Aprendizagem Automática

Árvores de atributos

Teresa Gonçalves, Universidade de Évora
tcg@uevora.pt

Sumário

- Árvores
- Algoritmo
- Função de impureza
- Geração de regras
- Poda da árvore
- Árvores de regressão

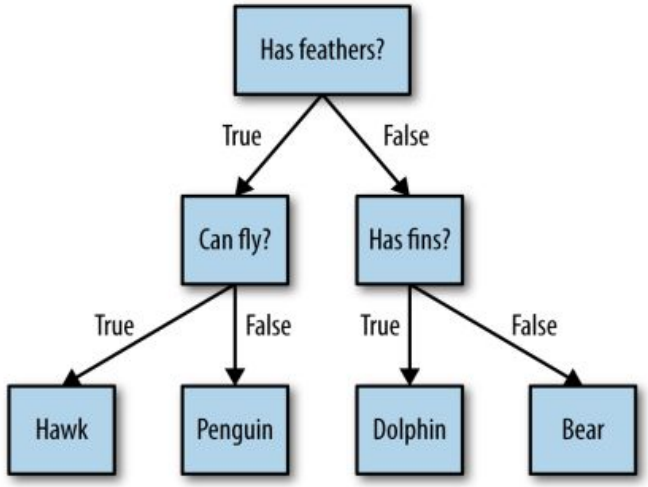
Árvores

Árvores

- Um dos modelos mais populares de AA
- Porquê?
  - Árvores são expressivas e fáceis de perceber
  - Atraente para informáticos devido à sua natureza de "divisão-e-conquista"
  - Transformação fácil numa expressão lógica
    - Basta fazer a disjunção de todos os caminhos da raíz até às folhas

Árvore de atributos

- Cada nó interno é etiquetado com um **atributo**
- Cada arco é etiquetado com um **literal** (valor do atributo)
  - ao conjunto de literais chama-se uma divisão (*split*)



Aprendizagem e previsão

- Construção do modelo
  - gerar uma árvore com base no conj de treino
- Previsão de um exemplo
  - percorrer a árvore usando os atributos até chegar a uma folha

Algoritmo

Construção do modelo

- Assume a definição de 3 funções (D: conj de exemplos, A: conj de atributos)
  - Homogeneo(D)
    - True se as instâncias em D são homogéneas o suficiente para atribuir uma única etiqueta
  - Etiqueta(D)
    - devolve a etiqueta mais apropriada para um conj de exemplos D
  - MelhorParticao(D, A)
    - devolve o melhor atributo para colocar na raíz da árvore
- As funções dependem da tarefa
  - classificação vs. regressão

Algoritmo

Input: conj dados D; conj atributos A
Output: árvore de atributos T com folhas etiquetadas

CresceArvore(D,F)

```
if Homogeneo(D)
        return Etiqueta(D);
MA ← MelhorParticao(D, A);
divide D em subconjuntos D_i de acordo com os literais i em MA;
for each i do
        if D_i não vazio
                F_i ← CresceArvore(D_i , A)
        else
                F_i é uma folha com etiqueta Etiqueta(D);
        end
end
return uma árvore de raíz com atributo MA e filhos F_i
```

Características do algoritmo

- Estratégia de divisão-e-conquista
  - divide os dados em sub-conjuntos
  - cria uma árvore para cada sub-conjunto e
  - combina essas sub-árvores numa única árvore
- Algoritmo guloso (*greedy*)
  - na escolha da melhor divisão, a **melhor alternativa** é seleccionada com base na informação disponível (e a escolha nunca é reconsiderada)
  - pode levar a **escolhas sub-ótimas**

Classificação

Árvore de decisão

- É uma árvore de atributos para problemas de classificação

- Funções
  - Homogeneo(D)
    - D é homogéneo se os exemplos forem de uma única classe
  - Etiqueta(D)
    - Etiqueta com a classe maioritária
  - MelhorParticao(D, A)
    - Pesquisa a partição com maior **pureza**
      - Um nó é "puro" se todos os seus exemplos pertencerem à mesma classe

MelhorParticao: algoritmo

Input: conj dados D; conj atributos A
Output: atributo a para partição

I_{min} = 1; // valor maximo da impureza

for each a em A **do**
    Divide D em subconjuntos D_1, ... D_n, de acordo com os valores v_i de a
    imp = Impureza({D1, ... Dn})
    if imp < I_{min}
        I_{min} = imp;
        a_{best} = a;
    end
end
return a_{best}

Partição: atributos nominais

- Um ramo para cada valor do atributo
- Em cada caminho, da raíz às folhas, é feito no máximo um teste a cada atributo

Partição: atributos contínuos

- É feita uma discretização binária usando um **ponto de corte**
  - partição do intervalo em dois sub-intervalos
  - condição a <=**limiar**
    - se True: ramo esquerda
    - se False: ramo direita
- São avaliados todos os pontos de corte possíveis
  - os exemplos são ordenados pelo valor do atributo; os pontos de corte estão na fronteira entre classes
- em cada caminho, da raíz às folhas, podem existir vários testes ao mesmo atributo (com outros pontos de corte)

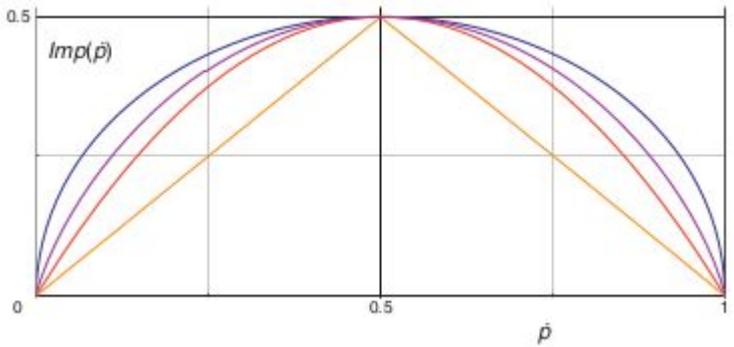
Função de impureza

Características da função de impureza

- Caso mais simples
  - classificação binária e atributos booleanos
- Função real
  - mínimo: 0 (partição pura)
  - máximo: 1
- Características
  - depende apenas da magnitude relativa (proporção **p**) do nº de exemplos de cada classe
  - deve ter o mesmo valor trocando a classe positiva e negativa
  - deve ser 0 sempre que a proporção é 0 ou 1
  - deve ser máxima quando a proporção é 1/2

Funções de impureza: 2 classes

- Razão do erro
  - min(p, 1-p)
  - Também conhecida como classe minoritária
- Índice de Gini
  - 2 p (1-p)
  - É o erro esperado se etiquetar as folhas aleatoriamente
- Entropia
  - $- p \log_2 p - (1-p) \log_2 (1-p)$
  - É a informação esperada em bits



amarelo: razão do erro
vermelho: índice de Gini
púrpura: entropia

Funções para múltiplas classes

- Índice de Gini

$$gini(D) = 1 - \sum_{c=1}^{n} p_c^2$$

- Entropia

$$entropy(D) = -\sum_{c=1}^{n} p_c \, log_2 \, p_c$$

Impureza de uma partição

- Impureza do nó D_j
  - $Imp(D_j)$
- Impureza de uma partição
  - Impureza de um conjunto de nós $\{D_1, ..., D_n\}$ mutuamente exclusivos

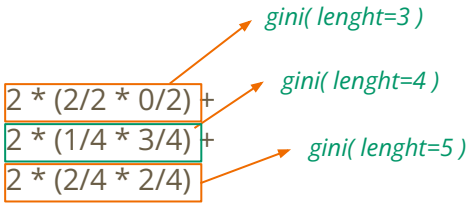
$$Imp(D_1, \ldots, D_n) = \sum_{j=1}^{n} \frac{|D_j|}{|D|} Imp(D_j)$$

Cálculo da impureza: índice de Gini

- 4 atributos + classe
  - Length={3,4,5}
  - Gills={yes, no}
  - Beak={yes, no}
  - Teeth={many, few}
  - class={+,-}
- 10 exemplos
  - (3, no, yes, many, +)
  - (4, no, yes, many, +)
  - (3, no, yes, few, +)
  - (5, no, yes, many, +)
  - (5, no, yes, few, +)
  - (5, yes, yes, many, -)
  - (4, yes, yes, many, -)
  - (5, yes, no, many, -)
  - (4, yes, no, many, -)
  - (4, no, yes, few, -)

- Possíveis divisões na raíz
  - Length: [2+, 0−] [1+, 3−] [2+, 2−]
  - Gills: [0+, 4−] [5+, 1−]
  - Beak: [5+, 3−] [0+, 2−]
  - Teeth : [3+, 4−] [2+, 1−]

- Índice de Gini
  - Length = 0.35
    - 2/10 * 2 * (2/2 * 0/2) +
      4/10 * 2 * (1/4 * 3/4) +
      4/10 * 2 * (2/4 * 2/4)



  - Gills=0.17
  - Beak=0.38
  - Teeth=0.48

- Melhor atributo
  - Gills (menor impureza)

Cálculo da impureza: entropia

- 4 atributos + classe
  - Length={3,4,5}
  - Gills={yes, no}
  - Beak={yes, no}
  - Teeth={many, few}
  - class={+,-}
- 10 exemplos
  - (3, no, yes, many, +)
  - (4, no, yes, many, +)
  - (3, no, yes, few, +)
  - (5, no, yes, many, +)
  - (5, no, yes, few, +)
  - (5, yes, yes, many, -)
  - (4, yes, yes, many, -)
  - (5, yes, no, many, -)
  - (4, yes, no, many, -)
  - (4, no, yes, few, -)

- Possíveis divisões na raíz
  - Length: [2+, 0−] [1+, 3−] [2+, 2−]
  - Gills: [0+, 4−] [5+, 1−]
  - Beak: [5+, 3−] [0+, 2−]
  - Teeth : [3+, 4−] [2+, 1−]

- Entropia
  - Length = 0.72
    - 2/10 * (- 2/2 $\log_2$ 2/2 - 0/2 $\log_2$ 0/0) + ***0***
      4/10 * (- 1/4 $\log_2$ 1/4 - 3/4 $\log_2$ 3/4) + ***0.81***
      4/10 * (- 2/4 $\log_2$ 2/4 - 2/4 $\log_2$ 2/4) ***1***
  - Gills=0.39
  - Beak=0.76
  - Teeth=0.97

- Melhor atributo
  - Gills (menor impureza)

Exemplos

Exemplo 1: atributos nominais

- 4 atributos + classe
  - Length={3,4,5}
  - Gills={yes, no}
  - Beak={yes, no}
  - Teeth={many, few}
  - class={+,-}
- 10 exemplos
  - (3, no, yes, many, +)
  - (4, no, yes, many, +)
  - (3, no, yes, few, +)
  - (5, no, yes, many, +)
  - (5, no, yes, few, +)
  - (5, yes, yes, many, -)
  - (4, yes, yes, many, -)
  - (5, yes, no, many, -)
  - (4, yes, no, many, -)
  - (4, no, yes, few, -)

- Raíz = Gills
  - *Gills*
    - Yes: [0+, 4−]: folha pura; etiqueta −
    - No: [5+, 1−]

- Próxima divisão
  - Length: [2+, 0−][1+, 1−][2+, 0−]
  - Teeth: [3+, 0−][2+, 1−]
  - Beak: não diminui impureza (todos os exemplos são yes)
  - Divisão *Length* é mais pura que *Teeth*
  - a escolha é *Length*

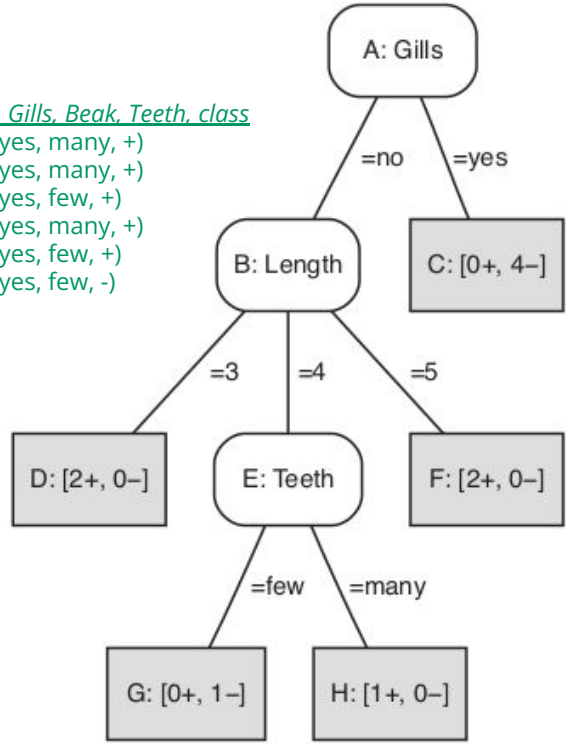
nº de exemplos no nó filho mantém-se

- Divisão seguinte
  - *Teeth*

Exemplo 1: árvore de decisão

- Raíz = Gills
  - Gills
    - Yes: [0+, 4−]: pure leaf labelled negative
    - No: [5+, 1−]
- Próxima divisão (Gills=no, 6 exemplos)
  - Length: [2+, 0−][1+, 1−][2+, 0−]
  - Beak: não diminui impureza (todos os exemplos são yes)
  - Teeth: [3+, 0−][2+, 1−]
  - Divisão *Length* é mais pura que *Teeth*
- Próxima divisão (Length=4, 2 exemplos)
  - *Teeth*

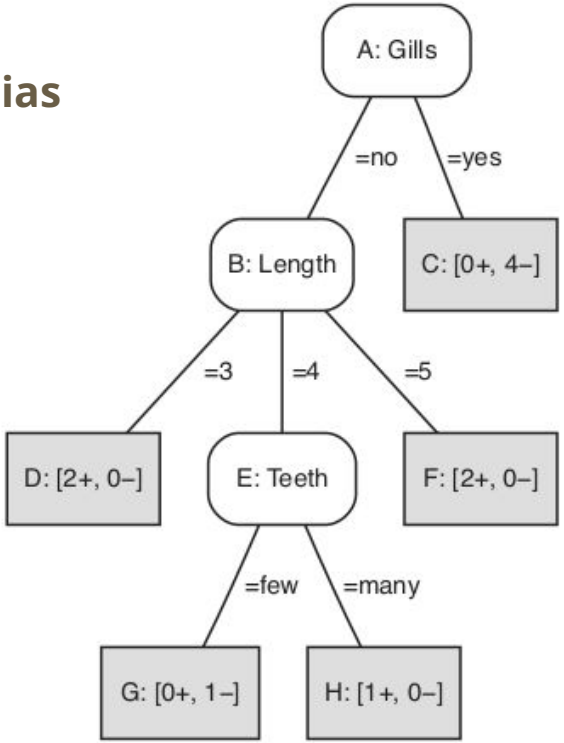
Length, Gills, Beak, Teeth, class
(3, no, yes, many, +)
(4, no, yes, many, +)
(3, no, yes, few, +)
(5, no, yes, many, +)
(5, no, yes, few, +)
(4, no, yes, few, -)



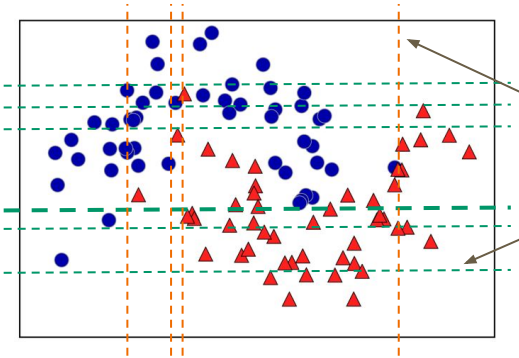
Exemplo 1: generalização

- A árvore representa uma **partição do espaço de instâncias**
- A árvore generaliza o conj de treino
  - atribui uma classe aos exemplos que não pertencem ao conj treino
- Generalização
  - Folha C
    - 3 atributos não especificados
    - totalizam 3*2*2 = 12 possíveis combinações de valores
    - conj treino: 4 exemplos; 8 exemplos por classificar
  - Folha D, Folha F
    - 2 atributos não especificados
    - totalizam 2*2=4 combinações de valores possíveis
    - conj treino: 2 exemplos; 2 exemplos por classificar

lenght: 3 valores
beak: 2 valores
teeth: 2 valores

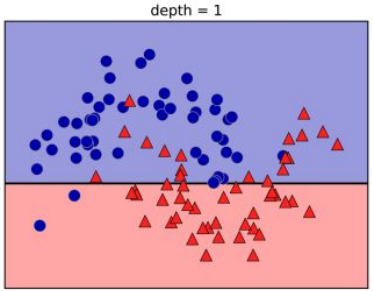


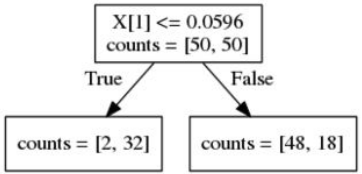
Exemplo 2: atributos contínuos

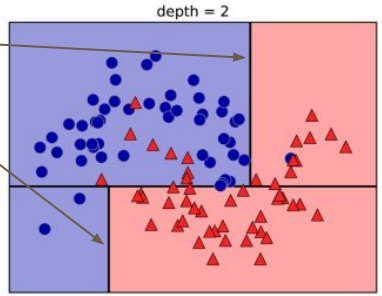


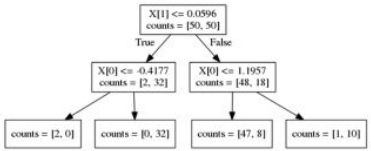
possíveis pontos de corte (não exaustivo)

