Mistura de modelos

Marcelo K. Albertini

19 de Junho de 2017

Mistura de modelos

- Ideia básica:

  Em vez de aprender somente um modelo, aprender vários e combiná-los
- Isso melhora acurácia
- Muitos métodos, exemplos:
  - Bagging
  - Boosting
  - ECOC (codificação saída de correção de erro)
  - Stacking

Bagging

1. Gerar réplicas do conjunto de treino
   - usar amostragem com substituição
   - exemplo: criar 20 conjuntos-réplicas de treino com mesmas propriedades estatísticas
2. Aprender um modelo em cada réplica
   - exemplo: criar uma árvore para cada conjunto-réplica
3. Combinar por votação
   - exemplo: 15 árvores concluem que e-mail é spam e 5 que não é

Bagging

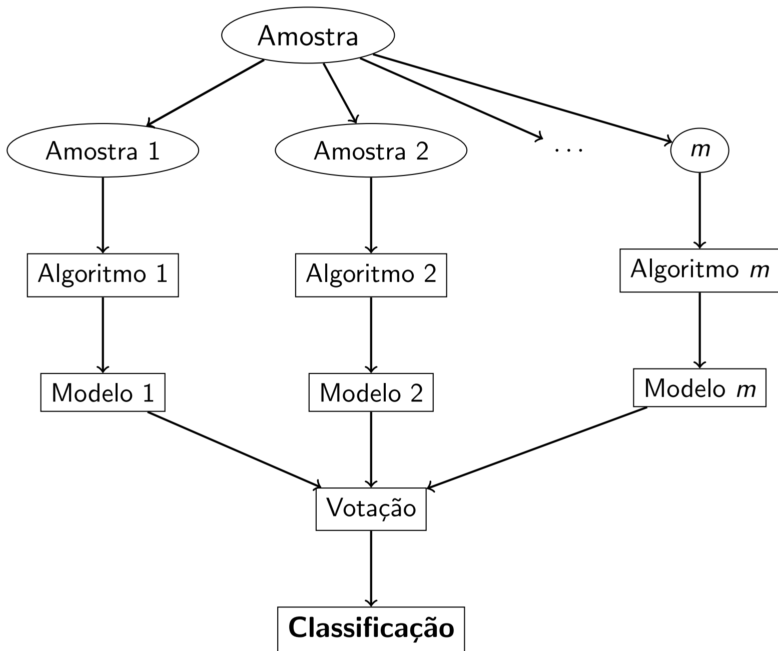
1. Gerar réplicas do conjunto de treino
   - usar amostragem com substituição
   - exemplo: criar 20 conjuntos-réplicas de treino com mesmas propriedades estatísticas
2. Aprender um modelo em cada réplica
   - exemplo: criar uma árvore para cada conjunto-réplica
3. Combinar por votação
   - exemplo: 15 árvores concluem que e-mail é spam e 5 que não é

- Útil em algoritmos que são instáveis
  - Árvore de decisão é instável, principalmente com menos poda
  - Naïve Bayes é mais estável em instâncias, porém instável em atributos

Bagging

1. Gerar réplicas do conjunto de treino
   - usar amostragem com substituição
   - exemplo: criar 20 conjuntos-réplicas de treino com mesmas propriedades estatísticas
2. Aprender um modelo em cada réplica
   - exemplo: criar uma árvore para cada conjunto-réplica
3. Combinar por votação
   - exemplo: 15 árvores concluem que e-mail é spam e 5 que não é

- Útil em algoritmos que são instáveis
  - Árvore de decisão é instável, principalmente com menos poda
  - Naïve Bayes é mais estável em instâncias, porém instável em atributos
- Bagging pode ser vista como técnica de controle de overfitting


Amostra
Amostra 1
Amostra 2
...
m
Algoritmo 1
Algoritmo 2
Algoritmo m
Modelo 1
Modelo 2
Modelo m
Votação
Classificação

