

CSC411 Fall 2014
Machine Learning & Data Mining

Ensemble Methods

Slides by Rich Zemel

Ensemble methods

Typical application: classification

Ensemble of classifiers is a set of classifiers whose individual decisions combined in some way to classify new examples

Simplest approach:

1. Generate multiple classifiers
2. Each votes on test instance
3. Take majority as classification

Classifiers different due to different sampling of training data, or randomized parameters within the classification algorithm

Aim: take simple mediocre algorithm and transform it into a super classifier without requiring any fancy new algorithm

Ensemble methods: Summary

Differ in training strategy, and combination method

1. Parallel training with different training sets: **bagging**
2. Sequential training, iteratively re-weighting training examples so current classifier focuses on hard examples: **boosting**
3. Parallel training with objective encouraging division of labor: **mixture of experts**

Notes:

- Also known as **meta-learning**
- Typically applied to weak models, such as decision stumps (single-node decision trees), or linear classifiers

Variance-bias tradeoff?

Minimize two sets of errors:

1. **Variance**: error from sensitivity to small fluctuations in the training set
2. **Bias**: erroneous assumptions in the model

Variance-bias decomposition is a way of analyzing the generalization error as a sum of 3 terms: variance, bias and irreducible error (resulting from the problem itself)

Why do ensemble methods work?

Based on one of two basic observations:

1. **Variance reduction**: if the training sets are completely independent, it will always help to average an ensemble because this will reduce variance without affecting bias (e.g., bagging) -- reduce sensitivity to individual data pts
2. **Bias reduction**: for simple models, average of models has much greater capacity than single model (e.g., hyperplane classifiers, Gaussian densities). Averaging models can reduce bias substantially by increasing capacity, and control variance by fitting one component at a time (e.g., boosting)

Ensemble methods: Justification

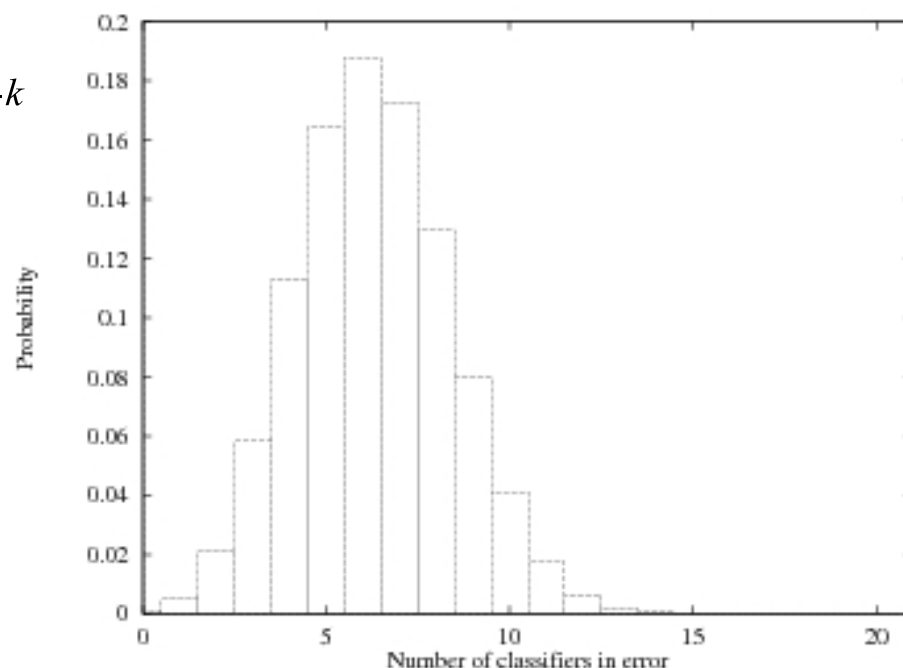
Ensemble methods more accurate than any individual members:

- Accurate (better than guessing)
- Diverse (different errors on new examples)

Independent errors: prob k of N classifiers (independent error rate ε) wrong:

$$P(\# \text{ errors} = k) = \binom{N}{k} \varepsilon^k (1 - \varepsilon)^{N-k}$$

Probability that majority vote wrong: error under distribution where more than $N/2$ wrong



Ensemble methods: Netflix

Clear demonstration of the power of ensemble methods

Original progress prize winner (BellKor) was ensemble of 107 models!

“Our experience is that most efforts should be concentrated in deriving substantially different approaches, rather than refining a simple technique.”

“We strongly believe that the success of an ensemble approach depends on the ability of its various predictors to expose different complementing aspects of the data. Experience shows that this is very different than optimizing the accuracy of each individual predictor.”

