

# 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))**

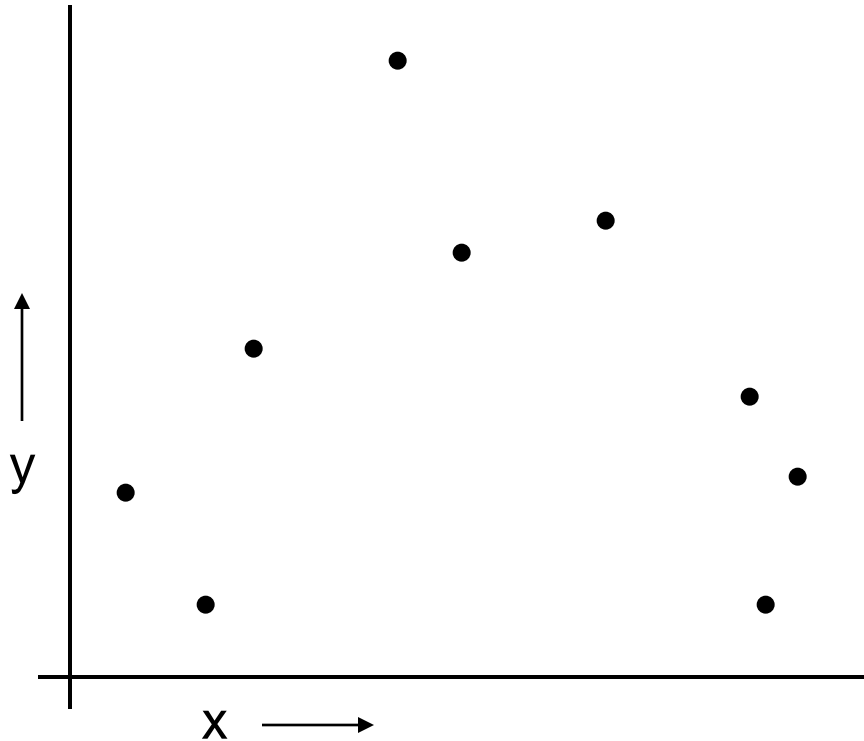
**Office: SAL 229**

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

**Class web page:**

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

# A Regression Problem

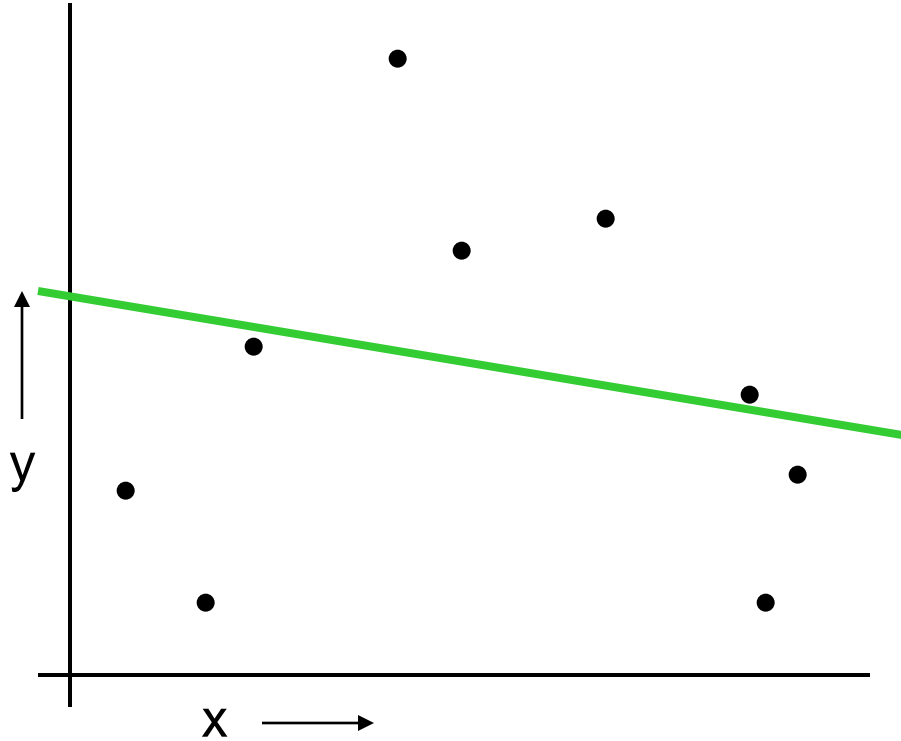


$$y = f(x) + \text{noise}$$

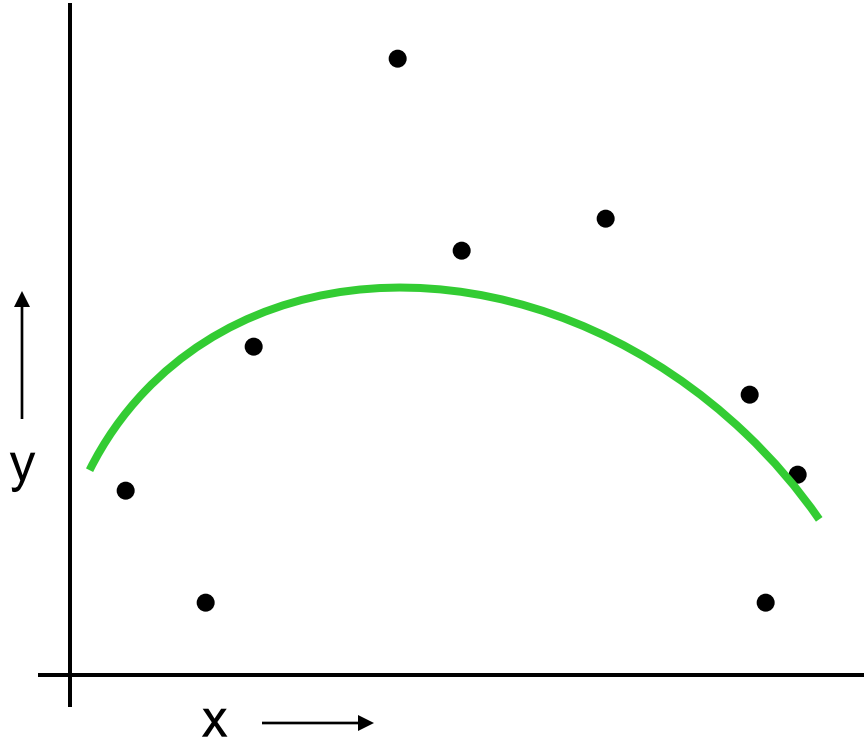
Can we learn  $f$  from this data?

Let's consider three methods...

