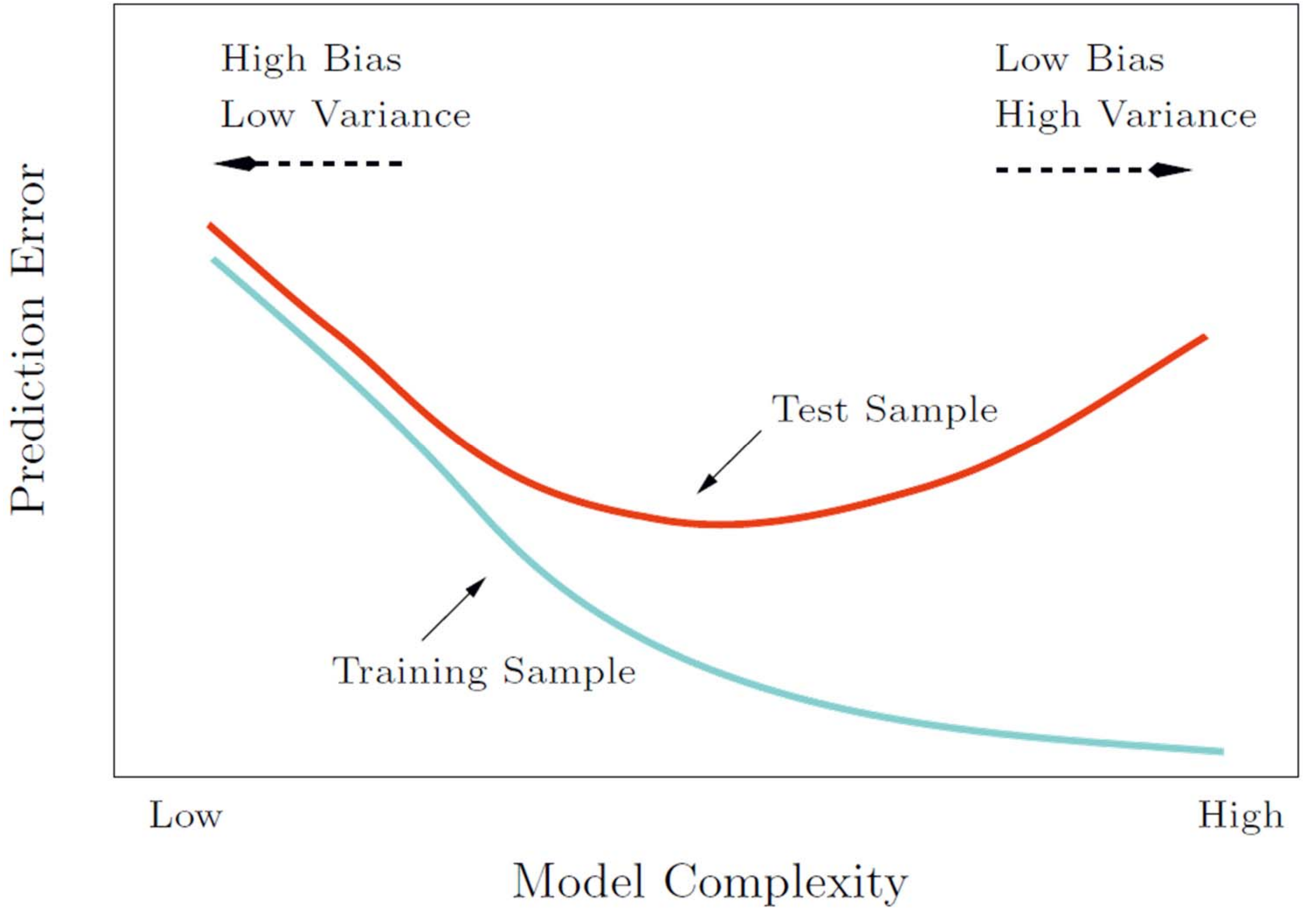
So Far…

- Our goal (supervised learning)
  - Naïve Bayesian classifier
  - Linear Regression
  - Logistic Regression
  - SVM
  - Perceptron
  - Neural Network
  - k Nearest Neighbor
  - Decision Tree

  - **Single Classifier**
- Can the classifiers be combined to achieve better performance?
  - Two heads are better than one



$\mathrm{EPE}(f) =$ **(bias)^2 + variance + noise**

Ensemble Methods & Random Forest

Deng Cai (蔡登)

College of Computer Science
Zhejiang University

dengcai@gmail.com

The Bagging Algorithm

- The name Bagging came from the abbreviation of *Bootstrap Aggregating* [Breiman, 1996d], implies the two key ingredients of Bagging are bootstrap and aggregation.

