COMP 652: Machine Learning

Lecture 22

Today

- □ Generalization error and the bias-variance-noise decomposition
- □ Bias-variance trade-off
- □ Estimating bias and variance
- □ Bagging
- □ Boosting

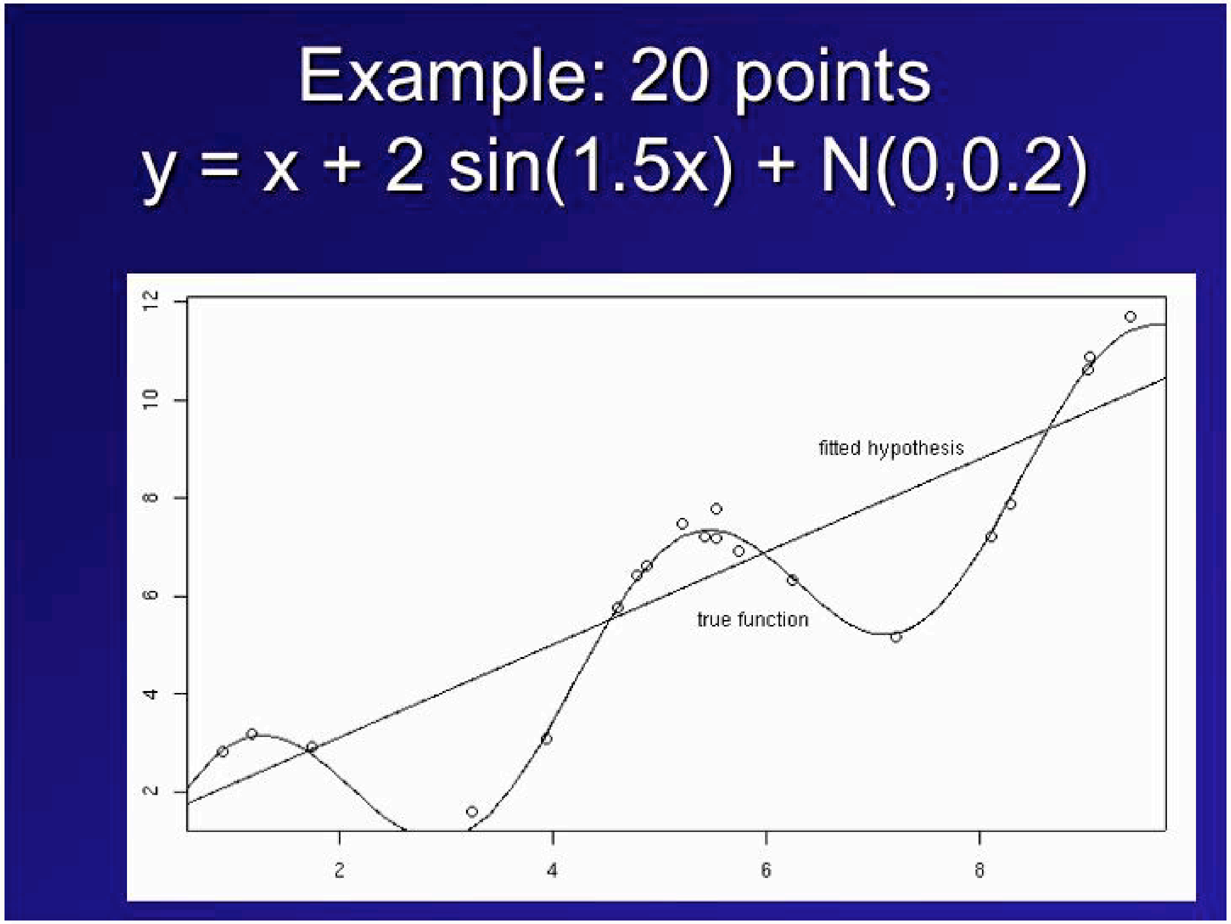
Linear regression revisited

- Suppose we have examples $\langle \mathbf{x}, y \rangle$ where $y = f(\mathbf{x}) + \epsilon$ and ϵ is Gaussian noise with zero mean and standard deviation σ
- In linear regression, given a set of examples $\langle \mathbf{x_i}, y_i \rangle_{i=1\ldots m}$, we fit a linear hypothesis $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$, such as to minimize sum-squared error over the training data:

$$\sum_{i=1}^{m}(y_i - h(\mathbf{x}_i))^2$$

- Because of the hypothesis class that we chose (linear hypotheses) for some functions f we will have a *systematic* prediction error
- Depending on the data set we have, the parameters $\mathbf{w}$ that we find will be different

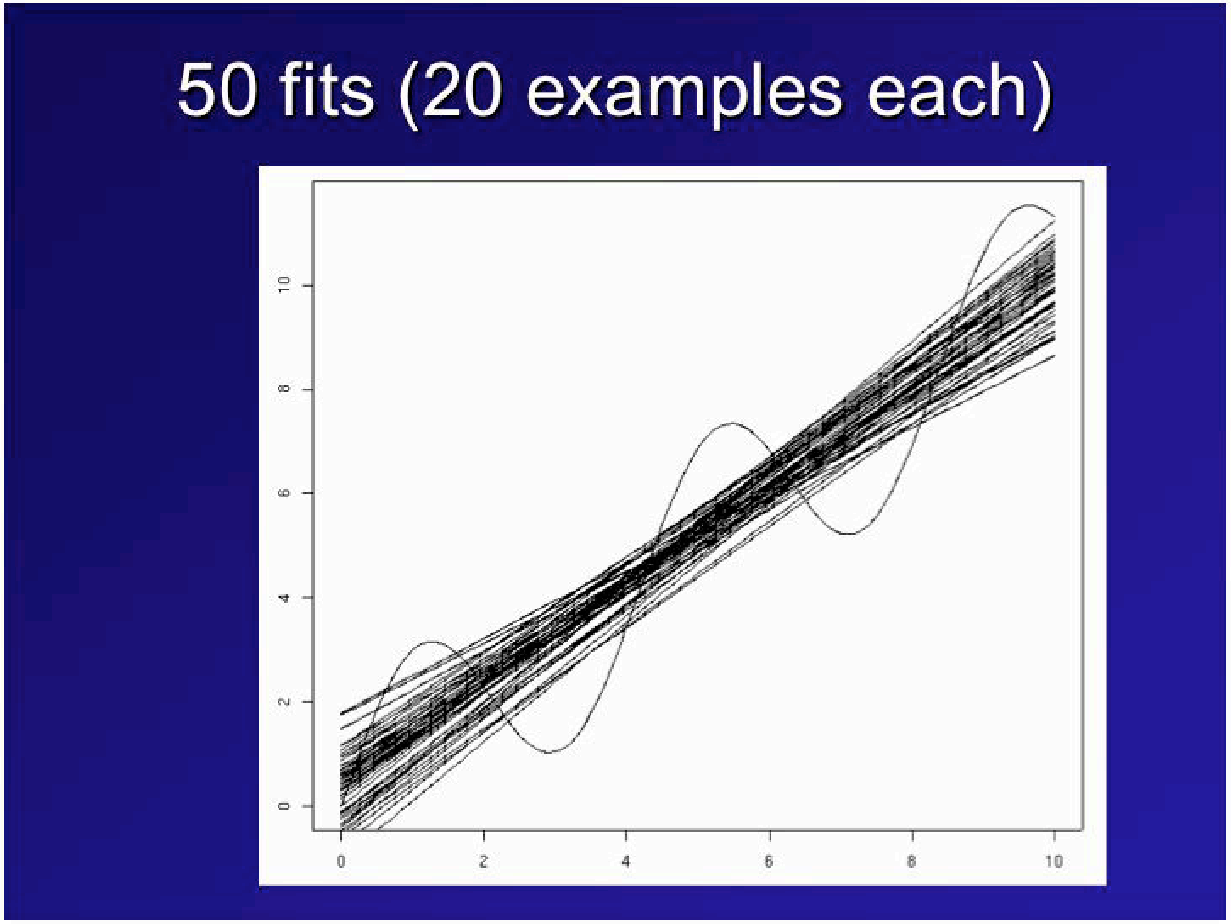
An example (Tom Dietterich)


Example: 20 points

$y = x + 2\sin(1.5x) + N(0,0.2)$

The sine is the true function, the circles are the data points and the straight line is the linear regression fit

