Aprendizagem Bayesiana e baseada em protótipos

V 3.1, V.Lobo, EN/ISEGI, 2007

Classificação Bayesiana

Victor Lobo

Contexto

- Existem um conjunto de dados conhecidos
  - Conjunto de treino
- Queremos prever o que vai ocorrer noutros casos
- Exemplo
  - Empresa de seguros de saúde quer estimar custos com um novo cliente

Conjunto de treino (dados históricos)

Altura	Peso	Sexo	Idade	Ordenado	Usa ginásio	Encargos para seguradora
1.60	79	M	41	3000	S	N
1.72	82	M	32	4000	S	N
1.66	65	F	28	2500	N	N
1.82	87	M	35	2000	N	S
1.71	66	F	42	3500	N	S

E o Manel ?

Altura=1.73
Peso=85
Idade=31
Ordenado=2800
Ginásio=N

Terá encargos para a seguradora ?

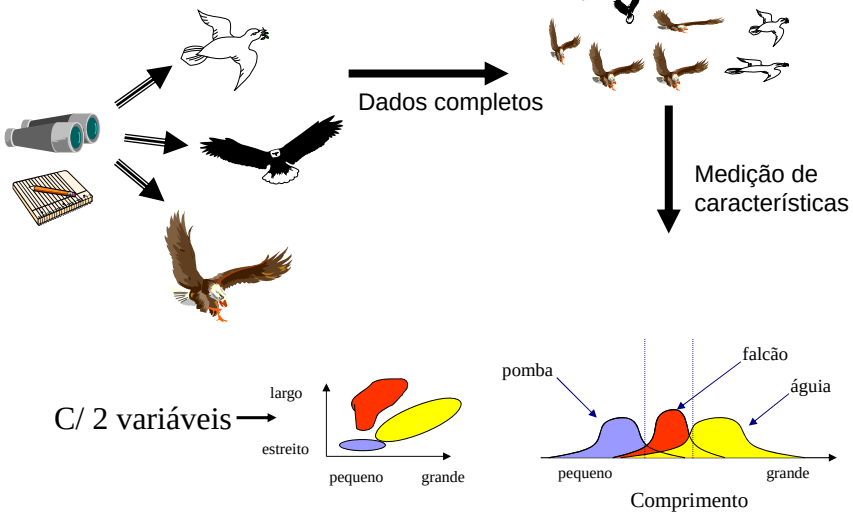
Aprendizagem Bayesiana e baseada em protótipos

V 3.1, V.Lobo, EN/ISEGI, 2007

Tema central:

- Existe alguma maneira ÓPTIMA de fazer a classificação de um padrão de dados ?
  - Sim: classificação Bayesiana (óptima segundo um dado critério...)
- Conseguimos usar sempre esse método ?
  - Não: geralmente é impossível obter o classificador de Bayes
- É útil conhecê-lo ?
  - Sim: Dá um limite e um termo de comparação

O nosso exemplo...



Noção de Classificação Bayesiana

- Escolhe a classe mais provável, dado um padrão de dados
  - max $P(C_i|\boldsymbol{x})$
- É sempre a escolha óptima !
- Problema:
  - Estimar $P(C_i|\boldsymbol{x})$
  - Solução: dado um dado, eu posso não saber à priori a classe, mas dado uma classe, eu talvez saiba à priori como são dos dados dessa classe...

Teorema de Bayes

- Formulação do teorema de Bayes
  - $P(C,\boldsymbol{x}) = P(C|\boldsymbol{x})P(x) = P(\boldsymbol{x}|C)P(C)$

    logo.. $P(C|\boldsymbol{x}) = P(\boldsymbol{x}|C)P(C) / P(\boldsymbol{x})$

  - Dado um $\boldsymbol{x}$, $P(\boldsymbol{x})$ é constante, o classificador Bayesiano escolhe a classe que maximiza $P(\boldsymbol{x}|C)P(C)$
- Classificador que maximiza $P(C|\boldsymbol{x})$ é conhecido como classificador MAP (*maximum a posterioi*)

Custos variáveis

- A escolha óptima da classe tem que ter em conta os custos de cometer erros
  - Exemplos: detectar aviões num radar, detectar fraudes ou defeitos em peças
- Custo: $ct(c_i,c_j)$ = custo de escolher c_j dado que a classe é de facto c_j
- Matriz de custos
  - Matriz com todos os custos de classificação
- Determinação dos custos
  - ...