  - **Bootstrap sampling:** re-sample N examples from D **uniformly with replacement**—can also use arbitrary N' instead of original N

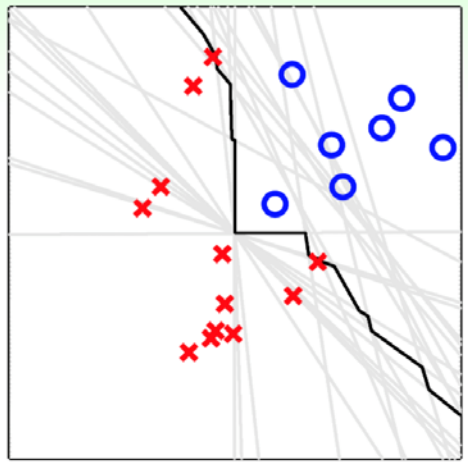
  - **Aggregating**: adopts the most popular strategies for aggregating the outputs of the base learners, that is, *voting* for classification and *averaging* for regression.

L. Breiman. Bagging predictors. Machine Learning, 24(2):123–140, 1996d

The Bagging Algorithm

Pocket is a modified Perceptron Learning Algorithm.

Bagging 25 Pocket Models. The max iteration for each Pocket Algorithm is 1000.



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

- very diverse g_t from bagging
- proper **non-linear** boundary after aggregating binary classifiers
- bagging works reasonably well **if base algorithm sensitive to data randomness**

Why Bagging Works

- Let f denote the ground-truth function and $h(\boldsymbol{x})$ denote a learner trained from the bootstrap distribution D_{bs}. The aggregated learner generated by Bagging is

$$H(\boldsymbol{x}) = E_{D_{bs}}[h(\boldsymbol{x})]$$

- With simple algebra and the inequality $(E[X])^2 \leq E[X^2]$, we have

$$\left(f - H(\boldsymbol{x})\right)^2 \leq E_{D_{bs}}\left[\left(f - h(\boldsymbol{x})\right)^2\right]$$

- Thus, by integrating both sides over the distribution, we can get that the mean-squared error of $H(\boldsymbol{x})$ is smaller than that of $h(\boldsymbol{x})$ averaged over the bootstrap sampling distribution

Why Bagging Works: Bias & Variance

$$\left\{E_D\left(f(\boldsymbol{x};D)\right) - E(y|\boldsymbol{x})\right\}^2 + E_D\left\{\left[f(\boldsymbol{x};D) - E_D\left(f(\boldsymbol{x};D)\right)\right]^2\right\}$$

- Let f denote the ground-truth function and $h(\boldsymbol{x})$ denote a learner trained from the bootstrap distribution D_{bs}. The aggregated learner generated by Bagging is

$$H(\boldsymbol{x}) = E_{D_{bs}}[h(\boldsymbol{x})]$$

- Bias: $E_D[h(\boldsymbol{x})] - f \approx H(\boldsymbol{x}) - f$
- Variance:
  - Assume there are T bs samples, if data is $i.i.d.$ and each variance is σ^2 . The ensemble variance is $\frac{1}{T}\sigma^2$

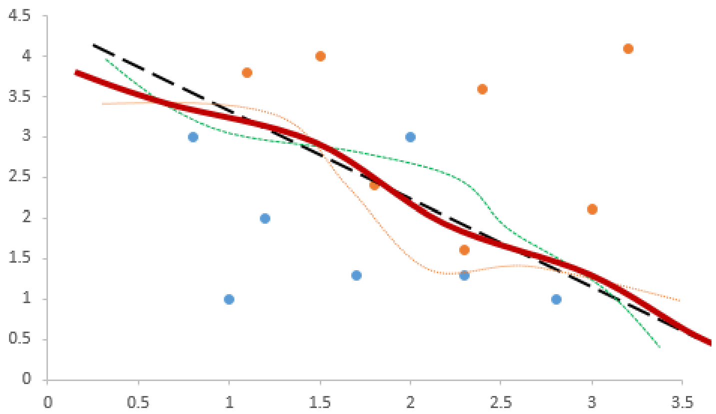
$$H(\boldsymbol{x}) = E_{D_{bs}}[h(\boldsymbol{x})] = \frac{1}{T}\sum_{i=1}^{T} h_i(\boldsymbol{x})$$