Preparação de dados

- Divisão dos dados
  - `createFolds()`, `createMultiFolds()`,`createResamples()`

```
require(caret);require(C50); data(churn)
allData <- rbind(churnTrain, churnTest)
inTrainingSet <- createDataPartition(allData$churn,
                                     p=.75, list=FALSE)
churnTrain <- allData[inTrainingSet,]
churnTest  <- allData[-inTrainingSet,]
```

Bagging de árvores

```r
require(doMC); registerDoMC(8) # num. threads

ctrl <- trainControl(method="cv",repeats=2)
bagOfTrees <- train(churn ~ ., data=churnTrain,
                    method="treebag", trControl=ctrl)
table(predict(bagOfTrees, churnTest), churnTest$churn)

##
##         yes   no
##   yes   130    9
##   no     46 1064
```

Boosting

- AdaBoost: Robert Shapire e Yoav Freund
  - Qual a diferença entre aprendizado fraco e forte?
  - Aprendizado com $0.5 + \varepsilon$ é igual a $1 - \varepsilon$?

Boosting

- AdaBoost: Robert Shapire e Yoav Freund
  - Qual a diferença entre aprendizado fraco e forte?
  - Aprendizado com $0.5 + \varepsilon$ é igual a $1 - \varepsilon$?

1. Definir pesos para exemplos
2. Inicializar com pesos uniformes
3. Iterar (turnos)
   - Aplicar aprendizado para exemplos com pesos
     - se algoritmo não aceita pesos, amostrar com probabilidade proporcional aos pesos
   - Aumentar pesos dos exemplos classificados erroneamente
     - Ou reduzir daqueles classificados corretamente
4. Combinar modelos por votação com pesos

```
void treinoAdaBoost(Conjunto<Exemplo> S,
                    Classificador h[], double[] beta) {
 int e, N = S.size();
 double pesos[] = new double[N],
        probs[] = new double[N];
 for (e = 0; e < N; e++) pesos[e] = 1.0/N;//inicializa

 for (int t = 0; t < h.length; t++) { // turnos
   for (e = 0; i < N; e++) prob[e] /= soma(pesos);

   h[t] = Classificador.treinar(S, prob);
   double erroPonderado = 0;
   for (e = 0; e < N; e++) { // obter erros com pesos
       int erro = (h.classifica(e) != S.classe(e))?1:0;
       erroPonderado += prob[e] * erro;
       if (erroPonderado > 0.5) {h[t] = null; return;}
   }
   // vai reduzir pesos dos corretos
   beta[t] = erroPonderado / (1 - erroPonderado);
   for (e = 0; e < S.size(); e++) {
       int erro = (h.classifica(e) != S.classe(e))?1:0;
       pesos[e] *= Math.pow(beta[t], 1 - erro);
   }
}}
```

Classificação AdaBoost

Após treino usar

$$\arg\max_{y \in Y} \sum_{r=1}^{k} \left(\log \frac{1}{\beta_r}\right) \mathbf{1}[h_r(x) = y]$$

```
1  int adaBoostClassificador(Classificador h[], double[]
       beta, Exemplo e, int nClasses){
2    int maxC = 0; double maxSoma = Double.MIN_VALUE;
3    for (int c = 0; c < nClasses; c++) {
4      double soma = 0.0;
5
6      for (int t =0; t < h.length && h[t] != null; t++) {
7        soma += log(1/beta[t])*((h.classifica()==c)?1:0);
8      }
9      if (soma > maxSoma) {
10       maxSoma = soma;
11       maxC = c;
12     }
13   }
14   return maxC;
15 }
```

AdaBoost

```
require(freestats)
z <- runif(n=5)
mydata <- fakedata(w=z,n=100)
X<- mydata$S[,1:4]
y <- mydata$y
res <- adaBoost(dat.train=X,y.train=y,B=3)
confusionMatrix(classify(res,X),y)$table

##            Reference
## Prediction -1  1
##         -1 57  9
##         1   0 34
```

