# STA 4273H: Statistical Machine Learning

Russ Salakhutdinov

Department of Statistics
rsalakhu@utstat.toronto.edu
http://www.utstat.utoronto.ca/~rsalakhu/
Sidney Smith Hall, Room 6002

Lecture 12

# Combining Models

- In practice, it is often found that one can improve performance by combining multiple models, instead of using a single model.
- Example: We may train K different models and then make predictions using the average of predictions made by each model.
- Such combinations of models are called **committees**.
- One important variant of the committee method is called **boosting**.
- Another approach is to use different models in different regions of the input space.
- One widely used framework is known as a decision tree.
- One can take a probabilistic approach -- mixture of experts framework.
- The hope of "**meta-learning**" is that it can "supercharge" a mediocre learning algorithm into an excellent learning algorithm.

# Model Averaging

- It is useful to distinguish between: Bayesian model averaging and model combination.

- Example: Consider a mixture of Gaussians:

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k).$$

- Hence for i.i.d. data:

$$p(\mathbf{X}) = \prod_{n=1}^{N} \sum_{\mathbf{z}_n} p(\mathbf{x}_n, \mathbf{z}_n).$$

- This is an example of model combination.

- Different data points **within the same dataset** can be generated from different values of the latent variables (or by different components).

# Bayesian Model Averaging

- Suppose we have several different models, indexed by h=1,..,H, with prior probabilities p(h).

- Example: one model can be a mixture of Gaussians, another one can be a mixture of Cauchy distributions.

- The marginal over the dataset is:

$$p(\mathbf{X}) = \sum_{h=1}^{H} p(\mathbf{X}|h)p(h).$$

- This is an example of the Bayesian model averaging.

- Interpretation: Just one model is responsible for generating the whole dataset!
- The distribution over h reflects our uncertainty as to which model that is.

- As we observe more data, the uncertainty decreases, and the posterior p(h|**X**) becomes focused on just one model.

- The same reasoning apply for the conditional distributions p(t|x,**X**,**T**).

# Committees

- Average the predictions of a set of individual models.
- Motivation: Bias-variance trade-off:
  - bias: different between the model and the true function to be predicted.
  - variance: sensitivity of the model due to the given dataset.

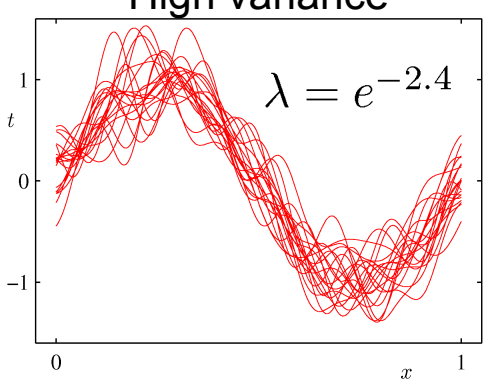
$$\text{expected loss} = (\text{bias})^2 + \text{variance} + \text{noise}$$

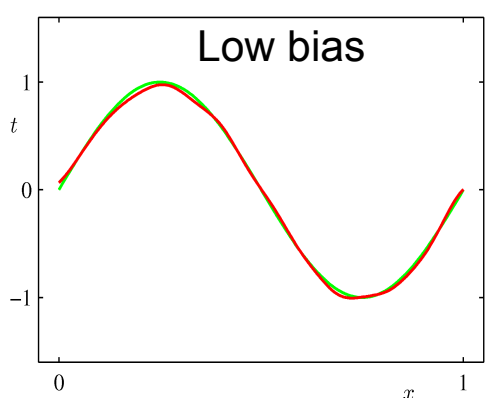
Average predictions over all datasets differ from the optimal regression function.

Solutions for individual datasets vary around their averages -- how sensitive is the function to the particular choice of the dataset.

Intrinsic variability of the target values.

High variance




- When we average a set of low-bias models (e.g. higher-order polynomials, we obtain accurate predictions.

# Bagging

- **Bagging** = Bootstrap aggregation.

- In practice, we only have one dataset: Need a way to introduce variability between different models.

- One idea: Generate M bootstrap samples from your original training set and train B separate models.
  - For regression, average predictions.
  - For regression, average class probabilities (or take the majority vote if only hard outputs available).

- The size of each bootstrap sample is equal to the size of the original training set, but they are drawn with replacement, so each one contains some duplicates of certain training points and leaves out other training points completely.

# Variance Reduction by Averaging

- Suppose we M bootstrap datasets and train M models $y_m(\mathbf{x})$.
- The committee prediction is given by:

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

- Assume that the true function is h(**x**), hence

$$y_m(\mathbf{x}) = h(\mathbf{x}) + \epsilon_m(\mathbf{x}).$$

Expectation with respect to the distribution over the input vector **x**.

