

# Machine Learning (CS 567)

**Fall 2008**

**Time: T-Th 5:00pm - 6:20pm**

**Location: GFS 118**

**Instructor: Sofus A. Macskassy ([macskass@usc.edu](mailto:macskass@usc.edu))**

**Office: SAL 216**

**Office hours: by appointment**

**Teaching assistant: Cheol Han ([cheolhan@usc.edu](mailto:cheolhan@usc.edu))**

**Office: SAL 229**

**Office hours: M 2-3pm, W 11-12**

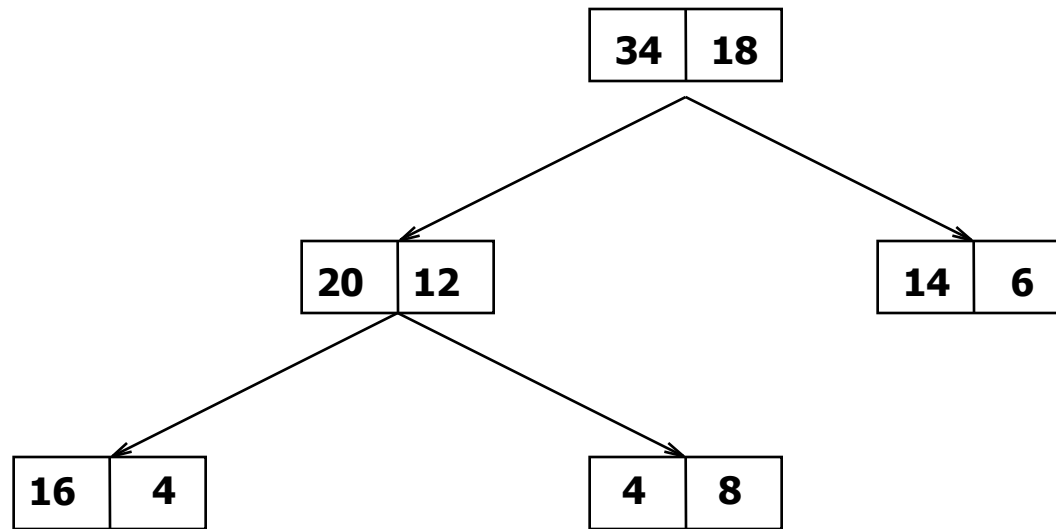
**Class web page:**

<http://www-scf.usc.edu/~csci567/index.html>

# Penalty Methods for decision trees, neural networks, and SVMs

- Decision Trees
  - pessimistic pruning
  - MDL pruning
- Neural Networks
  - weight decay
  - weight elimination
  - pruning methods
- Support Vector Machines
  - maximizing the margin

# Pessimistic Pruning of Decision Trees



# Pessimistic Pruning of Decision Trees

- Error rate on training data is  $4/20 = 0.20 = p$ .
- Binomial confidence interval (using the normal approximation to the binomial distribution) is



16	4
----	---

$$p - z_{\alpha/2} \cdot \sqrt{\frac{p(1-p)}{n}} \leq p \leq p + z_{\alpha/2} \cdot \sqrt{\frac{p(1-p)}{n}}$$

- If we use  $\alpha = 0.25$ , then  $z_{\alpha/2} = 1.150$  so we obtain

$$0.097141 \leq p \leq 0.302859$$

- We use the upper bound of this as our error rate estimate. Hence, we estimate  $0.302859 \times 20 = 6.06$  errors

# Pruning Algorithm (1): Traversing the Tree

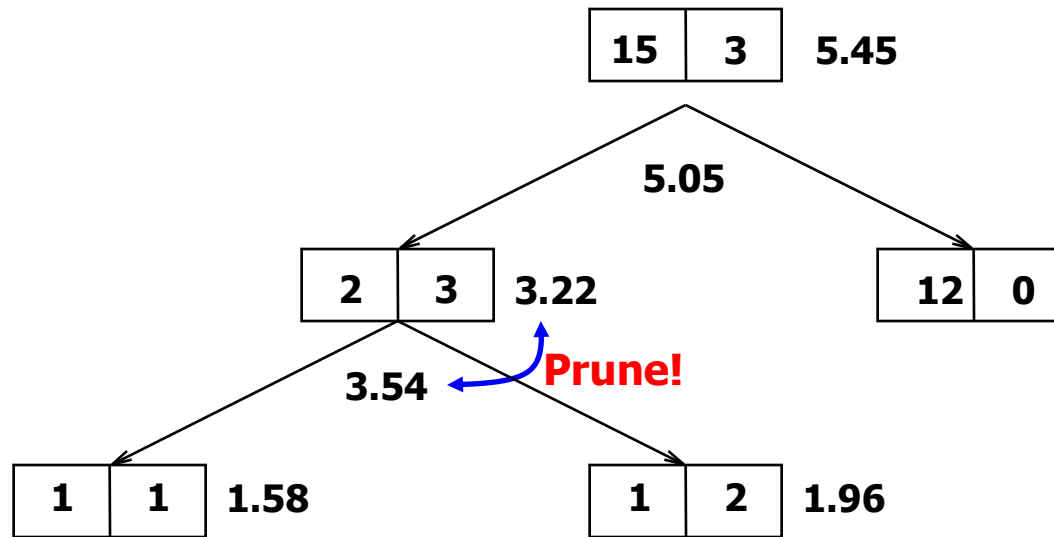
```
float Prune(Node node)
{
    float prunedError = PessimisticError(node);
    if (node.leaf) return prunedError;
    float childError = Prune(node.left) + Prune(node.right);
    if (prunedError < childError)
    {
        // prune
        node.leaf = true;
        node.left = node.right = NULL;
        return prunedError
    }
    else // don't prune
        return childError;
}
```

# Pruning Algorithm (2): Computing the Pessimistic Error

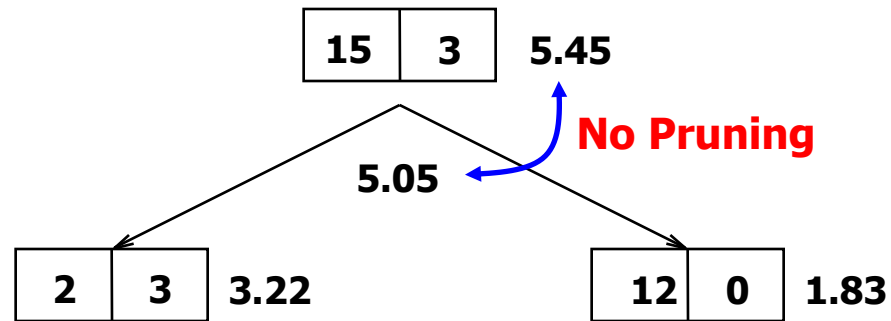
```
const float zalpha2 = 1.150; // p = 0.25 two-sided
```

```
float PessimisticError(Node node)
{
    float n = node.class[0] + node.class[1];
    float wrong = min(node.class[0], node.class[1]);
    // Laplace estimate of error rate
    float p = (wrong + 1.0) / (n+2.0);
    return n * (p+zalpha2 * sqrt(p*(1.0-p) / (n+2.0)));
}
```

# Pessimistic Pruning Example



# Pessimistic Pruning Example



# MDL Pruning

$$h_{\text{MDL}} = \operatorname{argmin} L(\text{tree}) + L(\text{examples misclassified})$$

- The MDL principle trades off tree size for training errors.
- If a tree is small, it can make some training errors, and the message produced would still be more compact.

