Aprendizado baseado em instâncias

Prof. Marcelo K. Albertini
Faculdade de Computação
Universidade Federal de Uberlândia

29 de Maio de 2017

Aprendizado baseado em Instâncias (ABI)

- Instance-based learning (IBL)
- K vizinhos mais proximos
- Outras formas de ABI
- Filtragem colaborativa

Aprendizado baseado em Instâncias

Ideia-chave

- Armazenar todos exemplos de treino $\langle x_i, f(x_i) \rangle$

Classificação usando o Vizinho mais próximo

- Dada instância de consulta x_q, encontrar o exemplo de treino mais próximo x_n e estimar $\hat{f}(x_q) \leftarrow f(x_n)$

k-vizinhos mais próximos

- Dado x_q, *votar* entre os k vizinhos mais próximos (se f for discreta)
- Obter a média de f dos k vizinhos mais próximos (se f for contínuo)

$$\hat{f}(x_q) \leftarrow \frac{1}{k} \sum_{i=1}^{k} f(x_i)$$

Vantagens e desvantagens

Vantagens

- Treino é muito rápido
- Aprendizado fácil de funções complexas
- Não perde informações

Desvantagens

- Lento em momento de consulta
- Uso excessivo de memória
- Problemático para atributos irrelevantes

Medidas de distância

Em geral, a escolha é arbitrária e deve codificar conhecimento.

Atributos numéricos

- Euclidiano, Manhattan, norma L^n

$$L^n(\vec{x}_1, \vec{x}_2) = \sqrt[n]{\sum_{i=1}^{num.atr.} \|\vec{x}_{1,i} - \vec{x}_{2_i}\|^n}$$