Bagging de AdaBoost (caret)

```r
grid <- expand.grid(mfinal=c(2,3), maxdepth=c(3,4,5) )
bagOfAda <- train(churn ~ ., data=churnTrain,
                  method="AdaBag", trControl=ctrl,
                  tuneGrid = grid)
bagOfAda$bestTune

##   mfinal maxdepth
## 6      3        5

table(predict(bagOfAda, churnTest), churnTest$churn)

##
##         yes   no
##   yes   126   18
##   no     50 1055
```

Codificação de correção de erro de saída

- Error-Correcting Output Coding
- **Motivação**
  - aplicar classificadores binários para problemas multi-classes

Codificação de correção de erro de saída

- Error-Correcting Output Coding
- **Motivação**
  - aplicar classificadores binários para problemas multi-classes
- **Treino** - Repetir L vezes:
  - Formar um problema binário por atribuir aleatoriamente classes a "superclasses" 0 e 1
    - h_1: A, B, D $\rightarrow$ 0 e C,E $\rightarrow$ 1
    - h_2: B, D, E $\rightarrow$ 0 e A,C $\rightarrow$ 1
    - h_3: A, B $\rightarrow$ 0 e C, D, E $\rightarrow$ 1
    - h_4: A, E $\rightarrow$ 0 e B, C, D $\rightarrow$ 1
  - Cada classe é representada por um vetor binário
    - A = 0100, B = 0001, C=1111, D=0011, E=1010

Codificação de correção de erro de saída

- Error-Correcting Output Coding
- **Motivação**
  - aplicar classificadores binários para problemas multi-classes
- **Treino** - Repetir L vezes:
  - Formar um problema binário por atribuir aleatoriamente classes a "superclasses" 0 e 1
    - h_1: A, B, D $\rightarrow$ 0 e C,E $\rightarrow$ 1
    - h_2: B, D, E $\rightarrow$ 0 e A,C $\rightarrow$ 1
    - h_3: A, B $\rightarrow$ 0 e C, D, E $\rightarrow$ 1
    - h_4: A, E $\rightarrow$ 0 e B, C, D $\rightarrow$ 1
  - Cada classe é representada por um vetor binário
    - A = 0100, B = 0001, C=1111, D=0011, E=1010
- **Teste**
  - Aplicar cada classificador para exemplo de teste, formando vetor de predições P
  - Predizer classes cujo vetor é o mais próximo a P (distância de Hamming)
    - Se $h_1 = 1$, $h_2 = 0$, $h_3 = 0$, $h_4 = 0$ então a classe é o que tem mais bits em comum, ou seja, E=1010.

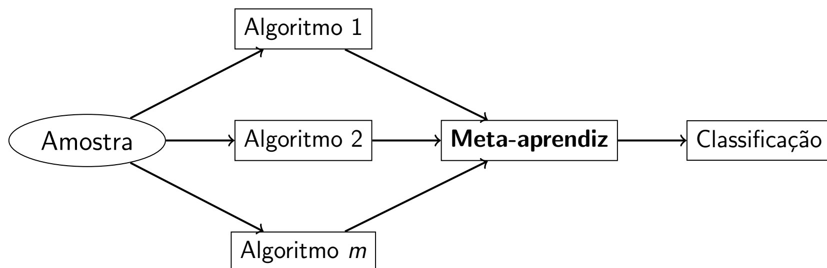
ECOC usando pacote R mlr

```r
require(mlr)

task = makeClassifTask(data = iris, target = "Species")

lrn  <- makeLearner("classif.lda")
lrnMulticlass <- makeMulticlassWrapper(lrn) # ECOC

mod  <- train(lrnMulticlass, task)
pred <- predict(mod, task)
```

```r
calculateConfusionMatrix(pred)

##               predicted
## true            setosa versicolor virginica -err.-
##   setosa            50          0         0      0
##   versicolor         7         31        12     19
##   virginica          0         11        39     11
##   -err.-             7         11        12     30
```

Stacking (Empilhamento) – Meta-aprendizado

- Aplicar múltiplos algoritmo de aprendizado base
  - Exemplo: árvores de decisão, Naive Bayes, redes neurais
- Meta-aprendizado: entradas = predições dos algoritmos base
- Perigo: *overfitting*
  - Solução: treino por validação cruzada do tipo deixa-um-fora