# Penalty methods for Neural Networks

- Weight Decay

$$J_i(W) = \frac{1}{2}(\hat{y}_i - y_i)^2 + \lambda \sum_j w_j^2$$

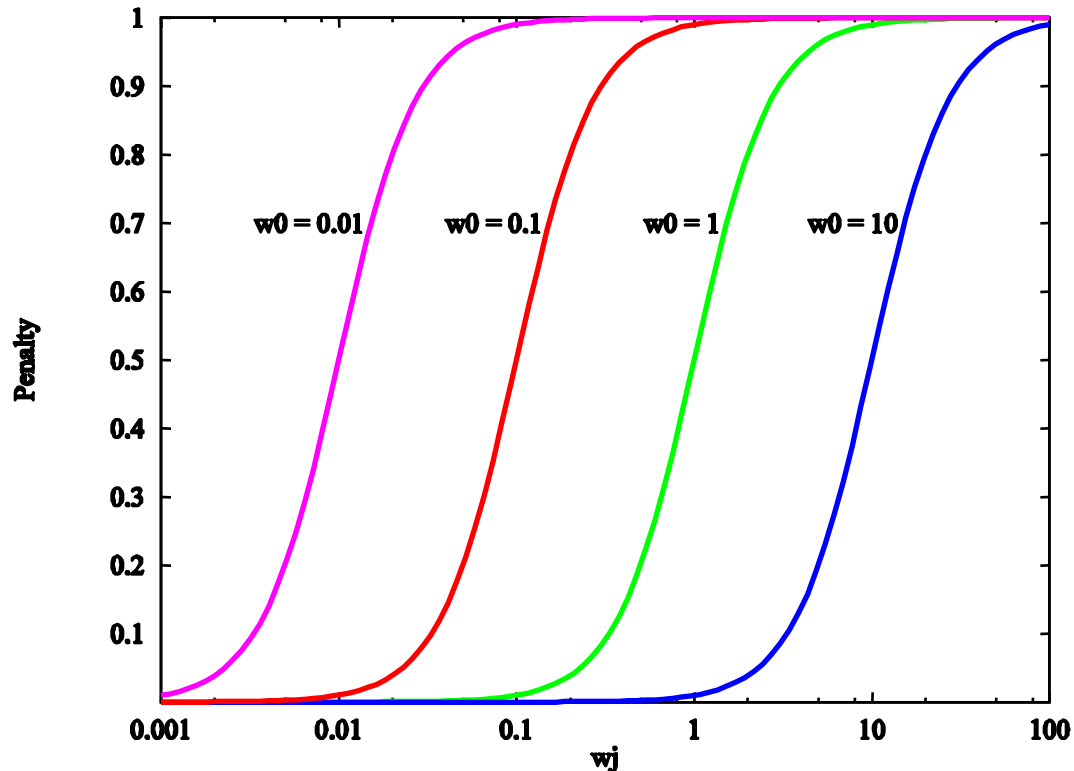
- Weight Elimination

$$J_i(W) = \frac{1}{2}(\hat{y}_i - y_i)^2 + \lambda \sum_j \frac{w_j^2/w_0^2}{1 + w_j^2/w_0^2}$$

$w_0$  large encourages many small weights

$w_0$  small encourages a few large weights

# Weight Elimination



- This essentially counts the number of large weights. Once they are large enough, their penalty does not change

# Neural Network Pruning Methods: Optimal Brain Damage

(LeCun, Denker, Solla, 1990)

1. Choose a reasonable network architecture
2. Train the network until a reasonable solution is obtained
3. Compute the second derivatives  $h_{jj}$  for each weight  $w_j$
4. Compute the saliencies for each weight:  $h_{jj}w_j^2/2$
5. Sort the weights by saliency and delete some low-saliency weights
6. Repeat from step 2

# OBD Results

- On an OCR problem, they started with a highly-constrained and sparsely connected network with 2,578 weights, trained on 9300 training examples. They were able to delete more than 1000 weights without hurting training or test error.
- Optimal Brain Surgeon attempts to do this before reaching a local minimum
- Experimental evidence is mixed about whether this reduced overfitting, but it does reduce the computational cost of using the neural network

# Penalty Methods for Support Vector Machines

- Recall the basic linear SVM tries to fit the training data perfectly (possibly by using kernels). However, this will quickly lead to overfitting.
- Recall the margin-based bound. With probability  $1 - \delta$ , a linear separator with unit weight vector and margin  $\gamma$  on training data lying in a ball of radius  $R$  will have an error rate on new data points bounded by

$$\epsilon \leq \frac{C}{m} \left[ \frac{R^2 + \|\xi\|^2}{\gamma^2} \log^2 m + \log \frac{1}{\delta} \right]$$

For some constant  $C$ .  $\xi$  is the margin slack vector such that

$$\xi_i = \max\{0, \gamma - y_i g(\mathbf{x}_i)\}$$

# Preventing SVM Overfitting

(This is a review of what we did for SVM)

- Maximize margin  $\gamma$
- Minimize slack vector  $||\xi||$
- Minimize  $R$
  
- The (reciprocal) of the margin acts as a penalty to prevent overfitting

# The Geometric Margin is the Inverse of $\|\mathbf{w}\|$

- Lemma:  $\gamma_g = 1/\|\mathbf{w}\|$

- Proof:

- Let  $\mathbf{w}$  be an arbitrary weight vector such that the positive point  $\mathbf{x}^+$  has a functional margin of 1. Then

$$\mathbf{w} \cdot \mathbf{x}^+ + b = 1$$

- Now normalize this equation by dividing by  $\|\mathbf{w}\|$ .

$$\frac{\mathbf{w}}{\|\mathbf{w}\|} \cdot \mathbf{x}^+ + \frac{b}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|} = \gamma_g$$

- Implication: We can hold the functional margin at 1 and minimize the norm of the weight vector

# Handling Non-Separable Data: Introduce Margin Slack Variables

- Find:  $\mathbf{w}, \xi$
- Minimize:  $||\mathbf{w}'||^2 + C||\xi||^2$
- Subject to:
  - $y_i \cdot (\mathbf{w} \cdot \mathbf{x}_i + b) + \xi_i \geq 1$
  - $\xi_i$  is positive only if example  $\mathbf{x}_i$  does not have a functional margin of at least 1
  - $||\xi||^2$  measures how well the SVM fits the training data
  - $||\mathbf{w}'||^2$  is the penalty term
  - $C$  is the tradeoff parameter that determines the relative weight of the penalty compared to the fit to the data
- Remember we reformulated this in the lagrangian dual form to enable the use of kernels

# Setting the Value of $C$

- The full SVM algorithm requires choosing the value of  $C$ , which controls the tradeoff between fitting the data and obtaining a large margin.
- To set  $C$ , we could train an SVM with different values of  $C$  and plug the resulting  $C$ ,  $\gamma$ , and  $\xi$  into the margin bounds theorem to choose the  $C$  that minimizes the bound on  $\varepsilon$ .
- In practice, this does not work well, and we must rely on holdout methods.