Bootstrap estimation

- Repeatedly draw n samples from D
- For each set of samples, estimate a statistic
- The bootstrap estimate is the mean of the individual estimates
- Used to estimate a statistic (parameter) and its variance
- Bagging: **b**ootstrap **a**ggregation (Breiman 1994)

Bagging

Simple idea: generate M bootstrap samples from your original training set. Train on each one to get y_m , and average them

$$y_{bag}^M(\mathbf{x}) = \frac{1}{M} \sum_{m=1}^M y_m(\mathbf{x})$$

For regression: average predictions

For classification: average class probabilities (or take the majority vote if only hard outputs available)

Bagging approximates the Bayesian posterior mean. The more bootstraps the better, so use as many as you have time for

Each bootstrap sample is drawn *with replacement*, so each one contains some duplicates of certain training points and leaves out other training points completely

Cross-validated committees

Bagging works well for *unstable* algorithms: can change dramatically with small changes in training data

But can hurt a stable algorithm: a Bayes optimal algorithm may leave out some training examples in every bootstrap

Alternative method based on different training examples: **cross-validated committees:**

- Here k disjoint subsets of data are left out of training sets
- Again uses majority for combination

Boosting

Also works by manipulating training set, but classifiers trained sequentially

Each classifier trained given knowledge of the performance of previously trained classifiers: **focus on hard examples**

Final classifier: weighted sum of component classifiers

Making weak learners stronger

- Suppose you have a weak learning module (a “base classifier”) that can always get $0.5 + \epsilon$ correct when given a two-way classification task
 - That seems like a weak assumption but beware!
- Can you apply this learning module many times to get a strong learner that can get close to zero error rate on the training data?
 - Theorists showed how to do this and it actually led to an effective new learning procedure (Freund & Shapire, 1996)

Boosting (ADABOOST)

- First train the base classifier on all the training data with equal importance weights on each case.
- Then re-weight the training data to emphasize the hard cases and train a second model.
 - How do we re-weight the data?
- Keep training new models on re-weighted data
- Finally, use a weighted committee of all the models for the test data.
 - How do we weight the models in the committee?

Boosting

- Probably one of the **most influential ideas** in machine learning in the last decade.
- In the PAC framework, boosting is a way of converting a “**weak**” learning model (behaves slightly better than chance) into a “**strong**” learning mode (behaves arbitrarily close to perfect).
- Strong theoretical result, but also lead to a very powerful and practical algorithm which is used all the time in real world machine learning.
- Basic idea, for binary classification with $t_n = \pm 1$.

$$y_{boost} = \text{sign} \left(\sum_{m=1}^M \alpha_m y_m(\mathbf{x}) \right),$$

where $y_m(\mathbf{x})$ are models trained with **reweighted** datasets D_m , and the weights α_m are non-negative.

How to train each classifier

input : \mathbf{x} , output : $y(\mathbf{x}) \in \{1, -1\}$

target : $t \in \{1, -1\}$,


weight on case n for classifier m : w_n^m


Cost function for classifier m :

$$J_m = \sum_{n=1}^N w_n^m [y_m(\mathbf{x}_n) \neq t_n] = \sum \text{weighted errors}$$

↑
1 if error,
0 if correct

How to weight each training case for classifier m

Let $\epsilon_m = \frac{J_m}{\sum_n w_n^m}$  weighted error rate of classifier

Let $\alpha_m = \ln\left(\frac{1 - \epsilon_m}{\epsilon_m}\right)$  This is the **quality of the classifier**. It is zero if the classifier has weighted error rate of 0.5 and infinity if the classifier is perfect

$$w_n^{m+1} = w_n^m \exp\left\{ \alpha_m [y_m(x_n) \neq t_n] \right\}$$

How to weight each training case for classifier m

- Weight the binary prediction of each classifier by the quality of that classifier:

$$Y_M(\mathbf{x}) = \textit{sign} \left(\sum_{m=1}^M \alpha_m y_m(\mathbf{x}) \right)$$

AdaBoost Algorithm

- Initialize the data weights $w_n = 1/N$.
- For $m=1, \dots, M$:
 - Fit a classifier $y_m(\mathbf{x})$ to the training data by minimizing the weighted error function:

$$J_m = \sum_{n=1}^N w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n),$$

where $I(y_m(\mathbf{x}_n) \neq t_n)$ is the indicator function and equals to one when $y_m(\mathbf{x}_n) \neq t_n$ and zero otherwise.

- Evaluate:

$$\alpha_m = \ln \frac{1 - \epsilon_m}{\epsilon_m}, \quad \epsilon_m = \frac{\sum_{n=1}^N w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n)}{\sum_{n=1}^N w_n^{(m)}}.$$

Weighting coefficients.

weighted measures of the error rates.

