CMPSC 448: Machine Learning

Lecture 12. Ensemble Learning: Bagging and Boosting

Rui Zhang

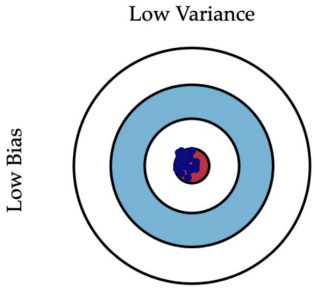
Fall 2021

PennState

Outline

- Revisit Bias-Variance Tradeoff
- Ensemble Learning
- Bagging
  - Bootstrap Resampling
  - Random Forests
- Boosting
  - AdaBoost
  - Gradient Boosting

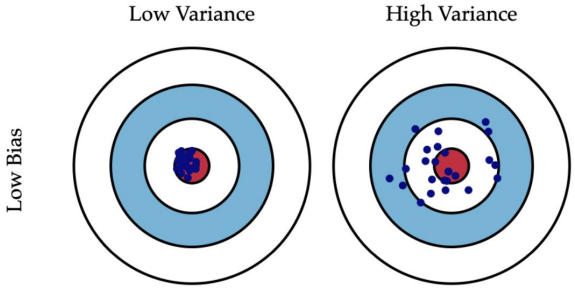
Bias-Variance Tradeoff



The ultimate goal of any learning algorithm is to make accurate predictions on test data or generalize well (low-bias and low-variance)

Source: http://scott.fortmann-roe.com/docs/BiasVariance.html

Bias-Variance Tradeoff

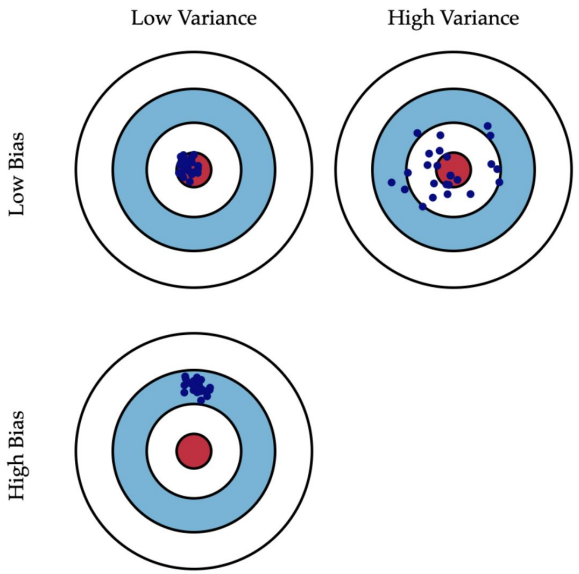


The ultimate goal of any learning algorithm is to make accurate predictions on test data or generalize well (low-bias and low-variance)

Source: http://scott.fortmann-roe.com/docs/BiasVariance.html

Bias-Variance Tradeoff

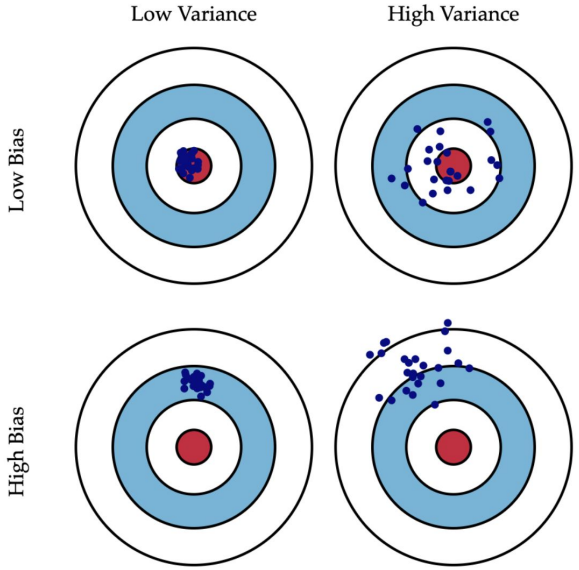
The ultimate goal of any learning algorithm is to make accurate predictions on test data or generalize well (low-bias and low-variance)



Source: http://scott.fortmann-roe.com/docs/BiasVariance.html

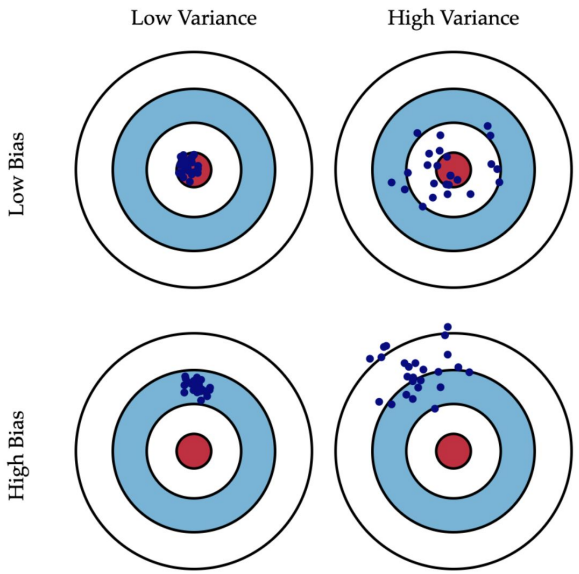
Bias-Variance Tradeoff

The ultimate goal of any learning algorithm is to make accurate predictions on test data or generalize well (low-bias and low-variance)



Source: http://scott.fortmann-roe.com/docs/BiasVariance.html

Ensemble Learning



Source: http://scott.fortmann-roe.com/docs/BiasVariance.html

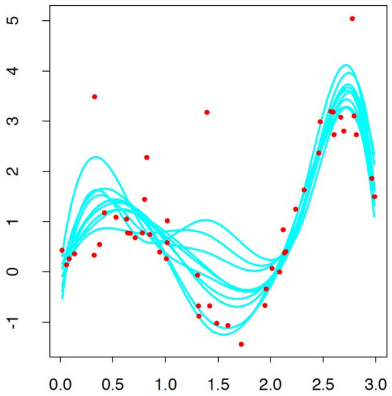
The ultimate goal of any learning algorithm is to make accurate predictions on test data or generalize well (low-bias and low-variance)

Ensemble methods use <u>multiple learning algorithms</u> to obtain better predictive performance than any of the constituent learning algorithms alone.

This is accomplished by either reducing variance, or reducing bias, or both.

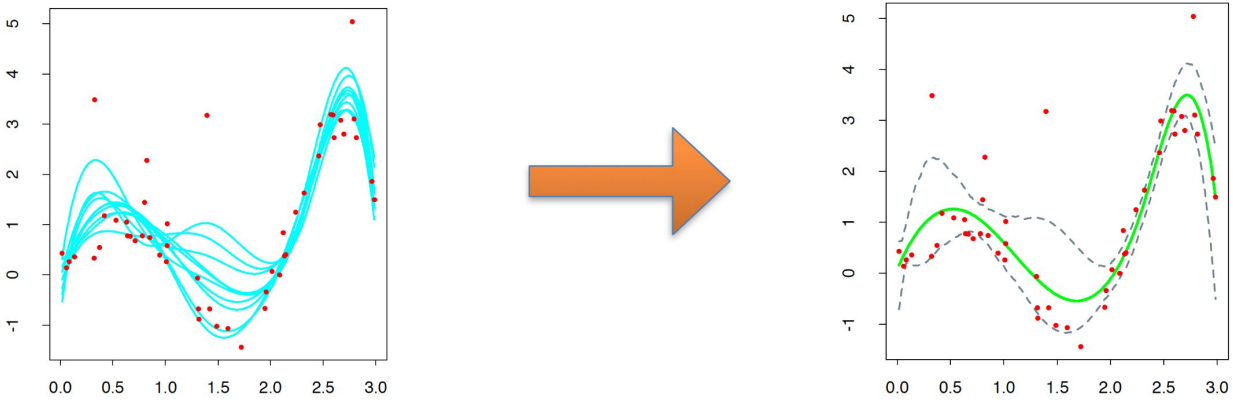
Reducing variance

Average many noisy but approximately unbiased models, and hence reduce the variance.



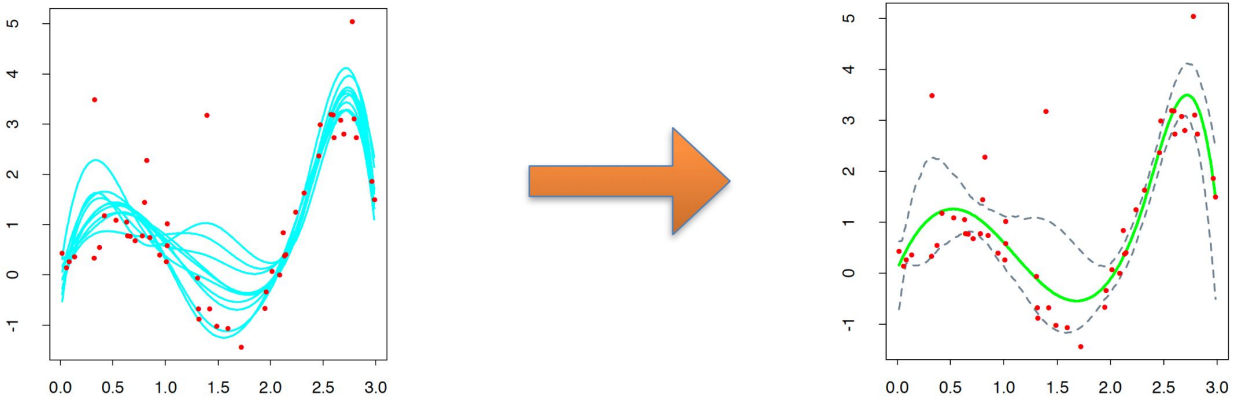
Reducing variance

Average many noisy but approximately unbiased models, and hence reduce the variance.