Example continued



With different sets of 20 points, we get different lines

Bias-variance analysis

- Given a new data point $\mathbf{x}$, what is the **expected prediction error**?
- Assume that the data points are drawn i.i.d. from *a unique underlying probability distribution P*
- The goal of the analysis is to compute, for an arbitrary new point $\mathbf{x}$,

$$E_P\left[(y - h(\mathbf{x}))^2\right]$$

  where y is the value of $\mathbf{x}$ that could be present in a data set, and the expectation is over all *all training sets* (of a certain size) drawn according to P
- We will decompose this expectation into three components: bias, variance and noise

Bias-variance decomposition

- First, a "variance lemma": $E(X^2) = \text{Var}(X) + (E(X))^2$
- Then:

$$\begin{aligned} E_P\left[(y - h(\mathbf{x}))^2\right] &= E_P\left[(h(\mathbf{x}))^2 - 2yh(\mathbf{x}) + y^2\right] \\ &= E_P\left[(h(\mathbf{x}))^2\right] + E_P\left[y^2\right] - 2E_P[y]E_P\left[h(\mathbf{x})\right] \end{aligned}$$

- Let $\bar{h}(\mathbf{x}) = E_P[h(\mathbf{x})]$ denote the *mean prediction* of the hypothesis at $\mathbf{x}$, when h is trained with data drawn from P
- For the first term, using the variance lemma, we have:

$$E_P[(h(\mathbf{x}))^2] = E_P[(h(\mathbf{x}) - \bar{h}(\mathbf{x}))^2] + (\bar{h}(\mathbf{x}))^2$$

- Note that $E_P[y] = E_P[f(\mathbf{x}) + \epsilon] = f(\mathbf{x})$
- For the second term, using the variance lemma, we have:

$$E[y^2] = E[(y - f(\mathbf{x}))^2] + (f(\mathbf{x}))^2$$

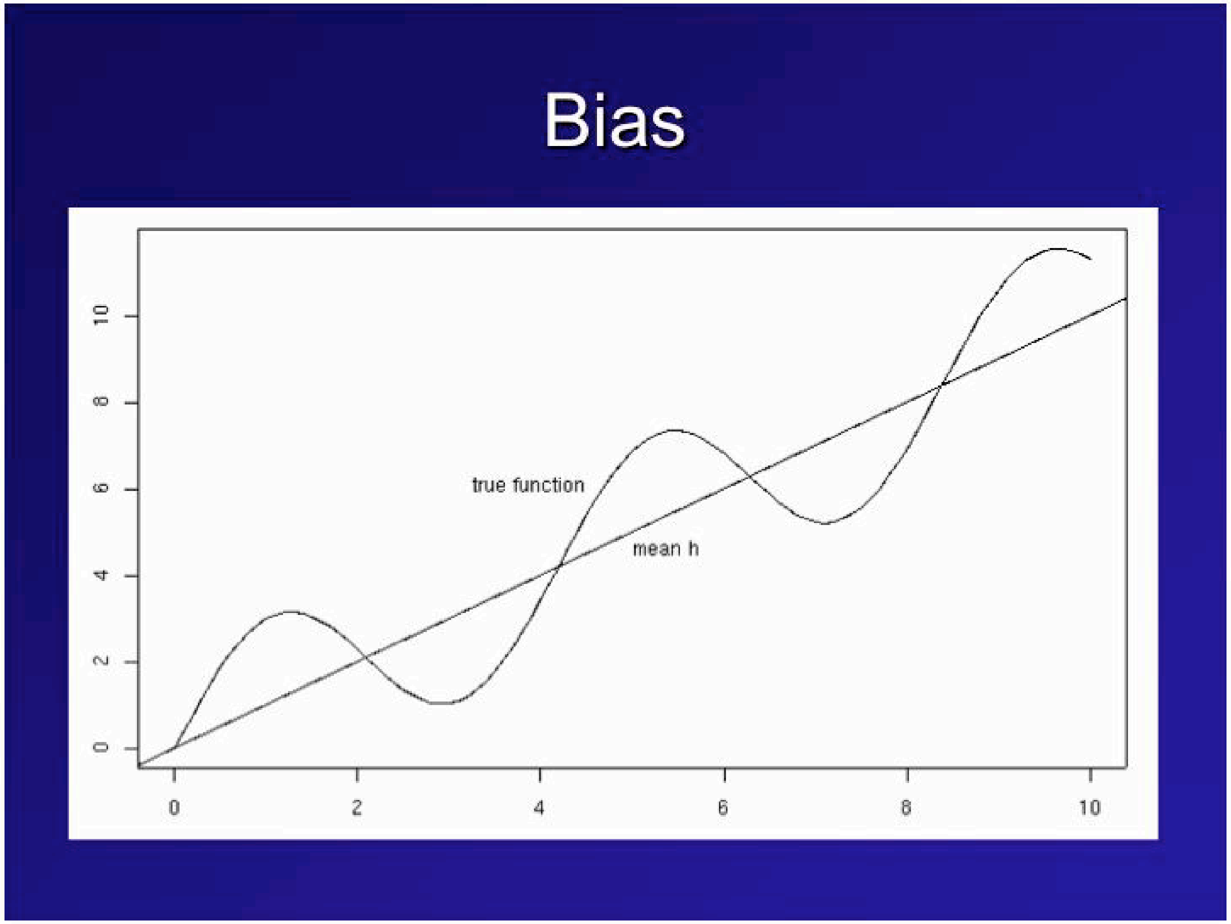
Bias-variance decomposition (II)

- Putting everything together, we have:

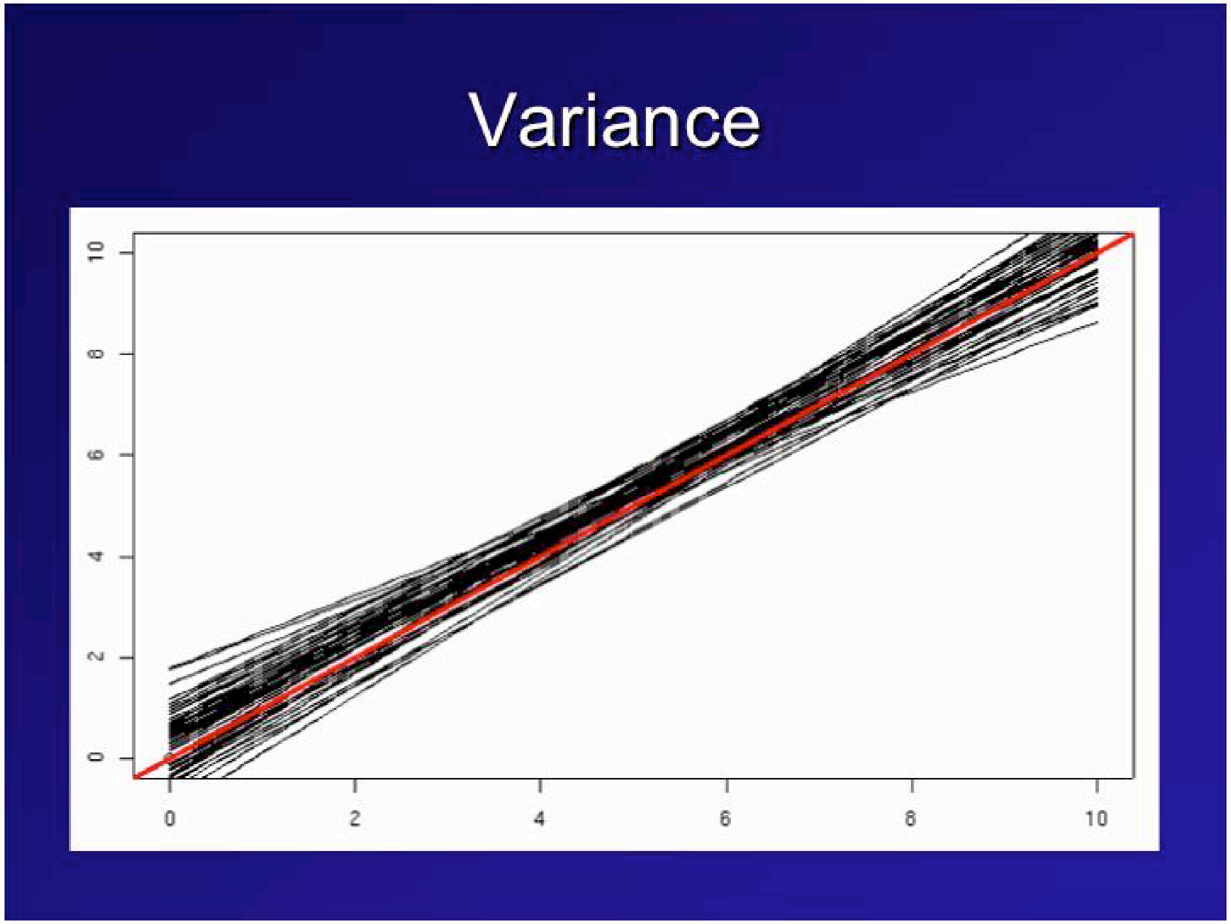
$$\begin{aligned}
E_P\left[(y-h(\mathbf{x}))^2\right] &= E_P[(h(\mathbf{x})-\bar{h}(\mathbf{x}))^2] + (\bar{h}(\mathbf{x}))^2 - 2f(\mathbf{x})\bar{h}(\mathbf{x}) \\
&+ E_P[(y-f(\mathbf{x}))^2] + (f(\mathbf{x}))^2 \\
&= E_P[(h(\mathbf{x})-\bar{h}(\mathbf{x}))^2] + (f(\mathbf{x})-\bar{h}(\mathbf{x}))^2 \\
&+ E[(y-f(\mathbf{x}))^2]
\end{aligned}$$