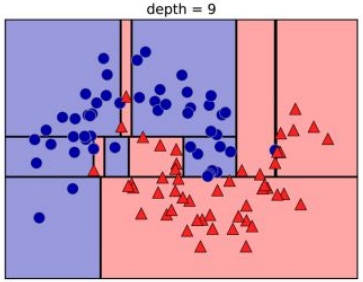
pontos de corte

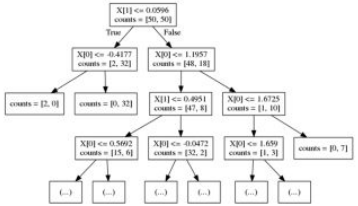






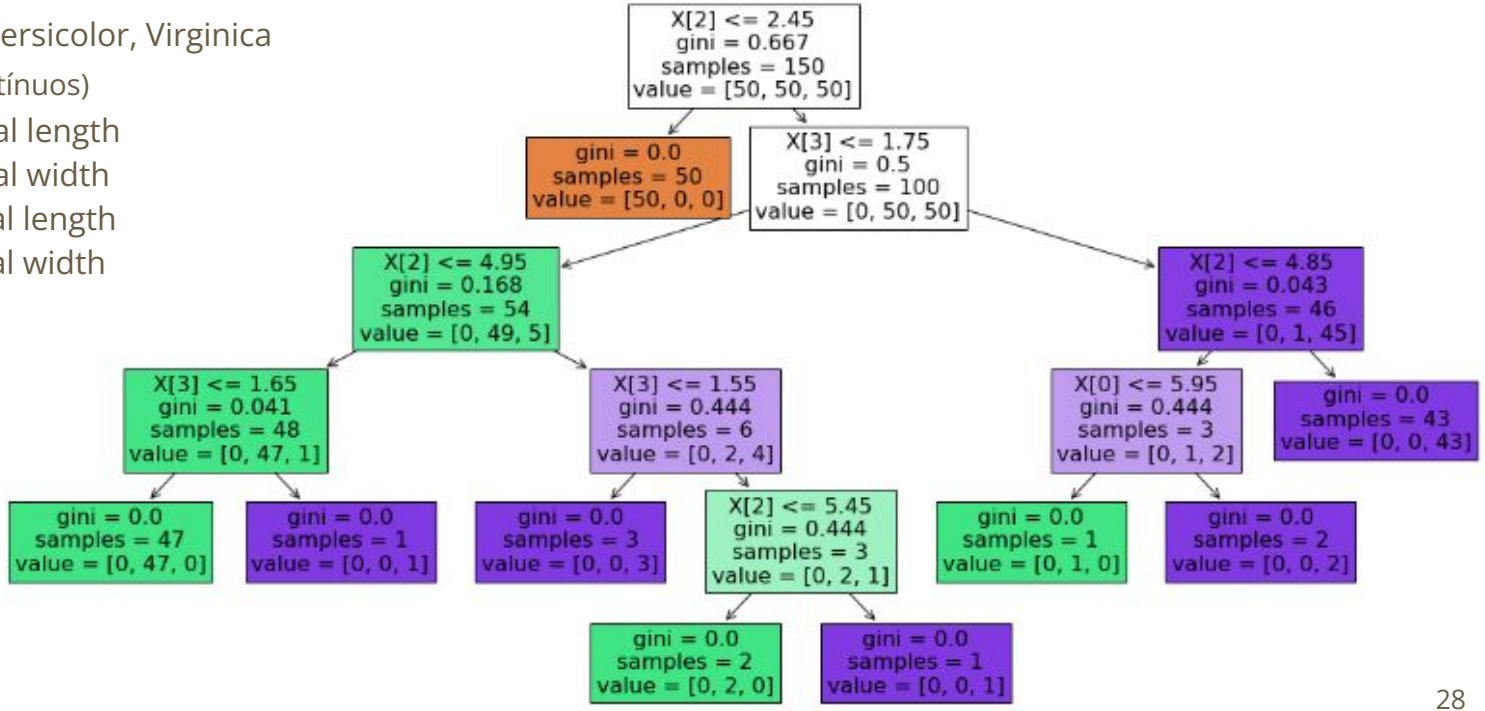






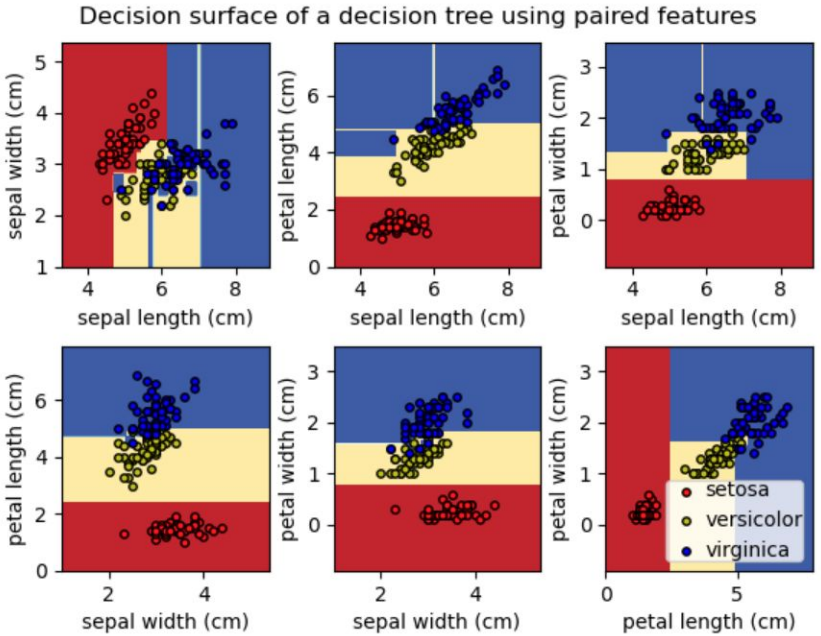
Iris: árvore de decisão

- classe
  - Setosa, Versicolor, Virginica
- atributos (contínuos)
  - X[0]: sepal length
  - X[1]: sepal width
  - X[2]: petal length
  - X[3]: petal width



Iris: fronteira de decisão

- Partições sucessivas perpendiculares a um dos eixos



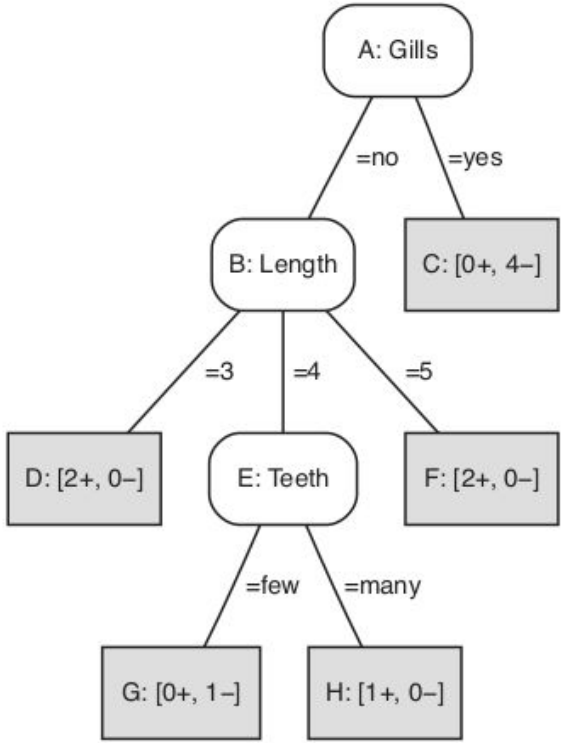
Geração de regras

Geração de regras

- Cada caminho da raíz a uma folha corresponde à conjunção de condições indicadas
- A classificação dos exemplos numa determinada classe corresponde a todos os caminhos até folhas etiquetadas com essa classe
  - Disjunção de conjunções

Exemplo 1: regras

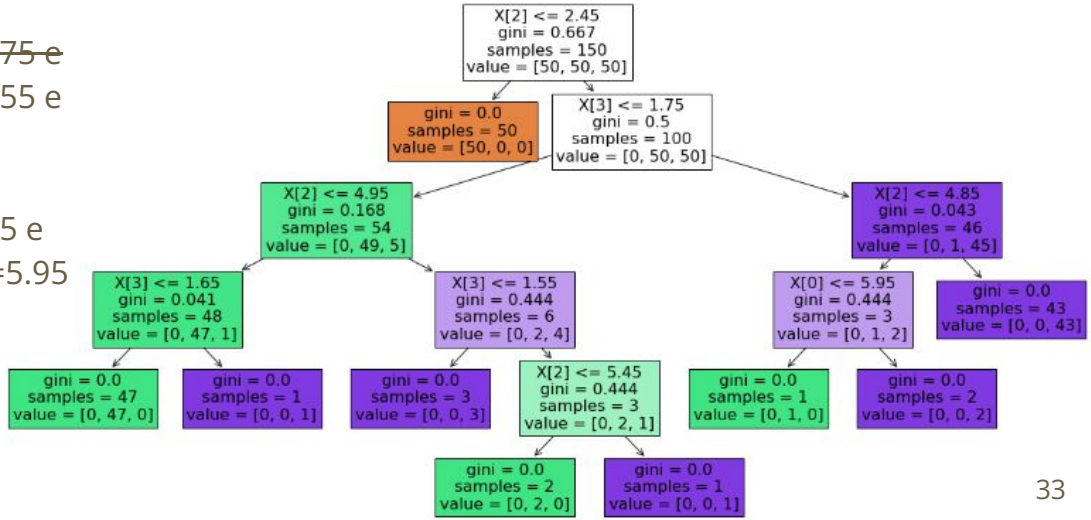
- Classe +
  - Gills==no e Length==3 ou
    Gills==no e Length=5 ou
    Gill==no e Length==4 e Teeth==many
- Classe -
  - Gills== yes ou
    Gills==no e Lenght==4 e Teeth==few



Iris

- Setosa
  - petal_lenght<=2.45
- Versicolor
  - petal_lenght>2.45 e ~~petal_width<=1.75 e~~ petal_lenght<=4.95 e petal_width<=1.65
    ou
    ~~petal_lenght>2.45 e petal_width<=1.75 e~~ petal_lenght>4.95 e petal_width<=1.55 e petal_lenght<=5.45
    ou
    petal_lenght>2.45 e petal_width>1.75 e petal_lenght<=4.85 e sepal_lenght<=5.95
- Classe Virginica
  - ...

classe
Setosa, Versicolor, Virginica
Atributos
X[0]: sepal length
X[1]: sepal width
X[2]: petal length
X[3]: petal width



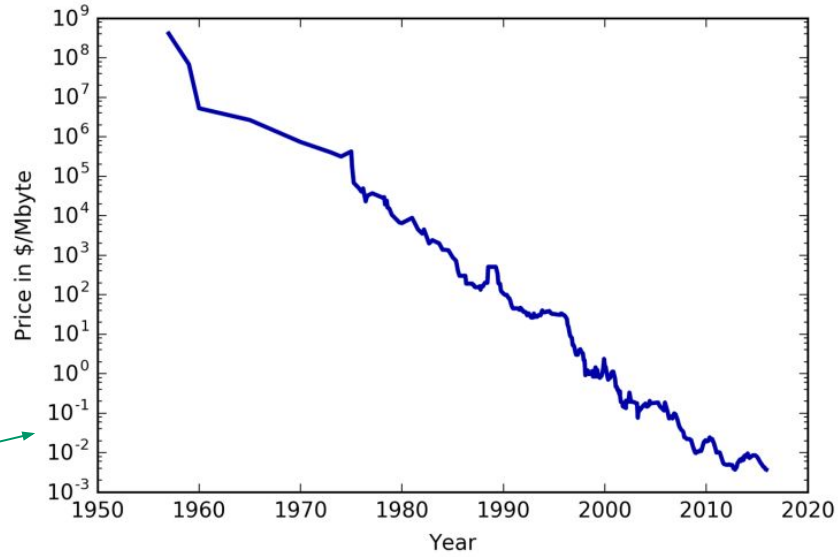
Regressão

Árvore de regressão

- Utiliza exatamente o mesmo algoritmo que a classificação
- Previsão
  - percorre a árvore com base nos testes em cada nó até chegar a uma folha
  - o resultado é o valor médio dos exemplos de treino da folha
- Extrapolação não é possível
  - o modelo não consegue prever valores fora do intervalo do conj de treino

Exemplo

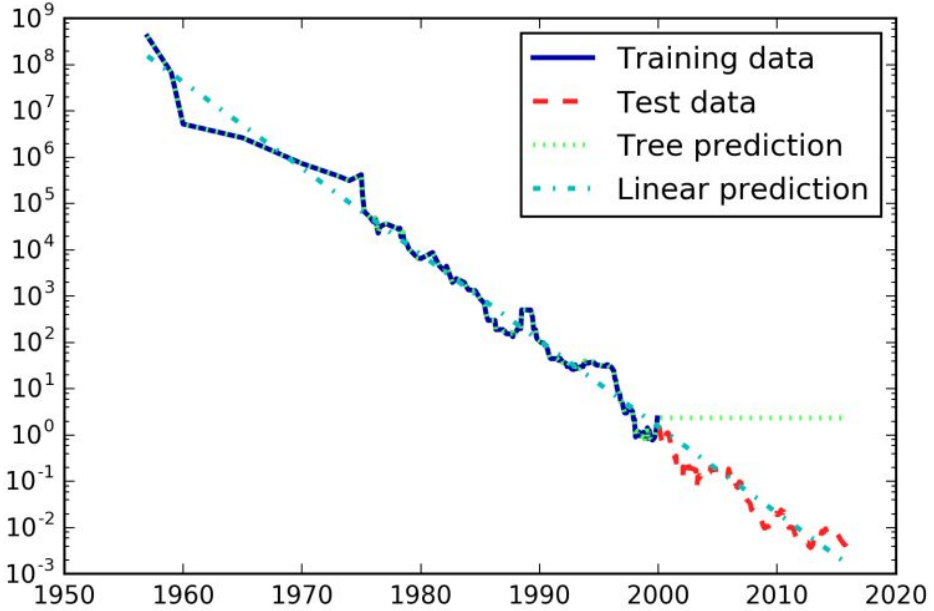
- Preço da RAM ao longo do tempo
- Experiência
  - treinar com dados até 2000
  - testar com dados posteriores
  - dados escalados com a função logaritmo
    - importante para o modelo linear



escala logarítmica

Geração de "novas" respostas

- Modelo linear
  - boa previsão de novos dados
  - variações no tempo são esbatidas
- Árvore
  - faz previsões perfeitas no treino
  - prevê sempre o mesmo valor no teste



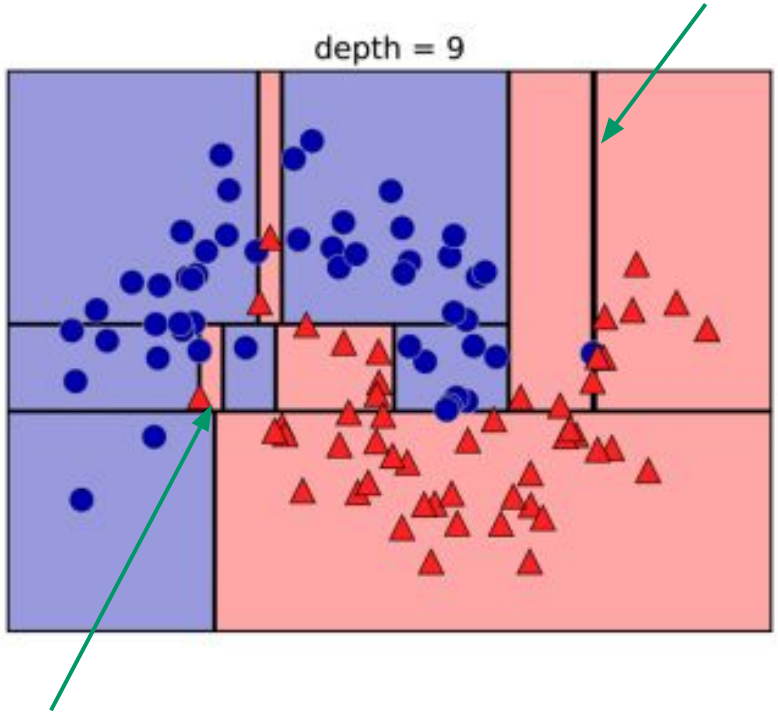
Poda da árvore

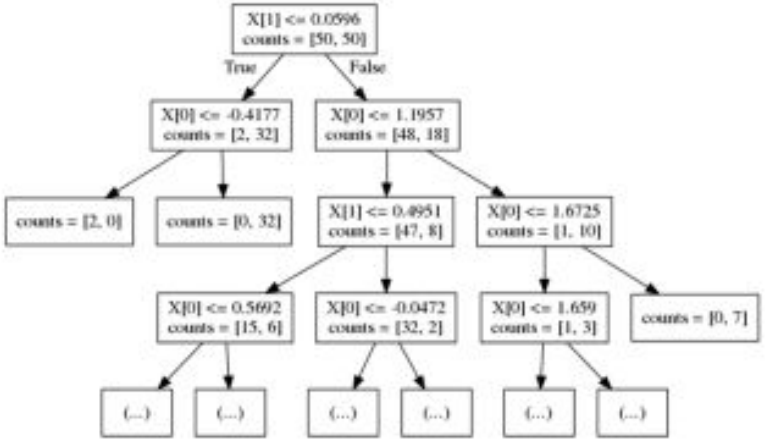
Fontes de incerteza nos dados

- Em problemas reais pode existir incerteza nos dados
- Tipos
  - ruído
    - medições erradas: valores do atributo e/ou classe
    - valores em falta
  - variação residual
    - fatores alheios não registados mas que afetam os resultados
- Consequência
  - árvores muito grandes
    - muitos ramos refletem ocorrências casuais e não relações subjacentes
    - improvável ocorrerem noutros exemplos

Sobre-ajustamento!

