Blending and Bagging

JrPhy

Introduction

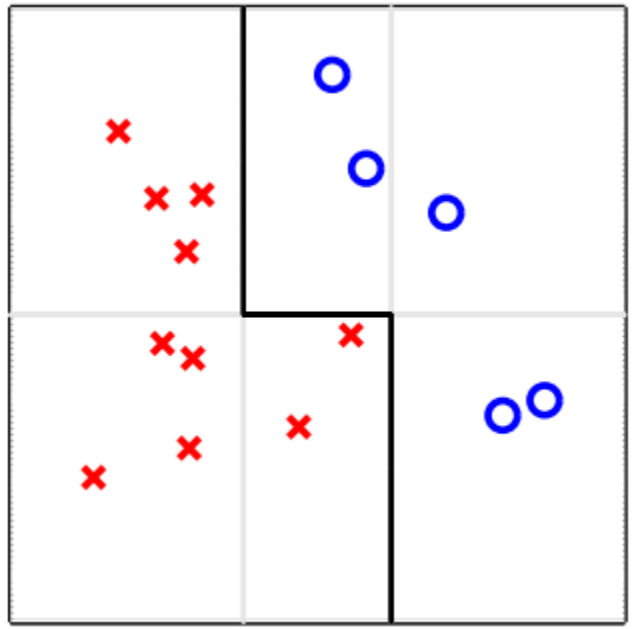
- Suppose you separate your data into k part uniformly, then you get k models, so how to choose the model ?
- One is use the validation, you will get a minimum $E_{in} + E_{out}$.
- Another method is voting, then blend the result, so maybe all model are used. It’s called “Aggregation”.
- This process can use the strong feature transform and regularize by the voting, both advantages are reserved.

Uniform blending

- Suppose each voting are with the same weight, and if we just do the binary classification, then

$$G(x) = sign\left(\sum_{i=1}^{N} g_i(x)\right)$$

- Where $g_i(x)$ is the model you get, $G(x)$ is the blending result.
- $g_i(x)$ should be different from each others so that the advantage of each model will be reserved.



Hsuan Tien Lin, mltech/207_handout

Square error in uniform blending

- Suppose the target is $f(x)$, $E_{out} = (G(x) - f(x))^2$, and we want to show
  - **$\text{avg}(g_i(x) - f(x))^2 \geq (G(x) - f(x))^2$**
- *Proof* :
- $\text{avg}((g_i(x) - f(x)))^2 = \text{avg}(g_i^2(x) - 2g_i(x) f(x) + f^2(x))$
- $= \text{avg}(g_i^2(x)) - f^2(x) + 2G(x) f(x)$
- $= \text{avg}(g_i^2(x)) - G^2(x) + (G(x) - f(x))^2$
- $= \text{avg}(g_i^2(x)) - 2G^2(x) + (G(x) - f(x))^2 + G^2(x)$
- $= \text{avg}(g_i^2(x) - 2g_i(x)G(x) + G^2(x)) + (G(x) - f(x))^2$
- $= \text{avg}((g_i(x) - G(x))^2) + (G(x) - f(x))^2$
- $\geq (G(x) - f(x))^2$

Performance of Uniform blending

- So that we use the ensemble average of the model, the average E_{out} is smaller than each error of one model.
- Suppose the model from the same data, but may be split into many parts, the models follow the same distribution. If the more models are in the ensemble, then $$\bar{g} = \lim_{N\to\infty} G = \lim_{N\to\infty} \frac{1}{N}\sum_{i=1}^{N} g_i(x)$$
- By the previous derivation, we can say $(G(x) - f(x))^2$ is the bias and $\text{avg}((g_i(x) - G(x))^2)$ is the variance. So the process is to reduce the variance.

Linear blending

- Now the weighting coefficient is different, so

$$G(x) = sign\left(\sum_{i=1}^{N} w_i g_i(x)\right) \quad w_i \geq 0$$

- The square error is $$error = \frac{1}{N}\sum_{j=1}^{N}\left(y_j - \left(\sum_{i=1}^{N} w_i g_i(x)\right)\right)^2$$
- Here $g(x)$ is like the kernel function in the SVM, so the blending can be seen as
- Linear blending = Linear model + hypothesis transform + constrains

How about $w_i < 0$?