Reducing variance

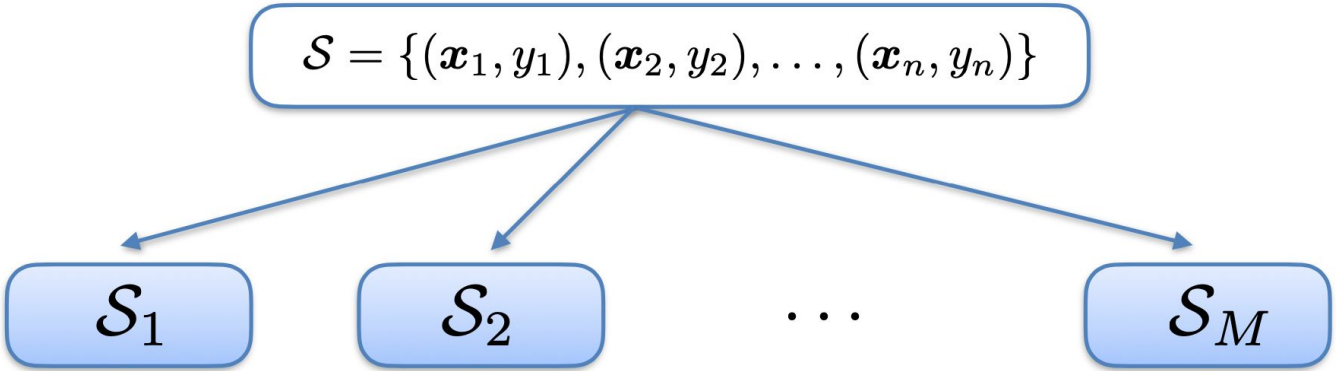
Average many noisy but approximately unbiased models, and hence reduce the variance.



How to get multiple models? Two options:

1. Fragment your original data set. But every model is trained on only a very small part of the entire data set and is likely to perform poorly.
2. Train a single type of model on multiple data sets. The question is: where do these multiple data sets come from, since we are only given one at training time?

Bootstrap Samples



Bootstrap samples: each bootstrap sample is independently generated from original data with sampling with replacement with same size n.

Bagging: Bootstrap Aggregating

The question is: where do these multiple data sets come from, since we are only given one at training time?

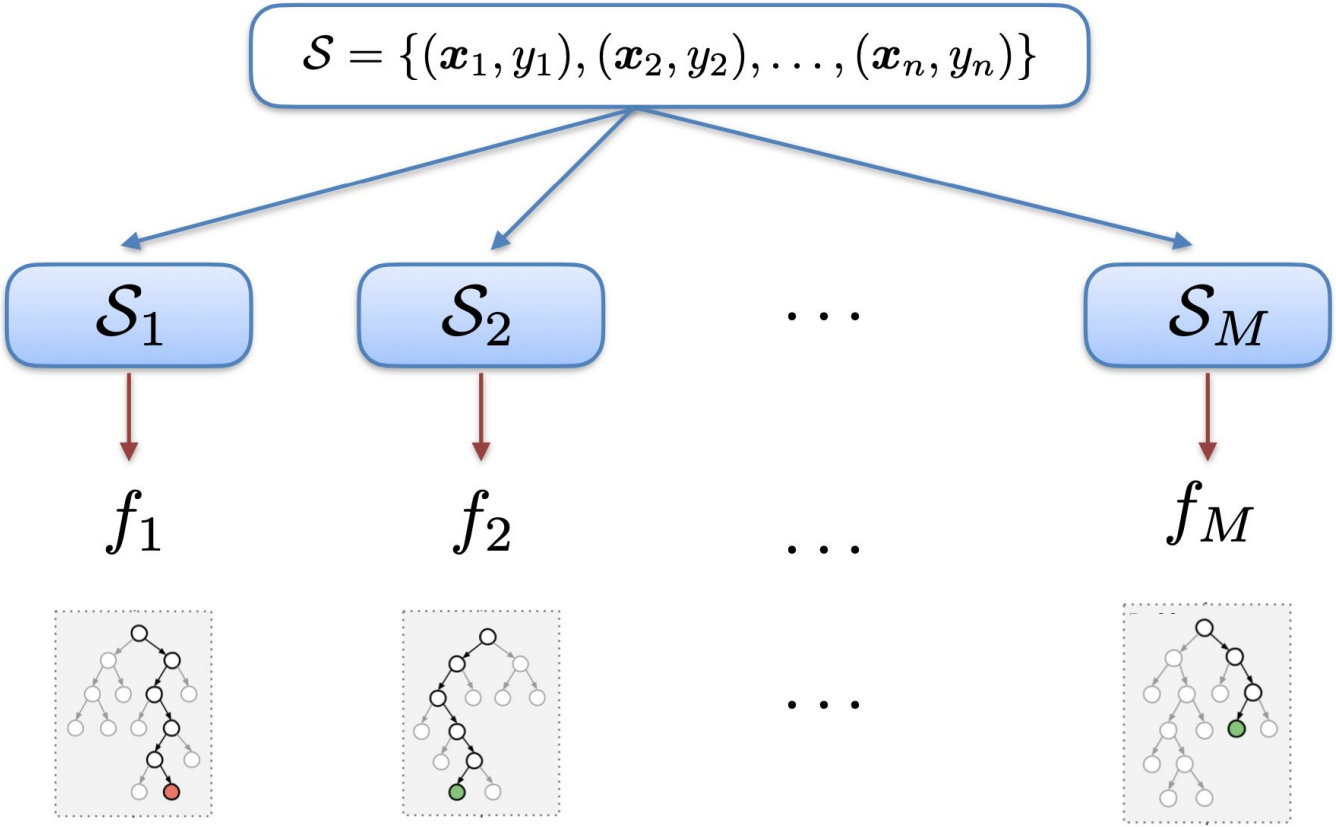
A better solution is to use **bootstrap resampling**: **randomly draw datasets with replacement** from the training data (**bootstrap samples**), each sample the same size as the original training set.

Build a separate prediction model using each bootstrap training set. Then we (combine) average the resulting predictions.

Bagging = Bootstrap Aggregating

Bootstrap Aggregating or Bagging is a general-purpose procedure for reducing the variance of a statistical learning method.

Bagging: fitting



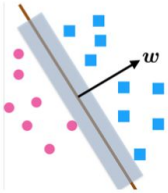
We fit a model (e.g., decision tree) to different Bootstrap samples.

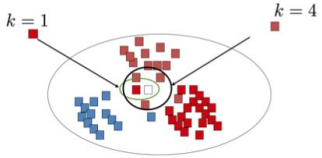
Bagging: fitting

$$\mathcal{S} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)\}$$

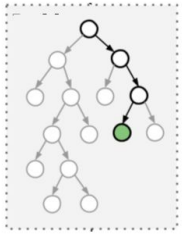
$$\mathcal{S}_1 \qquad \mathcal{S}_2 \qquad \ldots \qquad \mathcal{S}_M$$

$$f_1 \qquad f_2 \qquad \ldots \qquad f_M$$



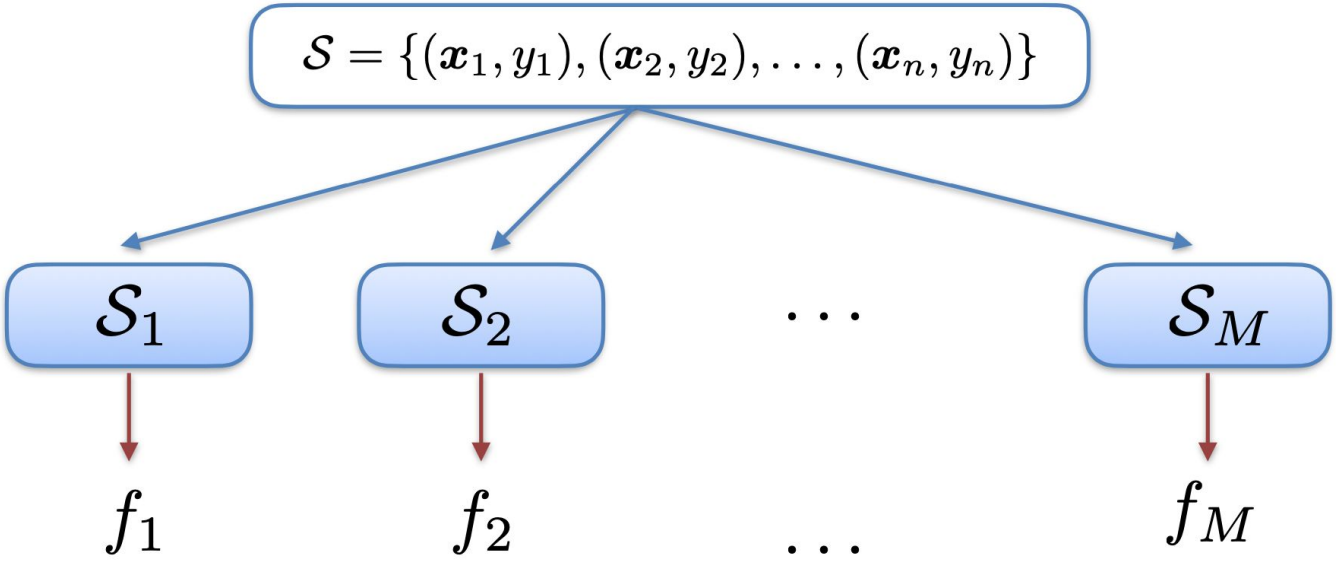


$\ldots$



In generalized bagging, you can use different learners on different samples!

Bagging: final model



The final model is aggregation of all models (voting for classification or averaging for regression)

$$f(\boldsymbol{x}) = \frac{1}{M} \sum_{i=1}^{M} f_i(\boldsymbol{x})$$

How different are bootstrap samples?

$$
\begin{aligned}
&\mathbb{P}[\text{a training example in a bootstrap}] \\
=&1 - \mathbb{P}[\text{a training example is not picked for } n \text{ times}] \\
=&1 - (\mathbb{P}[\text{a training example not picked in one time}])^n \\
=&1 - (1 - \frac{1}{n})^n \to 1 - e^{-1} = 0.632
\end{aligned}
$$

Bagging for Decision Trees

Bagging works especially well for high-variance, low-bias models, such as Decision Trees.

But, it has a problem: The decision trees produced by different Bootstrap samples can be very similar.

- Suppose that there is one very strong feature in the data set, along with a number of other moderately strong features.
- Then in the collection of bagged trees, most or all of the trees will use this strong feature in the top split.
- All of the bagged trees will look quite similar to each other and the predictions from the bagged trees will be highly correlated.
- Averaging many highly correlated quantities does not lead to as large of a reduction in variance as averaging many uncorrelated quantities.
- In particular, this means that bagging will not lead to a substantial reduction in variance over a single tree in this setting.

Random Forests - Choose different features for each Decision Tree

Random Forests provide an improvement over bagging decision trees by way of a small tweak to **de-correlate** the decision trees.

As in bagging, we build a number of decision trees on bootstrapped training samples.

