Aprendizado de Máquina

Aula 3

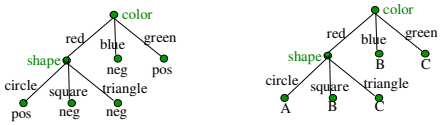
http://www.ic.uff.br/~bianca/aa/

Tópicos

1. Introdução – Cap. 1 (16/03)
2. Classificação Indutiva – Cap. 2 (23/03)
3. **Árvores de Decisão – Cap. 3 (30/03)**
4. Ensembles - Artigo (06/04)
5. Avaliação Experimental – Cap. 5 (13/04)
6. Teoria do Aprendizado – Cap. 7 (20/04)
7. Aprendizado de Regras – Cap. 10 (27/04)
8. Redes Neurais – Cap. 4 (04/05)
9. Máquinas de Vetor de Suporte – Artigo (11/05)
10. Aprendizado Bayesiano – Cap. 6 e novo cap. online (18/05)
11. Aprendizado Baseado em Instâncias – Cap. 8 (25/05)
12. Classificação de Textos – Artigo (01/06)
13. Aprendizado Não-Supervisionado – Artigo (08/06)

Árvores de Decisão

- São classificadores baseados em árvores para instâncias representadas como vetores de características.
- Os nós internos testam os valores de uma característica.
  - Existe um ramo para cada valor possível da característica.
- As folhas especificam a classe.



Árvores de Decisão

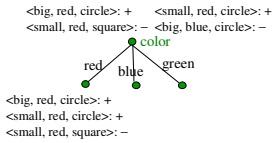
- Podem representar conjunções e disjunções arbitárias.
- Podem representar qualquer função de classificação com atributos discretos.
- Podem ser reescritas como um conjunto de regras, por exemplo, em forma normal disjuntiva (DNF).
  - red $\wedge$ circle $\rightarrow$ pos
    green $\rightarrow$ pos
  - red $\wedge$ circle $\rightarrow$ A
    blue $\rightarrow$ B; red $\wedge$ square $\rightarrow$ B
    green $\rightarrow$ C; red $\wedge$ triangle $\rightarrow$ C

Propriedades de Árvores de Decisão

- Atributos **contínuos** podem ser usados fazendo o nó dividir o domínio do atributo entre dois intervalos baseados em um limite (ex. tamanho < 3 e tamanho ≥3)
- Árvores de classificação têm valores discretos nas folhas, **árvores de regressão** têm valores reais nas folhas.
- Algoritmos para encontrar árvores consistentes são **eficientes** e podem processar grandes quantidades de dados de treinamento.
- Os métodos de aprendizado de árvore de decisão lidam bem com **ruído**.
- Existem métodos de árvore de decisão que permitem **valores desconhecidos** para algumas características.

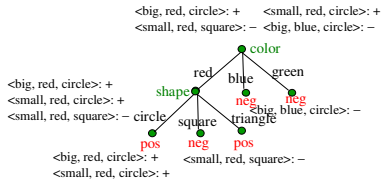
Indução Top-Down de Árvores de Decisão

- Árvore é construída recursivamente de cima para baixo, usando divisão e conquista.



Indução Top-Down de Árvores de Decisão

- Árvore é construída recursivamente de cima para baixo, usando divisão e conquista.



Pseudocódigo

DTree(*exemplos*, *atributos*) retorna uma árvore
Se todos *exemplos* pertencem a uma única classe,
retorna um nó folha com essa classe.
Senão Se o conjunto de *atributos* estiver vazio,
retorna um nó folha com a classe mais comum entre os exemplos.
Senão escolha um atributo F e crie um nó R para ele
Para cada possível valor v_i de F:
Seja $exemplos_i$ o subconjunto de exemplos que tenha valor v_i para F
Coloque uma aresta E a partir do nó R com o valor v_i.
Se $exemplos_i$ estiver vazio
coloque um nó folha ligado a aresta E com a classe mais comum entre os exemplos.
Senão chame DTree($exemplos_i$, $atributos - \{F\}$) e ligue a árvore resultante como uma sub-árvore sob E.
Retorne a sub-árvore com raiz R.

