Lecture 13: More ensemble methods

TTIC 31020: Introduction to Machine Learning

Instructor: Kevin Gimpel

TTI-Chicago

November 12, 2019

Review: AdaBoost

1. Initialize weights: $W_i^{(0)} = 1/n$
2. Iterate for $m = 1, \ldots, M$:
   - Find "weak" classifier h_m that attains weighted error ϵ_m
   - Let $\alpha_m = \frac{1}{2} \log \frac{1-\epsilon_m}{\epsilon_m}$
   - Update and renormalize the weights:

$$W_i^{(m)} \propto W_i^{(m-1)} e^{-\alpha_m y_i h_m(\mathbf{x}_i)}$$

3. The combined classifier: $\mathrm{sign}\left(\sum_{m=1}^{M} \alpha_m h_m(\mathbf{x})\right)$

- Optimizes exponential loss on training data
- Regularization: (a) via early stopping, (b) via regularization of weak learners

Boosting and bias-variance tradeoff

$$H(\mathbf{x}) = \sum_{m=1}^{M} \alpha_m h_m(\mathbf{x})$$

- What determines the complexity of boosted classifier?

Boosting and bias-variance tradeoff

$$H(\mathbf{x}) = \sum_{m=1}^{M} \alpha_m h_m(\mathbf{x})$$

- What determines the complexity of boosted classifier?
  complexity of weak classifiers h_m and their number M

Boosting and bias-variance tradeoff

$$H(\mathbf{x}) = \sum_{m=1}^{M} \alpha_m h_m(\mathbf{x})$$

- What determines the complexity of boosted classifier?
  complexity of weak classifiers h_m and their number M
- Regularization of H: early stopping
  Watch validation performance; stop before it deteriorates
  This may be *after* training error reaches zero!
- Typically we prefer simple weak classifiers: stumps, shallow decision trees

Boosting stumps

- Suppose $\mathbf{x} = [\phi_0(\mathbf{x}), \phi_1(\mathbf{x}), \ldots, \phi_d(\mathbf{x})]^\top$
- Decision stump: a simple classifier

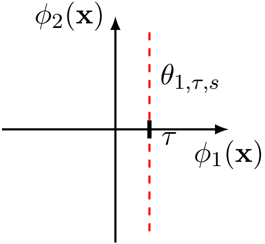
$$\theta_{j,\tau,s}(\mathbf{x}) \;=\; \begin{cases} +1 & \text{if } s\phi_j(\mathbf{x}) \geq \tau \\ -1 & \text{if } s\phi_j(\mathbf{x}) < \tau \end{cases}$$

Boosting stumps

- Suppose $\mathbf{x} = [\phi_0(\mathbf{x}), \phi_1(\mathbf{x}), \ldots, \phi_d(\mathbf{x})]^\top$
- Decision stump: a simple classifier

$$\theta_{j,\tau,s}(\mathbf{x}) \;=\; \begin{cases} +1 & \text{if } s\phi_j(\mathbf{x}) \geq \tau \\ -1 & \text{if } s\phi_j(\mathbf{x}) < \tau \end{cases}$$

- Decision boundary: linear, axis parallel

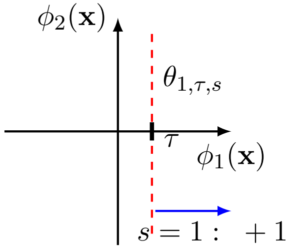


Boosting stumps

- Suppose $\mathbf{x} = [\phi_0(\mathbf{x}), \phi_1(\mathbf{x}), \ldots, \phi_d(\mathbf{x})]^\top$
- Decision stump: a simple classifier

$$\theta_{j,\tau,s}(\mathbf{x}) = \begin{cases} +1 & \text{if } s\phi_j(\mathbf{x}) \geq \tau \\ -1 & \text{if } s\phi_j(\mathbf{x}) < \tau \end{cases}$$

- Decision boundary: linear, axis parallel

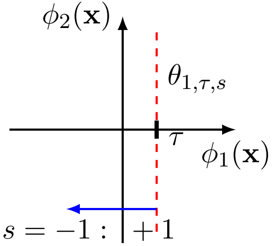


Boosting stumps

- Suppose $\mathbf{x} = [\phi_0(\mathbf{x}), \phi_1(\mathbf{x}), \ldots, \phi_d(\mathbf{x})]^\top$
- Decision stump: a simple classifier

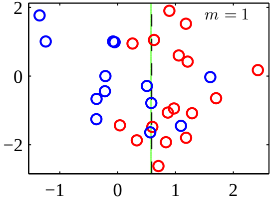
$$\theta_{j,\tau,s}(\mathbf{x}) \ = \ \begin{cases} +1 & \text{if } s\phi_j(\mathbf{x}) \geq \tau \\ -1 & \text{if } s\phi_j(\mathbf{x}) < \tau \end{cases}$$

- Decision boundary: linear, axis parallel



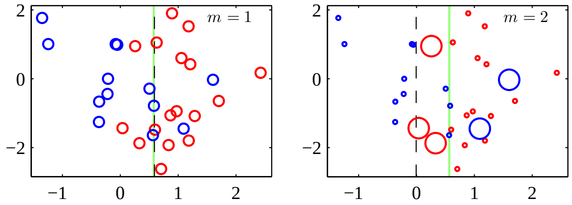
Boosting decision stumps

- Example of a boosting run (from Bishop)
- Green line = combined decision boundary of ensemble; dotted black line = decision boundary of most recent weak learner (decision stump)



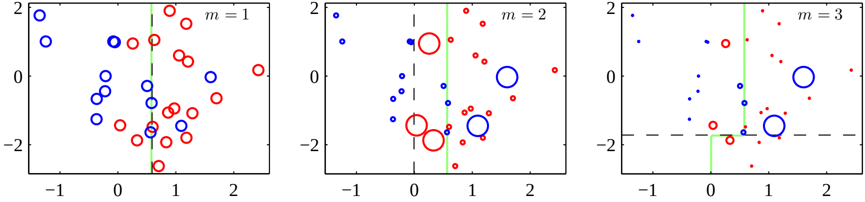
Boosting decision stumps

- Example of a boosting run (from Bishop)
- Green line = combined decision boundary of ensemble; dotted black line = decision boundary of most recent weak learner (decision stump)



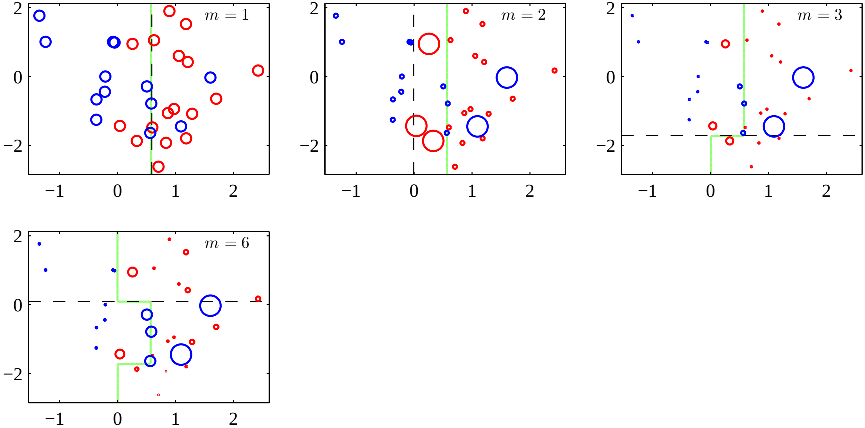
Boosting decision stumps

- Example of a boosting run (from Bishop)
- Green line = combined decision boundary of ensemble; dotted black line = decision boundary of most recent weak learner (decision stump)