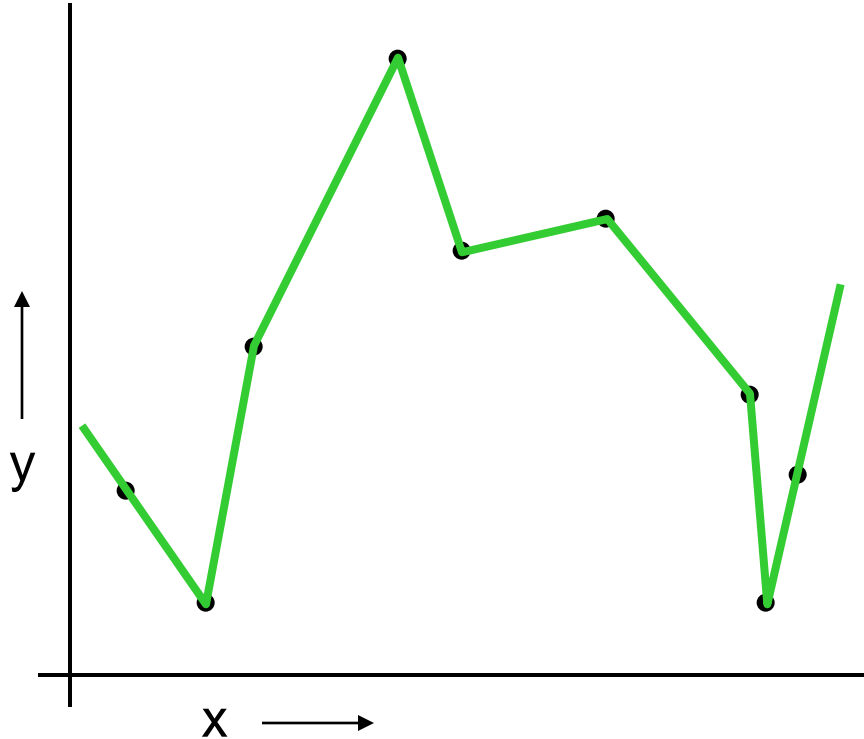
# Linear Regression



# Quadratic Regression

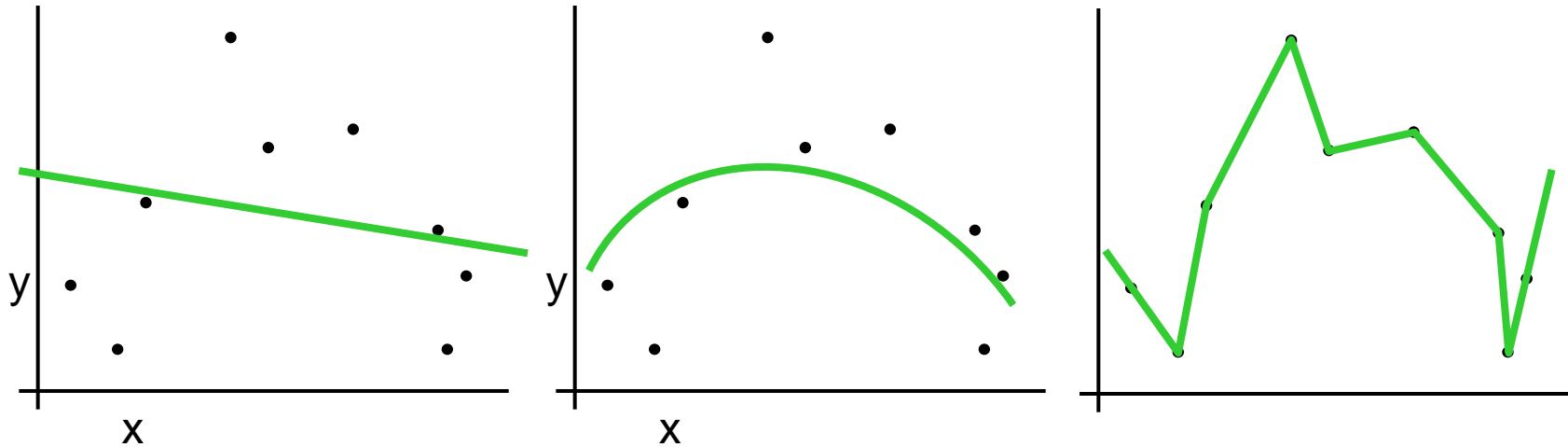


# Join-the-dots



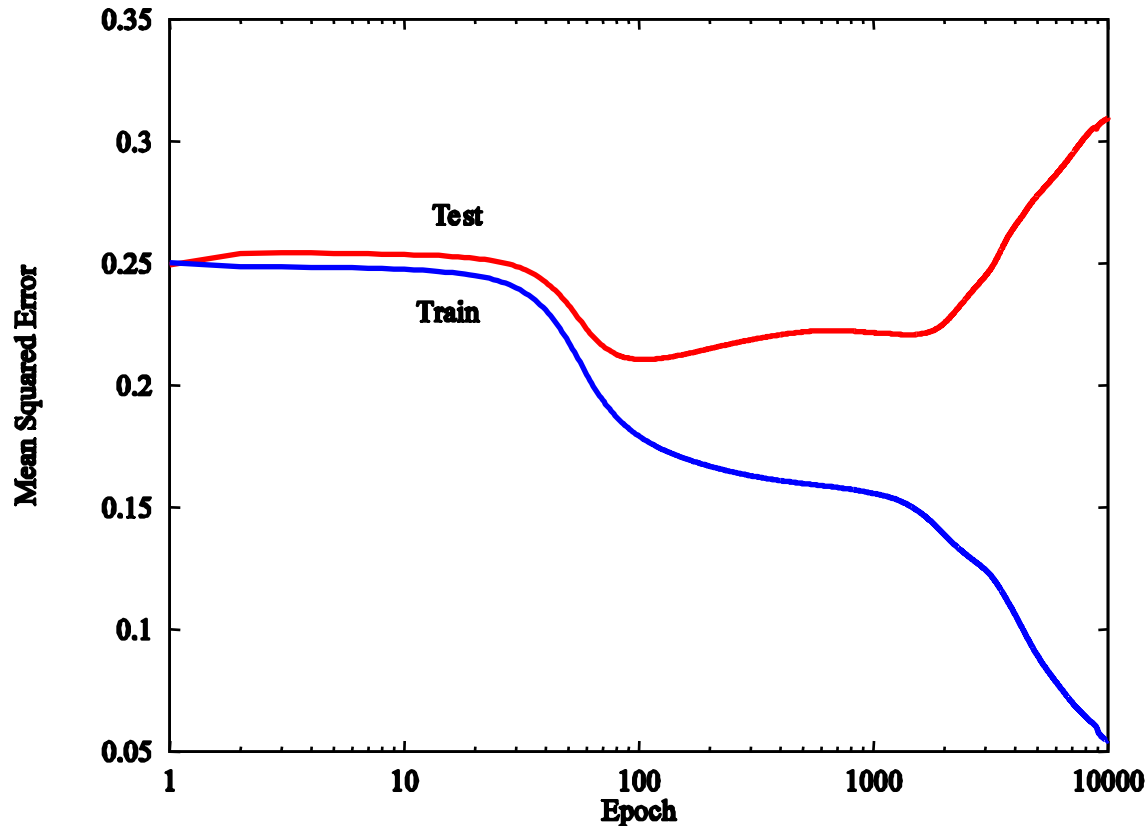
Also known as **piecewise linear nonparametric regression**

# What do we really want?



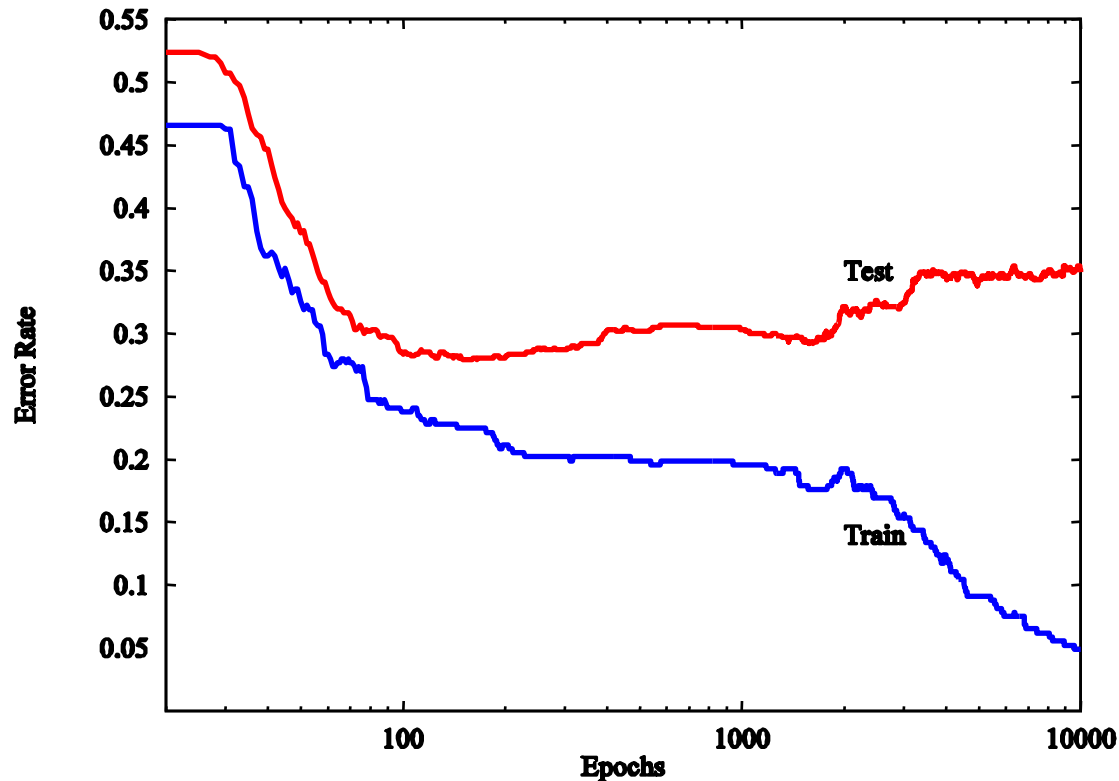
Why not choose the method with the best fit to the data?

# The Problem of Overfitting



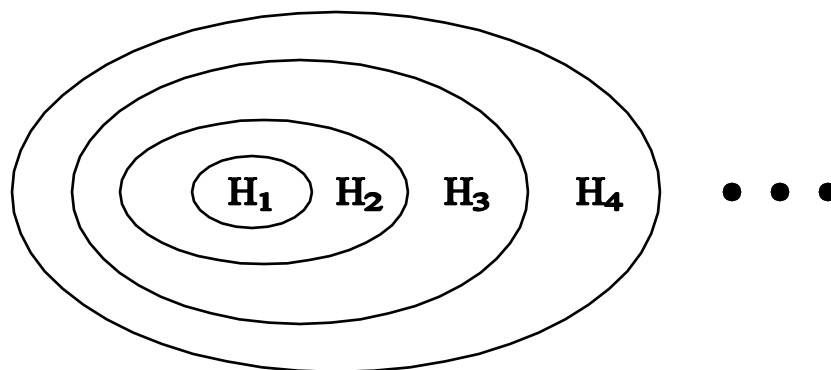
BR data: neural network with 20% classification noise, 307 training examples

# Overfitting on BR (2)



- **Overfitting:**  $h \in H$  overfits training set  $S$  if there exists  $h' \in H$  that has higher training set error but lower test error on new data points. (More specifically, if learning algorithm  $A$  explicitly considers and rejects  $h'$  in favor of  $h$ , we say that  $A$  has overfit the data.)

# Overfitting and model selection



$$H_1 \subset H_2 \subset H_3 \subset \dots$$

- If we use an hypothesis space  $H_i$  that is too large, eventually we can trivially fit the training data. In other words, the VC dimension will eventually be equal to the size of our training sample  $m$ .
- This is sometimes called model selection, because we can think of each  $H_i$  as an alternative “model” of the data

# Approaches to Preventing Overfitting

- Penalty methods
  - MAP provides a penalty based on  $P(H)$
  - Minimum Description Length (MDL) principle
  - Structural Risk Minimization
  - Generalized Cross-validation
  - Akaike's Information Criterion (AIC)
- Holdout and Cross-validation methods
  - Experimentally determine when overfitting occurs
- Ensembles
  - Full Bayesian methods vote many hypotheses
$$\sum_h P(y|\mathbf{x},h) P(h|S)$$
  - Many practical ways of generating ensembles

# Penalty methods

- Let  $\varepsilon_{\text{train}}$  be our training set error and  $\varepsilon_{\text{test}}$  be our test error. Our real goal is to find the  $h$  that minimizes  $\varepsilon_{\text{test}}$ . The problem is that we can't directly evaluate  $\varepsilon_{\text{test}}$ . We can measure  $\varepsilon_{\text{train}}$ , but it is optimistic.
- Penalty methods attempt to find some penalty such that
$$\varepsilon_{\text{test}} = \varepsilon_{\text{train}} + \text{penalty}$$
where we directly penalize model complexity
- The penalty term is also called a regularizer or regularization term.
- During training, we set our objective function  $J$  to be
$$J(\mathbf{w}) = \varepsilon_{\text{train}}(\mathbf{w}) + \text{penalty}(\mathbf{w})$$
and find the  $\mathbf{w}$  to minimize this function

# Penalty methods

- We can also represent the error function as:

$$\varepsilon_{\text{test}} = \varepsilon_{\text{train}} + \lambda \cdot (\text{model complexity})$$

- For example, in the case of feed-forward neural nets, we can use the norm of the weight vector as a measure of model complexity
- Note that this penalizes mainly large weights, but also, possibly, many weights (many connections)
- $\lambda$  is crucial and controls the bias-variance trade-off
  - if it is too high, only simple models are allowed and we introduce bias
  - if it is too low, we only look at error, so there may be overfitting (high variance)
- Often  $\lambda$  is itself tuned with cross-validation (coming up...)

# MAP penalties

$$h_{\text{map}} = \operatorname{argmax}_h P(S|h) P(h)$$

- As  $h$  becomes more complex, we can assign it a lower prior probability. A typical approach is to assign equal probability to each of the nested hypothesis spaces so that

$$P(h \in H_1) = P(h \in H_2) = \dots = \alpha$$

- Because  $H_2$  contains more hypotheses than  $H_1$ , each individual  $h \in H_2$  will have lower prior probability:

$$P(h) = \sum_i P(h \in H_i) = \sum_i \alpha / |H_i| \text{ for each } i \text{ where } h \in H_i$$

- If there are infinitely many  $H_i$ , this will not work, because the probabilities must sum to 1. In this case, a common approach is

$$P(h) = \sum_i 2^{-i} / |H_i| \text{ for each } i \text{ where } h \in H_i$$

This is not usually a big enough penalty to prevent overfitting, however

# MDL penalties

- Consider the following alternative view of  $h_{\text{map}}$ :

$$\begin{aligned}h_{\text{map}} &= \operatorname{argmax}_h P(S|h) P(h) \\ &= \operatorname{argmax}_h [\log_2 (S|h) + \log_2 P(h)] \\ &= \operatorname{argmin}_h [-\log_2 (S|h) - \log_2 P(h)]\end{aligned}$$

- We know from information theory that optimal encoding for an event with probability  $p$  will provide a code of  $-\log_2 p$  bits.
- So we can interpret the above equation as:
  - $-\log_2 P(h)$  is length of  $h$  under the optimal encoding
  - $-\log_2 P(S|h)$  is length of  $S$  given  $h$  under the optimal encoding

# A communication Scenario

- Suppose I want to send you a large classification data set
- I could send you all the data as I have it
- Or I could try to formulate a hypothesis that *summarizes* the data
- If the hypothesis is perfect, you do not need to know the data at all!
- If it is not perfect, I may still need to send you the exceptions (the instances for which my hypothesis is incorrect)