Meta-aprendizado com “deixa um fora”

```
1  void meta(Conjunto<Exemplo> treino,
2      Classificador metaAlg, Classificador[] algoritmos){
3
4    Conjunto<Exemplo> metaExemplos;
5    for (Exemplo umFora: treino) {
6      Conjunto C = treino.subtracao(umFora);
7      int i = 0;
8      int[] metaAtributos = new int[algoritmos.size()];
9      for (Classificador a: algoritmos) {
10         metaAtributos[i++] = a.treino(C).classifica(
               umFora);
11     }
12
13     metaExemplos.add(new Exemplo(metaAtributos, umFora.
           classe()));
14   }
15   for (Classificador a: algoritmos) {
16     a.treino(treino);
17   }
18   metaAlg.treino(metaExemplos);
19 }
```

Pacote caretEnsemble

```
require(mlbench); require(caret); require(caretEnsemble)
data(Ionosphere)
dataset <- Ionosphere[,-2]
dataset$V1 <- as.numeric(as.character(dataset$V1))

algs <- c('lda', 'rpart')
ctl <- trainControl(method="repeatedcv", number=10,
          repeats=3, savePredictions=TRUE, classProbs=TRUE)

models <- caretList(Class~., data=dataset, trControl=ctl,
                    methodList=algs)
stack.randFor <- caretStack(models, method="rf",
                     metric="Accuracy", trControl=ctl)

## note: only 1 unique complexity parameters in default gri
```

```
confusionMatrix(predict(stack.randFor,dataset),
                dataset$Class)$table

##           Reference
## Prediction bad good
##       bad  113   14
##       good  13  211
```

Mistura de modelos: sumário

- Aprender vários modelos e combinar
- Bagging: reamostragem aleatória
- Boosting: reamostragem ponderada
- ECOC: de 2 classes para várias classes
- Stacking: múltiplos algoritmos de aprendizado e 1 meta-aprendiz

Pacotes R

- caret
- mlr

Pacote caret

- R caret http://caret.r-forge.r-project.org/

```
require(caret)
```

- `churn`: perda de clientes de telecom

```
require(C50)
data(churn)
```

```r
str(churnTrain)
```

```
## 'data.frame': 3333 obs. of  20 variables:
##  $ state                         : Factor w/ 51 levels "A
##  $ account_length                : int  128 107 137 84 75
##  $ area_code                     : Factor w/ 3 levels "ar
##  $ international_plan            : Factor w/ 2 levels "no
##  $ voice_mail_plan               : Factor w/ 2 levels "no
##  $ number_vmail_messages         : int  25 26 0 0 0 0 24
##  $ total_day_minutes             : num  265 162 243 299 1
##  $ total_day_calls               : int  110 123 114 71 11
##  $ total_day_charge              : num  45.1 27.5 41.4 50
##  $ total_eve_minutes             : num  197.4 195.5 121.2
##  $ total_eve_calls               : int  99 103 110 88 122
##  $ total_eve_charge              : num  16.78 16.62 10.3
##  $ total_night_minutes           : num  245 254 163 197 1
##  $ total_night_calls             : int  91 103 104 89 121
##  $ total_night_charge            : num  11.01 11.45 7.32
##  $ total_intl_minutes            : num  10 13.7 12.2 6.6
```

```r
str(churnTest)
```

```
## 'data.frame': 1667 obs. of  20 variables:
##  $ state                         : Factor w/ 51 levels "A
##  $ account_length                : int  101 137 103 99 10
##  $ area_code                     : Factor w/ 3 levels "ar
##  $ international_plan            : Factor w/ 2 levels "no
##  $ voice_mail_plan               : Factor w/ 2 levels "no
##  $ number_vmail_messages         : int  0 0 29 0 0 0 32 0
##  $ total_day_minutes             : num  70.9 223.6 294.7
##  $ total_day_calls               : int  123 86 95 123 78
##  $ total_day_charge              : num  12.1 38 50.1 36.9
##  $ total_eve_minutes             : num  212 245 237 126 1
##  $ total_eve_calls               : int  73 139 105 88 101
##  $ total_eve_charge              : num  18 20.8 20.2 10.7
##  $ total_night_minutes           : num  236 94.2 300.3 22
##  $ total_night_calls             : int  73 81 127 82 107
##  $ total_night_charge            : num  10.62 4.24 13.51
##  $ total_intl_minutes            : num  10.6 9.5 13.7 15.
```

