EECS 445: Machine Learning **Fall 2016**

Discussion 6: Ensemble Methods

Written by: Chansoo Lee

Edited by:

Last Updated: November 2, 2016 6:32pm

6.1 Important Concepts

True vs Empirical Risk Simply put, *risk* is the *expected loss.* If we take the expectation over true data distribution, we get true risk. If we take the expectation over *empirical probability density estimate* (which is uniform over a finite data, HW3 problem 2), we get empirical risk.

Now, let us define these formally. Suppose the data is i.i.d. samples from a distribution Q. Then, the *true risk* (or simply *risk*) of hypothesis h is the expectation of the loss function ℓ over Q:

$$R(h) = \mathbb{E}_{(\mathbf{x},y)\sim Q}[\ell(h(\mathbf{x}), y)].$$

We have seen a very similar expression in homework 3.

The true risk cannot be computed, because Q is unknown. The learning algorithm, however, can approximate the true risk using training examples $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$:

$$R_S(h) = \frac{1}{n}\sum_{i=1}^{n} \ell(h(\mathbf{x}_i), y_i).$$

As n goes to infinity, the empirical risk converges to the true risk.

For example, consider linear regression. The weight vector $\mathbf{w}$ is the model parameter, and the prediction function $h(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$ is the associated hypothesis. Define our loss function as $\ell(h(\mathbf{x}), y) = (h(\mathbf{x}) - y)^2$. The unregularized linear regression corresponds to *empirical risk minimization* where your *hypothesis class* is restricted to linear functions.

Bagging vs Boosting Bagging focuses on aggregating outputs of multiple hypotheses independently trained on different (sub)samples of train data.

Boosting is more holistic approach, which sequentially chooses the weights on hypotheses and the weights on train examples.

Excercise 6.1. Can each hypothesis in the ensemble be trained in a parallel manner for bagging? How about for boosting?

Answer: Bagging can be, but boosting cannot be due to its sequential nature. ■

Excercise 6.2. With access to distributed system, is bagging always computationally superior to boosting?

Answer: Bagging reduces the variance only; its performance is essentially the average performance of the hypotheses in the ensemble. Each hypothesis in a boosted ensemble, on the other hand, needs to only satisfy weak learning hypothesis, which usually takes only a trivial amount of computing power. ■

6.2 AdaBoost

Let $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ be the training set of examples. Let WL be a γ-weak learning algorithm that takes a distribution over input space (as a tuple of weights and samples) and returns a hypothesis (classifier) f_t with error at most $\frac{1}{2} - \gamma$. Let T be the number of iterations.

Algorithm 1: AdaBoost

```
Input: $S, \mathrm{WL}, T$ as defined in the text
initialize $\mathbf{w}^1 = (1, \ldots, 1)$
for $t = 1, \ldots, T$ do
    normalize $\mathbf{w}^t$
    invoke weak learner $f_t = \mathrm{WL}(\mathbf{w}^t, S)$
    compute $r_t = \sum_{i=1}^n w_i^t \mathbf{1}[y_i \neq f_t(\mathbf{x}_i)]$
    let $\alpha_t = \dfrac{1}{2}\log(\dfrac{1}{r_t} - 1)$
    update $w_i^{t+1} = w_i^t \exp(-\alpha_t y_i f_t(\mathbf{x}_i))$
end
Output: Return $F_T = \sum_{t=1}^T \alpha_t f_t$
```

Theorem 6.1. *The training error (empirical risk) of AdaBoost satisfies:*

$$R_S(F_T) := \frac{1}{n}\sum_{i=1}^n \mathbf{1}[F_T(\mathbf{x}_i) \neq y_i] \leq \exp(-2\gamma^2 T).$$

Proof: For each t, let

$$F_t = \textstyle\sum_{s=1}^t \alpha_s f_s$$

and let z_t be the normalizing constant for $\mathbf{w}^t$, which is (by inductive argument)

$$z_t = \frac{1}{n}\sum_{i=1}^n \exp(-y_i F_t(\mathbf{x}_i)).$$

(Optional paragraph.) The proof is left for exercise. Roughly speaking, we update w_i in a multiplicative manner, adding $-\alpha_t y_i f_t(\mathbf{x}_i)$ to the exponent every round. So at round t, we have $\exp(-\sum_{s=1}^t \alpha_s y_i f_s(\mathbf{x}_i)) = \exp(-y_i F_s(\mathbf{x}_i))$. The base case for $t = 1$ gives the $(1/n)$.