- Normalizado por: intervalo, desvio padrão

Medidas de distância

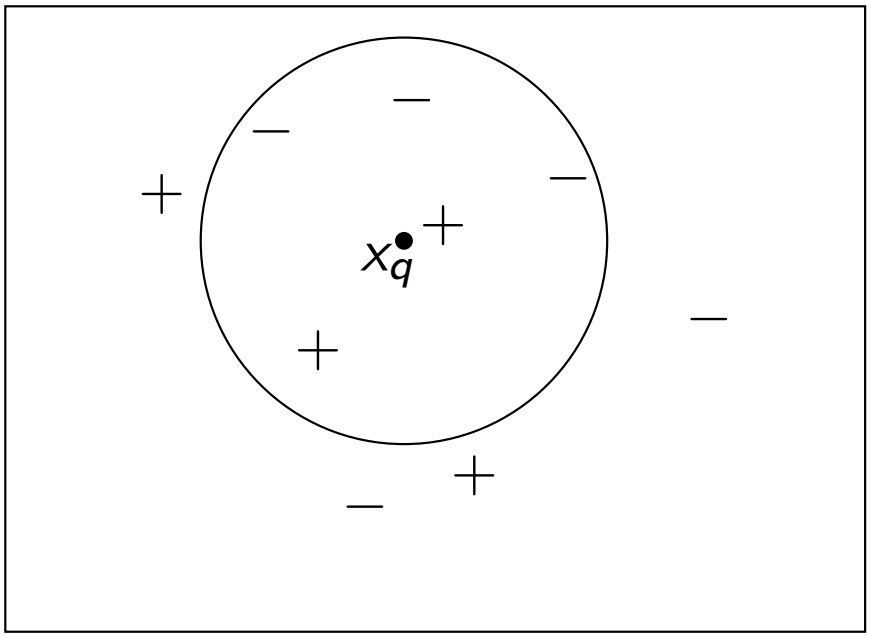
Atributos simbólicos/categóricos

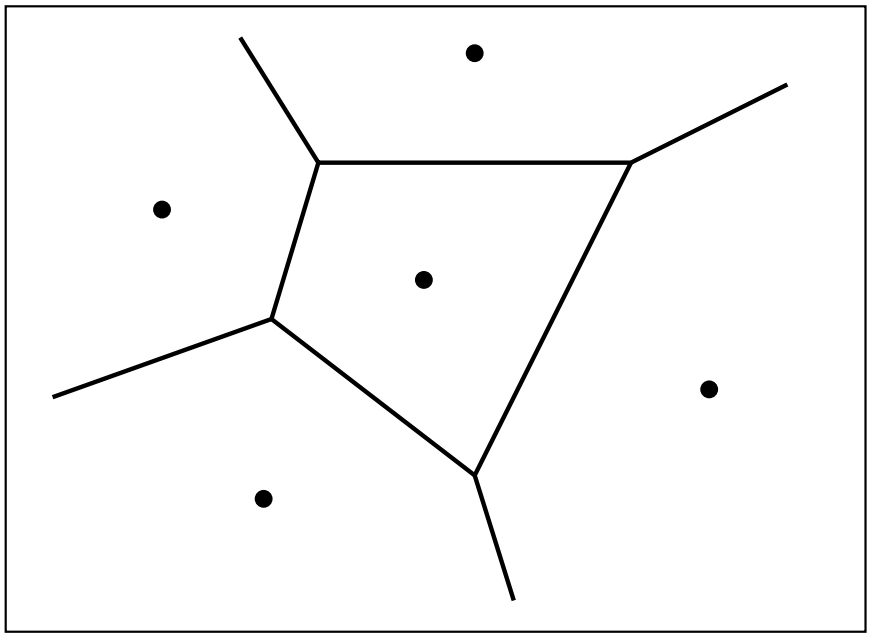
- Distância de Hamming (codificação binária)
- Medida de diferença de valores

$$\delta(val_i, val_j) = \sum_{h=1}^{num.classes} |P(c_h|val_i) - P(c_h|val_j)|^n$$

- onde $n \geq 1$ define o tipo da norma

Diagrama de Voronoi





Célula de Voronoi de $\vec{x} \in S$

Todos os pontos mais perto a $\vec{x}$ que qualquer outra instância em S, onde S é o conjunto de exemplo de treino.

Região da classe C

União das células de Voronoi de C em S

Comportamento no limite

$\epsilon^*(x)$: Erro de predição ótima. Relacionado ao erro mínimo e classe minoritária.
$\epsilon_{NN}(x)$: Erro do vizinho mais próximo.

Teorema para 1-NN

$\lim_{n\to\infty} \epsilon_{NN} \leq 2\epsilon^*$

Comportamento no limite

$\epsilon^*(x)$: Erro de predição ótima. Relacionado ao erro mínimo e classe minoritária.
$\epsilon_{NN}(x)$: Erro do vizinho mais próximo.

Teorema para 1-NN

$\lim_{n\to\infty} \epsilon_{NN} \leq 2\epsilon^*$

Ideia da prova (caso para 2 classes)

$\epsilon_{NN} = p_+ p_{NN\in-} + p_- p_{NN\in+}$

Comportamento no limite

$\epsilon^*(x)$: Erro de predição ótima. Relacionado ao erro mínimo e classe minoritária.
$\epsilon_{NN}(x)$: Erro do vizinho mais próximo.

Teorema para 1-NN

$\lim_{n\to\infty} \epsilon_{NN} \leq 2\epsilon^*$

Ideia da prova (caso para 2 classes)

$\epsilon_{NN} = p_+ p_{NN\in-} + p_- p_{NN\in+}$
$\epsilon_{NN} = p_+(1 - p_{NN\in+}) + (1 - p_+)p_{NN\in+}$

Comportamento no limite

$\epsilon^*(x)$: Erro de predição ótima. Relacionado ao erro mínimo e classe minoritária.
$\epsilon_{NN}(x)$: Erro do vizinho mais próximo.

Teorema para 1-NN

$\lim_{n\to\infty} \epsilon_{NN} \leq 2\epsilon^*$

Ideia da prova (caso para 2 classes)

$\epsilon_{NN} = p_+ p_{NN\in-} + p_- p_{NN\in+}$
$\epsilon_{NN} = p_+(1 - p_{NN\in+}) + (1 - p_+)p_{NN\in+}$

Usando: $\lim_{n\to\infty} p_{NN\in+} = p_+$
$\lim_{n\to\infty} p_{\in NN} = p_+(1 - p_+) + (1 - p_+)p_+$

Comportamento no limite

$\epsilon^*(x)$: Erro de predição ótima. Relacionado ao erro mínimo e classe minoritária.
$\epsilon_{NN}(x)$: Erro do vizinho mais próximo.

