Boosting

September 29, 2019

Motto

Dictionary

boost *[bu:st] verb gerund or present participle: boosting*
help or encourage (something) to increase or improve.
"a range of measures to boost tourism"

synonyms: *improve, raise, uplift, increase, augment, magnify, swell, amplify, enhance, encourage, heighten, help, promote, foster, nurture, arouse, stimulate, invigorate, revitalize, inspire, perk up;*

Setting

- Consider a two-class problem, with the output variable coded as $Y \in \{-1, 1\}$.
- Given a vector of predictor variables X, a classifier $G(X)$ produces a prediction taking one of the two values $\{-1, 1\}$. The error rate on the training sample is

$$err = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}_{y_i^c}(G(x_i)),$$

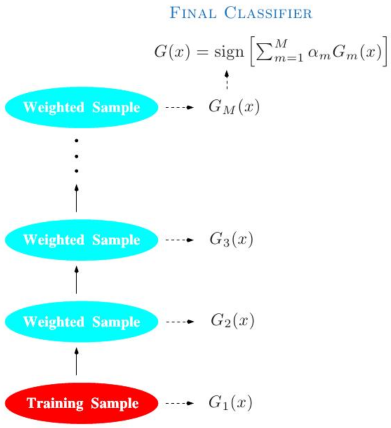
  where y^c is of the opposite sign to y, i.e. $y^c = -y$.
- A **weak classifier** is one whose error rate is only slightly better than **random guessing** (what would be the rate for this?).
- Boosting sequentially applies the weak classification algorithm to repeatedly modified versions of the data and producing in the process a sequence of classifiers $G_1, \ldots, G_M$.
- Data are modified to improve the success rate (reduce the error rate).

Majority vote

- For a sequence of qualifiers $G_1(x), \ldots, G_M(x)$, the majority vote with non-negative weights α_i, $\sum_{i=1}^{M} \alpha_i = 1$ is given by

$$G(x) = \operatorname{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$$

- It is considered that $G_1(x)$'s is a weak classifier (not very accurate) but by intelligently modifying data subsequent $G_m(x)$, $m > 1$ are improving their performance.
- The improvement of performance is obtained by weighting data so that the emphasis is on the data that were previously misclassified.



Learning on mistakes

- Improvement of performance at each step is obtained by weighting the data so that the **emphasis is on the data that were previously misclassified**.
- The weights are higher for misclassified data and lower for the correct one
- It can be viewed that the misclassified data increase their percentage presence (they are 'repeated' according to the weights or, in other words, their empirical distribution is replaced by the distribution given by the weights)
- Start with $w_{1i} = 1/N$, $i = 1, \dots N$. For a given $\alpha_m > 0$ (details on its choice are given later), the **recurrently updated weights** are

$$w_{mi} \sim w_{m-1i} \exp(\alpha \mathbb{I}_{y_i^c}(G_m(x_i)))$$

and put more emphasis on the misclassified data, where y^c is the opposite sign to y and $\mathbb{I}_{y_i^c}(y)$ is the indicator function.

Summary

- We obtain a sequence of classifiers $G_1(x), \dots, G_M(x)$ and each of these classifiers is simple – for example, a **stump**: a single branching point $\tilde{x}_{m,i_m}$ and the rule that classifies according to the i_mth coordinate being bigger or smaller than $\tilde{x}_{m,i_m}$
- They work on **subsequently modified data** (w_{mi}, x_i), i.e. on the data with distribution given by the weights w_{mi}, $i = 1, \dots, N$. These weights implicitely belong to the decision rule and the **logit of the missclassification rate** is evaluated for each rule