Classificador de Bayes

- Custo de uma decisão:
  - $ct_j(\boldsymbol{x}) = \Sigma\ ct(c_i,c_j)\ P(c_i,\boldsymbol{x})$
    - Custo de escolher A é a soma dos custos de escolher as OUTRAS classes vezes a probabilidade de ocorrerem as OUTRAS classes
- Classificador de Bayes
  - Escolhe a classe que minimiza o custo de classificação
  - $c=c_k : k= \arg\min ct_j(\boldsymbol{x})$

Classificador de máxima verosimilhança

- Maximum Likelihood (ML)
  - Muitas vezes podemos admitir que, à partida, todas as classes são equiprováveis
  - Nesse caso, o classificador MAP simplifica para:

$$P(C|\boldsymbol{x}) = P(\boldsymbol{x}|C)P(C) / P(\boldsymbol{x}) \propto P(\boldsymbol{x}|C)$$

  - Ou seja a classe mais provável é a que com maior probabilidade gera esse dado!
  - Na prática, um bom critério !

Problemas em estimar P($\boldsymbol{x}$,C)

- Desconhece-se geralmente a forma analítica de P($\boldsymbol{x}$,C)
- Estimação de P($\boldsymbol{x}$,C) a partir dos dados
  - **Problema central em classificação !!!**
  - Estimação paramétrica
    - Assumir que P($\boldsymbol{x}$,C) tem uma distribuição "conhecida" (gausseana, uniforme, etc), e estimar os parâmetros dessa distribuição
  - Estimação não paramétrica
    - Calcular P($\boldsymbol{x}$,C) directamente a partir dos dados

Exemplo de classificação Bayesiana : Jogar ténis ou não ?

Outlook	Temperature	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Overcast	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

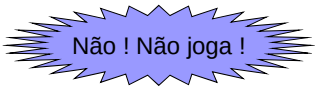
Caso 1: sabendo só o “outlook”

- Queremos saber P(jogo|outlook), em concreto, se outlook = “sunny”
  - Classificador MAP:
    - P(jogo|outlook) $\propto$ P(outlook|jogo)P(jogo)
    - P(jogo=sim)=9/14=0.64 P(jogo=não)=5/14=0.36
    - P(outlook=“sunny”|jogo=sim)=2/9=0.22
    - P(outlook=“sunny” |jogo=não)=3/5=0.60
    - P(jogo=sim|outlook=“sunny”) $\propto$ 0.22 x 0.64 = 0.14
    - P(jogo=não|outlook=“sunny”) $\propto$ 0.60 x 0.36 = 0.22

NÃO ! Não joga !

Caso 1: sabendo só o "outlook"

- Classificador ML:
  - P(jogo|outlook) $\propto$ P(outlook|jogo)
  - P(outlook="sunny"|jogo=sim)=2/9=0.22
  - P(outlook="sunny" |jogo=não)=3/5=0.60



Problema quando $\boldsymbol{x}$ tem dimensão grande

- Se a dimensão de $\boldsymbol{x}$ é muito grande, devido à praga da dimensionalidade, é difícil calcular P($\boldsymbol{x}$,C)
- Solução:
  - Assumir independência entre atributos
  - Exemplo:
    - Classificação de texto

Classificador *naive* de Bayes

- Assume independência dos atributos:

$$P(\boldsymbol{x},C) = \Pi\ P(\boldsymbol{x}^m,C)$$

- Na prática tem bons resultados
  - Evitar que $P(\boldsymbol{x}^m,C)$ seja 0:
    - Estimativa m:
      - $P=(n_c + m \times p) / (n + m)$

n_c= exemplos de c — n= total de exemplos
m= ponderação (+/-prioi) — p= estimativa à priori (equiprovável ?)