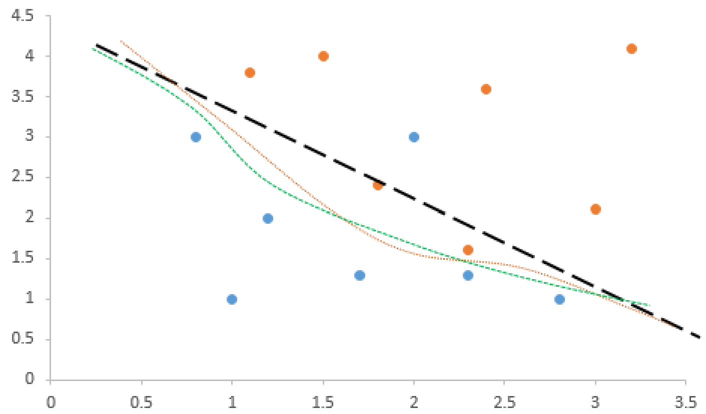
$$Var\left(H(x)\right) = Var\left(\frac{1}{T}\sum_{i}^{T} h_i(x)\right) = \frac{1}{T^2}Var\left(\sum_{i=1}^{T} h_i(\boldsymbol{x})\right) = \frac{1}{T^2}\sum_{i=1}^{T} Var(h_i(\boldsymbol{x})) = \frac{1}{T}\sigma^2$$

  - In reality, Bootstrap Sampled data is $i.d.$ (not necessarily independent) with correlation ρ

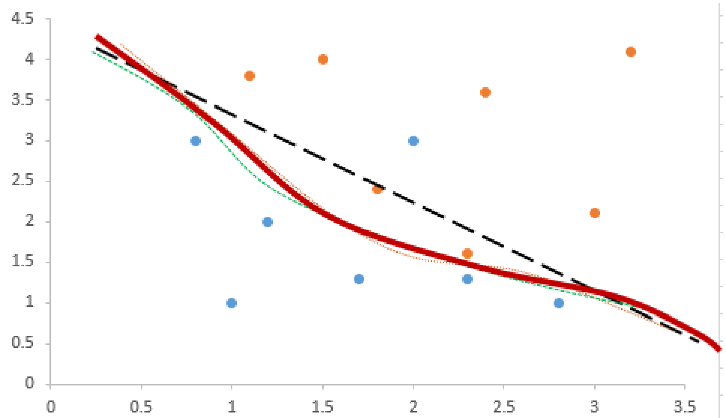
$$\rho\sigma^2 + \frac{1-\rho}{T}\sigma^2$$

Why Bagging Works





Why?



If Bootstrap Sampled data is $i.d.$ with correlation ρ:

$$\rho\sigma^2 + \frac{1-\rho}{T}\sigma^2$$

Why Bagging Works

- Performance improvement brought by Bagging is large when the base learner is unstable(large variance).
  - The base learner should aware the little change, sometimes overfitting is allowable.

- Thus, Friedman and Hall [2007] concluded that Bagging can reduce the variance of higher-order components, yet not affect the linear components. This implies that Bagging is better applied with highly nonlinear learners.

J. H. Friedman and P. Hall. On bagging and nonlinear estimation. Journal of Statistical Planning and Inference, 137(3):669–683, 2007.

Which Classifier is a good choice for base learner?

- Naïve Bayesian classifier
- Perceptron
- Linear Regression
- Logistic Regression
- SVM
- Neural Network
- k Nearest Neighbor
- Decision Tree

Model with low bias benefits from bagging

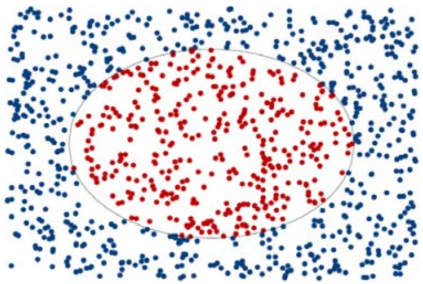
- Decision Tree!
  - Non-linear classifier
  - Easy to use and interpret
  - Can perfectly fit to any training data (overfitting) with high test error. (zero bias, high variance)

Random Forest

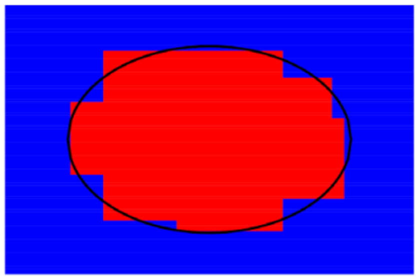
- Random Forest, which uses decision tree as basic unit in bagging, is an ensemble model.
- Why Random Forest?
  - Advantages of Decision Tree:
    - Handling missing value
    - Robust to outliers in input space
    - Fast
    - Good Interpretability
  - Limitations of Decision Tree:
    - Low accuracy, low bias with high variance and easy to overfit
  - Ensemble to maintain advantages while increasing accuracy

Random Forest

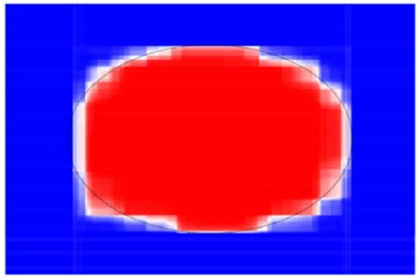
- Bagging sample is a random process.
  - Bootstrap sample process: random sampling he given N samples with replacement
  - Repeat T times and create new T training sets
  - Get T models
- Aggregating all models will fuzzy up the decision boundary, which help reduce the variance, and prevent the ***one man rule*** danger