# Minimum Description Length Principle (MDL)

$$h_{\text{map}} = \operatorname{argmin}_h L_{C_1}(h) + L_{C_2}(S|h)$$

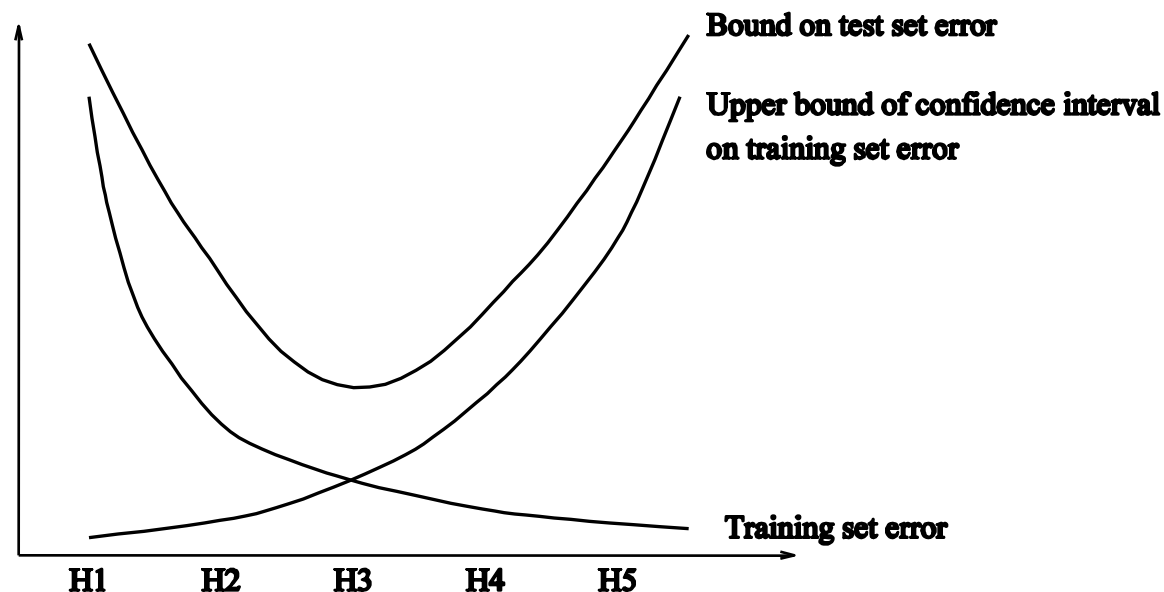
- MDL: Suppose I want to send my data  $S$  over. The best hypothesis is the one that minimizes the length of the message to send.
  - We will send the hypothesis (encoded optimally)
  - If data is correctly labeled by the hypothesis, we do not need to send it ( $h$  summarizes it)
  - But we do need to send the misclassified data
- *This is precisely the MAP hypothesis!*

# Structural Risk Minimization

- Define regularization penalty using PAC theory

$$\epsilon \leq 2\epsilon_T^k + \frac{4}{m} \left[ d_k \log \frac{2em}{d_k} + \log \frac{4}{\delta} \right]$$

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



# Other Penalty Meth

- Generalized Cross Validation
- Akaike's Information Criterion (AIC)

$$\text{AICSCORE} = LL(\text{Data} | \text{MLE params}) - (\# \text{ parameters})$$

- Bayesian Information Criterion (BIC)

$$\text{BICSCORE} = LL(\text{Data} | \text{MLE params}) - \frac{\# \text{ params}}{2} \log N$$

- ...

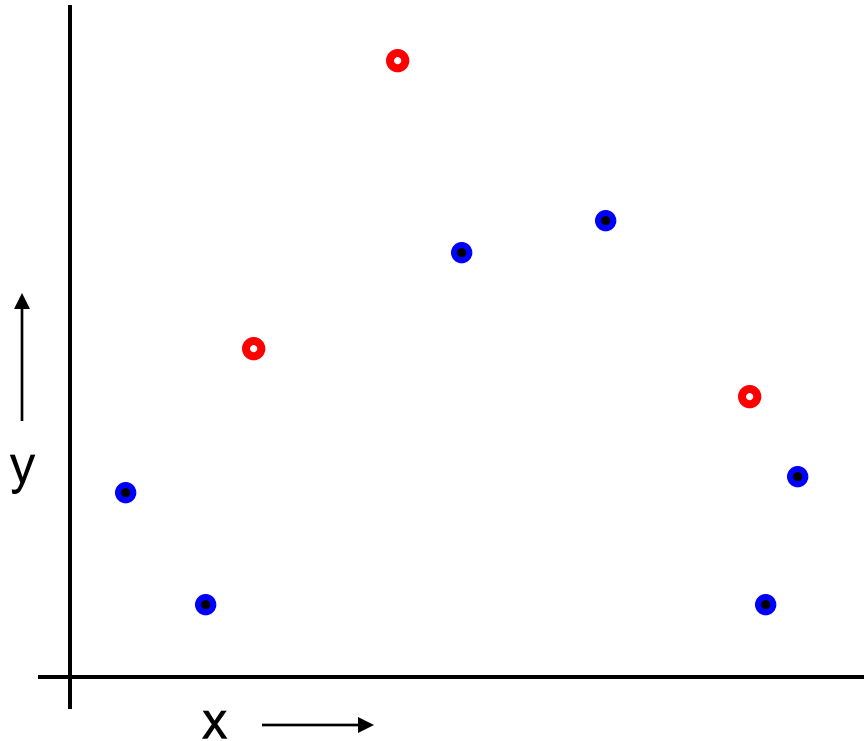
As the amount of data goes to infinity, AIC promises\* to select the model that'll have the best likelihood for future data

\*Subject to about a million caveats

As the amount of data goes to infinity, BIC promises\* to select the model that the data was generated from. More conservative than AIC.

\*Another million caveats

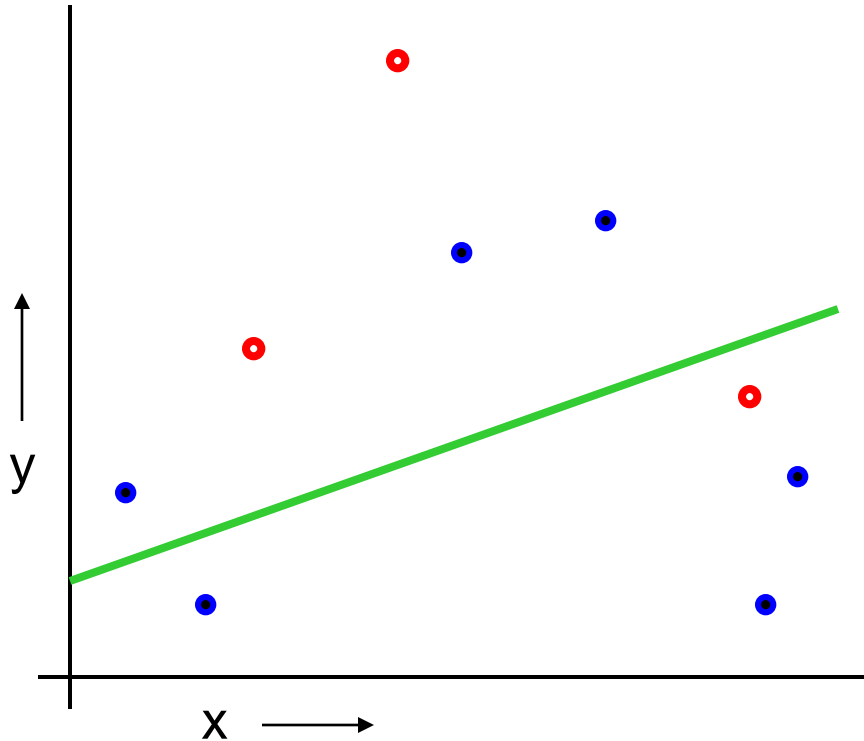
# Simple Holdout Method



1. Randomly choose 30% of the data to be in a **test set**

2. The remainder is a **training set**

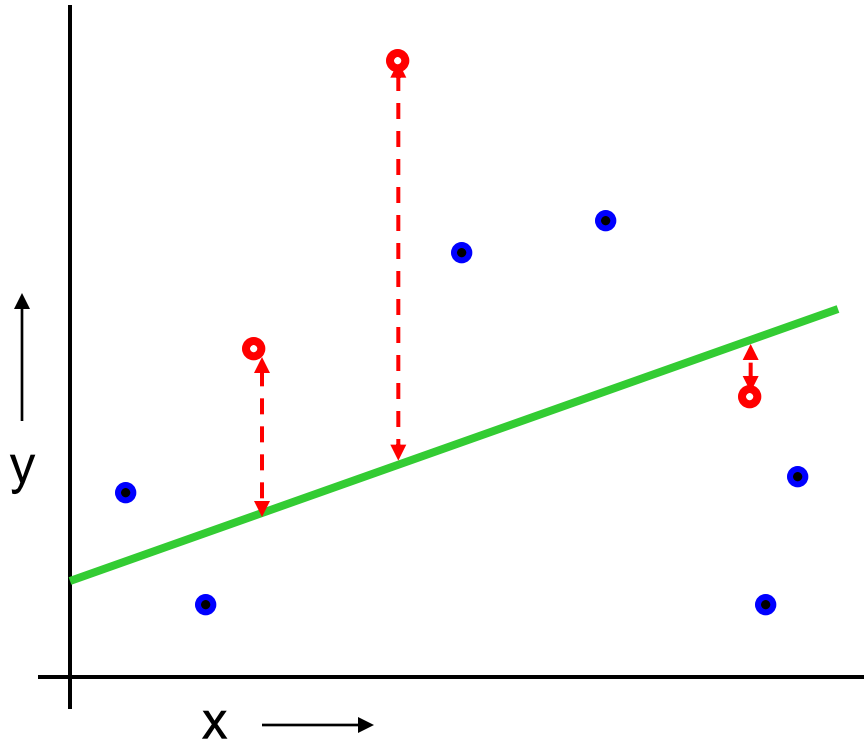
# Simple Holdout Method



(Linear regression example)

1. Randomly choose 30% of the data to be in a **test set**
2. The remainder is a **training set**
3. Perform your regression on the training set

# Simple Holdout Method



(Linear regression example)

Mean Squared Error = 2.4

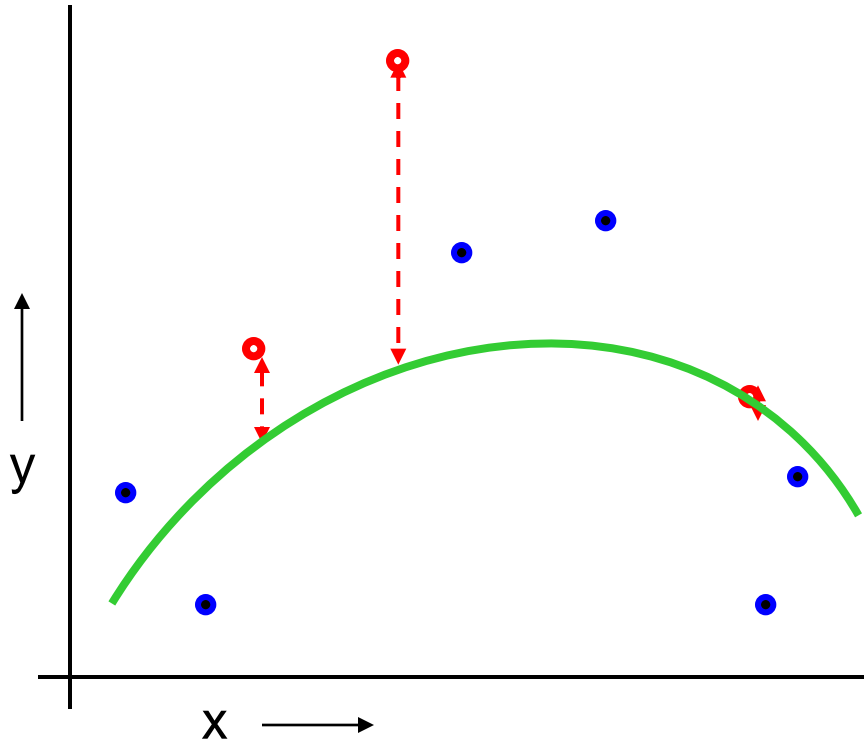
1. Randomly choose 30% of the data to be in a **test set**

2. The remainder is a **training set**

3. Perform your regression on the training set

4. Estimate your future performance with the **test set**

# Simple Holdout Method



(Quadratic regression example)