Sobre-ajustamento





Poda (pruning)

- Objetivo
  - identificar os ramos menos confiáveis e removê-los
- Resultado
  - modelo mais geral
    - erros no conj treino aumentam; erros no conj teste diminuem
- Tipos de pruning
  - pre-pruning
  - post-pruning

Pre-pruning

- Limita o crescimento da árvore
- Métodos
  - limitar profundidade
  - limitar nº de folhas
  - limitar o nº mínimo de exemplos em cada nó

Post-pruning

- O corte da árvore é feito depois da árvore completa ser construída
  - é necessário um conj de dados distinto do conj de teste (conj. de poda) para a escolha da melhor árvore podada
- Métodos
  - cost-complexity (error-complexity)
  - reduced-error
  - critical value
  - minimum-error
  - pessimistic error
  - ...

Cost-complexity pruning

- Medida de cost-complexity
  - $R(T)$: erro de classificação nas folhas
  - $|T|$: nº de folhas

$$R_\alpha(T) = R(T) + \alpha|T|$$

- Considera
  - erro de classificação
  - complexidade da árvore
- Algoritmo
  - produz uma série de árvores podadas
  - seleciona aquela com o menor erro sobre um conj. dados de poda (diferente do usado para construir a árvore)

Reduced-error pruning

- Algoritmo
  - Treina a árvore de decisão
  - Utilizando o conj de poda, e para **cada nó interno**, conta o nº de erros quando
    - o nó torna-se folha vs. a sub-árvore é mantida
    - a diferença (se positiva) é uma medida do ganho obtido com a poda
  - De todos os nós, escolhe, como sub-árvore para podar, aquele com a maior diferença
  - Continua até que não seja possível diminuir mais o erro
- Características
  - Utiliza o conj. de poda para gerar as árvores podadas
  - Produz a versão mais pequena da árvore mais correta sobre o conj. de poda

Reduced-error pruning (2)

- Algoritmo 2
  - Treina a árvore de decisão
  - Utilizando o conj de poda, e para **cada nó apenas com folhas**, conta o nº de erros quando
    - se torna uma folha e a sub-árvore é mantida
    - a diferença (se positiva) é uma medida do ganho obtido com a poda
  - De todos os nós testados, escolhe aquele com a maior diferença tornado-o uma folha
  - Continua até que não seja possível diminuir mais o erro

Exemplo

- Breast cancer
  - 30 atributos, 2 classes
- Avaliação
  - treino: 75%, teste: 25%
- Medida de desempenho
  - exatidão
- Algoritmo
  - Árvore de decisão (CART)

- Desempenho (sem poda)
  - treino: 1.000
  - teste: 0.937
- Desempenho (prof-max=4)
  - treino: 0.998
  - teste: 0.951

Parâmetros e características

Parâmetros

- Função de impureza
  - Gini, entropia, ...
- Pre-pruning
  - max_profundidade, max_folhas, min_exemplos_folha
- Post-pruning
  - cost_complexity, reduced_error, critical_value, ...

Características, Algoritmos e Variações

Características

- Algoritmo
  - rápido para criar modelo
  - rápido para prever classe
- Pontos fortes
  - modelo facilmente visualizável e inteligível
  - algoritmo **invariante** à escala dos dados
- Pontos fracos
  - tendência para sobre-ajustamento
  - não garante a criação da melhor árvore (algoritmo guloso)
  - árvore pode ser **tendenciosa** se os dados não forem equilibrados (relativamente à classe) ***Enviesamento***
  - modelo instável: uma **variação** nos dados pode gerar uma árvore completamente diferente ***Variância***

Algoritmos

- ID3 (Iterative Dichotomiser 3)
  - Ross Quilan, 1986
  - Classificação
  - entropia, atributos nominais
- C4.5
  - Ross Quilan, 1993
  - Classificação, extensão do ID3
  - atributos nominais e contínuos, valores desconhecidos, atributos com custos diferentes, post-pruning
  - gera um conj de regras if-then-else a partir da árvore
- CART (Classification And Regression Trees)
  - Leo Breiman, 1984
  - Classificação e regressão
  - Árvores binárias
  - Gini, atributos nominais e contínuos, valores desconhecidos, pre- e post-pruning

Variações

- Partição de atributos contínuos
  - combinação linear de vários atributos
  - comparação entre valores de 2 ou mais atributos
  - ...
- Pesquisa da melhor partição
  - testar um subconjunto (aleatório) de atributos
  - utilização de algoritmos evolucionários para evitar decisões ótimas locais

Aprendizagem Automática

Avaliação de modelos

Teresa Gonçalves, Universidade de Évora
tcg@uevora.pt

Sumário

- Avaliação de modelos
  - Divisão treino-teste
  - Validação cruzada
- Afinação de parâmetros
- Medidas de desempenho
  - classificação binária
  - classificação multi-classe
  - regressão

Avaliação de modelos

Avaliação de modelos

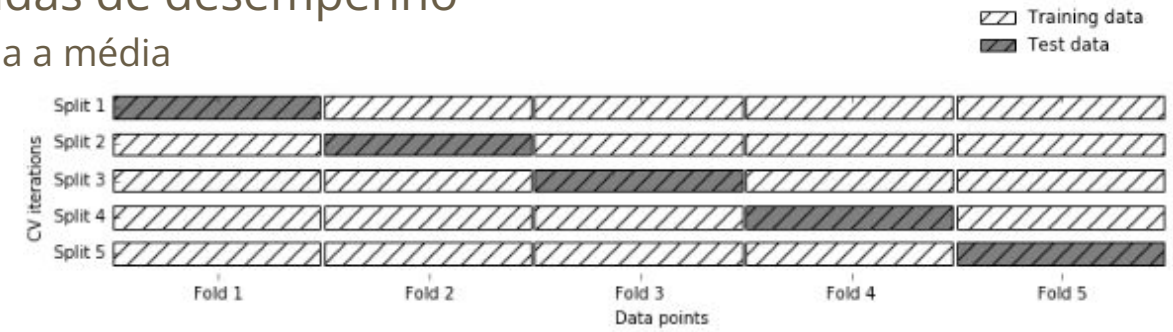
- Objetivo
  - Verificar quão bem o modelo generaliza
- Procedimento
  - divisão treino-teste
    - não interessa o ajuste ao conj de treino
    - mas sim, quão bem consegue fazer previsões em dados não observados

Validação cruzada

- Forma mais robusta de avaliar a generalização de um modelo
  - procedimento mais estável e completo
- Procedimento
  - Os dados são divididos repetidamente e são treinados múltiplos modelos
- Validação cruzada k-pastas
  - k: parâmetro
    - usualmente 5 ou 10
    - nº de sub-conjuntos em que o conj de dados é partido
      - todos com sensivelmente o mesmo tamanho

Validação cruzada k-pastas

1. partir o conj em k "pastas"
2. treinar k modelos
   - no modelo 1 a primeira "pasta" constitui o conj de teste; as restantes "pastas" (2 a k) constituem o conj de treino
     - o treino é feito com exemplos das pastas {2, ..., k} e avaliado com exemplos da pasta 1
   - depois a 2ª pasta constitui o teste e as restantes o treino
   - ...
3. no final existem k medidas de desempenho
   - normalmente é calculada a média



Exemplo

- Regressão logística sobre conjunto iris
- Exatidão com K=5
  - 1.000
  - 0.967
  - 0.933
  - 0.900
  - 1.000
- Exatidão média
  - 0.960
- Existe uma variância relativamente grande
  - entre 90% e 100%
- Razões possíveis
  - o modelo é muito dependente das pastas usadas para o treino
  - consequência da dimensão reduzida do conjunto

Benefícios da validação cruzada

- Cada exemplo está numa pasta e cada pasta é o conj de teste uma vez
  - o modelo tem de generalizar bem para todos os exemplos para que todos os valores de desempenho (e a sua média) sejam altos
- Múltiplas divisões
  - Fornece alguma informação sobre quão sensível o modelo é à seleção do conj de treino
    - fornece uma ideia de como o modelo se vai comportar no melhor e pior cenários quando aplicado a novos dados
- Tamanho maior do conjunto de treino
  - K=10, treino: 90% dos dados
  - Utilização mais efetiva dos dados
    - maior quantidade de dados normalmente resulta em modelos mais exatos

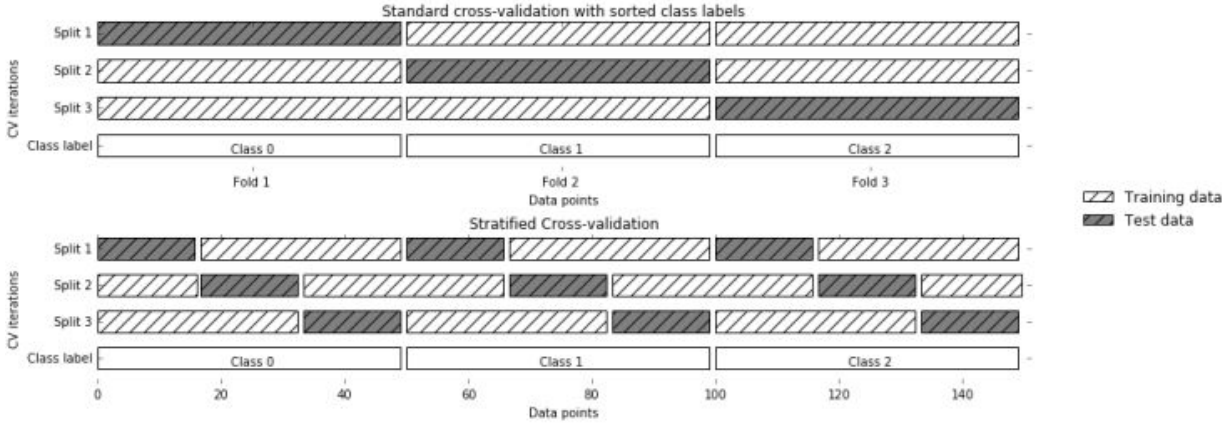
Desvantagens da validação cruzada

- Principal desvantagem
  - Custo computacional acrescido

- A validação cruzada não uma forma de construir um modelo!
- Objetivo
  - avaliar quão bem determinado algoritmo generaliza com um conjunto de dados específico

Validação cruzada k-pastas estratificada

- Mantém a proporção entre classes do conjunto completo em cada pasta
  - resulta em estimativas mais confiáveis do desempenho de generalização



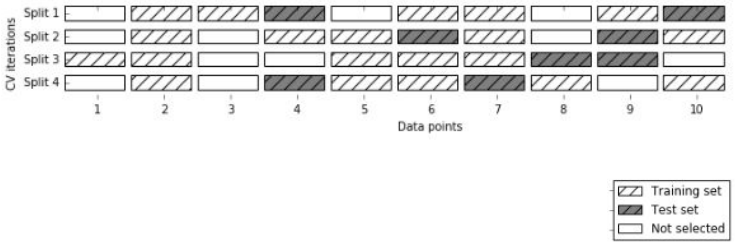
Validação cruzada leave-one-out

- Funcionamento
  - Validação cruzada k-pastas, com k=num exemplos
- Estratégia muito demorada
  - principalmente para conj de dados grandes
- Pode fornecer melhores estimativas para conj de dados pequenos

Validação cruzada shuffle-split

- Parâmetros
  - nº de exemplos para o treino
  - nº exemplos para o teste
  - nº de iterações
- Formação dos conjuntos
  - exemplos escolhidos aleatoriamente com reposição
  - garante-se que um exemplo está apenas num conjunto
- Características
  - permite a definição do nº de iterações (independentemente do tamanho do treino e teste)
  - permite utilizar apenas parte dos dados
  - pode ser útil para conj grandes
  - pode ser estratificado

- Exemplo
  - nº total exemplos: 10
  - nº exemplos treino: 5
  - nº exemplos teste: 2
  - iterações: 4



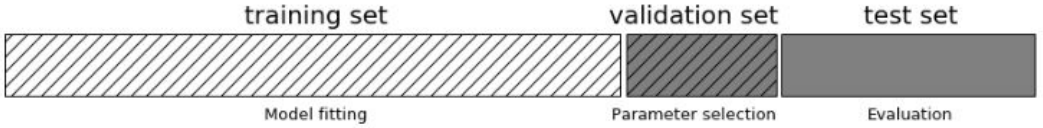
Afinação de parâmetros

Afinação de parâmetros

- O que é?
  - Tarefa de encontrar os valores dos parâmetros (importantes) de um algoritmo que originam o modelo com o melhor desempenho de generalização
- É uma tarefa morosa, mas necessária!