AdaBoost Algorithm

- Initialize the data weights $w_n = 1/N$.
- For $m=1, \dots, M$:
 - Fit a classifier $y_m(\mathbf{x})$ to the training data by minimizing:

$$J_m = \sum_{n=1}^N w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n),$$

- Evaluate:

$$\alpha_m = \ln \frac{1 - \epsilon_m}{\epsilon_m}, \quad \epsilon_m = \frac{\sum_{n=1}^N w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n)}{\sum_{n=1}^N w_n^{(m)}}.$$

- Update the data weights:

$$w_n^{(m+1)} = w_n^{(m)} \exp(\alpha_m I(y_m(\mathbf{x}_n) \neq t_n)).$$

- Make predictions using the final model:

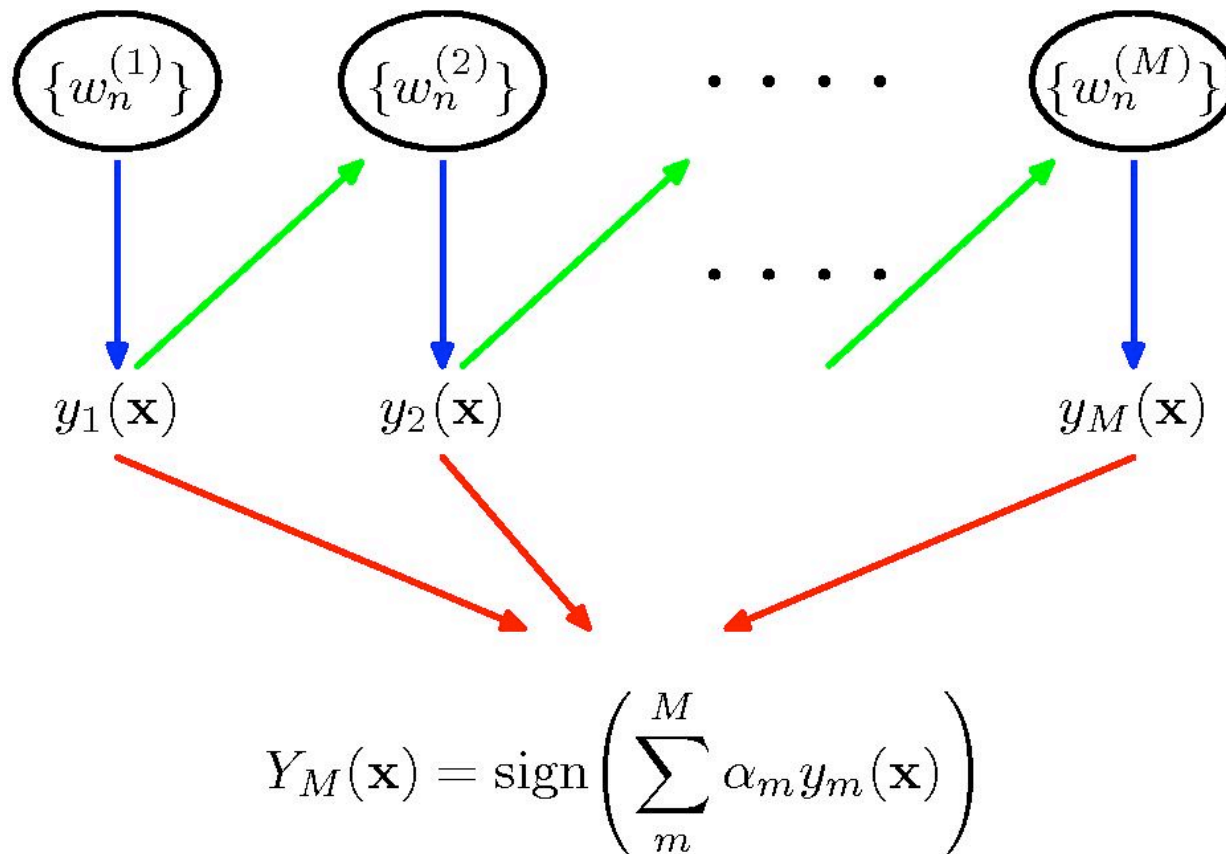
$$Y_M(\mathbf{x}) = \text{sign} \left(\sum_{m=1}^M \alpha_m y_m(\mathbf{x}) \right).$$

Some Intuitions

- The first classifier corresponds to the usual procedure for training a single classifier.
- At each round, boosting:
 - increases the weight on those examples the last classifier got wrong,
 - decreases the weight on those it got right.
- Over time, AdaBoost focuses on the examples that are consistently difficult and forgets about the ones that are consistently easy.
- The weight each intermediate classifier gets in the final ensemble depends on the error rate it achieved on its weighted training set at the time it was created.
- Hence the weighting coefficients α_m give greater weight to more accurate classifiers.

Some Intuitions

- Schematic illustration of AdaBoost:

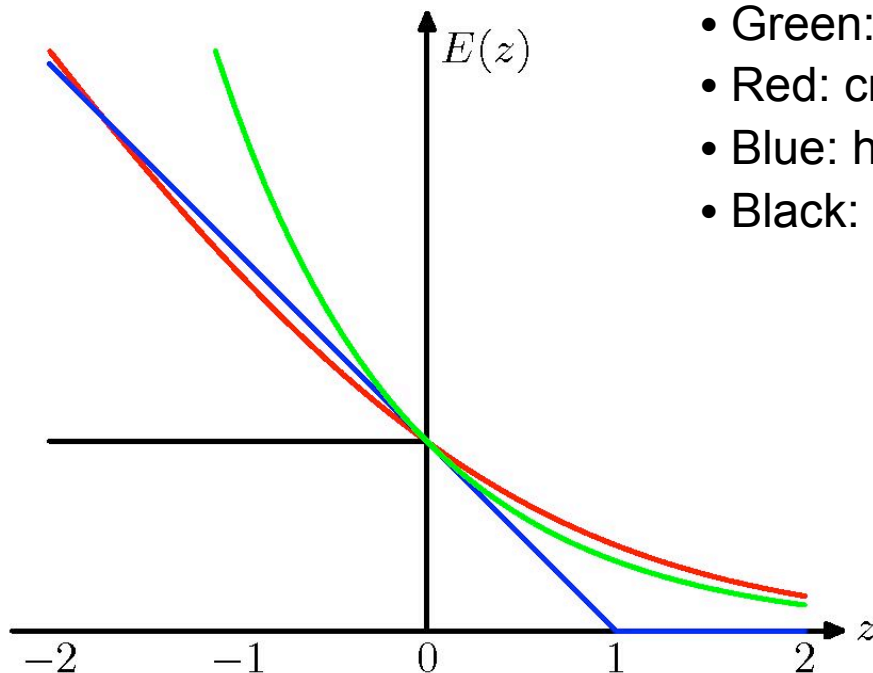


Exponential Loss

- One explanation, which helps a lot to understand how boosting really works, is that classification boosting is equivalent to **sequential minimization** of the following loss (error) function:

$$L(t, f(\mathbf{x})) = \exp(-tf(\mathbf{x})).$$

- This is called **exponential loss** and it is very similar to other kinds of loss, e.g. classification loss.



- Green: exponential
- Red: cross-entropy
- Blue: hinge loss
- Black: misclassifications error (0-1 loss)

Problem Setup

- Consider the exponential error:

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

where $f_m(\mathbf{x})$ is a classifier defined in terms of **linear combination of base classifiers**:

$$f_m(\mathbf{x}) = \frac{1}{2} \sum_{k=1}^m \alpha_k y_k(\mathbf{x}),$$

and $t_n = \pm 1$.

- Our goal is to minimize this objective with respect to parameters of the base classifiers and coefficients α .


Boosting as Forward Additive Modeling

- Suppose that the base classifiers: $y_1(\mathbf{x}), \dots, y_{m-1}(\mathbf{x})$ and their coefficients $\alpha_1, \dots, \alpha_{m-1}$ are fixed.
- We minimize only with respect to α_m and $y_m(\mathbf{x})$.

Remember:

$$f_m(\mathbf{x}) = \frac{1}{2} \sum_{k=1}^m \alpha_k y_k(\mathbf{x}),$$

$$\begin{aligned} E &= \sum_{n=1}^N \exp(-t_n f_m(\mathbf{x}_n)), \\ &= \sum_{n=1}^N \exp \left(-t_n f_{m-1}(\mathbf{x}_n) - \frac{1}{2} t_n \alpha_m y_m(\mathbf{x}_n) \right) \\ &= \sum_{n=1}^N w_n^{(m)} \exp \left(-\frac{1}{2} t_n \alpha_m y_m(\mathbf{x}_n) \right). \end{aligned}$$



fixed optimize

where we defined:

$$w_n^{(m)} = \exp(-t_n f_{m-1}(\mathbf{x}_n)).$$

Boosting as Forward Additive Modeling

- Let A be the set of points that are correctly classified by $y_m(\mathbf{x})$, and B be the set of points that are misclassified by $y_m(\mathbf{x})$.

$$\begin{aligned} E &= e^{-\alpha_m/2} \sum_{n \in A} w_n^{(m)} + e^{\alpha_m/2} \sum_{n \in B} w_n^{(m)} \\ &= \left(e^{\alpha_m/2} - e^{-\alpha_m/2} \right) \sum_{n=1}^N w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n) + e^{-\alpha_m/2} \sum_{n=1}^N w_n^{(m)}. \end{aligned}$$

- So minimizing with respect to $y_m(\mathbf{x})$ is equivalent to minimizing:

$$J_m = \sum_{n=1}^N w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n),$$

- and minimizing with respect to α_m leads to:

$$\alpha_m = \ln \frac{1 - \epsilon_m}{\epsilon_m}, \quad \epsilon_m = \frac{\sum_{n=1}^N w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n)}{\sum_{n=1}^N w_n^{(m)}}.$$

Updating the Weighting Coefficients

- The weights on the data points are updated as:

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

Remember:


$$w_n^{(m)} = \exp(-t_n f_{m-1}(\mathbf{x}_n)).$$

- Using the following identity:

$$t_n y_m(\mathbf{x}_n) = 1 - 2I(y_m(\mathbf{x}_n) \neq t_n),$$

we obtain:

$$w_n^{(m+1)} = w_n^{(m)} \exp(\alpha_m/2) \exp(\alpha_m I(y_m(\mathbf{x}_n) \neq t_n)).$$