- 1º nível da variável é o interesse: "yes"

```
predictors <- names(churnTrain)[
                 names(churnTrain) != "churn"]
```

Pré-processamento

```r
numerics <- c("account_length", "total_day_calls",
              "total_night_calls")
procValues <- preProcess(churnTrain[,numerics],
                method=c("center", "scale", "YeoJohnson"))
trainScaled <- predict(procValues, churnTrain[, numerics])
testScaled  <- predict(procValues, churnTest[,numerics])
```

```
procValues

## Created from 3333 samples and 3 variables
##
## Pre-processing:
##   - centered (3)
##   - ignored (0)
##   - scaled (3)
##   - Yeo-Johnson transformation (3)
##
## Lambda estimates for Yeo-Johnson transformation:
## 0.89, 1.17, 0.93
```

Boosted trees: pacote gbm

- Parâmetros:
  - Número de iterações (de árvores)
  - Complexidade da árvore
  - Taxa de aprendizado: shrinkage

```
require(gbm)
forGBM <- churnTrain
forGBM$churn <- ifelse(forGBM$churn == "yes", 1, 0)

gbmFit <- gbm(formula = churn ~. ,
              distribution = "bernoulli",
              data = forGBM,
              n.trees = 2000,
              interaction.depth = 7,
              shrinkage = 0.01,
              verbose = FALSE)
```

Tuning

```
gbmTune <- train(churn ~. ,  # usando Formula
                 data= churnTrain,
                 method = "gbm",
                 verbose=FALSE)
## gbmTune <- train(x = churnTrain[,predictors],
##                  y = churnTrain£churn,
##                  method = "gbm",
##                  verbose=FALSE)
```

```
gbmTune

## Stochastic Gradient Boosting
##
## 3333 samples
##   19 predictor
##    2 classes: 'yes', 'no'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 3333, 3333, 3333, 3333, 3333, 3
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.8641599  0.2184827
##   1                  100      0.8723765  0.3271875
##   1                  150      0.8755494  0.3690583
##   2                   50      0.9100793  0.5497650
##   2                  100      0.9303487  0.6794514
```

Busca em grade

```
ctrl <- trainControl(method="repeatedcv", repeats = 2,
            classProbs = TRUE)

grid <- expand.grid(.interaction.depth = seq(1, 7, by = 3),
                    .n.trees = seq(100, 1000, by = 300),
                    .shrinkage = c(0.01, 0.1),
                    .n.minobsinnode = c(1,2,3))

gbmTune <- train(churn ~ . , data = churnTrain,
                 method = "gbm",
                 tuneGrid = grid,
                 verbose = FALSE,
                 trControl = ctrl)
```

Busca em grade com curva ROC

```
ctrl <- trainControl(method="repeatedcv", repeats = 5,
            classProbs = TRUE,
            summaryFunction = twoClassSummary)#ROC AUC

grid <- expand.grid(.interaction.depth = seq(1, 7, by = 2),
                    .n.trees = seq(100, 1000, by = 50),
                    .shrinkage = c(0.01, 0.1),
                    .n.minobsinnode = c(1,2,3))

gbmTune <- train(churn ~ . , data = churnTrain,
                 method = "gbm",
                 metric = "ROC",
                 tuneGrid = grid,
                 verbose = FALSE,
                 trControl = ctrl)
```

```r
gbmPred <- predict(gbmTune, churnTest)
str(gbmPred)

##  Factor w/ 2 levels "yes","no": 2 2 2 2 2 2 2 2 2 2 ...
```

```r
gbmProbs <- predict(gbmTune, churnTest, type="prob")
plot(gbmProbs)
```