Escolhendo um atributo

- O objetivo é ter a menor árvore possível, de acordo com a lâmina de Occam.
- Encontrar a árvore de decisão mínima (nós, folhas, ou profundidade) é um problema NP-difícil.
- O método top-down de divisão e conquista faz uma busca gulosa pela árvore mais simples mas não garante que a menor seja encontrada.
- Queremos escolher um atributo que gere subconjuntos de exemplos que sejam relativamente "puros" para que eles estejam mais perto da folha.
- Há uma variedade de heurísticas pra escolher o atributo, uma muito conhecida é o ganho de informação que é usada no sistema ID3 de Quinlan (1979).

Entropia

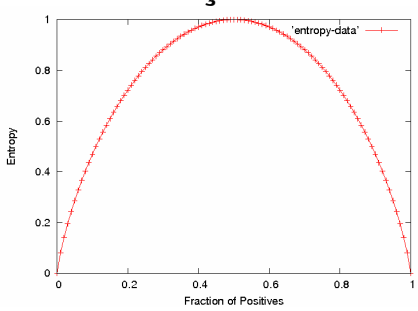
- Entropia (desordem, impureza) de um conjunto de exemplos, S, relativa a classificação binária é:

$$Entropy\ (S) = -p_1 \log_2(p_1) - p_0 \log_2(p_0)$$

onde p_1 é a fração de exemplos positivos em S e p_0 é a fração de negativos.
- Se todos os exemplos forem da mesma classe, a entropia é zero (definimos $0 \cdot \log(0)=0$)
- Se os exemplos estiverem igualmente misturados ($p_1=p_0=0.5$), entropia é 1.
- Entropia pode ser vista como o número médio de bits necessários para codificar a classe de um exemplo em S onde a compressão dos dados é usada para dar códigos mais curtos para classes mais frequentes.
- Para problemas multi-classe com c classes, a entropia generaliza para:

$$Entropy(S) = \sum_{i=1}^{c} -p_i \log_2(p_i)$$

Entropia para Classificação Binária

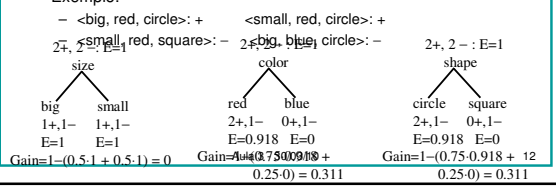


Ganho de Informação

- O ganho de informação de um atributo F é a redução esperada em entropia que resulta de escolher o atributo para um nó

$$Gain(S,F) = Entropy(S) - \sum_{v \in Values(F)} \frac{|S_v|}{|S|} Entropy(S_v)$$

onde S_v é o subconjunto de S que tem valor v para o atributo F.
- A entropia de cada subjconjunto resultante tem um peso proporcional ao tamanho do subconjunto.
- Exemplo:



Busca no espaço de hipóteses

- Faz um aprendizado ***batch*** que processa todos os exemplos de treinamento simultaneamente (contrário ao aprendizado ***incremental*** que atualiza a hipótese depois de cada exemplo).
- Faz subida de encosta (busca gulosa) que pode encontrar apenas um máximo local. Tem garantia de encontrar uma árvore consistente pra qualquer conjunto de dados que não tenha conflitos (i.e. instâncias com características idênticas sempre tem a mesma classe), mas não necessariamente a árvore mais simples.

Viés na indução de árvores de decisão

- Ganho de informação dá um viés a favor de árvores com profundidade mínima.
- Corresponde a um viés de busca (preferência) e não a um viés de linguagem (restrição).

Histórico da pesquisa em árvores de decisão

- Hunt e colegas usam métodos de árvore de decisão com busca (CLS) para modelar o aprendizado humano de conceitos na década de 1960.
- No final dos anos 70, Quinlan desenvolve o ID3 com a heurística de ganho de informação para aprender sistemas especialistas a partir de exemplos.
- Simultaneamente, Breiman, Friedman e seus colegas desenvolvem CART (Classification and Regression Trees), semelhante ao ID3.
- Na década de 1980 uma série de melhorias foram introduzidas para lidar com o ruído, características contínuas, características ausentes, e critérios de divisão melhores.
- Pacote de árvore de decisão atualizado de Quinlan (C4.5) lançado em 1993.
- Weka inclui uma versão Java do C4.5 chamada J48.