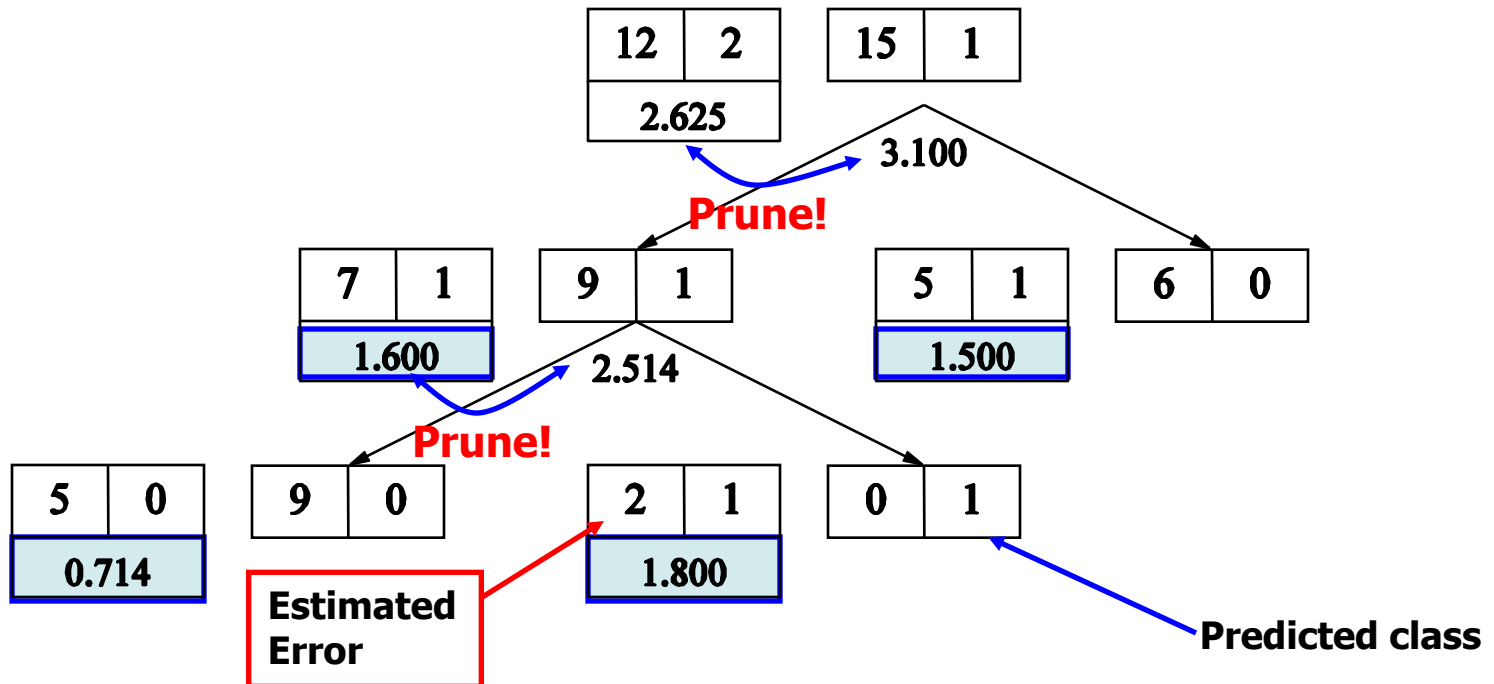
# Holdout and Cross-Validation Methods for Overfitting Avoidance

- Decision Trees
  - Reduce error pruning
  - Cost-complexity pruning
- Neural Networks
  - Early stopping
  - Adjusting Regularizers via Cross-Validation
- Nearest Neighbor
  - Choose number of neighbors
- Support Vector Machines
  - Choose  $C$
  - Choose  $\sigma$  for Gaussian Kernels

# Reduce Error Pruning

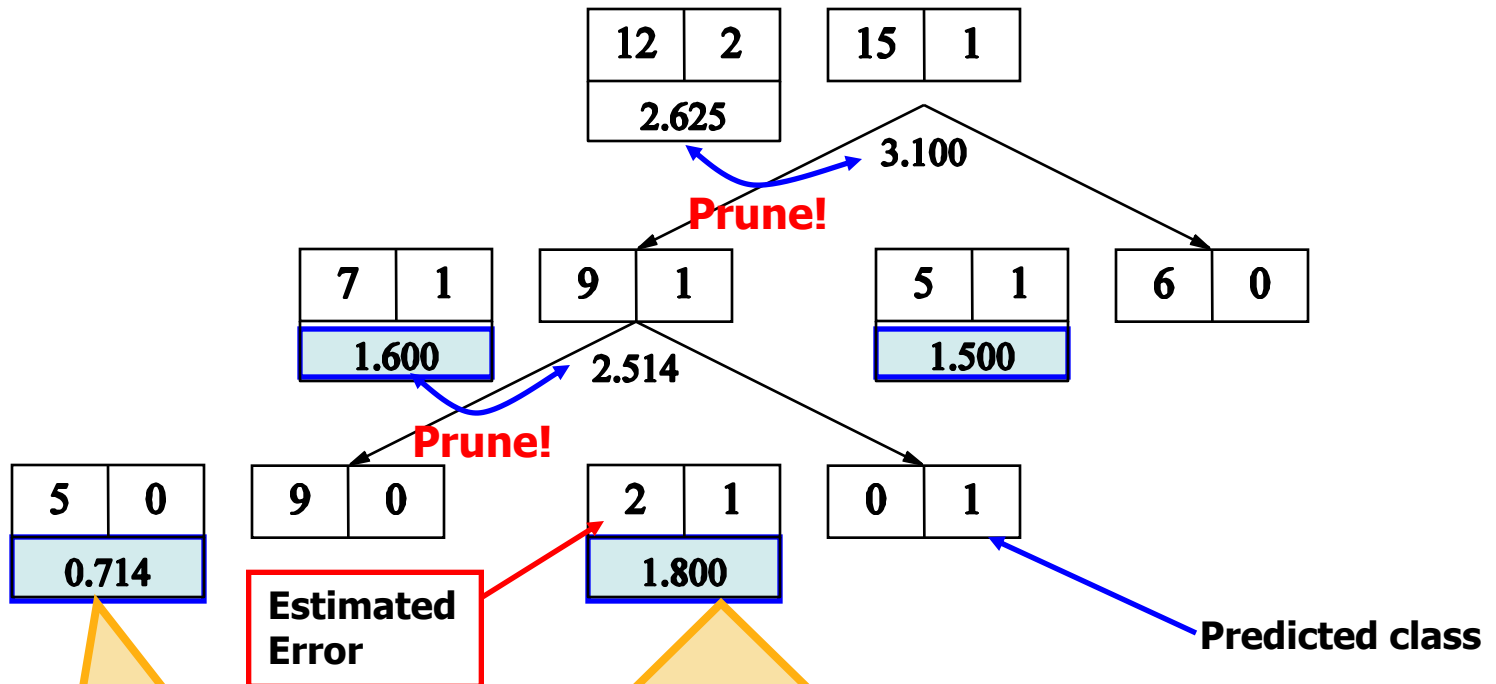
- Given a data sets  $S$ 
  - Subdivide  $S$  into  $S_{\text{train}}$  and  $S_{\text{dev}}$  (development set)
  - Build tree using  $S_{\text{train}}$
  - Pass all of the  $S_{\text{dev}}$  training examples through the tree and estimate the error rate of each node using  $S_{\text{dev}}$
  - Convert a node to a leaf if it would have lower estimated error than the sum of the errors of its children

# Reduce Error Pruning Example



- Boxes on left show development set examples and number of errors (Laplace-corrected)
- Tree will be pruned all the way to the root!

# Reduce Error Pruning Example



Remember we are doing Laplace correction:  
 $\epsilon = 5 \cdot (0+1) / (5+2)$   
 $= 0.714$

Tricky!  
 The leaf will predict the class on the right. Therefore the class on the left is wrong. The Laplace correction will therefore be:  
 $\epsilon = 3 \cdot (2+1) / (3+2)$   
 $= 1.8$

# Cost-Complexity Pruning

- The CART system (Breiman et al, 1984), employs cost-complexity pruning:

$$J(\text{Tree}, S) = \text{ErrorRate}(\text{Tree}, S) + \alpha |\text{Tree}|$$

where  $|\text{Tree}|$  is the number of nodes in the tree and  $\alpha$  is a parameter that controls the tradeoff between the error rate and the penalty