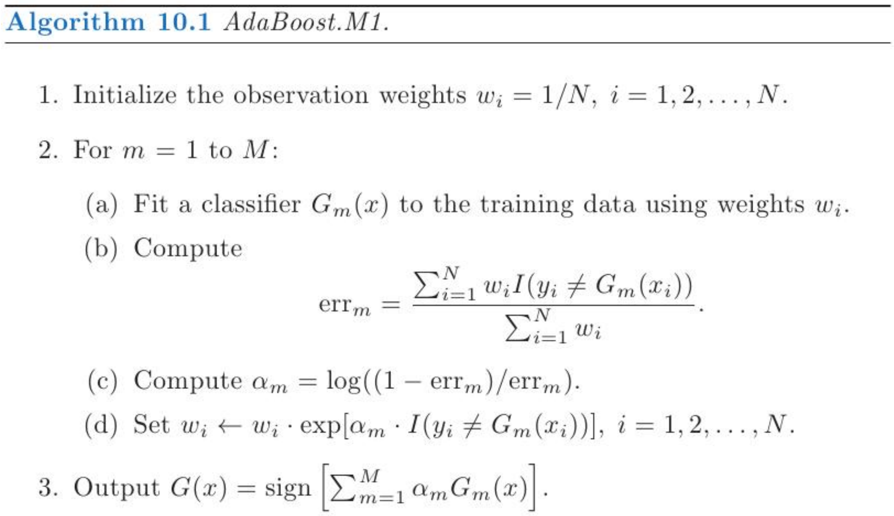
$$\alpha_m = \log \frac{1 - err_m}{err_m}$$

- We note that α_m should be positive (error rate less 50%) and do not need to be rescaled to add to one. The final classifier is given by 'popular vote'

$$G(x) = \operatorname{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$$

Ada Boost Algorithm

Here are details of the adaptive boosting algorithm



Algorithm 10.1 *AdaBoost.M1.*

```
1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to $M$:
    (a) Fit a classifier $G_m(x)$ to the training data using weights $w_i$.
    (b) Compute
            $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
    (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
    (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.
```

Details on stump classifiers

How G_i's are chosen?

- The data: (x_i, y_i).
- The weights: $w_m = (w_{im})$.
- A stump $G_{\tilde{x}_m}$ (single branched tree):
  $\tilde{x}_m$ for the i_m coordinate and $G(x_i)$ is 1 or -1 depending if the i_mth coordinate of x_i, i.e. x_{i,i_m} is bigger or smaller than $\tilde{x}_m$.
- The error rate with respect to weights is the expected value of $\mathbb{I}_{y^c}(G_{\tilde{x}_m}(x))$ with respect to the distribution on x_i's given by the weights, i.e.
  $$err_{w_m} = \sum_{i=1}^{N} \mathbb{I}_{y_i^c}(G_{\tilde{x}_m}(x_i))w_{im},$$
- Choose $(i_m, \tilde{x}_{m,i_m})$ that yields the most optimal (minimal error) among all possible stumps as measured by err_{w_m}

Illustrative example

- The following artificial example illustrates the power of boosting
- The response is deterministic

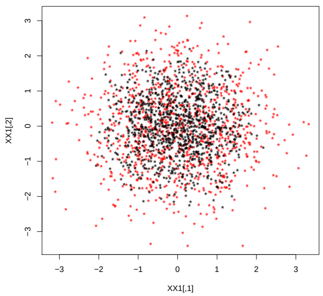
$$Y = \begin{cases} 1 & : \sum_{j=1}^{10} X_j^2 > 9.34 \\ -1 & : \text{ otherwise} \end{cases}$$

  X_j are iid standard normal, 9.34 is the median of the chi-square distribution with 10 degrees of freedom
- The weak classifier is a simple tree with two sets in the partition – a **“stump”**.
- The splitting variable and the split point are based on minimizing the mean square error with respect to the distribution given by the weights.
- The error variable is either zero (correct) or one (misclassified), thus mean square error is the same as the mean value. Why?

