Based on Notes of Rutgers CS536, taught by **Charles Wes Cowan**

Consider the problem of classification. We define the true error of a hypothesis and the sample error of a hypothesis (on a dataset S) as

$err(f) = P(f(\underline{X}) \neq Y)$

$err_S(f) = \frac{1}{|S|} \sum_{(\underline{x},y)\in S} I\{f(\underline{x}) \neq y\}$

The general problem is to take a set of the data S and generate a hypothesis f, such as to achieve the minimal true error.

To deal with a collection of models, $f_1, \ldots, f_T$, use the idea of voting.

$F(\underline{x}) = majority(f_1(\underline{x}), \ldots, f_T(\underline{x}))$

Consider the idea of **Bagging** - a representative subsample of the data is taken, and a model trained on that subsample, and this is repeated until sufficiently many models are generated. This is a highly parallelizable approach - non of the models will affect others.

Another heuristic is to use a sequential approach - each new model tries to make up for the weaknesses of the previous. If some model misclassifies some points, the next model can be trained to put more weight on these points. This is the idea behind **Boosting**

Adaptive Boosting (AdaBoost):

We specifically consider AdaBoost in the case of Binary Classification

AdaBoost aims to build classifiers of the form $F(x) = sign(\sum_{t=1}^{T} \alpha_t f_t(x))$

Denote $Q_t(x) = \alpha_1 f_1(x) + \alpha_2 f_2(x) + \ldots + \alpha_t f_t(x)$.

Denote the 'exponential error' at time t to be $E_t = \sum_{(x,y)\in S} e^{-yQ_t(x)}$

We can easily understand that $\sum_{(x,y)\in S} I\{sign(Q_t(s) \neq y\}) \leq E_t$, and our target is to minimize

$E_{t+1} = \sum_{(x,y)\in S} e^{-yQ_{t+1}(x)} = \sum_{(x,y)\in S} e^{-yQ_t(x)} e^{-y\alpha f_{t+1}(x)}$

If the weak models are restricted to only output ± 1, for any point, $yf_{t+1}(x) = 1$ if f_{t+1} correctly classifies, -1 otherwise. We can write the error as this form.

$E_{t+1} = [\sum_{(x,y)\in S: f_{t+1}(x)=y} e^{-yQ_t(x)}]e^{-\alpha} + [\sum_{(x,y)\in S: f_{t+1}(x)\neq y} e^{-yQ_t(x)}]e^{\alpha}$

We introduce the weight of correctly classified points and incorrectly classified points:

$B_{t+1} = \sum_{(x,y)\in S: f_{t+1}(x)\neq y} e^{-yQ_t(x)}$

$G_{t+1} = \sum_{(x,y)\in S: f_{t+1}(x)= y} e^{-yQ_t(x)}$

and $E_{t+1} = G_{t+1}e^{-\alpha} + B_{t+1}e^{\alpha}$

By using the Cauchy-Swartz Inequality, we can achieve the minimum by set $\alpha_{t+1} = \frac{1}{2}ln(\frac{G_{t+1}}{B_{t+1}})$, and optimal value is $E_{t+1} = 2\sqrt{G_{t+1}B_{t+1}}$

$E_{t+1} = 2\sqrt{(E_t - B_{t+1})B_{t+1}} = 2E_t\sqrt{(1-\frac{B_{t+1}}{E_t})(\frac{B_{t+1}}{E_t})}$

Once more, apply the Cauchy-Swartz Inequality, the equation is optimized when $B_{t+1}/E_t = 1/2$.

To minimize the error at time $t+1$, we need to choose f_{t+1} to minimize B_{t+1}/E_t, or equally, to solve

$f_{t+1} = argmin_f \frac{\sum_{(x,y)\in S} e^{-yQ_t(x)} I(f(x)\neq y)}{\sum_{(x,y)\in S} e^{-yQ_t(x)}}$

Given a weighting function $w = e^{-yQ_t(x)}$, the weighted sample error of f is given by

$err_{w,S}(f) = \frac{\sum_{(x,y)\in S} I\{f(x)\neq y\}w(x)}{\sum_{(x,y)\in S} e^{-yQ_t(x)}}$

The problem can be transferred to $\alpha_{t+1} = \frac{1}{2}log(\frac{1-err_{w,S}(f_{t+1})}{err_{w,S(f_{t+1})}})$

The overall exponential error can be expressed as
$E_{t+1} = 2E_t\sqrt{(1-err_{w,S}(f_{t+1}))err_{w,S}(f_{t+1})}$

