simultaneamente.
- Os valores comuns para k são 5 ou 10 (o exemplo acima é para $k = 4$). Os pontos nos rectângulos são usados no conjunto de teste e os restantes no de treino.
- Se usarmos $k = n$ temos a forma de avaliação chamada leave-one-out: treinamos em todos os pontos excepto um que é usado para teste e repetimos o processo n vezes. Esta seria a forma ideal de avaliação mas normalmente é muito demorada.

Avaliar o desempenho

- Dissemos acima que se aumentarmos o tamanho do conjunto de treino, os modelos têm melhores resultados.
- A **curva de aprendizagem** ilustra esse facto:



Conteúdo

Ruído e sobre-ajuste

- Imaginemos que pretendíamos uma AD capaz de prever o resultado do lançamento de dados. E que tinham sido efectuadas experiências com vários dados e em datas diferentes.
- Tinham sido recolhidos os seguintes atributos conjuntamente com o resultado do lançamento:
  - Dia: o dia do mês em que o lançamento tinha sido efectuado;
  - Mês: o nome do mês em que o lançamento tinha sido efectuado;
  - Cor: a cor do dado usado no lançamento.
- O nosso algoritmo de construção de AD vai conseguir encontrar sempre uma hipótese consistente, ou seja, consegue sempre prever com erro zero o resultado dos lançamentos no conjunto de **treino** (desde que não existam 2 pontos com os mesmos atributos e classes diferentes).
- Mas nós sabemos que não é possível prever o resultado do lançamento de dados pois é um processo aleatório. Então o que se passa?

Ruído e sobre-ajuste

- Está a ocorrer **sobre-ajuste** (overfitting em inglês): o nosso algoritmo de aprendizagem está a tirar partido dos atributos que são todos irrelevantes para este problema, e a criar hipóteses que são consistentes mas que não irão generalizar bem.
- Quando o espaço de hipóteses é muito grande (temos a possibilidade de ajustar muitos parâmetros de h) corremos o risco de **memorização** dos dados de treino sem que se consiga obter uma boa hipótese h (que consiga generalizar).
- Todos os algoritmos de aprendizagem podem sofrer deste problema, não apenas as AD.

Ruído e sobre-ajuste

- Uma forma de evitar o sobre-ajuste nas AD consiste em usar **poda**.
- A ideia é não permitir que, quando os atributos não são relevantes, gerem sub-árvores.
- Como saber se um atributo é ou não relevante?
- Podemos usar o ganho de informação para efectuar essa avaliação.
- Para medirmos informação, usamos a **entropia**: se existirem n possíveis respostas a uma questão, cada uma com probabilidade $P(a_i)$ de ocorrer, a **quantidade de informação** que uma resposta fornece é dada por

$$E(P(a_1), P(a_2), \ldots, P(a_n)) = -\sum_{i=1}^{n} P(a_i) \log_2 P(a_i)$$

- A entropia decresce quando adquirimos mais informação, pois ela é uma medida da incerteza de uma variável aleatória.

Ruído e sobre-ajuste

- Imaginemos o lançar duma moeda ao ar: o resultado de cada lançamento irá dar-nos um bit de informação no caso em que a moeda não está viciada. Vai permitir responder à questão: irá sair cara? (com um sim ou não).
- Vejamos as contas:

$$E(P(cara), P(coroa)) =$$

$$-P(cara)\log_2(P(cara)) - P(coroa)\log_2(P(coroa)) =$$

$$-0.5\log_2(0.5) - 0.5\log_2(0.5) = 1 \text{ bit}$$

- No caso da moeda estar viciada e nós sabermos, o resultado dum lançamento irá dar muito menos informação.

Ruído e sobre-ajuste

- Estando a moeda viciada, com a probabilidade de sair cara de 0.9, vem que cada lançamento nos dá a seguinte quantidade de informação:

$$E(0.9, 0.1) = -0.9\log_2(0.9) - 0.1\log_2(0.1) = 0.47 \text{ bits}$$

- Conforme aumenta a probabilidade de sair cara, diminui a quantidade de informação que recebemos com um dado lançamento.
- Até que no limite, quando essa probabilidade for 1 (sai sempre cara) a quantidade de informação que recebemos dum lançamento é zero.

Ruído e sobre-ajuste

- O **ganho de informação** obtido com o teste do atributo A é igual à diferença entre a informação necessária inicialmente e aquela que passa a ser necessária após o teste:

$$Ganho(A) = E\left(\frac{p}{p+n}, \frac{n}{p+n}\right) - R(A)$$

onde

$$R(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p+n} E\left(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}\right)$$

e n é o número de pontos cuja classe é Não, p o número de pontos cuja classe é Sim, n_i e p_i são equivalentes mas agora para os v sub-conjuntos que se criam com as possíveis respostas ao teste do atributo A.

Ruído e sobre-ajuste

- Exemplo: para o exemplo do restaurante, se tivermos um conjunto de treino com 12 pontos em que:
  - atributo Tipo:
    - dois pontos tenham valor Francês (um ponto de cada classe)
    - dois pontos tenham valor Italiano (um ponto de cada classe)
    - quatro pontos tenham valor Tailandês (dois pontos de cada classe)
    - quatro pontos tenham valor Hamburguer (dois pontos de cada classe)
  - atributo Clientes:
    - dois pontos tenham valor Nenhum (ambos da classe Não)
    - quatro pontos tenham valor Algum (todos da classe Sim)
    - seis pontos tenham valor Cheio (dois da classe Sim e o resto da Não)
- Vamos então decidir qual destes atributos será menos relevante e poderá ser podado.

Ruído e sobre-ajuste

- Quanta informação é necessária para tomarmos a decisão, ou seja, para respondermos à questão se devemos ficar ou procurar outro restaurante? 1bit.
- Quanta informação é necessária após o teste relativo ao atributo Tipo?

$$R(Tipo) = \frac{2}{12}E(1/2,1/2) + \frac{2}{12}E(1/2,1/2) + \frac{4}{12}E(2/4,2/4) + \frac{4}{12}E(2/4,2/4) = 1$$

- O ganho do teste é então 1-1=0 bits, ou seja, não se ganha nada com este teste!
- Quanta informação é informação necessária após o teste relativo ao atributo Cliente?

$$R(Cliente) = \frac{2}{12}E(0,1) + \frac{4}{12}E(1,0) + \frac{6}{12}E(2/6,4/6) = 0.459$$

- O ganho que se obtém usando este atributo é 1-0.459=0.541 bits.
- Concluímos que entre estes dois atributos devemos podar o Tipo.
- Na realidade, o atributo Clientes é o mais informativo de todos e deve ser escolhido como o primeiro teste na AD.

Ruído e sobre-ajuste

- A noção de ganho de informação é usada também na escolha dos atributos mais informativos durante a construção da árvore: devem ser escolhidos primeiro os testes que envolvem os atributos mais informativos.
- O uso da poda permite que as árvores de decisão consigam também lidar adequadamente com eventual ruído que exista nos dados: quanto mais simples for o modelo, menos se pode ajustar ao ruído.
- As árvores podadas são mais curtas em média que as não podadas e por isso permitem uma melhor compreensão do processo de decisão (existem menos nodos com testes).

Leitura recomendada

- Russell e Norvig, sec. 18.1 a 18.3.