Teorema para 1-NN

$\lim_{n\to\infty} \epsilon_{NN} \leq 2\epsilon^*$

Ideia da prova (caso para 2 classes)

$\epsilon_{NN} = p_+ p_{NN\in-} + p_- p_{NN\in+}$
$\epsilon_{NN} = p_+(1 - p_{NN\in+}) + (1 - p_+)p_{NN\in+}$

Usando: $\lim_{n\to\infty} p_{NN\in+} = p_+$
$\lim_{n\to\infty} p_{\in NN} = p_+(1 - p_+) + (1 - p_+)p_+ = 2\epsilon^*(1 - \epsilon^*) \leq 2\epsilon^*$

Comportamento no limite

$\epsilon^*(x)$: Erro de predição ótima. Relacionado ao erro mínimo e classe minoritária.
$\epsilon_{NN}(x)$: Erro do vizinho mais próximo.

Teorema para 1-NN

$\lim_{n\to\infty} \epsilon_{NN} \leq 2\epsilon^*$

Ideia da prova (caso para 2 classes)

$\epsilon_{NN} = p_+ p_{NN\in-} + p_- p_{NN\in+}$
$\epsilon_{NN} = p_+(1 - p_{NN\in+}) + (1 - p_+)p_{NN\in+}$

Usando: $\lim_{n\to\infty} p_{NN\in+} = p_+$
$\lim_{n\to\infty} p_{\in NN} = p_+(1 - p_+) + (1 - p_+)p_+ = 2\epsilon^*(1 - \epsilon^*) \leq 2\epsilon^*$

Teorema para k-NN

$\lim_{n\to\infty,k\to\infty,k/n\to\infty 0} \epsilon_{kNN} = \epsilon^*$

k-NN ponderado com a distância

Vizinhos mais próximos podem ser mais importantes

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i}$$

onde

$$w_i \equiv \frac{1}{d(x_q, x_i)^2}$$

e $d(x_q, x_i)$ é distância entre x_q (teste) e x_i (treino).

Observe que agora faz sentido usar todos os exemplos de treino em vez de somente k

Maldição da dimensionalidade

Exemplo instâncias descritas por 20 atributos mas somente 2 são relevantes para a função-alvo

Maldição da dimensionalidade

- Vizinho mais próximo é facilmente enganado em alta dim.
- Problemas fáceis em baixa dim. são difíceis em alta
- Intuições em baixa dim. não funcionam em alta dim.

Casos

- Distribuição normal
- Pontos em hiper-grides
- Aproximação de esfera por cubo
- Volume da hiper-esfera

Evitando a maldição: seleção de atributos

Abordagem de filtros

Pré-selecionar atributos individualmente

- Exemplo, por ganho de informação

Abordagem "envelope" (*wrapper*)

Selecionar subconjuntos de atributos de acordo com a execução do algoritmo de aprendizado completo

- Seleção direta
- Eliminação retroativa
- Busca exaustiva (tentar todas as combinações)
- Outros...

Seleção Direta

```
1  // CA: Conjunto de Atributos descritores de exemplos
2  Conjunto SelecaoDireta(Conjunto<Atributo> CA) {
3    Conjunto<Atributo> SubCA = new Conjunto();
4    double melhorAval = 0;
5
6    do {
7      Atributo melhorAtr = null;
8      for (Atributo A: CA.subtracao(SubCA))  {
9        Conjunto<Atributo> tmpSubCA= new Conjunto(SubCA);
10       tmpSubCA.inserir(A);
11       if (tmpSubCA.avaliacao() > melhorAval) {
12         melhorAtr = A;
13         melhorAval = tmpSubCA.avaliacao();
14       }
15     }
16     if (melhorAtr != null)
17       SubCA.inserir(melhorAtr);
18   } while (melhorAtr != null && !SubCA.igual(CA));
19
20   return SubCA;
21 }
```