- The first term is the **variance** of the hypothesis h when trained with finite data sets sampled randomly from P
- The second term is the squared **bias** (or systematic error) which is associated with the class of hypotheses we are considering
- The last term is the **noise**, which is due to the problem at hand, and cannot be avoided

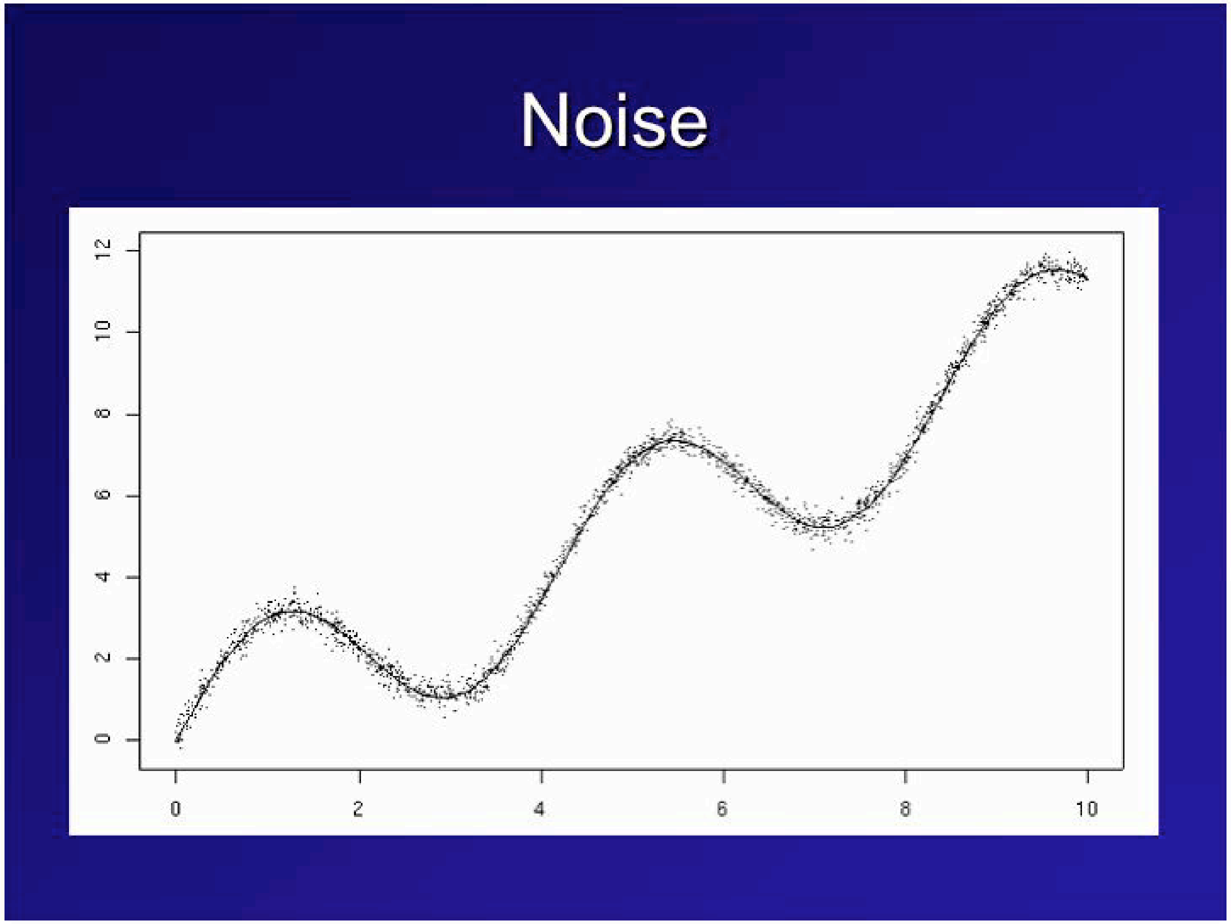
Example revisited: Bias



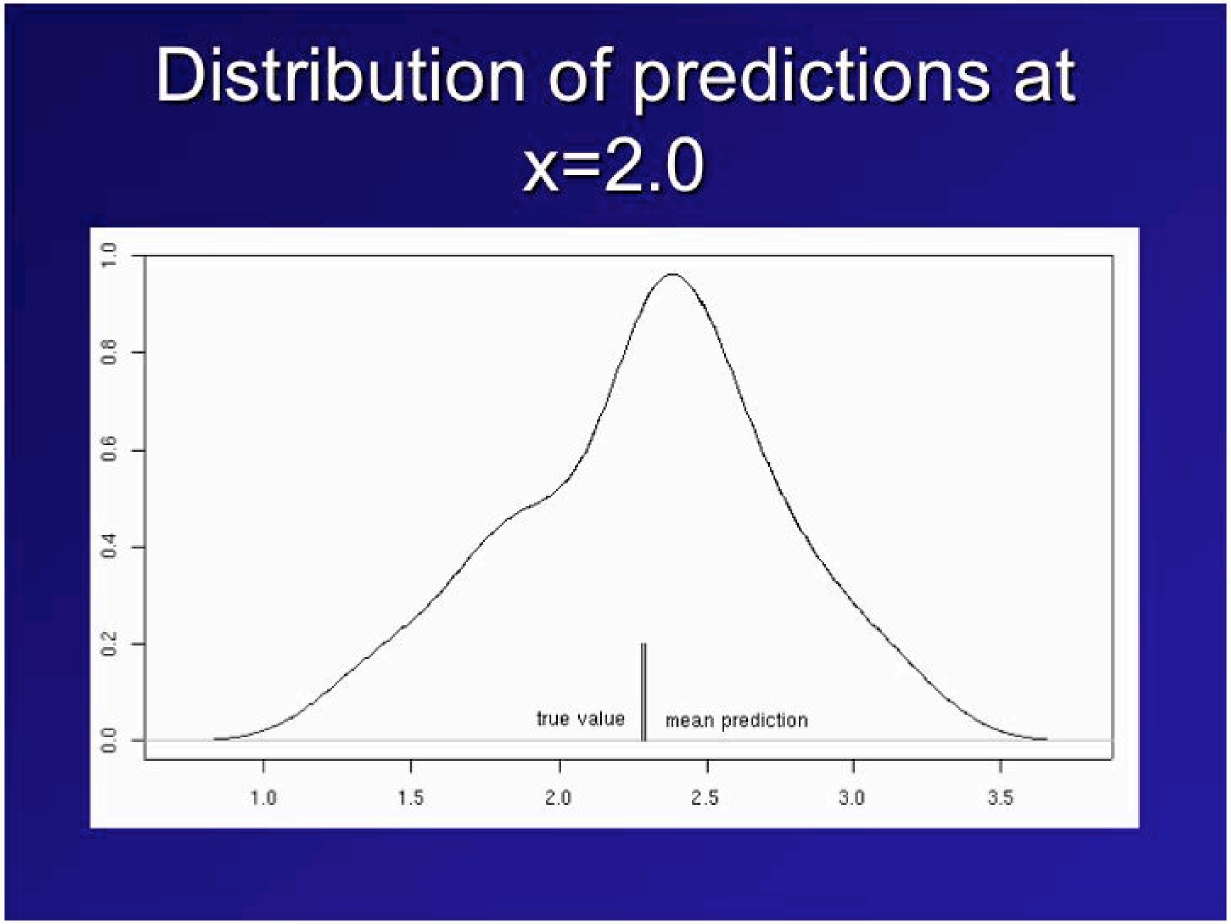
Example revisited: Variance



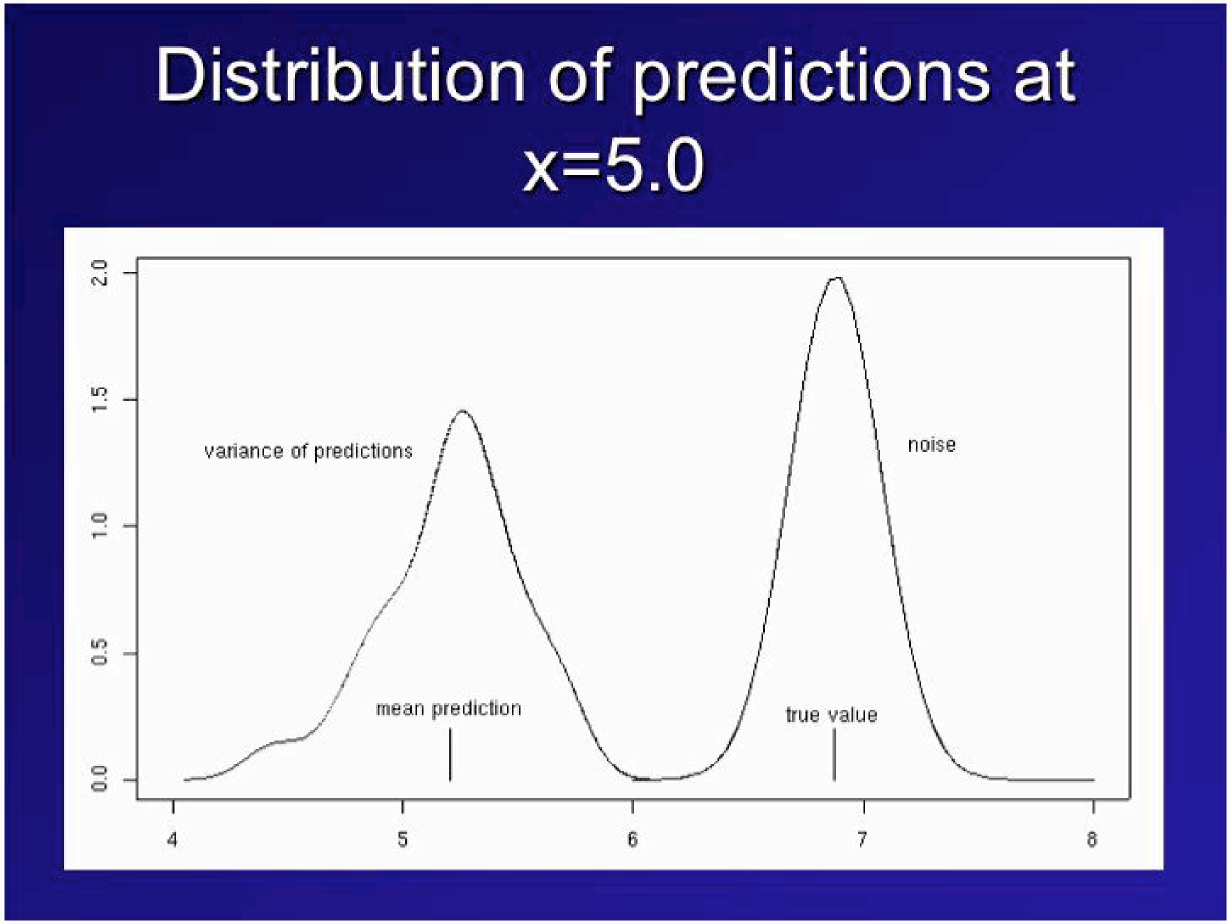
Example revisited: Noise



A point with low bias



A point with high bias



Bias-variance trade-off

- Consider fitting a logistic regression neuron to a data set, vs fitting a large neural net.
- Which one do you expect to have higher bias? Higher variance?

Bias-variance trade-off

- Consider fitting a logistic regression neuron to a data set, vs fitting a large neural net.
- Which one do you expect to have higher bias? Higher variance?
- Typically, *bias* comes from not having good hypotheses in the considered class
- *Variance* results from the hypothesis class containing too many hypotheses
- Hence, we are faced with a *trade-off*: choose a more expressive class of hypotheses, which will generate higher variance, or a less expressive class, which will generate higher bias

