ENGR-E 511; ENGR-E 399

Machine Learning for Signal Processing

Module 12:

Adaptive Basis Function Models

Minje Kim

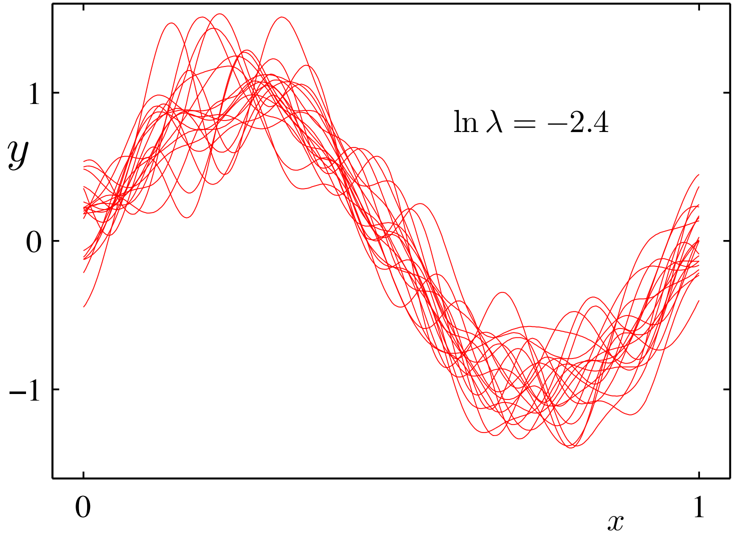
Department of Intelligent Systems Engineering

Email: minje@indiana.edu

Website: http://minjekim.com

Research Group: http://saige.sice.indiana.edu

Meeting Request: http://doodle.com/minje



INDIANA UNIVERSITY

SCHOOL OF INFORMATICS,
COMPUTING, AND ENGINEERING

Adaptive Basis Function Models

- Really, nothing but weighted sum of features

- ○ Let's go against the argument about the Kernel methods
- ○ Basis functions
  - □ Feature transformation function $\phi_m(\boldsymbol{x})$
  - □ In adaptive basis function models we explicitly learn this function from data
    - Instead of using kernels
- ○ Adaptive basis functions?
  - □ First you assume that there are M such basis functions

$$f(\boldsymbol{x}) = w_0 + \sum_{m=1}^{M} w_m \phi_m(\boldsymbol{x})$$

  - □ The basis function is parameterized and learned from data

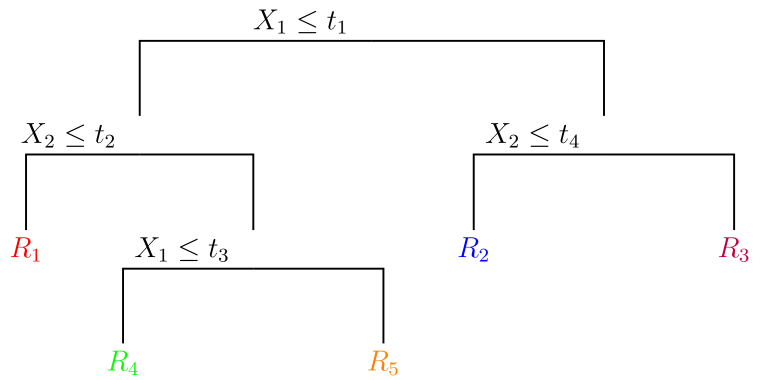
$$\phi_m(\boldsymbol{x}) = \phi(\boldsymbol{x}; \boldsymbol{v}_m)$$

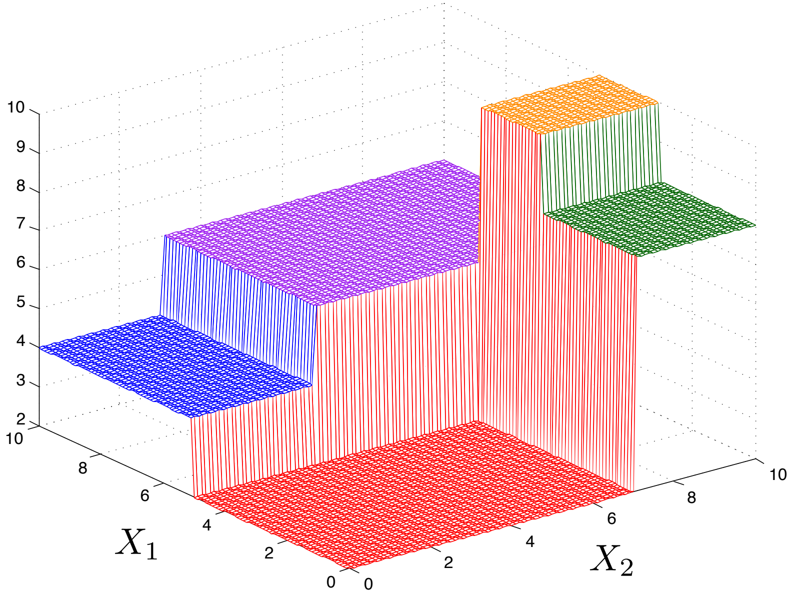
    - The entire parameter set

$$\boldsymbol{\theta} = (w_0, w_1, \cdots, w_M, \boldsymbol{v}_1, \cdots, \boldsymbol{v}_M)$$

Decision Trees

- Classification and Regression Trees (CART)





- ○ Axis parallel splits
- ○ Decision trees are an adaptive basis function model

$$f(\boldsymbol{x}) = \mathbb{E}(y|\boldsymbol{x}) = \sum_{m=1}^{M} w_m \mathcal{I}(\boldsymbol{x} \in \mathcal{R}_m) = \sum_{m=1}^{M} w_m \phi(\boldsymbol{x}; \boldsymbol{v}_m)$$

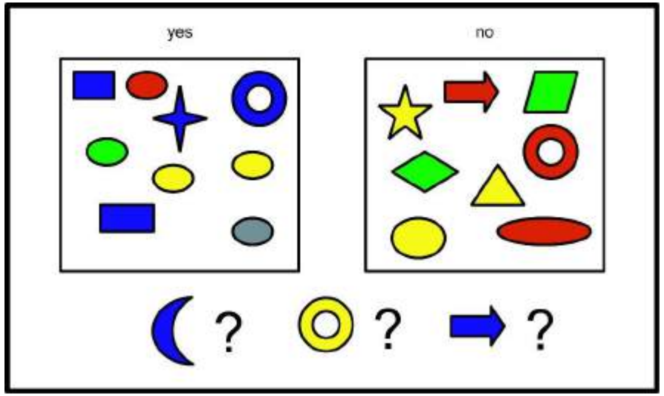
$$\phi(\boldsymbol{x}; \boldsymbol{v}_m) = \begin{cases} 1 & \text{if } \boldsymbol{x} \in \mathcal{R}_m \\ 0 & \text{otherwise} \end{cases}$$

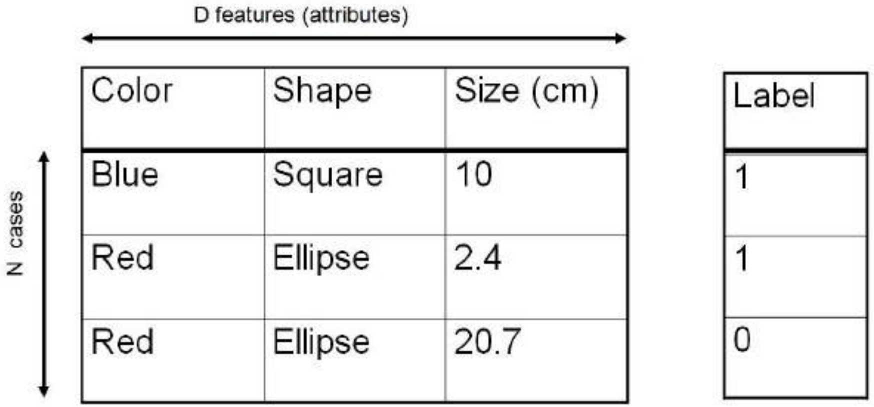
- ○ w_m?
  - ☐ Mean response
- ○ $\boldsymbol{v}_m$?
  - ☐ Which variable to cut, where to cut, and the path from the root to the leaf