- The average sum-of-squares error takes the form:

$$\mathbb{E}_{\mathbf{x}}\left[(y_m(\mathbf{x}) - h(\mathbf{x}))^2\right] = \mathbb{E}_{\mathbf{x}}\left[\epsilon_m(\mathbf{x})^2\right].$$

- The average error made by the models acting individually is therefore:

$$E_{AV} = \frac{1}{M}\sum_{m=1}^{M} \mathbb{E}_{\mathbf{x}}\left[\epsilon_m(\mathbf{x})^2\right].$$

# Variance Reduction by Averaging

- The committee prediction is given by:

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

- The expected error from the committee is given by:

$$\begin{aligned} E_{COM} &= \mathbb{E}_{\mathbf{x}}\left[\left(\frac{1}{M}\sum_{m=1}^{M} y_m(\mathbf{x}) - h(\mathbf{x})\right)^2\right] \\ &= \mathbb{E}_{\mathbf{x}}\left[\left(\frac{1}{M}\sum_{m=1}^{M} \epsilon_m(\mathbf{x})\right)^2\right]. \end{aligned}$$

- Assuming the errors are uncorrelated:

$$\begin{aligned} \mathbb{E}_{\mathbf{x}}[\epsilon_m(\mathbf{x})] &= 0 \\ \mathbb{E}_{\mathbf{x}}[\epsilon_m(\mathbf{x})\epsilon_k(\mathbf{x})] &= 0, \end{aligned} \longrightarrow E_{COM} = \frac{1}{M}E_{AV}.$$

# Variance Reduction by Averaging

- Hence we have:

$$E_{COM} = \frac{1}{M} E_{AV}.$$

- This dramatic result suggests that the average error of a model **can be reduced by a factor of M** simply by averaging M versions of the models.

- Too good to be true!

- The above result depends on the key assumption that the errors of the individual models are uncorrelated.

- In practice, the errors will be highly correlated (remember, we are using bootstrap datasets).

# Why do Committees Work?

- All committee learning (often called meta-learning) is based on one of two observations:
  - **Variance Reduction**: If we had completely independent training sets it always helps to average together an ensemble of learners because this reduces variance without changing bias.
  - **Bias Reduction**: For many simple models, a weighted average of those models (in some space) has much greater capacity than a single model (e.g. hyperplane classifiers, single-layer networks). Averaging models can often reduce bias substantially by increasing capacity; we can keep variance low by only fitting one member of the mixture at a time.
- Either reduces variance substantially without affecting bias (**bagging**), or vice versa (**boosting**).

# Finite Bagging Can Hurt

- Bagging helps when a learning algorithm is good on average but unstable with respect to the training set.
- But if we bag a stable learning algorithm, we can actually make it worse. (For example, if we have a Bayes optimal algorithm, and we bag it, we might leave out some training samples in every bootstrap, and so the optimal algorithm will never be able to see them.)
- Bagging almost always helps with regression, but even with unstable learners it can hurt in classification. If we bag a poor and unstable classifier we can make it horrible.
- Example: true class = A for all inputs.
Our learner guesses class A with probability 0.4 and class B with probability 0.6 regardless of the input. (Very unstable).
It has error 0.6. But if we bag it, it will have error 1.

# 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 $\alpha_m$ are non-negative.

# AdaBoost Algorithm

- Initialize the data weights $w_n = 1/N$.
- For m=1,..,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:

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

Weighting coefficients.

weighted measures of the error rates.

# AdaBoost Algorithm

- Initialize the data weights $w_n = 1/N$.
- For m=1,..,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:

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

  - Update the data weights:

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

- Make predictions using the final model:

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

# 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.

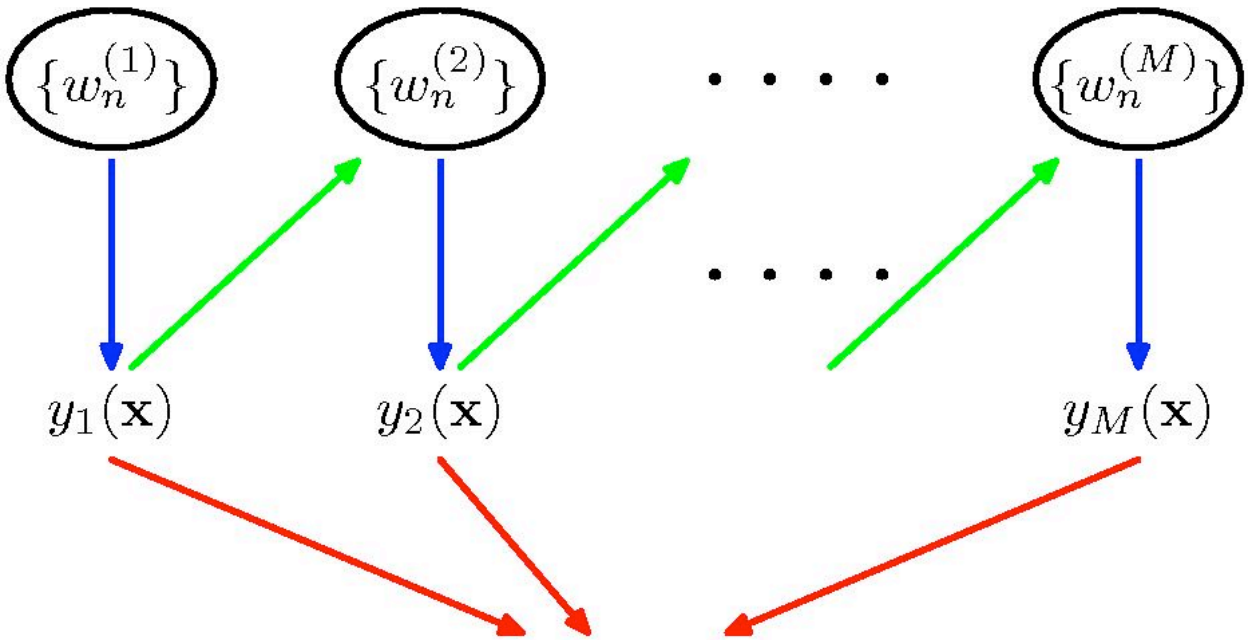
- Over time, AdaBoost focuses on the examples that are consistently difficult and forgets about the ones that are consistently easy.