Sources of bias

- □ Inability to represent certain decision boundaries
  E.g. linear threshold units, naive Bayes, decision trees
- □ Incorrect assumptions
  E.g. failure of independence assumption in naive Bayes
- □ Classifiers that are "too global" (or, sometimes, too smooth)
  E.g. a single linear separator, a small decision tree

If the bias is high, the model is <u>*underfitting*</u> the data

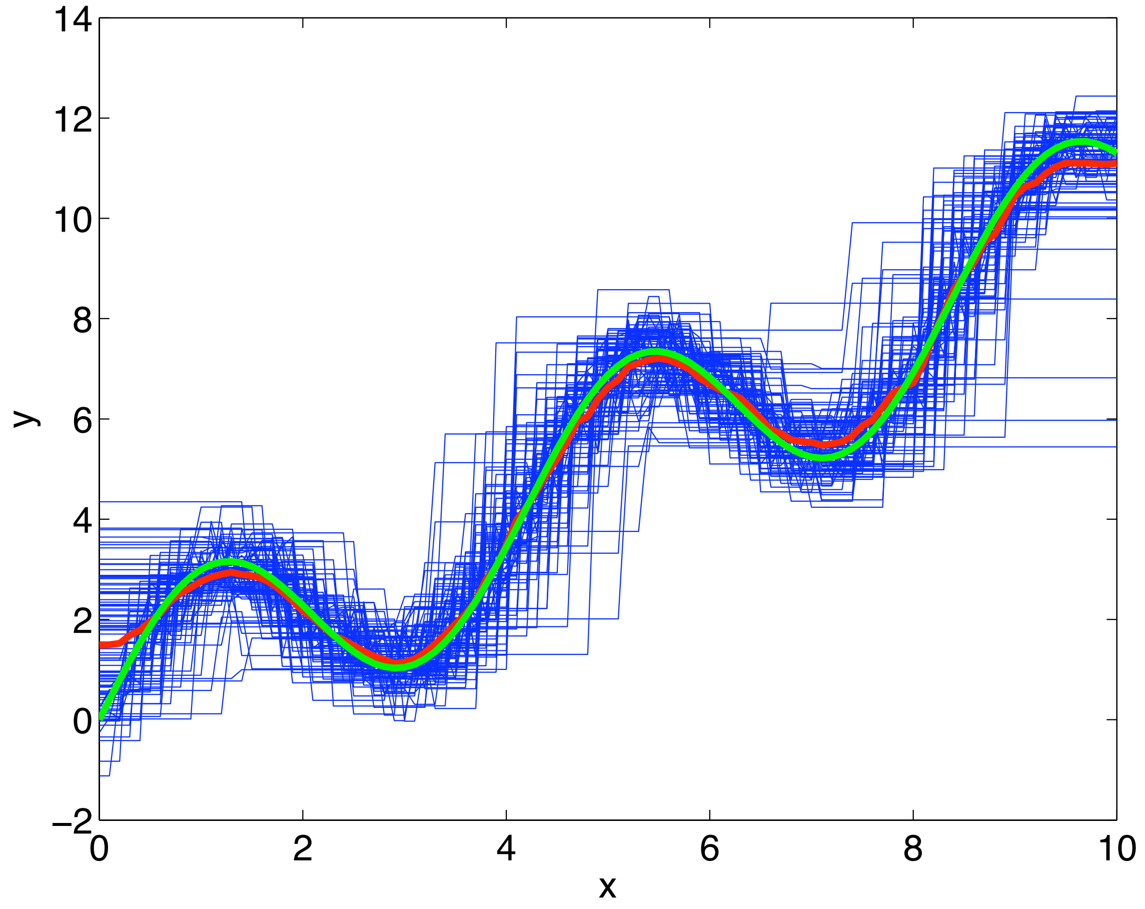
Sources of variance

- Classifiers that are "too local" and can easily fit the data
  E.g. nearest neighbor, large decision trees, RBF
- Making decisions based on small subsets of the data
  E.g. decision tree splits near the leaves
- Randomization in the learning algorithm
  E.g. neural nets with random initial weights
- Learning algorithms that make sharp decisions can be unstable (e.g. the decision boundary can change if one training example changes)

If the variance is high, the model is *overfitting* the data

One nearest neighbor

- 100 data set, each with 20 data points
- Blue is individual predictors, red is mean of predictors, green is true function



Measuring bias and variance

- Bias, variance, noise are all well-defined theoretically
- But we can't compute them directly – because we don't know P, f, or σ
- Can they be estimated somehow?