MLaPP Figure 16.1

Decision Trees

- CART for classification

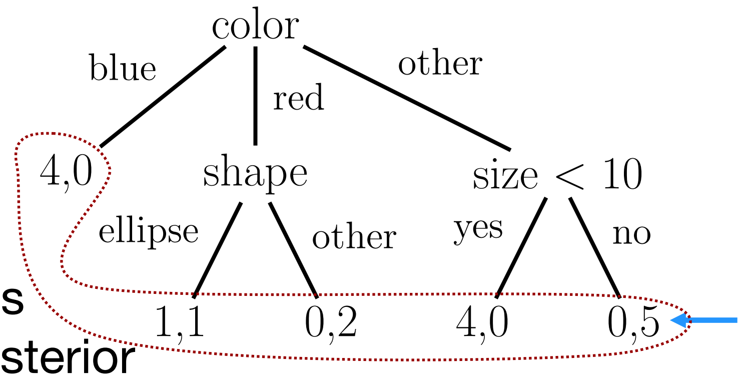




Color	Shape	Size (cm)
Blue	Square	10
Red	Ellipse	2.4
Red	Ellipse	20.7

Label
1
1
0

$$p(y = 1|\boldsymbol{x}) = 4/4$$



Number of training examples per class

- Fraction of the positive examples can be used to construct the posterior

MLaPP Figure 1.1, 16.2

Decision Trees

- Training

- We need to decide
  - Which feature to threshold
  - Where to threshold
  - Based on the cost minimization
- Optimization

$$(f^*, \tau^*) = \arg \min_{f\in\{1,\cdots,F\}} \min_{\tau\in\mathcal{T}_f} \mathrm{cost}(\{\boldsymbol{x}_t, y_t;\ x_{ft} \le \tau\}) + \mathrm{cost}(\{\boldsymbol{x}_t, y_t;\ x_{ft} > \tau\})$$

t	Sample index
f	Features
τ	Threshold
$\mathcal{T}_f$	Set of possible threshold

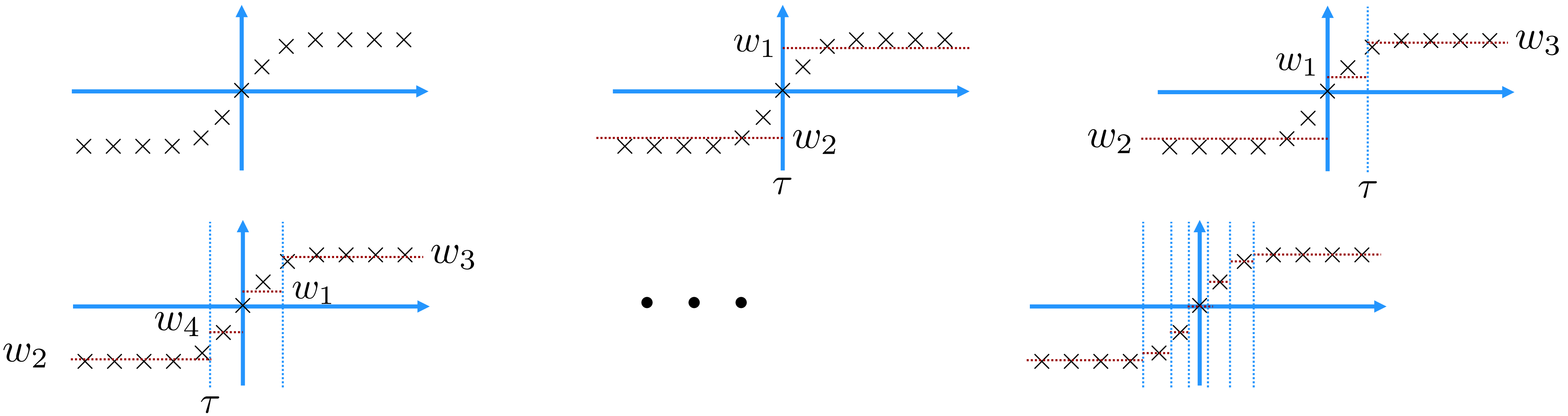
Decision Trees

- Regression cost

- ○ Regression cost $\text{cost}(\mathcal{D}) = \sum_{t\in\mathcal{D}} (y_t - \bar{y})^2$ ⟵ Set of examples in the same region (leaf) $\bar{y} = \frac{1}{|\mathcal{D}|}\sum_{t\in\mathcal{D}} y_t$

- ○ For 1-d data samples

$$(\cancel{f^*}, \tau^*) = \arg \cancel{\min_{f\in\{1,\cdots,F\}}} \min_{\tau\in\mathcal{T}_f} \text{cost}(\{\boldsymbol{x}_t, y_t;\ x_{ft} \le \tau\}) + \text{cost}(\{\boldsymbol{x}_t, y_t;\ x_{ft} > \tau\})$$



Decision Trees

- Classification cost

- ○ Classification cost

$$\hat{\pi}_c = \frac{1}{|\mathcal{D}|} \sum_{t \in \mathcal{D}} \mathbb{I}(y_t = c)$$

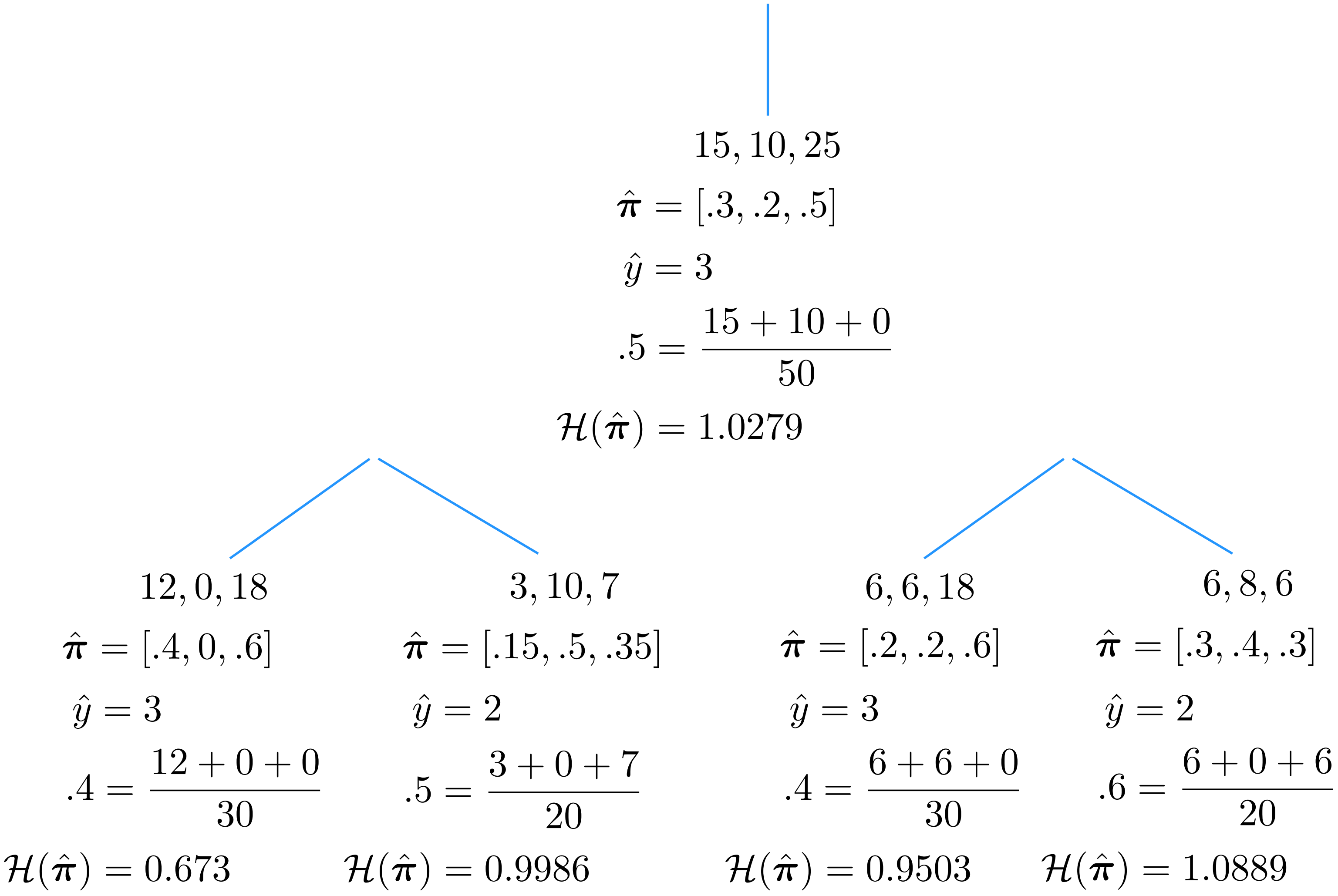
$$\hat{y} = \arg\max_c \hat{\pi}_c$$

  - ☐ Misclassification rate

$$\frac{1}{|\mathcal{D}|} \sum_{t \in \mathcal{D}} \mathbb{I}(y_t \neq \hat{y}) = 1 - \hat{\pi}_{\hat{y}}$$