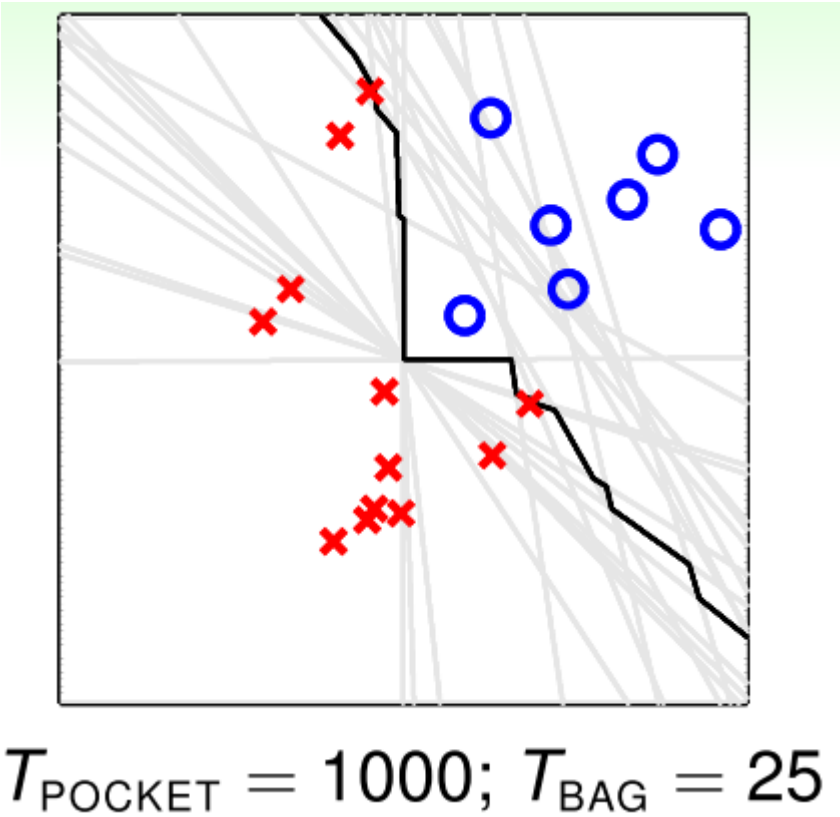
- In SVM, it's always non-negative, here we choose the absolute value and the negative sign goes to $-g_i(x)$.
- If you are doing binary classification, it means the answer is wrong. But it doesn't matter, because the other is correct, so that the constrain in neglected in pratical.
- Linear blending = Linear model + hypothesis transform

In practice

- You'll get many models g_i in the sub-dataset, and take E_{in} as criterion to get the $G(x)$, but the complexity is very high and it may over-fit.
- It recommends to use the training model g_i^- and the $E_{validation}$, g_i^- is from the cross-validation, then transform the model by $\Phi^-(x) = z$, $\Phi^-(x) = \{g_1^-(x), \ldots, g_n^-(x)\}$, calculate $w_n = Linear(\{(z_n, y_n)\})$ and return $G_{linear}(x) = LinearH(<w, z>)$
- There is also the nonlinear blending, it's also called stacking, but it's very powerful than linear, so care for the over-fitting.

BAGging

- By the derivation previous, $g_i(x)$ should be more so that we can reduce the error, but it's hard to realize, so number of $g_i(x)$ is finite and the data is from the same dataset.
- One of the method is bootstrap, it chooses some data D' in D to form a new dataset, and sampling with replacement, so some data may be chosen many times and some may not be chosen.
- It's mechanism uses bootstrap and aggregation.
- The black line is the result of aggregation.



$T_{\text{POCKET}} = 1000;\ T_{\text{BAG}} = 25$

Hsuan Tien Lin, mltech/207_handout

Re-weighting

- Because of bootstrapping, the weighting of each data may be different, so we should time the weight. The original error is

$$E_{in}(h) = \frac{1}{N}\sum_{i=1}^{N} err(y \neq h(x))$$

- Times its weight

$$\tilde{E}_{in}(h) = \frac{1}{N}\sum_{i=1}^{N} u_i err(y \neq h(x))$$

- So that we want to minimize it.
- It's like the SVM that the constrain $0 \leq a \leq Cu_i$, or logistic regression with constrain with the probability with $u_{i.}$

Re-weighting

- So that we want the diversity g_j , that means g_j and g_{j+1} should be very different

$$g_i \leftarrow \arg\min_{h \in H} \left(\sum_{j=1}^{N} u_j^i [y \neq h(x)] \right) \quad g_{i+1} \leftarrow \arg\min_{h \in H} \left(\sum_{j=1}^{N} u_j^{i+1} [y \neq h(x)] \right)$$