Eliminação retroativa

```
1  // CA: Conjunto de Atributos descritores de exemplos
2  Conjunto EliminacaoRetroativa(Conjunto<Atributo> CA) {
3    Conjunto<Atributo> SubCA = new Conjunto(CA);
4    double melhorAval = SubCA.avaliacao();
5
6    do {
7      Atributo piorAtr = null;
8      for (Atributo A: SubCA)  {
9        Conjunto<Atributo> tmpSubCA = SubCA.subtracao(A);
10       if (tmpSubCA.avaliacao() >= melhorAval) {
11         piorAtr = A;
12         melhorAval = tmpSubCA.avaliacao();
13       }
14     }
15     if (piorAtr != null)
16       SubCA.subtracao(piorAtr);
17   } while (piorAtr != null && SubCA != null);
18
19   return SubCA;
20 }
```

Reduzindo custo computacional

- Recuperação eficiente:
  - árvores k-D: divisor de "hiper-espaços"
  - árvores-R: "hiper-retangulos"
  - árvores-M: "hiper-esferoides"
- Comparação de similaridade eficiente
  - Aproximação rápida para eliminar maior parte
  - Usar medida exata no resto
- Formar protótipos
- k-NN editado
  - Remover instâncias que não afetam fronteira

k-NN editado

```
1 Conjunto knnEditadoPorEliminacao(Conjunto<Instancia> C)
      {
2   for (Instancia i: C) {
3     if (C.subtracao(i).classificao(i) != i.classe())
4       C.inserir(i);
5   }
6   return C;
7 }
```

```
1 Conjunto knnEditadoPorInsercao(Conjunto<Instancia> C) {
2   T = new Conjunto();
3   for (Instancia i: C) {
4     if (T.classifica(i) != i.classe())
5       T.inserir(i);
6   }
7   return T;
8 }
```

Como evitar memorização (*overfitting*)

- Encontrar k por validação cruzada
  - Validação cruzada cria por amostra diferentes conjuntos de validação
  - Avaliar erro médio em diferentes conjuntos de validação
- Formar protótipos
  - Exemplos muito similares podem formar um protótipo
- Remover instâncias com ruídos
  - Exemplo: remover $\vec{x}$ se todos os vizinhos de $\vec{x}$ são de outra classe

Regressão localmente ponderada

k-NN forma aproximação local para f para cada exemplo de consulta x_q

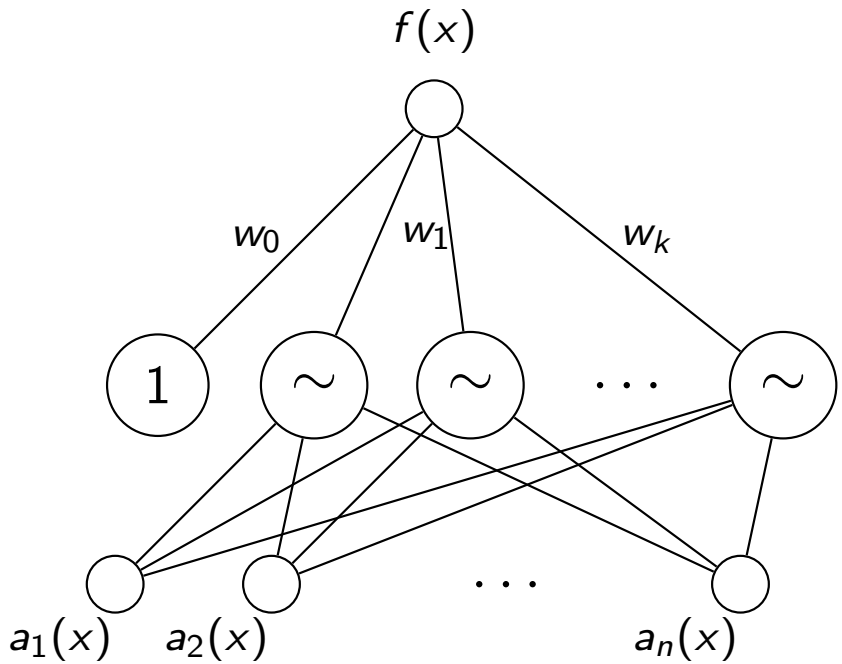
Porque não formar uma aproximação explícita $\hat{f}(x)$ para a região em volta de x_q?