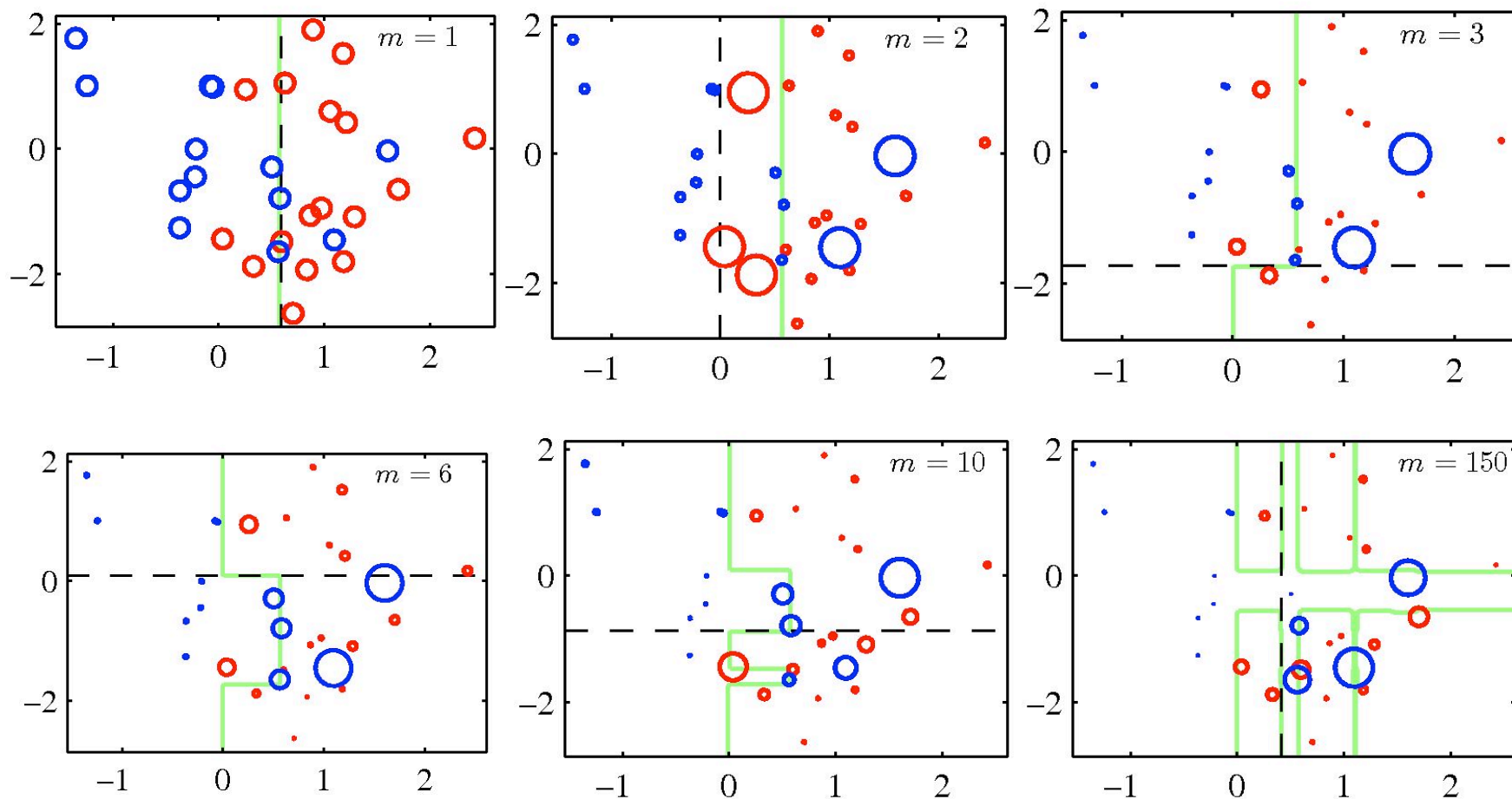


Does not depend on n, just
rescales all the weights

- This is the update performed by the AdaBoost.

Example

- Base learners are simple thresholds applied to one or another axis.