Algumas considerações...

- Aprendizagem incremental
  - Um classificador Bayesiano pode ir actualizando as suas estimativas
- Separababilide
  - $P(\boldsymbol{x},c_i)>0 \Rightarrow P(\boldsymbol{x},c_j)=0\ \forall \boldsymbol{x}\ \forall \boldsymbol{j \neq i}$
  - Erro de Bayes = 0
- Não separabilidade
  - Inconsistência (com os atributos conhecidos):
    - Um mesmo $\boldsymbol{x}$, tanto pode pertencer a c_i como c_j
  - Erro de Bayes > 0

Classificadores bayesianos:

- Classificador de Bayes, com custos
  - Entra em linha de conta com custos
- MAP
  - Assume custos iguais
- ML
  - Assume classes equiprováveis
- Naive de Bayes
  - Assume independência entre atributos
- Erro de Bayes
  - Erro do classificador bayesiano (geralmente MAP)

Aprendizagem baseada em instâncias

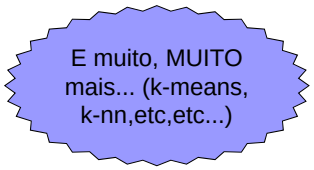
Victor Lobo

Tema central

- Sistemas de aprendizagem que guardam "exemplos" dos dados
  - Ex: Guardar a "pomba típica" ou "som característico"
- A classificação (ou decisão) é feita comparando a nova instância com os exemplos guardados
  - Exemplos ≈ protótipos ≈ instâncias ≈ neurónios

Muitos nomes para a "mesma coisa"

- Estatística
  - Kernel-based density estimation (Duda & Hart 68)
  - Locally-weighted regression (Hardle 90)
- Machine Learning
  - Memory-based classification (Stanfill & Waltz 86)
  - Exemplar-based knowlegde acquisition (Bareiss 89)
  - Instance-based classification (Aha 91)
  - Case-based reasoning (Shank 82)
  - Lazy Learning (Alpaydin 97)
- Redes Neuronais
  - Prototype-based networks (Kohonen 95)
  - RBF (Lowe 88), LVQ, etc, etc....



Fundamentos:

- Classificador óptimo escolhe classe mais provável:
  - P(C|***x***) = P(***x***|C)P(C) / P(***x***)
  - No caso de um classificador MAP, basta saber P(***x***|C)
- Estimação de P(***x***|C) quando os atributos de x têm valores contínuos:
  - P(***x***|C)=0, mas podemos calcular p(***x***|C) (densidade)
  - No limite temos $p(x \mid C) = \frac{k/n}{\Delta V}$

k=nº de dados da classe (emΔV)
n=nª total de dados (emΔV)
ΔV=Volume considerado

Fundamentos

- Para que $p(x \mid C) = \frac{kc/k}{\Delta V}$

ΔV = um dado volume em torno da nova instância
k= nº total de exemplos nesse volume
kc=nº de exemplos que pertencem à classe C

…é necessário que k $\to \infty$, e $\lim_{n\to\infty} \Delta V = 0$

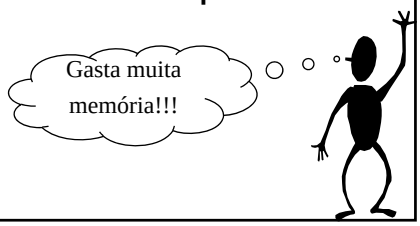
(em princípio teremos também $\lim_{n\to\infty} kc = \infty$)

  - Mas isso é impossível…
- Duas grandes famílias
  - *k* = cte ***k*-vizinhos**, vizinho mais próximo, etc
  - ΔV =cte **Janelas de Parzen**

k-vizinhos

k-vizinhos e vizinho mais próximo (*k*=1)

- Todos os exemplos são memorizados e usados na fase de aprendizagem.
- A classificação de um exemplo *X* consiste em encontrar os *k* elementos do conjunto de treino mais próximos e decidir por um critério de maioria.