- Obter função linear para os k vizinhos mais próximos
- Obter função quadrática
- Produzir "aproximação por pedaços" para f
- Redes de base radial

Redes de funções de base radial

- Aproximação global para função alvo, em termos de combinação linear de aproximações locais
- Usado, por exemplo, para classificação de imagens
- Associado a redes neurais artificiais

Redes de funções de base radial



onde $a_i(x)$ são os atributos que descrevem a instância x e

$$f(x) = w_0 + \sum_{u=1}^{k} w_u K_u(d(x_u, x))$$

- Escolha comum para K_u:

$$K_u(d(x_u, x)) = \exp^{-\frac{1}{2\sigma_u^2} d^2(x_u, x)}$$

Treinando redes de funções de base radial

Qual x_u para cada função $K_u(d(x_u, x))$?

- Espalhar uniformemente pelo espaço de instâncias
- Usar instâncias de treino (reflete distribuição)
- Agrupar instâncias e usar centroides

Como treinar pesos (assumindo K_u distribuído pela função normal)

1. Escolher variância (talvez média) para cada K_u
   - Exemplo, usar EM (maximização de expectativa)
2. Então manter K_u fixo e treinar camada de saída linear
   - Métodos eficientes para obter função linear

Ou usar método de retro-propagação de erros

Aprendizado Preguiçoso vs. Impaciente

Preguiçoso: esperar antes de generalizar

- k vizinhos mais próximos

Impaciente: generalizar antes de usar

- Árvore de decisão, Naive Bayes, redes neurais

Importa?

- Aprendizado impaciente precisa criar aproximação global
- Aprendizado preguiçoso pode criar muitas aproximações locais
- Se usam a mesma linguagem H, método preguiçoso pode representar funções mais complexas (exemplo, considere $H =$ funções lineares)