But when building these decision trees, each time a split in a tree is considered, a random sample of $p < d$ features (predictors) is chosen as split candidates from the full set of all d features.

The split is allowed to use only one of those p features. A fresh sample of features is taken at each split, and typically we choose $p = \sqrt{d}$

Random Forests - Choose different features for each Decision Tree

Random forests overcome this problem by forcing each split to consider only a subset of the features.

Therefore, on average (d - p)/d of the splits will not even consider the strong feature, and so other features will have more of a chance.

We can think of this process as de-correlating the trees, thereby making the average of the resulting trees less variable and hence more reliable.

Boosting: Build Strong Models from Weak Models

Spam filtering

Imagine you want to build an email filter that can distinguish spam from non-spam.

The general way we would approach this problem:

1. Gathering as many examples as possible of both spam and non-spam emails.
2. Train a classifier using these examples and their labels.
3. Take the learned classifier, or prediction rule, and use it to filter your emails.
4. The goal is to train a classifier that makes the most accurate predictions possible on new test examples.

Spam filtering

Imagine you want to build an email filter that can distinguish spam from non-spam.

But, building a highly accurate classifier is a difficult task (you still get spam, right?)

Spam filtering

Imagine you want to build an email filter that can distinguish spam from non-spam.

But, building a highly accurate classifier is a difficult task (you still get spam, right?)

We could probably come up with many quick rules of thumb. These could be only moderately accurate

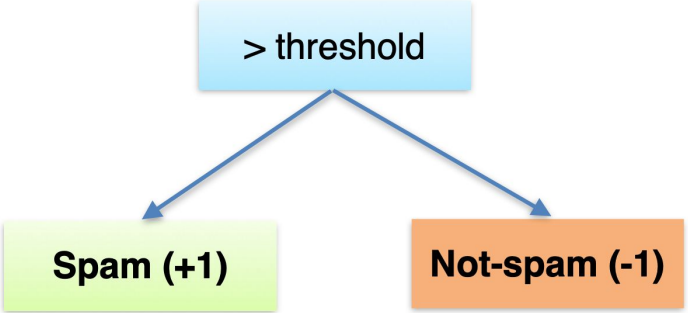
- if the subject line contains "buy now", then classify as spam.
- if the body contains "free money", then classify as spam

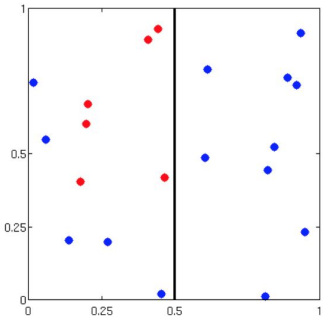
This certainly doesn’t cover all spams, but it will be significantly better than random guessing.

Weak Models: Decision Stumps

Hard to find single highly accurate prediction rule.
Easy to find “rules of thumb” that are “often” correct

A **decision stump** (a.ka., shallow decision tree) is a machine learning model consisting of a one-level decision tree. That is, it is a decision tree with one internal node (the root) which is immediately connected to the terminal nodes (its leaves). A decision stump makes a prediction based on the value of just a single input feature.





Boosting

Can we make dumb learners smart?

Boosting is the process of taking a crummy learning algorithm (technically, called a **weak learner**) and turning it into a great learning algorithm (technically, a **strong learner**)

Boosting also aims at overcoming bias-variance dilemma:

Simple (a.k.a weak learners) models such as decision stumps:

- low variance, don't usually overfit, easy to learn
- high bias, can not learn very hard problems

Boosting

Boosting refers to a general and provably effective method of producing a very accurate classifier by combining rough and moderately inaccurate rules of thumb (slightly better than random guess)

Boosting

Boosting refers to a general and provably effective method of producing a very accurate classifier by combining rough and moderately inaccurate rules of thumb (slightly better than random guess)

Instead of learning a single (weak) classifier, learn many weak classifiers that are good at different parts of the input space!

Boosting

Boosting refers to a general and provably effective method of producing a very accurate classifier by combining rough and moderately inaccurate rules of thumb (slightly better than random guess)

Instead of learning a single (weak) classifier, learn many weak classifiers that are good at different parts of the input space!

Output class: (weighted) vote of each classifier

- Classifiers that are most “sure” will vote with more conviction
- Classifiers will be most “sure” about a particular part of the space
- On average, do better than single classifier

AdaBoost: Adaptive Boosting

Idea: given a weak learner, run it multiple times on **re-weighted** training data, then let learned classifiers vote

AdaBoost Algorithm

On each iteration k = 1, 2, …, K (number of weak learners):

- **Weight each training examples by how incorrectly it was classified (i.e, hardness of classifying example)**
- **Learn a week classifier on weighted training data** f_k
- **Compute the strength of this classifier** α_k

Return final classifier $f(\boldsymbol{x}) = \text{sign}\left(\sum_{k=1}^{K} \alpha_k f_k(\boldsymbol{x})\right)$

Practically useful

Theoretically interesting

AdaBoost

Inputs: training samples + K (number of weak learners) + a weak learner

1: Initialize the weights $\boldsymbol{w}^{(0)} = [\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}]$

2: for $k = 1, 2, \ldots, K$ do:

3: Learn a weak learner $f_k \longleftarrow \text{WeakLearner}\left(\mathcal{S}, \boldsymbol{w}^{(k-1)}\right)$

4: Compute the error $\epsilon_k = \sum_{i=1}^{n} w_i^{(k-1)} \mathbb{I}\left[y_i \neq f_k(\boldsymbol{x}_i)\right]$

5: Compute the strength $\alpha_k = \frac{1}{2} \log\left(\frac{1-\epsilon_k}{\epsilon_k}\right)$

6: Adjust the weights $w_i^{(k)} = \frac{1}{Z} w_i^{(k-1)} \exp\left(-\alpha_k y_i f_k(\boldsymbol{x}_i)\right)$

Return $(\alpha_1, f_1), (\alpha_2, f_2), \ldots, (\alpha_K, f_K)$

AdaBoost

1. At the beginning, all training examples are considered equally hard. Do not mistaken weights with parameters in SVM or Regression!

Inputs: training samples + K (number of weak learners) + a weak learner

1: Initialize the weights $\boldsymbol{w}^{(0)} = [\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}]$

2: for $k = 1, 2, \ldots, K$ do:

3: Learn a weak learner $f_k \longleftarrow \text{WeakLearner}\left(\mathcal{S}, \boldsymbol{w}^{(k-1)}\right)$

4: Compute the error $\epsilon_k = \sum_{i=1}^{n} w_i^{(k-1)} \mathbb{I}\left[y_i \neq f_k(\boldsymbol{x}_i)\right]$

5: Compute the strength $\alpha_k = \frac{1}{2} \log\left(\frac{1-\epsilon_k}{\epsilon_k}\right)$

6: Adjust the weights $w_i^{(k)} = \frac{1}{Z} w_i^{(k-1)} \exp\left(-\alpha_k y_i f_k(\boldsymbol{x}_i)\right)$

Return $(\alpha_1, f_1), (\alpha_2, f_2), \ldots, (\alpha_K, f_K)$

AdaBoost

2. We will learn K different weak learners.

Inputs: training samples + K (number of weak learners) + a weak learner

1: Initialize the weights $\boldsymbol{w}^{(0)} = [\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}]$

2: for $k = 1, 2, \ldots, K$ do:

3: Learn a weak learner $f_k \longleftarrow \text{WeakLearner}\left(\mathcal{S}, \boldsymbol{w}^{(k-1)}\right)$

4: Compute the error $\epsilon_k = \sum_{i=1}^{n} w_i^{(k-1)} \mathbb{I}\left[y_i \neq f_k(\boldsymbol{x}_i)\right]$

5: Compute the strength $\alpha_k = \frac{1}{2} \log\left(\frac{1-\epsilon_k}{\epsilon_k}\right)$

6: Adjust the weights $w_i^{(k)} = \frac{1}{Z} w_i^{(k-1)} \exp\left(-\alpha_k y_i f_k(\boldsymbol{x}_i)\right)$

Return $(\alpha_1, f_1), (\alpha_2, f_2), \ldots, (\alpha_K, f_K)$

AdaBoost

3. In each stage, introduce a weak learner to compensate the shortcomings of existing weak learners.

Inputs: training samples + K (number of weak learners) + a weak learner

1: Initialize the weights $\boldsymbol{w}^{(0)} = [\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}]$

2: for $k = 1, 2, \ldots, K$ do:

3: Learn a weak learner $f_k \longleftarrow \text{WeakLearner}\left(\mathcal{S}, \boldsymbol{w}^{(k-1)}\right)$

4: Compute the error $\epsilon_k = \sum_{i=1}^{n} w_i^{(k-1)} \mathbb{I}\left[y_i \neq f_k(\boldsymbol{x}_i)\right]$

5: Compute the strength $\alpha_k = \frac{1}{2} \log\left(\frac{1-\epsilon_k}{\epsilon_k}\right)$

6: Adjust the weights $w_i^{(k)} = \frac{1}{Z} w_i^{(k-1)} \exp\left(-\alpha_k y_i f_k(\boldsymbol{x}_i)\right)$

Return $(\alpha_1, f_1), (\alpha_2, f_2), \ldots, (\alpha_K, f_K)$

AdaBoost

4. Compute the error of weak learner on training examples weighted by the current weight of each sample!

Inputs: training samples + K (number of weak learners) + a weak learner

1: Initialize the weights $\boldsymbol{w}^{(0)} = [\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}]$

2: for $k = 1, 2, \ldots, K$ do:

3: Learn a weak learner $f_k \longleftarrow \text{WeakLearner}\left(\mathcal{S}, \boldsymbol{w}^{(k-1)}\right)$

4: Compute the error $\epsilon_k = \sum_{i=1}^{n} w_i^{(k-1)} \mathbb{I}\left[y_i \neq f_k(\boldsymbol{x}_i)\right]$

5: Compute the strength $\alpha_k = \frac{1}{2} \log\left(\frac{1-\epsilon_k}{\epsilon_k}\right)$

6: Adjust the weights $w_i^{(k)} = \frac{1}{Z} w_i^{(k-1)} \exp\left(-\alpha_k y_i f_k(\boldsymbol{x}_i)\right)$

Return $(\alpha_1, f_1), (\alpha_2, f_2), \ldots, (\alpha_K, f_K)$

AdaBoost

Inputs: training samples + K (number of weak learners) + a weak learner

1: Initialize the weights $\boldsymbol{w}^{(0)} = [\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}]$