  - ☐ Entropy

$$\mathcal{H}(\hat{\boldsymbol{\pi}}) = -\sum_{c=1}^{C} \hat{\pi}_c \log \hat{\pi}_c$$



$$15, 10, 25$$
$$\hat{\boldsymbol{\pi}} = [.3, .2, .5]$$
$$\hat{y} = 3$$
$$.5 = \frac{15 + 10 + 0}{50}$$
$$\mathcal{H}(\hat{\boldsymbol{\pi}}) = 1.0279$$

$$12, 0, 18$$
$$\hat{\boldsymbol{\pi}} = [.4, 0, .6]$$
$$\hat{y} = 3$$
$$.4 = \frac{12 + 0 + 0}{30}$$
$$\mathcal{H}(\hat{\boldsymbol{\pi}}) = 0.673$$

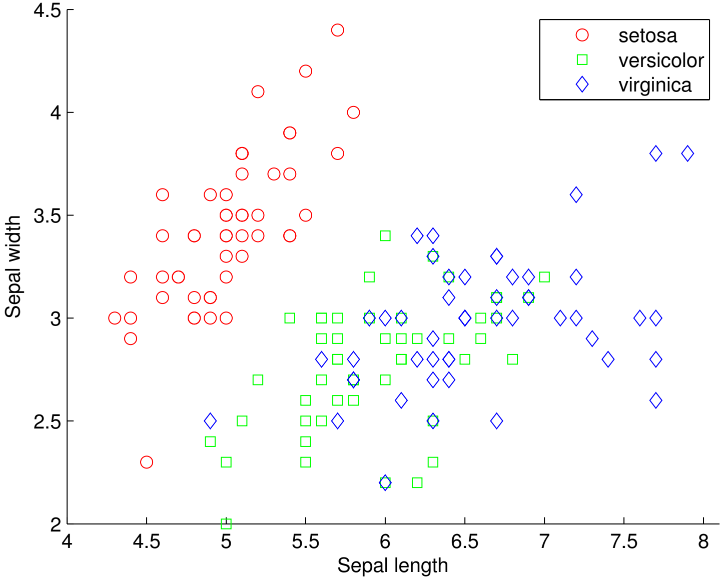
$$3, 10, 7$$
$$\hat{\boldsymbol{\pi}} = [.15, .5, .35]$$
$$\hat{y} = 2$$
$$.5 = \frac{3 + 0 + 7}{20}$$
$$\mathcal{H}(\hat{\boldsymbol{\pi}}) = 0.9986$$

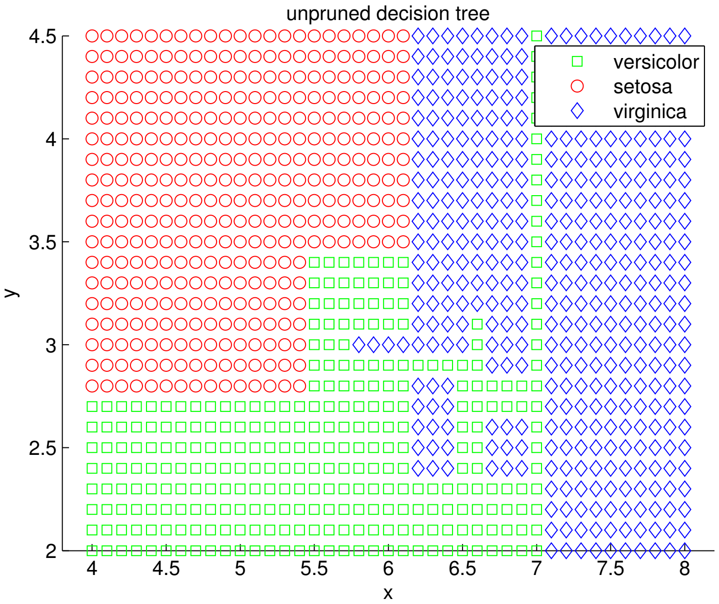
$$6, 6, 18$$
$$\hat{\boldsymbol{\pi}} = [.2, .2, .6]$$
$$\hat{y} = 3$$
$$.4 = \frac{6 + 6 + 0}{30}$$
$$\mathcal{H}(\hat{\boldsymbol{\pi}}) = 0.9503$$

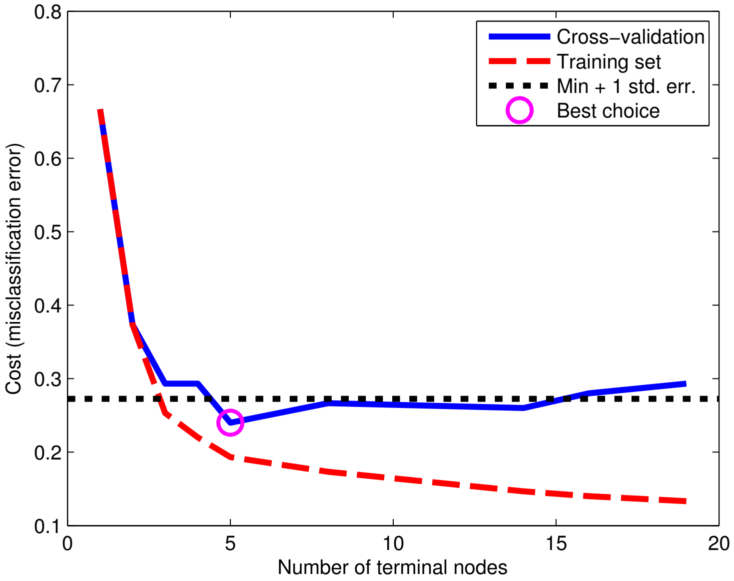
$$6, 8, 6$$
$$\hat{\boldsymbol{\pi}} = [.3, .4, .3]$$
$$\hat{y} = 2$$
$$.6 = \frac{6 + 0 + 6}{20}$$
$$\mathcal{H}(\hat{\boldsymbol{\pi}}) = 1.0889$$

Decision Trees

- Iris example







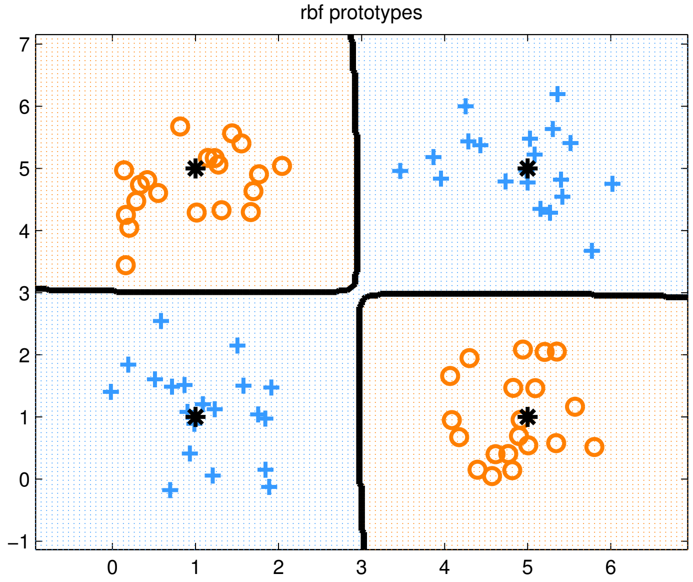
- ○ Overfitting?

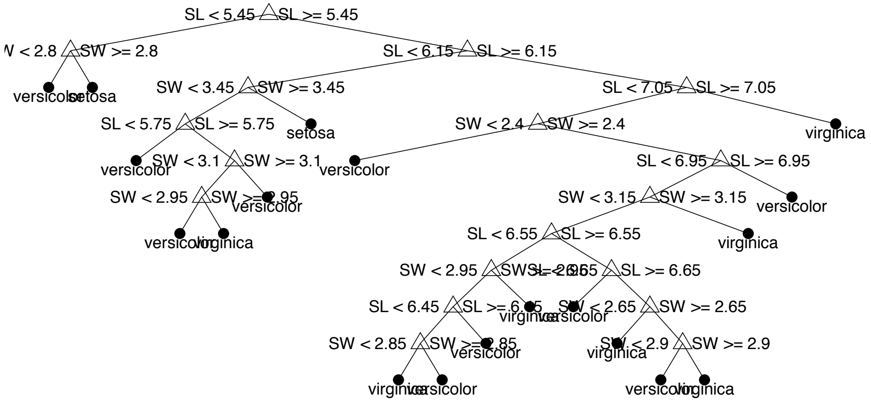
MLaPP Figure 16.4, 16.5(b)