Data Set

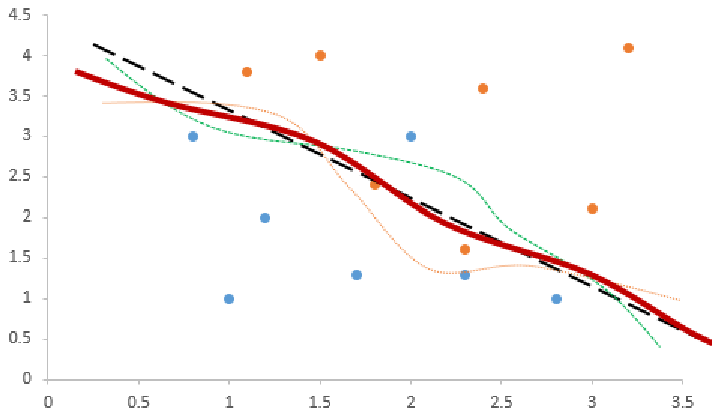


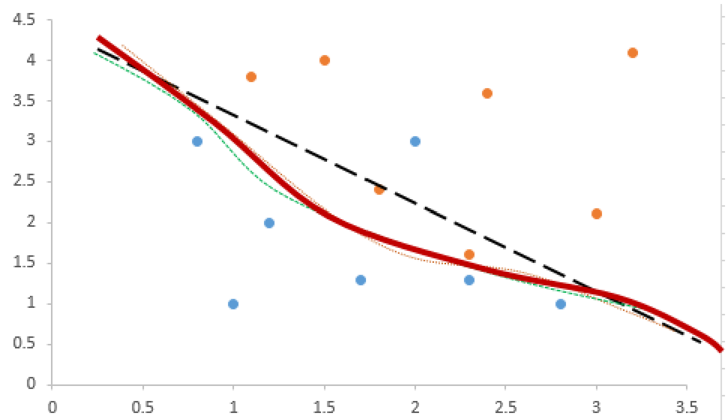
Decision Tree



Random Forest

How to increase the variation of each tree?





Random Forest

- The main difference between the method of RF and Bagging is that RF incorporate randomized feature selection at each split step.

- At each split step : Get feature subsets ***f*** from the whole **F**
  - More efficient in building trees
    - each time we only pick the best feature from size(f) rather than size(F).
      - We often let size(f) = $\sqrt{k}$ in classification and k/3 in regression
  - Each tree is not as good as DT, but work better when they are combined.

Construct Random Forest

- Let N be the number of trees and K be the feature subset size.
- For each N iterations: (building a tree in forest)
  - 1. Select a new bootstrap sample from original training set
  - 2. Growing a tree…
  - 3. At each internal node, randomly select K features from ALL features and then, determine the best split in ONLY the K features.
  - 4. Do not pruning
  - 5. Until Test Error never decrease (here means Validation error in RF)
- At last, overall prediction as the average(or vote) from N trees.

3 Rules of RF makes the learners more Diverse.

- Random forest need basic learner aware the little change, sometimes overfit is allowable.

- Each time, basic learner doesn't learn from all data, but from Random bootstrap sampled data.

- Basic learner doesn't use all features, but Random select some features.

Other ways to generate base learner?

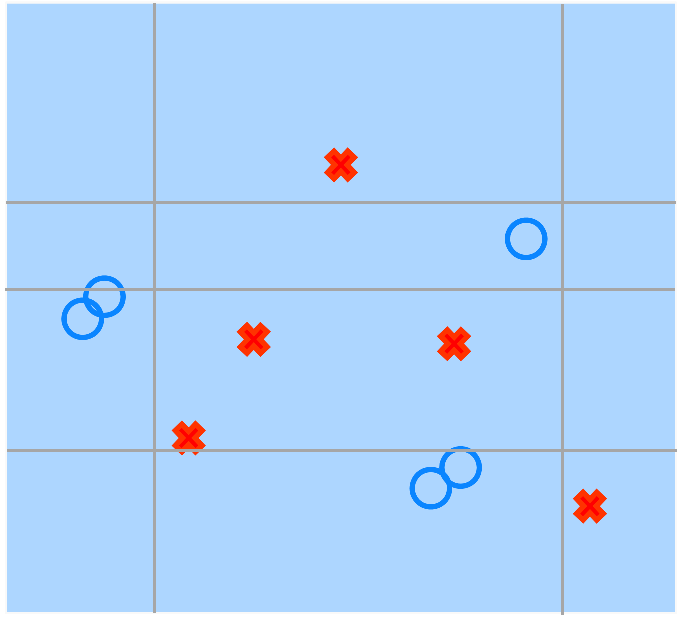
- The base learner should have low bias? Not necessary
- The base learner should have high variance? What does this mean?
  - Different base learners should be different.
  - Different base learners should have different result.

