Machine Learning (CSE 446): Ensembles

Noah Smith

University of Washington
nasmith@cs.washington.edu

November 29, 2017

Ensembles of Different Learners

Different learning algorithms tend to make different mistakes.

Ensembles of Different Learners

Different learning algorithms tend to make different mistakes.

One strategy to get better predictions is to train different learning algorithms on the same data, and let them vote.

Ensembles of Different Learners

Different learning algorithms tend to make different mistakes.

One strategy to get better predictions is to train different learning algorithms on the same data, and let them vote.

For a colleciton of K classifiers that predict $y \in \{-1, +1\}$:

$$f_{\text{ensemble}}(x) = \text{sign}\left(\sum_{k=1}^{K} f_k(x)\right)$$

(For regression, take the mean.)

Ensembles of Different Learners

Different learning algorithms tend to make different mistakes.

One strategy to get better predictions is to train different learning algorithms on the same data, and let them vote.

For a colleciton of K classifiers that predict $y \in \{-1, +1\}$:

$$f_{\text{ensemble}}(x) = \text{sign}\left(\sum_{k=1}^{K} f_k(x)\right)$$

(For regression, take the mean.)

You should recall from the voted perceptron that letting a collection of classifiers **vote** can offer a richer decision boundary than any individual classifier offers.

Theoretical View

Ensembles tend to reduce *variance*—if your learning algorithms are sensitive to small changes in the training data, ensembling can help.

Theoretical View

Ensembles tend to reduce *variance*—if your learning algorithms are sensitive to small changes in the training data, ensembling can help.

This leads to a different kind of ensemble: train on different datasets.

Theoretical View

Ensembles tend to reduce *variance*—if your learning algorithms are sensitive to small changes in the training data, ensembling can help.

This leads to a different kind of ensemble: train on different datasets.

Where do we get different datasets?

Data: $D = \langle (x_n, y_n) \rangle_{n=1}^{N}$, number of bootstrap samples required B
Result: Resampled datasets $\tilde{D}_1, \ldots, \tilde{D}_B$
for $b \in \{1, \ldots, B\}$ **do**
    for $n \in \{1, \ldots, N\}$ **do**
        sample i uniformly at random from $\{1, \ldots, N\}$;
        add (x_i, y_i) to $\tilde{D}_b$;
    end
end
return $\tilde{D}_1, \ldots, \tilde{D}_B$;

Algorithm 1: BootstrapResampling

Bootstrap Samples

They will be similar . . . but not *too* similar.

Bootstrap Samples

They will be similar ... but not *too* similar.

Instance n will be missing from a given $\tilde{D}_b$ with probability $(1 - \frac{1}{N})^N$, which tends to $\frac{1}{e} \approx 0.3679$ as N gets large.

Bootstrap Samples

They will be similar ... but not *too* similar.

Instance n will be missing from a given $\tilde{D}_b$ with probability $(1 - \frac{1}{N})^N$, which tends to $\frac{1}{e} \approx 0.3679$ as N gets large.

This means that in any given $\tilde{D}_b$, only about two thirds of the training examples will appear!

Bootstrap Samples

They will be similar …but not *too* similar.

Instance n will be missing from a given $\tilde{D}_b$ with probability $(1 - \frac{1}{N})^N$, which tends to $\frac{1}{e} \approx 0.3679$ as N gets large.

This means that in any given $\tilde{D}_b$, only about two thirds of the training examples will appear!

Why is it called "bootstrap"?

Bagging: Ensembles of the *Same* Learner

1. Apply $\textsc{BootstrapResampling}$ to D, creating datasets $\tilde{D}_1, \ldots, \tilde{D}_B$.
2. Apply your learning algorithm to each $\tilde{D}_b$.
3. Let them vote.

Bagging: Ensembles of the *Same* Learner

1. Apply $\textsc{BootstrapResampling}$ to D, creating datasets $\tilde{D}_1, \ldots, \tilde{D}_B$.
2. Apply your learning algorithm to each $\tilde{D}_b$.
3. Let them vote.

You can think of bagging as a variance-reducing technique; it tends to have similar benefits to regularization.

Boosting

Boosting (not to be confused with bootstrapping, discussed earlier!) is a family of sophisticated techniques for building an ensemble.

Boosting

Boosting (not to be confused with bootstrapping, discussed earlier!) is a family of sophisticated techniques for building an ensemble.

We'll consider one particular method called AdaBoost (for "adaptive boosting") that has some interesting theoretical properties *and* is widely used.