Mean Squared Error = 0.9

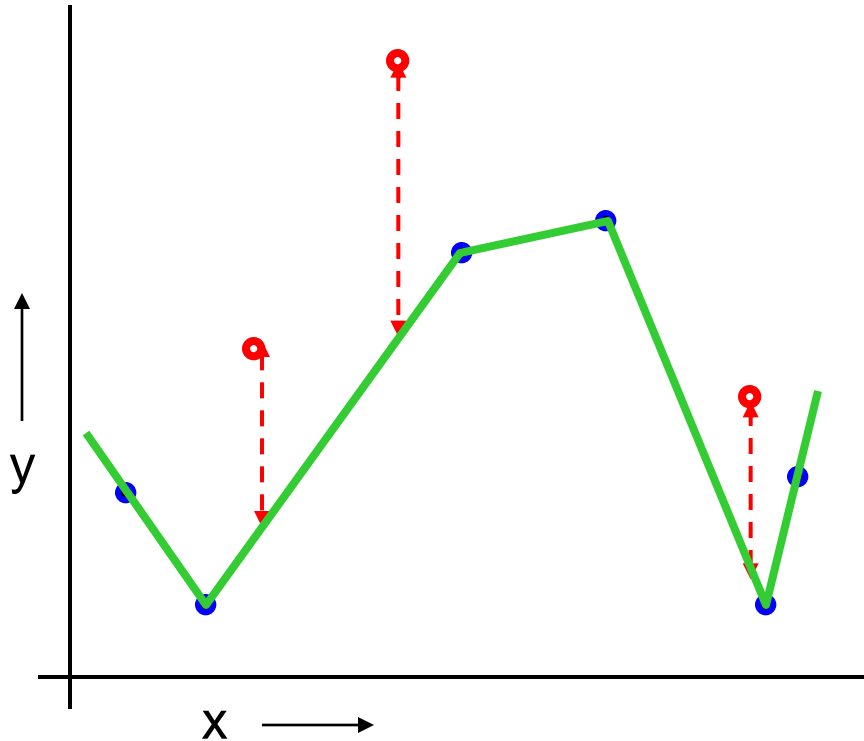
1. Randomly choose 30% of the data to be in a **test set**

2. The remainder is a **training set**

3. Perform your regression on the training set

4. **Estimate your future performance with the test set**

# Simple Holdout Method



(Join the dots example)

Mean Squared Error = 2.2

1. Randomly choose 30% of the data to be in a **test set**

2. The remainder is a **training set**

3. Perform your regression on the training set

4. **Estimate your future performance with the test set**

# Simple Holdout Method

Good news:

- Very very simple
- Can then simply choose the method with the best test-set score

Bad news:

- What's the downside?

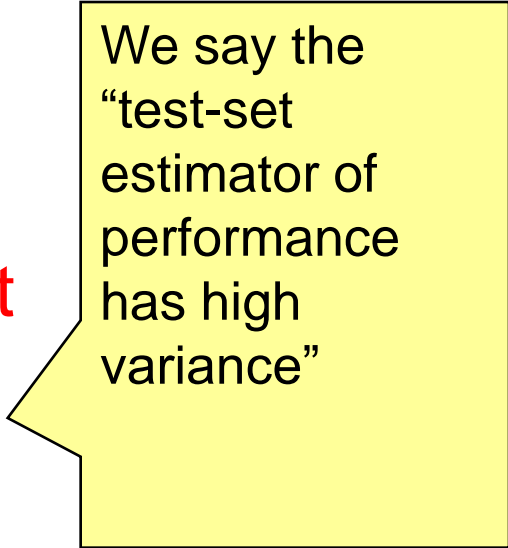
# Simple Holdout Method

## Good news:

- Very very simple
- Can then simply choose the method with the best test-set score

## Bad news:

- Wastes data: we get an estimate of the best method to apply to 30% less data
- If we don't have much data, our test-set might just be lucky or unlucky



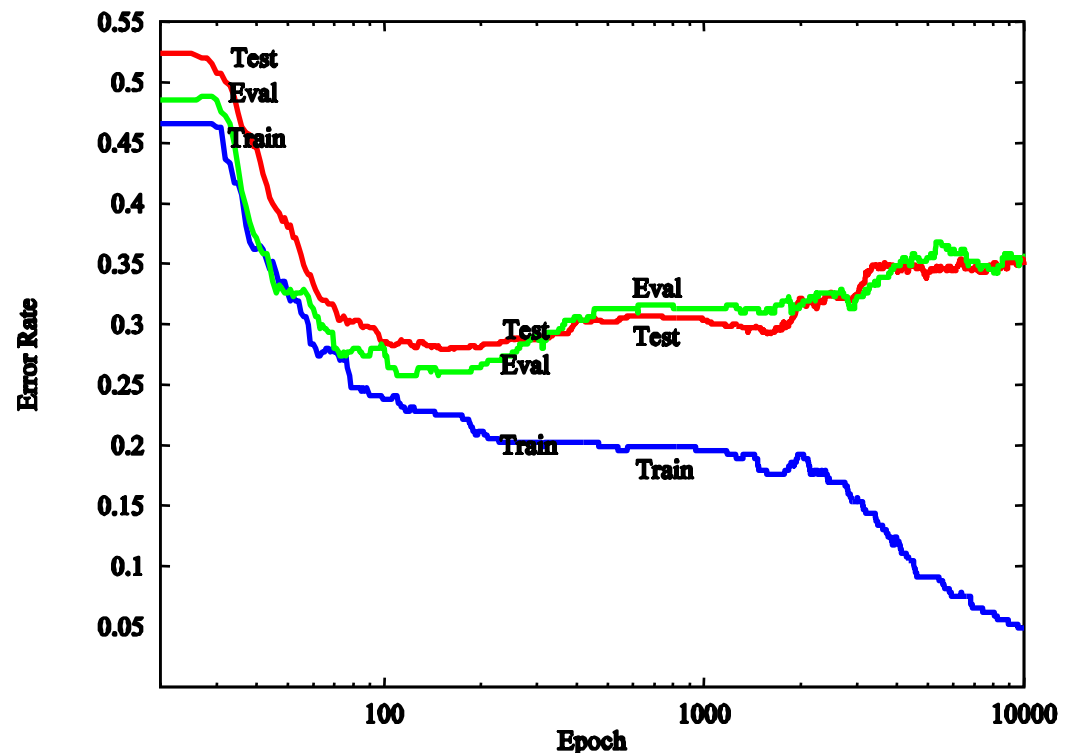
We say the “test-set estimator of performance has high variance”

# Simple Holdout Method

- Subdivide  $S$  into  $S_{\text{train}}$  and  $S_{\text{eval}}$
- For each  $H_i$ , find  $h_i \in H_i$  that best fits  $S_{\text{train}}$
- Measure the error rate of each  $h_i$  on  $S_{\text{eval}}$
- Choose  $h_i$  with the best error rate

Example: let  $H_i$  be the set of neural network weights after  $i$  epochs of training on  $S_{\text{train}}$

Our goal is to choose  $i$



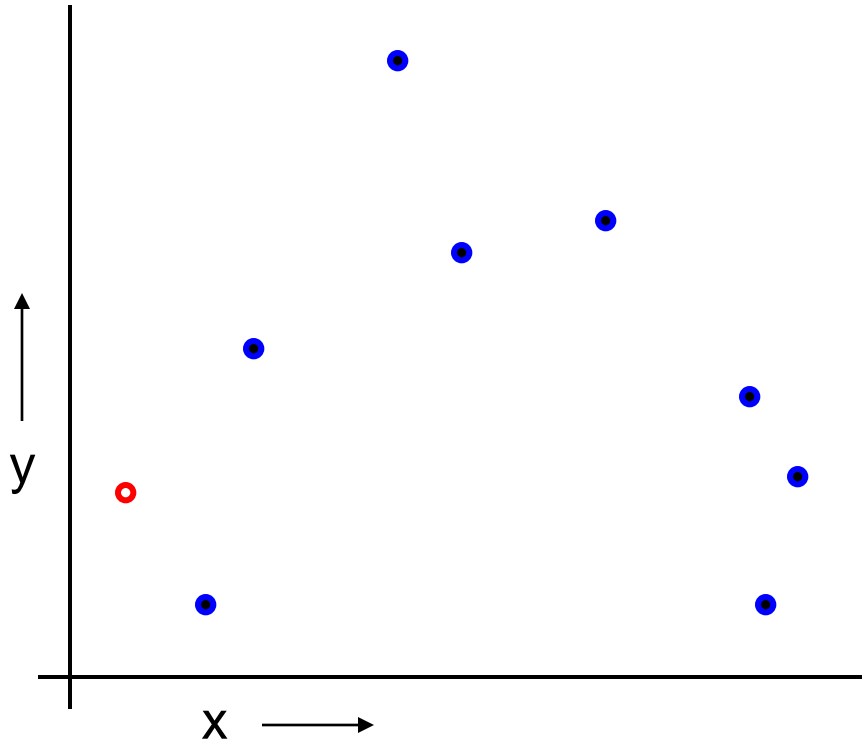
# Simple Holdout Assessment

- Advantages
  - Guaranteed to perform within a constant factor of any penalty method (Kearns, et al., 1995)
  - Does not rely on theoretical approximations
- Disadvantages
  - $S_{\text{train}}$  is smaller than  $S$ , so  $h$  is likely to be less accurate
  - If  $S_{\text{eval}}$  is too small, the error rate estimates will be very noisy
- Simple Holdout is widely applied to make other decisions such as learning rates, number of hidden units, SVM kernel parameters, relative size of penalty, which features to include, feature encoding methods, etc.

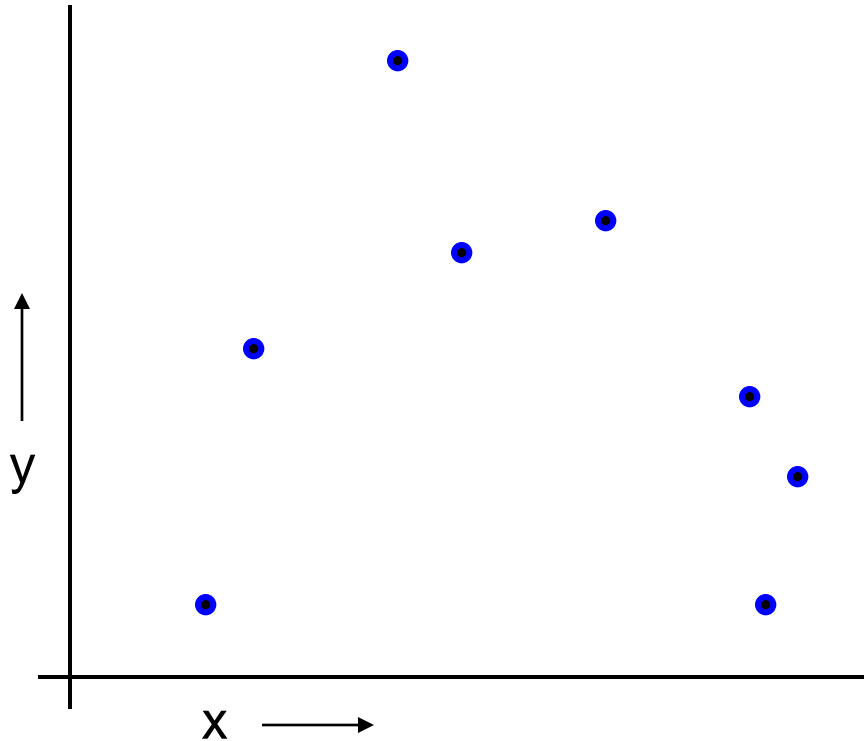
# LOOCV (Leave-one-out Cross Validation)