An impressive example of boosting

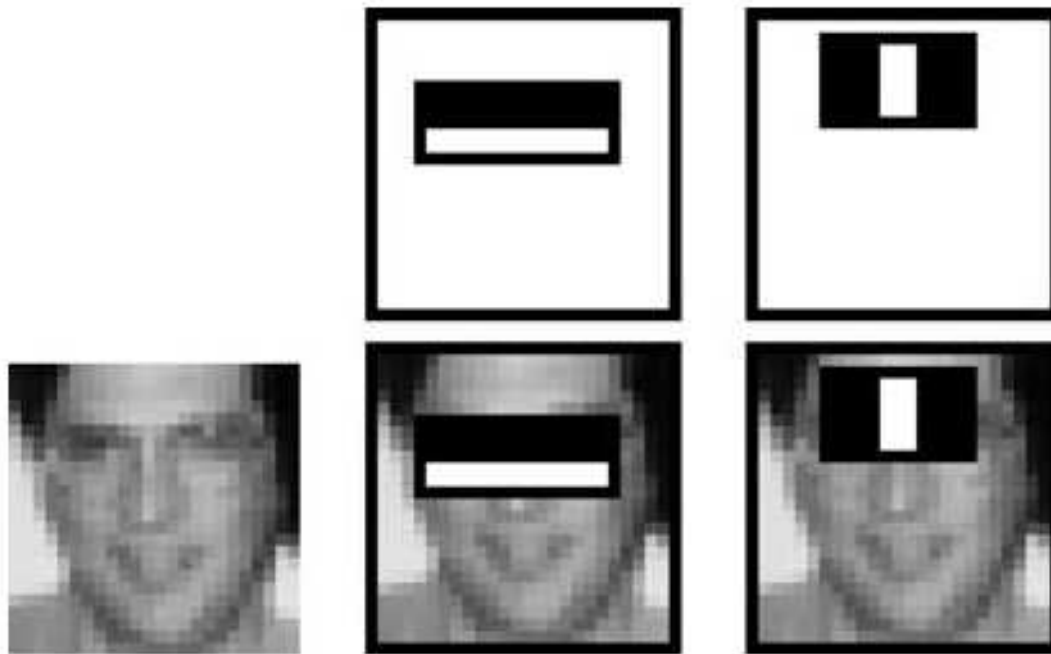
- Viola and Jones created a very fast face detector that can be scanned across a large image to find the faces.
- The base classifier/weak learner just compares the total intensity in two rectangular pieces of the image.
 - There is a neat trick for computing the total intensity in a rectangle in a few operations.
 - So its easy to evaluate a huge number of base classifiers and they are very fast at runtime.
 - The algorithm adds classifiers greedily based on their quality on the weighted training cases.

AdaBoost in face detection

Famous application of boosting: detecting faces in images

Two twists on standard algorithm

- 1) Pre-define weak classifiers, so optimization=selection
- 2) Change loss function for weak learners: false positives less costly than misses



AdaBoost face detection results



What are the base classifiers?

Popular choices of base classifier for boosting and other ensemble methods:

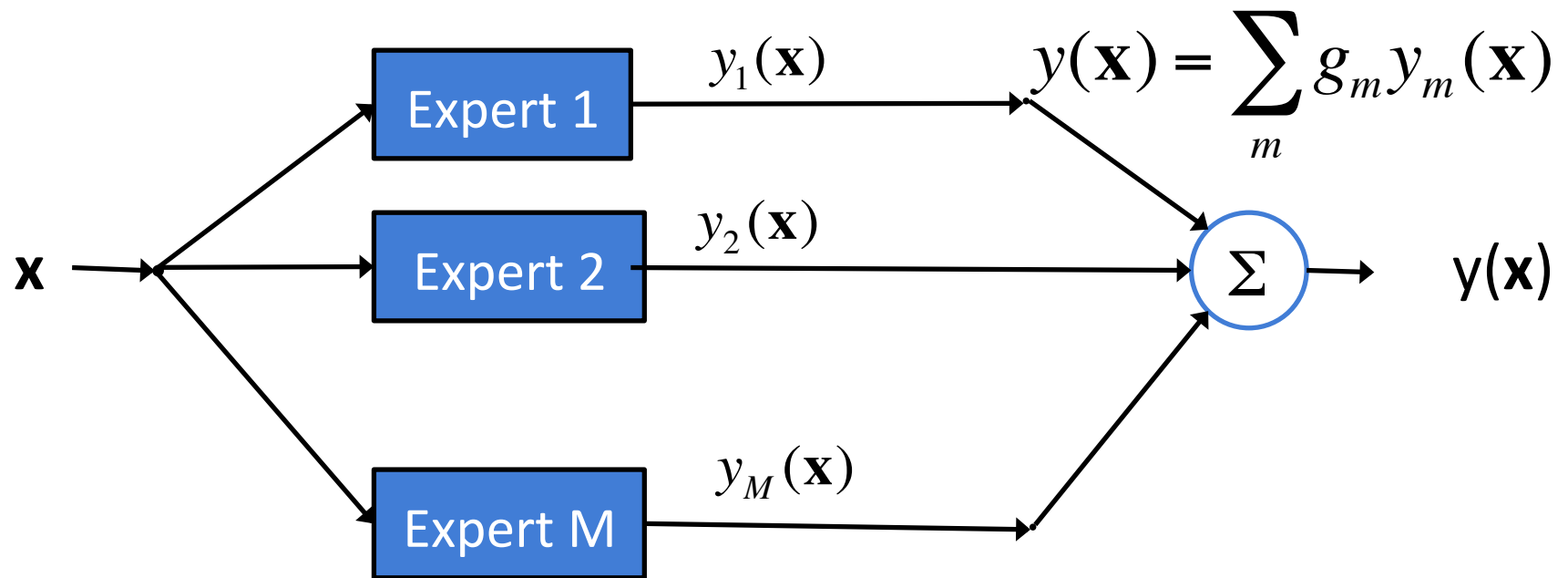
- Linear classifiers
- Decision trees

Random/Decision Forests

- Definition: Ensemble of decision trees
- Algorithm:
 - Divide training examples into multiple training sets (bagging)
 - Train a decision tree on each set (can randomly select subset of variables to consider)
 - Aggregate the predictions of each tree to make classification decision (e.g., can choose mode vote)