Algoritmos de IBL

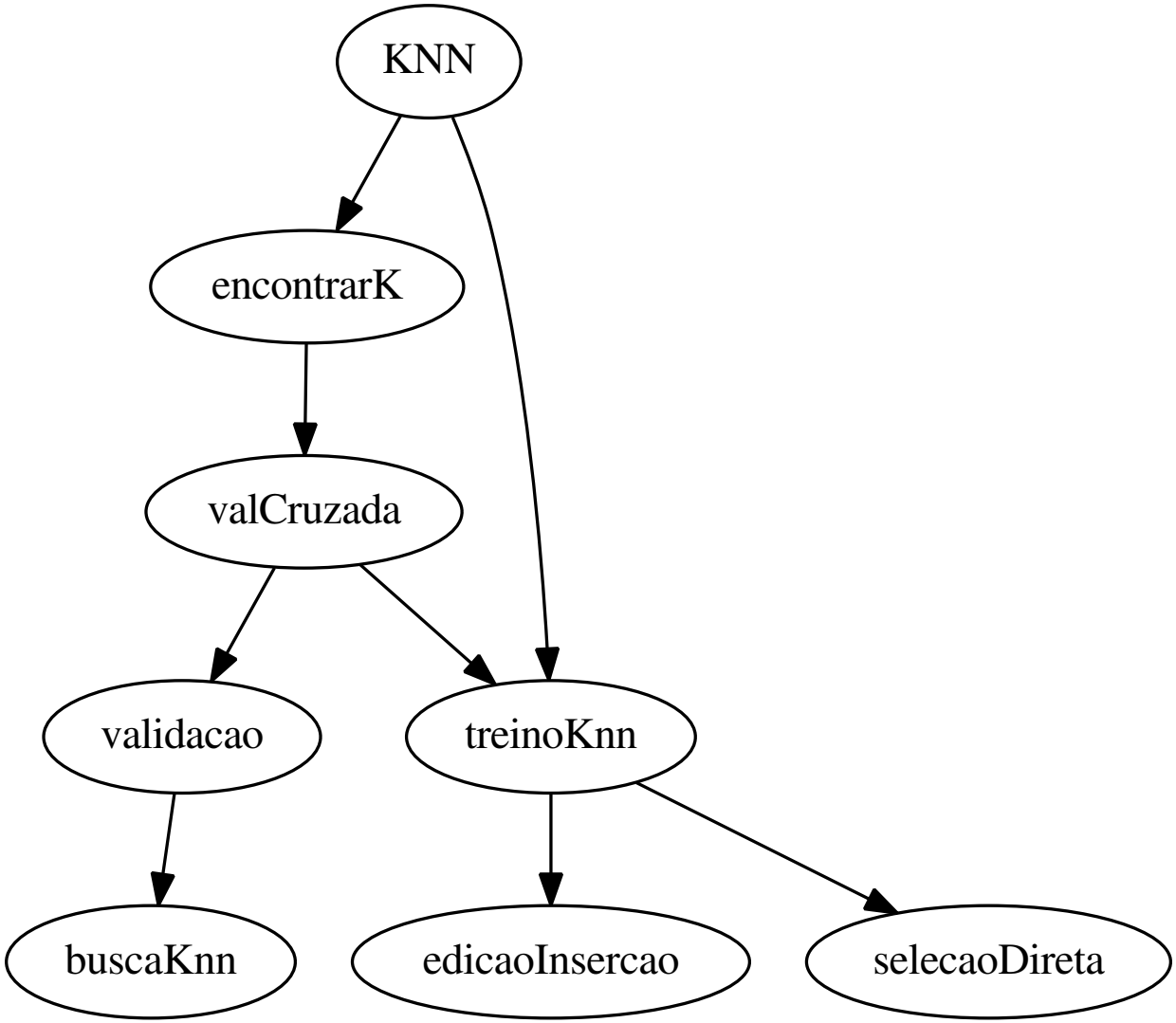


Figura: A seta saindo de `KNN` e chegando em `encontrarK` indica que `KNN` usa a função `encontrarK`.

Algoritmos de IBL

```
1  /** Inicia o processo de treino de um algoritmo IBL*/
2  Object[] KNN(Conjunto<Exemplo> base,
3               Conjunto<Exemplo> validacao,
4               Conjunto<Atributo> atributos,
5               int kMax,
6               int partes) {
7
8      int K = encontrarK(base, kMax, partes);
9      return treinoKnn(base, validacao, K);
10 }
```

Algoritmos de IBL

```
1  // Encontra o k que resulta na melhor acuracia estimada
       com
2  //  a tecnica de validacao cruzada em P partes
3  int encontrarK(Conjunto<Exemplo> base, int kMax,int P){
4
5      int melhorK = 1; int k = 1;
6      double melhorAcc = valCruzada(k, base, P);
7
8      for(int k = 2; k < kMax; k++)  {
9          double acc = valCruzada(k, base, P);
10
11         if(acc > melhorAcc) {
12             melhorAcc = acc;
13             melhorK = k;
14         }
15     }
16
17     return melhorK;
18 }
```

Algoritmos IBL

```
1  /** Implementa metodo de validaicao cruzada em nP
       partes para estimacao da acuracia */
2  double valCruzada(int k,Conjunto<Exemplo> base,int nP){
3
4   int tamP = (int) (base.size()/nP); // tam. iguais
5   Conjunto<Exemplo> partes[] = new Conjunto<>();
6   for (int p = 0; p < nP; p++) // separa as partes
7     partes[p] = sorteioSemReposicao(base, tamP);
8
9   Conjunto<Exemplo> treino, subC;
10  for(int pTreino = 0; pTreino < nP; pTreino++) {
11    treino = null;
12    for(int pVal = 0; pVal < nP; ++pVal)
13      if (pTreino != pVal) treino.add(partes[pTreino])
14
15    Object[] res = treinoKnn(treino, partes[pVal], k)
16    subC = (Conjunto) res[0];
17    Conjunto<Atributo> subA = (Conjunto) res[1];
18    acc += validacao(subC, subA, partes[val], k);
19  }
20 }
```