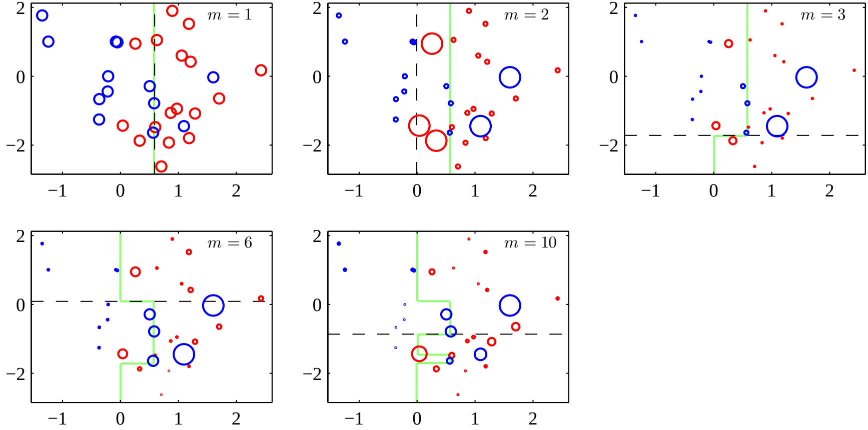
Boosting decision stumps

- Example of a boosting run (from Bishop)
- Green line = combined decision boundary of ensemble; dotted black line = decision boundary of most recent weak learner (decision stump)



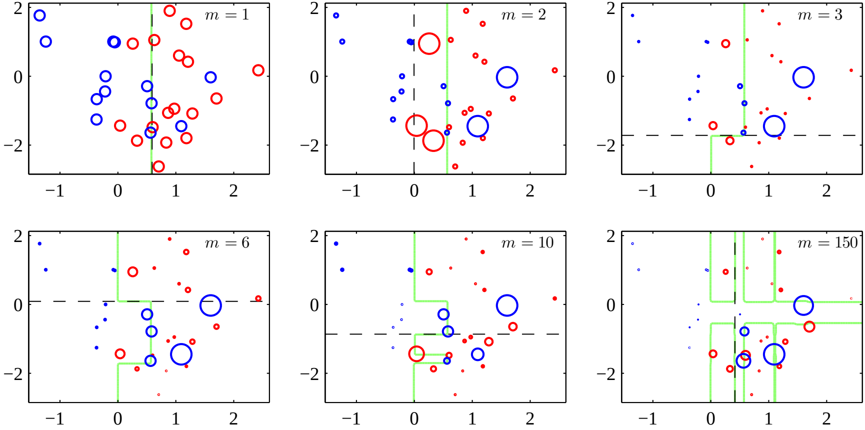
Boosting decision stumps

- Example of a boosting run (from Bishop)
- Green line = combined decision boundary of ensemble; dotted black line = decision boundary of most recent weak learner (decision stump)



Boosting decision stumps

- Example of a boosting run (from Bishop)
- Green line = combined decision boundary of ensemble; dotted black line = decision boundary of most recent weak learner (decision stump)



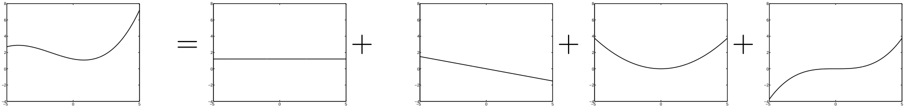
Combination of regressors

- Consider linear regression model

$$y = F(\mathbf{x};\mathbf{w}) = w_0\phi_0(\mathbf{x}) + w_1\phi_1(\mathbf{x}) + \ldots + w_d\phi_d(\mathbf{x})$$

- We can see this as a combination of $d+1$ simple regressors:

$$F(\mathbf{x};\mathbf{w}) = \sum_{j=0}^{d} f_j(\mathbf{x};\mathbf{w}), \qquad f_j(\mathbf{x};\mathbf{w}) \triangleq w_j\phi_j(\mathbf{x})$$



Forward stepwise regression

$$F(\mathbf{x};\mathbf{w}) = \sum_{j=0}^{d} f_j(\mathbf{x};\mathbf{w}), \qquad f_j(\mathbf{x};\mathbf{w}) \ = \ w_j\phi_j(\mathbf{x})$$

- We can build this combination greedily, one function at a time

Forward stepwise regression

$$F(\mathbf{x};\mathbf{w}) = \sum_{j=0}^{d} f_j(\mathbf{x};\mathbf{w}), \qquad f_j(\mathbf{x};\mathbf{w}) \;=\; w_j\phi_j(\mathbf{x})$$

- We can build this combination greedily, one function at a time
- New notation: Parameterize the set of functions: $f(\mathbf{x};\theta)$, $\theta = [w, j]$

Forward stepwise regression

$$F(\mathbf{x};\mathbf{w}) = \sum_{j=0}^{d} f_j(\mathbf{x};\mathbf{w}), \qquad f_j(\mathbf{x};\mathbf{w}) \;=\; w_j \phi_j(\mathbf{x})$$

- We can build this combination greedily, one function at a time
- New notation: Parameterize the set of functions: $f(\mathbf{x};\theta)$, $\theta = [w, j]$
- Our final model will be denoted

$$F_M(\mathbf{x};\theta_1,\ldots,\theta_M) \;=\; \sum_{m=1}^{M} f(\mathbf{x};\theta_m)$$

Forward stepwise regression

$$F(\mathbf{x};\mathbf{w}) = \sum_{j=0}^{d} f_j(\mathbf{x};\mathbf{w}), \qquad f_j(\mathbf{x};\mathbf{w}) \;=\; w_j \phi_j(\mathbf{x})$$

- We can build this combination greedily, one function at a time
- New notation: Parameterize the set of functions: $f(\mathbf{x};\theta)$, $\theta = [w, j]$
- Our final model will be denoted

$$F_M(\mathbf{x};\theta_1,\ldots,\theta_M) \;=\; \sum_{m=1}^{M} f(\mathbf{x};\theta_m)$$

- Step 1: fit the first simple model

$$\theta_1 \;=\; \operatorname*{argmin}_{\theta} \sum_{i=1}^{n} \left(y_i - f(\mathbf{x}_i;\theta)\right)^2$$

Forward stepwise regression

- Step 1: fit the first simple model

$$\theta_1 \;=\; \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{n} \left(y_i - f(\mathbf{x}_i;\theta)\right)^2$$

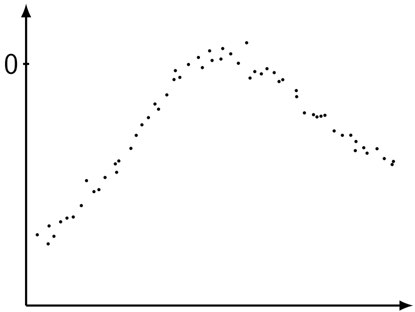
- Step 2: fit second simple model *to the residuals* of the first:

$$\theta_2 \;=\; \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{n} \Big(\underbrace{(y_i - f(\mathbf{x}_i;\theta_1))}_{\text{residual}} - f(\mathbf{x}_i;\theta)\Big)^2$$

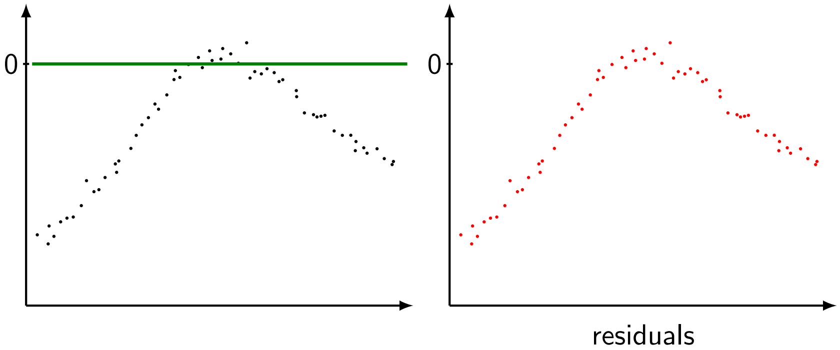
- ... Step m: fit a simple model to the residuals of the previous step, $y_i - F_{m-1}(\mathbf{x}_i;\theta_1,\ldots,\theta_{m-1})$
- Stop when no significant improvement in training error.
- Final estimate after M steps:

$$\hat{y}(\mathbf{x}) \;=\; F_M(\mathbf{x};\theta_1,\ldots,\theta_M) \;=\; \sum_{m=1}^{M} f(\mathbf{x};\theta_m)$$

