Introduction to Machine Learning

CS307 --- Fall 2022

Instance-Based Learning

Reading:
Sections 18.3, 20.4, R&N
Sections 8.1-8.3, Mitchell

Instance-Based Learning (IBL)

Unlike most learning algorithms, case-based, also called exemplar-based or instance-based, approaches do not construct an abstract hypothesis but instead base classification of test instances on similarity to specific training instances.

Training is typically very simple: Just store the training instances

Generalization is postponed until a new instance must be classified. Therefore, case-based methods are sometimes called “lazy” learners.

Given a new instance, its relationship to the stored examples is examined in order to assign a target function value for the new instance.

Distance Metrics

Case-based methods assume a function for calculating the (dis)similarity of two instances.

- similarity metric: computes similarity between instances
- distance metric: computes dissimilarity between instances

For continuous feature vectors, just use Euclidean distance

$$Dist(c_1, c_2) = \sqrt{\sum_{i=1}^{N} \left(attr_i(c_1) - attr_i(c_2)\right)^2}$$

For discrete features, just assume distance between two values is 0 if they are the same, 1 if difference

To compensate for differences in units, scale all continuous values to normalize their values to be between 0 and 1.

1-Nearest Neighbor

$$Dist(c_1, c_2) = \sqrt{\sum_{i=1}^{N} \left(attr_i(c_1) - attr_i(c_2)\right)^2}$$

$$NearestNeighbor = MIN_j(Dist(c_j, c_{test}))$$

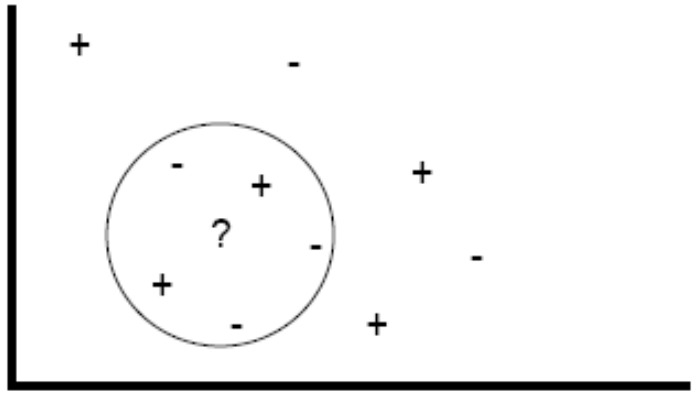
$$prediction_{test} = class_j(or\ \ value_j)$$

works well if no attribute or class noise

k-Nearest Neighbor

Calculate the distance between a test instance and every training instance

Pick the *k* closest training examples and assign the test example to the most common category or the average value among these “nearest” neighbors”



Voting multiple neighbors helps increase resistance to noise. For binary classification tasks, odd values of *k* are normally used to avoid ties.

Effect of "k"

Large *k*:

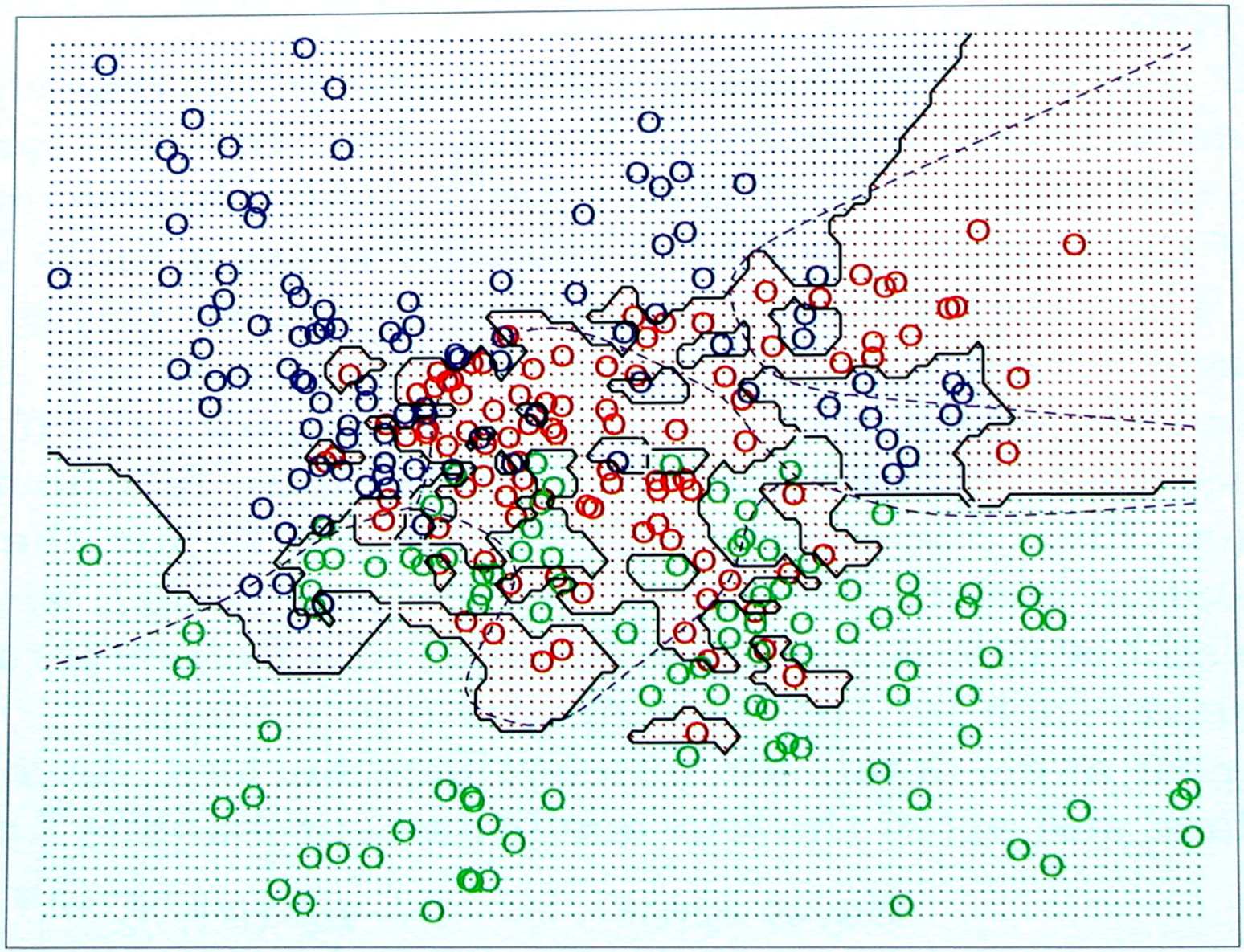
- less sensitive to noise (particularly class noise)
- better probability estimates for discrete classes
- larger training sets allow larger values of *k*

Small *k*:

- captures fine structure of space better
- may be necessary with small training sets

