STAT562 Lecture 12 Boosting

Beidi Qiang

SIUE

Ensemble Methods

The basic idea behind boosting is converting many weak learners to form a single strong learner.

- A weak classifier is one that performs better than random guessing, but still performs poorly at designating classes to objects.
- A model formed using a single predictor/classifier is not powerful individually.

We create an ensemble model by combining several weak leaners.

Type of Ensemble Methods

The modes that combines several base algorithms to form one optimized predictive algorithm is called Ensemble Methods. Ensemble Methods can also be divided into two groups:

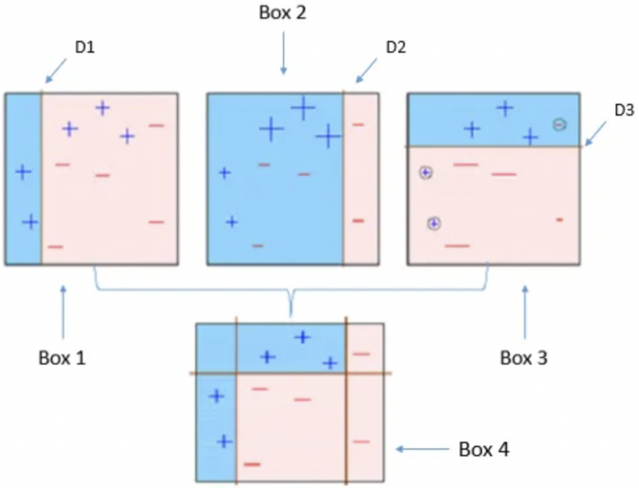
- Sequential Learners, where different models are generated sequentially and the mistakes of previous models are learned by their successors. This aims at exploiting the dependency between models by giving the mislabeled examples higher weights (e.g. AdaBoost).
- Parallel Learners, where base models are generated in parallel. This exploits the independence between models by averaging out the mistakes (e.g. Random Forest).

Boosting

Recall in bagging, each tree is built on a bootstrap data set, independent of the other trees. Boosting works in a similar way, except that the trees are grown sequentially: each tree (Decision Stump) is grown using information from previously trees (stumps).

- The boosting algorithm assigns equal weight to each data sample. It feeds the data to the first model, called the base algorithm. The base algorithm makes predictions for each data sample.
- Now we do the following till maximum number of iteration is reached:
  - Assesses model predictions and increases the weight of samples with a more significant error.
  - We then passes the weighted data to the train next decision tree.
- After the iterations have been completed, we combine weak rules (trees) to form a single strong rule, which will then be used as our model.

Example:



Adaptive Boosting (AdaBoost)

Type of boosting approach described in the previous example is known as Adaptive Boosting or AdaBoost. Weights are calculated as follows:

- x is the set of data features. y is the target variable.
- Initially, $w = 1/n$ for all examples.
- At each step, we calculate:

$$Weakness(error) = \frac{\sum_{i=1}^{n} w_i I(\hat{y}_i \neq y_i)}{\sum_{i=1}^{n} w_i}$$

$$\alpha = ln\left(\frac{1 - error}{error}\right)$$

$$\text{update weights} = w_i \cdot exp(\alpha I(\hat{y}_i \neq y_i))/Z$$

The algorithm trains the weak classifier at each step with the current sample weights. Weak models are added sequentially. Once completed, you are left with a pool of weak learners each with a α value. The final prediction is obtained by aggregating the predictions.

Pseudocode

```
Algorithm 1  AdaBoost (Freund & Schapire 1997)
  1. Initialize the observation weights $w_i = 1/n,\ i = 1, 2, \ldots, n$.
  2. For $m = 1$ to $M$:
      (a) Fit a classifier $T^{(m)}(\boldsymbol{x})$ to the training data using weights $w_i$.
      (b) Compute
              $err^{(m)} = \sum_{i=1}^{n} w_i \mathbb{I}\left(c_i \neq T^{(m)}(\boldsymbol{x}_i)\right) / \sum_{i=1}^{n} w_i.$
      (c) Compute
              $\alpha^{(m)} = \log \frac{1 - err^{(m)}}{err^{(m)}}.$
      (d) Set
              $w_i \leftarrow w_i \cdot \exp\left(\alpha^{(m)} \cdot \mathbb{I}\left(c_i \neq T^{(m)}(\boldsymbol{x}_i)\right)\right),\ i = 1, 2, \ldots, n.$
      (e) Re-normalize $w_i$.
  3. Output
              $C(\boldsymbol{x}) = \arg\max_k \sum_{m=1}^{M} \alpha^{(m)} \cdot \mathbb{I}(T^{(m)}(\boldsymbol{x}) = k).$
```