Weka J48 Trace 1

```
data> java weka.classifiers.trees.J48 -t figure.arff -T figure.arff -U -M 1
Options: -U -M 1
J48 unpruned tree
------------------
color = blue: negative (1.0)
color = red
|   shape = circle: positive (2.0)
|   shape = square: negative (1.0)
|   shape = triangle: positive (0.0)
color = green: positive (0.0)

Number of Leaves  :     5
Size of the tree :      7

Time taken to build model: 0.03 seconds
Time taken to test model on training data: 0 seconds
```


Weka J48 Trace 2

```
data> java weka.classifiers.trees.J48 -t figure3.arff -T figure3.arff -U -M 1
Options: -U -M 1
J48 unpruned tree
------------------
shape = circle
|   color = blue: negative (1.0)
|   color = red: positive (2.0)
|   color = green: positive (1.0)
shape = square: positive (0.0)
shape = triangle: negative (1.0)

Number of Leaves  :     5
Size of the tree :      7

Time taken to build model: 0.02 seconds
Time taken to test model on training data: 0 seconds
```


Weka J48 Trace 3

```
data> java weka.classifiers.trees.J48 -t contact-lenses.arff
J48 pruned tree
------------------
tear-prod-rate = reduced: none (12.0)
tear-prod-rate = normal
|   astigmatism = no: soft (6.0/1.0)
|   astigmatism = yes
|   |   spectacle-prescrip = myope: hard (3.0)
|   |   spectacle-prescrip = hypermetrope: none (3.0/1.0)

Number of Leaves  :     4
Size of the tree :      7

Time taken to build model: 0.03 seconds
Time taken to test model on training data: 0 seconds

=== Error on training data ===

Correctly Classified Instances          22               91.6667 %
Incorrectly Classified Instances         2                8.3333 %
Kappa statistic                          0.8447
Mean absolute error                      0.0833
Root mean squared error                  0.2041
Relative absolute error                 22.6257 %
Root relative squared error             48.1223 %
Total Number of Instances               24

=== Confusion Matrix ===

 a  b  c   <-- classified as
 5  0  0 |  a = soft
 0  3  1 |  b = hard
 1  0 14 |  c = none
```



```
=== Stratified cross-validation ===

Correctly Classified Instances          20               83.3333 %
Incorrectly Classified Instances         4               16.6667 %
Kappa statistic                          0.71
Mean absolute error                      0.15
Root mean squared error                  0.3249
Relative absolute error                 39.7059 %
Root relative squared error             74.3898 %
Total Number of Instances               24

=== Confusion Matrix ===

 a  b  c   <-- classified as
 5  0  0 |  a = soft
 0  3  1 |  b = hard
 1  2 12 |  c = none
```

Complexidade Computacional

- No pior caso cria uma árvore completa onde todos os caminhos testam todos os atributos. Suponha *n* exemplos e *m* atributos.

Máximo de *n* exemplos espalhados por todos os nós em cada um dos *m* níveis

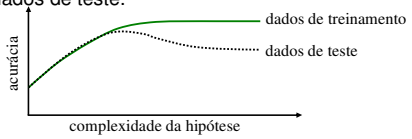
- Em cada nível, *i*, na árvore, temos que examinar os *m*– *i* atributos restantes para cada instância do nível para calcular ganhos de informação.

$$\sum_{i=1}^{m} i \cdot n = O(nm^2)$$

- Porém, a árvore aprendida raramente é completa (número de folhas é ≤ *n*). Na prática, a complexidade é linear em relação ao número de atributos (*m*) e o número de exemplos de treinamento (*n*).

Superajuste (*Overfitting*)

- Aprender uma árvore que classifica os dados de treinamento corretamente pode não levar a uma árvore com boa generalização para os dados de teste.
  - Pode haver ruído nos dados
  - Escolhas não-confiáveis baseadas em poucos casos.
- Uma hipótese, *h*, está superajustada aos dados de treinamento se existe outra hipótese *h´*, tal que *h* erra menos que *h´* para os dados de treinamento e erra mais para os dados de teste.

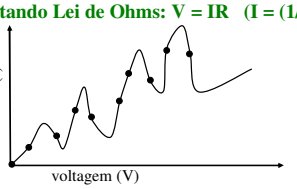


Exemplo de Superajuste (*Overfitting*)

Testando Lei de Ohms: V = IR (I = (1/R)V)

Medir experimentalmente 10 pontos

Ajustar uma curva aos dados resultantes

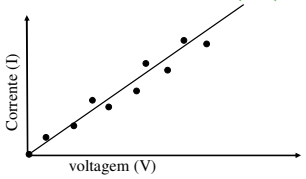


Ajuste perfeito aos dados de treinamento
com um polinômio de grau 9
(podemos passar por *n* pontos com um polinômio de grau *n*-1)

Ohm estava errado, encontramos uma função mais correta!