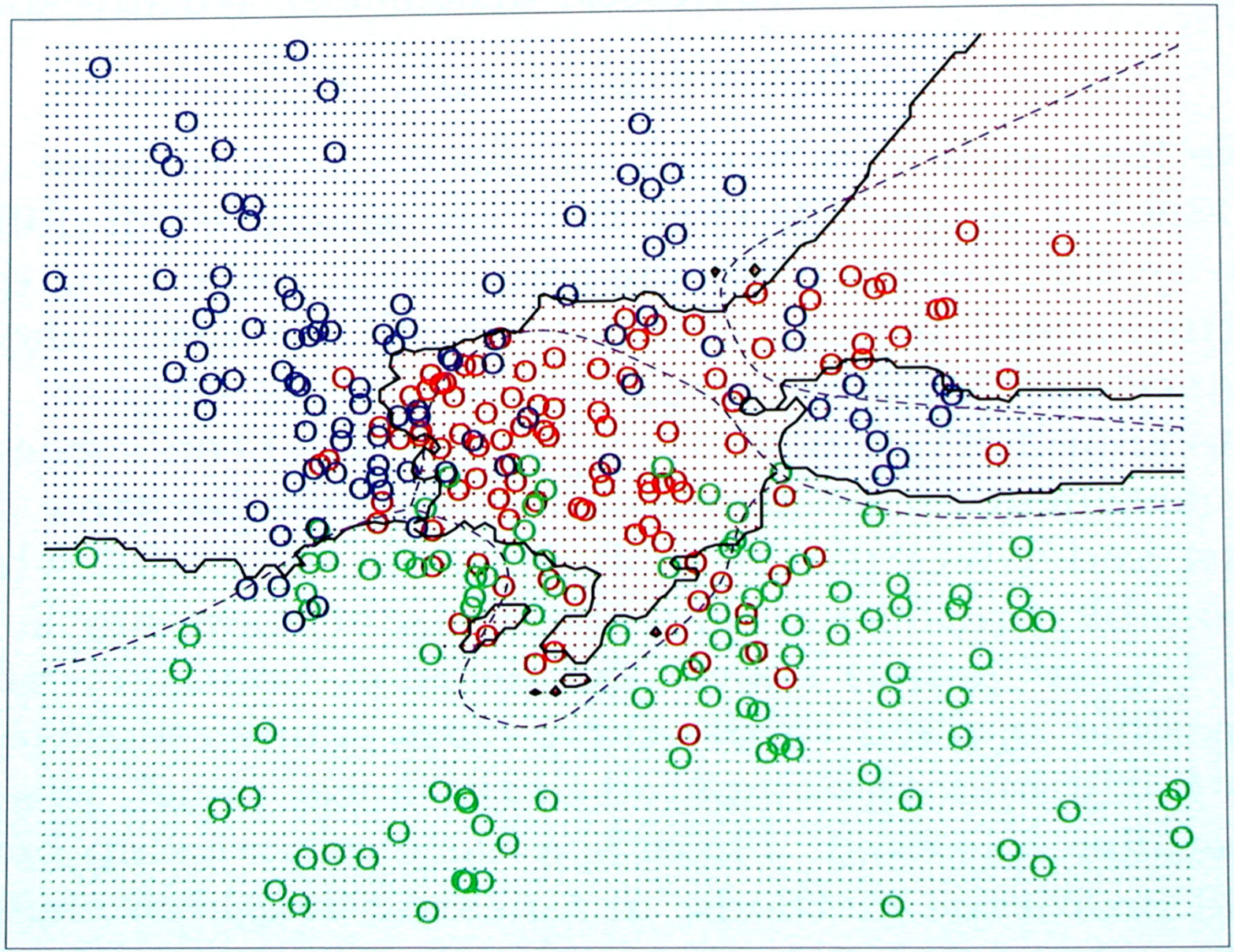
Balance must be struck between large and small *k*