For  $k=1$  to  $R$

1. Let  $(x_k, y_k)$  be the  $k^{\text{th}}$  record



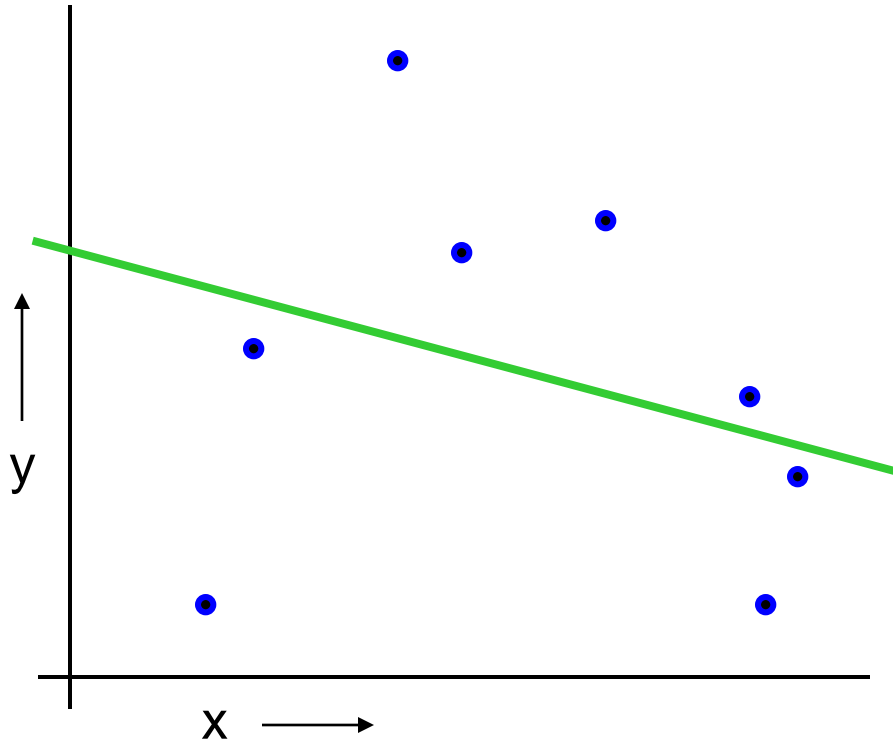
# LOOCV (Leave-one-out Cross Validation)



For  $k=1$  to  $R$

1. Let  $(x_k, y_k)$  be the  $k^{\text{th}}$  record
2. Temporarily remove  $(x_k, y_k)$  from the dataset

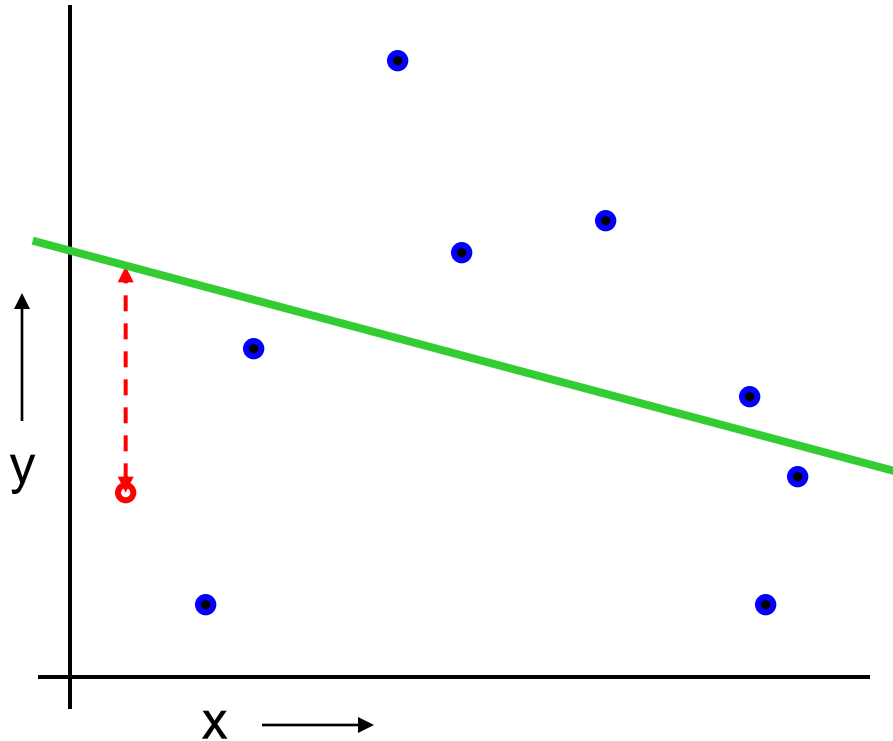
# LOOCV (Leave-one-out Cross Validation)



For  $k=1$  to  $R$

1. Let  $(x_k, y_k)$  be the  $k^{\text{th}}$  record
2. Temporarily remove  $(x_k, y_k)$  from the dataset
3. Train on the remaining  $R-1$  datapoints

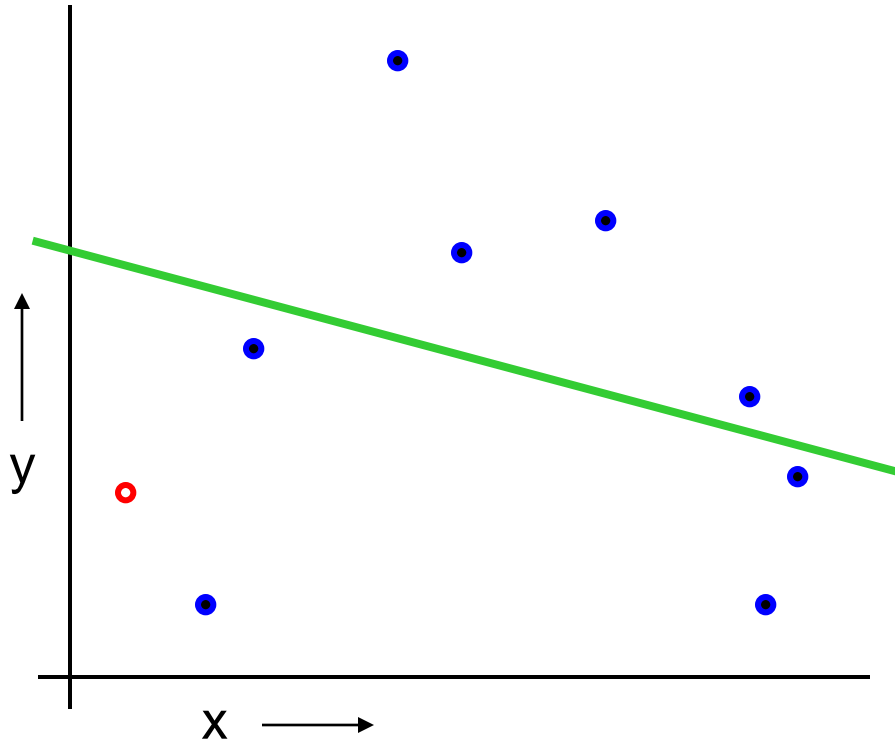
# LOOCV (Leave-one-out Cross Validation)



For  $k=1$  to  $R$

1. Let  $(x_k, y_k)$  be the  $k^{\text{th}}$  record
2. Temporarily remove  $(x_k, y_k)$  from the dataset
3. Train on the remaining  $R-1$  datapoints
4. Note your error  $(x_k, y_k)$

# LOOCV (Leave-one-out Cross Validation)

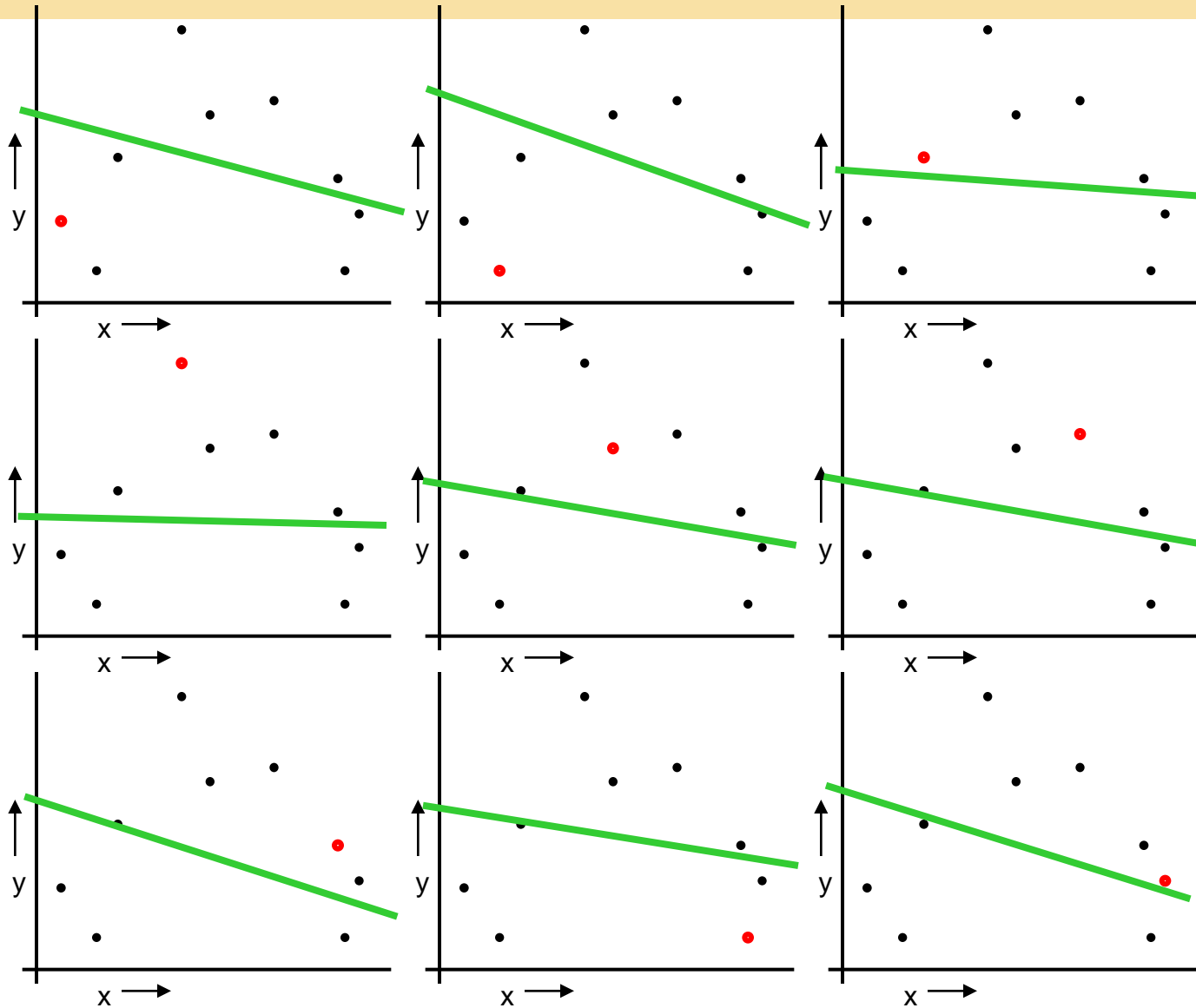


For  $k=1$  to  $R$

1. Let  $(x_k, y_k)$  be the  $k^{\text{th}}$  record
2. Temporarily remove  $(x_k, y_k)$  from the dataset
3. Train on the remaining  $R-1$  datapoints
4. Note your error  $(x_k, y_k)$

When you've done all points, report the mean error.