2: for $k = 1, 2, \ldots, K$ do:

3: Learn a weak learner $f_k \longleftarrow \text{WeakLearner}\left(\mathcal{S}, \boldsymbol{w}^{(k-1)}\right)$

4: Compute the error $\epsilon_k = \sum_{i=1}^{n} w_i^{(k-1)} \mathbb{I}\left[y_i \neq f_k(\boldsymbol{x}_i)\right]$

5: Compute the strength $\alpha_k = \frac{1}{2} \log\left(\frac{1-\epsilon_k}{\epsilon_k}\right)$

6: Adjust the weights $w_i^{(k)} = \frac{1}{Z} w_i^{(k-1)} \exp\left(-\alpha_k y_i f_k(\boldsymbol{x}_i)\right)$

Return $(\alpha_1, f_1), (\alpha_2, f_2), \ldots, (\alpha_K, f_K)$

5. Compute the strength of weak learner!

AdaBoost

Inputs: training samples + K (number of weak learners) + a weak learner

1: Initialize the weights $\boldsymbol{w}^{(0)} = [\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}]$

2: for $k = 1, 2, \ldots, K$ do:

3: Learn a weak learner $f_k \longleftarrow \text{WeakLearner}\left(\mathcal{S}, \boldsymbol{w}^{(k-1)}\right)$

4: Compute the error $\epsilon_k = \sum_{i=1}^{n} w_i^{(k-1)} \mathbb{I}\left[y_i \neq f_k(\boldsymbol{x}_i)\right]$

5: Compute the strength $\alpha_k = \frac{1}{2} \log\left(\frac{1-\epsilon_k}{\epsilon_k}\right)$

6: Adjust the weights $w_i^{(k)} = \frac{1}{Z} w_i^{(k-1)} \exp\left(-\alpha_k y_i f_k(\boldsymbol{x}_i)\right)$

Return $(\alpha_1, f_1), (\alpha_2, f_2), \ldots, (\alpha_K, f_K)$

6. Adjust the weight of training examples: If it is correctly classified decrease the weight; otherwise increase its weight! Z is the normalization factor

AdaBoost

Inputs: training samples + K (number of weak learners) + a weak learner

1: Initialize the weights $\boldsymbol{w}^{(0)} = [\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}]$

2: for $k = 1, 2, \ldots, K$ do:

3: Learn a weak learner $f_k \longleftarrow \text{WeakLearner}\left(\mathcal{S}, \boldsymbol{w}^{(k-1)}\right)$

4: Compute the error $\epsilon_k = \sum_{i=1}^{n} w_i^{(k-1)} \mathbb{I}\left[y_i \neq f_k(\boldsymbol{x}_i)\right]$

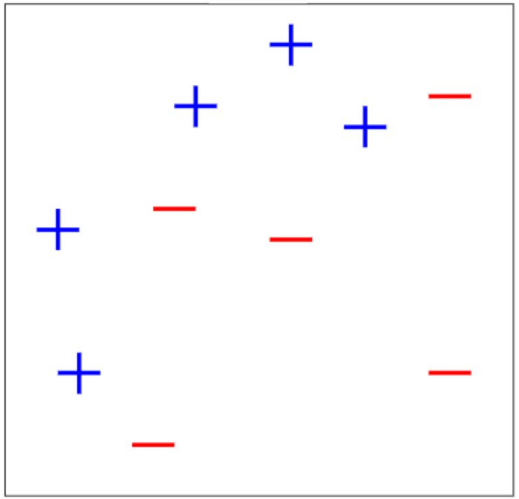
5: Compute the strength $\alpha_k = \frac{1}{2} \log\left(\frac{1-\epsilon_k}{\epsilon_k}\right)$

6: Adjust the weights $w_i^{(k)} = \frac{1}{Z} w_i^{(k-1)} \exp\left(-\alpha_k y_i f_k(\boldsymbol{x}_i)\right)$

Return $(\alpha_1, f_1), (\alpha_2, f_2), \ldots, (\alpha_K, f_K)$

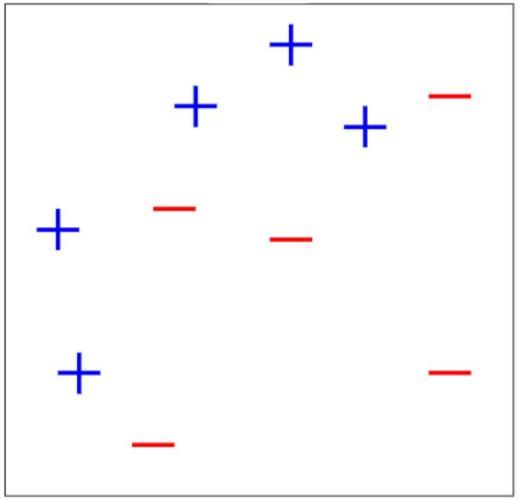
The final model is weighted average of all weak learners

Toy example (decision stumps)



$\mathcal{S}$

Toy example (decision stumps)

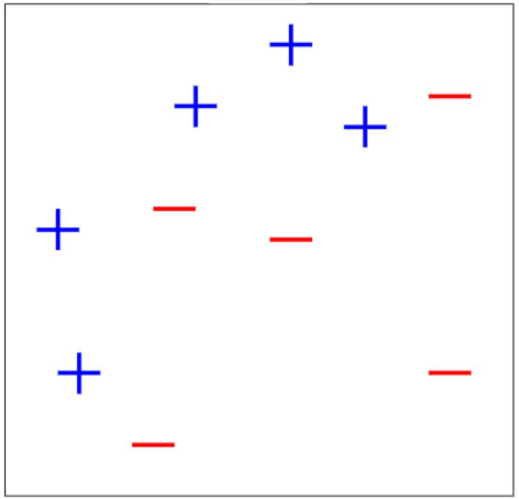


$\mathcal{S}$

weak classifiers: vertical or horizontal half-planes

i.e, $\mathrm{WeakLearner}\,(\mathcal{S}, \boldsymbol{w})$ module returns either a vertical or horizontal line that minimizes weighted loss on training data!

Toy example (decision stumps)



$\mathcal{S}$

weak classifiers: vertical or horizontal half-planes

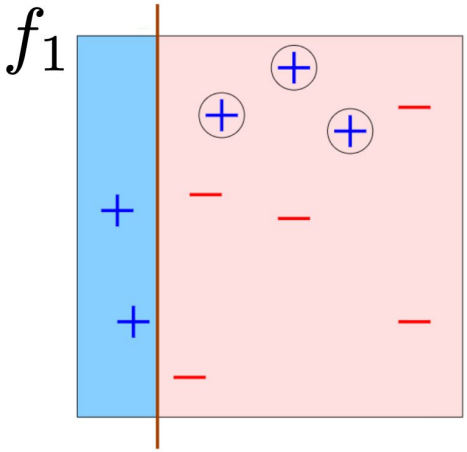
i.e, $\mathrm{WeakLearner}\,(\mathcal{S}, \boldsymbol{w})$ module returns either a vertical or horizontal line that minimizes weighted loss on training data!

$$\boldsymbol{w}^{(0)} = [\frac{1}{10}, \frac{1}{10}, \ldots, \frac{1}{10}]$$

At the beginning, all the training data share the same weight (1/10 here)

Round I

We learn a vertical predictor on $(\mathcal{S}, \boldsymbol{w}^{(0)})$. It makes mistake on three $+$ examples!



$$\epsilon_1 = 0.30$$

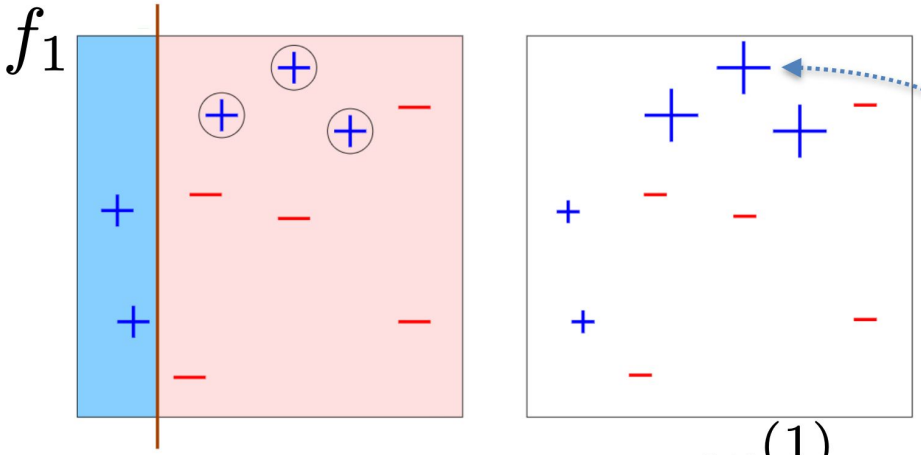
$$\alpha_1 = 0.42$$

Compute the error $\epsilon_k = \sum_{i=1}^{n} w_i^{(k-1)} \mathbb{I}\left[y_i \neq f_k(\boldsymbol{x}_i)\right]$

Compute the strength $\alpha_k = \frac{1}{2} \log\left(\frac{1-\epsilon_k}{\epsilon_k}\right)$

Round I

We learn a vertical predictor on $(\mathcal{S}, \boldsymbol{w}^{(0)})$. It makes mistake on three + examples! We increase the weight of these three data and decrease the weight of rest! (telling the next learner to mostly focus on these three). The new weights vector is $\boldsymbol{w}^{(1)}$



The size of label shows its current weight!

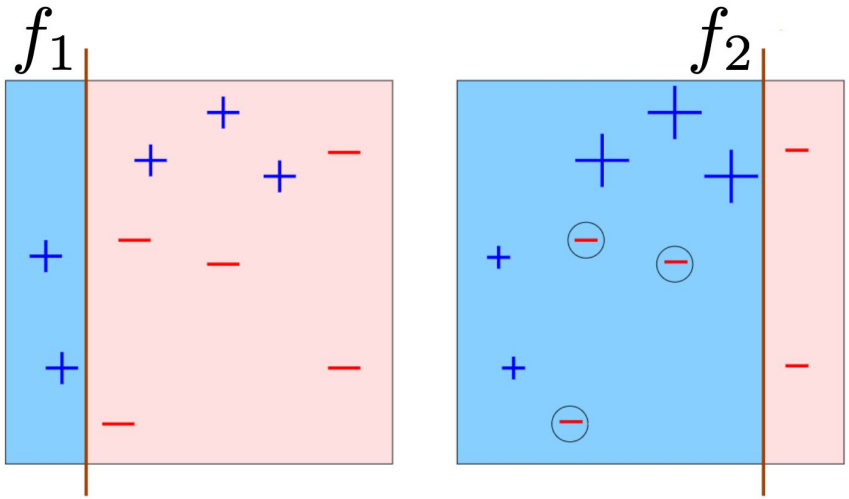
$$\epsilon_1 = 0.30$$

$$\alpha_1 = 0.42$$

Adjust the weights $w_i^{(k)} = \frac{1}{Z} w_i^{(k-1)} \exp\left(-\alpha_k y_i f_k(\boldsymbol{x}_i)\right)$

Round II

We learn another vertical predictor on ($\mathcal{S}, \boldsymbol{w}^{(1)}$) . It makes mistake on three examples!