$$\min_{f} \left\{ \sum_{i=1}^{n} \ell(f, \boldsymbol{x}_i, y_i) + \lambda R(f) \right\}$$

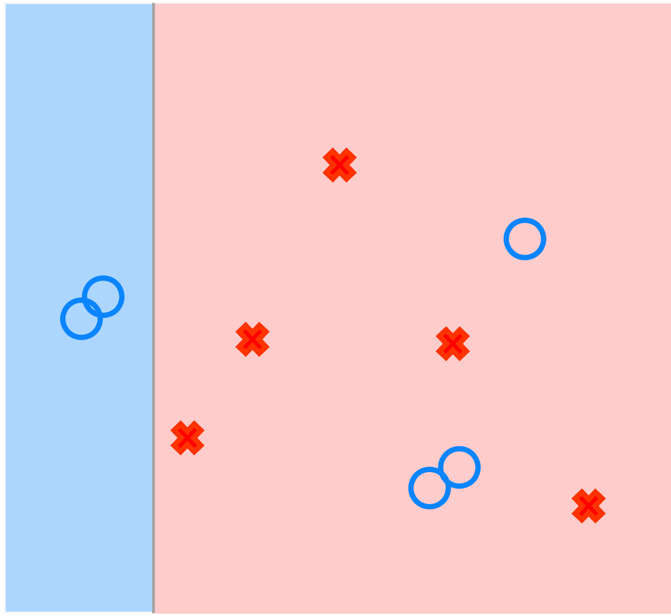
$$\min_{f} \left\{ \sum_{i=1}^{n} w_i \ell(f, \boldsymbol{x}_i, y_i) + \lambda R(f) \right\}$$

- Boosting
  - Training base learners by assigning different weights to the samples

