COS424 Scribe Notes
Lecture 14: Ensembles

Donghun Lee

April 8, 2010

1 Ensembles

A set of classifiers can combine their outputs to make an ensemble of the classifiers. Two requirements of such an ensemble to work:

- accuracy: classifiers should have some predictive power in their own.
- diversity: see the example below.

Slide 6: Example

21 classifiers with 30% error rate each ("accuracy"). Now, for "diversity" requirement, we assume that the classifiers are uncorrelated. For the example ensemble (majority vote of 21) to make an error, at least 11 classifiers need to make the same error, and the probability for that event to happen is ¡ 2%, much smaller than 30%. This collective strength comes from assuming that 21 classifiers are uncorrelated.

Slide 7-10: Motivations for ensembles

- Statistical: best classifier may be within a smaller set that contains good classifiers that we trained.
- Computational: combining classifiers may find a better optimum if our method of computation has local optima guarantee only.
- Representational: combination of classifiers from model space H may be able to represent classifier outside space H.
- Practical success: it seems to work.

Slide 11: Bayesian ensemble

Bayesian ensemble shows a canonical formulation of an ensemble – weighted average. Often algoritms are approximation of this theoretical form.

2 Combining Outputs

- Simple averaging works well if all classifiers are uncorrelated. Simple and effective.
- Weighted averaging a priori approximates Bayesian ensemble formulation, because its weights are approximating $P(h|D)$ (see slide 11) with some loss function.
- Weighted averaging with trained weights requires total two *validation set* – one for training weights and the other to measure the performace of the ensemble itself – to avoid overfitting.
- Stacked classifiers is a generalization of "Weighted averaging with trained weights" method. It also requires two validation sets.

As a sidenote, needing two validation sets can be beneficial in some contests, where more than one validation sets are provided (but the examples in those sets cannot be used in the training set).

3 Constructing Ensembles

An ensemble may not work if the pattern is innately too difficult for its member classifiers to learn. Also, there are situations that can be handled by constructing ensembles:

- Dealing with overfitting: vary training sets with techniques like bootstrapping or bagging. (Typo on slide 19: "Bootstrap", not Boostrap)
- Dealing with noisy features: ensemble methods using feature subset selection (random forest) or preprocessing the inputs (multiband speech recognition). Both works well in practice.
- Dealing with multiclass problems: To resist corruption in multiclass data class labels, encode class label data into binaries ("Error Correcting Code: ECC"), and traing a set of classifiers, where each of which will predict the bits of ECC. Because of ECC, the class labels can be reconstructed easily.

4 Boosting

The motivation of Boosting technique is: an ensemble of weak classifier makes stronger classifier. An example of boosting algorithm, called Adaboost (Freund and Schapire 1995), is analyzed below.

Slide 24-29: How it works

Adaboost algorithm is on Slide 24. Magic terms are $D_t(i)$ (example i's weight for classifier t) and α_t (ensembling weight for classifier t). Note that class labels are 1 or -1, so $y_i h_t(x_i) = -1$ only when classifier h_t makes error in predicting y_i. Z_t is a normalization factor.

Roughly speaking, $D_t(i)$ is the importance weight for examples. We update this each time. If example i is correctly recognized, then $D_t(i)$ decreases. Otherwise, the $D_t(i)$ will grow. i.e. incorrectly classified examples will *look bigger* when calculating error rate ε_t (see toy example visualization in Slides 25-28).

Slide 30-31: Analysis

There is typo in Slide 30: second line of formula, second term, inside summation sign, should be $e^{-y_i f_T(x_i)}$, not $e - y_i f_T(x_i)$. α_t is the weight of ensembling. The update rule $\alpha_t = \frac{1}{2}\log\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$ can be explained from the following perspective on how Adaboost works in each round:

1. Using weighted examples $D_t(i)\, x_i$ instead of raw examples x_i, train classifier h_t that minimizes the weighted error.
2. Update α_t with a value that minimizes the weighted sum of exponential losses of all classifiers.