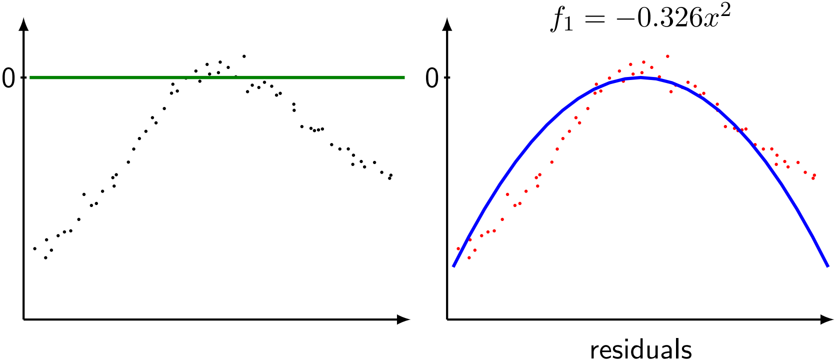
Stepwise regression



Stepwise regression



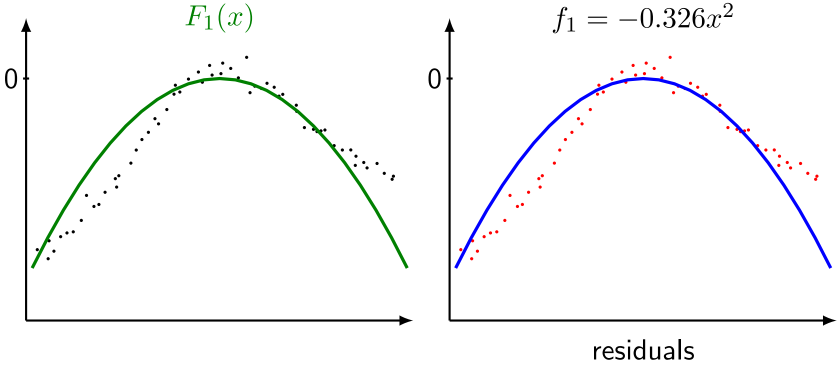
Stepwise regression



- Stepwise model fitting:

$$F(x) = -0.326x^2$$

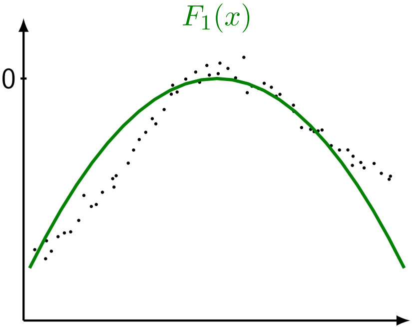
Stepwise regression

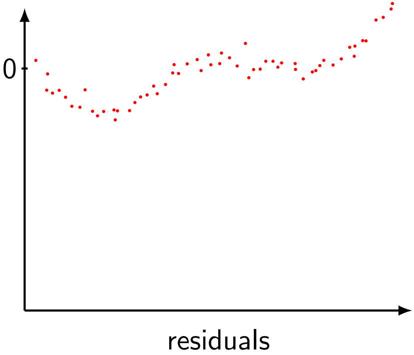


- Stepwise model fitting:

$$F(x) = -0.326x^2$$

Stepwise regression

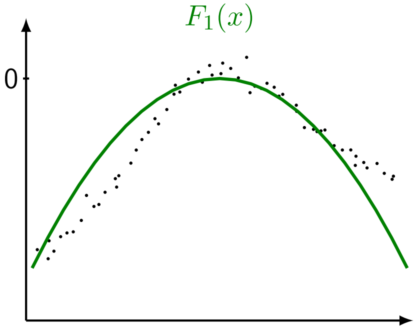


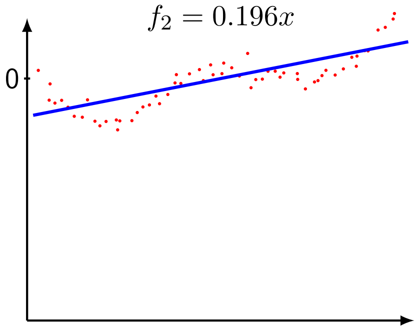


- Stepwise model fitting:

$$F(x) \;=\; -\,0.326x^2$$

Stepwise regression

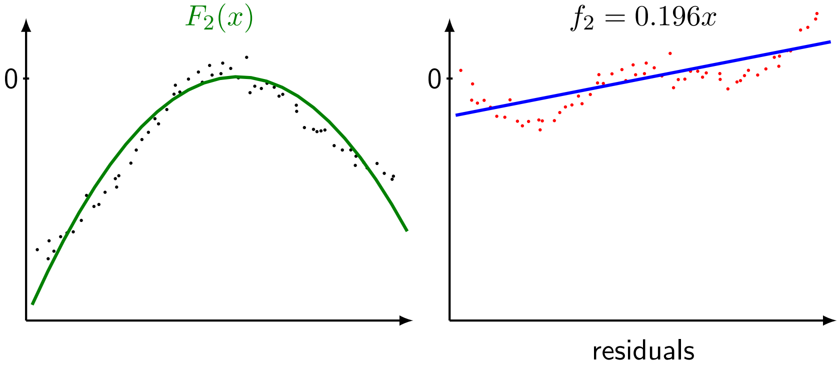




- Stepwise model fitting:

$$F(x) = -0.326x^2 + 0.196x$$

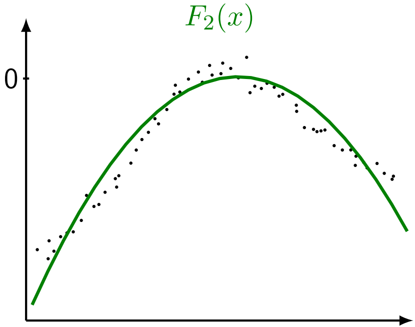
Stepwise regression

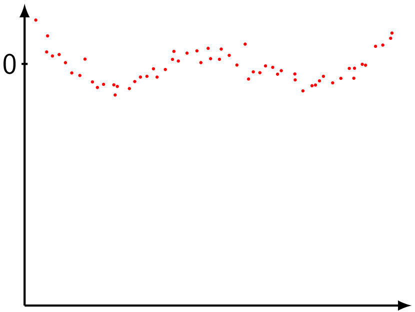


- Stepwise model fitting:

$$F(x) = -0.326x^2 + 0.196x$$

Stepwise regression



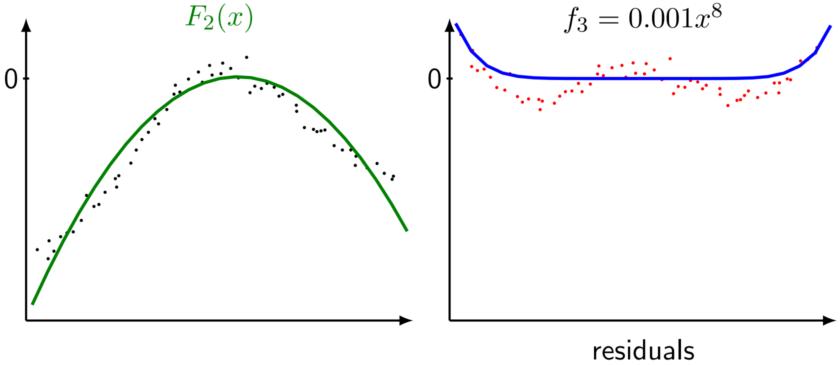


residuals

- Stepwise model fitting:

$$F(x) = -0.326x^2 + 0.196x$$

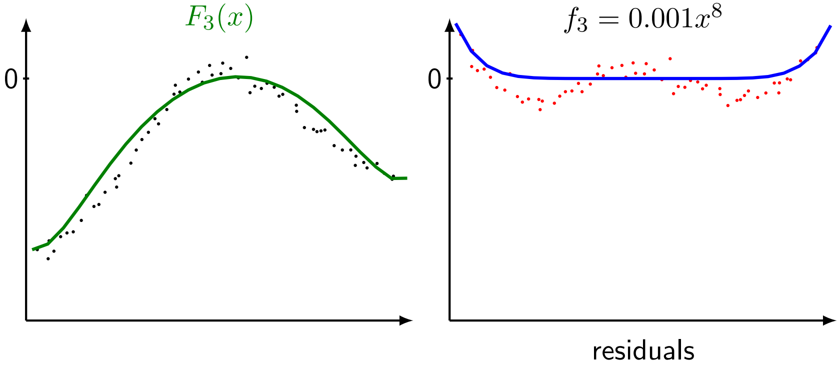
Stepwise regression



- Stepwise model fitting:

$$F(x) = -0.326x^2 + 0.196x + 0.0001x^8$$

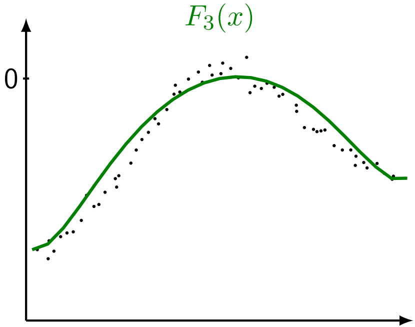
Stepwise regression

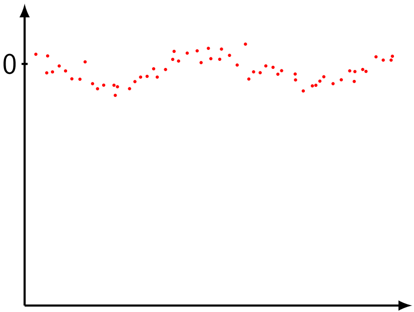


- Stepwise model fitting:

$$F(x) = -0.326x^2 + 0.196x + 0.0001x^8$$

Stepwise regression

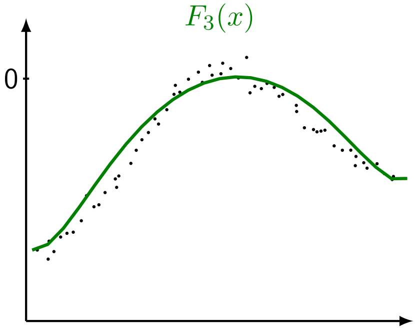


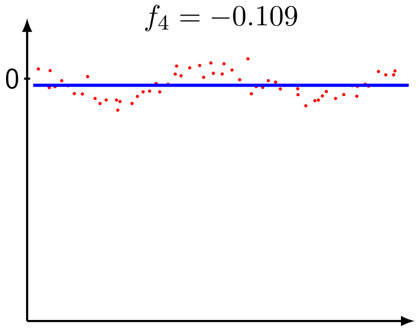


- Stepwise model fitting:

$$F(x) = -0.326x^2 + 0.196x + 0.0001x^8$$

Stepwise regression

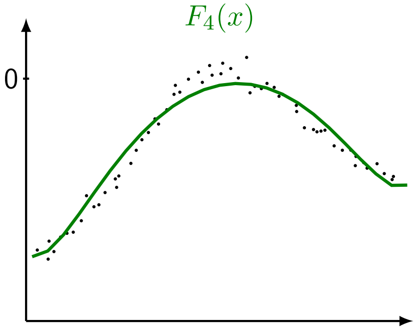


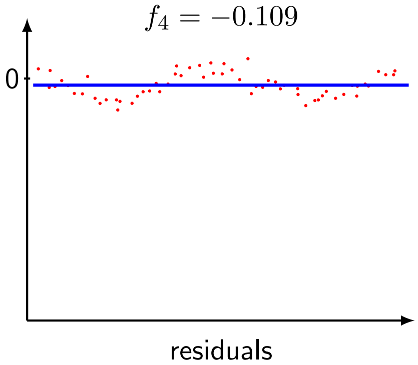


- Stepwise model fitting:

$$F(x) = -0.326x^2 + 0.196x + 0.0001x^8$$

Stepwise regression

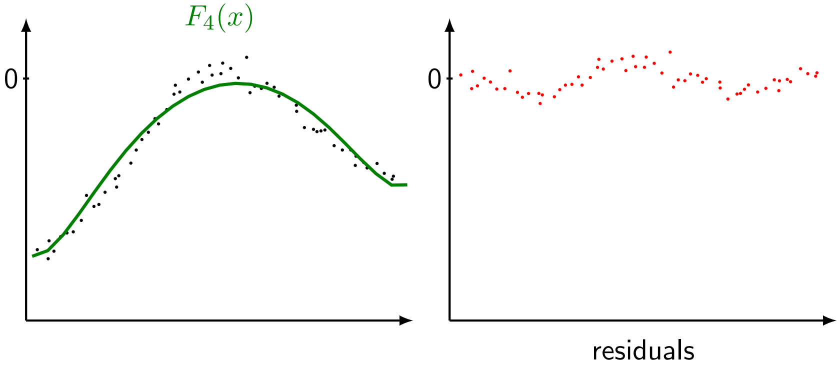




- Stepwise model fitting:

$$F(x) = -0.326x^2 + 0.196x + 0.0001x^8 - 0.109$$

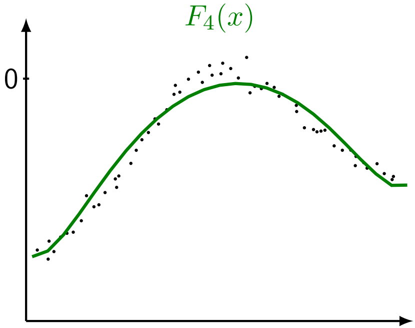
Stepwise regression

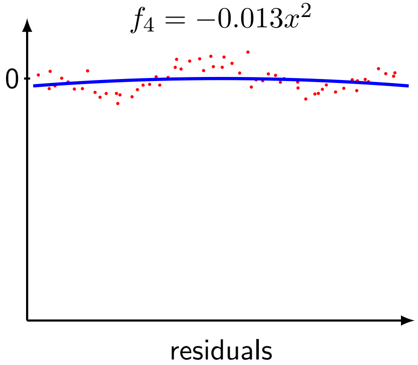


- Stepwise model fitting:

$$F(x) = -0.326x^2 + 0.196x + 0.0001x^8 - 0.109$$

Stepwise regression

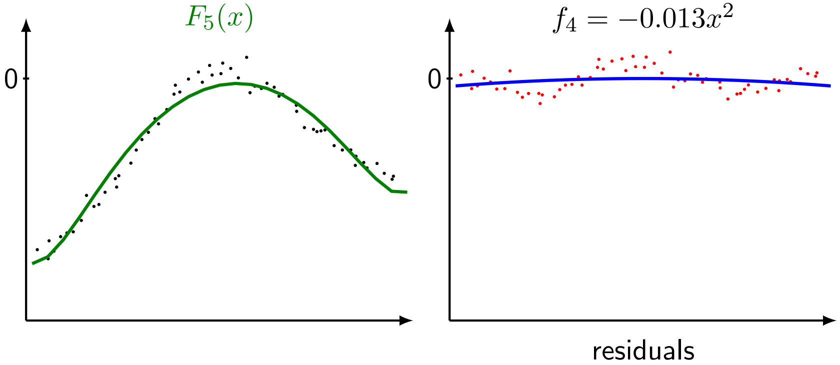




- Stepwise model fitting:

$$F(x) = -0.326x^2 + 0.196x + 0.0001x^8 - 0.109 - 0.013x^2$$

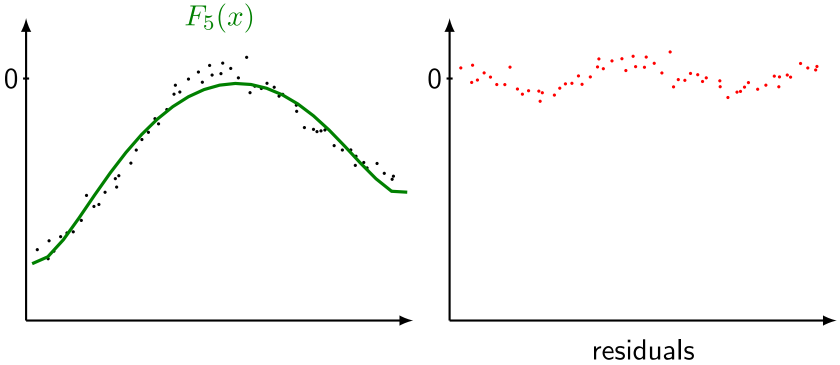
Stepwise regression



- Stepwise model fitting:

$$F(x) = -0.326x^2 + 0.196x + 0.0001x^8 - 0.109 - 0.013x^2$$

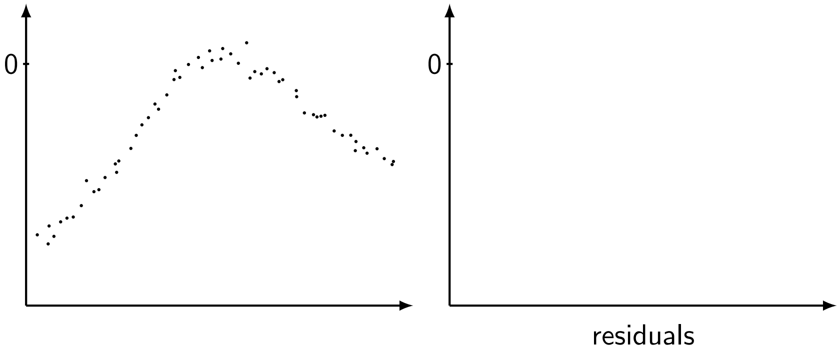
Stepwise regression



- Stepwise model fitting:

$$F(x) = -0.326x^2 + 0.196x + 0.0001x^8 - 0.109 - 0.013x^2$$

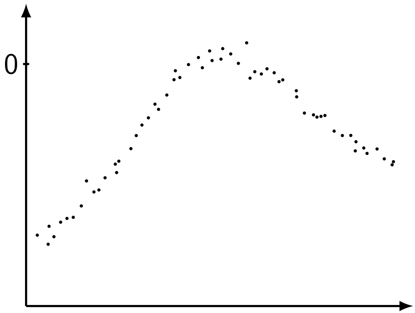
Stepwise regression

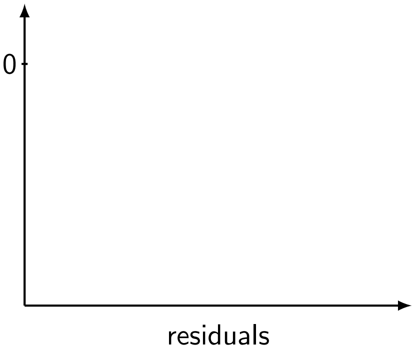


- Stepwise model fitting:

$$F(x) = -0.326x^2 + 0.196x + 0.0001x^8 - 0.109 - 0.013x^2$$

Stepwise regression

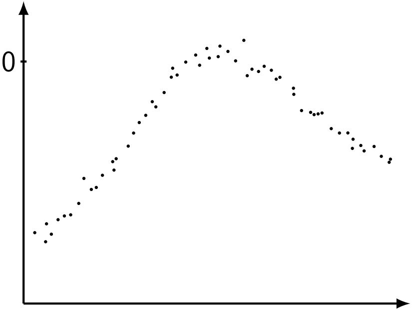




- Stepwise model fitting:

$$F(x) = -0.326x^2 + 0.196x + 0.0001x^8 - 0.109 - 0.013x^2$$
$$= -0.339x^2 + 0.196x + 0.0001x^8 - 0.109$$

Stepwise regression



- Stepwise model fitting:

$$
\begin{aligned}
F(x) &= \color{red}{-\,0.326x^2} + 0.196x + 0.0001x^8 - 0.109 \color{red}{-\,0.013x^2} \\
&= -0.339x^2 + 0.196x + 0.0001x^8 - 0.109
\end{aligned}
$$

- What are good stopping criteria?

Another look at stepwise fitting

- We want to fit the strong (ensemble) F to minimize

$$L(\mathbf{X}, \mathbf{y}; F) \;=\; \frac{1}{2}\sum_i (y_i - F(\mathbf{x}_i))^2$$

- Let's think of the values $F(\mathbf{x}_1), \ldots, F(\mathbf{x}_n)$ as parameters of L. Gradient of the loss w.r.t. these parameters:

$$\frac{\partial L(\mathbf{X}, \mathbf{y}; F)}{\partial F(\mathbf{x}_i)} \;=\; F(\mathbf{x}_i) - y_i$$

  so the residuals $y_i - F(\mathbf{x}_i)$ are the negative gradients of the loss!

- Recall: in stepwise regression we fit $f(\mathbf{x}; \theta_m)$ to residuals $\{y_i - F_{m-1}(\mathbf{x}_i; \theta_1, \ldots, \theta_{m-1})\}$

Gradient boosting: generic algorithm

- Start with initial model, say, best constant fit

Gradient boosting: generic algorithm

- Start with initial model, say, best constant fit

$$F_1(\mathbf{x}) = \frac{1}{n}\sum_i y_i$$

Gradient boosting: generic algorithm

- Start with initial model, say, best constant fit

$$F_1(\mathbf{x}) \;=\; \frac{1}{n}\sum_i y_i$$

- For $m = 1, \ldots$: until convergence
  Calculate negative gradients for each $i = 1, \ldots, n$

$$-g(\mathbf{x}_i) \;=\; -\frac{\partial L(y_i, F_m(\mathbf{x}_i))}{\partial F_m(\mathbf{x}_i)} \;=\; y_i - F_m(\mathbf{x}_i)$$

  fit (least squares) a regression function f_{m+1} to negative gradients

$$f_{m+1}(\mathbf{x}_i) \;\approx\; -g(\mathbf{x}_i)$$

  update model by making a step in the direction of negative gradient

$$F_{m+1} \;=\; F_m + \eta f_{m+1}$$

Gradient boosting: generic algorithm

- Start with initial model, say, best constant fit

$$F_1(\mathbf{x}) \;=\; \frac{1}{n}\sum_i y_i$$

- For $m = 1, \ldots$: until convergence
  Calculate negative gradients for each $i = 1, \ldots, n$

$$-g(\mathbf{x}_i) \;=\; -\frac{\partial L(y_i, F_m(\mathbf{x}_i))}{\partial F_m(\mathbf{x}_i)} \;=\; y_i - F_m(\mathbf{x}_i)$$

  fit (least squares) a regression function f_{m+1} to negative gradients

$$f_{m+1}(\mathbf{x}_i) \;\approx\; -g(\mathbf{x}_i)$$

  update model by making a step in the direction of negative gradient

$$F_{m+1} \;=\; F_m + \eta f_{m+1}$$

- Can modify L, derive new gradient boosting algorithm

Gradient boosting for classification

- With C-way classification, F predicts a *score matrix*

$$\begin{array}{ccc} F_1(\mathbf{x}_1) & \cdots & F_C(\mathbf{x}_1) \\ F_1(\mathbf{x}_2) & \cdots & F_C(\mathbf{x}_2) \\ \cdots & \cdots & \cdots \\ F_1(\mathbf{x}_n) & \cdots & F_C(\mathbf{x}_n) \end{array}$$

  which can be converted to posteriors
  $p\,(y_i = c \,|\, \mathbf{x}_i) = e^{F_c(\mathbf{x}_i)} / \sum_k e^{F_k(\mathbf{x}_i)}$
- Negative gradients are also a matrix, with entry at (c, i)

$$-g_c(\mathbf{x}_i) \;=\; -\frac{\partial L}{\partial F_c(\mathbf{x}_i)}$$

- Gradient boosting: start with $F_1^{(1)}, \ldots, F_C^{(1)}$, at iter. m fit $f_c^{(m+1)}$ to negative gradients $-g_c(\mathbf{x}_1), \ldots, -g_c(\mathbf{x}_n)$

Gradient boosting: step size

- Model update in GB:

$$F_{m+1} = F_m + \eta f_{m+1}$$

  How do we choose η?