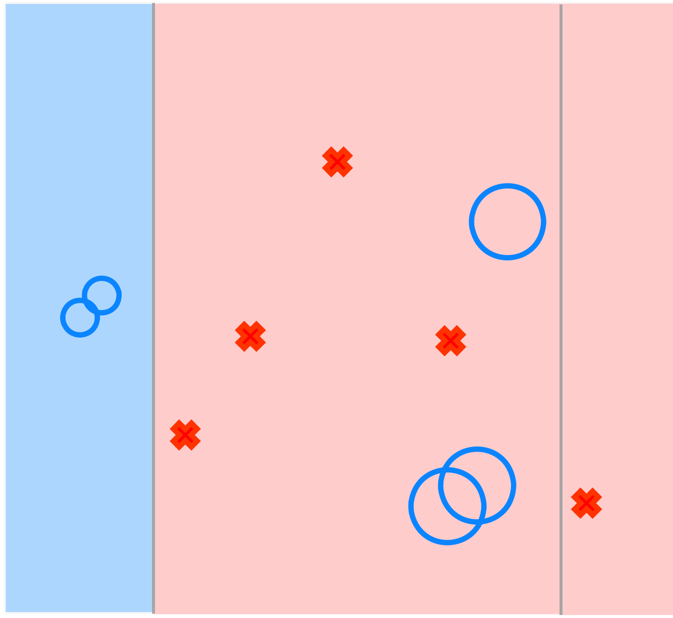
Boosting



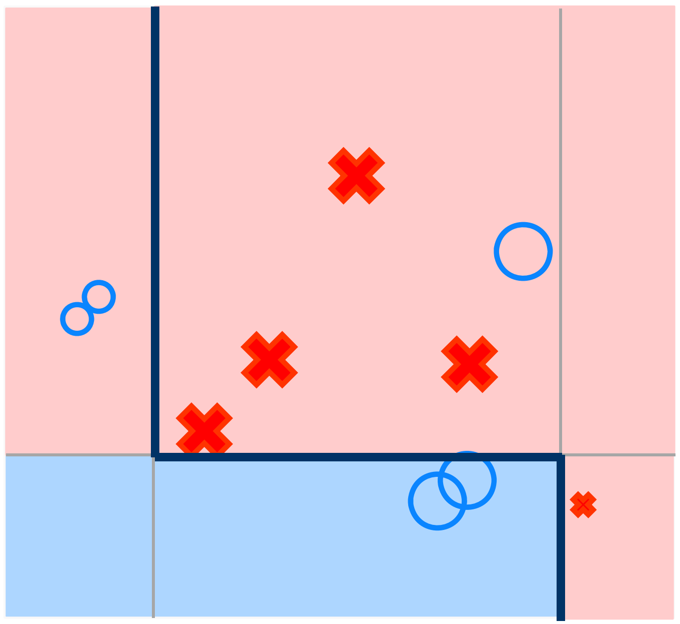
Boosting



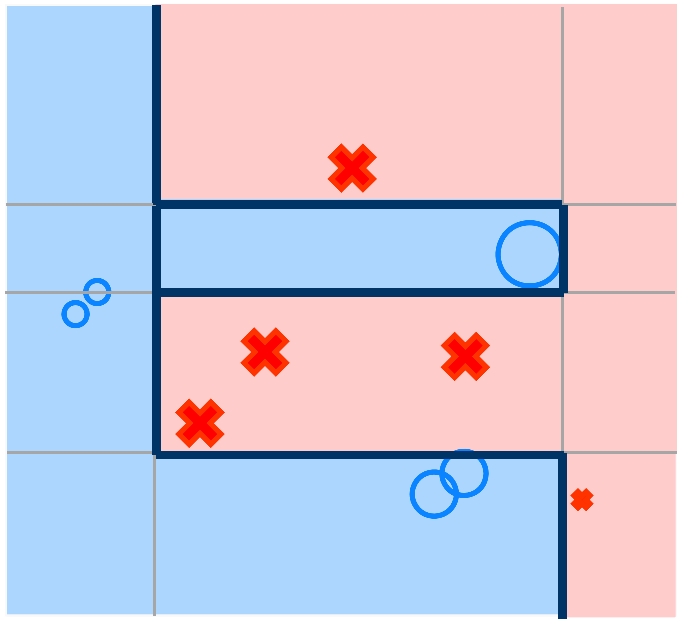
Boosting



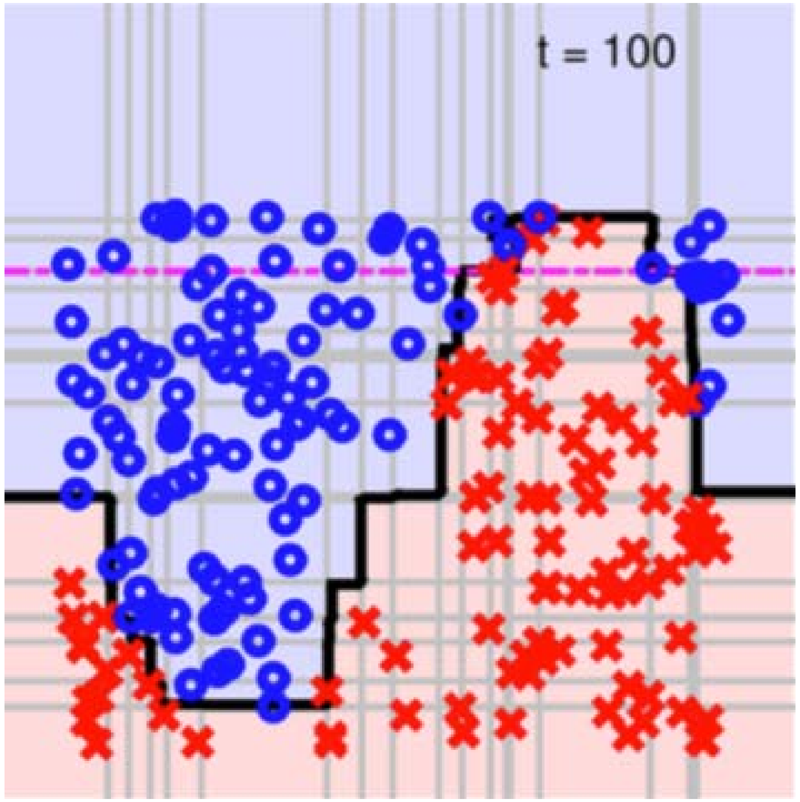
Boosting



Boosting



Boosting



Observation

- In order to come up an algorithm, we need to answer two underlying questions:
  - Question 1: How to change the weights of samples so that misclassified samples get more weight
  - Question 2: How to combine base learners in final phase

AdaBoost

- 1. Initialize the data weighting coefficients $\boldsymbol{w}$ by setting $w_n^{(1)} = 1/N$ for $n = 1, \dots, N$.

- 2. For $m = 1, \dots, M$:

  (a) Fit a classifier $y^{(m)}(\boldsymbol{x})$ to the training data by minimizing the weighted error function

$$J_m = \sum_{n=1}^{N} w_n^{(m)} I\left(y^{(m)}(\boldsymbol{x}_n) \neq t_n\right)$$

© Deng Cai, College of Computer Science, Zhejiang University

AdaBoost

(b) Evaluate the Errorrate:

$$\epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} I\left(y^{(m)}(\boldsymbol{x}_n) \neq t_n\right)}{\sum_{n=1}^{N} w_n^{(m)}}$$

and then use this to evaluate

$$\alpha_m = \ln \frac{1-\epsilon_m}{\epsilon_m}$$

Log odds, smaller the error rates, bigger this value.