Measuring bias and variance in practice

- Recall that bias and variance are both defined as expectations:

$$Bias(\mathbf{x}) = E_P[f(\mathbf{x}) - \bar{h}(\mathbf{x})]$$

$$Var(\mathbf{x}) = E_P[(h(\mathbf{x}) - \bar{h}(\mathbf{x}))^2]$$

- We can try to estimate these for a particular $\mathbf{x}$ (though which one?)
- Or, we can try to estimate these averaged over the input space (say, according to the distribution P)
- If we had multiple data sets, we could estimate these by averaging (This is what we did in the earlier example.)
- What if we have only one data set?

Bootstrap replicates

- Given data set D, construct a *bootstrap replicate* of D, called D_b, which has the same number of examples, by drawing samples from D *with replacement*
- Use the learning algorithm to construct a hypothesis h_b by training on D_b
- Compute the prediction of h_b on each of the *remaining* points, from the set $T_b = D - D_b$
- This process is repeated B times, where B is typically a few hundred
- If D is very large, the replicates should contain $m < |D|$ points (still drawn with replacement)

Estimating bias and variance

- For each point, we have a set of estimates $h_1(\mathbf{x}), \ldots h_K(\mathbf{x})$, with $K \leq B$
- The average prediction, determined empirically, is:

$$\bar{h}(\mathbf{x}) = \frac{1}{K}\sum_{k=1}^{K} h_k(\mathbf{x})$$

- We will estimate the bias as:

$$y - \bar{h}(\mathbf{x})$$

  (This conflates bias and noise, really.)

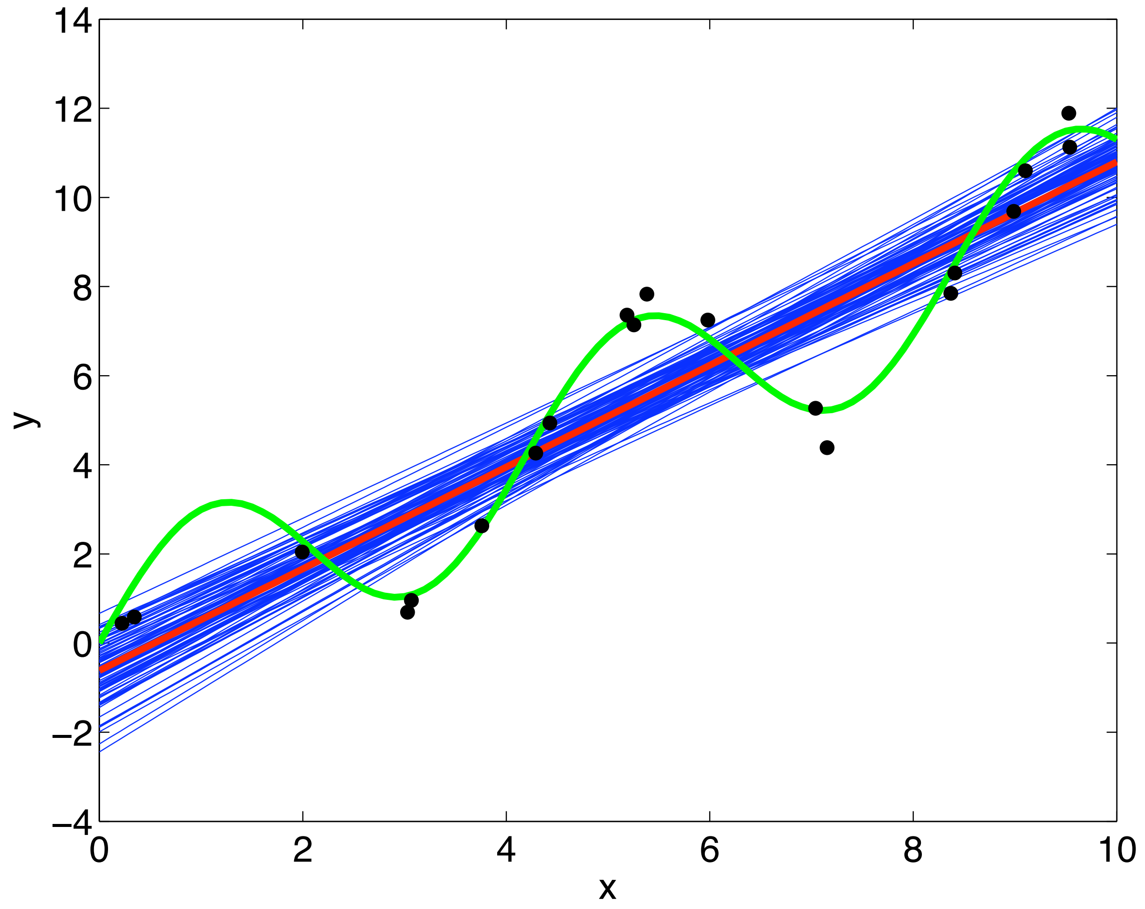
- We estimate the variance as:

$$\frac{1}{K-1}\sum_{k=1}^{K}(\bar{h}(\mathbf{x}) - h_k(\mathbf{x}))^2$$

Approximations

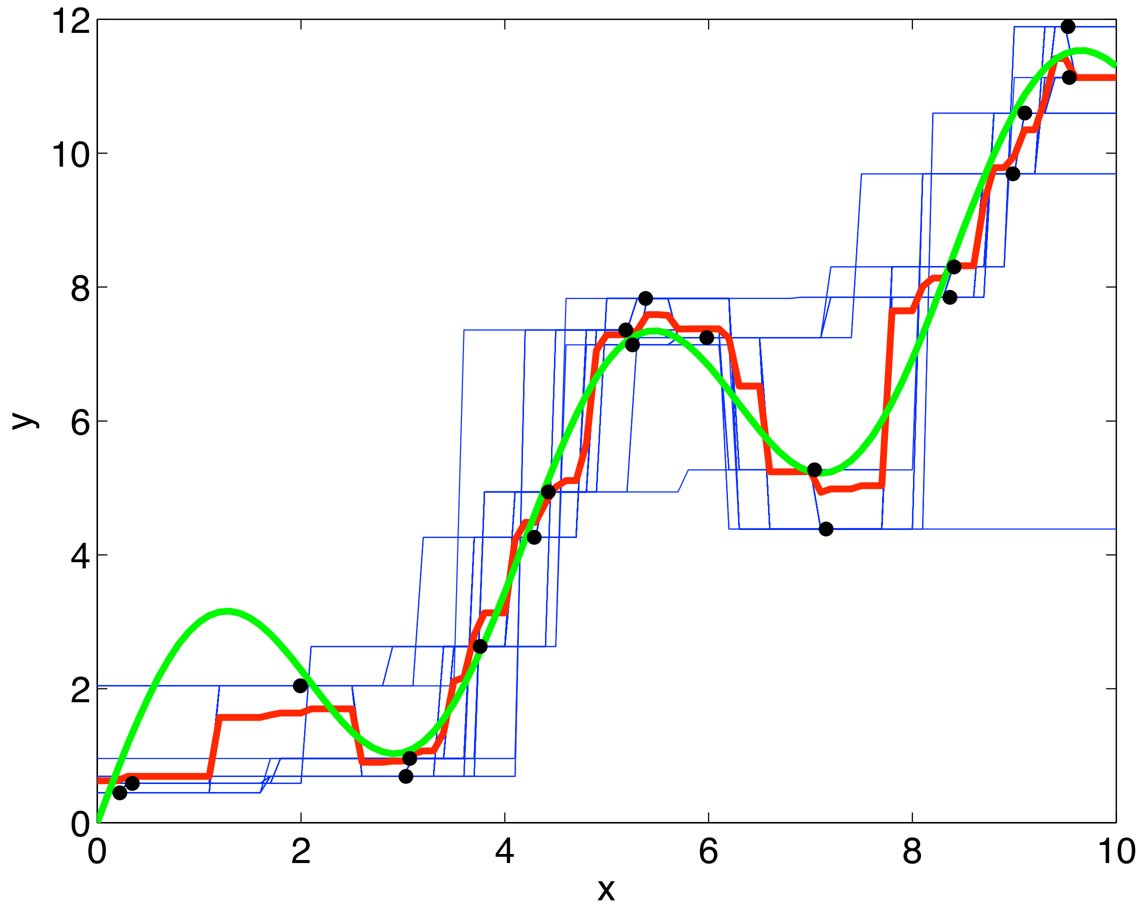
- Bootstrap replicates are not real data
  - We'll never get an $\mathbf{x}$ or y value not in the original data set
  - Resampling the data approximates resampling from P, but is affected by the amount of data / statistical irregularities in data
- We typically ignore the noise, although
  - If we had multiple points with the same $\mathbf{x}$ value, we can estimate the noise
  - Alternatively, we can do an estimation using "similar points"
    - ▷ (E.g., if we have tightly clustered points, and assume f is smooth in that region, we can fit a locally-linear model, and estimate variability by the residuals.)
    - ▷ That assumes Gaussian additive noise, of course