- An aggressive strategy: steepest descent in the direction of f_{m+1}
- Once we fit f_{m+1} to the function gradient, solve

$$\eta_m = \operatorname*{argmin}_{\eta} \sum_i L\left(y_i, F_m(\mathbf{x}_i) + \eta f_{m+1}(\mathbf{x}_i)\right)$$

- Alternative strategies (may regularize better): fixed $\eta < 1$, or decaying η_m
- Note computational tradeoffs (lower η will likely require more trees)

Gradient tree boosting

- Very successful/popular approach: gradient boosting with CART trees
- Widely used tool: XGBoost

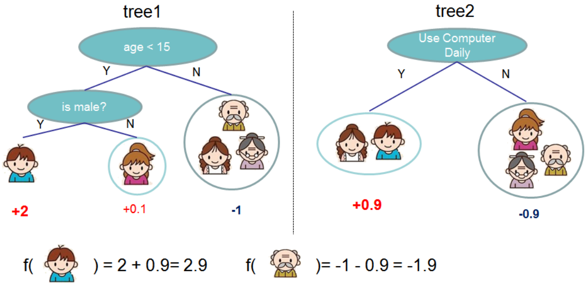


figure: Chen & Guestrin (XGBoost)

Ensemble methods: summary so far

- Main benefit of ensembles: control overfitting
- Boosting: build ensemble of low-variance, high-bias predictors sequentially
  AdaBoost: binary classification, exponential surrogate loss
  gradient boosting: more general framework
- Bagging: build ensemble of high-variance, low-bias predictors in parallel
  use randomness and averaging to reduce variance (e.g., random forests)
- Relatively straightforward to tune; robust
- Not very interpretable
- Computationally expensive (train *and* test time)

Reminder: optimal classification

- Expected classification error is minimized by

$$h(\mathbf{x}) = \underset{c}{\operatorname{argmax}}\, p\,(y = c \,|\, \mathbf{x})$$

- The **Bayes classifier**:

$$h^*(\mathbf{x}) \;=\; \underset{c}{\operatorname{argmax}}\, p\,(y = c \,|\, \mathbf{x})$$

Reminder: optimal classification

- Expected classification error is minimized by

$$h(\mathbf{x}) = \underset{c}{\operatorname{argmax}}\, p\,(y = c \,|\, \mathbf{x})$$

- The **Bayes classifier**:

$$\begin{aligned} h^*(\mathbf{x}) &= \underset{c}{\operatorname{argmax}}\, p\,(y = c \,|\, \mathbf{x}) \\ &= \underset{c}{\operatorname{argmax}}\, \frac{p\,(\mathbf{x} \,|\, y = c)\, p(y = c)}{p(\mathbf{x})} \end{aligned}$$

Reminder: optimal classification

- Expected classification error is minimized by

$$h(\mathbf{x}) = \operatorname*{argmax}_c p\,(y = c \,|\, \mathbf{x})$$

- The **Bayes classifier**:

$$\begin{aligned} h^*(\mathbf{x}) &= \operatorname*{argmax}_c p\,(y = c \,|\, \mathbf{x}) \\ &= \operatorname*{argmax}_c \frac{p\,(\mathbf{x} \,|\, y = c)\, p(y = c)}{p(\mathbf{x})} \\ &= \operatorname*{argmax}_c p\,(\mathbf{x} \,|\, y = c)\, p(y = c) \end{aligned}$$

Note: $p(\mathbf{x}) = \sum_c p(\mathbf{x}, y = c)$ is equal for all c and can be ignored

Reminder: optimal classification

- Expected classification error is minimized by

$$h(\mathbf{x}) = \operatorname*{argmax}_c p\,(y = c \,|\, \mathbf{x})$$

- The **Bayes classifier**:

$$\begin{aligned}
h^*(\mathbf{x}) &= \operatorname*{argmax}_c p\,(y = c \,|\, \mathbf{x}) \\
&= \operatorname*{argmax}_c \frac{p\,(\mathbf{x} \,|\, y = c)\, p(y = c)}{p(\mathbf{x})} \\
&= \operatorname*{argmax}_c p\,(\mathbf{x} \,|\, y = c)\, p(y = c) \\
&= \operatorname*{argmax}_c \left\{\log p\,(\mathbf{x} \,|\, y = c) \,+\, \log p(y = c)\right\}
\end{aligned}$$

Note: $p(\mathbf{x}) = \sum_c p(\mathbf{x}, y = c)$ is equal for all c and can be ignored

Bayes risk

- The risk (probability of error) of Bayes classifier h^* is called the **Bayes risk** R^*
- This is the *minimal achievable* risk for the given $p(\mathbf{x}, y)$ with any classifier!
- In a sense, R^* measures the inherent difficulty of the classification problem.

$$R^* \; = 1 - \int_{\mathbf{x}} \max_c \left\{ p\left(\mathbf{x} \mid c = y\right) \, p(y = c) \right\} d\mathbf{x}$$

Discriminant functions

- We can construct, for each class c, a **discriminant function**

$$\delta_c(\mathbf{x}) \triangleq \log p(\mathbf{x} \mid y = c) + \log p(y = c)$$

such that

$$h^*(\mathbf{x}) = \operatorname*{argmax}_c \delta_c(\mathbf{x}).$$

- Can simplify δ_c by removing terms and factors common for all δ_c since they won't affect the decision boundary
For example, if $p(y = c) = 1/C$ for all c, can drop the prior term:

$$\delta_c(\mathbf{x}) = \log p(\mathbf{x} \mid y = c)$$

Two-category case

- In case of two classes $y \in \{\pm 1\}$, the Bayes classifier is

$$h^*(\mathbf{x}) \;=\; \underset{c=\pm 1}{\operatorname{argmax}}\, \delta_c(\mathbf{x}) \;= \operatorname{sign}\left(\delta_{+1}(\mathbf{x}) - \delta_{-1}(\mathbf{x})\right)$$

- Decision boundary is given by $\delta_{+1}(\mathbf{x}) - \delta_{-1}(\mathbf{x}) = 0$
  - Sometimes $f(\mathbf{x}) = \delta_{+1}(\mathbf{x}) - \delta_{-1}(\mathbf{x})$ is referred to as a discriminant function
- With equal priors, this is equivalent to the **(log)-likelihood ratio test**:

$$h^*(\mathbf{x}) \;=\; \operatorname{sign}\left[\log \frac{p\left(\mathbf{x} \,|\, y=+1\right)}{p\left(\mathbf{x} \,|\, y=-1\right)}\right]$$

Equal covariance Gaussian case

- Consider the case of $p_c(\mathbf{x}) = \mathcal{N}(\mathbf{x};\, \boldsymbol{\mu}_c, \boldsymbol{\Sigma})$, and equal prior for all classes.