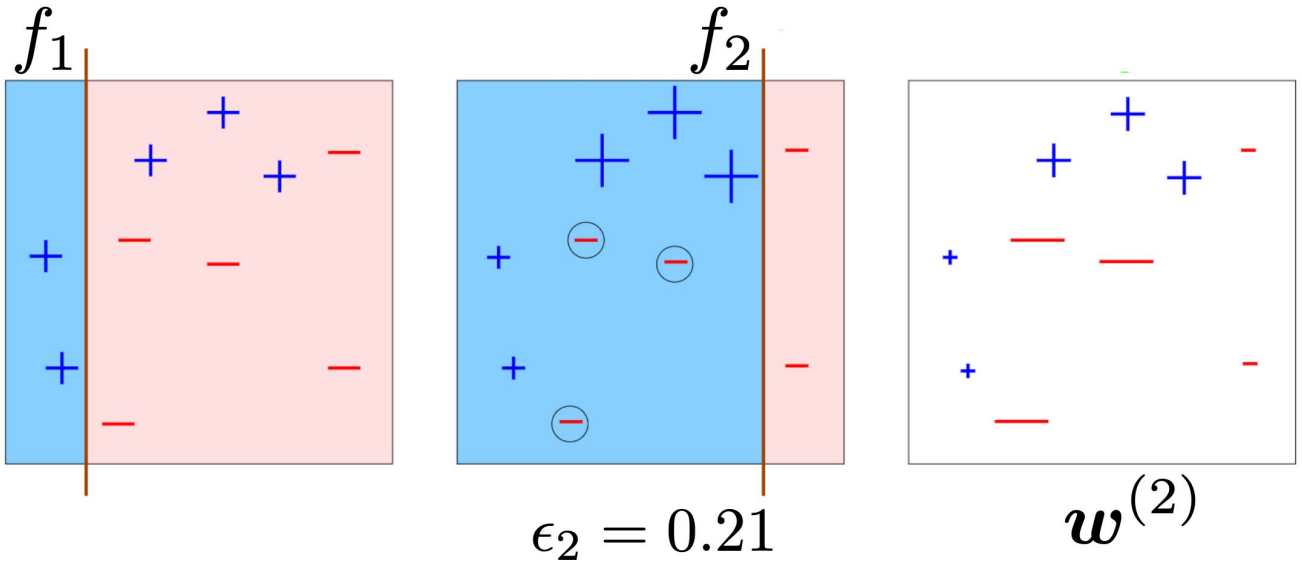


$$\epsilon_2 = 0.21$$

$$\alpha_2 = 0.65$$

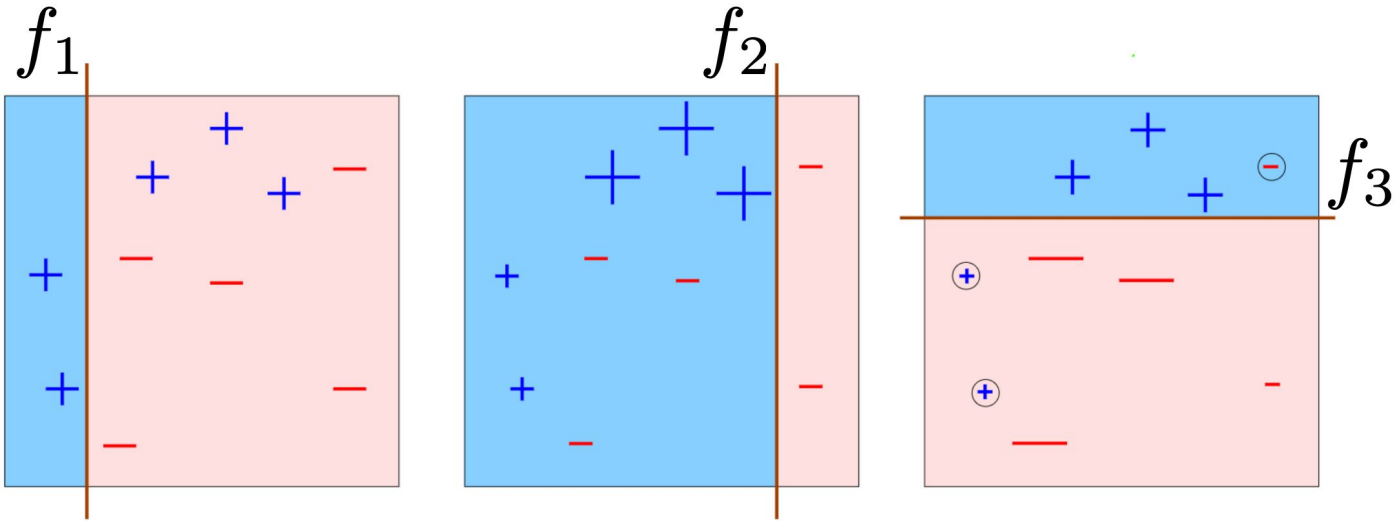
Round II

We learn another vertical predictor on ($\mathcal{S}, \boldsymbol{w}^{(1)}$) . It makes mistake on three examples! We increase the weight of these two data and decrease the weight of rest! The new weights vector is $\boldsymbol{w}^{(2)}$.



Round III

Finally, we learn a horizontal predictor on $(\mathcal{S}, \boldsymbol{w}^{(2)})$ and compute its weighted error on training data to calculate its weight!

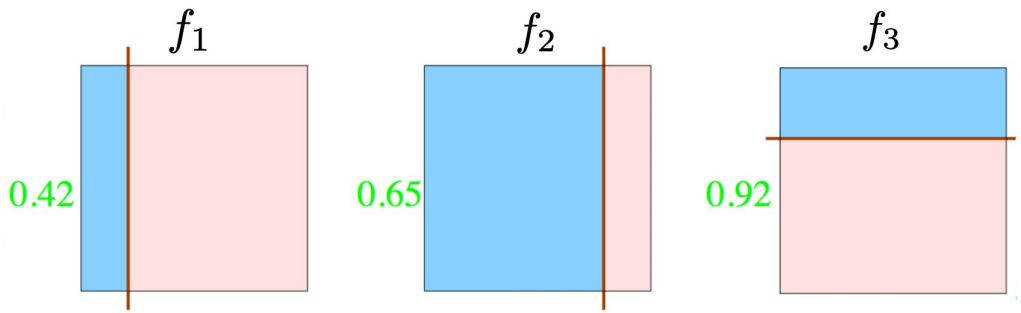


$$\epsilon_3 = 0.14$$

$$\alpha_3 = 0.92$$

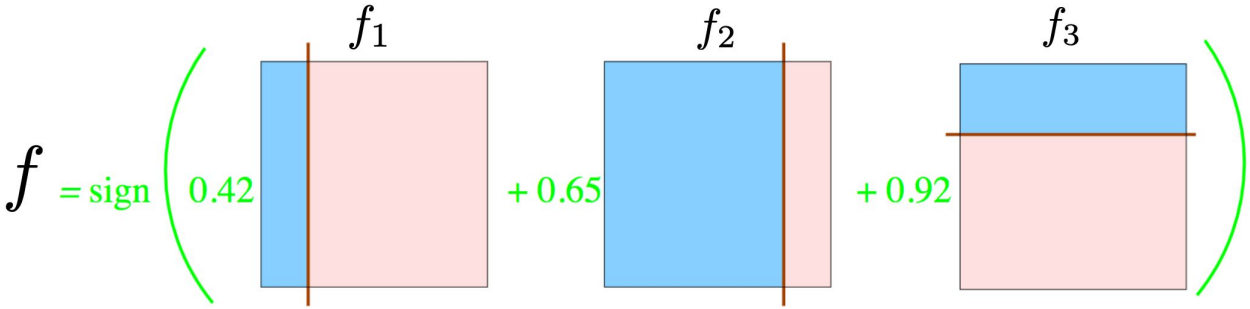
Final classifier

The final classifier is the weighted combination of learned classifiers:

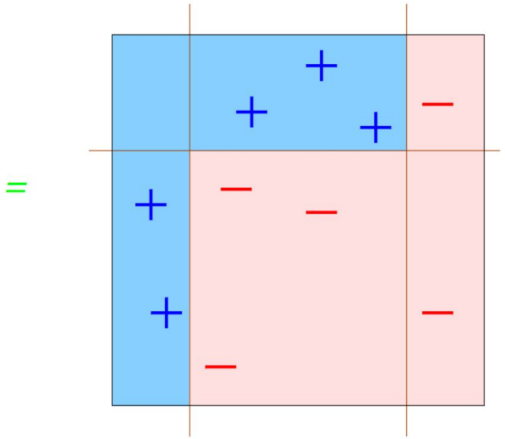


Final classifier

The final classifier is the weighted combination of learned classifiers:



$$f(\boldsymbol{x}) = \text{sign}\left(\sum_{k=1}^{K} \alpha_k f_k(\boldsymbol{x})\right)$$



This is how the decision boundary of final combined classifier looks like!