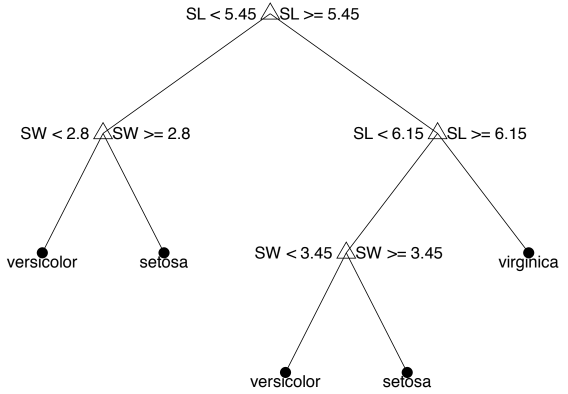
Decision Trees

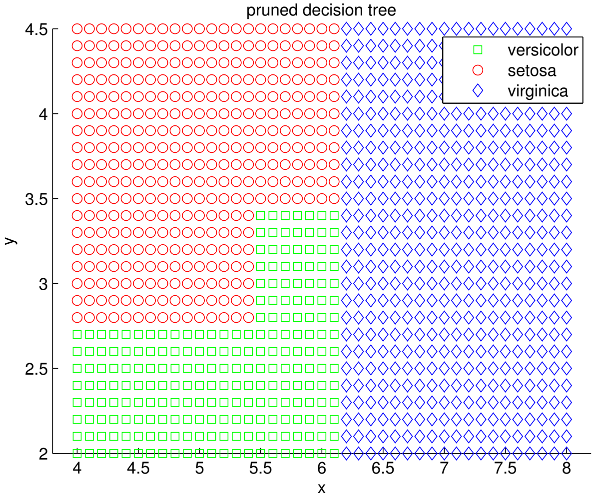
- Pruning

- ○ Too complex trees can overfit
- ○ To stop growing at some point?
  - ☐ e.g. when the cost doesn't go down after a split
  - ☐ Doesn't work if there's a confusing case
- ○ Pruning
  - ☐ Grow a "full" tree, and then prune









MLaPP Figure 14.2(c), 16.5(a),16.6

Decision Trees

- Pros and cons of decision trees

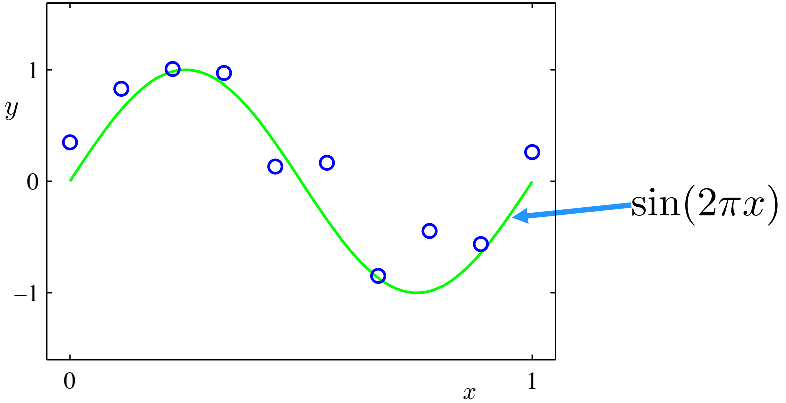
- Pros
  - Easy to interpret (e.g. for medical diagnosis)
  - Can handle discrete input
  - Robust to monotone transformation and scaling (e.g. log)
  - Comes with feature selection
  - Works well with large datasets
  - Easy to handle missing variables
- Cons
  - There are other outperforming models
    - The greedy construction algorithm is not very optimal
  - Unstable—small changes in the top node propagate down to the leaf nodes
  - High variance

Overfitting and Regularization

- Polynomial curve fitting

- ○ Minimizing an error function

$$\mathcal{E}(\boldsymbol{w}) = \frac{1}{2}\sum_{t}\big(f(x_t,\boldsymbol{w}) - y_t\big)^2 \qquad f(x,\boldsymbol{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \boldsymbol{w}^\top \boldsymbol{x}$$



  - □ Therefore,

$$\arg\min_{\boldsymbol{w}} \mathcal{E}(\boldsymbol{y}||\boldsymbol{w}^\top \boldsymbol{X}) = (\boldsymbol{X}\boldsymbol{X}^\top)^{-1}\boldsymbol{X}\boldsymbol{y} \qquad \boldsymbol{y} = [y_1, y_2, \cdots, y_T]^\top$$

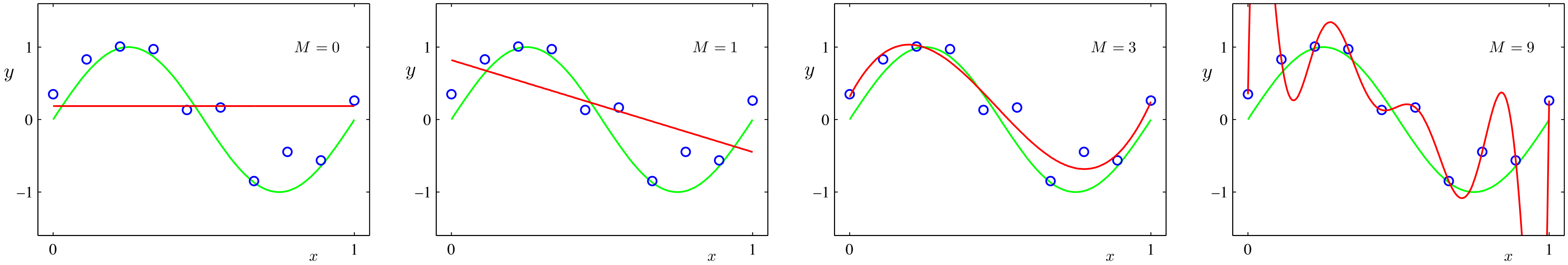
- ○ Question: what's the right order of polynomial, M?
  - □ The more the better?

PRML Figure 1.2

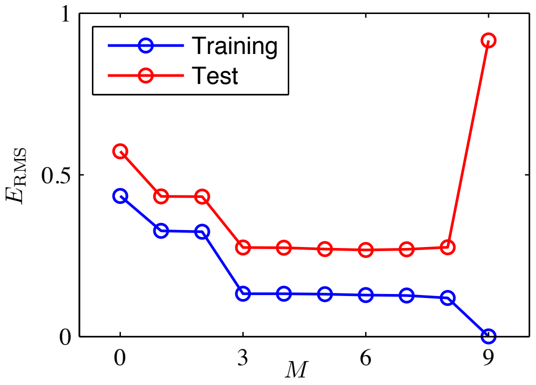
Overfitting and Regularization

- Overfitting

- The more the better?



  - Too complex models tend to **overfit**
    - i.e. The model cares too much about the training error
  - Too complex models don't generalize well
- Generalization?
  - A model trained on a dataset (training set) should work well in the other data set (testing set)



PRML Figure 1.4, 1.5

Overfitting and Regularization

- Preventing overfitting—early stopping

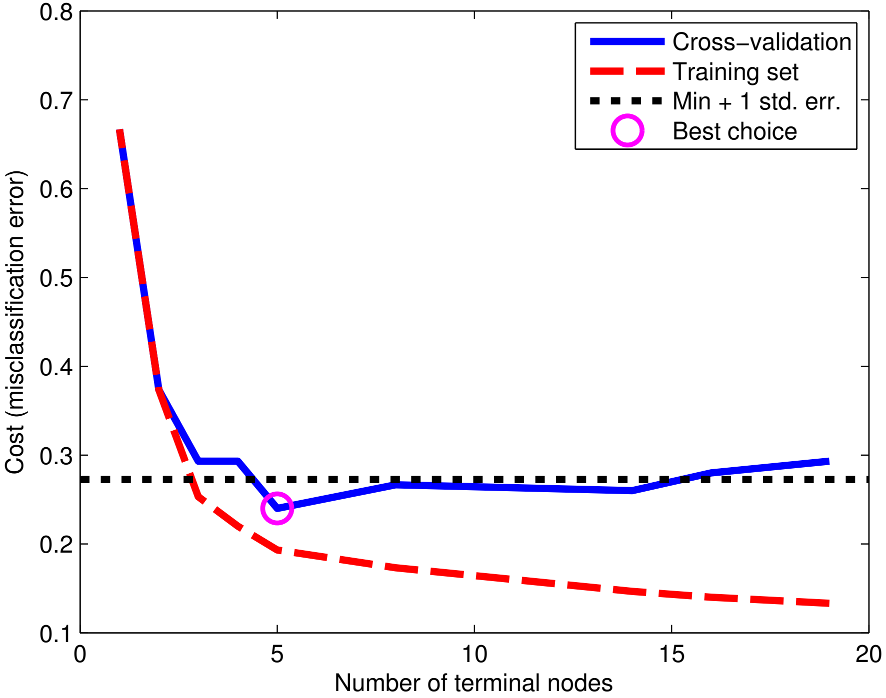
- Early stopping
  - Check on the simulated test error and stops earlier than the convergence (of the training error)
- N-fold Cross validation: simulate the testing environment using training data
  - Divide the training set into N exclusive subsets