Algoritmo k - vizinhos mais próximos

- Algoritmo de treino

  Para cada exemplo de treino (x, c(x)) adicionar à lista de exemplos de treino.

  Retorna lista de exemplos de treino.

Classificação por k-vizinhos

- *k*-NearestNeighbor(*x*, Exemplos de treino)

  Sejam $y_1, \ldots, y_k$, pertencentes à lista de exemplos de treino, os k vizinhos mais próximos de *x*.

  Retorna $\hat{c}(x) \leftarrow \arg\max_{v \in V} \sum_{i=1}^{k} \delta(v, c(y_i))$

  em que V é o conjunto das classes, v é uma classe em particular, $c(y)$ é a classe de y, e

$$\delta(x, y) = \begin{cases} 0 & se\ x \neq y \\ 1 & se\ x = y \end{cases}$$

Regressão por k-vizinhos

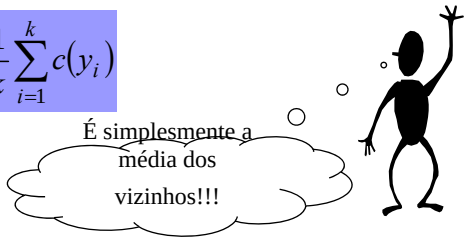
- Algoritmo de regressão

  k-NearestNeighbor(*x*, exemplos de treino)

  Sejam $y_1, \ldots, y_k$, pertencentes à lista de exemplos de treino, os *k* vizinhos mais próximos de *x*.

  Retorna

$$\hat{c}(x) \leftarrow \frac{1}{k}\sum_{i=1}^{k} c(y_i)$$

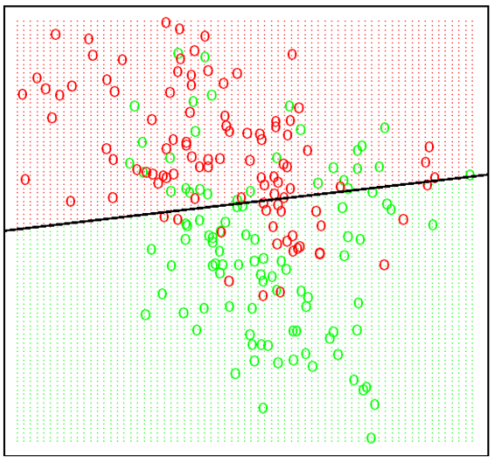


Fronteiras definidas pelo k-nn

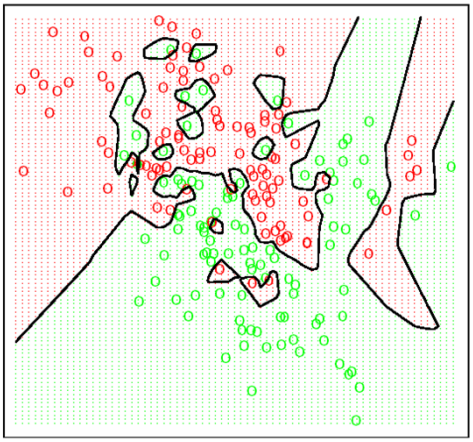
- k grande
  - Fronteiras suaves, "ponderadas"
  - Estimador razoável da densidade de probabilidade
  - Perde definição quando há variações pequenas
- k pequeno
  - Fronteiras mais rugosas, sensíveis a outliers
  - Mau estimador de densidade de probabilidade
- Margens de segurança
  - Pode-se exigir uma diferença mínima para tomar uma decisão

Regressão linear (quando $k \rightarrow n$)

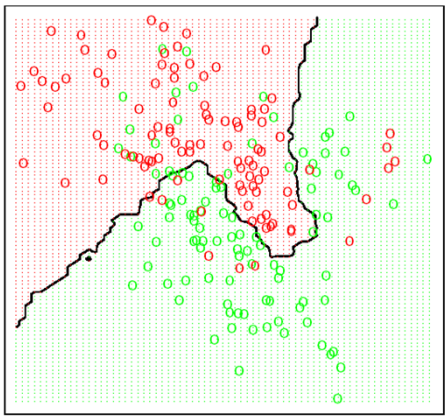
- Fronteiras do discriminante linear de Fisher