# LOOCV (Leave-one-out Cross Validation)



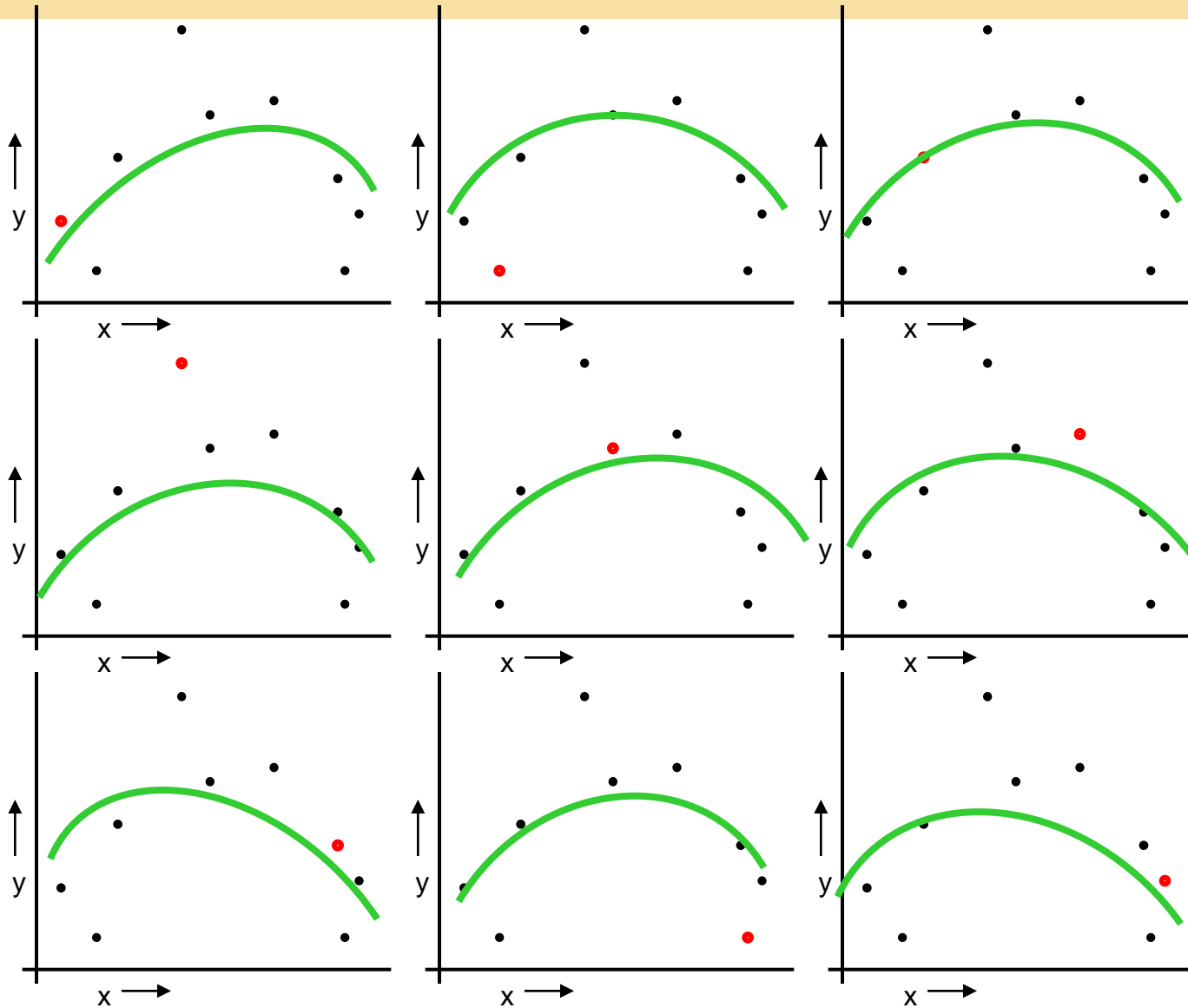
For  $k=1$  to  $R$

1. Let  $(x_k, y_k)$  be the  $k^{\text{th}}$  record
2. Temporarily remove  $(x_k, y_k)$  from the dataset
3. Train on the remaining  $R-1$  datapoints
4. Note your error  $(x_k, y_k)$

When you've done all points, report the mean error.

$$MSE_{LOOCV} = 2.12$$

# LOOCV for Quadratic Regression



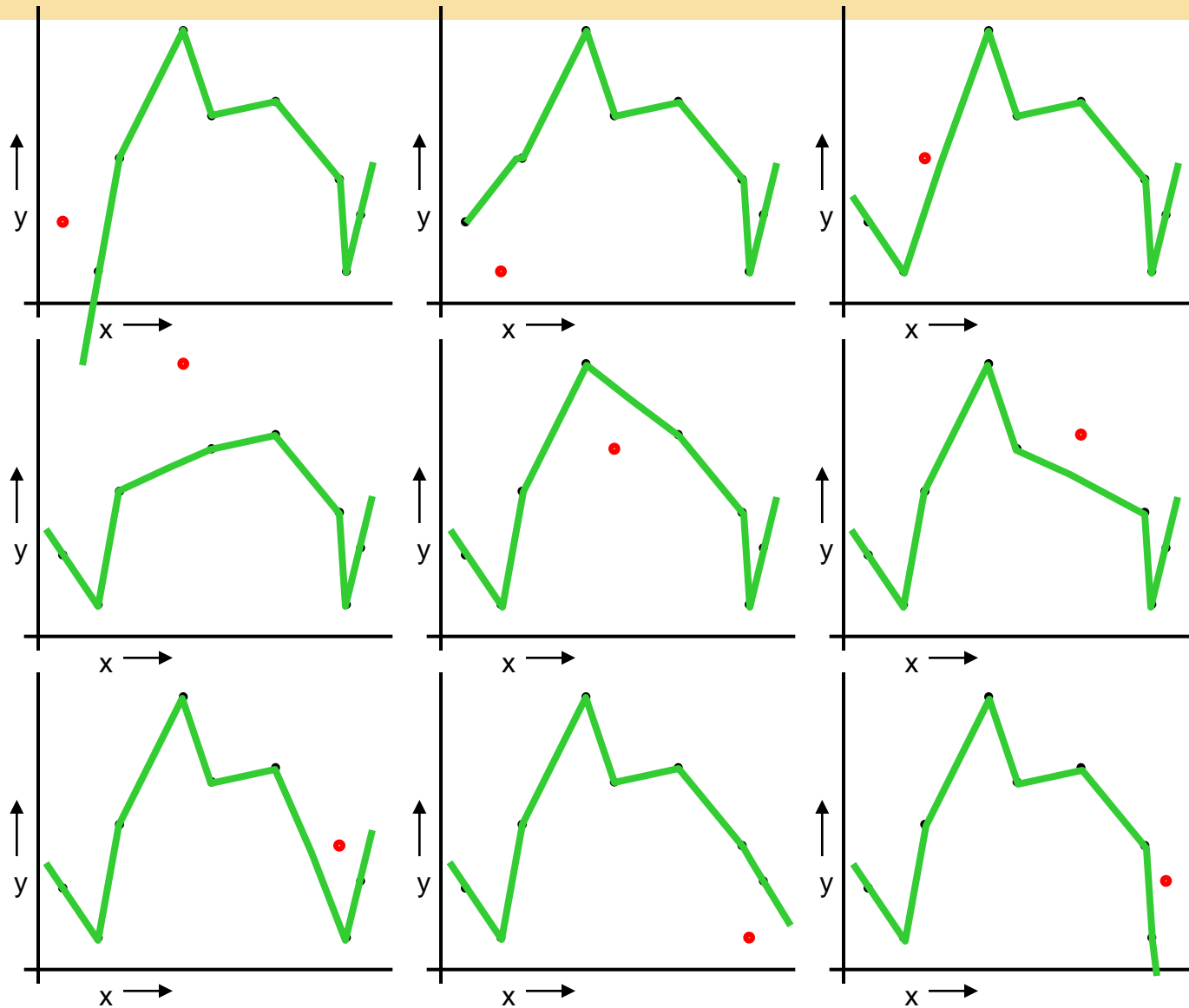
For  $k=1$  to  $R$

1. Let  $(x_k, y_k)$  be the  $k^{\text{th}}$  record
2. Temporarily remove  $(x_k, y_k)$  from the dataset
3. Train on the remaining  $R-1$  datapoints
4. Note your error  $(x_k, y_k)$

When you've done all points, report the mean error.

$$MSE_{LOOCV} = 0.962$$

# LOOCV for Join The Dots



For  $k=1$  to  $R$

1. Let  $(x_k, y_k)$  be the  $k^{\text{th}}$  record
2. Temporarily remove  $(x_k, y_k)$  from the dataset
3. Train on the remaining  $R-1$  datapoints
4. Note your error  $(x_k, y_k)$

When you've done all points, report the mean error.

$$MSE_{LOOCV} = 3.33$$

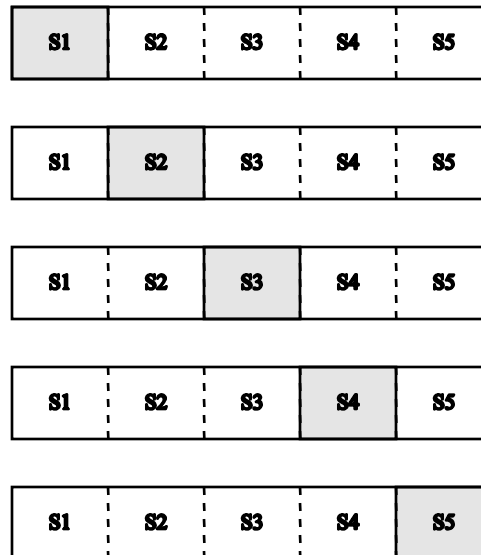
# Which kind of Cross Validation?

	<b>Downside</b>	<b>Upside</b>
<b>Test-set</b>	Variance: unreliable estimate of future performance	Cheap
<b>Leave-one-out</b>	Expensive. Has some weird behavior	Doesn't waste data

..can we get the best of both worlds?

# k-fold Cross-Validation to determine $H_i$

- Randomly divide  $S$  into  $k$  equal-sized subsets
- Run learning algorithm  $k$  times, each time use one subset for  $S_{\text{eval}}$  and the rest for  $S_{\text{train}}$
- Average the results



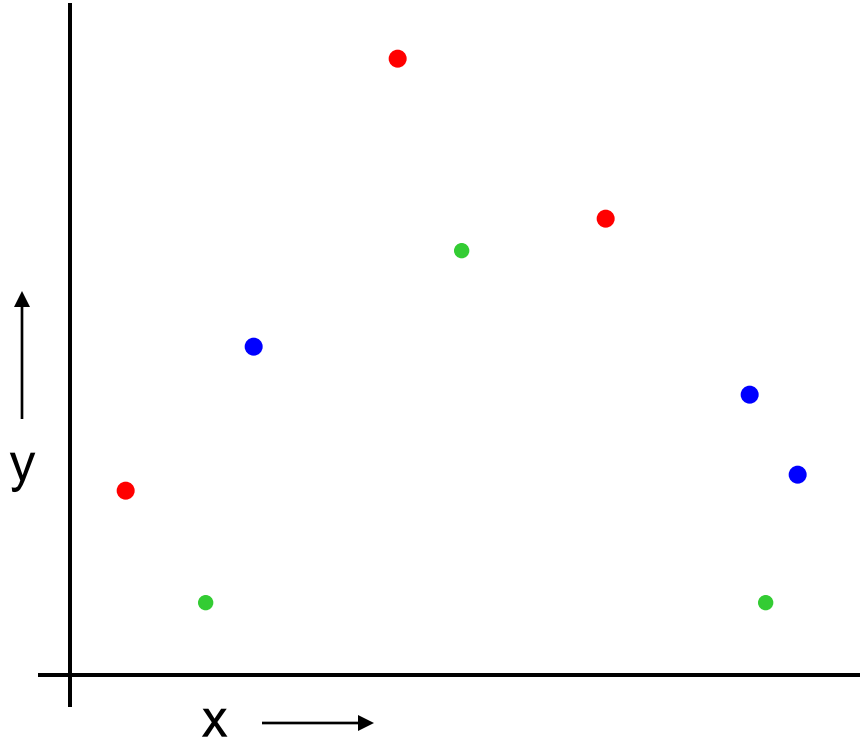
# K-fold Cross-Validation to determine $H_i$