Note that the exponential function acts as a *smooth* upper bound on the loss of a hypothesis h, as $\mathbf{1}[h(\mathbf{x}) \neq y] \leq \exp(-yh(\mathbf{x}))$. So, $R_S(F_T) \leq z_T$ and it suffices to show that $z_T \leq \exp(-2\gamma^2 T)$. In other words, our normalization constant z_t coincides with an upper bound on the empirical risk of F_t.

Rewrite

$$z_T = \frac{z_T}{z_{T-1}} \cdots \frac{z_2}{z_1}.$$

It suffices to show that for every t,

$$\frac{z_{t+1}}{z_t} \leq \exp(-2\gamma^2)$$

This means that the *upper bound* on the empirical risk decreases in a multiplicative manner (fraction of mistakes fixed as opposed to a fixed number of mistakes fixed) every round. The actual empirical risk, however, might oscillate. Higher γ means faster decay in the upper bound, as it should.

The rest of the proof is optional: Now,

$$\begin{aligned}
\frac{z_{t+1}}{z_t} &= \frac{\sum_{i=1}^n \exp(-y_i F_{t+1}(\mathbf{x}_i))}{z_t} \\
&= \frac{\sum_{i=1}^n \exp(-y_i F_t(\mathbf{x}_i)) \exp(-y_i \alpha_{t+1} f_{t+1}(\mathbf{x}_i))}{z_t} \\
&= \sum_{i=1}^n w_i^{t+1} \exp(-y_i \alpha_{t+1} f_{t+1}(\mathbf{x}_i)) \\
&= \exp(-\alpha_{t+1})(1 - r_{t+1}) + \exp(\alpha_{t+1}) r_{t+1} \\
&= \frac{1}{\sqrt{1/r_{t+1} - 1}}(1 - r_{t+1}) + \sqrt{1/r_{t+1} - 1}\; r_{t+1} \\
&= 2\sqrt{r_{t+1}(1 - r_{t+1})} \le 2\sqrt{\left(\frac{1}{2} - \gamma\right)\left(\frac{1}{2} + \gamma\right)} = \sqrt{1 - 4\gamma^2}.
\end{aligned}$$

To complete the proof, use the inequality $1 - a \le e^{-a}$. ■

Generally, minimizing a all-or-nothing (0-or-1 or 0-or-infinity) loss function is *computationally* hard in the sense of NP-hardness. Hence, we use a *surrogate loss* function that is continuous and convex. The *hinge loss* for soft-SVM is one such example which converts a 0-or-∞ loss function to a linear-and-flat function. Similarly, AdaBoost can be considered an algorithm that minimizes the exponential loss function. The key difference is that the exponential loss directly upper bounds the classification error, whereas the hinge loss does not.

Excercise 6.3. Explain the semantics of the variables used in AdaBoost: $\mathbf{w}^t$, f_t, r_t, α_t and F_t.

Excercise 6.4. Answer the following questions:

- Convince yourself that r_t is the risk of f_t when the true distribution is $\mathbf{w}^t$.

$$r_t = \mathbb{E}_{i \sim \mathbf{w}^t}\left[\mathbf{1}[f_t(\mathbf{x}_i) \neq y_i]\right].$$

- Is α_t an increasing or decreasing function of r_t? Why does it make sense, considering how we construct the final output F_T?

  Answer: Decreasing. f_t that had higher risk should contribute less to the final output F_T. ■

- The update rule for w_i shows that the amount of change in w_i^t is increasing or decreasing in α_t? Why does it make sense?

  Answer: Increasing. The weight w_i should reflect how difficult it is to classify the i-th example correctly, so we can focus on the difficult examples. If α_t is large, then f_t performed well overall and thus it is a "trustworthy" hypothesis. Its mistakes should affect our opinion on how difficult each example is, more so than a non-trustworthy hypothesis (low α_t) would. ■

Excercise 6.5. Does AdaBoost guarantee a zero test error?