Conjunto de validação

- Avaliação dos modelos obtidos com diferentes valores dos parâmetros
  - deve ser feita com um novo conjunto (que não o de treino nem o de teste) pelas mesmas razões usadas para dividir o conjunto inicial em treino e teste
- Divisão dos dados
  - conj treino: usado para construir o modelo
  - **conj validação**: usado para escolher os parâmetros
  - conj teste: usado para avaliar o desempenho dos parâmetros selecionados



Construção do modelo

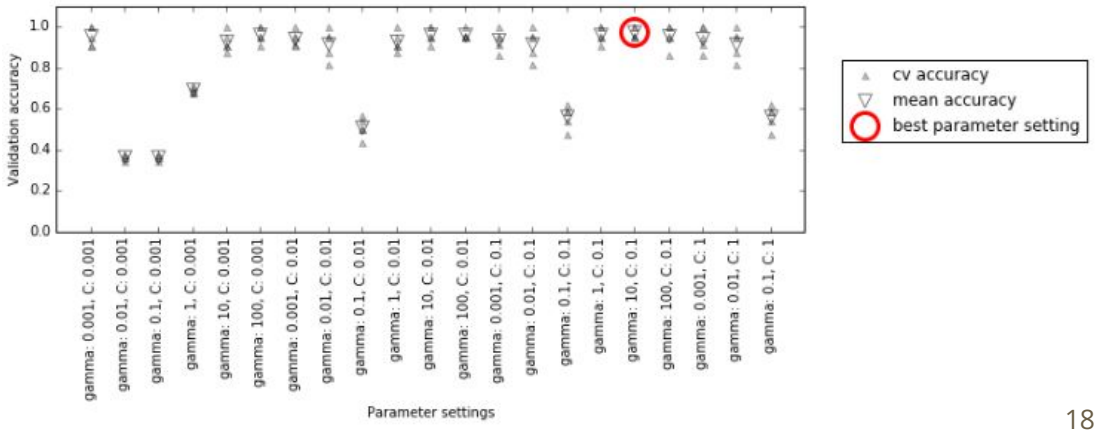
1. Dividir os dados em treino, validação, teste
2. Construir modelos usando o **conj de treino** com diferentes valores de parâmetros
3. Avaliá-los usando o **conj de validação**
4. Selecionar os valores de parâmetros que dão origem ao melhor desempenho
5. Reconstruir um modelo usando os "melhores" parâmetros com os dados de treino e validação
    - Desta forma, podemos usar o máximo de dados possível para construir o modelo
6. Avaliar o modelo final usando o **conj de teste**

Grid search

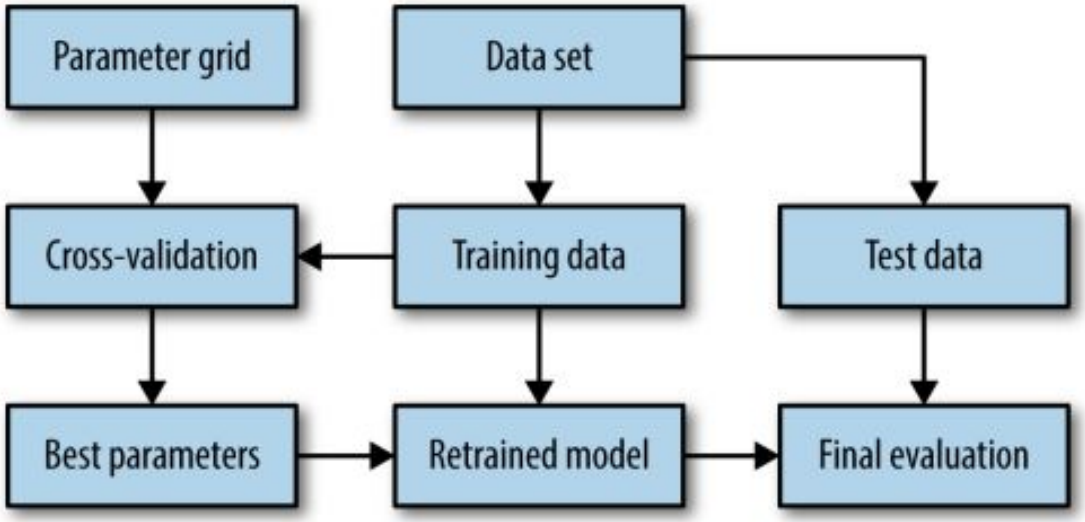
- Método de pesquisa
  - tenta todas as combinações possíveis de parâmetros
- Exemplo
  - SVM com núcleo RBF
    - Parâmetros
      - C: 0.001, 0.01, 0.1, 1, 10, 100
      - gamma: 0.001, 0.01, 0.1, 1, 10, 100
    - total de 36 combinações

Grid search e validação cruzada

- Divisão treino/validação/teste
  - método sensível à forma como os dados são divididos
- Melhor estimativa do desempenho dos modelos
  - utilizar validação cruzada para avaliar o desempenho de cada combinação de parâmetros
- Exemplo
  - xval 5 pastas
  - calcular a média do desempenho para cada conj de parâmetros
  - escolher os parâmetros que dão origem à melhor média



Processo de pesquisa do melhor modelo

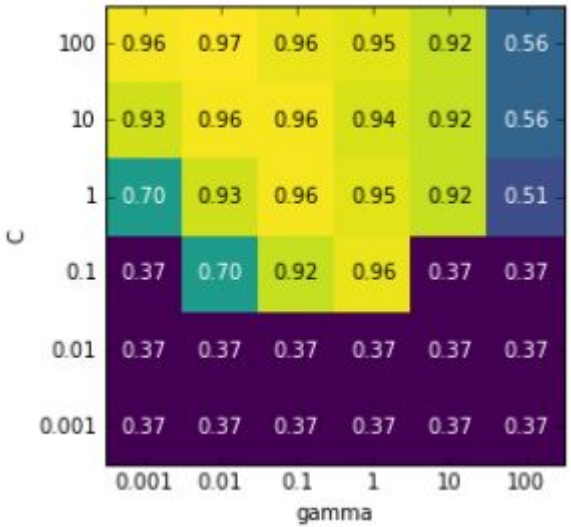


Drenagem de informação

- Drenagem de informação
  - Quaisquer escolhas feitas com base no desempenho do conj de teste drenam informação do conj de teste para o modelo
  - Resulta numa estimativa otimista do desempenho
- Boa prática
  - fazer todas as análises exploratórias e seleção de modelos usando a combinação dos conjs de treino e de validação
  - reservar o conjunto de teste APENAS para a avaliação final

Heatmap do desempenho

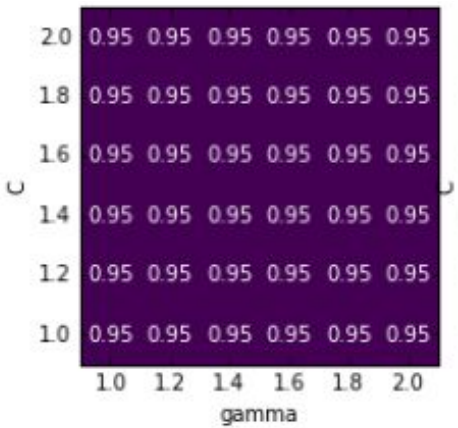
- Visualização do desempenho para cada conj de parâmetros
  - cada ponto corresponde à execução da validação cruzada com parâmetros particulares
  - as cores codificam o desempenho
    - cores escuras <-> desempenho baixo
    - cores claras <-> desempenho alto
- Exemplo
  - SVM com núcleo RBF
    - C, gamma: 0.001, 0.01, 0.1, 1, 10, 100
  - Conclusões
    - Algoritmo muito sensível à escolha dos parâmetros
    - Os parâmetros C e gamma são importantes
    - O intervalo dos parâmetros é grande o suficiente



Grelhas de pesquisa mal especificadas

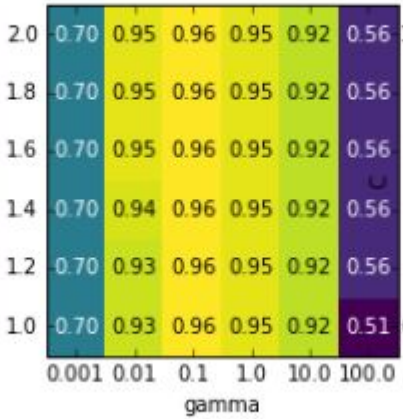
Não há variação do desempenho

- Razões possíveis
  - intervalo de variação não apropriado
  - os parâmetros não são importantes



Apenas gamma influencia desempenho

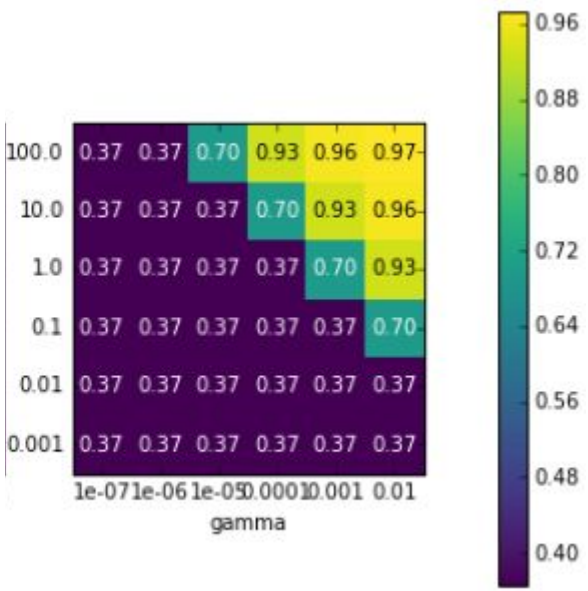
- Razões possíveis
  - pesquisa feita sobre intervalos de gamma interessantes mas não de C
  - o parâmetro C não é importante



Grelhas de pesquisa mal especificadas

Ambos os parâmetros influenciam o resultado

- Conclusões
  - podemos excluir valores pequenos dos parâmetros
  - os melhores de parâmetros estão na fronteira. É possível encontrar ainda melhores desempenhos?



Medidas de desempenho

Problema específico a resolver

- Exemplos de problemas
  - Evitar acidentes de trânsito
  - Diminuir o número de internamentos hospitalares
  - Atrair mais utilizadores para para um site
  - Incentivar as compras numa loja
- O objetivo de alto nível influencia quer os algoritmos utilizados, quer a escolha de **medidas de desempenho**

Métricas para classificação binária

- Classificação binária
  - aprendizagem de conceito
  - classes: **positiva**, negativa
- Exatidão
  - Nem sempre é a melhor medida porque existem **diferentes tipos de erro**
- Exemplo
  - deteção precoce de cancro
  - tipos de erro
    - paciente são ser diagnosticado como doente (falso positivo)
    - paciente doente ser diagnosticado como são (falso negativo)
  - As consequências de cada tipo de erro são muito distintas

Matriz de confusão (reminder...)

- Desempenho
  - Diagonal descendente indica as previsões corretas
  - Diagonal ascendente indica erros de predição
- Terminologia
  - VN: verdadeiros negativos
  - FP: falsos positivos
  - FN: falsos negativos
  - VP: verdadeiros positivos
  - Marginais
    - Última linha e coluna

	Prevista -	*Prevista +*	
Actual -	**VN**	FP	*Neg=TN+FP*
Actual +	FN	**VP**	*Pos=FN+TP*
	Marginais de predição		*# instâncias*

Exemplo

- 50 instâncias negativas
  - 40 classificadas corretamente
  - 10 mal classificadas
- 50 instâncias positivas
  - 30 classificadas corretamente
  - 20 mal classificadas

	Prevista -	*Prevista +*	
Real -	**40**	10	50
Real +	20	**30**	50
	60	40	100

Conjuntos de dados não balanceados

- O que são?
  - conjuntos onde o nº de exemplos de uma classe é muito maior que da outra
  - na realidade são os mais comuns...
- Exemplo
  - classe negativa: 99% dos exemplos
  - classe positiva: 1% dos exemplos
  - Qual a exatidão de um sistema (não inteligente) que indica a classe negativa para todos os exemplos?

Exemplo

- Classes
  - negativa: 90% dos exemplos
  - positiva: 10% dos exemplos
- Classificadores "dummy"
  - Prevê a classe mais frequente
  - Prevê a classe aleatoriamente (tendo em conta a distribuição de classes
- Exatidão
  - mais frequente: 0.90
  - "dummy" aleatório: 0.80
  - árvore de decisão: 0.92
  - regressão logística: 0.98
- A exatidão não é a medida adequada!

Exemplo: matrizes de confusão

- Mais frequente (exatidão: **0.90**)

403	0
47	0

- *Dummy* aleatório (exatidão: 0.80)

361	42
43	**4**

- Árvore de decisão (exatidão: **0.92**)

390	**12**
24	23

- Regressão logística (exatidão: 0.98)

401	**2**
8	39

Indicadores de desempenho

- Calculados sobre a matriz de confusão
- Existe uma grande variedade de indicadores
  - Exatidão (accuracy)
  - Taxa de erro (error rate)
  - Taxa de verdadeiros positivos, Taxa de verdadeiros negativos
    - Pode ser visto como uma exatidão por classe
  - Taxa de falsos negativos, Taxa de falsos positivos
    - Pode ser visto como uma taxa de erro por classe
  - Precisão (precision)

Exatidão e taxa de erro

- Exatidão
  - Proporção de instâncias de teste **corretamente** classificadas

$$Acc = \frac{TP + TN}{Pos + Neg}$$

- Taxa de erro
  - Proporção de instâncias **incorretamente** classificadas

$$Err = \frac{FP + FN}{Pos + Neg}$$

	Prev -	*Prev +*	
Real -	**TN**	FP	Neg
Real +	FN	**TP**	Pos

Taxa de verdadeiros positivos e taxa de verdadeiros negativos

- Taxa de verdadeiros positivos
  - Proporção de instâncias positivas classificadas corretamente
  - Também conhecida como **sensibilidade** ou **cobertura**
  - Usada quando é importante **evitar falsos negativos**

$$rec = \frac{TP}{TP + FN}$$

- Taxa de verdadeiros negativos
  - Proporção de instâncias negativas classificadas corretamente
  - Também conhecida como **especificidade** ou **cobertura negativa**

$$tnr = \frac{TN}{TN + FP}$$

Taxa de falsos negativos e taxa de falsos positivos

- Taxa de falsos negativos
  - Proporção de instâncias positivas mal classificadas

$$fnr = \frac{FN}{Pos}$$

- Taxa de falsos positivos
  - Proporção de instâncias negativas mal classificadas
  - Também conhecida como **falso alarme**

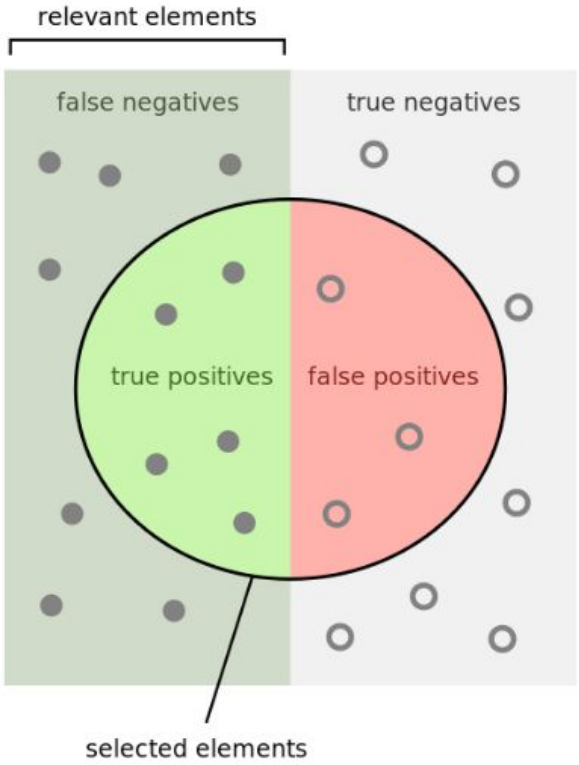
$$fpr = \frac{FP}{Neg}$$

Precisão

- Precisão
  - Proporção de verdadeiros positivos entre os classificados como positivos
  - Também conhecida como **confiança**
  - Usada quando se quer **limitar** o nº de **falsos positivos**

$$prec = \frac{TP}{TP + FP}$$

Exemplos relevantes e selecionados



How many selected items are relevant?

$$\text{Precision} = \frac{\text{true positives}}{\text{selected elements}}$$

How many relevant items are selected?

$$\text{Recall} = \frac{\text{true positives}}{\text{relevant elements}}$$

Compromisso entre precisão e cobertura

- Modelo que prevê todos os exemplos como positivos
  - não tem falsos negativos nem verdadeiros negativos
  - muitos falsos positivos
    - cobertura = 1
    - precisão baixa
- Modelo que apenas prevê como positivo um exemplo (aquele que há mais certeza)
  - tem no máximo um falso positivo
    - precisão alta
    - cobertura baixa

Medida F_β

- Considera a precisão e cobertura
  - β: parâmetro que valoriza mais a precisão ou cobertura quando comparada com a outra

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}$$

- Valores comuns
  - β=1
    - dá o mesmo peso à precisão e cobertura
  - β=2
    - dá mais peso à cobertura
  - β=0.5
    - dá mais peso à precisão

Medida F1

- Média harmónica da precisão e cobertura

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

- melhor medida que a exatidão quando os dados são não balanceados
    - ... mas mais dificil de interpretar

- Exemplo
    - Exatidão
        - mais frequente: 0.90
        - "dummy" aleatório: 0.80
        - árvore de decisão: 0.92
        - regressão logística: 0.98
    - F1
        - mais frequente: 0.0
        - "dummy" aleatório: **0.10**
        - árvore de decisão: **0.55**
        - regressão logística: 0.89

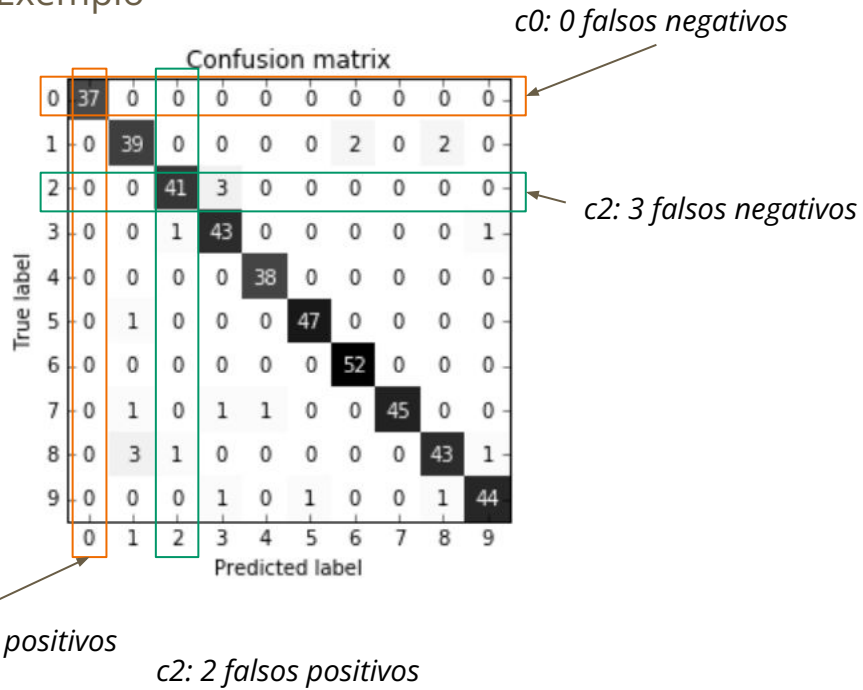
Adequação da medida de desempenho

- Diagnóstico de cancro
  - cobertura
    - é importante encontrar todas as pessoas doentes, possivelmente incluindo algumas saudáveis
- Previsão da eficácia de um medicamento num ensaio clínico
  - precisão
    - o modelo não deve produzir muitos falsos positivos devido ao alto custo dos ensaios