Advantages and Disadvantages of AdaBoost

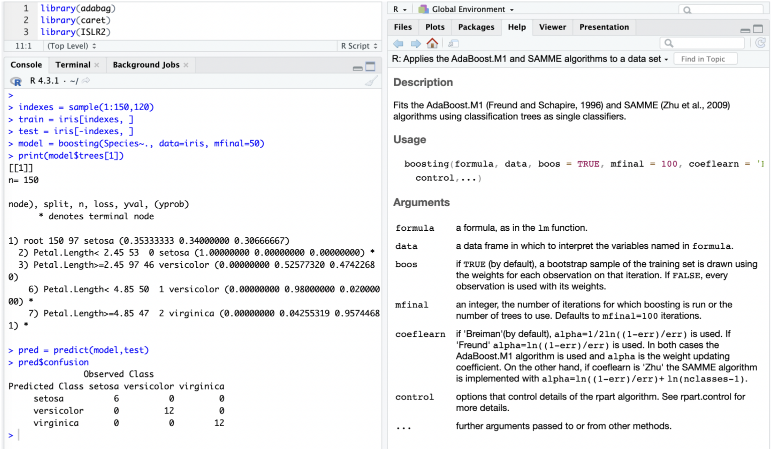
Advantages of AdaBoost:

- Easier to use with less need for tweaking parameters.
- Boosting can be used to improve the accuracy of your weak classifiers hence making it flexible.
- It can be extended beyond binary classification.
- Boosting is an approach that learns slowly, reducing the potentially overfitting issue.

A few Disadvantages of AdaBoost are :

- AdaBoost is also extremely sensitive to Noisy data and outliers.
- AdaBoost has also been proven to be slow.

Example: Iris data



```
library(adabag)
library(caret)
library(ISLR2)
```

```
>
> indexes = sample(1:150,120)
> train = iris[indexes, ]
> test = iris[-indexes, ]
> model = boosting(Species~., data=iris, mfinal=50)
> print(model$trees[1])
[[1]]
n= 150

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 150 97 setosa (0.35333333 0.34000000 0.30666667)
  2) Petal.Length< 2.45 53  0 setosa (1.00000000 0.00000000 0.00000000) *
  3) Petal.Length>=2.45 97 46 versicolor (0.00000000 0.52577320 0.4742268
0)
    6) Petal.Length< 4.85 50  1 versicolor (0.00000000 0.98000000 0.020000
00) *
    7) Petal.Length>=4.85 47  2 virginica (0.00000000 0.04255319 0.9574468
1) *

> pred = predict(model,test)
> pred$confusion
               Observed Class
Predicted Class setosa versicolor virginica
     setosa          6          0         0
     versicolor      0         12         0
     virginica       0          0        12
>
```

R: Applies the AdaBoost.M1 and SAMME algorithms to a data set

Description

Fits the AdaBoost.M1 (Freund and Schapire, 1996) and SAMME (Zhu et al., 2009) algorithms using classification trees as single classifiers.

Usage

```
boosting(formula, data, boos = TRUE, mfinal = 100, coeflearn = '
  control,...)
```

Arguments

`formula`	a formula, as in the `lm` function.
`data`	a data frame in which to interpret the variables named in `formula`.
`boos`	if `TRUE` (by default), a bootstrap sample of the training set is drawn using the weights for each observation on that iteration. If `FALSE`, every observation is used with its weights.
`mfinal`	an integer, the number of iterations for which boosting is run or the number of trees to use. Defaults to `mfinal=100` iterations.
`coeflearn`	if 'Breiman'(by default), `alpha=1/2ln((1-err)/err)` is used. If 'Freund' `alpha=ln((1-err)/err)` is used. In both cases the AdaBoost.M1 algorithm is used and `alpha` is the weight updating coefficient. On the other hand, if coeflearn is 'Zhu' the SAMME algorithm is implemented with `alpha=ln((1-err)/err)+ ln(nclasses-1)`.
`control`	options that control details of the rpart algorithm. See rpart.control for more details.
`...`	further arguments passed to or from other methods.

Boosting

Recall in bagging, each tree is built on a bootstrap data set, independent of the other trees. Boosting works in a similar way, except that the trees are grown sequentially: each tree (Decision Stumps) is grown using information from previously decision Stumps. The algorithm is

- Start with $\hat{f}(x) = 0$ and residuals $r_i = y_i$.
- b=1,2,...,B, Fit a new tree $\hat{f}^b$ with d splits to the data (X, r)
- update $\hat{f}$ by adding a shrunken of the new tree: $\hat{f}(x) + \lambda \hat{f}^b(x)$
- update the residual $r_i \to r_i - \lambda \hat{f}^b(x_i)$.
- Repeat, and finally output

$$\hat{f}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^b(x)$$

.

Comments

- In boosting we fit a tree using the current residual, rather than the outcome Y.
- Each of these trees can be rather small with d split (Often $d = 1$ works well).
- The shrinkage parameter λ, a small positive number usually 0.01 or 0.001, slows the updating process down more.
- Boosting is a approach that learns slowly, reducing the potentially overfitting issue and usually performs better.
- Boosting is a general approach that can be applied to many statistical learning methods, not restricted to tree.

Example: Boston data

```
library(gbm)
boost.boston=gbm(medv~.,data=Boston[train,],
distribution="gaussian",n.trees=5000, interaction.depth=4)
summary(boost.boston)
yhat.boost = predict(boost.boston,newdata=Boston[-train,])
```