$$
\begin{aligned}
\delta_k(x) &= \log p(\mathbf{x} \,|\, y = k) \\
&= \underbrace{-\log(2\pi)^{d/2} - \frac{1}{2}\log(|\boldsymbol{\Sigma}|)}_{\text{same for all } k} - \frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}_k)
\end{aligned}
$$

Equal covariance Gaussian case

- Consider the case of $p_c(\mathbf{x}) = \mathcal{N}(\mathbf{x};\, \boldsymbol{\mu}_c, \boldsymbol{\Sigma})$, and equal prior for all classes.

$$
\begin{aligned}
\delta_k(x) &= \log p(\mathbf{x} \mid y = k) \\
&= \underbrace{-\log(2\pi)^{d/2} - \frac{1}{2}\log(|\boldsymbol{\Sigma}|)}_{\text{same for all } k} - \frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}_k) \\
&\propto \text{const} - \underbrace{\mathbf{x}^\top \boldsymbol{\Sigma}^{-1}\mathbf{x}}_{\text{same for all } k} + \boldsymbol{\mu}_k^\top \boldsymbol{\Sigma}^{-1}\mathbf{x} + \mathbf{x}^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k - \boldsymbol{\mu}_k^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k
\end{aligned}
$$

Equal covariance Gaussian case

- Consider the case of $p_c(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_c, \boldsymbol{\Sigma})$, and equal prior for all classes.

$$\begin{aligned}
\delta_k(x) &= \log p(\mathbf{x} \,|\, y = k) \\
&= \underbrace{-\log(2\pi)^{d/2} - \frac{1}{2}\log(|\boldsymbol{\Sigma}|)}_{\text{same for all } k} - \frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}_k) \\
&\propto \text{const} - \underbrace{\mathbf{x}^\top \boldsymbol{\Sigma}^{-1}\mathbf{x}}_{\text{same for all } k} + \boldsymbol{\mu}_k^\top \boldsymbol{\Sigma}^{-1}\mathbf{x} + \mathbf{x}^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k - \boldsymbol{\mu}_k^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k
\end{aligned}$$

- Now consider two classes r and q:

$$\delta_r(\mathbf{x}) \propto 2\boldsymbol{\mu}_r^\top \boldsymbol{\Sigma}^{-1}\mathbf{x} - \boldsymbol{\mu}_r^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_r$$

$$\delta_q(\mathbf{x}) \propto 2\boldsymbol{\mu}_q^\top \boldsymbol{\Sigma}^{-1}\mathbf{x} - \boldsymbol{\mu}_q^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_q$$

Linear discriminant

- Two-class discriminants (contest between two classes):

$$\begin{aligned} \delta_r(\mathbf{x}) - \delta_q(\mathbf{x}) &= 2\boldsymbol{\mu}_r^\top \boldsymbol{\Sigma}^{-1}\mathbf{x} - \boldsymbol{\mu}_r^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_r \\ &\quad - 2\boldsymbol{\mu}_q^\top \boldsymbol{\Sigma}^{-1}\mathbf{x} - \boldsymbol{\mu}_q^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_q \end{aligned}$$

Linear discriminant

- Two-class discriminants (contest between two classes):

$$\begin{aligned}\delta_r(\mathbf{x}) - \delta_q(\mathbf{x}) &= 2\boldsymbol{\mu}_r^\top \boldsymbol{\Sigma}^{-1}\mathbf{x} - \boldsymbol{\mu}_r^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_r \\ &\quad - 2\boldsymbol{\mu}_q^\top \boldsymbol{\Sigma}^{-1}\mathbf{x} - \boldsymbol{\mu}_q^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_q \\ &= \mathbf{w}\cdot\mathbf{x} + b\end{aligned}$$

- If we know what $\boldsymbol{\mu}_{1,\ldots,C}$ and $\boldsymbol{\Sigma}$ are, we can compute the optimal $\mathbf{w}$, b directly
- What should we do when we don't know the Gaussians?

Linear discriminant

- Two-class discriminants (contest between two classes):

$$
\begin{aligned}
\delta_r(\mathbf{x}) - \delta_q(\mathbf{x}) &= 2\boldsymbol{\mu}_r^\top \boldsymbol{\Sigma}^{-1}\mathbf{x} - \boldsymbol{\mu}_r^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_r \\
&\quad - 2\boldsymbol{\mu}_q^\top \boldsymbol{\Sigma}^{-1}\mathbf{x} - \boldsymbol{\mu}_q^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_q \\
&= \mathbf{w} \cdot \mathbf{x} + b
\end{aligned}
$$

- If we know what $\boldsymbol{\mu}_{1,\ldots,C}$ and $\boldsymbol{\Sigma}$ are, we can compute the optimal $\mathbf{w}$, b directly
- What should we do when we don't know the Gaussians?
- Estimate $\{\boldsymbol{\mu}_c\}$, $\boldsymbol{\Sigma}$ – how?

Linear discriminant

- Two-class discriminants (contest between two classes):

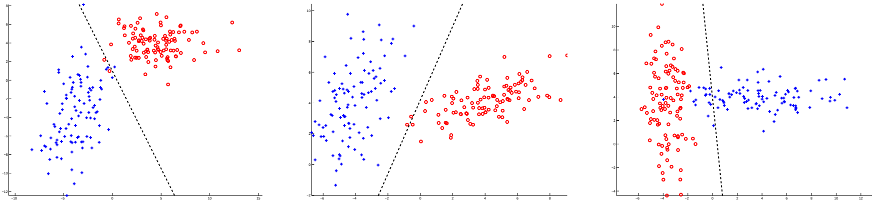
$$\begin{aligned}\delta_r(\mathbf{x}) - \delta_q(\mathbf{x}) &= 2\boldsymbol{\mu}_r^\top \boldsymbol{\Sigma}^{-1}\mathbf{x} - \boldsymbol{\mu}_r^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_r \\ &\quad - 2\boldsymbol{\mu}_q^\top \boldsymbol{\Sigma}^{-1}\mathbf{x} - \boldsymbol{\mu}_q^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_q \\ &= \mathbf{w}\cdot\mathbf{x} + b\end{aligned}$$

- If we know what $\boldsymbol{\mu}_{1,\ldots,C}$ and $\boldsymbol{\Sigma}$ are, we can compute the optimal $\mathbf{w}$, b directly
- What should we do when we don't know the Gaussians?
- Estimate $\{\boldsymbol{\mu}_c\}$, $\boldsymbol{\Sigma}$ – how?
  ML estimate for each $\boldsymbol{\mu}_c$; estimate $\boldsymbol{\Sigma}$ e.g., by

$$\frac{1}{n}\sum_i (\mathbf{x}_i - \boldsymbol{\mu}_{y_i})^\top (\mathbf{x}_i - \boldsymbol{\mu}_{y_i})$$

Generative models for classification

- In generative models one explicitly models $p(\mathbf{x}, y)$ or, equivalently, $p\left(\mathbf{x} \,|\, y = c\right)$ and $p(y = c)$, to derive discriminants.
- Typically, the model imposes certain parametric form on the assumed distributions, and requires estimation of the parameters from data.
  - Most popular: Gaussian for continuous, multinomial for discrete.
  - Will also see *non-parametric* models.
- Often, the classifier is OK even if data clearly don't conform to assumptions.