- $\alpha$  is set by cross-validation

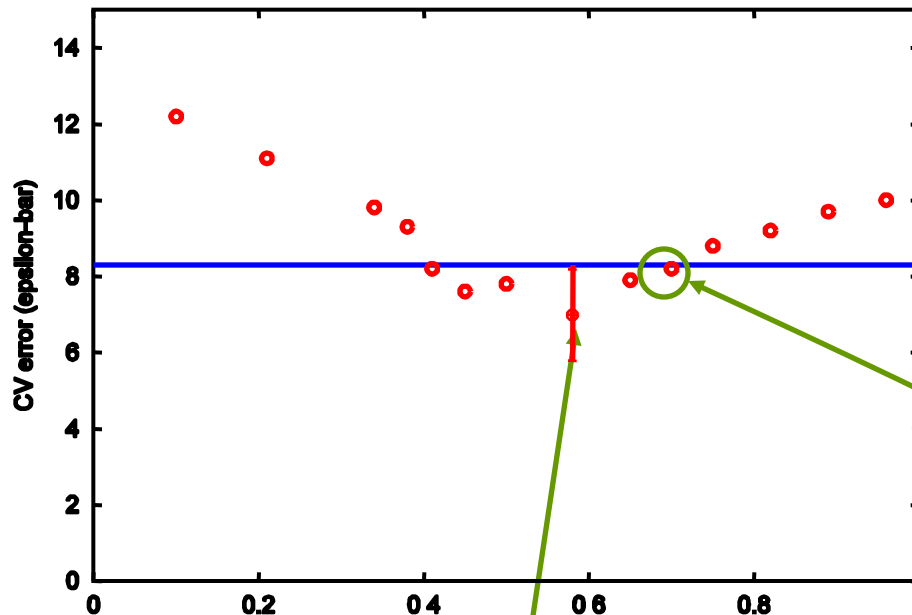
# Determining Important Values of $\alpha$

- Goal: Identify a finite set of candidate values for  $\alpha$ . Then evaluate them via cross-validation
- Set  $\alpha = \alpha_0 = 0; t = 0$
- Train  $S$  to produce tree  $T$
- Repeat until  $T$  is completely pruned
  - determine next larger value of  $\alpha = \alpha_{k+1}$  that would cause a node to be pruned from  $T$
  - prune this node
  - $t := t + 1$
- This can be done efficiently

# Choosing an $\alpha$ by Cross-Validation

- Divide  $S$  into 10 subsets  $S_0, \dots, S_9$
- In fold  $v$ 
  - Train a tree on  $U_{i \neq v} S_i$
  - For each  $\alpha_k$ , prune the tree to that level and measure the error rate on  $S_v$
  - Compute  $\underline{\varepsilon}_k$  to be the average error rate over the 10 folds when  $\alpha = \alpha_k$
  - Choose the  $\alpha_k$  that minimizes  $\underline{\varepsilon}_k$ . Call it  $\alpha_*$  and let  $\varepsilon_*$  be the corresponding error rate
- Prune the original tree according to  $\alpha_*$

# The 1-SE Rule for Setting $\alpha$



- Compute a confidence interval on  $\underline{\varepsilon}_*$  and let  $U$  be the upper bound of this interval.
- Compute the largest  $\alpha_k$  whose  $\varepsilon_k \leq U$ . If we use  $Z=1$  for the confidence interval computation, this is called the 1-SE rule, because the bound is one "standard error" above  $\underline{\varepsilon}_*$ . In this case, we would choose  $\alpha = 0.7$ .

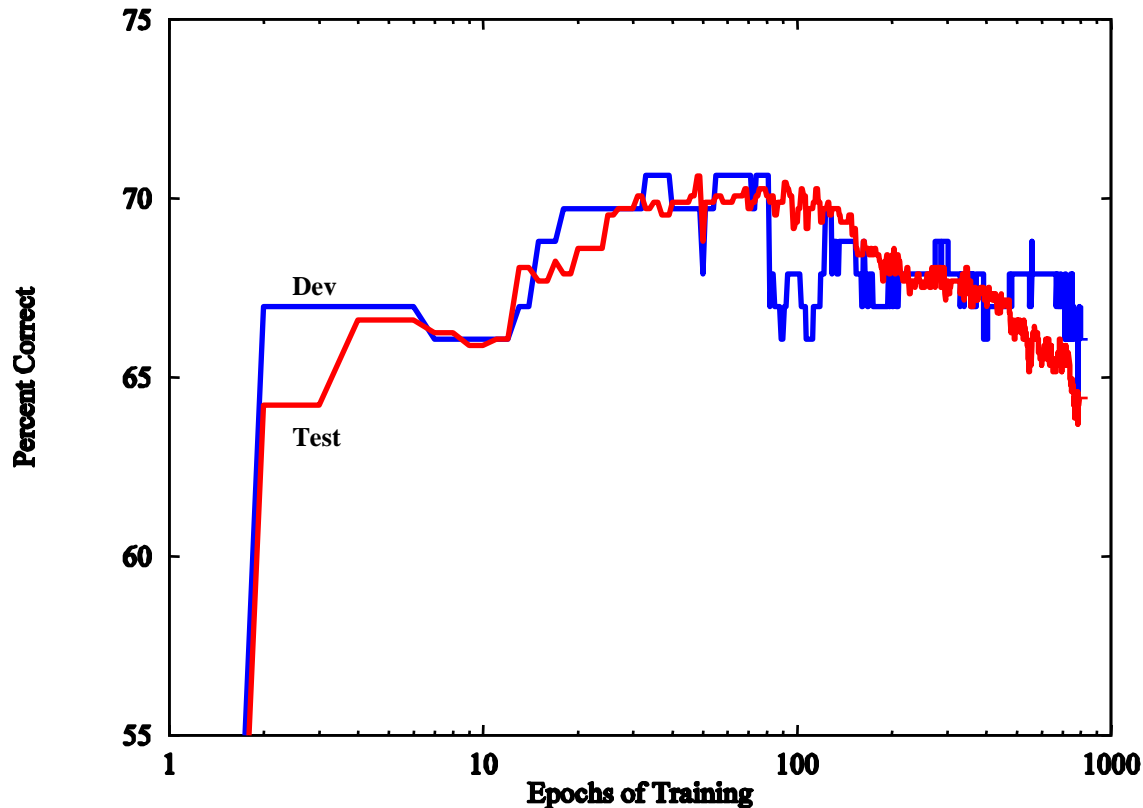
# Notes on Decision Tree Pruning

- Cost-complexity pruning usually gives best results in experimental studies
- Pessimistic pruning is the most efficient (does not require holdout or cross-validation) and it is quite robust
- Reduce-error pruning is rarely used, because it consumes training data
- Pruning is more important for regression trees than for classification trees
- Pruning has relatively little effect for classification trees. There are only a small number of possible prunings of a tree, and usually the serious errors made by the tree-growing process (i.e., splitting on the wrong features) cannot be repaired by pruning.
  - Ensemble methods work much better than pruning

# Holdout Methods for Neural Networks

- Early Stopping using a development set
- Adjusting Regularizers using a development set or via cross-validation
  - amount of weight decay
  - number of hidden units
  - learning rate
  - number of epochs

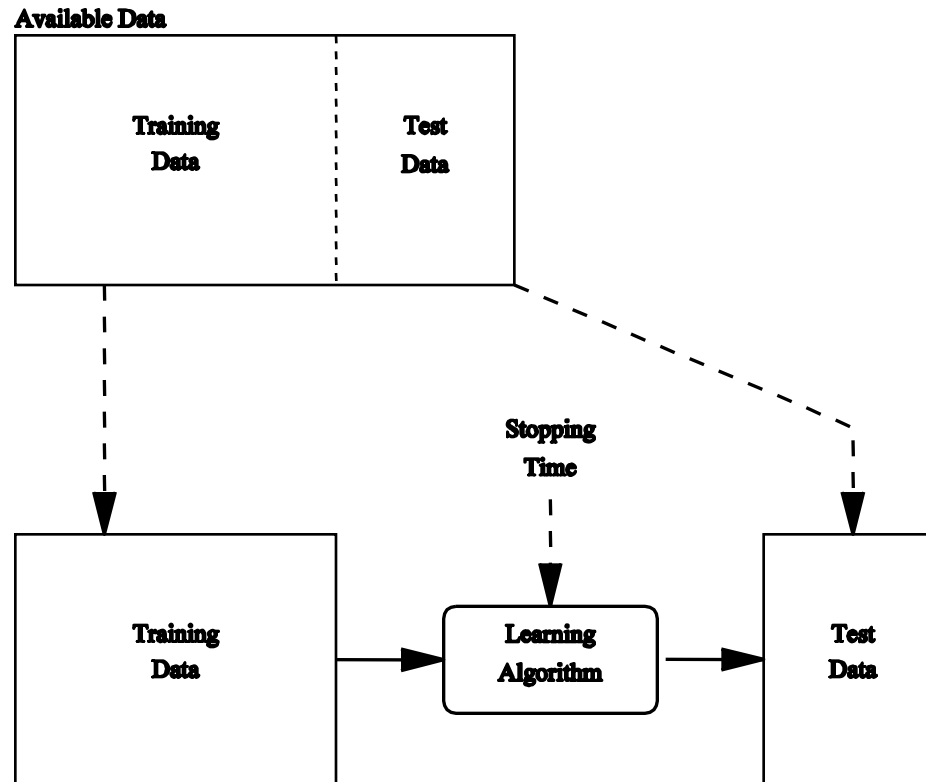
# Early Stopping using an Evaluation Set