The whole algorithm can be described as following

- Initialize the weights with $w_1(x) = 1$ for all $(x,y) \in S$
- At time t, find f_t to minimize $err_{w_t,S}(f)$
- For f_t, define the weight for this classifier as $\alpha_t = \frac{1}{2}ln(\frac{1-err_{w_t,S}(f_t)}{err_{w_t,S}(f_t)})$
- Update the weights on the data
  $w_{t+1}(x) = e^{-yQ_t(x)} = e^{-yQ_{t-1}(x)}e^{-y\alpha_t f_t(x)} = w_t(x)e^{-y\alpha_t f_t(x)}$
- Iterate this to generate models $f_1, \ldots, f_T$ and corresponding $\alpha_1, \ldots, \alpha_T$
- Construct the final model: $F(x) = sign(\sum_{t=1}^{T} \times \alpha_t f_t(x))$

Gradient Boosting

Given a model F and a data set, suppose the total loss or error is given by $E = \sum_{i=1}^{m} L(y^i, F(\underline{x}^i))$, where $L(y, f(x))$ is a generic loss function comparing the true value y to the predicted value $f(x)$.

Suppose that all the predicted $F(\underline{x}^i)$ values summarized into a single m-component vector $\underline{F}$, and we have $E(\underline{F}) = \sum_i L(y^i, F_i)$, our target is to construct a new vector $\underline{F}'$ with a smaller value of E based on a vector $\underline{F}$ and a function $E(\underline{F})$.

Take the gradient descent as heuristic. $\underline{F}' = \underline{F} - \lambda \nabla_{\underline{F}} E(\underline{F})$

This kind of updating would potentially allow us to make new predictions for each $\underline{x}^i$ value that reduced the overall error, and eventually reach a minimum of error.

However, immediately a problem came: specifying how to change individual predictions of the model doesn't specify how to change (improve) the model itself. So to deal with this problem, Gradient Boosting's solution is to change the objective function.

Compute the gradient of the error with respect to the predictions, and use these as the *training data for another model*. For any i, define $r^i = [\nabla_{\underline{F}} E(\underline{F})]_i = \frac{\alpha}{\alpha F_i} L(y^i, F_i)$.

Take the data set $\{(\underline{x}_i, r^i)\}_{i=1,\ldots,m}$ and fit a new model f to these 'pseudo-residuals', so that $\underline{f} = [f(x^1). f(x^2), \ldots, f(x^m)] \approx \nabla_{\underline{F}} E(\underline{F})$

Using the function f to approximate the gradient, we can ask how far to move in that direction, how far to tweak the individual predictions of F to try to reduce the overall error.

Define the one-dimensional optimization problem:

$\lambda^* = argmin_\lambda E(\underline{F} + \lambda \underline{f}) = argmin_\lambda \sum_{i=1}^m L(y^i, F(\underline{x}^i) + \lambda f(\underline{x}^i))$

We can construct a new model based on $F_{new}(\underline{x}) + \lambda^* f(\underline{x})$

The Gradient Boosting Algorithm:

- Train a model f_1 on the data to minimize $\sum_i L(y^i, f_1(\underline{x}^i))$, take $\lambda_1 = 1$
- At any time $t \geq 1$, with $F_t = \lambda_1 f_1 + \ldots + \lambda_t f_t$
    - Define the pseudo-residuals for each $\underline{x}^i, y^i$: $r_t^i = \frac{\alpha}{\alpha F_t(\underline{x}_i)} L(y^i, F_t(\underline{x}^i))$
    - Fit a model f_{t+1} to the data $\{(\underline{x}^i, r_t^i)\}_{i=1,\ldots,m}$
    - Solve or estimate λ_{t+1} as the solution to $min_\lambda \sum_{i+1}^m L(y^i, F_t(\underline{x}^i) + \lambda f_{t+1}(\underline{x}^i))$
    - Define the new model $F_{t+1} = F_t + \lambda_{t+1} f_{t+1}$

Comments

Adaptive Boosting and Gradient Boosting are both boosting algorithms, which means they convert a set of weak learners into a single strong learner. They both initialize a strong learner and iteratively create a weak learner that is added to the strong learner. They differ on **how they create the weak learners** during the iterative process.

At each iteration, adaptive boosting **changes the sample distribution** by modifying the weights attached to each of the instances. It increases the weights of the wrongly predicted instances and decreases the ones of the correctly predicted instances. The weak learner thus focuses more on the difficult instances. After being trained, the weak learner is added to the strong one **according to his performance**. The higher it performs, the more it contributes to the strong learner.

As an contrast, gradient boosting doesn't modify the sample distribution. Instead of training on a newly sample distribution, the weak learner trains on the remaining errors (so-called pseudo-residuals) of the strong learner. It is another way to give more importance to the difficult instances. At each iteration, the pseudo-residuals are computed and a weak learner is fitted to these pseudo-residuals. Then, the contribution of the weak learner (so-called multiplier) to the strong one isn't computed according to his performance on the newly distribution sample but using a **gradient descent optimization process**. The computed contribution is the one minimizing the overall error of the strong learner.