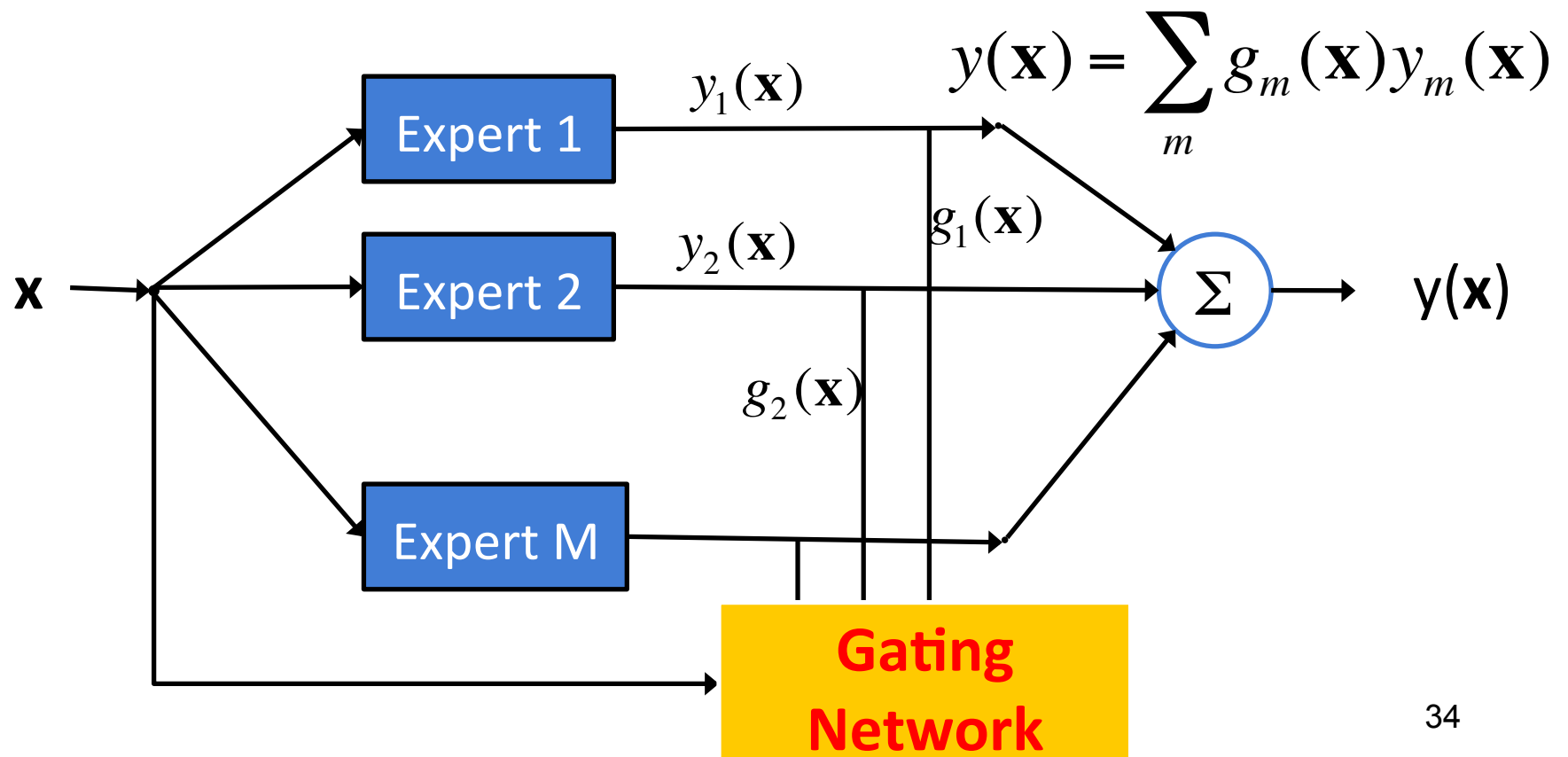
Ensemble learning: Boosting and Bagging

- Experts cooperate to predict output
- Vote of each expert has consistent weight for each test example



Mixture of Experts

- Weight of each expert not constant – depends on input \mathbf{x}
- Gating network encourages specialization (local experts) instead of cooperation



Mixture of Experts: Summary

1. Cost function designed to make each expert estimate desired output independently
2. Gating network softmax over experts: stochastic selection of who is the true expert for given input
3. Allow each expert to produce distribution over outputs