Algoritmos de IBL

```java
/** Calcula acuracia de um Knn em relacao a um conjunto
     de validacao.
O conjunto base e' utilizado para julgar a classe de
    elementos do conjunto de validacao. O conjunto de
    atributos define as informacoes utilizadas para
    julgar quem sao os k vizinhos mais proximos. */
double validacao(Conjunto<Exemplo> base,
                 Conjunto<Exemplo> validacao,
                 Conjunto<Atributo> atributos,
                 int k){

    double acertos = 0;

    for(Exemplo exemplo: validacao) {
        int classe = buscaKnn(exemplo, base, k);

        if (c == exemplo.classe())
            acertos++;
    }
    return acertos/(double)validacao.size();
}
```

Algoritmos de IBL

```
1  /** busca os k elementos da base mais proximos de e, e
       retorna a classe mais frequente entre esses k*/
2  int buscaKnn(Exemplo e, Conjunto<Exemplo> base, int k){
3
4      MinHeap<Distancia> maisProximos = new MinHeap<>();
5
6      for (Exemplo exemplo: base) // mede distancias
7          maisProximos.add(Distancia.get(e, exemplo));
8
9      int freqClasses[] = new int[numeroClasses];
10
11     for (int i = 0; i < k; i++) {// conta frequencia
12       int classe = maisProximos.poll(d).getClasse();
13       freqClasses[classe]++;
14     }
15
16     return maisFrequente(freqClasses);
17 }
```

Algoritmos de IBL

```
1  /** Algoritmo para obter um bom subconjunto de
       atributos e um bom subconjunto de exemplos usados
       para classificacao com k vizinhos mais proximos
2  Retorna array de Object contendo os atributos
       selecionados na primeira posicao e os exemplos mais
        importantes para classificacao na segunda posicao.
3  */
4  Object[] treinoKnn(treino, val, k){
5
6      // seleciona um subconjunto de atributos
7      subAtributos = selecaoDireta(CA, treino, val, k)
8
9      // seleciona um subconjunto de exemplos
10     subExemplos = edicaoInsercao(treino, subAtributos)
11
12     Object[] r = new Object[2];
13     r[0] = subAtributos;
14     r[1] = subExemplos;
15
16     return r;
17 }
```

IBL na prática: pacotes R

- `class::knn`: aceita somente dados numéricos
- `neighbr::knn`: aceita dados categóricos
- `kknn`: exemplos ponderados/Minkowski, kernels
- `caret`: regressão kNN
- `RWeka::IBk`

Visualização

```
## Executar
animation::knn.ani()
```

Pacote class

```
require(class)
#knn : usa dados "train" para classificar "test"
idxTrain   <- sample(nrow(iris), 50)
iris.train <- iris[idxTrain, -5]
iris.test  <- iris[-idxTrain, -5]
irisknn.predict<- knn(iris.train, iris.test,
                      iris[idxTrain,]$Species,k=5)
table(irisknn.predict, iris[-idxTrain,]$Species)

##
## irisknn.predict setosa versicolor virginica
##      setosa         36          0         0
##      versicolor      0         31         2
##      virginica       0          2        29
```

Pacote class

```
require(class)
#knn.cv : leave-one-out cross validation
res.knn.cv <- knn.cv(iris[,-5], iris$Species,k=5)
table(res.knn.cv,iris$Species)

##
## res.knn.cv    setosa versicolor virginica
##    setosa         50          0         0
##    versicolor      0         47         2
##    virginica       0          3        48
```