Recall that Z_t is a normalization factor (of D_{t+1}, precisely), and also Z_t is in fact the D_t-weighted sum of exponential loss from α_t-weighted classifier t (see the first two lines below):

$$\begin{aligned}
Z_t &= \sum_i D_{t+1}(i) \\
&= \sum_i D_t(i)\, e^{-\alpha_t y_i h_t(x_i)} \\
&= \sum_i \left[D_t(i)\, \mathbb{I}(h_t(x_i) \neq y_i)\, e^{-\alpha_\cdot - 1} + (1 - \mathbb{I}(h_t(x_i) \neq y_i))\, e^{-\alpha_\cdot + 1}\right] \qquad (1) \\
&= \varepsilon_t e^{\alpha_t} - (1-\varepsilon_t)\, e^{-\alpha_t} \qquad (2)
\end{aligned}$$

Scribe added Equation 1 to make sure that the derivation is easier to follow. The update rule of α_t is formulated to minimize Z_t, i.e.:

$$\begin{aligned}
\alpha_t &= \arg\min_{\alpha_t} Z_t \\
\text{such that} \quad \frac{\partial Z_t}{\partial \alpha_t} &= 0 \\
\varepsilon_t e^{\alpha_t} - (1-\varepsilon_t)\, e^{-\alpha_t} &= 0 \\
e^{2\alpha_t} &= \frac{1-\varepsilon_t}{\varepsilon_t} \\
\alpha_t &= \frac{1}{2}\log\frac{1-\varepsilon_t}{\varepsilon_t} \qquad (3)
\end{aligned}$$

Note that Equation 3 is the update rule of α_t. Scribe verified that $\frac{\partial^2 Z_t}{\partial \alpha_t^2} > 0$ with the update rule α_t, so it is indeed argmin, not argmax.

Slide 31-32: Bounding the training error

Applying this update rule to D_t allows us to upper-bound the training error of the ensemble, which is $\leq \prod Z_t$ (this inequality the scribe could not verify). First, we apply α_t to Z_t in Equation 2:

$$\begin{aligned}
Z_t &= \varepsilon_t \sqrt{\frac{1-\varepsilon_t}{\varepsilon_t}} + (1-\varepsilon_t)\sqrt{\frac{\varepsilon_t}{1-\varepsilon_t}} \\
&= \sqrt{\varepsilon_t(1-\varepsilon_t)} + \sqrt{\varepsilon_t(1-\varepsilon_t)} \\
&= \sqrt{4\varepsilon_t(1-\varepsilon_t)} \\
&= \sqrt{1-4\gamma_t^2} \qquad (4)
\end{aligned}$$

where $\gamma_t = \frac{1}{2} - \varepsilon_t$. Since all weak learners (classifiers) are assumed to have error rate ε_t lower than $\frac{1}{2}$, $\inf \gamma_t > 0$. Now, approximate bound Equation 4:

$$\sqrt{1 - 4\gamma_t^2} \leq 1 - 2\gamma_t^2 \leq e^{-2\gamma_t^2} \tag{5}$$

We'll use the above Inequalities 5 to bound the training error:

$$\text{Training Error} \leq \prod_{t=1}^{T} Z_t = \prod_{t=1}^{T} \sqrt{1 - 4\gamma_t^2} \leq \prod_{t=1}^{T} e^{-2\gamma_t^2} = e^{-2\sum_{t=1}^{T} \gamma_t^2}$$

Since $\inf \gamma_t > 0$, the training error is upper-bounded by an exponentially decreasing function. This suggests the power of Adaboost to make a strong ensemble out of classifiers that are assumed to be weak learners. It also suggests the fact that it gets exponentially harder to reduce training error the same amount. Note that this training error bound is obtained 1) without specific relationship between D_t (weight) and h_t (choice of classifier), and 2) without specific value of α_t (we used the update rule for α_t that is justified from argmin-of-exponential-loss perspective).

Slide 32-33: Conclusion and extension

In conclusion, this shows the general applicability of Adaboost as a meta-algorithm. From algorithmic perspective, Boosting is a greedy optimization of exponential loss.

Note that exponential loss will penalize any misclassifications ("-1") very harshly (black line in the graph on Slide 32). Adaboost variant using log-loss (less harsh penalty) is also available.

The graph on the left side of Slide 33 shows how the training error (lower) and the test error (higher) decreases versus the number of rounds. Note that the test error continues to decrease even after the training error hits 0. What happens during this period is that boosting keep decreasing the margin. From this point of view, boosting can be seen as an algorithm that increases classification margin over repeated loops, where margin is defined as on Slide 33: $\frac{\sum_t \alpha_t y h_t(x)}{\sum_t |\alpha_t|}$. The graph on the right side of Slide 33 is the cumulative distribution of margins, where the dotted line is from the algorithm with less rounds, and the other two are from that with more rounds.

This kind of behavior (the test error keep decreasing after the training error hits 0) may be simulated by using SVM with hard margin repeatedly, where in each repetition keep increasing the margin by tweaking the weight of the examples.