(c) the data weighting coefficients

$$w_n^{(m+1)} = w_n^{(m)} \exp\{\alpha_m I\left(y^{(m)}(\boldsymbol{x}_n) \neq t_n\right)\} = \begin{Bmatrix} w_n^{(m)} \dfrac{1-\epsilon_m}{\epsilon_m}, if\ y^{(m)} makes\ error \\ w_n^{(m)}, otherwise \end{Bmatrix}$$

Answer to question 1, How to change the weights of samples so that misclassified samples get more weight.

AdaBoost

- 3. Make predictions using the final model, which is given by

$$Y_M(\boldsymbol{x}) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m \, y^{(m)}(\boldsymbol{x})\right)$$

Question 2: How to Combine base learners in final phase

weighting coefficients α_m give greater weight to the more accurate classifiers when computing the overall output

Insights Behind Adaboost

- In each round, the algorithm try to get a different base learner, so that model diversity is achieved

Re-weight for More Diverse Base Learner

$$g_m \longleftarrow \min_{h \in H} \left(\sum_{n=1}^{N} w_n^{(m)} I(y_n^{(m)} \neq t_n) \right)$$

$$g_{m+1} \longleftarrow \min_{h \in H} \left(\sum_{n=1}^{N} w_n^{(m+1)} I(y_n^{(m+1)} \neq t_n) \right)$$

if g_m '**not good**' for $\boldsymbol{w}^{(m+1)} \Rightarrow g_{m+1}$ diverse from g_m

Idea: construct $\boldsymbol{w}^{m+1}$ to make g_m **random-like**

$$\frac{\sum_{n=1}^{N} w_n^{(m+1)} I(y_n^{(m)} \neq t_n))}{\sum_{n=1}^{N} w_n^{(m+1)}} = \frac{1}{2} \qquad \frac{\sum_{n=1}^{N} w_n^{(m+1)} I(y_n^{(m)} \neq t_n))}{\sum_{n=1}^{N} w_n^{(m+1)} I(y_n^{(m)} \neq t_n)) + \sum_{n=1}^{N} w_n^{(m+1)} I(y_n^{(m)} = t_n))} = \frac{1}{2}$$

Solve this equation will give us

$$w_n^{(m+1)} = w_n^{(m)} \sqrt{\frac{1-\epsilon_m}{\epsilon_m}}$$

$$w_n^{(m+1)} = w_n^{(m)} \frac{1-\epsilon_m}{\epsilon_m}$$

$$w_n^{(m+1)} = w_n^{(m)} \exp\left\{-\frac{1}{2}\alpha_m t_n y^{(m)}(\boldsymbol{x}_n)\right\}$$

$$w_n^{(m+1)} = w_n^{(m)} \sqrt{\frac{\epsilon_m}{1-\epsilon_m}}$$

which is the same as update equation in Adaboost Algorithm

Insights Behind Adaboost

- In each round, the algorithm try to get a different base learner, so that model diversity is achieved

- Adaboost can be see as a sequential optimization process of an additive model under exponential error

Optimization of Exponential Loss

- Consider the exponential error function defined by

$$E = \sum_{n=1}^{N} \exp\{-t_n f_m(\boldsymbol{x}_n)\}$$

where $f_m(\boldsymbol{x})$ is a classifier defined in terms of a linear combination of base classifiers $y_l(\boldsymbol{x})$ of the form

$$f_m(\boldsymbol{x}) = \frac{1}{2}\sum_{l=1}^{m} \alpha_l y_l(\boldsymbol{x})$$

and $t_n \in \{-1, 1\}$ are the training set target values. Our goal is to minimize E with respect to both the weighting coefficients α_l and the parameters of the base classifiers $y_l(x)$.

- Instead of doing a global error function minimization, however, we shall suppose that the base classifiers $y_1(x), \ldots, y_{m-1}(x)$ are fixed, as are their coefficients $\alpha_1, \ldots, \alpha_{m-1}$, and so we are minimizing only with respect to α_m and $y_m(x)$.

Optimization of Exponential Loss

$$E=\sum_{n=1}^{N}\exp\{-t_n f_m(\boldsymbol{x}_n)\} \qquad f_m(\boldsymbol{x})=\frac{1}{2}\sum_{l=1}^{m}\alpha_l y_l(\boldsymbol{x})$$