Exemplo de Overfitting

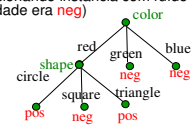
Testando Lei de Ohms: V = IR (I = (1/R)V)



Generalização melhor com função linear
que segue os dados de treinamento com menos precisão.

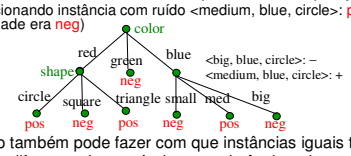
Ajustando ruído em árvores de decisão

- Ruído na classe ou características pode causar superajuste.
  - Adicionando instância com ruído <medium, blue, circle>: pos (mas na verdade era neg)



Ajustando ruído em árvores de decisão

- Ruído na classe ou características pode causar superajuste.
  - Adicionando instância com ruído <medium, blue, circle>: pos (mas na verdade era neg)



- O ruído também pode fazer com que instâncias iguais tenham classes diferente. Impossível ter consistência e devemos escolher a classe da maioria na folha.
  - <big, red, circle>: neg (mas na verdade pos)
- Exemplos conflitantes podem também aparecer se as características forem incompletas para determinar a classe ou se o conceito-alvo for não-determinístico.

Métodos de Prevenção de Superajuste (Poda)

- Duas abordagens básicas para árvores de decisão:
  - Pré-poda: para de crescer a árvore quando não tem mais dados suficientes para fazer previsões confiáveis
  - Pós-poda: constroi a árvore toda, depois removem sub-árvores que não evidência suficiente.
- A folha resultante da poda deve ser marcada com a classe majoritária ou uma distribuição de probabilidade de cada classe.

Métodos de Prevenção de Superajuste (Poda)

- Métodos para determinar quais sub-árvores podar:
  - Validação cruzada: Reservar alguns dados de treinamento (***conjunto de validação***) para avaliar utilidade das sub-árvores.
  - Teste estatístico: Usa o teste para determinar se a regularidade observada foi devida ao acaso.
  - Comprimento da descrição mínima (MDL): Determina se a complexidade adicional da hipótese é mais ou menos complexa que lembrar explicitamente as exceções resultantes da poda.

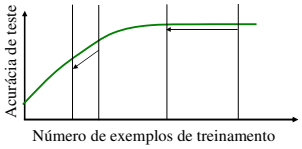
Reduced Error Pruning

- A post-pruning, cross-validation approach.

Partition training data in "grow" and "validation" sets.
Build a complete tree from the "grow" data.
Until accuracy on validation set decreases do:
  For each non-leaf node, n, in the tree do:
    Temporarily prune the subtree below n and replace it with a leaf labeled with the current majority class at that node.
    Measure and record the accuracy of the pruned tree on the validation set.
  Permanently prune the node that results in the greatest increase in accuracy on the validation set.

Problemas com a poda baseada em redução de erro

- O problema com essa abordagem é que ela potencialmente "perde" dados de treinamento no conjunto de validação.
- Severidade desse problema depende de onde estamos na curva de aprendizagem:



Validação Cruzada sem perder dados de treinamento

- Se o algoritmo for modificado para construir árvores em largura ao invés de profundidade, podemos parar o crescimento antes de atingir uma certa complexidade.
- First, run several trials of reduced error-pruning using different random splits of grow and validation sets.
- Record the complexity of the pruned tree learned in each trial. Let *C* be the average pruned-tree complexity.
- Grow a final tree breadth-first from all the training data but stop when the complexity reaches *C*.
- Similar cross-validation approach can be used to set arbitrary algorithm parameters in general.

Detalhes adicionais de árvores de decisão

- Melhores critérios de divisão
  - Ganho de informação prefere atributos com muitos valores.
- Atributos contínuos
- Previsão de função real (árvores de regressão)
- Ausência de valores de características
- Atributos com custos
- Custos de erros de classificação
- Aprendizado incremental
  - ID4
  - ID5
- Mineração de bases grandes que não cabem na memória