Frames

Features

1st fold	Train	Train	Test
2nd fold	Train	Test	Train
3rd fold	Test	Train	Train

  - N different train-validation pairs
  - Each pair is used to train a classifier and to evaluate it
  - Average the N results
  - The average shows the performance of your choice



MLaPP Figure 16.5(b)

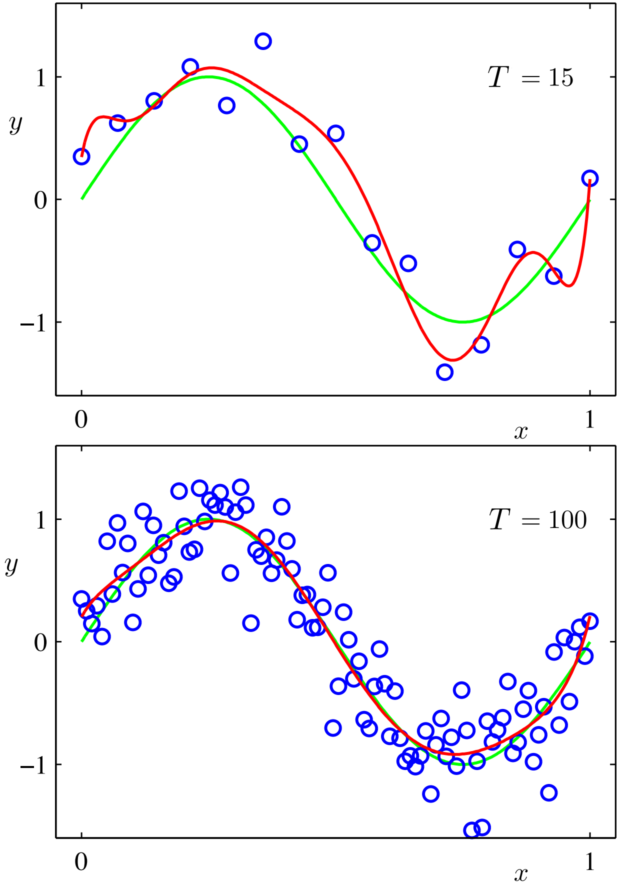
Overfitting and Regularization

- Preventing overfitting—more training samples

- Weights become larger if the model overfits

	M=0	M=1	M=3	M=9
w_0^*	0.19	0.82	0.31	0.35
w_1^*		-1.27	7.99	232.37
w_2^*			-25.43	-5321.83
w_3^*			17.37	48568.31
w_4^*				-231639.30
w_5^*				640042.26
w_6^*				-1061800.52
w_7^*				1042400.18
w_8^*				-557682.99
w_9^*				125201.43

- A big training dataset solves the problem



PRML Figure 1.6

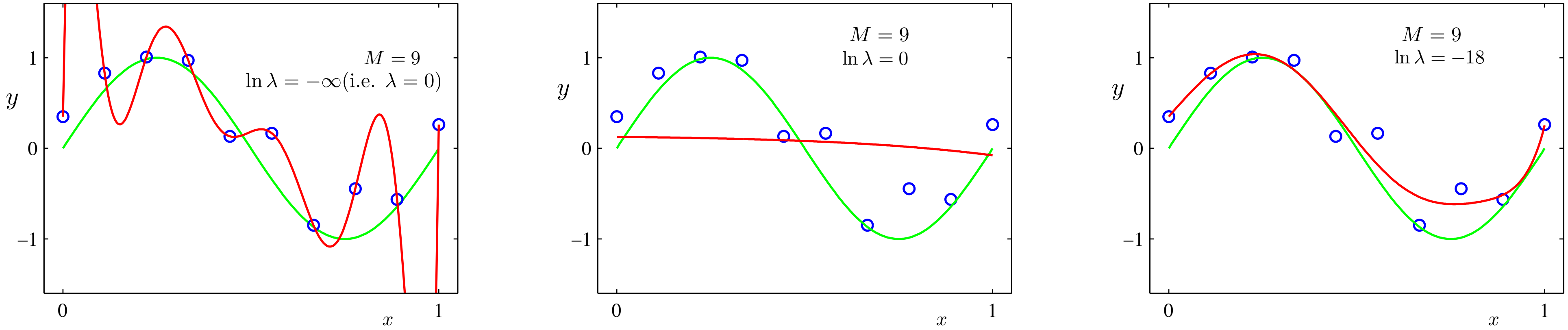
Overfitting and Regularization

- Preventing overfitting—weight decay

- The regularized objective function

$$\mathcal{E}(\boldsymbol{w}) = \frac{1}{2}\sum_{t}\left(f(x_t, \boldsymbol{w}) - y_t\right)^2 + \frac{\lambda}{2}\boldsymbol{w}^\top \boldsymbol{w}$$

- Keep M large
  - You want a complex-enough model to solve your complex problem
- Keep $\boldsymbol{w}$ small
  - You don't want the parameters to be too large
- λ is a hyperparameter that controls the amount of weight decaying



PRML Figure 1.7

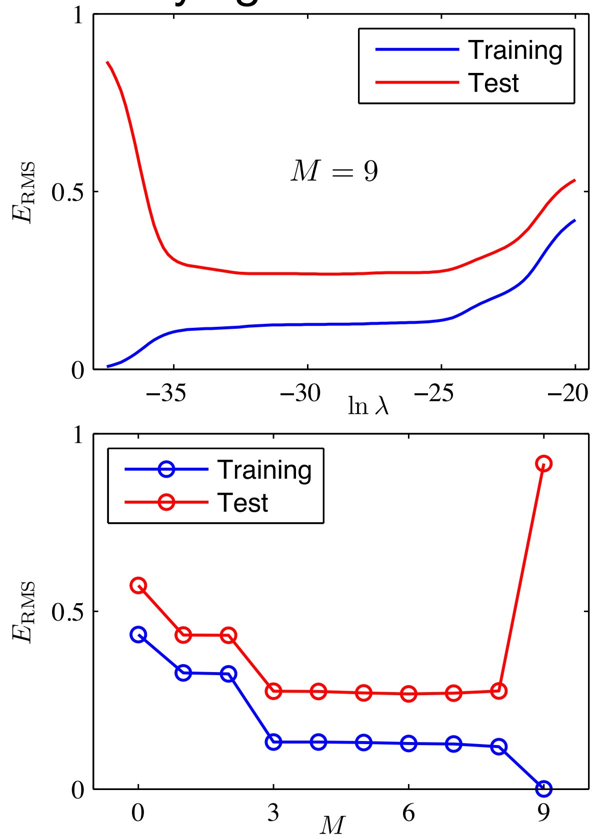
Overfitting and Regularization

- Preventing overfitting—weight decay

- Regularization can decay the weights

	$\ln\lambda = -\infty$	$\ln\lambda = -18$	$\ln\lambda = 0$
w_0^*	0.35	0.35	0.13
w_1^*	232.37	4.74	-0.05
w_2^*	-5321.83	-0.77	-0.06
w_3^*	48568.31	-31.97	-0.05
w_4^*	-231639.30	-3.89	-0.03
w_5^*	640042.26	55.28	-0.02
w_6^*	-1061800.52	41.32	-0.01
w_7^*	1042400.18	-45.95	-0.00
w_8^*	-557682.99	-91.53	0.00
w_9^*	125201.43	72.68	0.01

- Regularization lets us use a complex model without worrying about overfitting



PRML Figure 1.8

Overfitting and Regularization

- Preventing overfitting—another takes

- ○ Recall SVM objective function can be seen as a combination of the hinge loss and regularization

$$\frac{1}{2}\boldsymbol{w}^\top\boldsymbol{w} + C\sum_t \xi_t \leftrightarrow \sum_t \mathcal{E}\big(y_t, f(x_t)\big) + \lambda||\boldsymbol{w}||^2 \qquad \mathcal{E}\big(y_t, f(x_t)\big) = \xi_t = \begin{cases} 0 & \text{if } y_t f(x_t) \geq 1 \\ 1 - y_t f(x_t) & \text{otherwise} \end{cases}$$