1-Nearest Neighbor



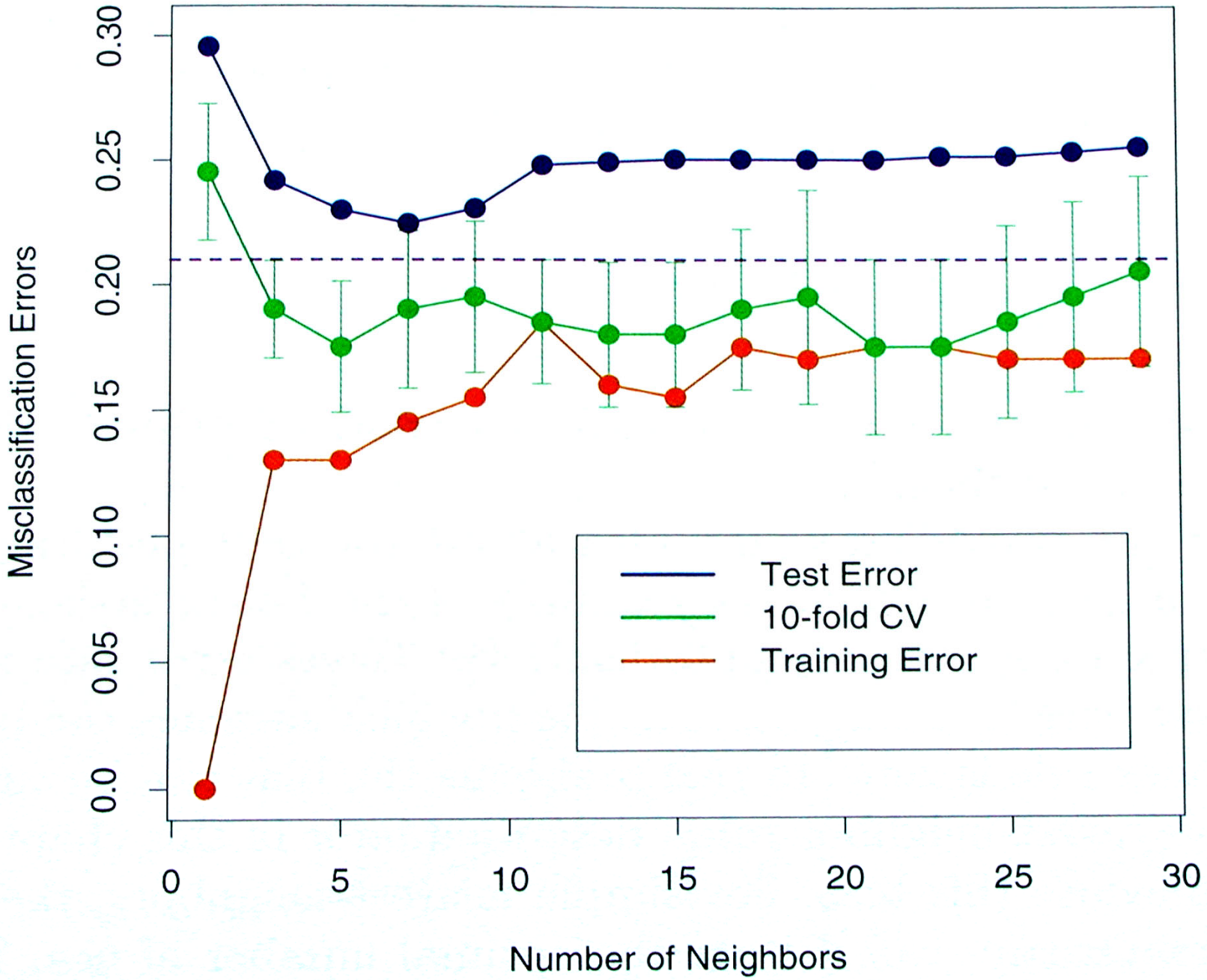
From Hastie, Tibshirani, Friedman 2001 p418

15-Nearest Neighbors



From Hastie, Tibshirani, Friedman 2001 p418

From Hastie, Tibshirani, Friedman 2001 p419



Distance-Weighted kNN

tradeoff between small and large *k* can be difficult

use large *k*, but more emphasis on nearer neighbors?

$$prediction_{test} = \frac{\sum_{i=1}^{k} w_i * class_i}{\sum_{i=1}^{k} w_i} \left(or \frac{\sum_{i=1}^{k} w_i * value_i}{\sum_{i=1}^{k} w_i} \right)$$

$$w_k = \frac{1}{Dist(c_k, c_{test})}$$

Locally Weighted Averaging

Let *m* = number of training points

Let weight fall-off rapidly with distance

$$prediction_{test} = \frac{\sum_{i=1}^{m} w_i * class_i}{\sum_{i=1}^{m} w_i} \left(or \frac{\sum_{i=1}^{m} w_i * value_i}{\sum_{i=1}^{m} w_i} \right)$$

$$w_i = \frac{1}{e^{KernelWidth \cdot Dist(c_i, c_{test})}}$$

KernelWidth controls size of neighborhood that has large effect on value (analogous to *k*)

Locally Weighted Regression

All algorithms so far are strict averagers

We may do weighted regression

- for each test point
  - train a regressor using its nearest neighbors as training examples, where each neighbor is weighted by its distance from test point
  - predict using the local regressor

Local regressor can be linear, quadratic, n-th degree polynomial, neural net, …

Curse of Dimensionality

As number of dimensions increases, distance between points becomes larger

If number of relevant attributes is fixed, increasing the number of less relevant attributes may swamp distance

$$D(c1,c2)=\sqrt{\sum_{i=1}^{relevant}\left(attr_i(c1)-attr_i(c2)\right)^2+\sum_{j=1}^{irrelevant}\left(attr_j(c1)-attr_j(c2)\right)^2}$$

When more irrelevant dimensions relevant dimensions, distance becomes less reliable

How can we deal with the curse of dimensionality?