- 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 $\alpha_m$ give greater weight to more accurate classifiers.

# Some Intuitions

- Schematic illustration of AdaBoost:



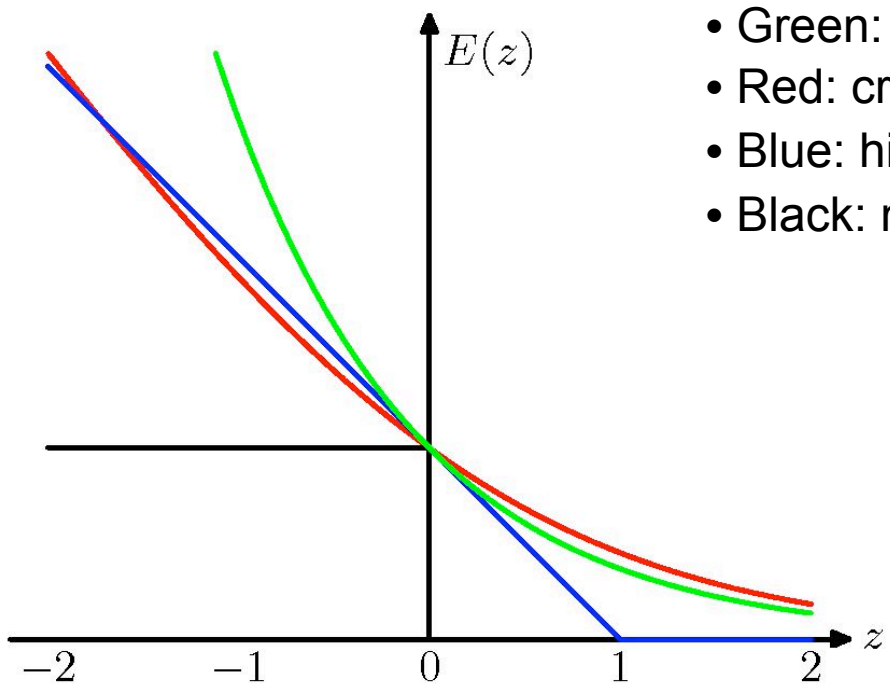

$$Y_M(\mathbf{x}) = \text{sign}\left(\sum_m^M \alpha_m y_m(\mathbf{x})\right)$$

# 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 an 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 $\alpha$.

# Boosting as Forward Additive Modeling

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

Remember:

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

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

fixed

optimize

where we defined:

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

# 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 $\alpha_m$ leads to:

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

# Updating the Weighting Coefficients

- The weights on the data points are updated as:

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

Remember:

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

- Using the following identity:

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

we obtain:

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

Does not depend on n, just rescales all the weights

- This is the update performed by the AdaBoost.

# More on Exponential Loss

- Exponential loss is very similar to other classification losses.
- Consider the expected error:

$$\mathbb{E}_{t,\mathbf{x}}\big[\exp(-ty(\mathbf{x}))\big] = \sum_t \int \exp(-ty(\mathbf{x}))p(t|\mathbf{x})p(\mathbf{x})d\mathbf{x}.$$

- Minimizing with respect to all possible functions, we obtain:

$$y(\mathbf{x}) = \frac{1}{2}\ln\left(\frac{p(t=1|\mathbf{x})}{p(t=-1|\mathbf{x})}\right),$$

  which is half of log-odds.

- Another loss function with the same population minimizer is the binomial negative log-likelihood:

$$-\log(1+\exp(-2ty(\mathbf{x})).$$

- But binomial loss places less emphasis on the bad cases, and so it is more robust when data is noisy. Optimizing this is called logit-Boost.

# Example

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