Regressão com kNN: pacote caret

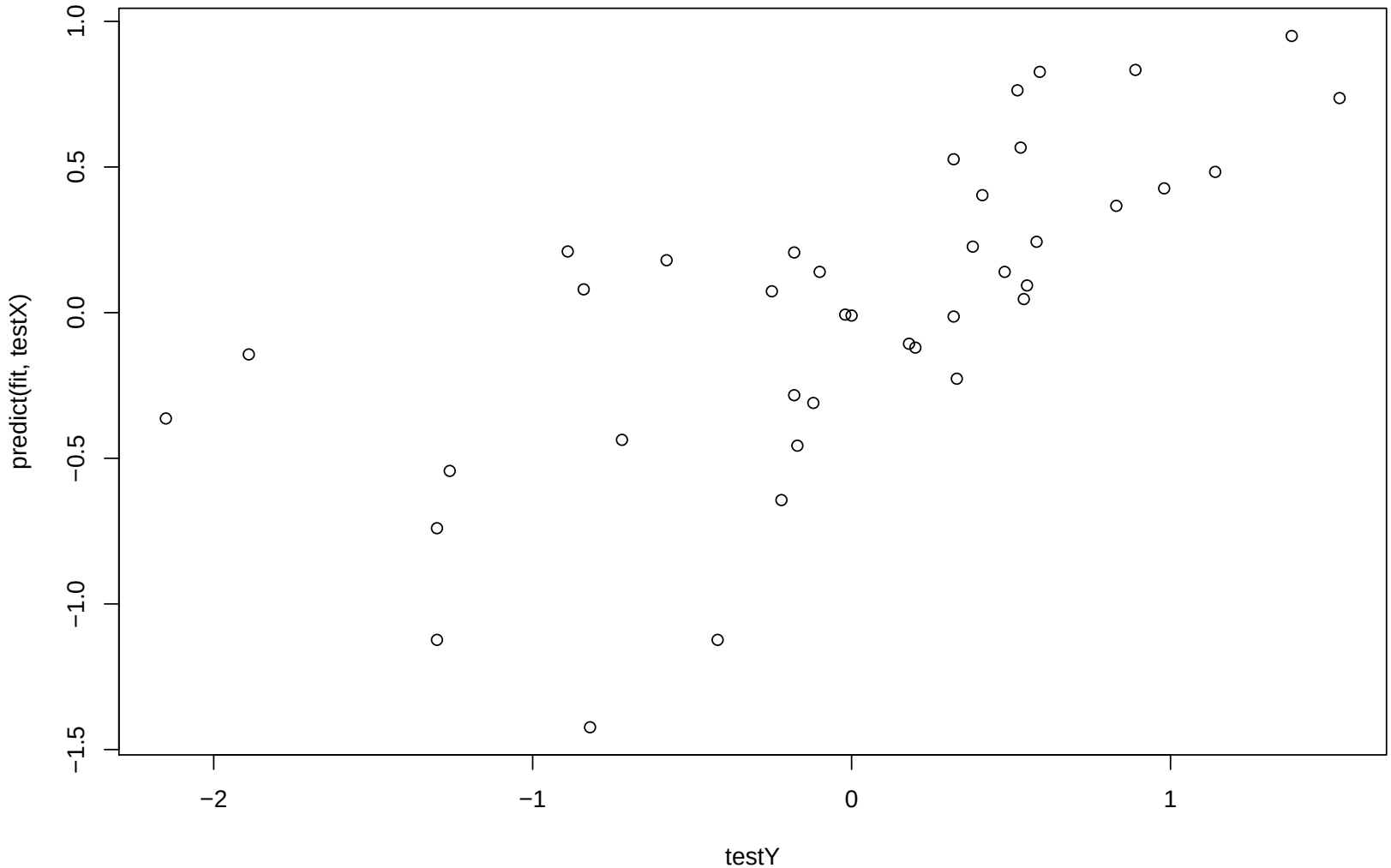
```
require(caret)
data(BloodBrain)
inTrain <- createDataPartition(logBBB, p = .8)[[1]]

trainX <- bbbDescr[inTrain,]
trainY <- logBBB[inTrain]

testX <- bbbDescr[-inTrain,]
testY <- logBBB[-inTrain]

fit <- knnreg(trainX, trainY, k = 3)
```

```
plot(testY, predict(fit, testX))
```



Pacote: RWeka

```
require(RWeka)
```

```
## Warning in library(package, lib.loc = lib.loc,
character.only = TRUE, logical.return = TRUE, :
there is no package called 'RWeka'
```

```
#  WOW("IBk") -- listar parâmetros de controle
ctl <- Weka_control(I=TRUE,K=2)
```

```
## Error in Weka_control(I = TRUE, K = 2):  não foi
possível encontrar a função "Weka_control"
```

```
IBk(Species ~. , iris, control=ctl)
```

```
## Error in IBk(Species ~ ., iris, control = ctl):
não foi possível encontrar a função "IBk"
```