Determining Feature Relevance

Use feature selection or feature weighting

Wrapper methods for feature selection generate a set of candidate features, run the induction algorithm with these features, use the accuracy of the resulting concept description to evaluate the feature set.

Filter methods for feature selection consider attributes independently of the induction algorithm that will use them.

Wrapper Methods for Feature Selection

Generate a set of candidate features, run the induction algorithm with these features, use the accuracy of the resulting concept description to evaluate the feature set.

Forward selection. Start with no features and successively add attributes. Continue until performance degrades.

Backward elimination. Start with all features and successfully remove attributes. Continue until performance degrades.

General method for feature selection in that it can be used in conjunction with any induction algorithm.

Disadvantage: Computational cost.

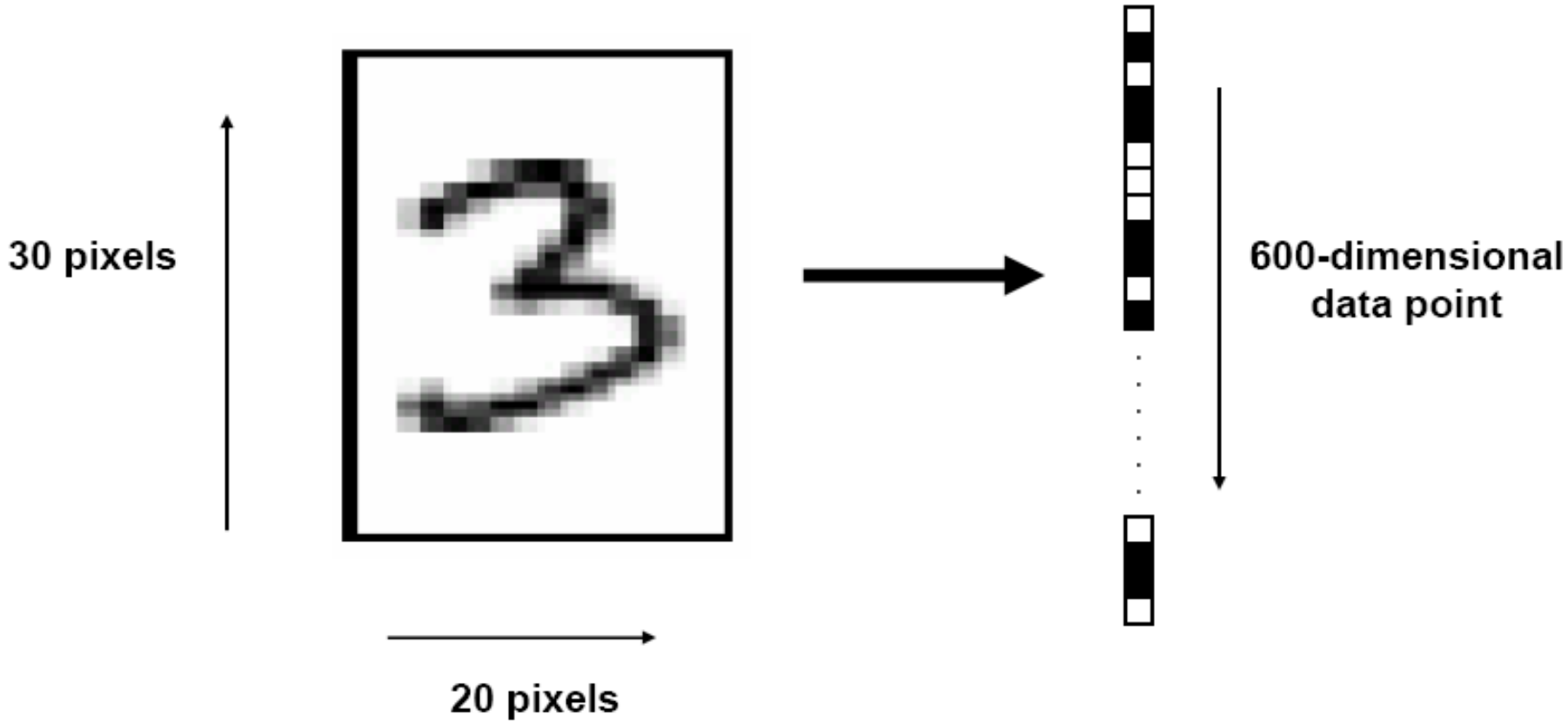
Filter Methods for Feature Selection

Filter methods consider attributes independently of the induction algorithm that will use them.