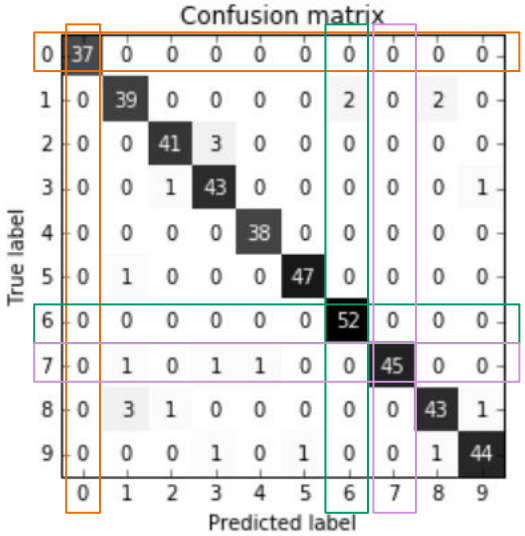
Métricas para classificação multi-classe

- São derivadas das métricas da classificação binária, pesadas pelas classes
- Base
  - Matriz de confusão
    - linhas: classes verdadeiras
    - colunas: classes previstas

Exemplo



Matriz de confusão e desempenho



	precision	recall	f1-score	support
0	1.00	1.00	1.00	37
1	0.89	0.91	0.90	43
2	0.95	0.93	0.94	44
3	0.90	0.96	0.92	45
4	0.97	1.00	0.99	38
5	0.98	0.98	0.98	48
6	0.96	1.00	0.98	52
7	1.00	0.94	0.97	48
8	0.93	0.90	0.91	48
9	0.96	0.94	0.95	47
avg / total	0.95	0.95	0.95	450

Medida de desempenho para multi-classe

- Calcular a medida para cada classe
  - sendo essa a classe positiva e as restantes a classe negativa
- Calcular a média
  - macro-média
    - média não pesada de cada classe
      - *usar quando todas as classes têm a mesma importância*
  - macro-média pesada
    - média pesada pelo nº de exemplos de cada classe
  - micro-média
    - soma-se o nº de falsos positivos, falsos negativos e verdadeiros positivos de todas as classes e depois calcula-se a medida
      - *usar quando todos os exemplos têm a mesma importância*

Métricas para regressão

- R^2
  - coeficiente de determinação
    - proporção da variância que é explicada pelas variáveis independentes
  - intervalo de variação
    - melhor valor: 1
    - pode ser negativo

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

- Erro quadrado médio

$$MSE(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- Erro absoluto médio

$$MAE(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Pontos a reter

Divisão do conj de dados

- Avaliação de modelo
  - Usar conj teste
- Construção do modelo
  - Usar uma divisão do conj de treino (treino e validação) para seleção do modelo e parâmetros
- Usar divisão simples ou validação cruzada
- Forma mais usada
  - divisão treino/teste para avaliação
  - validação cruzada para seleção do modelo e parâmetros

Escolha da métrica de avaliação

- A métrica usada para avaliar os modelos depende da aplicação
  - deve ser um bom substituto daquilo que o modelo será usado realmente
- Aplicações reais
  - os problemas de classificação raramente têm classes equilibradas
  - muitas vezes falsos positivos e falsos negativos têm consequências muito diferentes

Aprendizagem Automática

Comités de peritos

Teresa Gonçalves, Universidade de Évora
tcg@uevora.pt

Sumário

- Comités
- Estratégias
- Comités de árvores
  - Random Forest
  - Extra Trees
  - Gradient Boosting Machines

Comités

Comités de peritos

- Motivação
  - não existe um algoritmo que apresenta o melhor desempenho para qualquer conjunto!
  - (*no free lunch theorem*)
- Proposta
  - Combinar vários modelos para criar modelos mais poderosos!
- Obtenção de modelos
  - diferentes algoritmos
  - diferentes hiper-parâmetros
  - diferentes fontes de informação
  - diferentes conjuntos de treino

Combinação de peritos

- Múltiplos peritos (modelos)
  - Os modelos trabalham em paralelo
    - Cada instância é apresentada a todos os peritos que produzem a sua decisão
    - Um módulo reúne todas as respostas e apresenta a decisão final
- Múltiplas etapas
  - Os classificadores trabalham em série
    - Cada perito dá pesos diferentes aos exemplos ou "foca-se" num sub-conjunto dos exemplos

Votação

- Simples
  - todos os peritos têm o mesmo peso
- Pesada
  - utiliza probabilidades à posteriori

Estratégias

Estratégias

- Bagging
- Stacking
- Boosting
- Cascading

Bagging*

** Bootstrap aggregating*

- Método de múltiplos peritos
- Geração dos peritos
  - Cria L conjuntos de treino através do método *bootstrap*
- Agregação dos peritos
  - Votação simples dos L peritos

- *Bootstrap*
  - Dado um conj. de dados de tamanho N, retiram-se aleatoriamente com reposição, N exemplos

Desempenho

- É benéfico para algoritmos instáveis
  - pequenas alterações no conj. treino podem provocar grandes diferenças no modelo
  - Exemplo: árvores de decisão
- É robusto para dados com ruído

Stacking

- Método de múltiplos peritos
- Geração do peritos
  - Assume a existência de L peritos
- Agregação dos peritos
  - A combinação dos peritos é aprendida
  - O combinador é treinado com um conj de dados não utilizado na construção dos peritos
- Os peritos devem produzir previsões diferentes
  - algoritmos distintos e/ou diferentes representações ou projeções dos dados de treino
- Também conhecido como Stacked Generalization

Boosting

- Método de múltiplas etapas
- Geração dos peritos
  - O classificador seguinte é treinado com base nos erros dos classificadores anteriores
- Agregação dos peritos
  - Votação pesada

Algoritmo

- Algoritmo original
  - Divide aleatoriamente o conj. inicial em três: X_1 , X_2 , X_3
  - Utiliza X_1 e treina p_1
  - Classifica X_2 com p_1; utiliza todos os exemplos mal classificados de X_2 para construir p_2
  - Classifica X_3 com p_1 e p_2; os exemplos para os quais p_1 e p_2 discordam formam o conj. para criar p_3
  - Cada exemplo de teste é apresentado a p_1 e p_2. Se concordam, é essa a resposta, senão é a resposta de p_3
- AdaBoost (Adaptive Boosting)
  - Utiliza sempre o mesmo conjunto
  - Permite usar um número arbitrário de classificadores base
  - A votação é pesada pelo desempenho de cada classificador base

Desempenho

- Utiliza aprendizes fracos (*weak learners*)
  - algoritmos de aprendizagem simples
  - Exemplo: decision stumps (árvores de decisão de profundidade 1)
- É muito suscetível ao ruído e *outliers*

- Comparação
  - muitas vezes produz melhores classificadores que o *bagging*...
  - ... mas pode sofrer de sobre-ajustamento

Cascading

- Método de múltiplas etapas
- Os peritos base são ordenados por
  - complexidade de tempo e/ou espaço, ou
  - custo da representação utilizada
- O perito p_j é utilizado, se os classificadores precedentes (p_i, $i<j$) não forem confiáveis
- Cada perito p_j tem uma confiança associada w_j
  - p_j é confiável e pode ser utilizado, se $w_j > \theta_j$, com $1/K < \theta_j \leq \theta_{j+1}$ e K o nº de peritos
  - θ_j é o limiar de confiança de p_j

Base lógica

- Classificadores de complexidade crescente
  - Os classificadores iniciais (mais simples) tratam da maioria dos exemplos
  - Os classificadores mais complexos são utilizados numa pequena percentagem de exemplos

Comité de árvores de decisão

Comité de árvores de decisão

- Ideia
  - cada árvore é **diferente** das restantes
  - cada árvore pode fazer uma boa predição mas sofrer sobre-ajustamento em parte dos dados
  - no conjunto o poder preditivo das árvores mantém-se mas o sobre-ajustamento diminui ao agregar os seus resultados
- Algoritmos
  - random forest
  - extremely randomized trees (extra trees)
  - gradient boosting machines

Random Forest

Random forest

- Diversidade das árvores
  - introdução de processos **aleatórios** na construção
    - na seleção dos exemplos utilizados (conj treino)
    - na seleção dos atributos no teste de partição
- Estratégia bagging

Construção da árvore

- Criar uma amostra *bootstrap* dos dados
  - os conjs tem o mesmo nº de exemplos que o conj original
  - os exemplos são **escolhidos aleatoriamente com reposição**
  - alguns exemplos não estão presentes (cerca de 1/3), outros estão repetidos
- Criar a árvore com base na amostra
  - na escolha de um nó é escolhido aleatoriamente um **sub-conjunto de atributos** e pesquisada a melhor partição envolvendo um entre aqueles atributos
    - cada nó testa um sub-conjunto diferente de atributos

Previsão

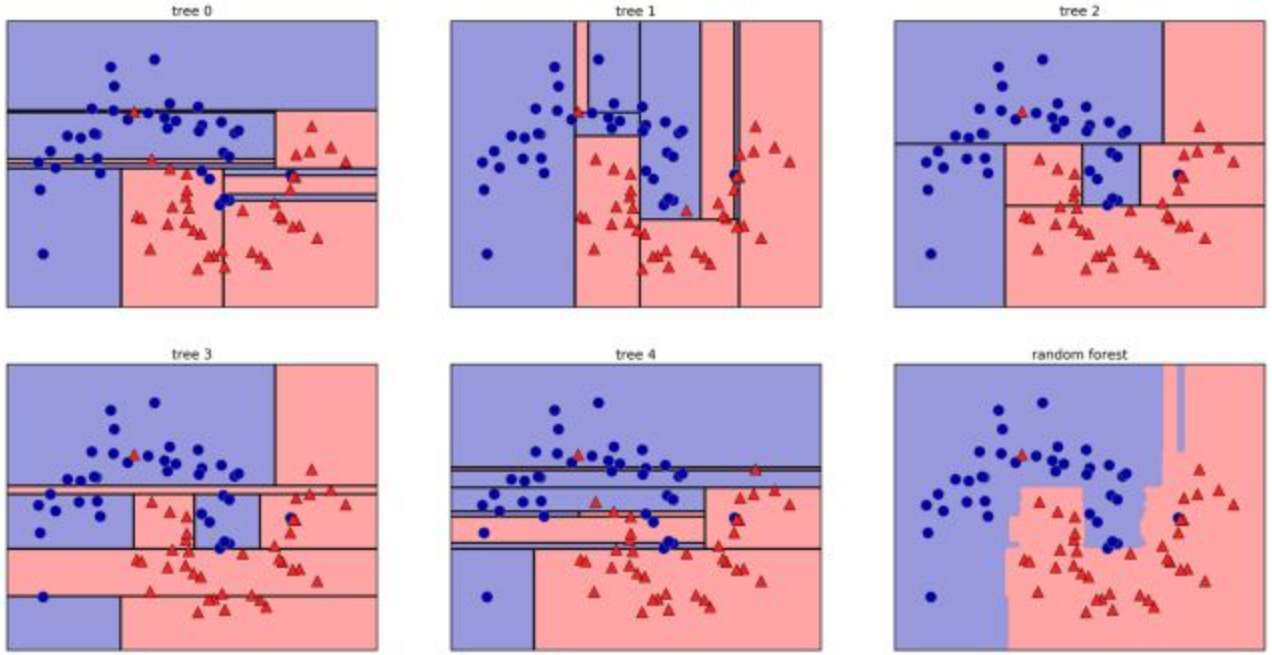
- Estratégia de votação "suave"
  - A previsão individual de cada árvore fornece uma probabilidade para cada classe
  - É calculado valor para cada classe, e aquela com probabilidade mais alta é escolhida
- Há implementações que usam votação simples

Número de atributos a testar

- Parâmetro crítico
  - se for igual ao total de atributos não há aleatoriedade
  - se for 1 não há escolha nos atributos a testar
- Valor alto
  - as árvores serão bastante semelhantes
  - o ajuste (aos dados) será mais fácil porque usa mais atributos
- Valor baixo
  - as árvores serão mais distintas
  - cada árvore poderá ser muito profunda para se ajustar aos dados

Exemplo

- Árvores individuais
  - As fronteiras de decisão são muito diferentes
  - Cada árvore erra alguns exemplos de treino porque não foram usados (no treino)
- Comité
  - tem menos sobre-ajustamento
  - fornece uma fronteira de decisão mais intuitiva



Características, vantagens e desvantagens

- Características
  - modelo poderoso
  - funciona bem sem grande esforço na afinação dos parâmetros
  - não é necessário escalar os dados
- Vantagens
  - É paralelizavel
  - Funciona bem em conjuntos muito grandes
- Desvantagens
  - Não funciona bem em dados com muitas dimensões e esparsos

Parâmetros importantes

- nº de árvores a criar
  - quanto mais árvores, melhor o modelo construído
    - fazendo a média de mais árvores torna o algoritmo mais robusto
  - com mais árvores é necessária mais memória e tempo para treinar
  - regra do "polegar"
    - tantas quantas a memória e o tempo permitir
- nº atributos a testar
  - regra do "polegar"
    - raiz quadrada do nº de atributos
- estratégia de poda (se necessário)
  - limitar a profundidade da árvore ou nº de folhas
    - pode reduzir drasticamente o tempo e espaço necessário ao treino e previsão

Mais informação

- É um dos métodos mais utilizados
  - tem todos os benefícios das árvores de decisão e resolve algumas das suas deficiências
- Normalmente são usadas centenas ou milhares de árvores!

Extremely Randomized Trees

Extremely Randomized Trees

- Diversidade das árvores
  - introdução de processos **aleatórios** na construção da árvore
    - na seleção dos atributos no teste de partição
    - pontos de corte (para atributos contínuos)
- Menos pesado computacionalmente que Random Forest
  - os pontos de corte são escolhidos aleatoriamente
- também conhecido como Extra Trees

Gradient boosting machines

Gradient boosting machines

- Diversidade das árvores
  - árvores construídas de forma sequencial
  - cada árvore tenta corrigir os erros da árvore anterior
  - forte pré-poda da árvore
    - árvores com profundidade um a cinco
- Estratégia *boosting*
- também conhecido como Gradient Boosted Regression Trees

Parâmetros

- nº de árvores a gerar
- profundidade da árvore
  - pré-poda
- taxa de aprendizagem
  - controla a força com que uma árvore tenta corrigir os erros da árvore anterior
    - valor maior: a árvore faz correções mais fortes, gerando modelos mais complexos

Relação entre parâmetros

- Grande correlação entre o nº de árvores e a taxa de aprendizagem
  - menor taxa implica a necessidade de mais árvores para construir um modelo de complexidade semelhante
- Prática comum
  - Ajustar o nº de estimadores (árvores) dependendo do tempo e memória disponíveis
  - Procurar diferentes taxas de aprendizagem

Exemplo: Breast cancer

- Parâmetros
  - árvores: 100
  - prof máxima: 3
  - taxa apr: 0.1
- Exatidão
  - treino: 1.000
  - teste: 0.958
- Possível sobre-ajustamento!