Simulating data

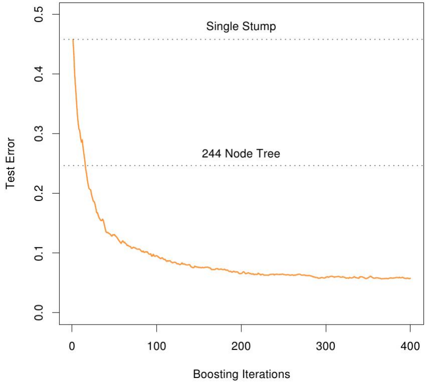
```
d=10
NTrain=2000
NTest=10000

#Predictors
XTrain=matrix(rnorm(NTrain*d),ncol=10)
XTest=matrix(rnorm(NTest*d),ncol=10)

#Response
YTrain=diag(XTrain%*%t(XTrain))>=qchisq(0.5,d)
YTest=diag(XTest%*%t(XTest))>=qchisq(0.5,d)
sum(YTrain)
sum(YTest)

#Ploting 2-dimensional projection

XX1=XTrain[YTrain==1,1:2]
XX2=XTrain[YTrain==0,1:2]

#quartz()
pdf("boostdata.pdf")
plot(XX1,col='red',pch="*")
points(XX2,pch="*")
dev.off()
```

2-dimensional projection of the training data



Results

- The initial classifier (based on a single stump) gets the error rate 45.8% – only slightly better than guessing
- Boosting improves to the error rate 5.8% after 400 iteration.
- It also outperforms a single large classification tree (error rate 24.7%).
- “best off-the-shelf classifier in the world”



Additive logistic regression – back to the past

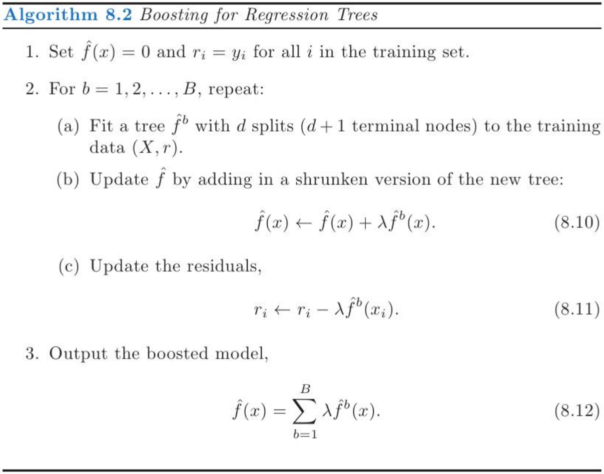
What kind of performance do you expect for the presented Illustrative example from the linear logistic regression?

What about the performance of the additive logistic regression?

A regression version of boosting

- Fitting a single large decision tree to the data has potential of overfitting – the **boosting approach** instead learns slowly.
- Regression boosting involves combining a large number of decision trees, $\widehat{f}_1, \ldots \widehat{f}_B$ in a **recursive process** to obtain a combined fit $\widehat{f}$.
- We fit a decision tree to the residuals from the model in the previous step of boosting – we fit a tree using the **current residuals**, rather than the outcome Y.
- The new decision tree is **added** into the fitted function in order to update the residuals.
- Each tree is rather small, with just a few terminal nodes, controled by the parameter d – **number of splits**.
- Fitting **small trees** slowly improves $\widehat{f}$ in areas where it does not perform well.
- The **shrinkage parameter** λ slows the process down

Regression boosting algorithm



Algorithm 8.2 *Boosting for Regression Trees*

```
1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all $i$ in the training set.
2. For $b = 1, 2, \ldots, B$, repeat:
    (a) Fit a tree $\hat{f}^b$ with $d$ splits ($d + 1$ terminal nodes) to the training data $(X, r)$.
    (b) Update $\hat{f}$ by adding in a shrunken version of the new tree:
            $\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$   (8.10)
    (c) Update the residuals,
            $r_i \leftarrow r_i - \lambda \hat{f}^b(x_i).$   (8.11)
3. Output the boosted model,
            $\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x).$   (8.12)
```