AdaBoost with cost-sensitive classification

Question: How to learn a classifier on a weighted training data

- The error is not uniform for all training examples
- The higher weight means, larger loss for making mistake
- The smaller weight means, smaller loss for making mistake

AdaBoost with cost-sensitive classification

Question: How to learn a classifier on a weighted training data

- The error is not uniform for all training examples
- The higher weight means, larger loss for making mistake
- The smaller weight means, smaller loss for making mistake

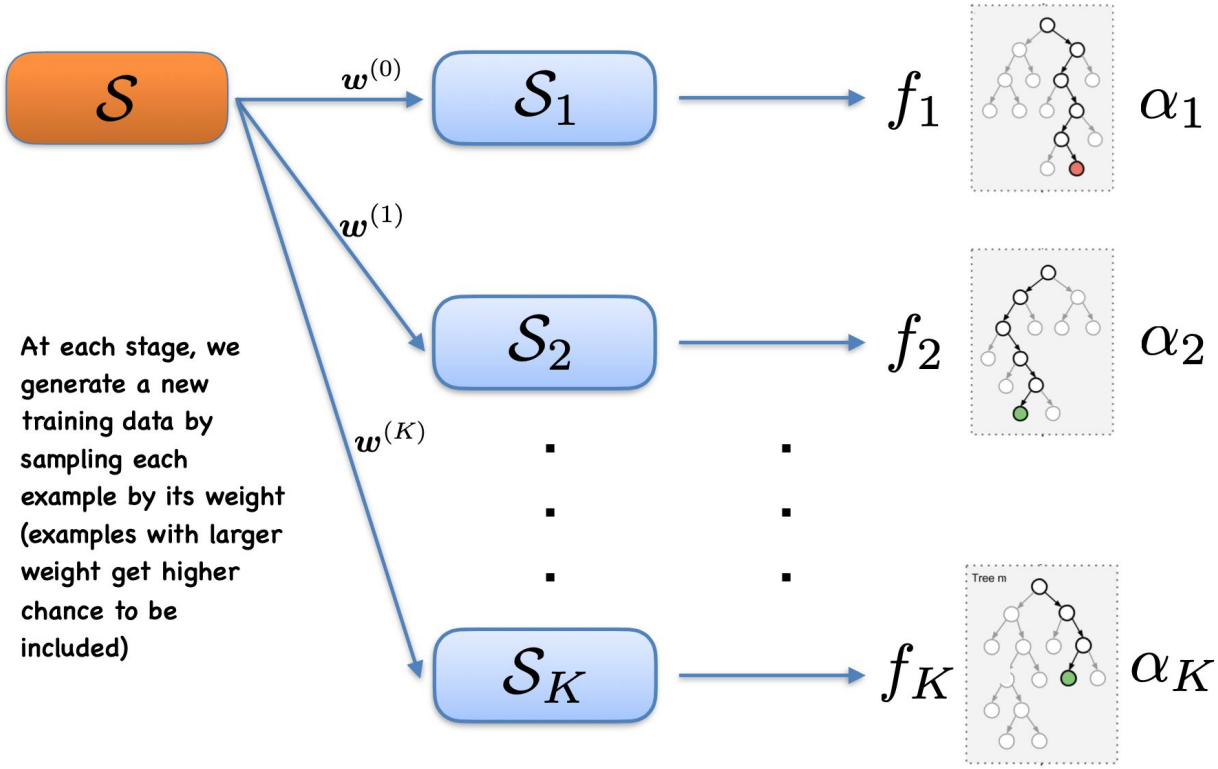
A simple solution is to change the training objective and make it weighted (i.e, penalize different levels of loss for different mistakes)

$$\arg\min_{f} \sum_{i=1}^{n} w_i \ell(f; (\boldsymbol{x}_i, y_i))$$

Known as cost-sensitive classification

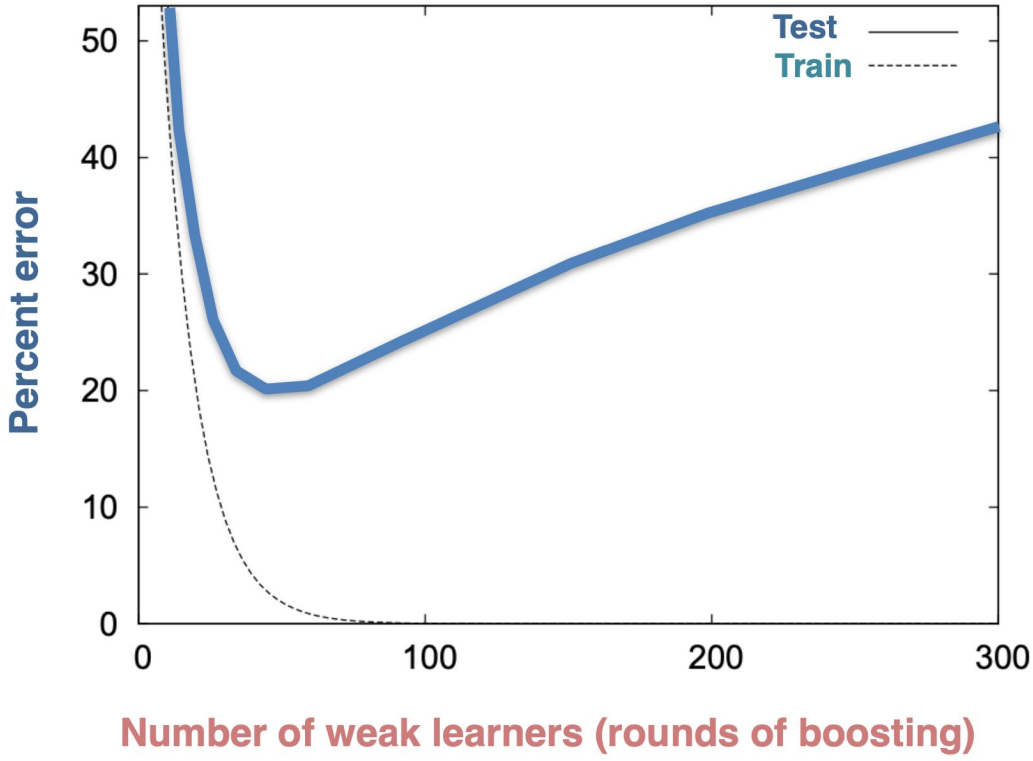
AdaBoost with weighted sampling

An alternative solution is to sample a new training data based on weight of individual examples (can easily use existing algorithms without any change!)



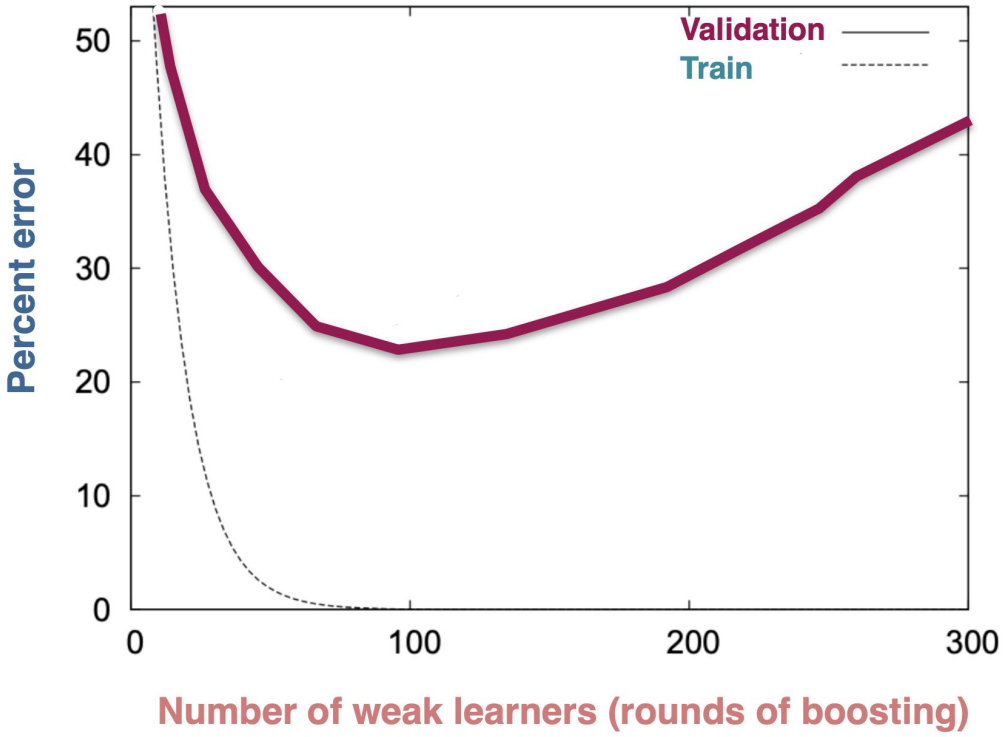
AdaBoost and # of weak classifiers

Need to decide the rounds of boosting (# of weak learners in final model) by **cross validation**