- Alteração da profundidade
  - prof máxima: 1
  - Exatidão
    - treino: 0.991
    - teste: 0.972

- Alteração da taxa de aprendizagem
  - taxa apr: 0.01
  - Exatidão
    - treino: 0.998
    - teste: 0.965

Comparação com Random Forest

- Menos pesado computacionalmente
  - a profundidade da árvore é muito menor
- Mais sensível à definição dos parâmetros
  - há menos aleatoriedade
- Nem sempre é melhor aumentar o nº de árvores
  - pode conduzir ao sobre-ajustamento

Estratégia

1. Experimentar random forest
   - produz modelos muito robustos
2. Experimentar gradient boosting
   - se o tempo de previsão for muito importante
   - se uma pequena diferença de desempenho for muito importante

Características, vantagens e desvantagens

- Características
  - algoritmo poderoso
  - algoritmo muito usado
- Vantagens
  - Não é necessário escalar os dados
  - Funciona bem numa mistura de dados binários e contínuos
- Desvantagens
  - Necessita de uma afinação de parâmetros cuidadosa
  - O tempo de treino pode ser longo
  - Muitas vezes não funciona bem em dados esparsos de grande dimensão

Mais informação

- Em competições, são os modelos vencedores com frequência
  - podem gerar melhores desempenhos que random forest se os parâmetros forem bem afinados
- São muito usados na indústria

Aprendizagem Automática

Redes neuronais

Teresa Gonçalves, Universidade de Évora
tcg@uevora.pt

Sumário

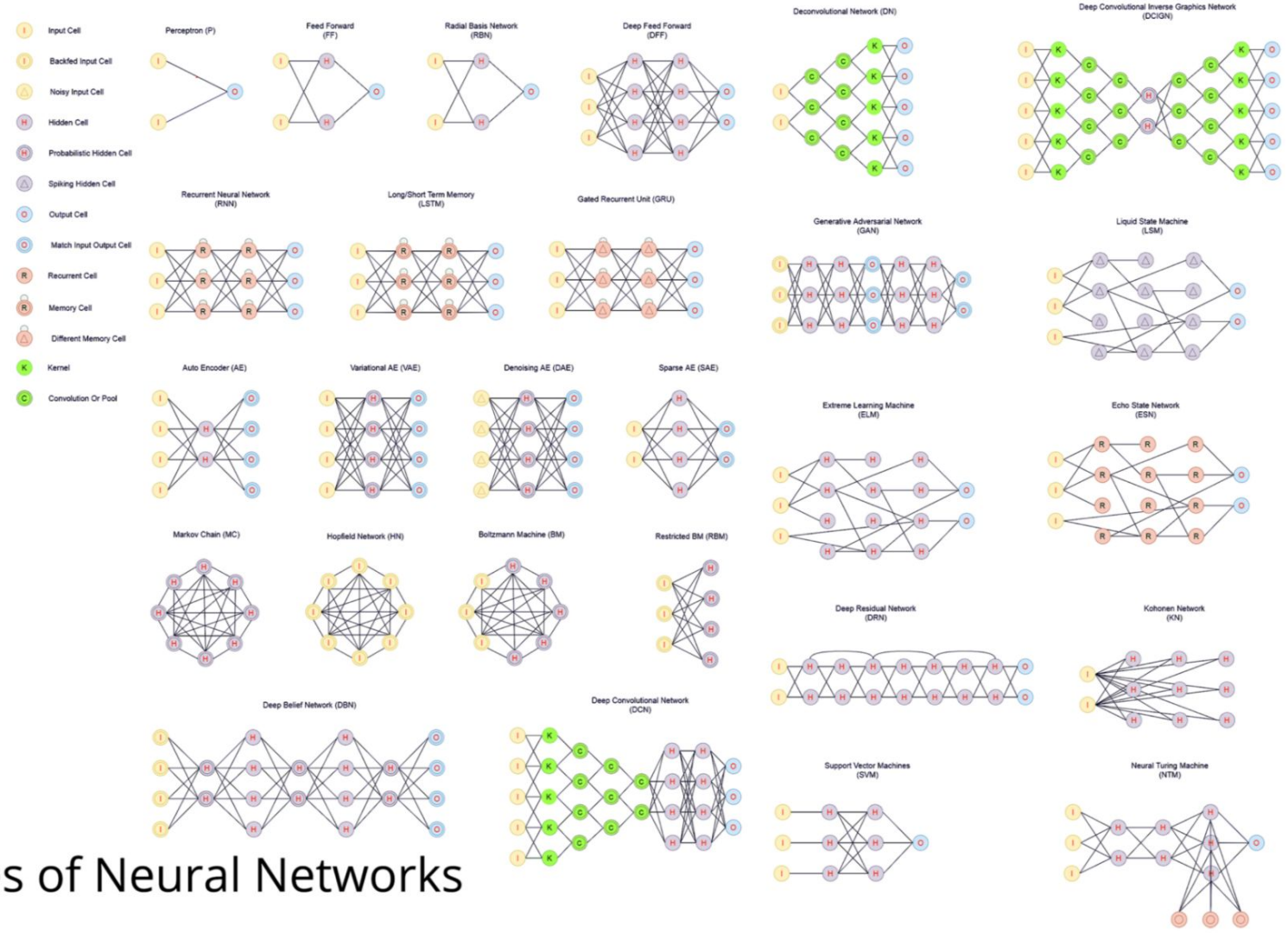
- Introdução
- Perceptrão
- Redes multi-camada
- Vantagens, desvantagens e parâmetros

Introdução

Redes Neuronais

- Rede Neuronal Feed-Forward
  - Perceptrão
  - Perceptrão multi-camada
  - Rede neuronal convolucional
  - Autoencoder
  - ...
- Rede de Função de Base Radial

- Rede Neuronal Recorrente
  - Recorrente completa
  - LSTM
  - Bi-directional
  - Rede Neuronal de Kohonen (Self Organizing)
  - ...
- Rede Neuronal Modular
- ...



Main Types of Neural Networks

Inspiração

- Forte interligação do cérebro humano
- Neurobiologia
  - A actividade do neurónio é activada/inibida através das ligações a outros neurónios
- Características do cérebro humano
  - Tempo de comutação: ~ 10^{-3} segundo
  - Neurónios: ~ 10^{11}
  - Ligações por neurónio: ~ 10^{4} (em média)
  - Tempo de reconhecimento: ~ 10^{-1} segundo
- Processamento altamente paralelo

Áreas de Aplicação

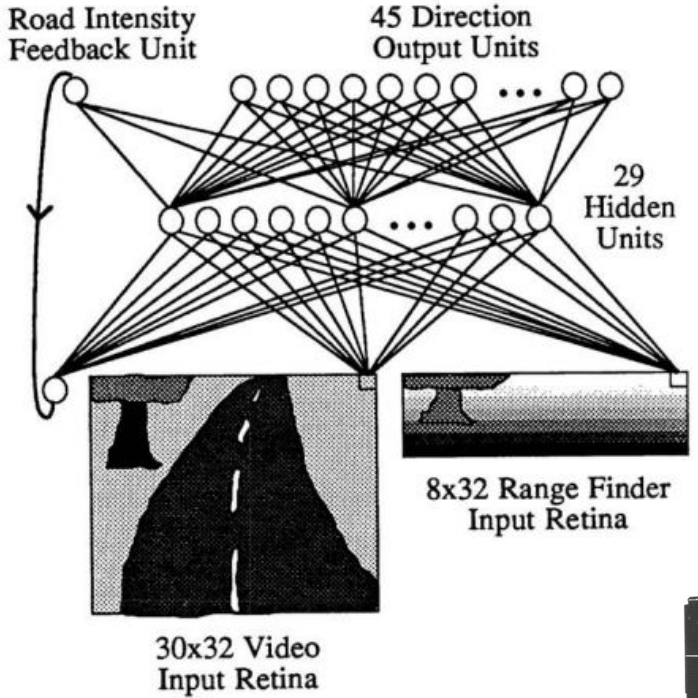
- Aprendizagem de informação sensorial complexa
  - Imagem
  - Voz
  - Texto
- Características
  - dados complexos e com ruído
  - exemplos representados por muitos pares atributo-valor
  - função objetivo discreta, real, vetor de valores discretos ou reais
  - tempo de treino longo
  - a explicabilidade não é importante

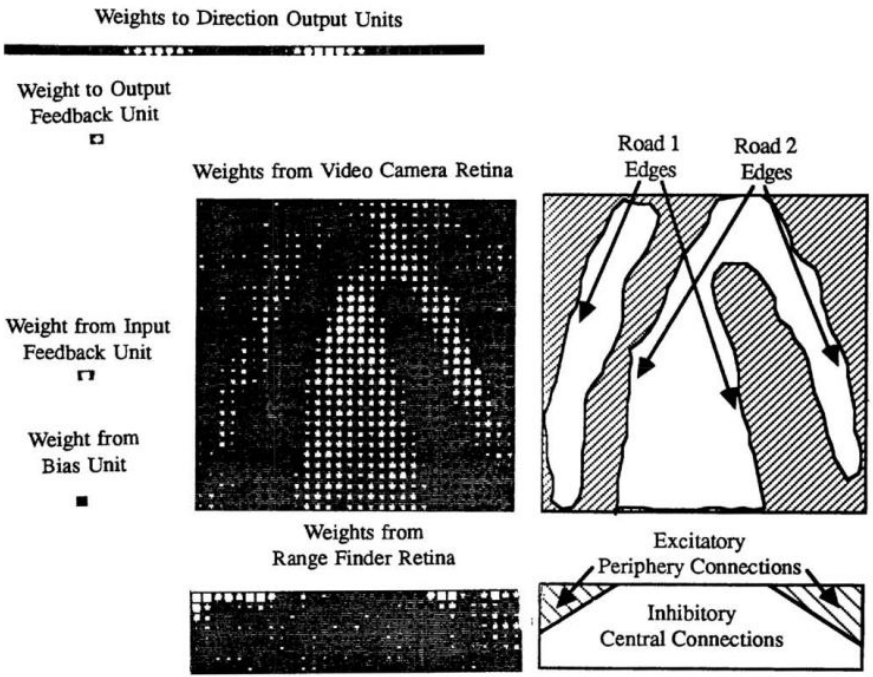
Exemplo 1

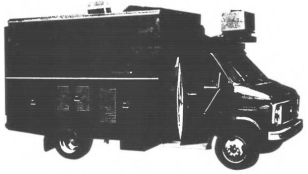
- ALVINN (Pomerleau, 1989)
  - Autonomous Land Vehicle In a Neural Network
- Objetivo
  - Condução de um veículo autónomo a velocidades normais numa auto-estrada pública

- Características
  - Entrada
    - imagem 30×32 pixels de intensidade
    - imagem 8×32 pixels de intensidade
  - Saída
    - direção de viragem do veículo
  - Treino
    - condução por um humano durante aprox. 5 minutos
  - Resultado
    - Condução, em auto-estrada, com velocidades até 70 mph numa distância de 90 milhas

ALVINN - sistema original

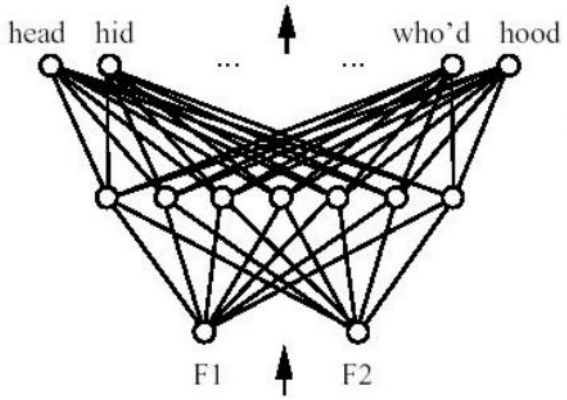


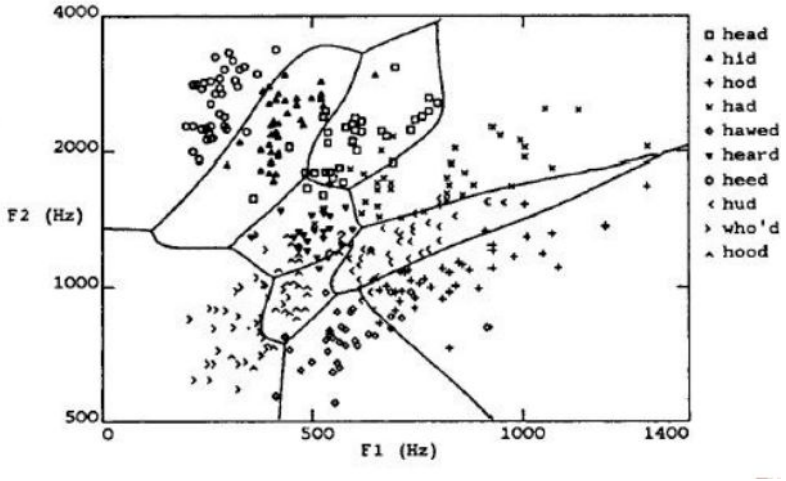




Exemplo 2

- Reconhecer 1 entre 10 sons
  - entrada: 2 valores (F1, F2) obtidos da análise espectral do som
  - saída: 10 unidades, uma para cada som (classe)





Perceptrão

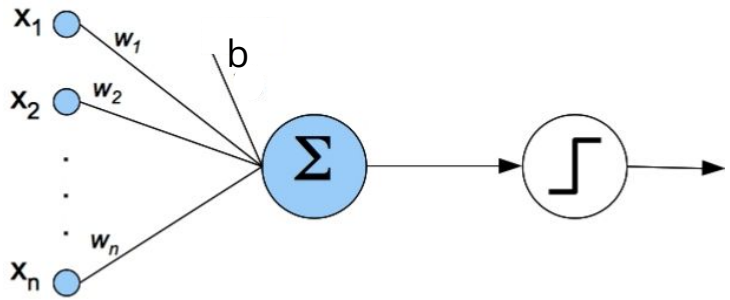
Modelo linear

- A saída é a soma pesada dos atributos de entrada x_1, ..., x_n
  - $y = w_1 * x_1 + ... + w_n x_n + b$
  - os coeficientes w_1, ..., w_n são parâmetros "aprendidos"
- Função de decisão
  - sgn(y)

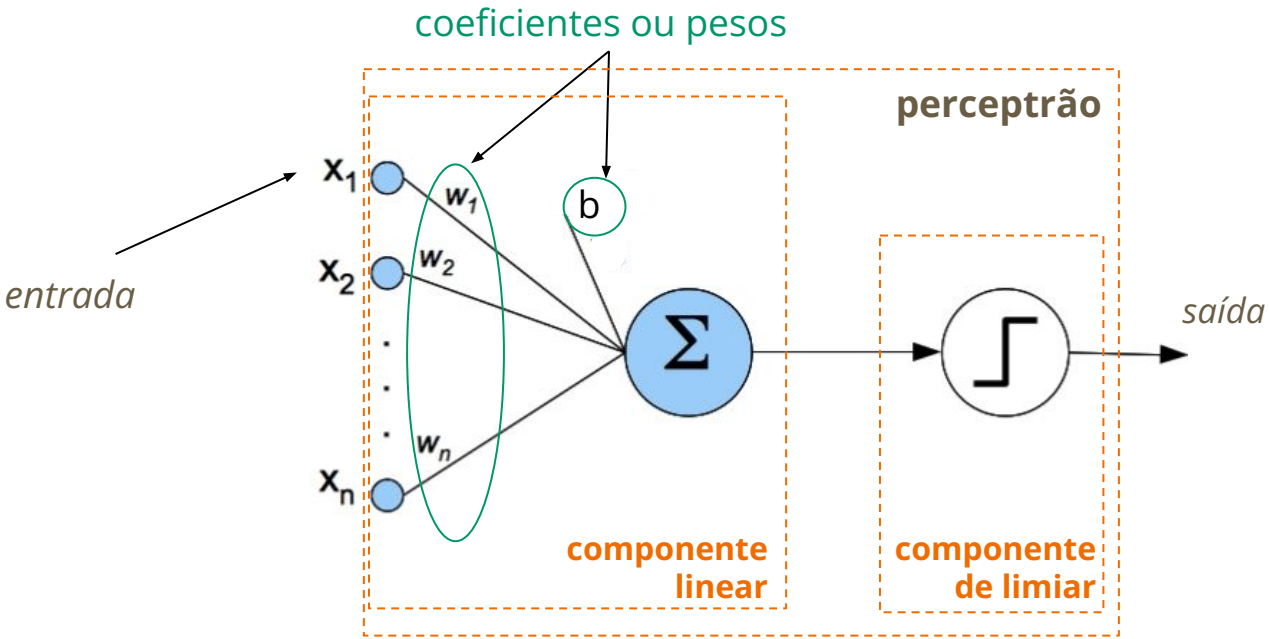
Perceptrão

- Modelo linear onde os coeficientes são determinados de forma iterativa
- Algoritmo
  - Atribuir pesos aleatórios a cada entrada x_i
  - Aplicar, iterativamente, o perceptrão a cada exemplo, modificando os pesos sempre que o exemplo for mal classificado
  - Repetir o processo até o perceptrão classificar correctamente todos os exemplos
- Treino do modelo
  - determinação dos coeficientes (pesos)

- Representação



Representação



Regra do perceptrão

- Atualização dos pesos
  - $w_i = w_i + \eta\ (t-o)\ x_i$
    - t : valor objetivo
    - o : valor saída perceptrão
    - η : ritmo de aprendizagem
- Convergência
  - converge quando os exemplos são linearmente separáveis
  - converge quando o ritmo de aprendizagem for suficientemente pequeno (<=0.1)