Answer: No. AdaBoost guarantees that if T is sufficently large, then the *empirical risk* ultimately becomes 0. But it says nothing about the test error aka true risk. ■

6.3 Generalized Boosting (Optional)

6.3.1 Coordinate Descent

Coordinate descent is a variation of gradient descent, where you change a single coordinate per iteration. For example, suppose we want to minimize

$$J(a,b) = (a-2)^2(b+1)^2$$

Fix the learning rate $\eta = .1$. Initialize (arbitrarily) $a = b = 0$. In the first round, we change a. So, we look at the partial derivative

$$\frac{\partial J}{\partial a} = 2(a-2)(b+1)^2$$

which we evaluate at $a = b = 0$ and get -4. Now we perform the coordinate descent update and get $a = .4, b = 0$. We repeat the process for b:

$$\frac{\partial J}{\partial b} = 2(a-2)^2(b+1)$$

which we evaluate at the current point and get 5.12. So we perform the update and get $a = .4, b = -.512$, we repeat this process, and eventually converge to $a = 2, b = -1$.

6.3.2 Boosting as Coordinate Descent

We observe that boosting is a slight variation of coordinate descent on a function of T variables, where we do a single sweep through each variable. We use the notation $\hat{\ell}$ (in lecture notes, its ϕ) to denote the surrogate loss function. Assume that $\partial\ell/\partial\alpha_t$ is always negative. We first define the objective function

$$J(\alpha_1,\dots,\alpha_T) = \frac{1}{n}\sum_{i=1}^{n}\hat{\ell}\left(y_i\sum_{t=1}^{T}\alpha_t f_t(\mathbf{x}_i)\right) = \frac{1}{n}\sum_{i=1}^{n}\hat{\ell}\left(y_i\sum_{t=1}^{T}F_t(\mathbf{x}_i)\right).$$

We initialize $\alpha_t = 0$ for all t, and we do coordinate descent. At t-th iteration, we have

$$J(\alpha_1,\dots,\alpha_T) = \frac{1}{n}\sum_{i=1}^{n}\hat{\ell}\left(y_i\sum_{s=1}^{t-1}\alpha_s f_s + y_i\alpha_t f_t(\mathbf{x}_i)\right),$$

where $\alpha_{t+1},\dots,\alpha_T$ terms are ommitted because they are still 0.

Hence, we update α_t using the partial derivative:

$$\frac{\partial J}{\partial \alpha_t} = \frac{1}{n}\sum_{i=1}^{n}\hat{\ell}'_t(y_iF_{t-1}(\mathbf{x}_i) + \alpha_t y_i f_t(\mathbf{x}_i))y_i f_t(\mathbf{x}_i)$$

where $\hat{\ell}'_t = \frac{\partial\hat{\ell}}{\partial\alpha_t}$ as a shorthand notation.

But we also have to choose f_t which is not a number but a hypothesis. A smart way of choosing f_t is such that the derivative is the largest, so we get the steepest coordinate descent update; that is, we choose f_t to maximize the (directional) derivative:

$$\arg\max_{f_t}\frac{\partial J_t}{\partial\alpha_t}\Big|_{\alpha_t=0} = \arg\max_{f_t}\frac{1}{n}\sum_{i=1}^{n}\hat{\ell}(y_iF_{t-1})y_if_t = \arg\min_{f_t}\frac{1}{n}\sum_{i=1}^{n}\frac{\hat{\ell}'_t(y_iF_{t-1})}{\sum_{i=1}^{n}\hat{\ell}'_t(y_iF_{t-1})}y_if_t.$$

The last equality is because ℓ' is always negative. We cannot compute the exact minimizer (if that is the case, we don't even need boosting!). Instead, we have an algorithm WL which will "try" to minimize the above and return a weak learning hypothesis f_t.

Once we fix f_t, it turns out that we can be very smart with how we update α_t, instead of doing the standard coordinate descent update. In particular, we update α_t to be the minimizer:

$$\alpha_t = \arg\min_{\alpha} J(\alpha_1, \ldots, \alpha_{t-1}, \alpha).$$