Example: Bootstrapping linear fits



Black = data, Blue = bootstrap fits, Red = mean fit, Green = true f

Example: Bootstrapping one nearest neighbor



Black = data, Blue = bootstrap fits, Red = mean fit, Green = true f

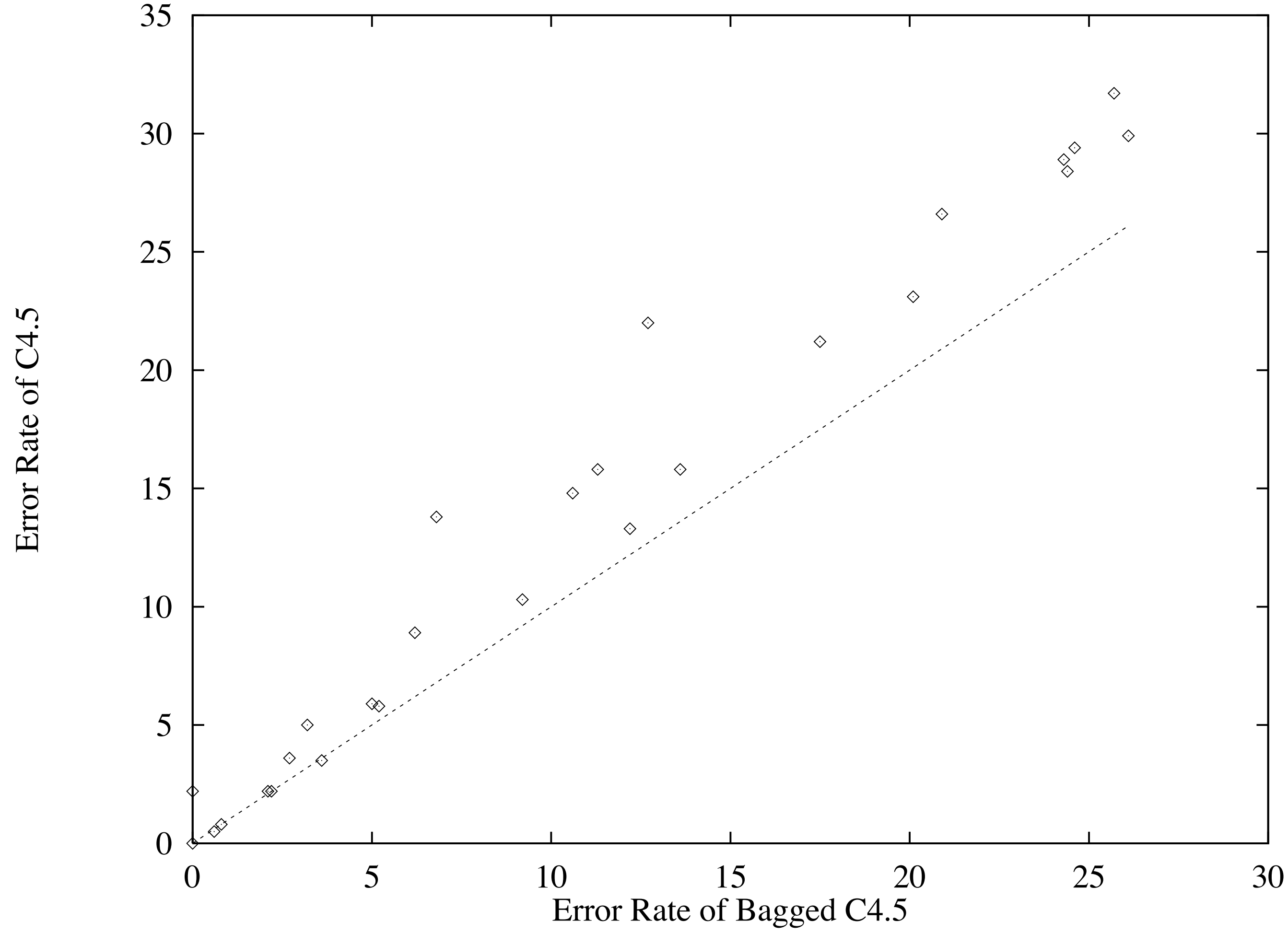
Bagging: Bootstrap aggregation

- If we did all the work to get the hypotheses h_b, why not use all of them to make a prediction?
- All hypotheses can have a vote, in the classification case, and we pick the majority class
- For regression, we can average all the predictions
- Which hypotheses classes would benefit most from this approach?

Estimated bias and variance of bagging

- According with our way of estimating variance and bias, bagging eliminates variance altogether!
- In practice, bagging tends to reduce variance and increase bias
- Hence, the main benefit is for “unstable” learners, i.e., learners with high variance.
- This includes complex hypotheses classes, e.g. decision trees (even unpruned), neural networks, nearest-neighbor-type methods

Experiment: Bagging decision trees (Dietterich)



Bagged C4.5 is as good as or better than building one decision tree, for all 30 data sets.

Ensemble learning in general

- Ensemble learning algorithms work by running a *base learning algorithm* multiple times, then *combining* the predictions of the different hypotheses obtained using some form of voting
- One approach is to construct several classifiers *independently*, then combine their predictions. Examples include:
  - Bagging
  - Using a different subset of input features to train different neural nets
  - Randomizing the learning/fitting
    (E.g., randomizing test selection in decision trees, different random initial parameters for neural nets, stochastic gradient descent)

  $\Rightarrow$ Again, more beneficial with higher variance predictors
- A second approach is to *coordinate* the construction of the hypotheses in the ensemble.

Boosting

- Why construct different members of an ensemble independently?
- Why not construct the members serially, and focus the “attention” of subsequent members on the examples / parts of input space that previous members get wrong?
- AdaBoost (short for Adaptive Boosting), is by far the best-known algorithm for doing this.
  - At each iteration, AdaBoost *reweights* the examples, where the weight is how “important” it is to get that example right
  - AdaBoost has a particular rule for combining the classifiers in the end

Notation

- We consider binary classification problems $\{(\mathbf{x_i}, y_i)\}_{i=1}^{m}$, where $y \in \{-1, +1\}$
- Let a weighting of the examples be a length-m vector that sums to one.
- Given the data set, a hypothesis h, and a weighting P, define the <u>*weighted 0-1 loss*</u> as:

$$\begin{aligned} J_P(h) &= \sum_{i=1}^{m} P(i) \begin{cases} 1 & \text{if } h(\mathbf{x_i}) \neq y_i \\ 0 & \text{otherwise} \end{cases} \\ &= \sum_{i:h(\mathbf{x_i}) \neq y_i} P(i) \end{aligned}$$

(This can be viewed as the expected 0-1 loss, with respect to distribution P.)

