Remarks on bagging and boosting

Introduction to Machine Learning – Lab Sessions

by Martin Holub
Charles University in Prague, 2010-11

Combining multiple learners

- the more **complementary** the learners are, the more useful their combining is
- the simpliest way to combine multiple learners is **voting**
- in **weighted voting** the voters (= base-learners) can have different weights

Unstable learning

- learning algorithm is called unstable if small changes in th etraining set cause large differences in generated models
- typical unstable algorithm is the decision trees learning
- bagging or boosting techniques are a natural remedy for unstable algorithms

Bagging

- Bagging is a voting method that uses slightly different training sets (generated by bootstrap) to make different base-learners. Generating complementary base-learners is left to chance and to unstability of the learning method.

Boosting

- Boosting is one of the most important developments in classification methodology. Boosting works by sequentially applying a classification algorithm to reweighted versions of the training data and then taking a weighted majority vote of the sequence of classifiers thus produced. For many classification algorithms, this simple strategy results in dramatic improvements in performance.

  [from Wikipedia]

- Like bagging, boosting is also a voting method. In contrast to bagging, boosting actively tries to generate complementary learners by training the next learner on the mistakes of the previous learners.

- **AdaBoost** (Adaptive Boosting)
  - originally proposed by Freund and Schapire (1996)
  - nice presentation including theoretical details and a demonstration available at http://cmp.felk.cvut.cz/~sochmj1/adaboost_talk.pdf

Implementation in R

- packages can be found at http://cran.at.r-project.org/
- `bagging()` (package: adabag)
- `ada()` (package: ada)
  - only binary classification
  - nice visualization
- `adaboost.M1()` (package: adabag)
  - simplier than `ada()`
  - multiple classes

ADABAG package in R

- implements Adaboost.M1 algorithm and Breiman's Bagging algorithm using classification trees
- Adaboost.M1 is a simple generalization of Adaboost for more than two classes
- a comprehensive reference manual available at http://cran.at.r-project.org/
- installation

```
> install.packages(“adabag”)
> library(adabag)
```

ADA package in R

- creates a classification model as an ensemble of rpart trees
- uses the rpart library as its engine
- can handle only two-class problems
- documentation at http://cran.at.r-project.org/
  also a comprehensive paper available at
  http://www.stat.wvu.edu/~mculp/math/ada/ada_manual.pdf
- another interesting reading/tutorial
  http://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Classification/adaboost

- installation

```
> install.packages(“ada”)
> library(ada)
```

- help pages for package 'ada' – list of R functions

ada	Fitting Stochastic Boosting Models
addtest	Add a test set to ada
pairs.ada	Pairwise Plots and Variable Importancs Plot for Ada
plot.ada	Plots for Ada
predict.ada	Predict a data set using Ada
print.ada	Model Information for Ada
soldat	Solubility Data
summary.ada	Summary of model fit for arbitrary data (test, validation, or training)
update.ada	Add more trees to an ada object

Practical exercise

 # use the „Solubility data“ from package ada

```
> library(ada)
> data("soldat")
> N <- nrow(soldat)
> set.seed(123); ind <- sample(1:N)

> train_size      <- N %/% 2
> train           <- soldat[ind[1:train_size],]
> test            <- soldat[ind[(train_size + 1):N],]
```

 # make a decision tree model, tune the parameters without using the test set, and (only then) compute accuracy on the test set

 # then use bagging() and compare the results

 # finally use adaboost.M1() and compare the results

****

Example solution using ada()

```
> train_size      <- N %/% 2
> test_size       <- N %/% 3
> train     <- soldat[ind[1:train_size],]
> test      <- soldat[ind[(train_size + 1):(train_size + test_size)],]
> valid     <- soldat[ind[(train_size + test_size + 1):N],]
# just to illustrate that you can work with more than one test set

> control   <- rpart.control(cp = -1, maxdepth = 14, xval = 0)
# cp = -1 forces the tree to split until the depth of the tree achieves the maxdepth setting
# xval is the number of cross-validations

> m   <- ada(y~., data = train, test.x = test[,-73], test.y = test[,73],
      + type = "gentle", control = control, iter = 70)

> summary(m)
> plot(m,test=T)
> varplot(m)      # shows the relationship between descriptors and the response
                  # the variable importance measure is based on improvement

> m1 <- addtest(m, valid[,-73], valid[,73])
> summary(m1)
> plot(m1,test=T)
> varplot(m1)
```