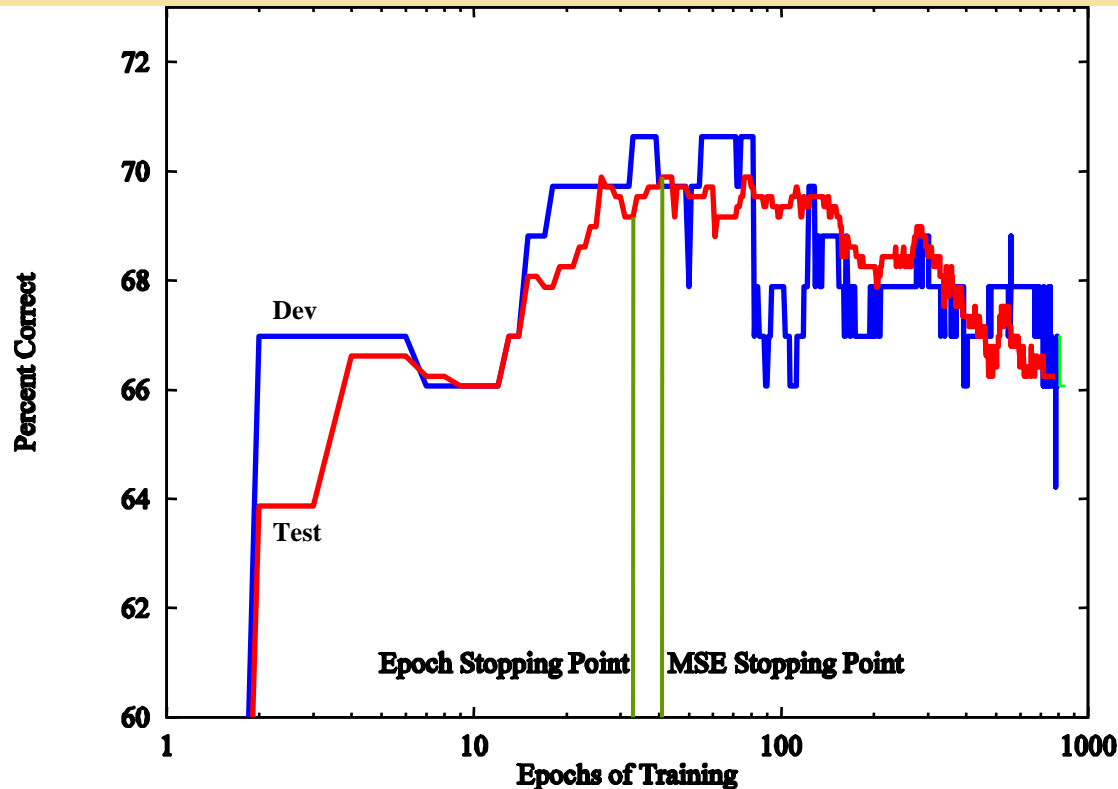
- Split  $S$  into  $S_{\text{train}}$  and  $S_{\text{dev}}$
- Train on  $S_{\text{train}}$ , after every epoch, evaluate on  $S_{\text{dev}}$ . If error rate is best observed, save the weights

# Reconstituted Early Stopping

- Recombine  $S_{\text{train}}$  and  $S_{\text{dev}}$  to produce  $S$
- Train on  $S$  and stop at the point (# of epochs or mean squared error) identified using  $S_{\text{dev}}$

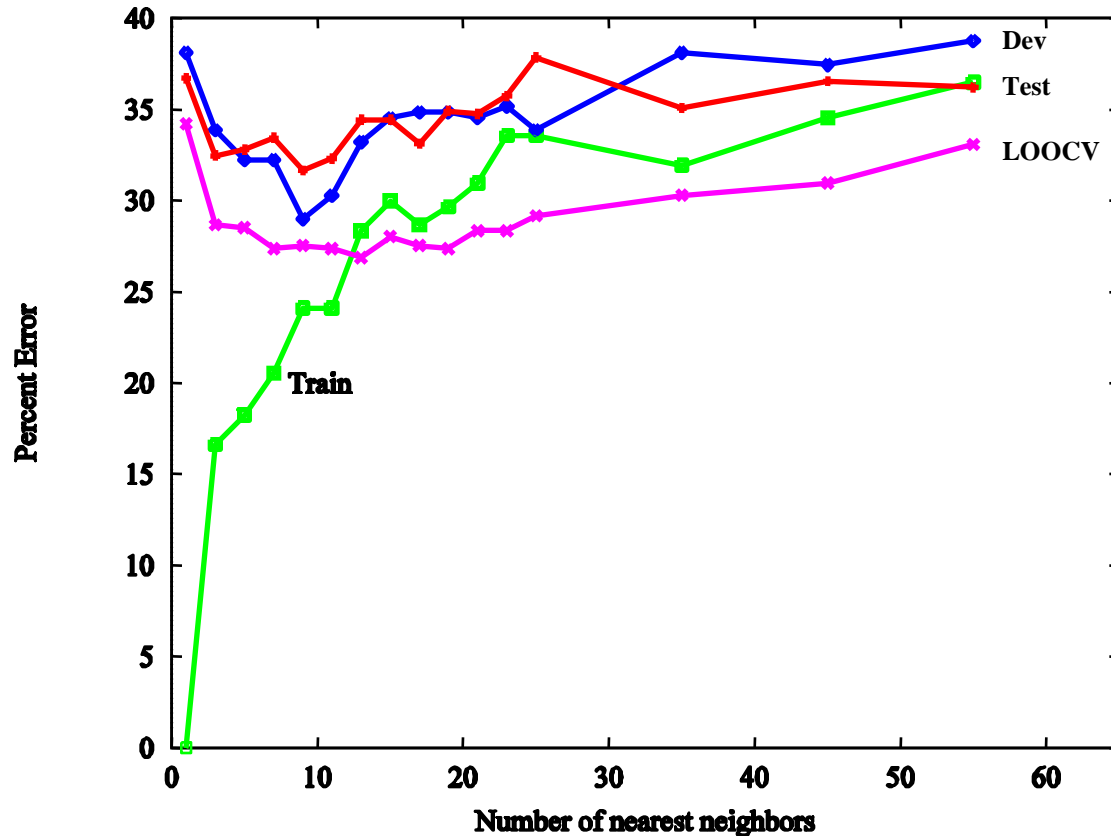


# Reconstituted Early Stopping



- We can stop either when MSE on the training set matches the predicted optimal MSE or when the number of epochs matches the predicted optimal number of epochs
- Experimental studies show little or no advantage for reconstituted early stopping. Most people just use simple holdout.