- Partition  $S$  into  $K$  disjoint subsets  $S_1, S_2, \dots, S_k$
- Repeat simple holdout assessment  $K$  times
  - In the  $k$ -th assessment,  $S_{\text{train}} = S - S_k$  and  $S_{\text{eval}} = S_k$
  - Let  $h_i^k$  be the best hypothesis from  $H_i$  from iteration  $k$ .
  - Let  $\varepsilon_i$  be the average  $S_{\text{eval}}$  of  $h_i^k$  over the  $K$  iterations
  - Let  $i^* = \operatorname{argmin}_i \varepsilon_i$
- Train on  $S$  using  $H_{i^*}$  and output the resulting hypothesis

	$i$									
	0	1	2	3	4	5	6	7	8	9
$k$	1									
	2									
	3									
	4									
	5									

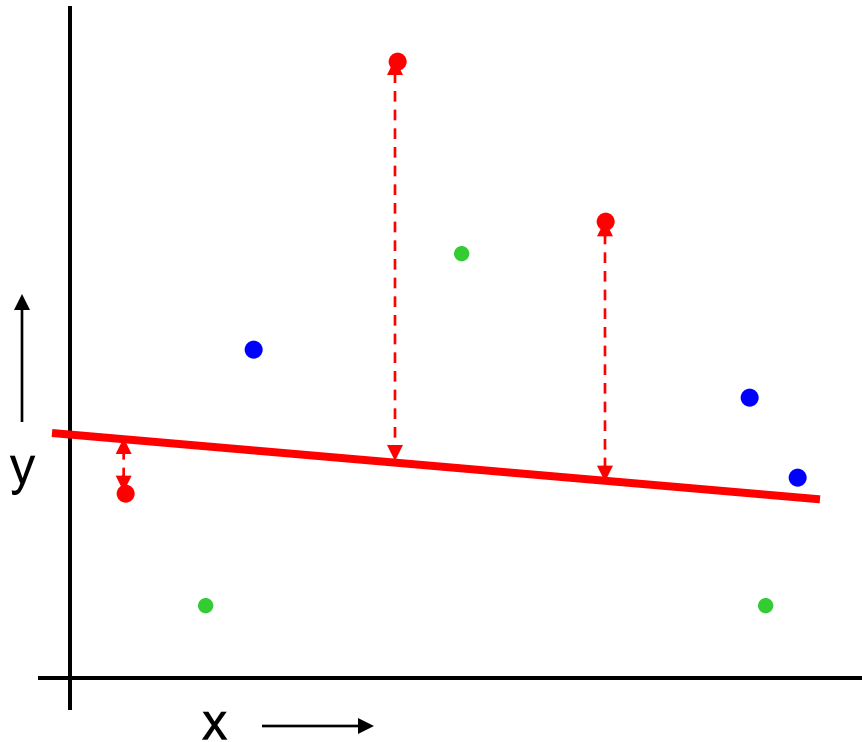
# k-fold Cross Validation

Randomly break the dataset into  $k$  partitions (in our example we'll have  $k=3$  partitions colored Red Green and Blue)



# k-fold Cross Validation

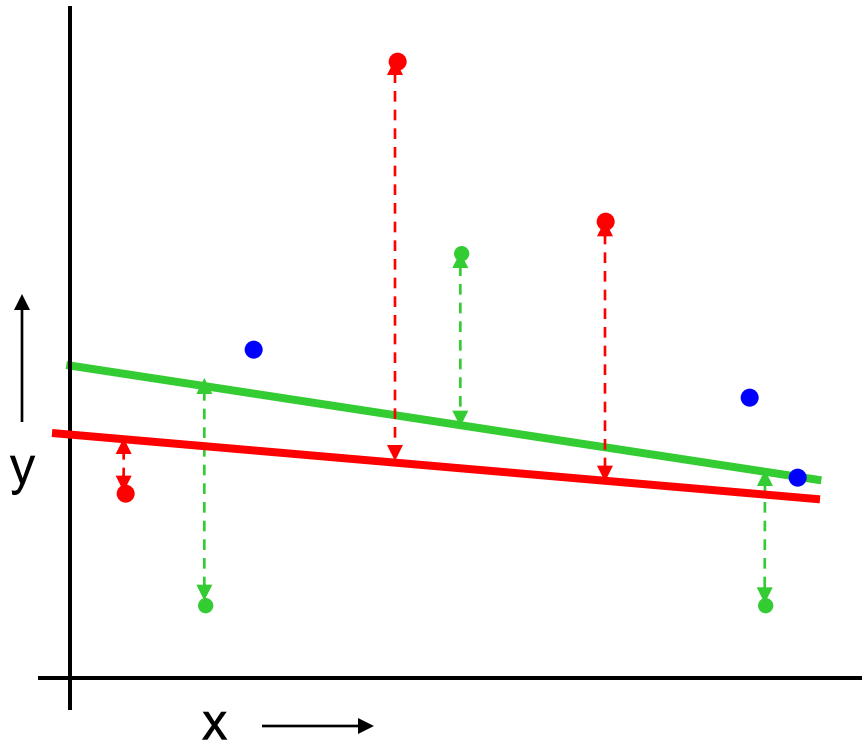
Randomly break the dataset into  $k$  partitions (in our example we'll have  $k=3$  partitions colored Red Green and Blue)



For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.

# k-fold Cross Validation

Randomly break the dataset into  $k$  partitions (in our example we'll have  $k=3$  partitions colored Red Green and Blue)

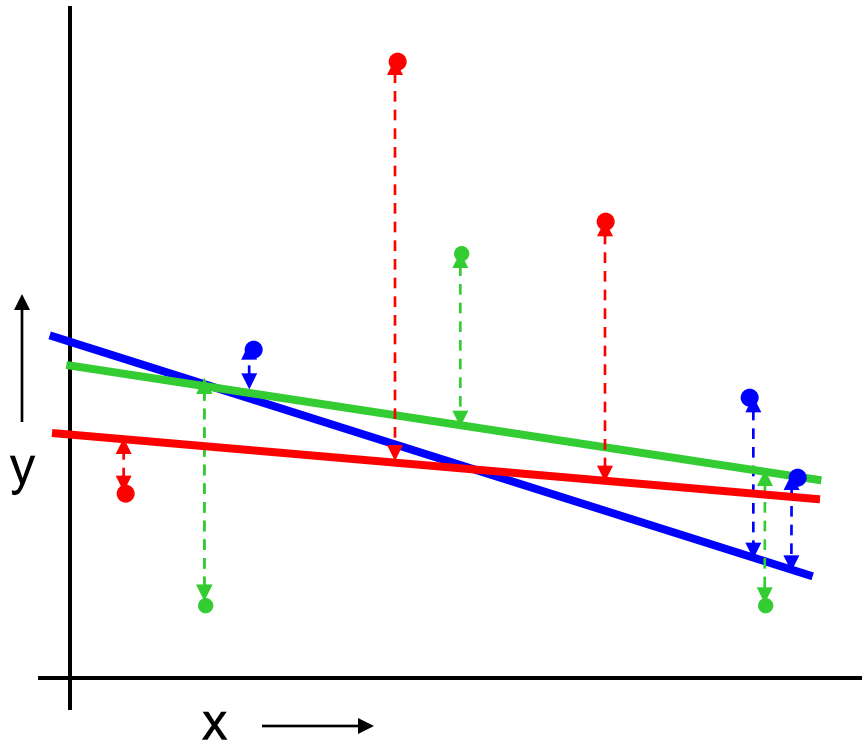


For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.

For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

# k-fold Cross Validation

Randomly break the dataset into  $k$  partitions (in our example we'll have  $k=3$  partitions colored Red Green and Blue)



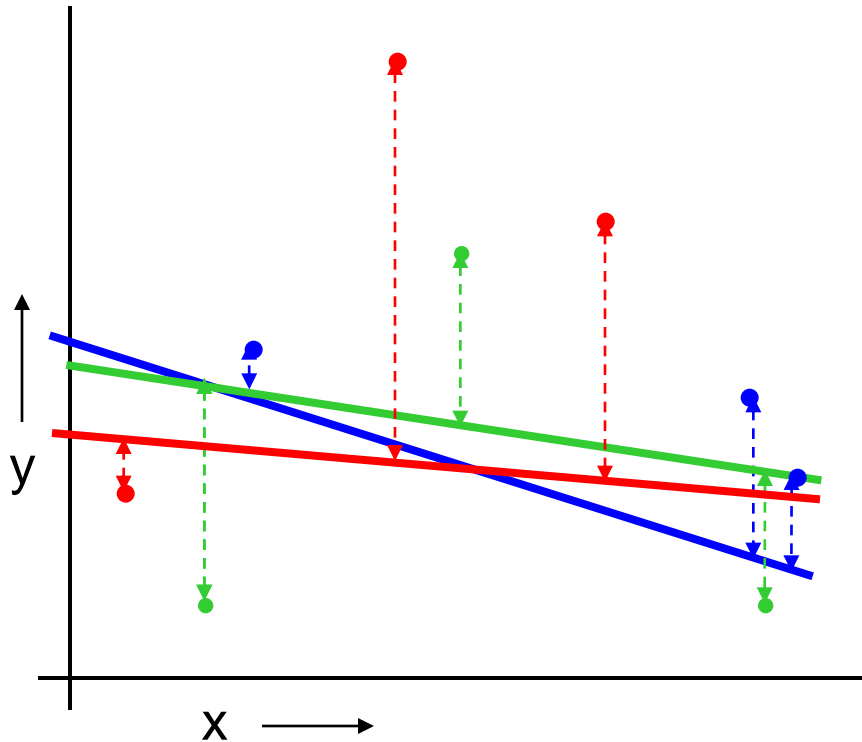
For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.

For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

For the blue partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.

# k-fold Cross Validation

Randomly break the dataset into k partitions (in our example we'll have k=3 partitions colored Red Green and Blue)



Linear Regression  
 $MSE_{3FOLD}=2.05$

For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.

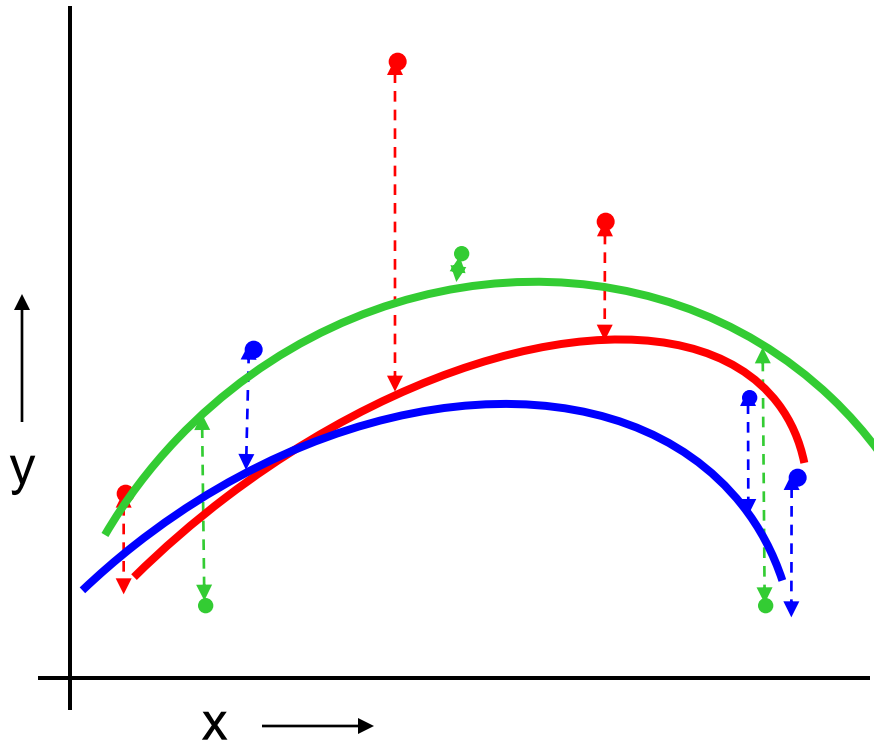
For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

For the blue partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.