Regression vs. classification boosting

- The regression boosting fit is made of many regression trees $f_b(x)$, $b = 1, \ldots, B$.
- The trees are rather simple, for example, **stumps**. These are defined by split points x_{b,i_b} and two values $f_{b,i_b}^{(1)}$ and $f_{b,i_b}^{(2)}$ yielding $f_b(x)$ depending on if $x_{i_b} \geq x_{b,i_b}$ or not (here x_{i_b} is the i_bth coordinate of x).
- Similarly to the boosting in classification, it does change the data in the building tree process.
- However it rather changes the response values y_i's by replacing them by recalculated residuals. It does not modify x_i's.
- Those trees are contributing with the same **small weight** λ

$$f(x) = \lambda \sum_{b=1}^{B} f_b(x)$$

Comments on regression boosting

Boosting has three tuning parameters:

- The number of trees B. Boosting can overfit if B is too large, although this overfitting tends to occur slowly with increase of B (if at all).
- Typically **cross-validation** is used to select B.
- The shrinkage parameter λ, a small positive number. This controls the rate at which boosting learns.
- Typical values are 0.01 or 0.001, and the right choice can depend on the problem. Very small λ can require using a very large value of B in order to achieve good performance.
- The number d of splits in each tree, which controls the complexity of the boosted ensemble. Often $d = 1$ works well, in which case each tree is a stump, consisting of a single split. In this case, the boosted ensemble is fitting an additive model, since each term involves only a single variable.
- More generally d is the **interaction depth**, and controls the interaction order of the boosted model, since d splits can involve (at most) d variables.

Identifying important input variables

- The input predictor variables are seldom equally relevant.
- Often a few of them have substantial influence on the response,
- The vast majority are irrelevant and could just as well have not been included.
- Question:
  How to quantify contribution of each input variable in predicting the response?

Relative importance – notation

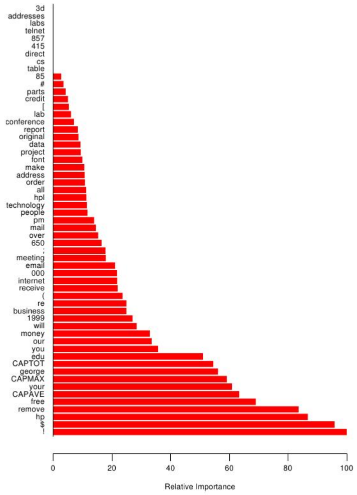
- A single decision tree T.
- Consider the input variable X_l.
- The **internal nodes** of the tree – the ones obtained during growing a tree, and $J-1$ is their number, so that there is J terminal nodes (why?[1]).
- For each $t = 1, \ldots, J-1$, let $v(t)$ be the index of splitting variable used at the t-th node, i.e. $X_{v(t)}$ is the splitting variable.
- Let $\hat{i}_t^2$ be the improvement in the **measure of the quality** of the fit (for example the reduction of the least squares) due to the splitting of the region under consideration.
- Define a **measure of relevance** of each predictor X_l:

$$\mathcal{I}_l^2(T) = \sum_{t=1}^{J-1} \hat{i}_t^2 \, \mathbb{I}_{v(t)}(l)$$

[1] think through mathematical induction

Relative importance

- This importance measure is generalized to boosting tree expansions by **averaging relevances** obtained for each of the boosting trees
- Due to the stabilizing effect of averaging, this measure turns out to be more reliable than for a single tree. Since these measures are relative, it is customary to assign to the most influential predictor the value of 100 and then scale the others accordingly.
- Figure shows the relevant importance of the 57 inputs in predicting spam versus email.


3d
addresses
labs
telnet
857
415
direct
cs
table
85
#
parts
credit
[
lab
conference
report
original
data
project
font
make
address
order
all
hpl
technology
people
pm
mail
over
650
;
meeting
email
000
internet
receive
(
re
business
1999
will
money
our
you
edu
CAPTOT
george
CAPMAX
your
CAPAVE
free
remove
hp
$
!
0
20
40
60
80
100
Relative Importance