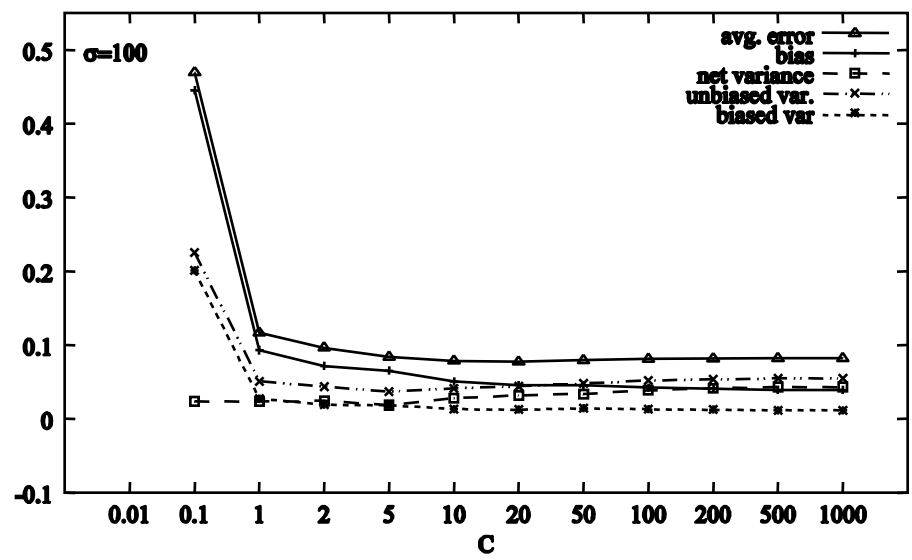
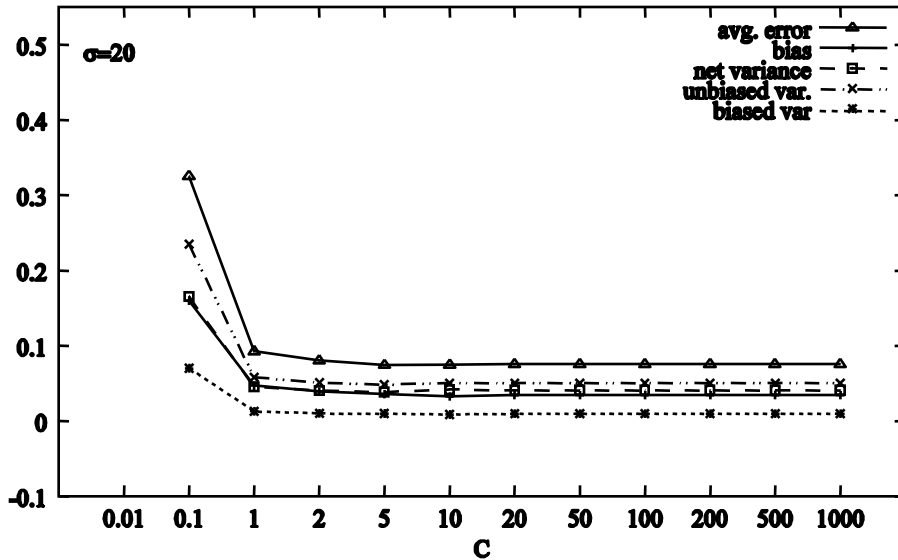
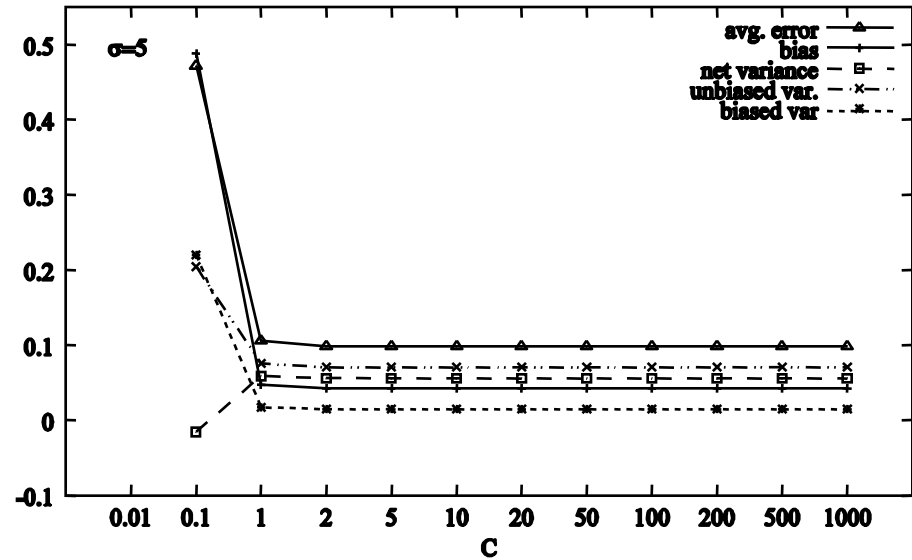
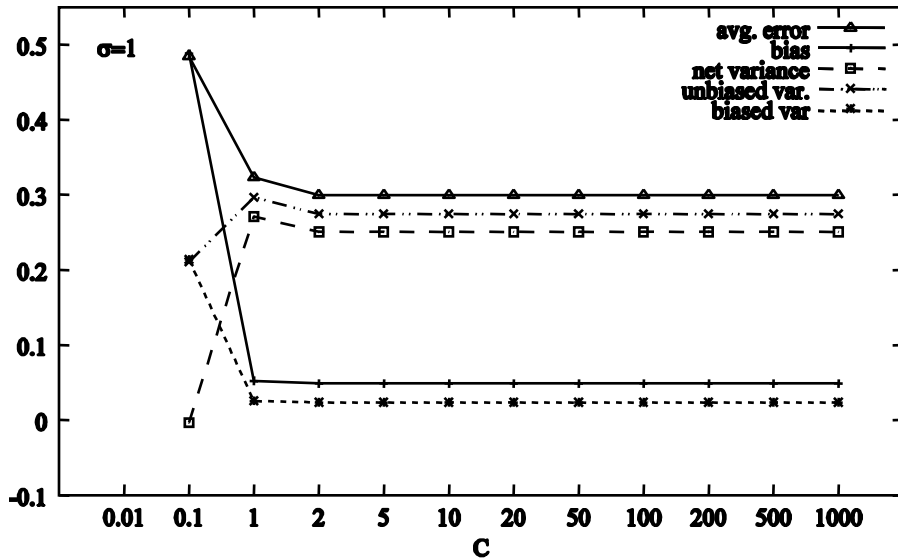
# Nearest Neighbor: Choosing k