Data: $D = \langle (x_n, y_n) \rangle_{n=1}^{N}$, number of epochs E, weighted learner $\mathcal{W}$
Result: classifier
$\boldsymbol{\beta}^{(0)} = \langle \frac{1}{N}, \frac{1}{N}, \ldots, \frac{1}{N} \rangle$; # initialize example weights
for $e \in \{1, \ldots, E\}$ **do**
    $f^{(e)} \leftarrow \mathcal{W}(D, \boldsymbol{\beta}^{(e-1)})$; # train the classifier on the weighted data
    $\hat{\epsilon}^{(e)} \leftarrow \sum_{n=1}^{N} \beta_n^{(e-1)} \cdot [\![f^{(e)}(x_n) \neq y_n]\!]$; # weighted error rate
    $\alpha^{(e)} \leftarrow \frac{1}{2} \log \left(\frac{1-\hat{\epsilon}^{(e)}}{\hat{\epsilon}^{(e)}} \right)$; # "adaptive" weight for $f^{(e)}$
    for $n \in \{1, \ldots, N\}$ **do**
        $\beta_n^{(e)} \leftarrow \frac{1}{Z^{(e)}} \cdot \beta_n^{(e-1)} \cdot \exp\left(-\alpha^{(e)} \cdot y_n \cdot f^{(e)}(x_n)\right)$; # update example weights
        ($Z^{(e)}$ is a normalization constant)
    end
end
return $f_{\text{boost}}(\cdot) = \text{sign}\left(\sum_{e=1}^{E} \alpha^{(e)} \cdot f^{(e)}(\cdot) \right)$;

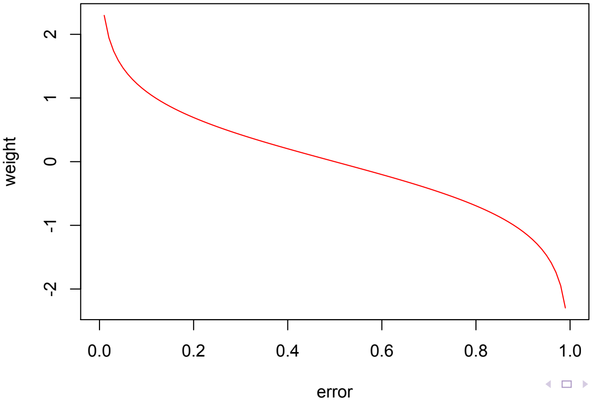
Algorithm 2: AdaBoost

Notes about AdaBoost

- Typically, $\mathcal{W}$ is a shallow decision tree, or a linear classifier. In the literature, it is often called a **weak** learner (definition comes later).

Notes about AdaBoost

- Typically, $\mathcal{W}$ is a shallow decision tree, or a linear classifier. In the literature, it is often called a **weak** learner (definition comes later).
- α as a function of $\hat{\epsilon}$:



Notes about AdaBoost

- Typically, $\mathcal{W}$ is a shallow decision tree, or a linear classifier. In the literature, it is often called a **weak** learner (definition comes later).
- α as a function of $\hat{\epsilon}$: goes to $+\infty$ when error is low, zero as error increases to 0.5.

Notes about AdaBoost

- Typically, $\mathcal{W}$ is a shallow decision tree, or a linear classifier. In the literature, it is often called a **weak** learner (definition comes later).
- α as a function of $\hat{\epsilon}$: goes to $+\infty$ when error is low, zero as error increases to 0.5.
- For examples we get right ($f^{(e)}(x_n) = y_n$), the weight β_n will *decrease*; we increase the weights of examples we get wrong.

Notes about AdaBoost

- Typically, $\mathcal{W}$ is a shallow decision tree, or a linear classifier. In the literature, it is often called a **weak** learner (definition comes later).
- α as a function of $\hat{\epsilon}$: goes to $+\infty$ when error is low, zero as error increases to 0.5.
- For examples we get right ($f^{(e)}(x_n) = y_n$), the weight β_n will *decrease*; we increase the weights of examples we get wrong.
  - See the book for more insight on what happens on the first epoch with a *very* simple $\mathcal{W}$. Each successive $f^{(e)}$ is intended to work harder wherever previous classifiers have been failing (hence, "adaptive").

Formula for $\beta_n^{(e)}$

$$\beta_n^{(e)} \leftarrow \frac{1}{Z^{(e)}} \cdot \beta_n^{(e-1)} \cdot \exp\left(-\alpha^{(e)} \cdot y_n \cdot f^{(e)}(x_n)\right)$$

$$\text{where } Z^{(e)} = \sum_{i=1}^{N} \beta_i^{(e-1)} \cdot \exp\left(-\alpha^{(e)} \cdot y_i \cdot f^{(e)}(x_i)\right)$$

($Z^{(e)}$ forces the sum $\sum_{n=1}^{N} \beta_n^{(e)} = 1$.)