- ○ Bayesian priors can work as a regularizer
  - □ Maximum likelihood

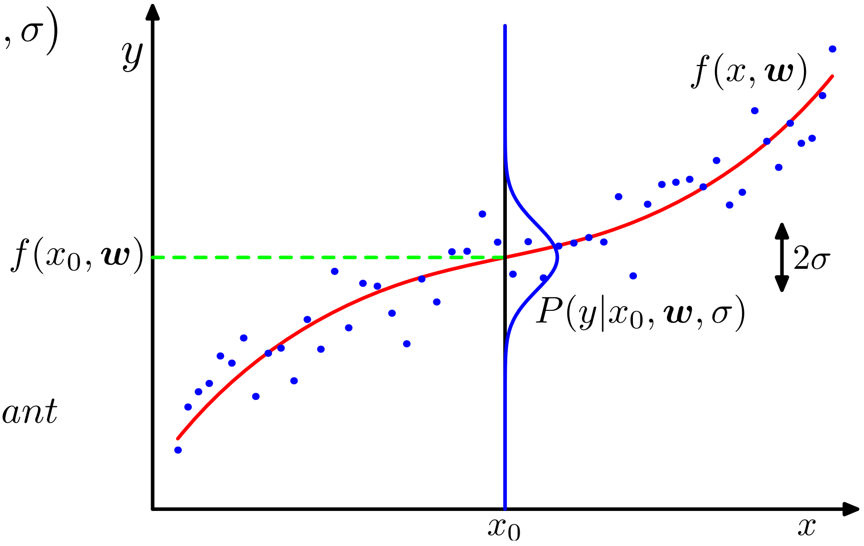
$$P(y|x,\boldsymbol{w},\sigma) = \mathcal{N}\big(y|f(x,\boldsymbol{w}),\sigma\big) \;\leftrightarrow P(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{w},\sigma) = \prod_t \mathcal{N}\big(y_t|f(x_t,\boldsymbol{w}),\sigma\big)$$

$$\ln P(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{w},\sigma) = -\frac{1}{2\sigma^2}\sum_t \big(f(x_t,\boldsymbol{w}) - y_t\big)^2$$

  - □ Prior $P(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{0},\sigma_\pi^2\boldsymbol{I}) \propto \exp\left(-\frac{\boldsymbol{w}^\top\boldsymbol{w}}{2\sigma_\pi^2}\right)$
  - □ MAP $P(\boldsymbol{w}|\boldsymbol{x},\boldsymbol{y},\sigma,\sigma_\pi) \propto P(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{w},\sigma)P(\boldsymbol{w}|\sigma_\pi)$

$$\ln P(\boldsymbol{w}|\boldsymbol{x},\boldsymbol{y},\sigma,\sigma_\pi) = -\frac{1}{2\sigma^2}\sum_t \big(f(x_t,\boldsymbol{w}) - y_t\big)^2 - \frac{1}{\sigma_\pi^2}\boldsymbol{w}^\top\boldsymbol{w} + constant$$

  - □ Or to minimize $\frac{1}{2\sigma^2}\sum_t \big(f(x_t,\boldsymbol{w}) - y_t\big)^2 + \frac{1}{\sigma_\pi^2}\boldsymbol{w}^\top\boldsymbol{w}$



PRML Figure 1.16

The Bias-Variance Trade-Off

- Bias-variance decomposition

- Bias-variance decomposition

Trained model varies depending on the training set

Dummy terms

$$\mathcal{E}\big(f(\boldsymbol{x};\mathcal{D})||y\big) = \big(f(\boldsymbol{x};\mathcal{D}) - y\big)^2$$

Training set
(subsampled from the GT
sample distribution)

$$= \big(f(\boldsymbol{x};\mathcal{D}) \boxed{- \mathbb{E}_D[f(\boldsymbol{x};\mathcal{D})] + \mathbb{E}_D[f(\boldsymbol{x};\mathcal{D})]} - y\big)^2$$

$$= \big(f(\boldsymbol{x};\mathcal{D}) - \mathbb{E}_D[f(\boldsymbol{x};\mathcal{D})]\big)^2 + \big(\mathbb{E}_D[f(\boldsymbol{x};\mathcal{D})] - y\big)^2$$

$$+2\big(f(\boldsymbol{x};\mathcal{D}) - \mathbb{E}_D[f(\boldsymbol{x};\mathcal{D})]\big)\big(\mathbb{E}_D[f(\boldsymbol{x};\mathcal{D})] - y\big)$$

- Expected squared error

$$\mathbb{E}_{\mathcal{D}}\left[\big(f(\boldsymbol{x};\mathcal{D}) - y\big)^2\right] = \underline{\mathbb{E}_{\mathcal{D}}\left[\big(f(\boldsymbol{x};\mathcal{D}) - \mathbb{E}_D[f(\boldsymbol{x};\mathcal{D})]\big)^2\right]} + \underline{\big(\mathbb{E}_D[f(\boldsymbol{x};\mathcal{D})] - y\big)^2}$$

Expected error;
Assumes multiple models
trained from different subsets
of the original dataset
(by varying $\mathcal{D}$)

Variance:
Trained models vary depending on the choice of $\mathcal{D}$?

Bias:
How accurate the model is

$$+2\mathbb{E}\left[\big(f(\boldsymbol{x};\mathcal{D}) - \mathbb{E}_D[f(\boldsymbol{x};\mathcal{D})]\big)\big(\mathbb{E}_D[f(\boldsymbol{x};\mathcal{D})] - y\big)\right]$$

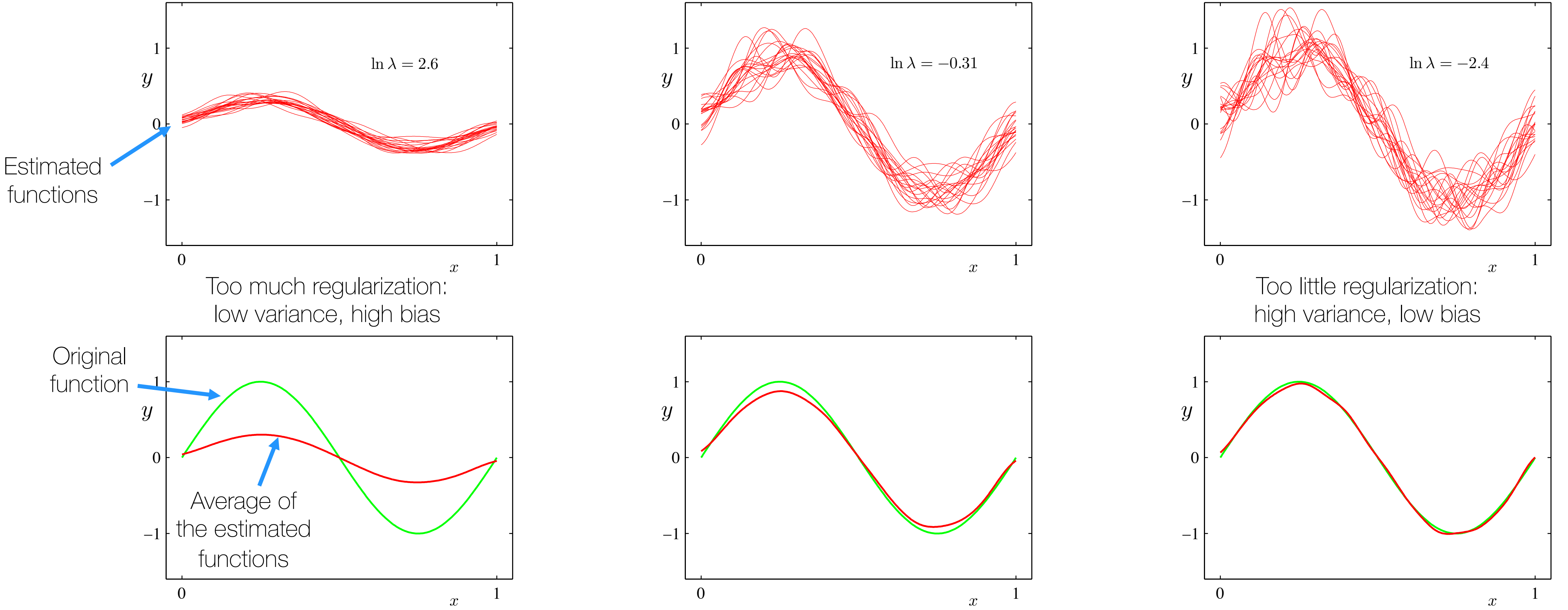
$$\leftrightarrow +2\big(\mathbb{E}_D[f(\boldsymbol{x};\mathcal{D})] - y\big)\mathbb{E}\left[\big(f(\boldsymbol{x};\mathcal{D}) - \mathbb{E}_D[f(\boldsymbol{x};\mathcal{D})]\big)\right]$$

$$\leftrightarrow +2\big(\mathbb{E}_D[f(\boldsymbol{x};\mathcal{D})] - y\big)\ \Big(\mathbb{E}\left[\big(f(\boldsymbol{x};\mathcal{D})\right] - \mathbb{E}_D[f(\boldsymbol{x};\mathcal{D})]\big)\Big) \rightarrow 0$$

The Bias-Variance Trade-Off

- Averaged model predictions can reduce variance

- ○ 100 models from 100 different $\mathcal{D}$



Estimated functions

Too much regularization: low variance, high bias

Too little regularization: high variance, low bias

Original function

Average of the estimated functions

Model averaging doesn’t help remove bias

Model averaging helps remove variance

PRML Figure 3.5

The Bias-Variance Trade-Off

- Bootstrap aggregation (or bagging); random forests

○ In theory, if you have multiple training datasets,

  ☐ Train multiple complex models and average the results→ low variance and low bias