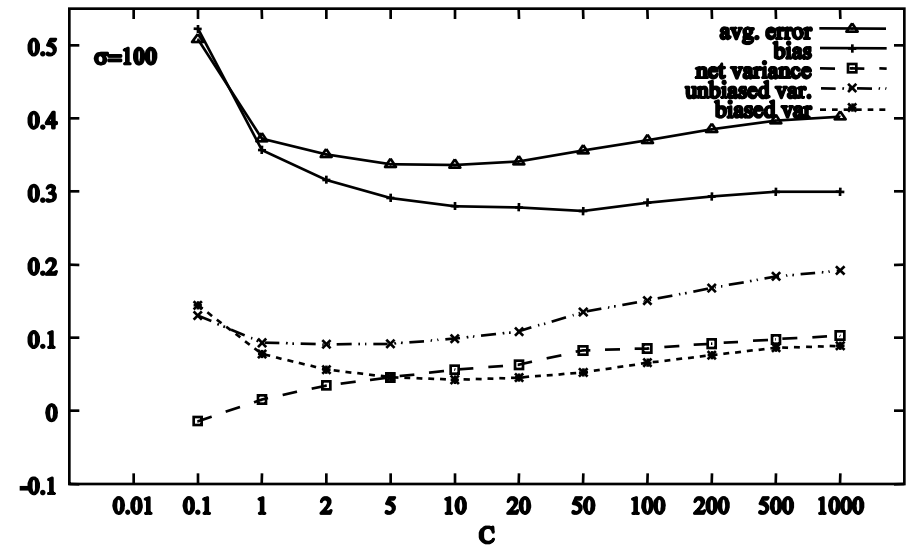
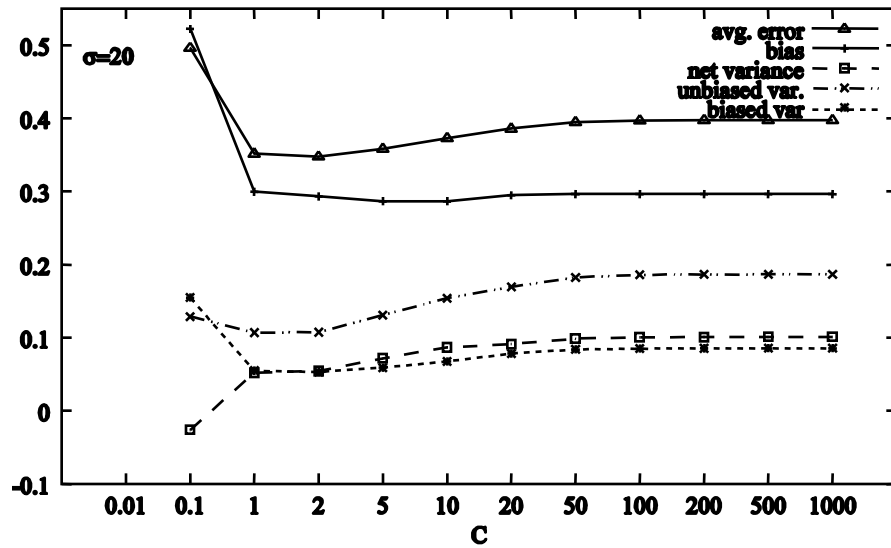
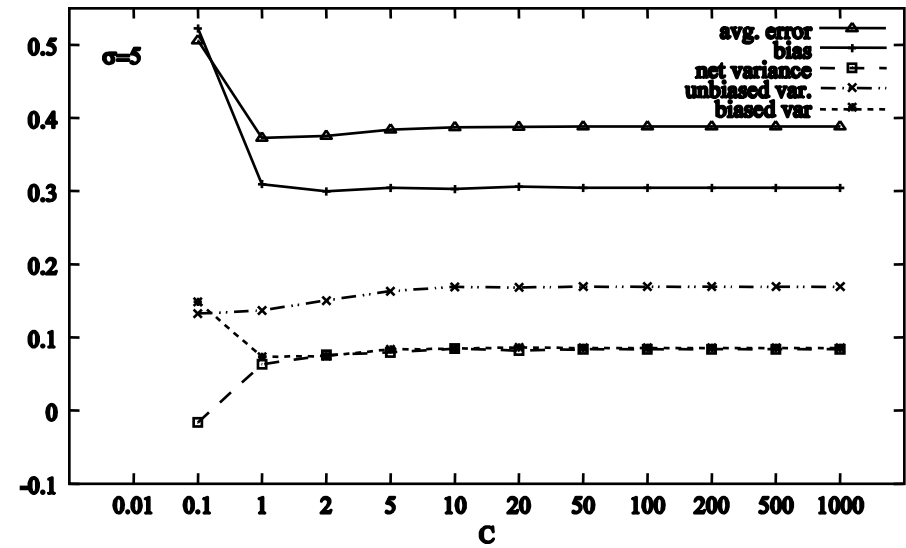
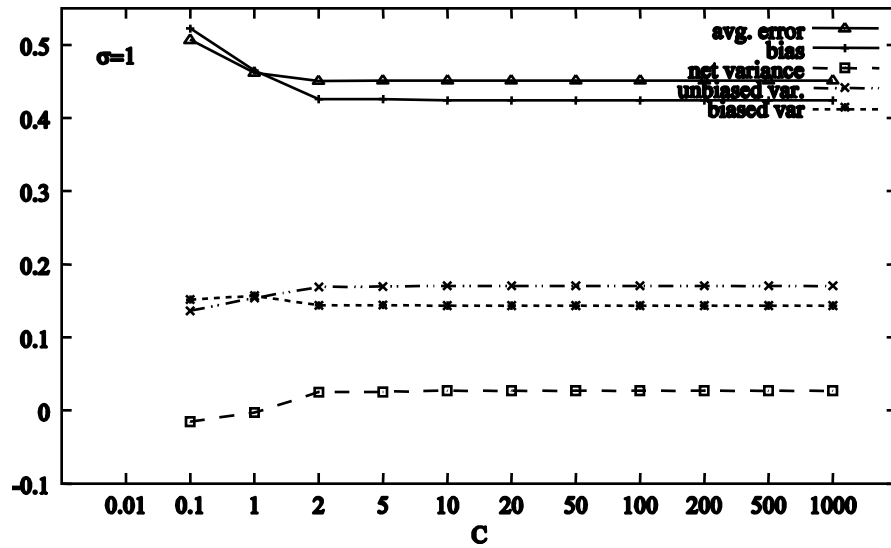
- $k=9$  gives best performance on development set
- $k=13$  gives best performance based on leave-one-out cross-validation