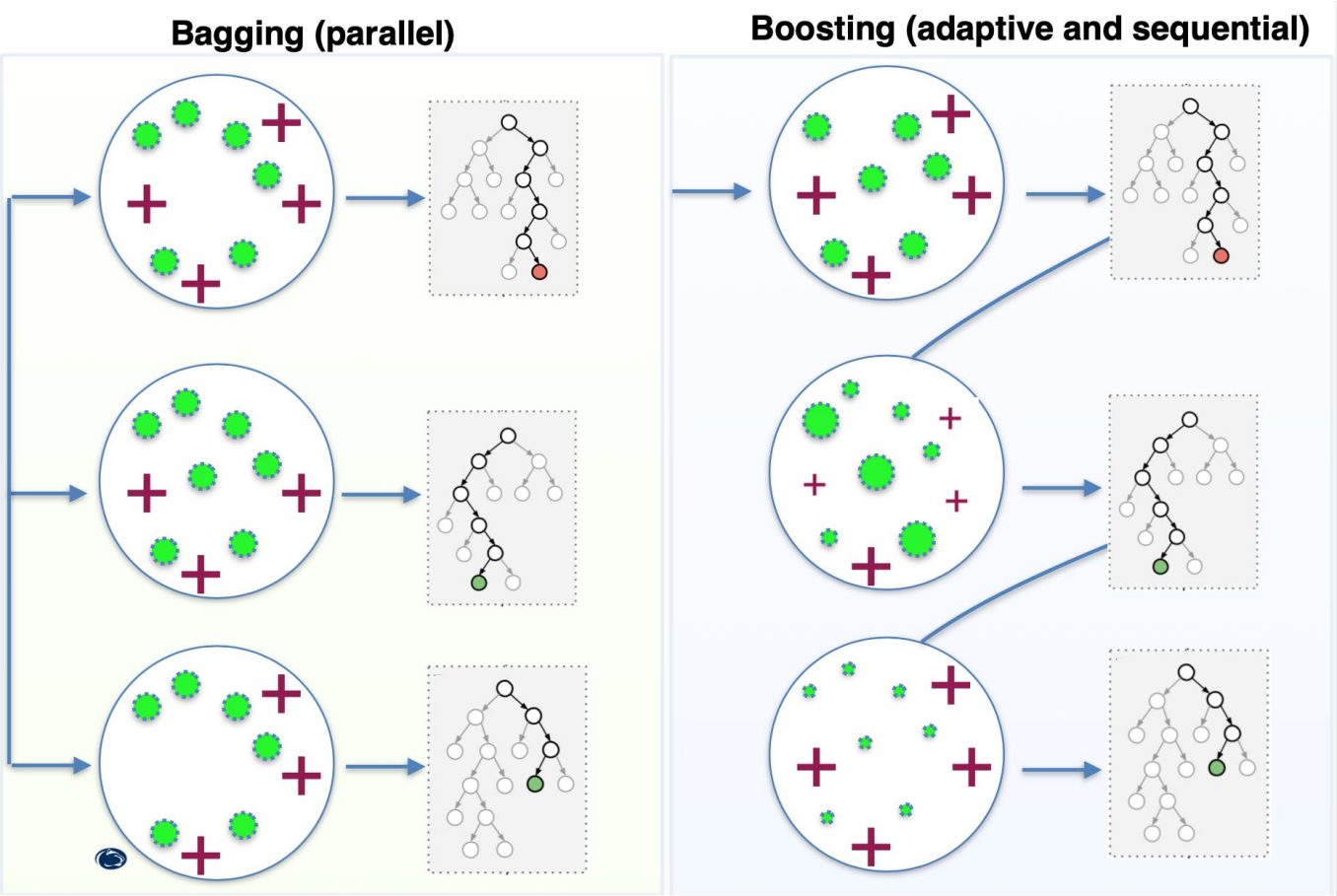


AdaBoost and # of weak classifiers

Early stopping: compute the error on validation set after every round of boosting, and stop training as soon as validation error starts increasing!



Bagging vs Boosting



Gradient Boosting

Gradient Boosting can be used for both regression and classification.

Gradient Boosting = Gradient Descent + Boosting

Gradient Boosting

Gradient Boosting can be used for both regression and classification.

Gradient Boosting = Gradient Descent + Boosting

AdaBoost

- In each stage, introduce a weak learner to compensate the "shortcomings" of existing weak learners
- shortcomings are identified by high-weight data points

Gradient Boosting

Gradient Boosting can be used for both regression and classification.

Gradient Boosting = Gradient Descent + Boosting

AdaBoost

- In each stage, introduce a weak learner to compensate the "shortcomings" of existing weak learners
- shortcomings are identified by high-weight data points

Gradient Boosting

- In each stage, introduce a weak learner to compensate the shortcomings of existing weak learners.
- shortcomings are identified by gradients

Both high-weight data points and gradients tell us how to improve our model.

Gradient boosting

Suppose we are given data $\mathcal{S} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)\}$

We already fit a model f_1 by linear regression, so that we have

$$f_1(\boldsymbol{x}_i) \approx y_i$$

Gradient boosting

Suppose we are given data $\mathcal{S} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)\}$
We already fit a model f_1 by linear regression, so that we have

$$f_1(\boldsymbol{x}_i) \approx y_i$$

Certainly, f_1 is not perfect, so there is a difference

$$y_i - f_1(\boldsymbol{x}_i)$$

$y_i - f_1(\boldsymbol{x}_i)$ are called residuals. These are the parts that existing model f_1 cannot do well.

residual = actual value - predicted value

Gradient boosting

Suppose we are given data $\mathcal{S} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)\}$
We already fit a model f_1 by linear regression, so that we have

$$f_1(\boldsymbol{x}_i) \approx y_i$$

Certainly, f_1 is not perfect, so there is a difference

$$y_i - f_1(\boldsymbol{x}_i)$$

$y_i - f_1(\boldsymbol{x}_i)$ are called residuals. These are the parts that existing model f_1 cannot do well.

residual = actual value - predicted value

If we are going to train a second model, how to compensate the residuals?

Gradient boosting

Suppose we want to fit a second model f_2 by linear regression, we want to have

$$f_1(\boldsymbol{x}_i) + f_2(\boldsymbol{x}_i) = y_i$$

Equivalently,

$$f_2(\boldsymbol{x}_i) = y_i - f_1(\boldsymbol{x}_i)$$

i.e., we want f_2 to compensate f_1 by predicting the residuals of f_1

Gradient boosting

Suppose we want to fit a second model f_2 by linear regression, we want to have

$$f_1(\boldsymbol{x}_i) + f_2(\boldsymbol{x}_i) = y_i$$

Equivalently,

$$f_2(\boldsymbol{x}_i) = y_i - f_1(\boldsymbol{x}_i)$$

i.e., we want f_2 to compensate f_1 by predicting the residuals of f_1

However, if f_2 is still not perfect, we still have residuals:

$$y_i - f_1(\boldsymbol{x}_i) - f_2(\boldsymbol{x}_i)$$

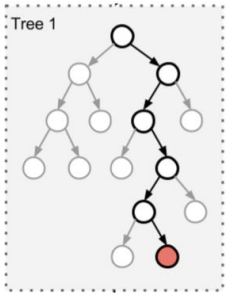
and we need f_3

$$f_3(\boldsymbol{x}_i) = y_i - f_1(\boldsymbol{x}_i) - f_2(\boldsymbol{x}_i)$$

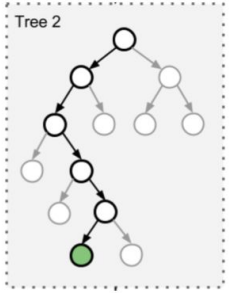
Gradient boosting

We can repeat, and the training labels at each round is the residual accumulated so far

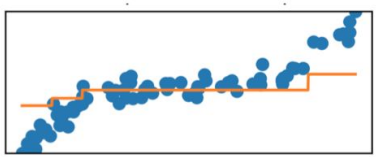
$$\mathcal{S} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)\} \longrightarrow f_1$$



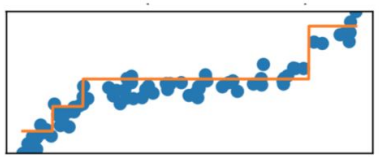
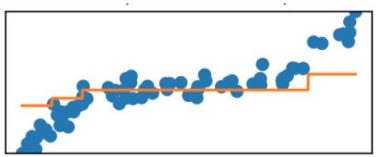
$$\mathcal{S} = \{(\boldsymbol{x}_1, y_1 - f_1(\boldsymbol{x}_1)), (\boldsymbol{x}_2, y_2 - f_1(\boldsymbol{x}_2)), \ldots, (\boldsymbol{x}_n, y_n - f_1(\boldsymbol{x}_n))\} \longrightarrow f_2$$



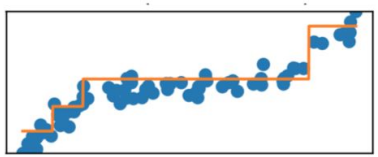
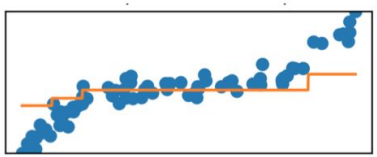
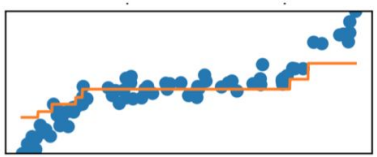
Gradient boosting