Then report the mean error

# k-fold Cross Validation

Randomly break the dataset into k partitions (in our example we'll have k=3 partitions colored Red Green and Blue)



For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.

For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

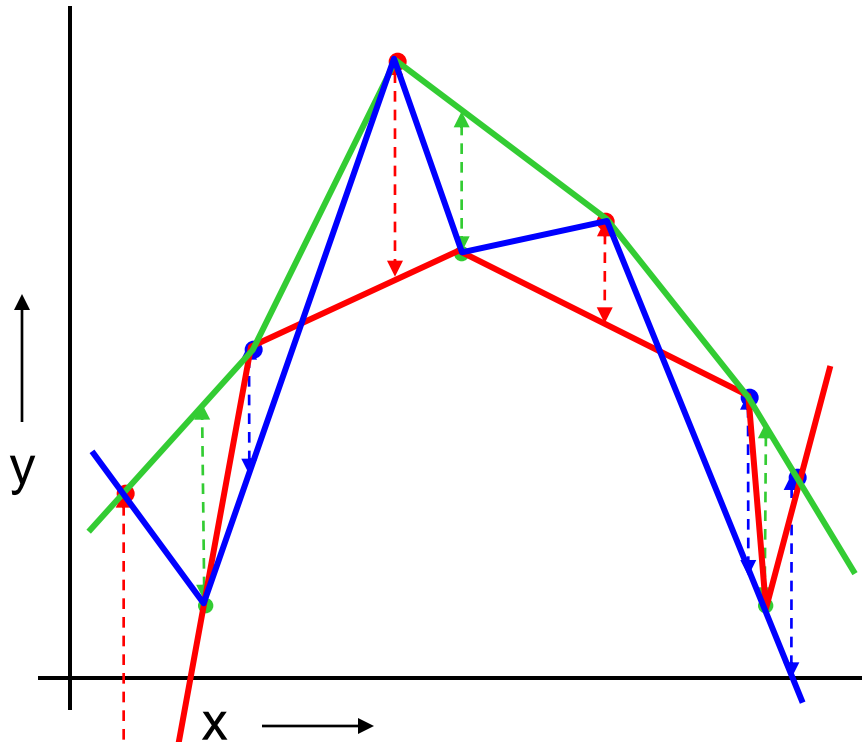
For the blue partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.

Quadratic Regression  
 $MSE_{3FOLD} = 1.11$

Then report the mean error

# k-fold Cross Validation

Randomly break the dataset into k partitions (in our example we'll have k=3 partitions colored Red Green and Blue)



Joint-the-dots  
 $MSE_{3FOLD}=2.93$

For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.

For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

For the blue partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.













Then report the mean error

# Which kind of Cross Validation?

	<b>Downside</b>	<b>Upside</b>
<b>Test-set</b>	Variance: unreliable estimate of future performance	Cheap
<b>Leave-one-out</b>	Expensive. Has some weird behavior	Doesn't waste data
<b>10-fold</b>	Wastes 10% of the data. 10 times more expensive than test set	Only wastes 10%. Only 10 times more expensive instead of R times.
<b>3-fold</b>	Wastier than 10-fold. Expensivier than test set	Slightly better than test-set
<b>R-fold</b>	Identical to Leave-one-out	













# CV-based Model Selection

- We're trying to decide which algorithm to use.
- We train each machine and make a table...

$i$	$f_i$	TRAINERR	10-FOLD-CV-ERR	Choice
1	$f_1$			
2	$f_2$			
3	$f_3$			⊗
4	$f_4$			
5	$f_5$			
6	$f_6$			

# CV-based Model Selection

- Example: Choosing number of hidden units in a one-hidden-layer neural net.
- Step 1: Compute 10-fold CV error for six different model classes:

Algorithm	TRAINERR	10-FOLD-CV-ERR	Choice
<i>0 hidden units</i>			
<i>1 hidden units</i>			
<i>2 hidden units</i>			⊗
<i>3 hidden units</i>			
<i>4 hidden units</i>			
<i>5 hidden units</i>			

- Step 2: Whichever model class gave best CV score: train it with all the data, and that's the predictive model you'll use.

# CV-based Model Selection

- Example: Choosing “k” for a k-nearest-neighbor regression.
- Step 1: Compute LOOCV error for six different model classes:

<i>Algorithm</i>	TRAINERR	10-fold-CV-ERR	Choice
$K=1$			
$K=2$			
$K=3$			
$K=4$			⊗
$K=5$			
$K=6$			

- Step 2: Whichever model class gave best CV score: train it with all the data, and that’s the predictive model you’ll use.

# CV-based Model Selection

- Example: Choosing “k” for a k-nearest-neighbor regression.

- Step 1: Compute LOOCV error for six different classes:

Algorithm	TR
$K=1$	
$K=2$	
$K=3$	
$K=4$	
$K=5$	
$K=6$	

Why did we use 10-fold-CV for neural nets and LOOCV for k-nearest neighbor?

And why stop at  $K=6$

Are we guaranteed that a local optimum of  $K$  vs LOOCV will be the global optimum?

What should we do if we are depressed at the expense of doing LOOCV for  $K=1$  through 1000?

The reason is Computational. For k-NN (and all other nonparametric methods) LOOCV happens to be as cheap as regular predictions.

No good reason, except it looked like things were getting worse as  $K$  was increasing

Sadly, no. And in fact, the relationship can be very bumpy.

Idea One:  $K=1, K=2, K=4, K=8, K=16, K=32, K=64 \dots K=1024$

Idea Two: Hillclimbing from an initial guess at  $K$

- Step 2: Whichever model class gave best CV score: train it with all the data, and that’s the predictive model you’ll use.

# CV-based Model Selection

- Can you think of other decisions we can ask Cross Validation to make for us, based on other machine learning algorithms in the class so far?
  - Degree of polynomial in polynomial regression
  - Whether to use full, diagonal or spherical Gaussians in a Gaussian Bayes Classifier.
  - The Kernel Width in Kernel Regression
  - The Kernel Width in Locally Weighted Regression
  - The Bayesian Prior in Bayesian Regression

These involve choosing the value of a real-valued parameter. What should we do?

Idea One: Consider a discrete set of values (often best to consider a set of values with exponentially increasing gaps, as in the K-NN example).

Idea Two: Compute  $\frac{\partial \text{LOOCV}}{\partial \text{Parameter}}$  and then do gradient descent.

# CV-based Algorithm Choice

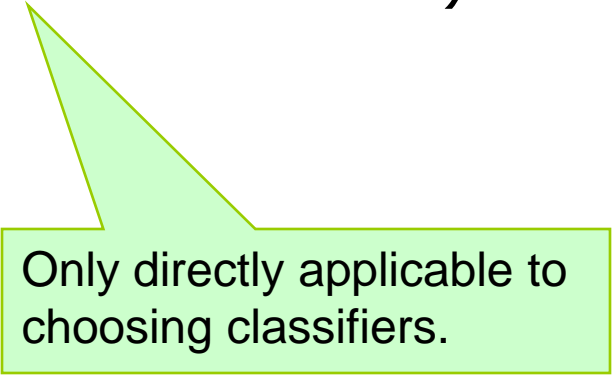
- Example: Choosing which regression algorithm to use
- Step 1: Compute 10-fold-CV error for six different model classes:

<i>Algorithm</i>	TRAINERR	10-fold-CV-ERR	Choice
<i>1-NN</i>			
<i>10-NN</i>			
<i>Linear Reg'n</i>			
<i>Quad reg'n</i>			⊗
<i>LWR, KW=0.1</i>			
<i>LWR, KW=0.5</i>			

- Step 2: Whichever algorithm gave best CV score: train it with all the data, and that's the predictive model you'll use.

# Alternatives to CV-based model selection

- Model selection methods:
  1. Cross-validation
  2. AIC (Akaike Information Criterion)
  3. BIC (Bayesian Information Criterion)
  4. VC-dimension (Vapnik-Chervonenkis Dimension)



Only directly applicable to choosing classifiers.

# Which model selection method is best?

1. (CV) Cross-validation
  2. AIC (Akaike Information Criterion)
  3. BIC (Bayesian Information Criterion)
  4. (SRMVC) Structural Risk Minimize with VC-dimension
- AIC, BIC and SRMVC advantage: you only need the training error.
  - CV error might have more variance
  - SRMVC is wildly conservative
  - Asymptotically AIC and Leave-one-out CV should be the same
  - Asymptotically BIC and carefully chosen k-fold should be same
  - You want BIC if you want the best structure instead of the best predictor (e.g. for clustering or Bayes Net structure finding)
  - Many alternatives---including proper Bayesian approaches.
  - It's an emotional issue.

# Very serious warning

- Intensive use of cross validation can overfit.
- How?
  - Imagine a dataset with 50 records and 1000 attributes.
  - You try 1000 linear regression models, each one using one of the attributes.
  - The best of those 1000 looks good!
  - But you realize it would have looked good even if the output had been purely random!
- What can be done about it?
  - Hold out an additional testset before doing any model selection. Check the best model performs well even on the additional testset.
  - Or: Randomization Testing

# Ensembles: Bayesian model averaging

- MDL picks the MAP hypothesis assuming that the Bayesian prior will give more probability to simple hypotheses
- In general, the priors can encode domain knowledge, and the simplest hypotheses will not necessarily be the most likely under the prior.
- In this case, picking the max a posteriori hypothesis is called Bayesian model selection.
- However, in the Bayesian framework, we can do more: instead of picking one hypothesis, take all hypotheses and let them vote, with a weight proportional to their posterior probability. This approach is called model averaging.

# Ensembles

- Bayesian Model Averaging. Sample hypotheses  $h_i$  according to their posterior probability  $P(h|S)$ . Vote them. A method called Markov chain Monte Carlo (MCMC) can do this (but it is quite expensive)
- Bagging. Overfitting is caused by high variance. Variance reduction methods such as bagging can help. Indeed, best results are often obtained by bagging overfitted classifiers (e.g., unpruned decision trees, over-trained neural networks) than by bagging well-fitted classifiers (e.g., pruned trees).

# Ensembles (2)

- Randomized Committees. We can train several hypotheses  $h_i$  using different random starting weights for backpropagation.
- Random Forests. Grow many decision trees and vote them. When growing each tree, randomly (at each node) choose a subset of the available features (e.g.,  $\sqrt{n}$  out of  $n$  features). Compute the best split using only those features.

# Overfitting Summary

- Minimizing training set error ( $\varepsilon_{\text{train}}$ ) does not necessarily minimize test set error ( $\varepsilon_{\text{test}}$ ).
  - This is especially true when the hypothesis space is too large (too expressive)
- Penalty methods add a penalty to  $\varepsilon_{\text{train}}$  to approximate  $\varepsilon_{\text{test}}$ 
  - Bayesian, MDL, and Structural Risk Minimization
- Holdout and Cross-Validation methods without a subset of the training data,  $S_{\text{eval}}$ , to determine the proper hypothesis space  $H_i$  and its complexity.
- Ensemble Methods take a combination of several hypotheses, which tends to cancel out overfitting errors.