# SVM Choosing $C$ and $\sigma$

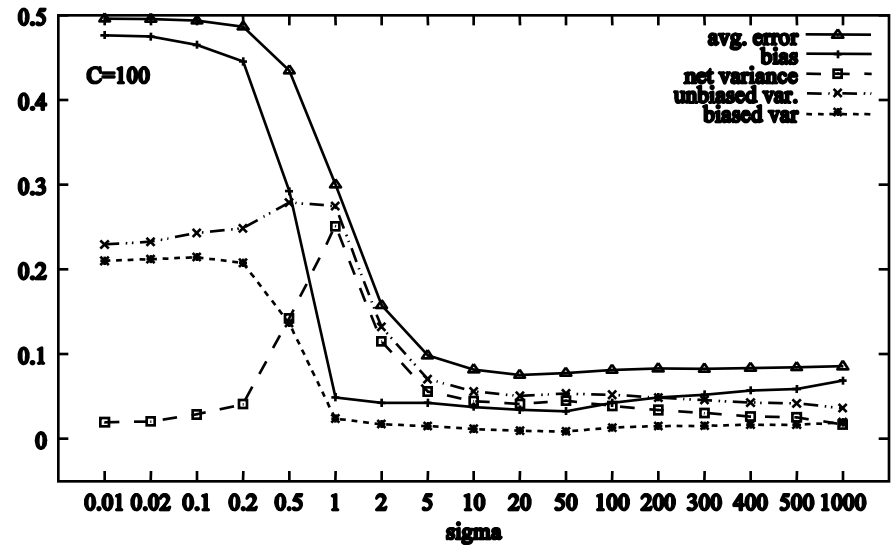
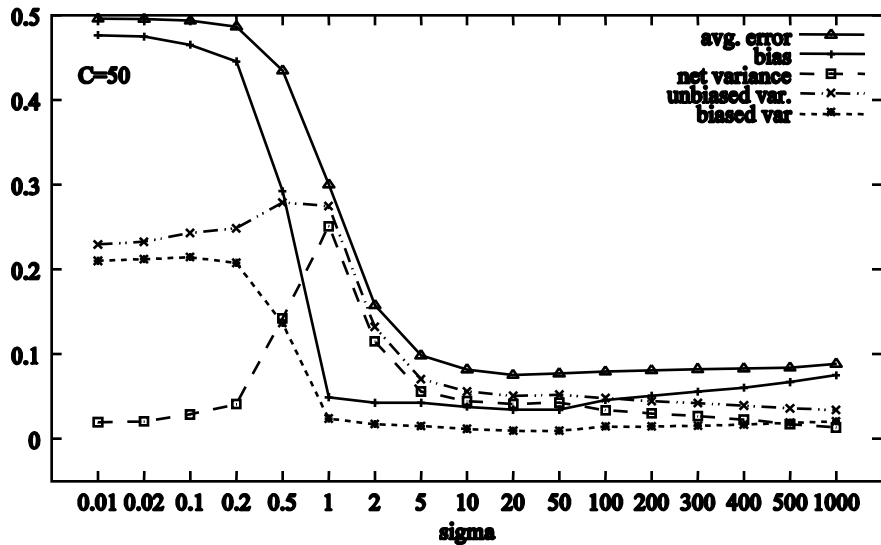
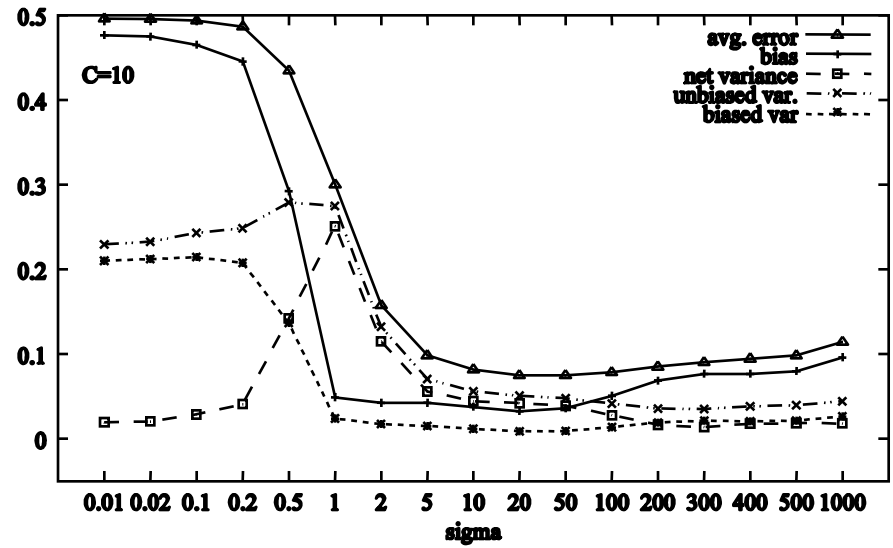
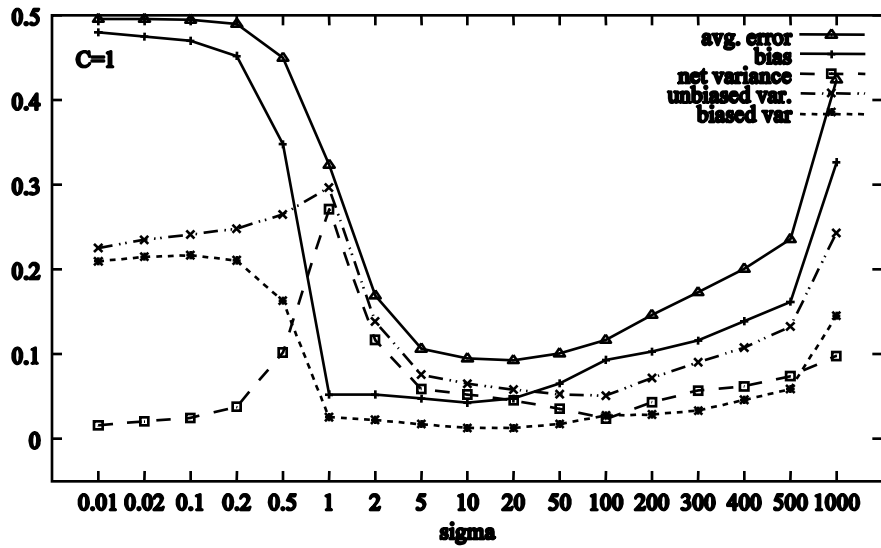
(BR Data Set; 100 examples; Valentini 2003)



# 20% label noise



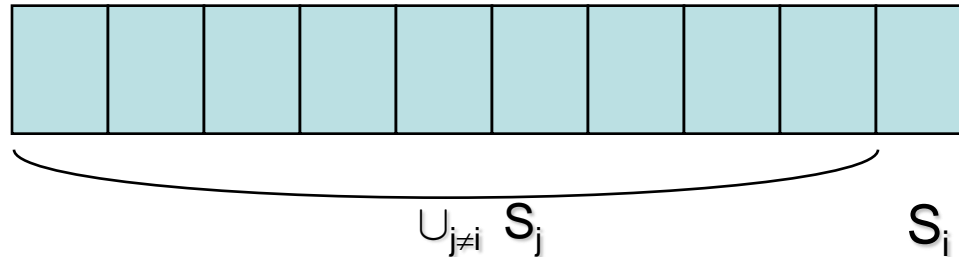
# BR Data Set: Varying $\sigma$ for fixed $C$



# Two Uses for Cross-Validation

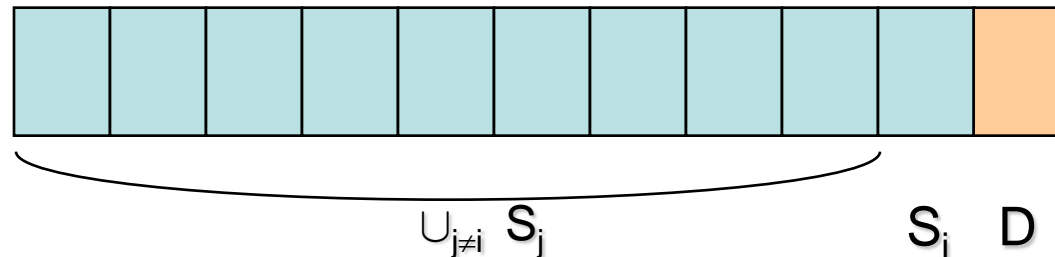
- Estimating the Error Rate of a Learning Algorithm
  - very important for choosing the right algorithm
  - a key problem in machine learning research
- Choosing parameter values ( $\alpha$ ,  $C$ ,  $\sigma$ , etc.)
- What if we need to do BOTH things?

# Nested Cross-Validation



- Outer Loop: estimate test set error by cross-validation:
  - For  $i$  from 1 to 10 do
    - Apply algorithm  $A$  to  $U_{j \neq i} S_j$  to obtain  $h_i$
    - Evaluate  $h_i$  on  $S_i$  to obtain  $\varepsilon_i$
  - Estimated error rate is  $(\sum_i \varepsilon_i)/10$
- Inner Loop: implement algorithm  $A$  by choosing internal parameters for algorithm  $B(\alpha)$  via 9-fold cross-validation
  - For each candidate  $\alpha$  do
    - For  $k$  from 1 to 10 except  $k=i$  do
      - Apply  $B(\alpha)$  to  $U_{j|j \neq i, j \neq k} S_j$  to obtain  $h_k$
      - Evaluate  $h_k$  on  $S_k$  to obtain  $\delta_k$
    - Estimated error rate is  $(\sum_k \delta_k)/9$
  - Choose  $\alpha^*$  with smallest error rate
- Requires  $10 \times 9 \times K$  executions of algorithm  $B(\alpha)$ , where  $K$  is the number of candidate values for  $\alpha_k$

# Alternative: Nested Holdout



- Divide data in 11 subsets:  $S_0, \dots, S_9, D$
- Outer Loop:
  - For  $i$  from 1 to 10 do
    - Apply algorithm A to  $\bigcup_{j \neq i} S_j$  to obtain  $h_i$
    - Evaluate  $h_i$  on  $S_i$  to obtain  $\varepsilon_i$
  - Estimate error rate as  $(\sum_i \varepsilon_i)/10$
- Inner Loop:
  - For each candidate  $\alpha_k$  do
    - Apply algorithm  $B(\alpha_k)$  to  $\bigcup_{j \neq i} S_j$  to obtain  $h_k$
    - Evaluate  $h_k$  on  $D$  to obtain  $\delta_k$
  - Choose the  $\alpha_k$  that minimizes  $\delta_k$  and return  $h_k$
- Requires  $10 \times K$  executions of  $B(\alpha)$ , where  $K$  is the number of candidate  $\alpha$  values

# Summary

- Holdout methods are the best way to choose a classifier
  - Reduce error pruning for trees
  - Early stopping for neural networks
- Cross-validation methods are the best way to set a regularization parameter
  - Cost-complexity pruning parameter  $\alpha$
  - Neural network weight decay setting
  - Number  $k$  of nearest neighbors in  $k$ -NN
  - $C$  and  $\sigma$  for SVMs