Weak learners

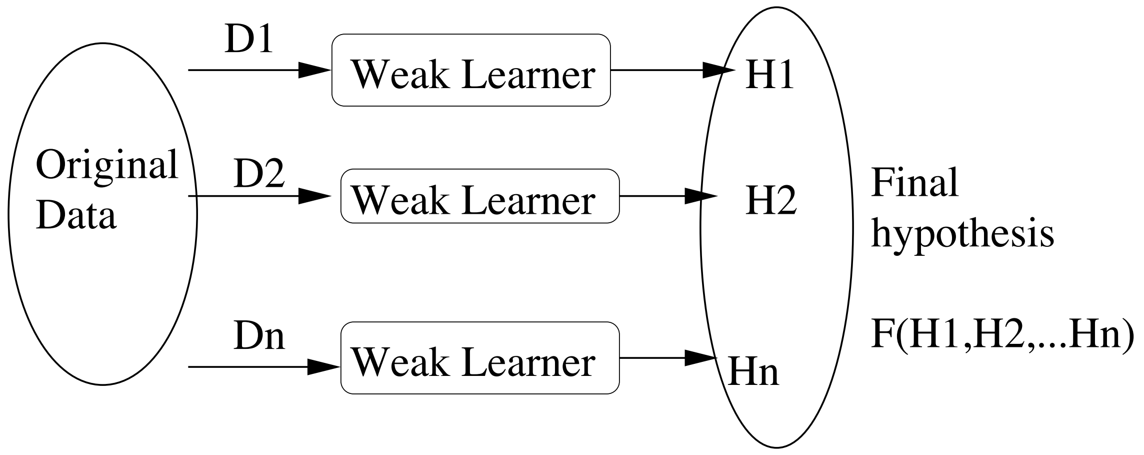
- Adaboost assumes access to a “weak” binary classification algorithm (E.g., decision stumps, maybe logistic regressor)
- The weak learning must be able to accept a weighted binary classification problem $\{(\mathbf{x_i}, y_i)\}_{i=1}^{m}$, P
- It should output a classifier h that satisfies

$$J_P(h) < 1/2 - \gamma \text{ where } \gamma > 0$$

(Note, $J_P(h) \leq 1/2$ is trivially achievable by choosing the hypothesis that always outputs the class whose examples have greater total weight.)

Boosting classifier

- In general, boosting produces a sequence of classifiers
- Each is based on a different weighting of the data set
- The weightings depend on the errors of the previous classifiers
- The classifiers are combined in a certain way at the end



AdaBoost

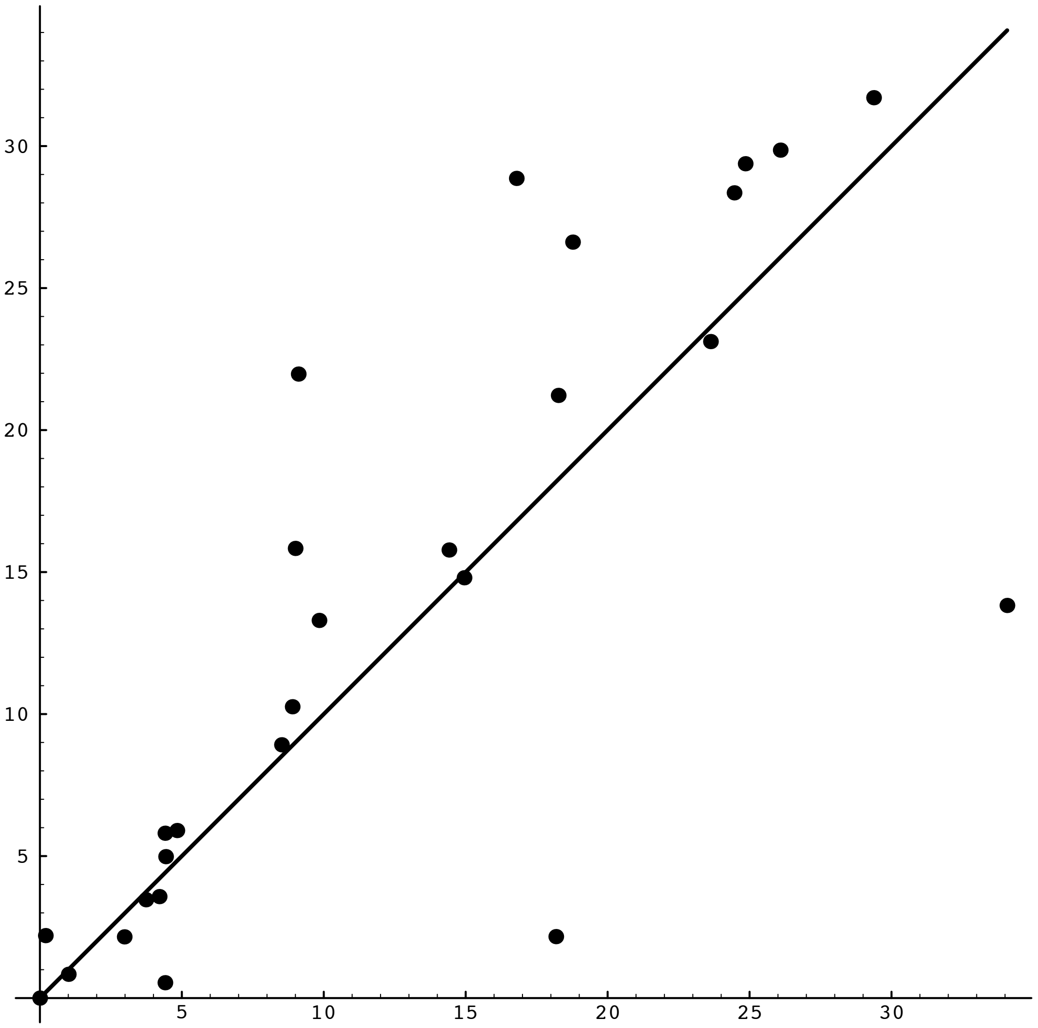
1. For data set $\{(\mathbf{x_i}, y_i)\}_{i=1}^m$, let the initial weighting be uniform: $D_1(i) = 1/m$
2. For $t = 1, 2, 3, \ldots, T$
   - □ Produce h_t by running the weak learner with the weights $D_t(i)$
   - □ If $J_{D_t}(h_t) \geq 1/2$, GOTO step 3!
   - □ Choose a "weight" for the classifier as a whole: $\alpha_t = \frac{1}{2} \log \frac{1-J_{D_t}(h_t)}{J_{D_t}(h_t)}$
   - □ Construct a new weighting for the data:

$$D_{t+1}(i) = \frac{D_t(i) e^{-\alpha_t y_i h_t(\mathbf{x_i})}}{\sum_j D_t(j) e^{-\alpha_t y_j h_t(\mathbf{x_j})}}$$

3. Final classifier is $h_f(\mathbf{x}) = \text{sign}\left(\sum_t \alpha_t h_t(\mathbf{x})\right)$

(Aside: These rules make some intuitive sense, but they are also justified by various learning-theoretic and statistical arguments — but so are other rules.)

Empirical comparison: Boosted stumps vs. C4.5

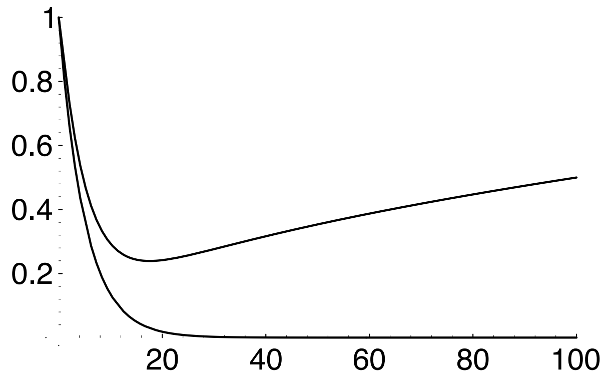


Why does boosting work?

- Weak learners have high bias
- By combining them, we get more expressive classifiers
- Hence, boosting is a *bias-reduction technique*
- What happens as we run boosting longer?
  $\Rightarrow$ Intuitively, we get more and more complex hypotheses
- How would you expect bias and variance to evolve over time?