1-vizinho mais próximo



15–vizinhos mais próximos



Exemplos de medidas de semelhança

- Distâncias
  - Euclidiana
  - Hamming
  - Minkowski

$$D_M(X,Y,\lambda)=\left(\sum_{i=1}^{K}|x_i-y_i|^{\lambda}\right)^{1/\lambda}$$

  - Mahalanobis

$$D_{Ma}(X,Y,\varphi)=\left((X-Y)^T\Psi_{[KK]}(X-Y)\right)^{1/2}$$

- Correlação
  - Não normalizada

$$C(X,Y)=X.Y=\sum_{i=1}^{K}x_iy_i$$

  - Máxima correlação

$$C_m(X,Y)=\max_j\sum_{i=1}^{K}x_iy_{i-j}$$

Classificação por *k*-vizinhos ponderados

- Algoritmo de classificação

  k-NearestNeighbor(*x*, Exemplos de treino)

  Sejam $y_1, \ldots, y_k$, pertencentes à lista de exemplos de treino, os *k* vizinhos mais próximos de *x*.

  Retorna

$$\hat{c}(x) \leftarrow \arg\max_{v \in V} \sum_{i=1}^{k} \varpi_i \delta(v, c(y_i))$$

  em que

$$\varpi_i = \frac{1}{D(x,y)}$$

Regressão pelos *k*-vizinhos ponderados

- Algoritmo de classificação

  k-NearestNeighbor(*x*, Exemplos de treino)

  Sejam $y_1, \ldots, y_k$, pertencentes à lista de exemplos de treino, os k vizinhos mais próximos de *x*.

  Retorna

$$\hat{c}(x) \leftarrow \frac{\sum_{i=1}^{k} \varpi_i c(y_i)}{\sum_{i=1}^{k} \varpi_i}$$

Vizinho mais próximo (*k*=1)

- É simples e eficaz
- Está muito bem estudado
- Erro assimptótico (quando n $\rightarrow \infty$)
  - Zero, se as classes forem separáveis
  - 2x erro de Bayes, se não o forem
    - (Cover 67; Ripley 96; Krishna 00)

Erro do vizinho mais próximo

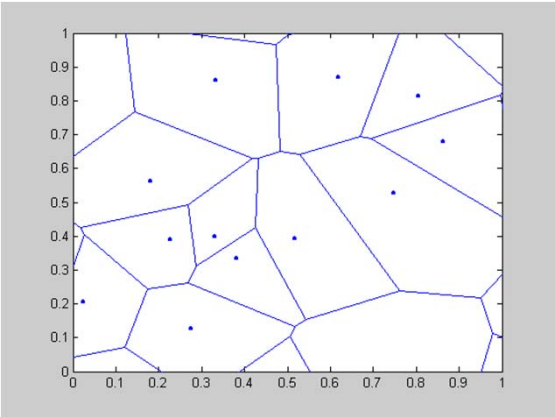
- Com *n* finito,e *c* classes

$$E_{bayes} \leq E_{nneighbour} \leq 2E_{bayes} - \frac{c}{c-1} E_{bayes}^{\ 2} + \sup_{x \in X} \partial_{mx}(x)(1 - \frac{cE_{bayes}}{c-1})$$

- $\delta(\boldsymbol{x})$ é a função de semelhança (Drakopoulos 95), que pode ser estimada, e tem geralmente um valor baixo

Fronteiras do vizinho mais próximo

- Partição de Voronoi do conjunto de treino



Problemas com *k*-nn

- Exigem MUITA memória para guardar o conjunto de treino
- Exigem MUITO tempo na fase de classificação
- São muito sensíveis a outliers
- São muito sensíveis à função de distância escolhida
  - Só de pode resolver com conhecimento à priori...