○ In practice, you don't have multiple training datasets

○ **Bootstrapping**

  ☐ **Subsample from one training dataset with replacement**

○ Train m-th model from m-th bootstrap dataset $\sum_{t \in \mathcal{D}_m} \mathcal{E}\left(\phi_m(\boldsymbol{x}_t; \mathcal{D}_m) || y_t\right)$

○ The M models construct a committee (**bagging**) $f(\boldsymbol{x}) = \frac{1}{M} \sum_{m=1}^{M} \phi_m(\boldsymbol{x})$

○ Variance $\mathbb{E}_{\mathcal{D}}\left[\left(f(\boldsymbol{x}; \mathcal{D}) - \mathbb{E}_D[f(\boldsymbol{x}; \mathcal{D})]\right)^2\right]$

  ☐ We hope

$$f(\boldsymbol{x}) = \frac{1}{M} \sum_{m=1}^{M} \phi_m(\boldsymbol{x}) \approx \mathbb{E}_D[f(\boldsymbol{x}; \mathcal{D})] \quad \text{if } M \to \infty$$

○ **Random forests**: subsample from dataset; subset of variables

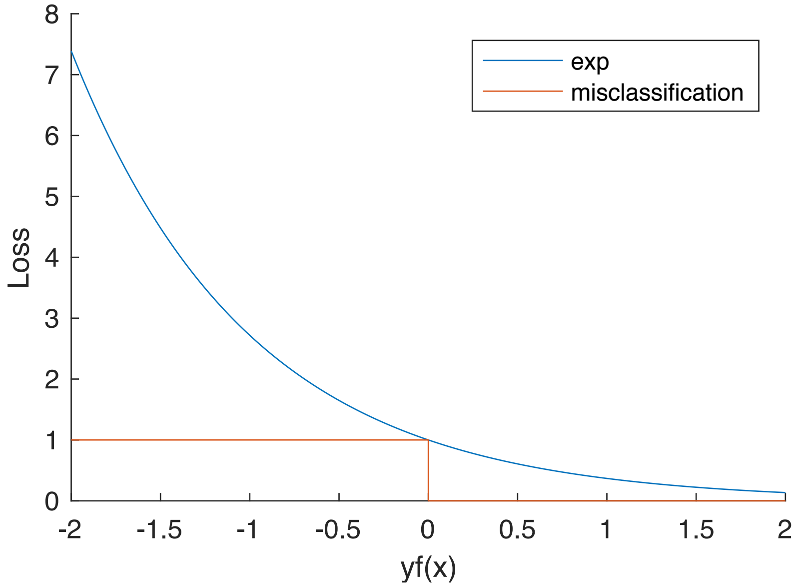
AdaBoost (Adaptive Boosting)

- Boosting basics

- Adaptive basis functions are called weak learners in boosting $f(\boldsymbol{x}) = w_0 + \sum_{m=1}^{M} w_m \phi_m(\boldsymbol{x})$
- $\phi_m(\boldsymbol{x})$ can be a shallow decision tree
  - Or just a perceptron
- The objective function for boosting $\min_f \sum_t \mathcal{E}\big(y_t, f(\boldsymbol{x}_t)\big)$
- Exponential loss

  $$\mathcal{E}\big(y, f(\boldsymbol{x})\big) = \exp(-y f(\boldsymbol{x}))$$

  - For bipolar binary output $y = \{-1, +1\}$



AdaBoost (Adaptive Boosting)

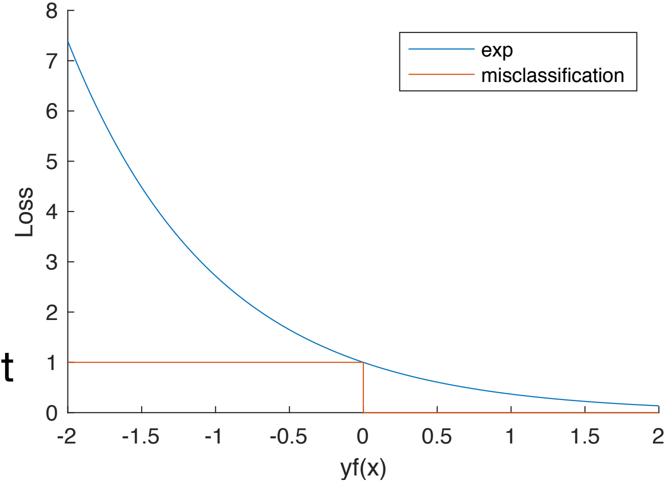
- Forward stagewise additive modeling

- ○ The exponential loss is magical
    - ☐ Let's see what happens
- ○ At m-th step of AdaBoost training, we add m-th weak learner to the model

$$f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \beta_m \phi_m(\boldsymbol{x})$$

- ☐ That minimizes the m-th step objective function

$$\begin{aligned}\mathcal{E}_m(\phi) &= \sum_t \exp\Big(-y_t\big(f_{m-1}(\boldsymbol{x}_t) + \beta_m\phi_m(\boldsymbol{x}_t)\big)\Big)\\ &= \sum_t \underbrace{\exp\Big(-y_t\big(f_{m-1}(\boldsymbol{x}_t)\big)\Big)}_{\text{Fixed constants}} \exp\Big(-y_t\big(\underbrace{\beta_m\phi_m}_{\text{Things to estimate}}(\boldsymbol{x}_t)\big)\Big)\\ &= \sum_t \underbrace{w_{tm}}_{\text{Works like weights over samples}} \exp\big(-\beta_m y_t \phi(\boldsymbol{x}_t)\big)\end{aligned}$$



- ☐ At every step, the weak learner is estimated by using a re-weighted dataset
    - Misclassified examples get more weights

AdaBoost (Adaptive Boosting)

- Forward stagewise additive modeling

- Rearrange the objective function

$$\exp\big(-\beta_m y_t \phi(\boldsymbol{x}_t)\big) = \begin{cases} \exp(-\beta) & \text{if } y_t = \phi(\boldsymbol{x}_t) \\ \exp(\beta) & \text{if } y_t \neq \phi(\boldsymbol{x}_t) \end{cases}$$

$$\begin{aligned}
\mathcal{E}_m(\phi) &= \sum_t w_{tm} \exp\big(-\beta_m y_t \phi(\boldsymbol{x}_t)\big) \\
&= \sum_{t:y_t=\phi(\boldsymbol{x}_t)} w_{tm}\exp(-\beta_m) + \sum_{t:y_t\neq\phi(\boldsymbol{x}_t)} w_{tm}\exp(\beta_m) \\
&= \sum_{t:y_t=\phi(\boldsymbol{x}_t)} w_{tm}\exp(-\beta_m) + \sum_{t:y_t\neq\phi(\boldsymbol{x}_t)} w_{tm}\exp(-\beta_m) - \sum_{t:y_t\neq\phi(\boldsymbol{x}_t)} w_{tm}\exp(-\beta_m) + \sum_{t:y_t\neq\phi(\boldsymbol{x}_t)} w_{tm}\exp(\beta_m) \\
&= \sum_{t=1}^{T} w_{tm}\exp(-\beta_m) + \big(\exp(\beta_m) - \exp(-\beta_m)\big) \sum_{t:y_t\neq\phi(\boldsymbol{x}_t)} w_{tm} \\
&= \sum_{t=1}^{T} w_{tm}\exp(-\beta_m) + \big(\exp(\beta_m) - \exp(-\beta_m)\big) \sum_{t=1}^{T} w_{tm}\mathcal{I}\big(y_t \neq \phi(\boldsymbol{x}_t)\big)
\end{aligned}$$

- Estimating the m-th weak learner is eventually $\phi_m = \arg\min_{\phi} \sum_{t=1}^{T} \underline{w_{tm}}\,\underline{\mathcal{I}\big(y_t \neq \phi(\boldsymbol{x}_t)\big)}$
  - ☐ For example $\mathrm{cost}(\mathcal{D}) = \sum_{t\in\mathcal{D}} w_{tm}(y_t - \bar{y})^2$

But, you give different weights to the different samples, based on the previous round, *(m-1)*-th step

This is a misclassification loss. So, basically you're looking for a simple, but good classifier

AdaBoost (Adaptive Boosting)

- Forward stagewise additive modeling

- Solve for β

$$\mathcal{E}_m(\phi) = \sum_{t=1}^{T} w_{tm} \exp(-\beta_m) + \big(\exp(\beta_m) - \exp(-\beta_m)\big) \sum_{t=1}^{T} w_{tm} \mathcal{I}\big(y_t \neq \phi_m(\boldsymbol{x}_t)\big)$$