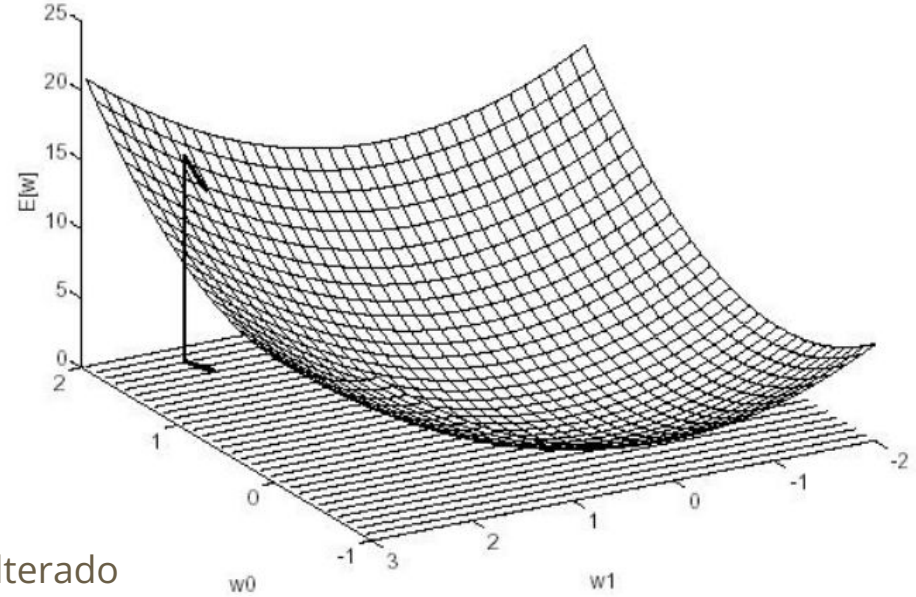
Regra do menor erro quadrático

- Atualização dos pesos
  - Utiliza o **gradiente** para pesquisar o espaço de possíveis vetores de peso
- Convergência
  - converge mesmo quando os exemplos de treino não são linearmente separáveis
- Também conhecida como
  - Regra delta
  - Regra Adaline
  - Regra Widrow-Hoff

Pesquisa

- Cálculo do erro
  - w: vetor de pesos
  - E: conj exemplos de treino
  - t_e: valor objetivo do exemplo e
  - y_e: saída da unidade linear

$$erro(\vec{w}) = \frac{1}{2}\sum_{e \in E}(t_e - y_e)^2$$



- Descida do gradiente
  - em cada iteração o vetor de pesos é alterado na direção que produz a descida mais íngreme na superfície de erro

Algoritmo 1: descida do gradiente

descida_gradiente(E, η) {

w_i = valor aleatório pequeno

Até a condição de terminação ser verdadeira

$dw_i = 0$

Para cada exemplo e=<X,t_e> em E

Calcular a saída y_e

Para cada peso w_i

$dw_i = dw_i + \eta (t_e - y_e) x_i$

Para cada peso w_i

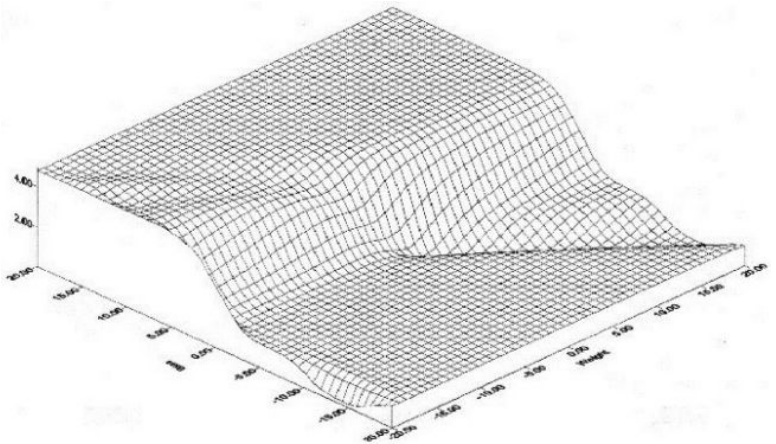
$w_i = w_i + dw_i$

}

*os pesos são atualizados após a apresentação de **todos** os exemplos de treino*

Algoritmo 2: descida do gradiente incremental

- Os pesos são atualizados após a apresentação de cada exemplo
  - desaparece a última atualização
  - a atualização dw_i é substituída por $w_i = w_i + \eta\ (t_e - y_e)\ x_i$
- Características
  - aproxima a descida do gradiente tanto quando se queira (se η for suficientemente pequeno)
  - requer menos cálculos por cada atualização de pesos
  - pode, por vezes, evitar cair nos mínimos locais, caso existam



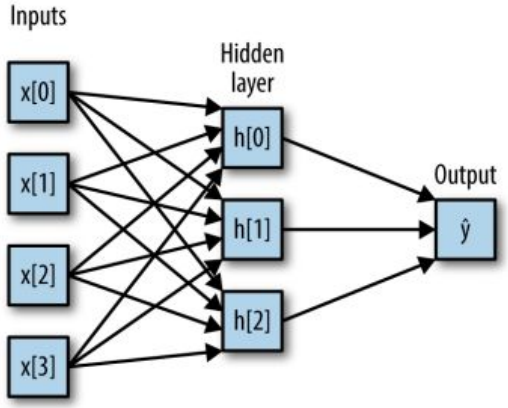
Resumo

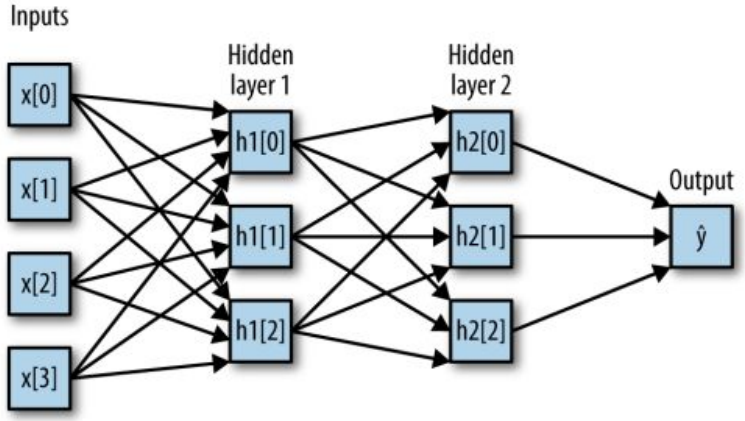
- Regra do perceptrão
  - atualiza os pesos de acordo com o erro à **saída do perceptrão**
  - converge após um número finito de iterações se
    - os exemplos de treino forem linearmente separáveis
    - ritmo de aprendizagem for suficientemente pequeno
- Regra do menor erro quadrático
  - atualiza os pesos de acordo com o erro na **saída da unidade linear**
  - converge assimptoticamente para a hipótese com o menor erro se ritmo de aprendizagem for suficientemente pequeno
    - mesmo quando os dados contêm ruído
    - mesmo quando os dados de treino não são linearmente separáveis

Redes multi-camada

Redes multi-camada

- Unidades básicas organizadas em camadas
- Topografia
  - uma camada de entrada
  - uma ou mais camadas escondidas
  - uma camada de saída



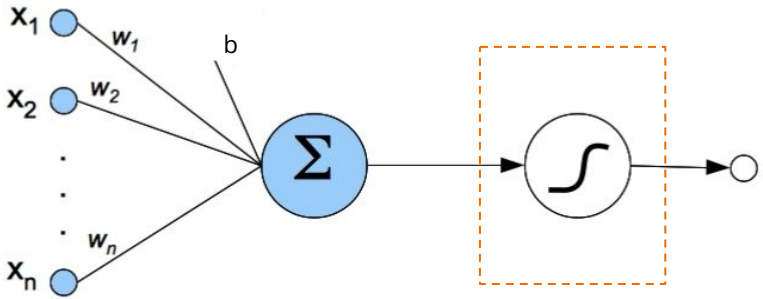


Camadas e unidades

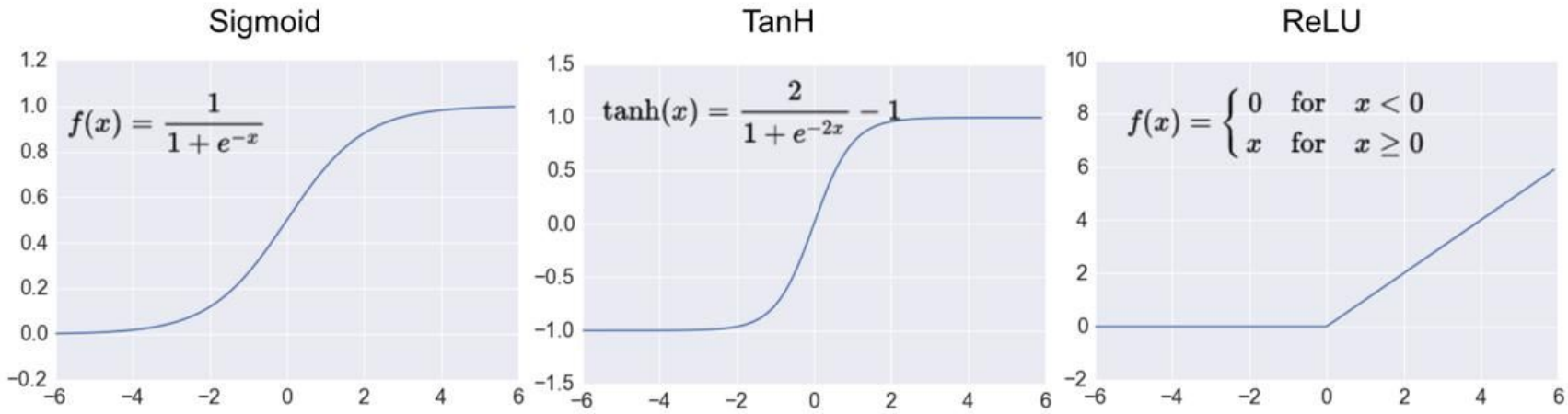
- Camada de entrada
  - atributos descritores
- Camada de saída
  - saídas (dependendo do tipo de problema pode ter um ou mais unidades)
- Camadas escondidas
  - número de unidades variável
  - número de camadas variável
- Unidades
  - Também conhecidas como
    - nós
    - neurónios

Componente de limiar

- À saída da componente linear é aplicada uma função **não linear** (que permite inibir ou ativar unidade)
- Funções
  - sigmoid
    - sigmoid()
  - tangens hyperbolicus
    - tanH()
  - rectifying nonlinearity
    - reLU()



Funções de ativação



Backpropagation

- Processo de atualização de pesos
- Funciona em 2 fases
  - Avanço dos padrões de entrada (*feedforward*)
    - cada unidade, começando nas de entrada, calcula a sua função de ativação e transmite-a a todas as unidades a que está ligada, propagando o sinal até às unidades de saída
  - Retrocesso da propagação dos erros (*backpropagation*)
    - cada unidade de saída compara a sua ativação com a saída desejada;
    - o erro é propagado "para trás" aos nós diretamente ligados à saída, ajustando os pesos das ligações com base no erro

Convergência do algoritmo

- Garante a convergência para um mínimo local
  - mas pode não ser o global...
- Variações
  - utilizar gradiente ou gradiente incremental
  - adicionar momento à regra de atualização dos pesos
- Porque o mínimo global não é garantido
  - treinam-se vários modelos inicializando a rede com pesos diferentes

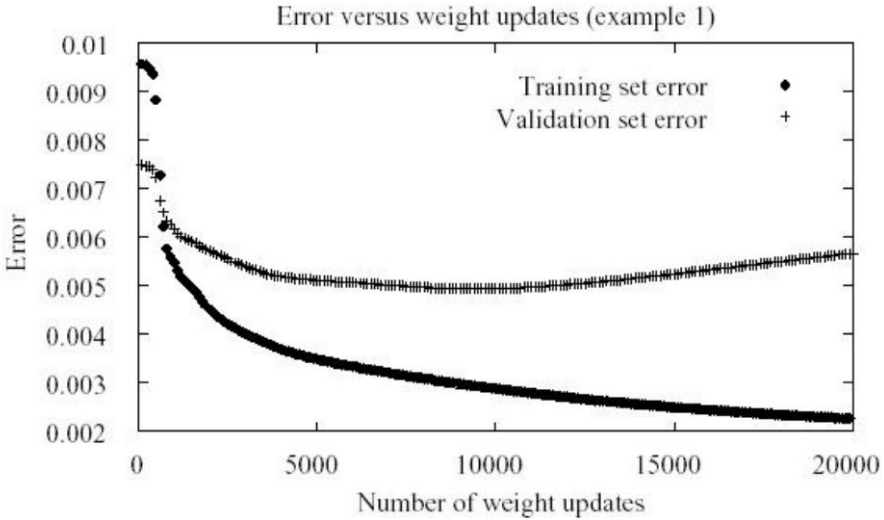
Momento

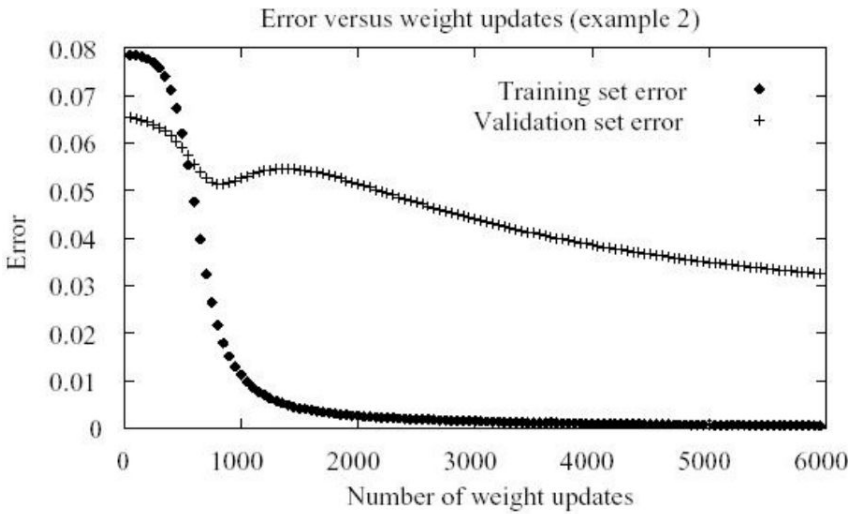
- Alteração da função de atualização de pesos
  - depende do gradiente e da atualização de pesos da iteração anterior
- Efeitos
  - Permite continuar a actualização na presença de
    - mínimos locais da superfície de erro
    - regiões planas
  - Permite acelerar a convergência
    - porque incrementa gradualmente o passo de pesquisa em regiões onde o gradiente não muda

Critérios de terminação

- Número fixo de iterações
  - poucas: pode não reduzir o erro de forma suficiente
  - muitas: pode provocar sobre-ajustamento
- Erro treino abaixo de certo valor
  - pode provocar sobre-ajustamento
- Aumento do erro no conjunto de validação
  - para prevenir sobre-ajustamento

Sobre-ajustamento





Unidades e Camadas escondidas

Unidades escondidas

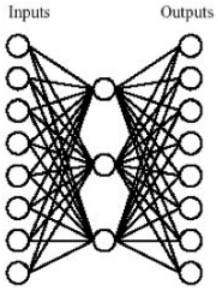
- Unidades situadas nas camadas escondidas (entre a camada de entrada e a de saída)
- Permitem
  - a aprendizagem de funções não lineares
  - representar combinações dos atributos de entrada
- Número
  - demais: a rede memoriza os padrões de entrada
  - de menos: a rede pode não conseguir representar todas as generalizações possíveis

Camadas escondidas

- Os exemplos apenas restringem as entradas e saídas da rede
  - a representação das unidades escondidas é aquela que minimiza o erro
- Consequência
  - o algoritmo pode originar características na(s) camada(s) escondida(s) não explícitas na representação de entrada mas que capturam propriedades relevantes

Exemplo: função identidade

- Rede
  - camada entrada: 8 unidades
  - camada saída: 8 unidades
  - camada escondida: 3 unidades
- Resultado após 5000 iterações
  - codificação das 8 entradas
    - 10000000: 100
    - 01000000: 001
    - 00100000: 010
    - 00010000: 111
    - 00001000: 000
    - 00000100: 011
    - 00000010: 101
    - 00000001: 110



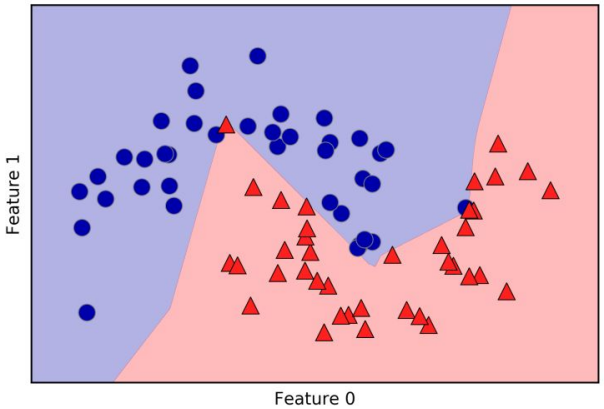
Input		Output
10000000	→	10000000
01000000	→	01000000
00100000	→	00100000
00010000	→	00010000
00001000	→	00001000
00000100	→	00000100
00000010	→	00000010
00000001	→	00000001

Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.01	.11	.88	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.22	.99	.99	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

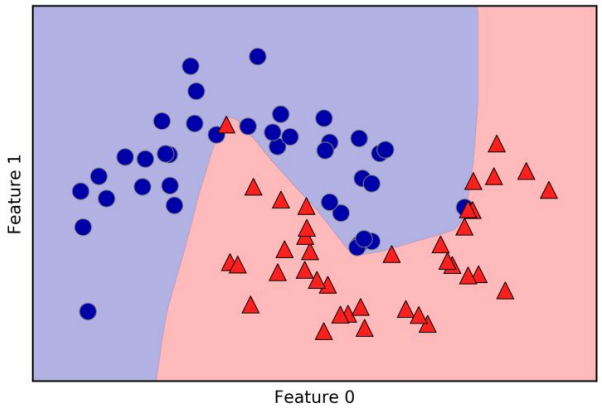
Superfícies de decisão

Variação com as unidades escondidas

- 10 unidades escondidas
  - camadas escondidas: 1
  - função ativação: reLU
    - fronteira composta por 10 segmentos de reta

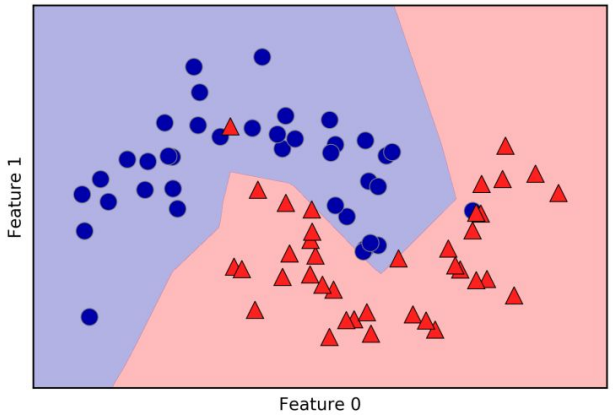


- 100 unidades escondidas
  - camadas escondidas: 1
  - função ativação: reLU
- fronteira mais suave

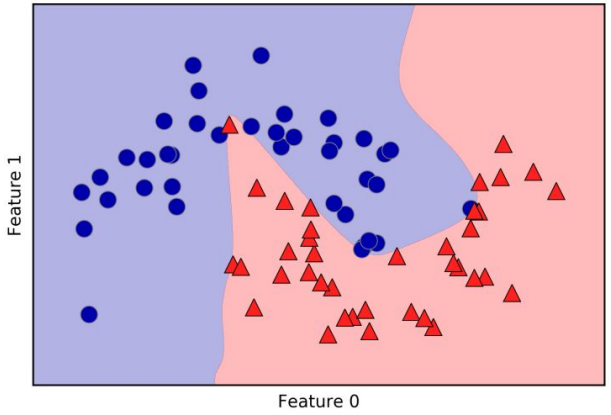


Variação com as camadas escondidas

- 1 camada escondida
  - unidades escondidas: 10
  - função ativação: reLU

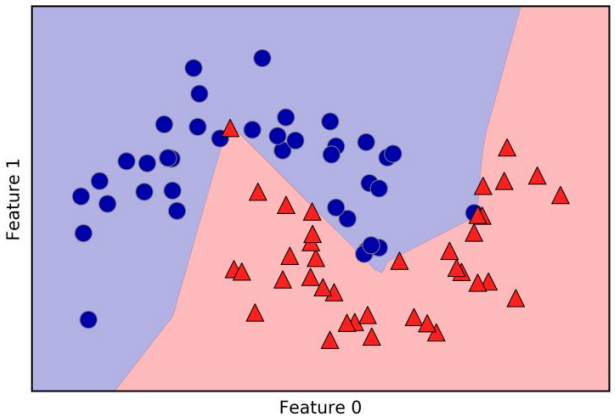


- 2 camadas escondidas
  - unidades escondidas: 10 (cada camada)
  - função ativação: reLU
- fronteira mais suave

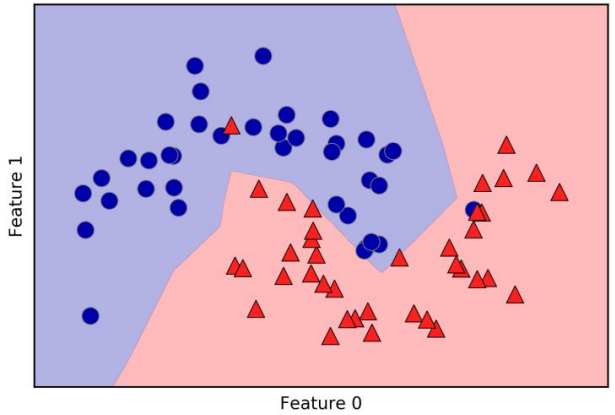


Variação com a função de ativação

- função ativação: reLU
  - camadas escondidas: 2
  - unidades escondidas: 10 (cada camada)

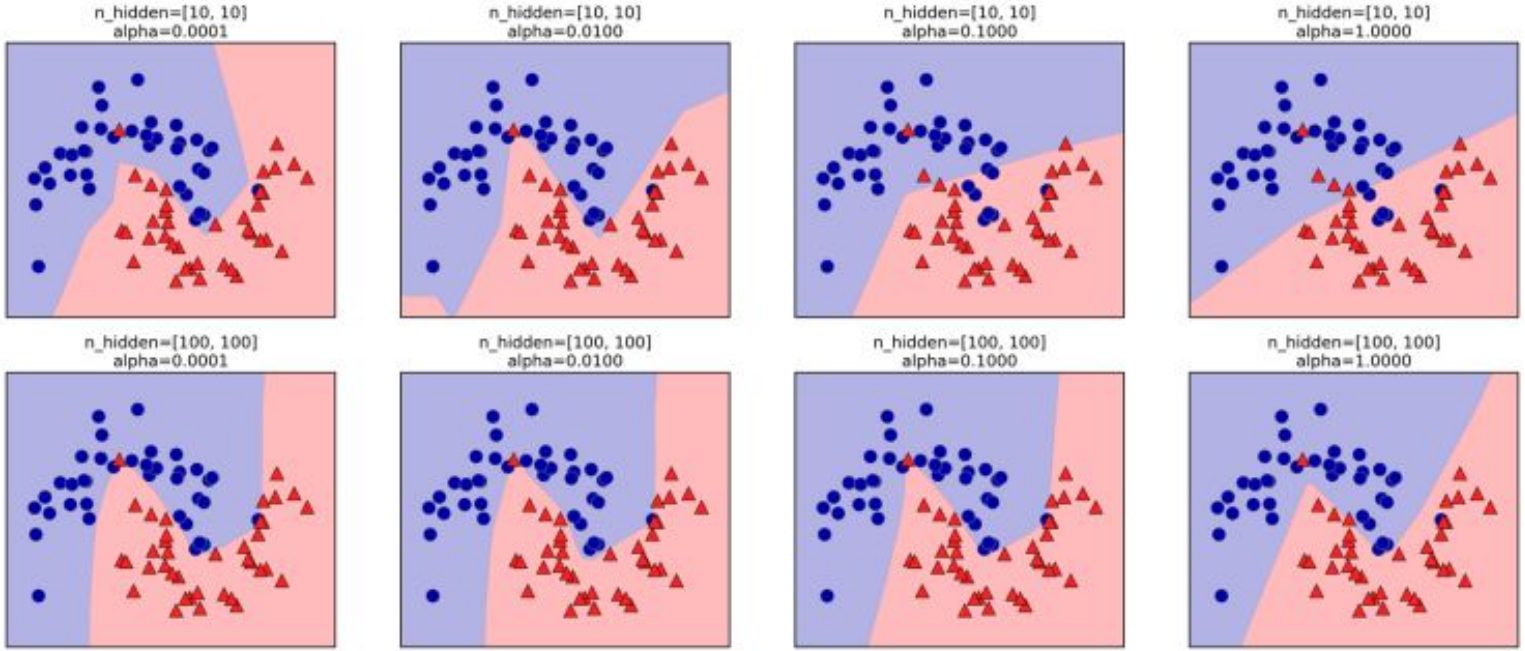


- função ativação: tanh
  - camadas escondidas: 2
  - unidades escondidas: 10 (cada camada)
  - fronteira mais suave



Variação com a regularização

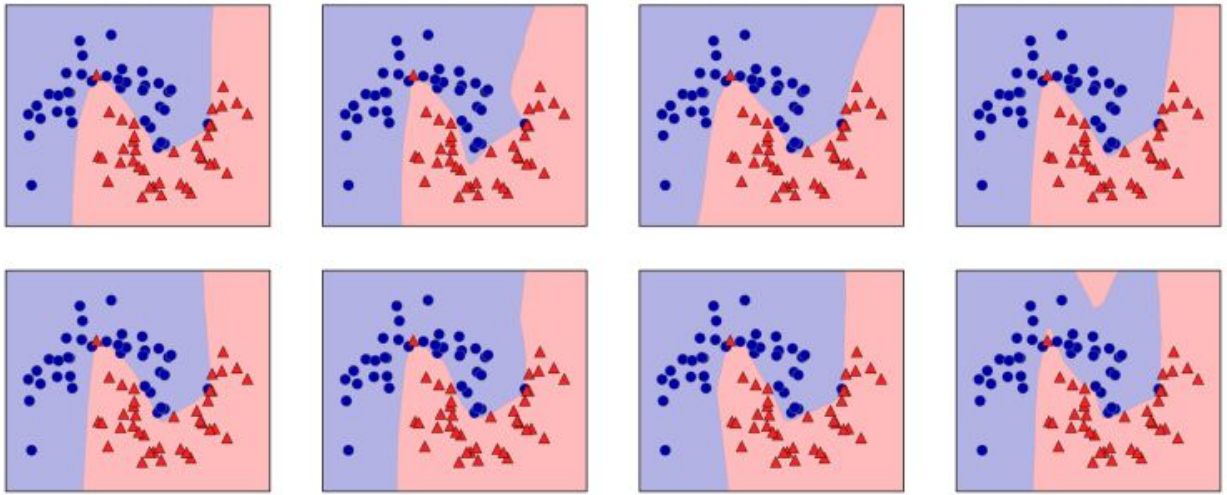
- parâmetro alfa
  - menor alfa, menor regularização



Variação com a inicialização dos pesos

- Pesos inicializados aleatoriamente
- Inicialização afeta o modelo aprendido
  - mais importante em redes pequenas

- Parâmetros
  - função ativação: relu
  - camadas escondidas: 2
  - unidades escondidas: 100 (cada camada)
  - alfa: 0.0001



Exemplo - Breast Cancer

Experiências: desempenho

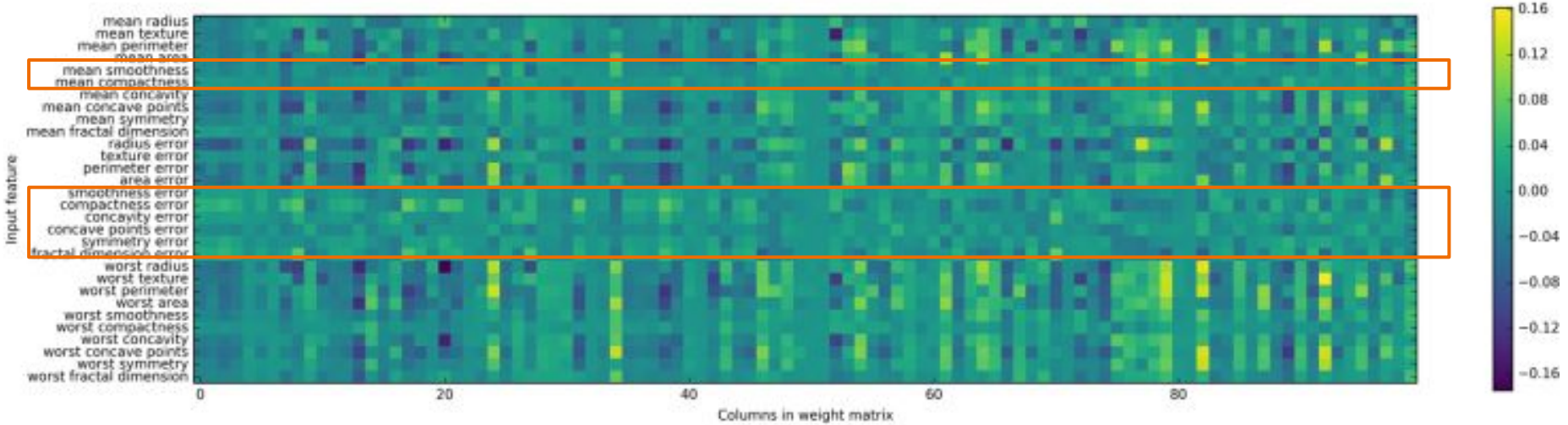
- Exp1: Parâmetros por omissão
  - alfa: 0.0001, função ativação: relu, camadas escondidas: 1, unidades escondidas: 100
  - Exatidão
    - treino: 0.92
    - **teste: 0.90**
- Exp2: Atributos normalizados
  - média=0, desvio=1
  - Exatidão
    - treino: 0.991
    - teste: 0.965
  - (atingido máximo iterações)
- Exp3: Aumento nº iterações
  - Exatidão
    - treino: 0.995
    - teste: 0.965
- Exp4: Aumento da regularização
  - alfa: 1
  - Exatidão
    - treino: 0.988
    - **teste: 0.972**

Modelo aprendido

- Difícil análise/compreensão
- Possibilidade
  - visualização dos pesos aprendidos
- Possível inferência
  - atributos com pesos muito pequenos para todas as unidades são "menos importantes"

Exemplo

- pesos aprendidos à entrada da camada escondida
  - linhas: atributos (30), colunas: unidades escondidas (100)
  - cor clara: valor positivo alto, cor escura: valor negativo



Aprendizagem de coeficientes

Algoritmos

- sgd
  - stochastic gradient descent
  - tem muitos parâmetros para serem afinados
- adam
  - otimizador baseado no gradiente estocástico
  - funciona bem na maioria das situações mas é muito sensível à escala dos dados
- l-bfgs
  - algoritmo da família dos métodos quasi-Newton
  - mais robusto mas mais demorado que o "adam" em grandes redes e muitos dados

Vantagens, desvantagens e parâmetros

Vantagens e desvantagens

- Vantagens
  - capacidade de capturar informação contida em quantidades enormes de dados e construir modelos complexos incríveis
- Desvantagens
  - grande tempo de computação
  - pre-processamento cuidado dos dados
  - afinação dos parâmetros

Camadas e Unidades

- Número de camadas
  - começar por treinar um modelo com uma ou duas camadas e, possivelmente, expandir para mais
- Número de unidades escondidas por camada
  - normalmente semelhante ao número de entradas mas raramente superior a poucos milhares

Ajuste dos parâmetros

1. Criar uma rede grande o suficiente para sobre-ajustar
    - garantindo que a tarefa pode ser aprendida pela rede
2. Alterar a rede
    - reduzir o tamanho da rede
    - aumentar alfa para aumentar a regularização

Complexidade do modelo

- O número de coeficientes a estimar é uma medida indicadora
- Exemplo (classificador binário)
  - 100 atributos, 100 unidades escondidas, 1 camada escondida
    - cf: 10100 (100*100+100)
  - mais uma camada escondida
    - cf: 20100 (100*100+100*100+100)
  - 100 atributos, 1000 unidades escondidas, 1 camada escondida
    - cf: 101000 (100*1000+1000)
  - mais uma camada escondida
    - cf: 1101000 (100*1000+1000*1000+1000)

Poder de representação

- Função booleana
  - pode ser representada por uma rede com uma camada escondida
- Função contínua limitada
  - pode ser representada com um erro arbitrariamente pequeno por uma rede com uma camada escondida
- Função contínua
  - pode ser representada com uma correção arbitrária por uma rede com duas camadas escondidas

Mais informação

Mais informação

- Tipos de redes neuronais
  - https://www.mygreatlearning.com/blog/types-of-neural-networks/
  - https://analyticsindiamag.com/6-types-of-artificial-neural-networks-currently-being-used-in-todays-technology/
- Feedforward e Backpropagation
  - https://mlfromscratch.com/neural-networks-explained/
- Funções de ativação
  - https://mlfromscratch.com/activation-functions-explained/
- Otimização (aprendizagem dos coeficientes)
  - https://mlfromscratch.com/optimizers-explained/