- the base classifiers $y_1(x),\ldots,y_{m-1}(x)$ are fixed, as are their coefficients $\alpha_1,\ldots,\alpha_{m-1}$,

$$=\sum_{n=1}^{N}\exp\left\{-t_n f_{m-1}(\boldsymbol{x}_n)-\frac{1}{2}t_n\alpha_m y_m(\boldsymbol{x}_n)\right\}$$

$$=\sum_{n=1}^{N}\exp\{-t_n f_{m-1}(\boldsymbol{x}_n)\}\exp\left\{-\frac{1}{2}t_n\alpha_m y_m(\boldsymbol{x}_n)\right\}$$

$$=\sum_{n=1}^{N}w_n^{(m)}\exp\left\{-\frac{1}{2}t_n\alpha_m y_m(\boldsymbol{x}_n)\right\}$$

$$w_n^{(m+1)}=w_n^{(m)}\exp\left\{-\frac{1}{2}t_n\alpha_m y_m(\boldsymbol{x}_n)\right\} \qquad t_n y_m(\boldsymbol{x}_n)=1-2I\left(y^{(m)}(\boldsymbol{x}_n)\neq t_n\right)$$

$$w_n^{(m+1)}=w_n^{(m)}\exp\left\{-\frac{\alpha_m}{2}\right\}\exp\left\{\alpha_m I\left(y^{(m)}(\boldsymbol{x}_n)\neq t_n\right)\right\}$$

Optimization of Exponential Loss

- If we denote by T_m the set of data points that are correctly classified by $y_m(x)$, and if we denote the remaining misclassified points by M_m, then we can in turn rewrite the error function in the form.

$$E = e^{-\frac{\alpha_m}{2}} \sum_{n \in T_m} w_n^{(m)} + e^{\frac{\alpha_m}{2}} \sum_{n \in M_m} w_n^{(m)}$$

$$= \left(e^{\frac{\alpha_m}{2}} - e^{-\frac{\alpha_m}{2}}\right) \sum_{n=1}^{N} w_n^{(m)} I(y_m(x_n) \neq t_n) + e^{-\frac{\alpha_m}{2}} \sum_{n=1}^{N} w_n^{(m)}$$

$$\boxed{J_m = \sum_{n=1}^{N} w_n^{(m)} I\left(y^{(m)}(\boldsymbol{x}_n) \neq t_n\right)}$$

When we minimize this with respect to $y_m(x)$, we see that the second term is constant, and so this is equivalent to the procedure in Adaboost Algorithm 2-(a) and we get ε_m which is the same as in Adaboost. Similarly, minimizing with respect to α_m, we get the α_m the same as α_m in Adaboost.

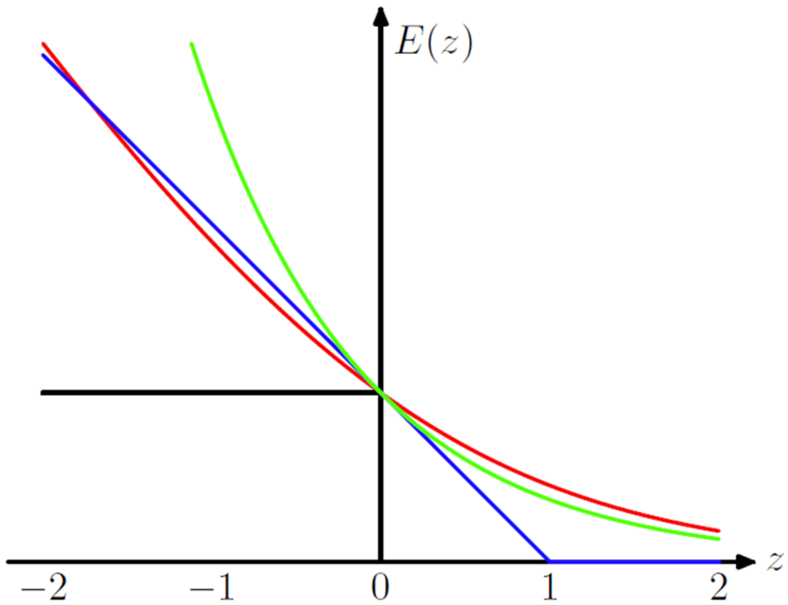
Loss functions for boosting

- The exponential loss function that is minimized by the AdaBoost algorithm is

$$E = \sum_{n=1}^{N} \exp\{-t_n f_m(\boldsymbol{x}_n)\}$$

- Why these loss function?
- What properties do this loss function have?
- Can we choose other loss functions?
- Those questions opens the door to a wide range of boosting-like algorithms, including multiclass extensions, by altering the choice of loss function. It also motivates the extension to regression problems

Comparing to Other Loss Function



- Comparing exponential loss, logistic loss, and hinge loss

- Error function grows exponentially with $|ty(x)|$, it penalizes large negative values of $ty(x)$, thus be much less robust to outliers or misclassified data

Summary

- They are two different Ensemble Paradigms
  - Boosting is **sequential ensemble methods,** where the base learners are generated sequentially.
  - Bagging is **parallel ensemble methods:** where the base learners are generated in parallel.
  - Boosting exploit the ***dependence*** between the base learners, since the overall performance can be boosted in a residual-decreasing way.
  - Bagging exploit the ***independence*** between the base learners, since the error can be reduced dramatically by combining independent base learners.

35