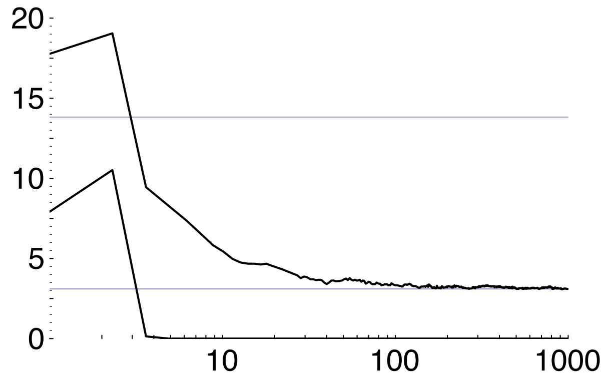
A naive (but reasonable) analysis of generalization error

- Expect the training error to continue to drop (until it reaches 0)
- Expect the test error to *increase* as we get more voters, and h_f becomes too complex.



Actual typical run of AdaBoost

Boosting C4.5 on the letter dataset:



- Test error *does not increase* even after 1000 runs! (more than 2 million decision nodes!)
- Test error *continues to drop* even after training error reaches 0!

These are consistent results through many sets of experiments!

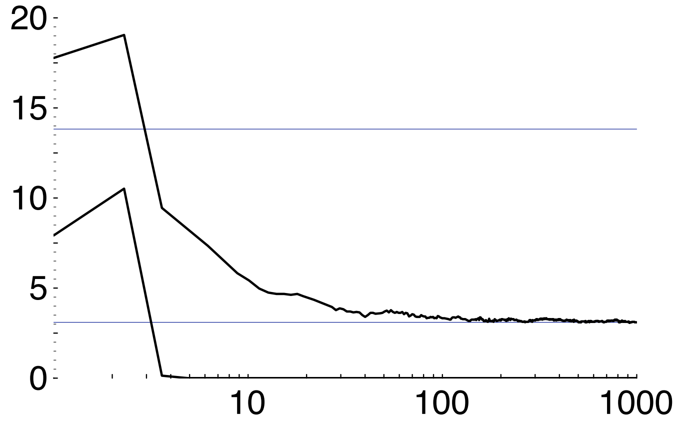
Classification margin

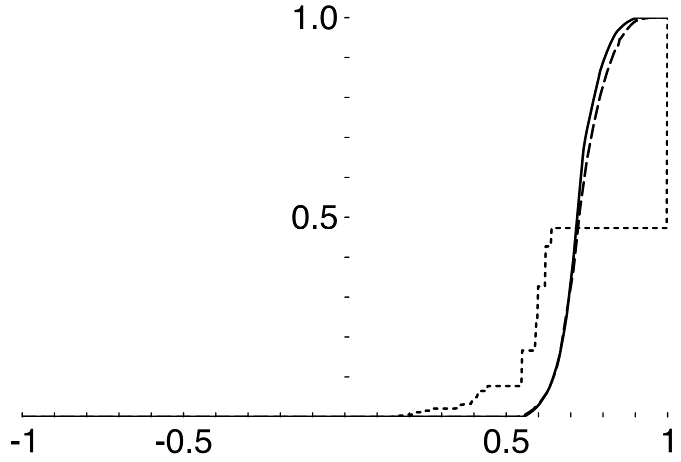
- Boosting constructs hypotheses of the form $h_f(\mathbf{x}) = \text{sign}(f(\mathbf{x}))$
- The classification of an example is correct if $\text{sign}(f(\mathbf{x})) = y$
- Consider the margin-related notion:

$$\text{margin}(f, \mathbf{x_i}, y_i) = y_i \cdot f(\mathbf{x_i})$$

- This margin tells us how close the decision boundary is to the data points on each side.
- A higher margin on the training set should yield a lower generalization error
- Intuitively, increasing the margin is similar to lowering the variance

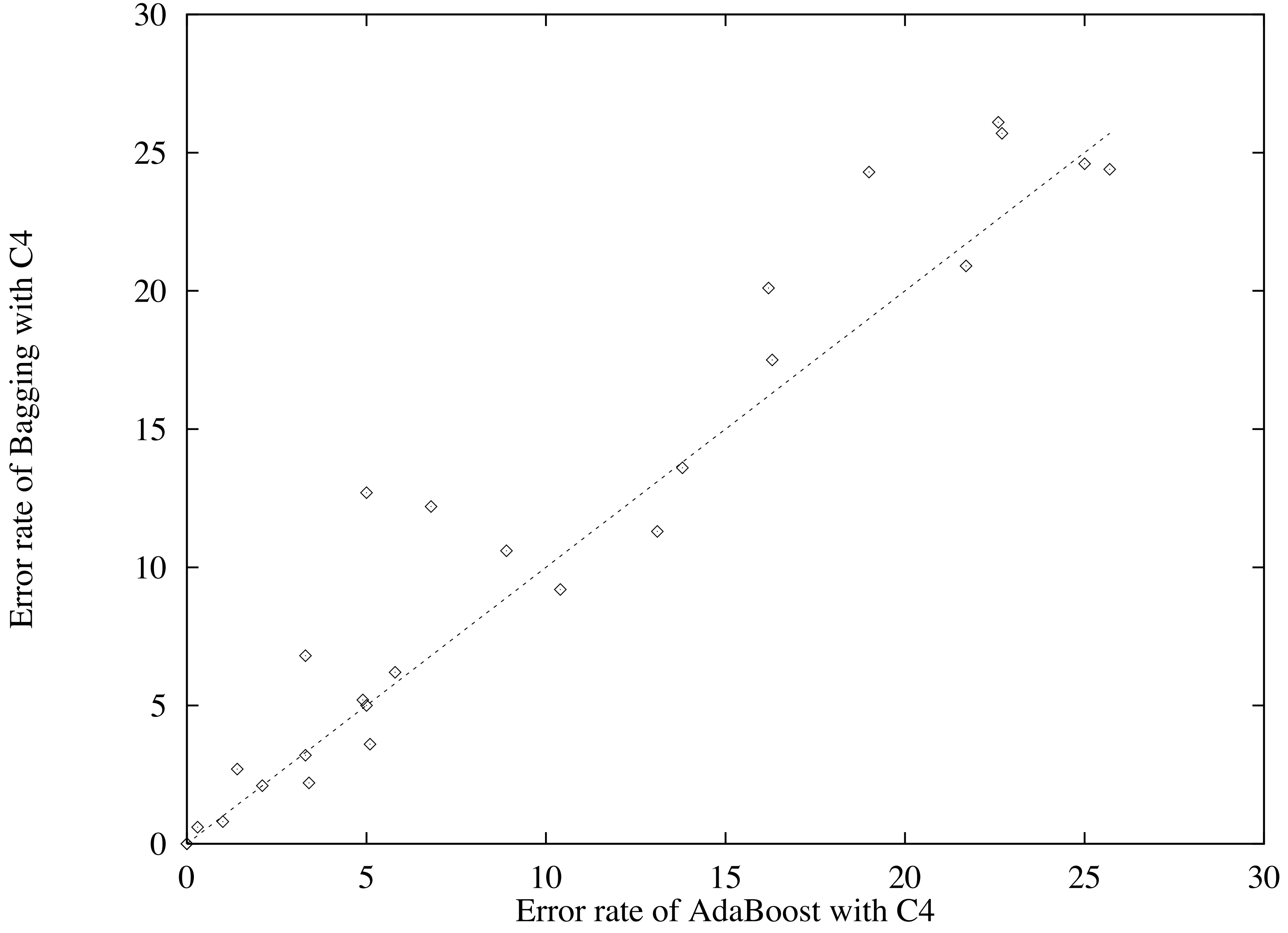
Effect of boosting on the margin





- □ Between rounds 5 and 10 there is no training error reduction
- □ But there is a *significant shift* in margin distribution!
- □ There is a formal proof that boosting increases the margin

Bagging vs. Boosting



Summary and final comments

- We showed generalization error (for regression) can be decomposed into bias, variance, and noise components
- Bias and variance can be estimated by bootstrapping (with some caveats)
- Ensemble methods combine several hypotheses into one prediction
- They work better than the best individual hypothesis from the same class because they reduce bias or variance (or both)
- Bagging is mainly a variance-reduction technique, useful for complex hypotheses
- Boosting is mainly a bias-reduction technique, which utilizes weak learners focussed on harder examples, and gives a weighted vote to the hypotheses.
- Boosting can be thrown off by mislabeled data
- Neither technique weights ensemble elements differently in different parts of input space – which some more elaborate techniques do