Cooperation vs. Specialization

- Consider regression problem
- To encourage cooperation, can train to reduce discrepancy between average of predictors with target

$$E = (t - \frac{1}{M} \sum_m y_m)^2$$

- This can overfit badly. It makes the model much more powerful than training each predictor separately.
- Leads to odd objective: consider adding models/experts sequentially – if its estimate for t is too low, and the average of other models is too high, then model m encouraged to *lower* its prediction

Cooperation vs. Specialization

- To encourage specialization, train to reduce the average of each predictor's discrepancy with target

$$E = \frac{1}{M} \sum_m (t - y_m)^2$$

- use a weighted average: weights are probabilities of picking that “expert” for the particular training case.

$$E = \frac{1}{M} \sum_m g_m(\mathbf{x})(t - y_m(\mathbf{x}))^2$$

- Gating output is softmax of $\mathbf{z} = \mathbf{U}\mathbf{x}$

$$g_m(\mathbf{x}) = \exp(z_m(\mathbf{x})) / \sum_{m'} \exp(z_{m'}(\mathbf{x}))$$

Derivatives of simple cost function

- Look at derivatives to see what cost function will do

$$E = \frac{1}{M} \sum_m g_m(\mathbf{x})(t - y_m(\mathbf{x}))^2$$

- For gating network, increase weight on expert when its error less than average error of experts

$$\frac{\partial E}{\partial y_m} = \frac{2}{M} g_m(\mathbf{x})(t - y_m(\mathbf{x}))$$

$$\frac{\partial E}{\partial z_m} = \frac{2}{M} g_m(\mathbf{x})[(t - y_m(\mathbf{x}))^2 - E]$$

Mixture of Experts: Final cost function

- Can improve cost function by allowing each expert to produce not just single value estimate, but distribution
- Result is a mixture model

$$p(y | MOE) = \sum_m g_m(\mathbf{x}) \mathbf{N}(y | y_m(\mathbf{x}), \Sigma)$$

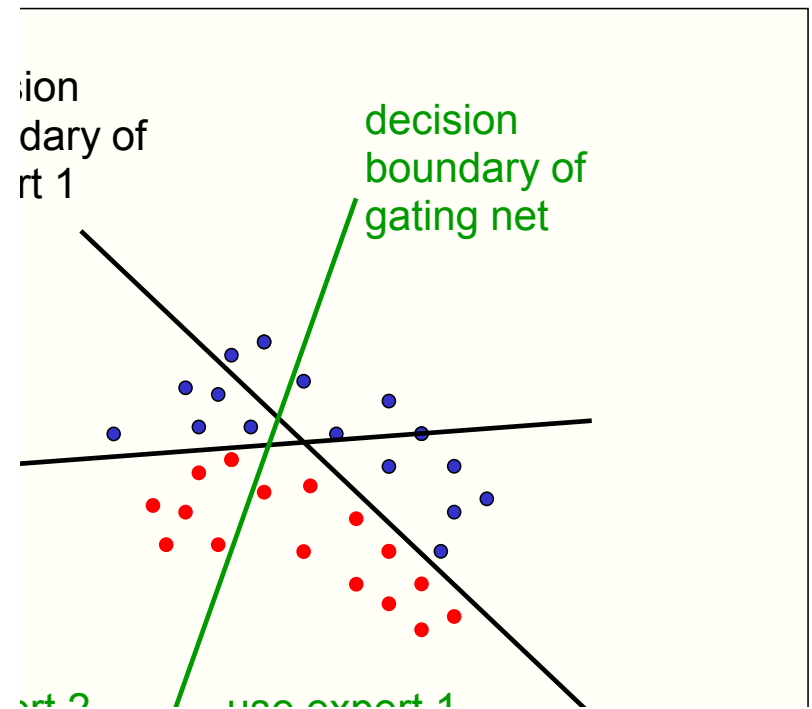
$$-\log p(t | MOE) = -\log \sum_m g_m(\mathbf{x}) \exp(-\|t - y_m(\mathbf{x})\|^2 / 2)$$

- Gradient: Error weighted by posterior probability of the expert

$$\frac{\partial E}{\partial y_m} = -2 \frac{g_m(\mathbf{x}) \exp(-\|t - y_m(\mathbf{x})\|^2 / 2)}{\sum_{m'} g_{m'}(\mathbf{x}) \exp(-\|t - y_{m'}(\mathbf{x})\|^2 / 2)} (t - y_m(\mathbf{x}))$$

Mixture of Experts: Summary

1. Cost function designed to make each expert estimate desired output independently
2. Gating network softmax over experts: stochastic selection of who is the true expert for given input
3. Allow each expert to produce distribution over outputs



Ensemble methods: Summary

Differ in training strategy, and combination method

- Parallel training with different training sets: **bagging** (bootstrap aggregation) – train separate models on overlapping training sets, average their predictions
- Sequential training, iteratively re-weighting training examples so current classifier focuses on hard examples: **boosting**
- Parallel training with objective encouraging division of labor: **mixture of experts**

Notes:

- Differ in: training strategy; selection of examples; weighting of components in final classifier