Variantes sobre k-vizinhos

Edited Nearest Neighbors

- Remover os outliers, e os exemplos demasiado próximos da fronteira
- Usar a regra de classificação (k-nn) sobre o próprio conjunto de treino, e eliminar os exemplos mal classificados
  - k=3 já produz bons resultados

Minimização do nº de protótipos

- Reduzir o nº de protótipos resolve os 2 primeiros problemas !
- Deixa de ser possível estimar p(x)
- Enquadramento formal
  - Q-Sets
- Heurísticas
  - Condensed Nearest Neighbors (= IB2, RIBL, etc)

Condensed Nearest Neighbors

[Hart 68]

```
1     Let
2
3           Train     Training Set
4           #train    Number of patterns in the training set
5           CNN       Condensed Nearest Neighbor set
6
7     Do
8
9           CNN = { Train 1}
10          Repeat
11                Additions =FALSE
12                For i =2 to  #train
13                      Classify  Train i with CNN
14                      If Train i is incorrectly classified
15                            CNN = CNN  ∩ {Train i}
16                            Additions =TRUE
17                      End_if
18                End_for
19          Until Additions  = FLASE
```

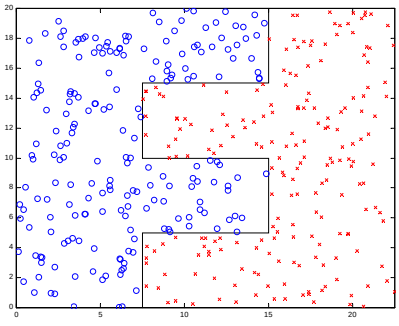
Reduced Nearest Neighbors

[Gates 72]

```
1       Let
2
3               Train       Training Set
4               #train      Number of patterns in the training set
5               #cnn        Number of patterns in the CNN set
6               CNN         Condensed Nearest Neighbor set
7               RNN         Reduced Nearest Neighbor Set
8
9       Do
10
11              RNN = CNN
12              For i =1 to #cnn
13                      Let Candidate_RNN = RNN – { RNN_i}
14                      Classify all Train with Candidate_RNN
15                      If all patterns in Train are correctly classified
16                              RNN = Candidate_RNN
17                      End_if
18              End_for
```

Toy problem para testes

- Double F ou Harts' Problem
  - Simples visualisação, fronteira "complexa"
  - Distribuição uniforme nas áreas indicadas
  - Usada por muitos autores como ref.



Harts'problem com 400 padrões

Avaliação experimental dos métodos

1 - Gerar *N* pontos para conjunto de treino

2 - Aplicar o método para obter um classificador

3 - Gerar *M* pontos para conjunto de validação

4 - Calcular o erro *E* no conjunto de validação

5 - Repetir os passos 1-4 várias vezes, e calcular os valores médios e desvios padrões para: Erro, Nº de protótipos, Tempo de treino e classificação

Cálculo do erro

- Qual o tamanho do conjunto de validação para estimar o erro ?

Para cada padrão de teste $x = \begin{cases} 1(erro) & p \\ 0(certo) & 1-p \end{cases}$

C/ $p \approx 1\%$ e N=10e6
$\sigma = 0.01\% \approx 0$

Erro médio $y = \frac{1}{N}\sum_{i=1}^{N} x_i$

$E(y) = \overline{E}(x_i) = \hat{p} \quad \hat{\sigma}_y^2 = \frac{\hat{p}(1-\hat{p})}{N}$ (desde que $N \times p \times (1-p) > 5$)

Rotinas Matlab (do Toolbox dos "Magos")

- Class_plot(x,y,class)
- [vx,vy]=Voronoi_boundary(x,y,class)
- [c,cp] = knn(t_data, t_label, x, k)
- [c] = knn_mat(t_data, t_label, x)
- [cnn,cnn_label]=Cnn(train, train_label)
- [rnn,rnn_label]=Rnn(train,train_label,cnn,cnn_label)
- outclass=SelfClassify(dataset,inclass)
- [data]=Remove_col(data,index)

Fronteiras típicas