Differentiation

$$0 = -\sum_{t=1}^{T} w_{tm} \exp(-\beta_m) + \big(\exp(\beta_m) + \exp(-\beta_m)\big) \sum_{t=1}^{T} w_{tm} \mathcal{I}\big(y_t \neq \phi_m(\boldsymbol{x}_t)\big)$$

$$0 = -\sum_{t=1}^{T} w_{tm} \exp(-\beta_m) \exp(\beta_m) + \big(\exp(\beta_m) \exp(\beta_m) + \exp(-\beta_m) \exp(\beta_m)\big) \sum_{t=1}^{T} w_{tm} \mathcal{I}\big(y_t \neq \phi_m(\boldsymbol{x}_t)\big)$$

$$0 = -\sum_{t=1}^{T} w_{tm} + \exp(2\beta_m) \sum_{t=1}^{T} w_{tm} \mathcal{I}\big(y_t \neq \phi_m(\boldsymbol{x}_t)\big) + \sum_{t=1}^{T} w_{tm} \mathcal{I}\big(y_t \neq \phi_m(\boldsymbol{x}_t)\big)$$

$$0 = \exp(2\beta_m) \sum_{t=1}^{T} w_{tm} \mathcal{I}\big(y_t \neq \phi_m(\boldsymbol{x}_t)\big) - \sum_{t=1}^{T} w_{tm} \mathcal{I}\big(y_t = \phi_m(\boldsymbol{x}_t)\big)$$

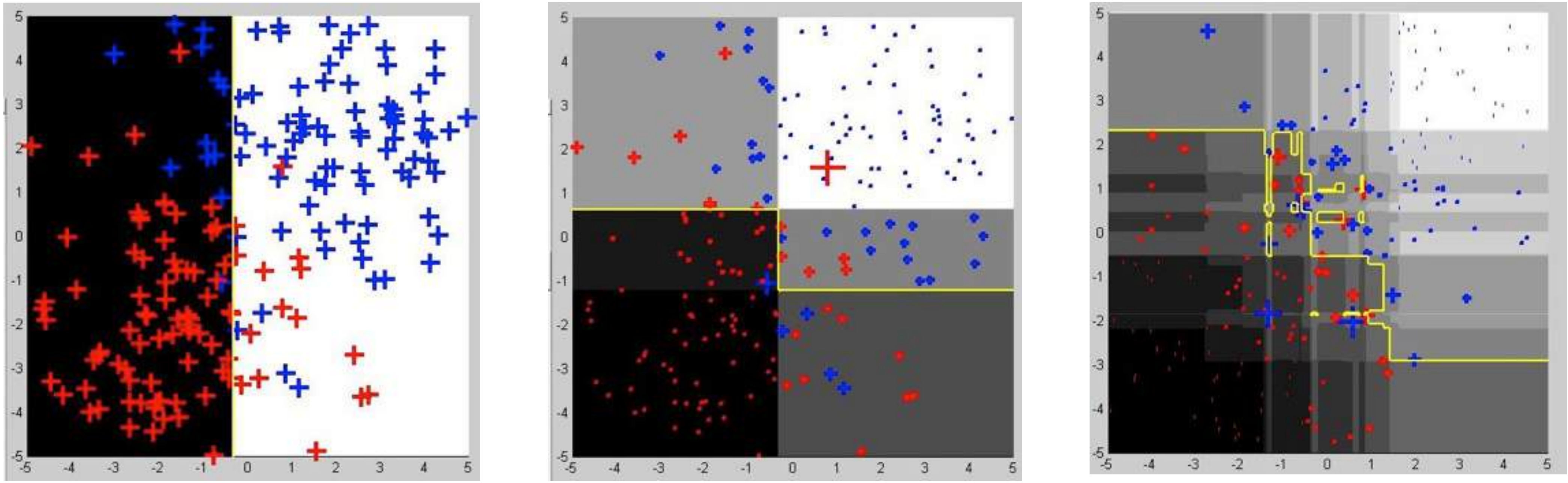
$$\exp(2\beta_m) \sum_{t=1}^{T} w_{tm} \mathcal{I}\big(y_t \neq \phi_m(\boldsymbol{x}_t)\big) = \sum_{t=1}^{T} w_{tm} \mathcal{I}\big(y_t = \phi_m(\boldsymbol{x}_t)\big)$$

$$\therefore \beta_m = \frac{1}{2} \ln \frac{\sum_{t=1}^{T} w_{tm} \mathcal{I}\big(y_t = \phi_m(\boldsymbol{x}_t)\big)}{\sum_{t=1}^{T} w_{tm} \mathcal{I}\big(y_t \neq \phi_m(\boldsymbol{x}_t)\big)}$$

AdaBoost (Adaptive Boosting)

- Forward stagewise additive modeling

- ○ Now that all the estimation jobs are done, $f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \beta_m \phi_m(\boldsymbol{x})$
- ○ Calculation of the new weights $w_{t,m+1} = w_{t,m} \exp\big(- \beta_m y_t \phi_m(\boldsymbol{x}_t)\big)$ See slide 22 if you want to proove
- ○ Repeat this process by adding more basis functions



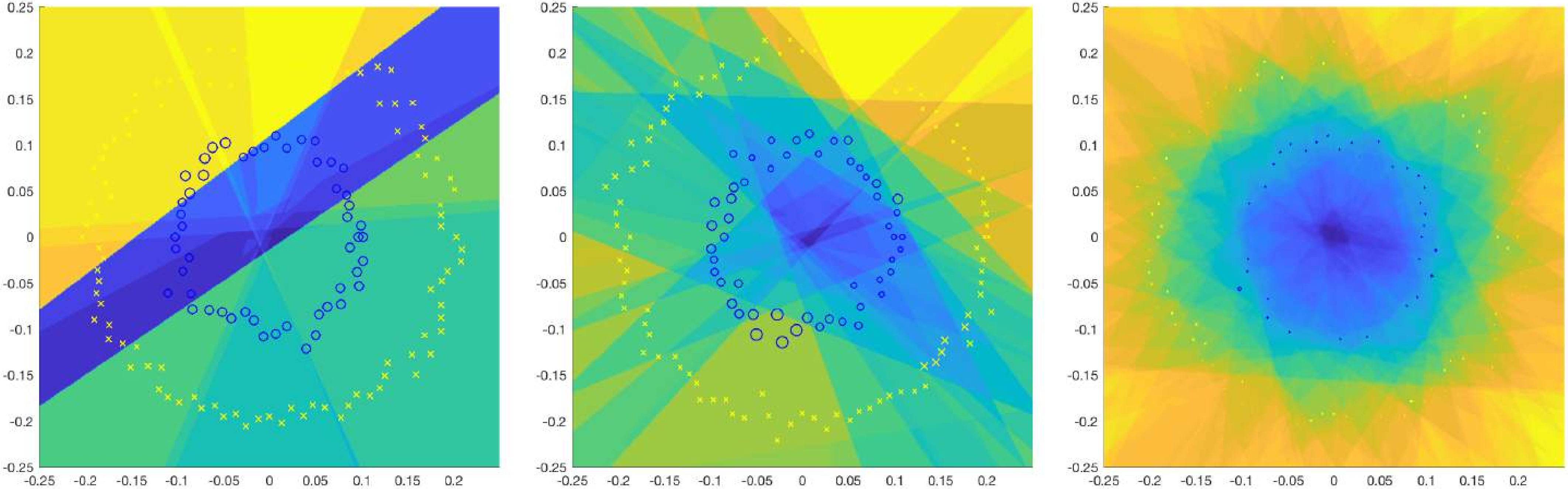
Size of the markers represents their weights

MLaPP Figure 16.10

AdaBoost (Adaptive Boosting)

- Concentric

- AdaBoost on the concentric dataset



$m = 20$ $m = 50$ $m = 500$

Deep Neural Networks

- A neural network is an adaptive basis function model

$$f(\boldsymbol{x}) = w_0 + \sum_{m=1}^{M} \underline{w_m}\,\underline{\phi_m(\boldsymbol{x})}$$

One of the Hidden unit outputs in the final layer

Weight associated with the unit

  - Deep neural networks does the feature transformation using all the layers up to the last one
    - So, the basis functions can be seen as features
    - Note that boosting sees the basis functions in this way, too
- DNNs tend to give you an unbiased solution
  - In other words, they overfit if the training set is small
    - Needs strong regularization with not enough data
  - Deep learning: big models for big data
  - The popularity of CNN: models are smaller by introducing a more complex operation, convolution
- ENGR E533 “Deep Learning Systems” in Spring 2019

Reading

- Kevin Murphy, “Machine Learning: a Probabilistic Perspective”
  - Chapter 6.4, 16.1-16.4
- Christopher Bishop, “Pattern Recognition and Machine Learning”
  - Chapter 1.1, 3.2

Thank You!