Round	Residual	Total Prediction
1		

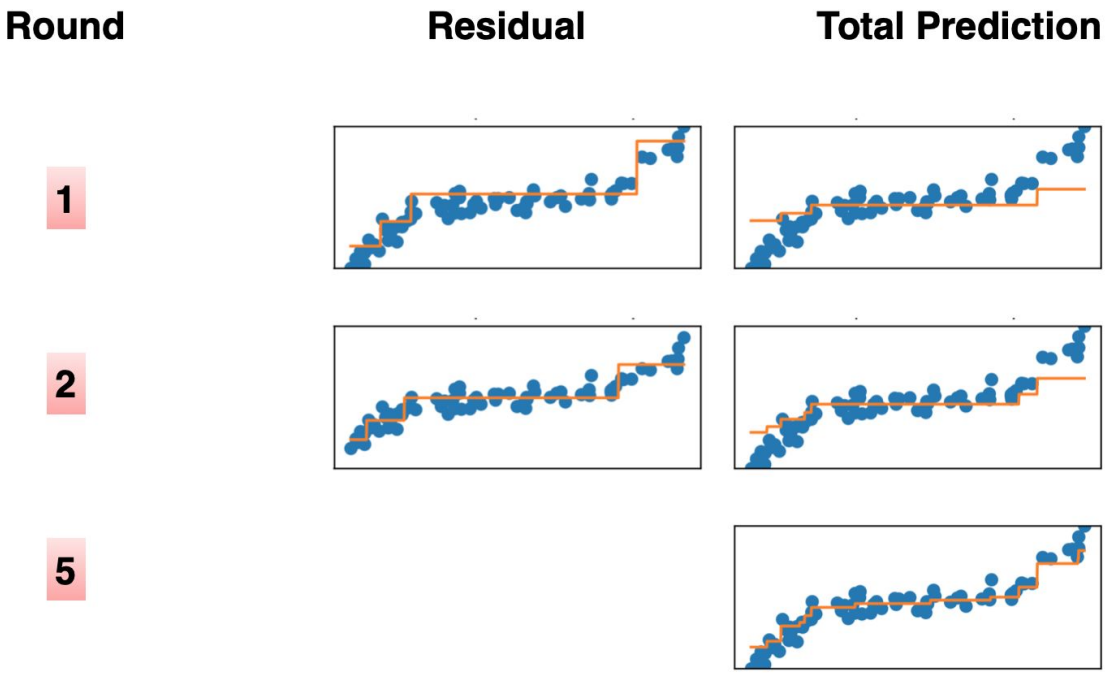
Gradient boosting

Round	Residual	Total Prediction
1		

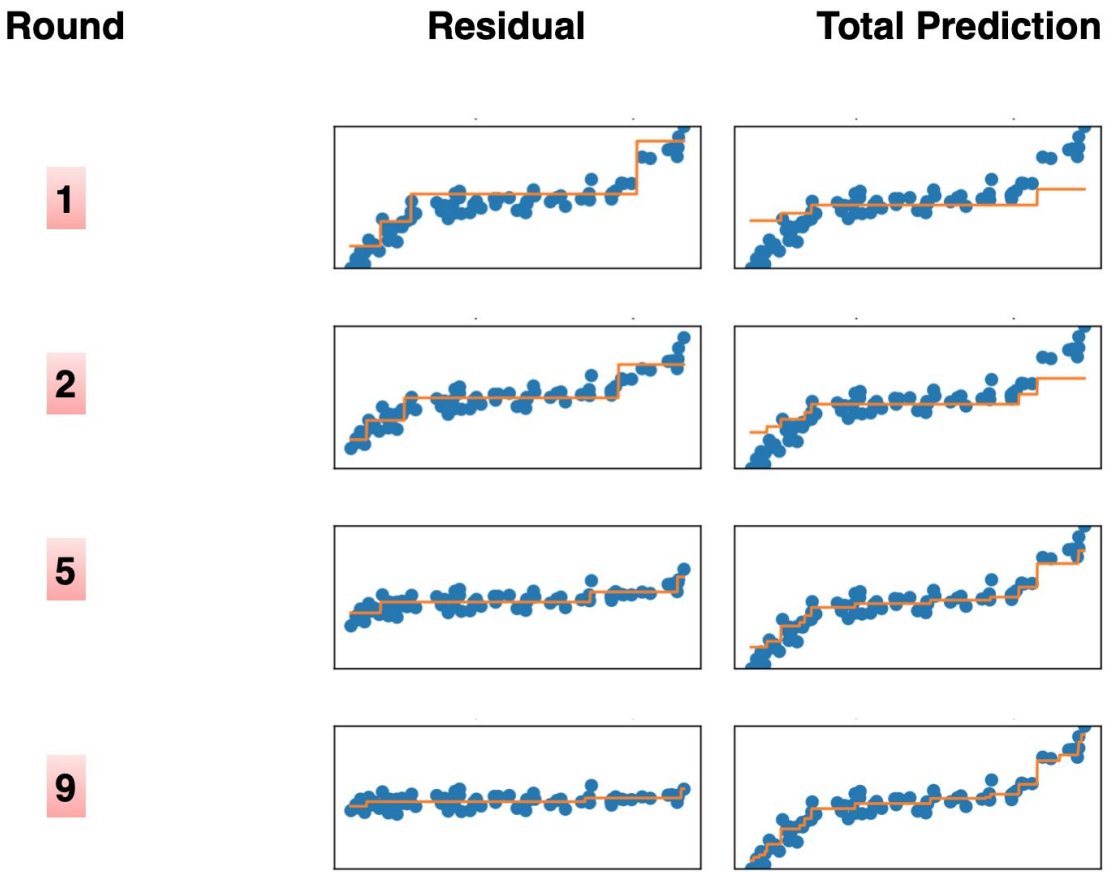
Gradient boosting

Round	Residual	Total Prediction
1		
2		

Gradient boosting



Gradient boosting



How is this related to gradient?

If we are using square error loss:

$$\frac{1}{2}(y_i - f_1(\boldsymbol{x}_i))^2$$

The gradient is

$$-(y_i - f_1(\boldsymbol{x}_i))$$

For regression with square loss, residuals is the same as negative gradients!

Shortcomings are identified by negative gradients.

Gradient boosting

For regression with square loss,

residuals ⟺ negative gradients

fit model to residuals ⟺ fit model to negative gradients

update model based on residuals ⟺ update model based on negative gradients

Gradient boosting

For regression with square loss,

residuals $\Longleftrightarrow$ negative gradients

fit model to residuals $\Longleftrightarrow$ fit model to negative gradients

update model based on residuals $\Longleftrightarrow$ update model based on negative gradients

Gradient Boosting Algorithms

start with an initial model, say, $F(x) = \frac{\sum_{i=1}^{n} y_i}{n}$

iterate until converge:

- calculate negative gradients $-g(x_i)$
- fit a regression tree h to negative gradients $-g(x_i)$
- $F := F + \rho h$, where $\rho = 1$

Negative gradient:

$$-g(x_i) = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} = y_i - F(x_i)$$

So, gradient boosting could be specialized to a gradient descent algorithm, and generalizing it entails "plugging in" a different loss and its gradient.

Gradient tree boosting

You can also use gradient boosting for classification.

Especially, people often use gradient boosting on decision trees.

- Build a series of decision trees
- We use CART: Classification And Regression Trees, where a real score is associated with each of the leaves
- Each decision tree predict the residuals of previous ensembles

XGBoost algorithm

XGBoost (Extreme gradient boosting): An efficient implementation of Gradient Boosted Trees

- Very successful in many competitions
- Winning Solutions in many Kaggle competitions

```
conda install -c conda-forge xgboost

from xgboost import XGBClassifier
xgb = XGBClassifier()
xgb.fit(X_train, y_train)
xgb.score(X_test, y_test))
```

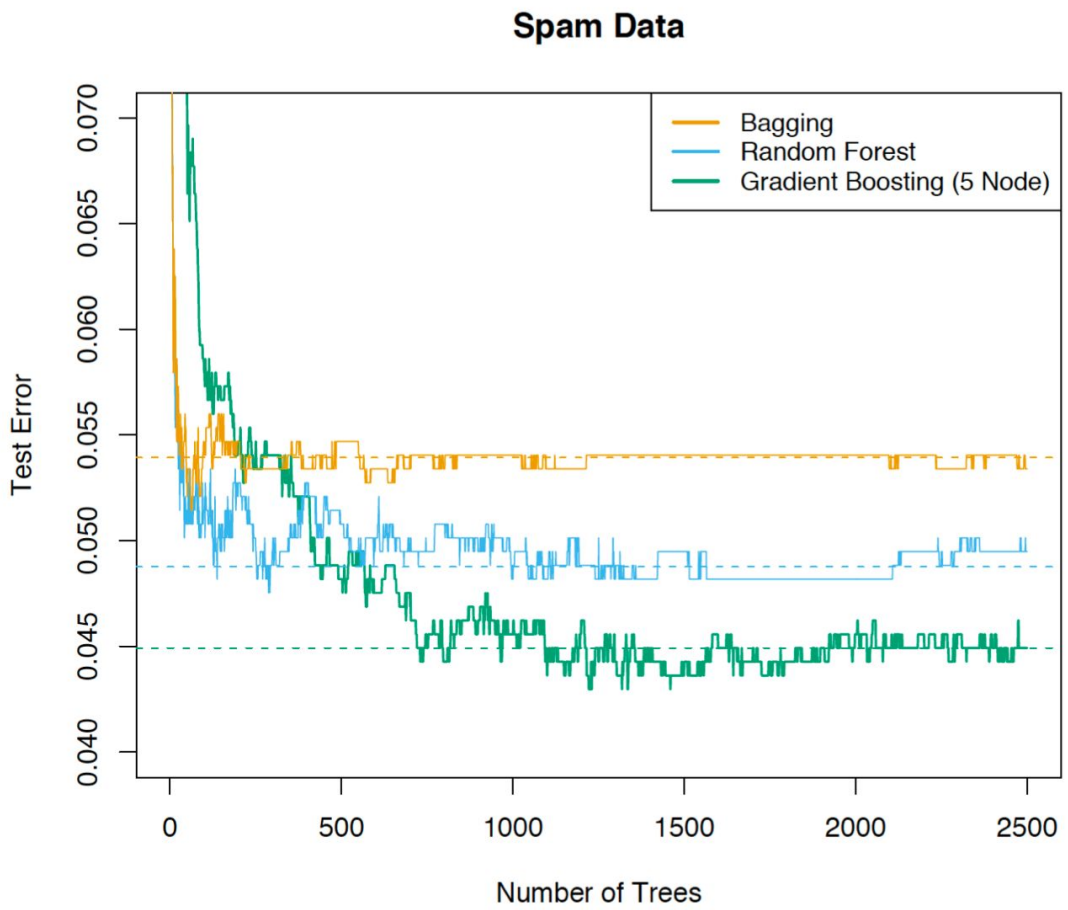
Machine Learning Challenge Winning Solutions

XGBoost is extensively used by machine learning practitioners to create state of art data science solutions, this is a list of machine learning winning solutions with XGBoost. Please send pull requests if you find ones that are missing here.

- Maksims Volkovs, Guangwei Yu and Tomi Poutanen, 1st place of the 2017 ACM RecSys challenge. Link to paper.
- Vlad Sandulescu, Mihai Chiru, 1st place of the KDD Cup 2016 competition. Link to the arxiv paper.
- Marios Michailidis, Mathias Müller and HJ van Veen, 1st place of the Dato Truely Native? competition. Link to the Kaggle interview.
- Vlad Mironov, Alexander Guschin, 1st place of the CERN LHCb experiment Flavour of Physics competition. Link to the Kaggle interview.
- Josef Slavicek, 3rd place of the CERN LHCb experiment Flavour of Physics competition. Link to the Kaggle interview.
- Mario Filho, Josef Feigl, Lucas, Gilberto, 1st place of the Caterpillar Tube Pricing competition. Link to the Kaggle interview.
- Qingchen Wang, 1st place of the Liberty Mutual Property Inspection. Link to the Kaggle interview.
- Chenglong Chen, 1st place of the Crowdflower Search Results Relevance. Link to the winning solution.
- Alexandre Barachant ("Cat") and Rafał Cycoń ("Dog"), 1st place of the Grasp-and-Lift EEG Detection. Link to the Kaggle interview.
- Halla Yang, 2nd place of the Recruit Coupon Purchase Prediction Challenge. Link to the Kaggle interview.
- Owen Zhang, 1st place of the Avito Context Ad Clicks competition. Link to the Kaggle interview.
- Keiichi Kuroyanagi, 2nd place of the Airbnb New User Bookings. Link to the Kaggle interview.
- Marios Michailidis, Mathias Müller and Ning Situ, 1st place Homesite Quote Conversion. Link to the Kaggle interview.

Bagging versus Boosting

Bagging vs Random Forest vs Gradient Boosting



When to use tree-based ensemble models

Model non-linear relationships

Doesn’t care about scaling, no need for feature engineering

Single tree: very interpretable (if small)

Random forests very robust, good benchmark

Gradient boosting often best performance with careful tuning

Summary

Decision trees can be simple, but often produce noisy (bushy) or weak (stunted) classifiers.

Bagging (Breiman, 1996): Fit many large trees to bootstrap-resampled versions of the training data, and classify by majority vote.

Random Forests (Breiman 1999): Fancier version of bagging.

Boosting (Freund & Schapire, 1996): Fit many large or small trees to re-weighted versions of the training data. Classify by weighted majority vote.

In general

Boosting > Random Forests > Bagging > Single Tree

[where ‘>’ means performs better]