Gaussians with unequal covariances

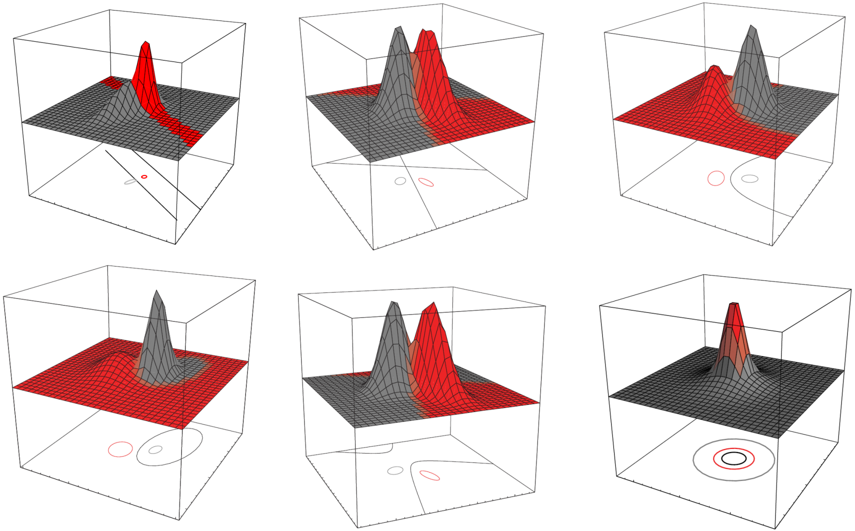
- What if we remove the restriction that $\forall c,\ \mathbf{\Sigma}_c = \mathbf{\Sigma}$?
- Compute ML estimate for $\boldsymbol{\mu}_c, \mathbf{\Sigma}_c$ for each c.
- We get discriminants (and decision boundaries) *quadratic* in $\mathbf{x}$:

$$\delta_c(\mathbf{x}) \;=\; -\frac{1}{2}\mathbf{x}^\top \mathbf{\Sigma}_c^{-1}\mathbf{x} + \boldsymbol{\mu}_c^\top \mathbf{\Sigma}_c^{-1}\mathbf{x} - \mathrm{const}_c(\mathbf{x})$$

- Decision boundary with two classes:

$$\delta_{1} - \delta_{0} \;=\; 0$$

Quadratic decision boundaries



[Duda, Hart & Stork]

Naïve Bayes classifier

- Suppose $\mathbf{x}$ is represented by m features $\phi_1(\mathbf{x}), \ldots, \phi_m(\mathbf{x})$.
- NB assumes that the features are *independent* given the class:

$$p(\mathbf{x} \mid c) \;=\; p(\phi_1(\mathbf{x}), \ldots, \phi_m(\mathbf{x}) \mid c) \;=\; \prod_{j=1}^{m} p(\phi_j(\mathbf{x}) \mid c)\,.$$

- Under this assumption, the Bayes classifier is

$$h^*(\mathbf{x}) \;=\; \operatorname{sign}\left[\sum_{j=1}^{m} \log \frac{p(\phi_j(\mathbf{x}) \mid +1)}{p(\phi_j(\mathbf{x}) \mid -1)} + \underbrace{\log p(y=1) - \log p(y=-1)}_{\text{(log)prior over classes}}\right].$$

Naïve Bayes for Gaussian model

$$p\left(\mathbf{x}\,|\,c\right) \;=\; p\left(\phi_1(\mathbf{x}),\ldots,\phi_m(\mathbf{x})\,|\,c\right) \;=\; \prod_{j=1}^{m} p\left(\phi_j(\mathbf{x})\,|\,c\right).$$

- Assume Gaussian feature class-conditional

  $$p(\phi_j(\mathbf{x})|c) \;=\; \mathcal{N}(\cdot;\mu_j,\sigma_j^2)$$

  NB assumption of independence is equivalent to

  $$\mathbf{\Sigma} = \begin{bmatrix} \sigma_1^2 & 0 & \ldots & 0 \\ 0 & \sigma_2^2 & \ldots & 0 \\ . & . & . & . & . & . & . & . & . & . & . \\ 0 & & . & . & . & . & \sigma_d^2 \end{bmatrix}$$

- Need to estimate the d marginal 1D Gaussian densities (one for each component of $\mathbf{x}$).

Example: generative models for documents

- A common task: given an e-mail message, classify it as SPAM or "ham" (a legitimate e-mail).
- Define a set of keywords $W_1, \ldots, W_m$.

$$\phi_j(\mathbf{x}) \;=\; \begin{cases} 1 & \text{document } \mathbf{x} \text{ includes } W_j, \\ 0 & \text{otherwise.} \end{cases}$$

- A document $\mathbf{x}$ (of arbitrary length!) is now represented as a vector in $\{0,1\}^m$: $\Phi(\mathbf{x}) = [\phi_1(\mathbf{x}), \ldots, \phi_m(\mathbf{x})]^T$.
- A natural distribution for $\phi_j(\mathbf{x})$ is *Bernoulli*: $p(\phi_j(\mathbf{x}) = 1;\, \theta_j) \;=\; \theta_j$

Example: generative models for documents

- A common task: given an e-mail message, classify it as SPAM or "ham" (a legitimate e-mail).
- Define a set of keywords $W_1, \ldots, W_m$.

$$\phi_j(\mathbf{x}) \;=\; \begin{cases} 1 & \text{document } \mathbf{x} \text{ includes } W_j, \\ 0 & \text{otherwise.} \end{cases}$$

- A document $\mathbf{x}$ (of arbitrary length!) is now represented as a vector in $\{0,1\}^m$: $\Phi(\mathbf{x}) = [\phi_1(\mathbf{x}), \ldots, \phi_m(\mathbf{x})]^T$.
- A natural distribution for $\phi_j(\mathbf{x})$ is *Bernoulli*: $p(\phi_j(\mathbf{x}) = 1;\, \theta_j) \;=\; \theta_j$

$$p\,(\phi_j \,|\, y = 1) \;=\; \theta_{j1}^{\phi_j}(1 - \theta_{j1})^{1-\phi_j},$$

$$p\,(\phi_j \,|\, y = 0) \;=\; \theta_{j0}^{\phi_j}(1 - \theta_{j0})^{1-\phi_j}.$$

Application: SPAM detection

- Given an e-mail message need to classify it as SPAM ($y = 1$) or "ham" ($y = 0$), based on the content.
- An important problem! P_1 pretty high...
- Typical binary features:
  - keywords;
  - HTML tags and patterns;
  - SCREAMING LINES (ALL CAPS);
  - number of recipients above certain threshold;
  - Comes from "blacklisted" relay...

SPAM detection with Naïve Bayes

- For simplicity, we will write ϕ_j instead of $\phi_j(\mathbf{x})$.
- For a single binary feature ϕ_j,

$$p\,(\phi_j\,|\,y=1) \;=\; \theta_{j1}^{\phi_j}(1-\theta_{j1})^{1-\phi_j},$$

$$p\,(\phi_j\,|\,y=0) \;=\; \theta_{j0}^{\phi_j}(1-\theta_{j0})^{1-\phi_j}.$$

- ML estimate of a Bernoulli variable:
  k SPAM documents with $\phi_j = 1$, and $N-k$ without $\Rightarrow \hat{\theta}_{j1} \;=\; k/N$.

Classifying a document

- Given new document $\mathbf{x} = [\phi_1, \ldots, \phi_m]^T$:

$$\hat{y} = 1 \Leftrightarrow \sum_{j=1}^{m} \phi_j \log \theta_{j1} + \sum_{j=1}^{m} (1 - \phi_j) \log(1 - \theta_{j1}) - \sum_{j=1}^{m} \phi_j \log \theta_{j0} - \sum_{j=1}^{m} (1 - \phi_j) \log(1 - \theta_{j0}) + \log p(y = 1) - \log p(y = 0) \ \geq \ 0.$$

- There are total of $2 + 2m$ parameters to estimate in this model.

Problems with ML estimation

- Recall the coin-tossing experiments:
  - ML is too sensitive to the data, and may violate some "reasonable" beliefs about θ, e.g., that $\theta = 1$ is very unlikely.
- A real problem in text classification. *Zipf's law* for English texts: the n-th most common word has relative frequency of $1/n^a$, with $a \approx 1$.
  - relative frequency means $\#$(this word)$/\#$(all words)
- According to ML, when a word appears in a message that we have never seen in SPAM, we *must* decide it's legit.
- If the same message contains a word never seen in non-SPAM, what do we do?