- For binary classification, the expectation value is close to 0.5, that means if g_j is close to 1, then g_{j+1} is close to 0.
- There is a method that uses g_j to update g_{j+1} , so it will get a bad result.

$$g_{j+1} \leftarrow \arg\min_{h \in H} \left(\sum_{j=1}^{N} u_j^{i+1} [y \neq g_j(x)] \right)$$

Optimize

- So what we want is

$$\frac{\sum_{j=1}^{N} u_j^{i+1}[y \neq g_j(x)]}{\sum_{j=1}^{N} u_j^{i+1}} = \frac{\sum_{j=1}^{N} u_j^{i+1}[y \neq g_j(x)]}{\sum_{j=1}^{N} u_j^{i+1}[y \neq g_j(x)] + \sum_{j=1}^{N} u_j^{i+1}[y = g_j(x)]} = \frac{true_{j+1}}{true_{j+1} + false_{j+1}} = \frac{1}{2}$$

- That means total true = total false. Suppose the number of true is n_1, number of true is n_2, then uses the “ensemble average”, number of true times number of false, and number of false times number of true, then

$$\frac{n_1 n_2}{n_1 n_2 + n_1 n_2} = \frac{1}{2}$$

Optimize

- In general, suppose the true ratio is $t = \alpha$, then the false ratio is $1 - t = \beta$,

$$\alpha(1-t) = \beta t \rightarrow \frac{\alpha(1-t)}{\sqrt{t(1-t)}} = \frac{\beta t}{\sqrt{t(1-t)}} \rightarrow \alpha\sqrt{\frac{(1-t)}{t}} = \beta\sqrt{\frac{t}{(1-t)}} \qquad r = \sqrt{\frac{(1-t)}{t}}$$

- Then $\alpha\, r = \beta / r$. if $t \leq 0.5$, then $r \geq 1$, it means the algorithm scales up the false and scale down the true, and learn from the false.

Algorithm

$\mathbf{u}^{(1)} = ?$

for $t = 1, 2, \ldots, T$

1. obtain g_t by $\mathcal{A}(\mathcal{D}, \mathbf{u}^{(t)})$,
   where $\mathcal{A}$ tries to minimize $\mathbf{u}^{(t)}$-weighted 0/1 error
2. update $\mathbf{u}^{(t)}$ to $\mathbf{u}^{(t+1)}$ by $\blacklozenge_t = \sqrt{\frac{1-\epsilon_t}{\epsilon_t}}$,
   where ϵ_t = weighted error (incorrect) rate of g_t

return $G(\mathbf{x}) = ?$

Hsuan Tien Lin, mltech/208_handout

- Because r is bigger than 1, so a better choice of u_j^1 is $1 / N$
- If $E_{in}(g_j)$ is good, then $E_{in}(g_{j+1})$ is bad by previous discussion, so it's not suggested use uniform blendeing.

Linear Aggregation on the Fly

- This method uses linear blending when calculating g_j,

$\mathbf{u}^{(1)} = [\frac{1}{N}, \frac{1}{N}, \cdots, \frac{1}{N}]$

for $t = 1, 2, \ldots, T$

1. obtain g_t by $\mathcal{A}(\mathcal{D}, \mathbf{u}^{(t)})$, where ...
2. update $\mathbf{u}^{(t)}$ to $\mathbf{u}^{(t+1)}$ by $\blacklozenge_t = \sqrt{\frac{1-\epsilon_t}{\epsilon_t}}$, where ...
3. compute $\alpha_t = \ln(\blacklozenge_t)$

return $G(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t g_t(\mathbf{x})\right)$

Hsuan Tien Lin, mltech/208_handout

AdaBoost

- For a good g_j, times the bigger α, where α is the coefficient of linear model.
- A method is choosing $\alpha = \ln\sqrt{\frac{(1-t)}{t}}$, then it grows as g_j grows.
  - If $t = 0.5$, then $\alpha = 0$ → the worst with no weight.
  - If $t = 0$, then $\alpha = \infty$ → the strongest with all weight.
- It's called adaptive boosting algorithm, so-called **AdaBoost**, it needs a weak base learning algorithm + optimal re-weighting factor + linear aggregation α.
- The algorithm just better than random, by **AdaBoost**, the E_{in} → 0 and E_{out} is small.

- There are more examples in the **Hsuan Tien Lin, mltech/208_handout**