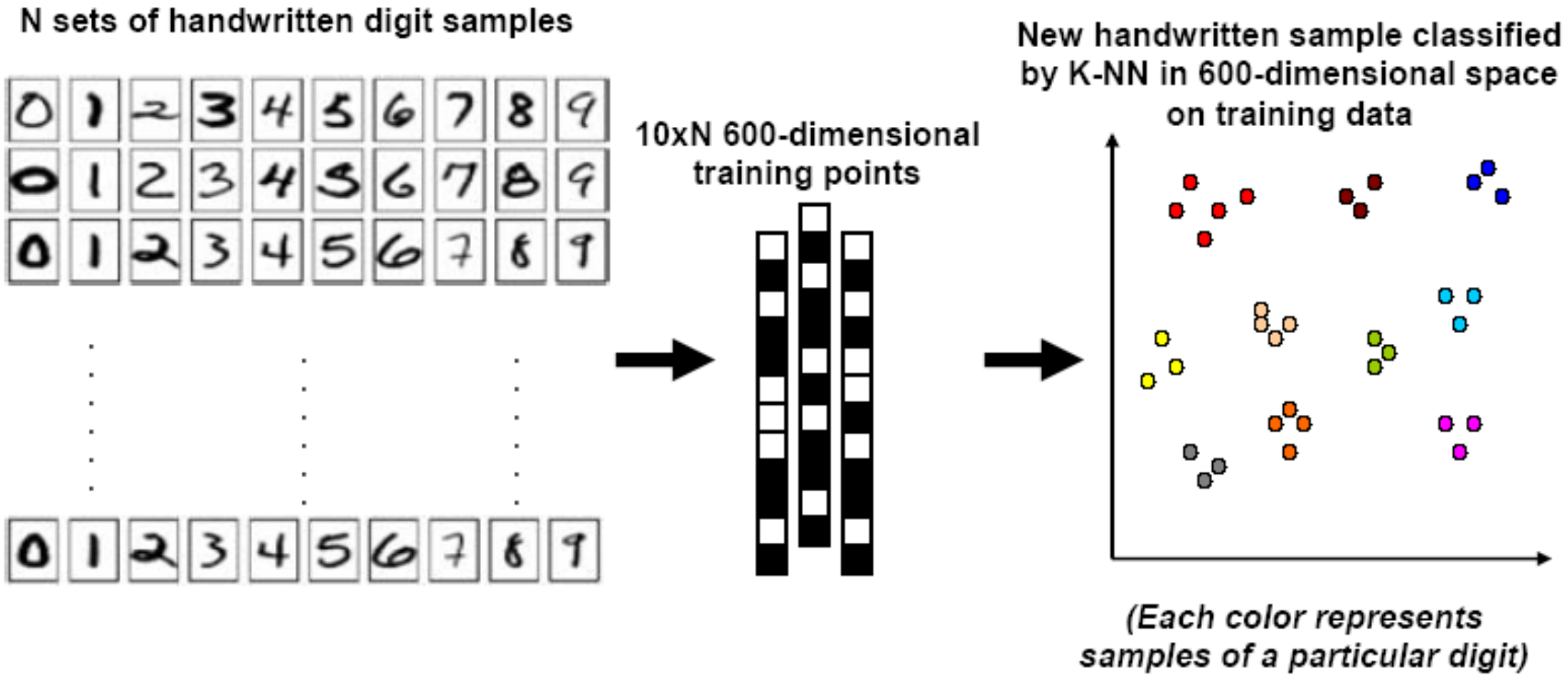
E.g., decision trees for feature selection

In addition to storing training cases in a case base, use them to induce a decision tree. Features that do not appear in the decision trees are considered irrelevant for the learning task and can be discarded from the instance representation for the case-based learning system.

Example: Recognition of Handwritten Digits



Example: Recognition of Handwritten Digits



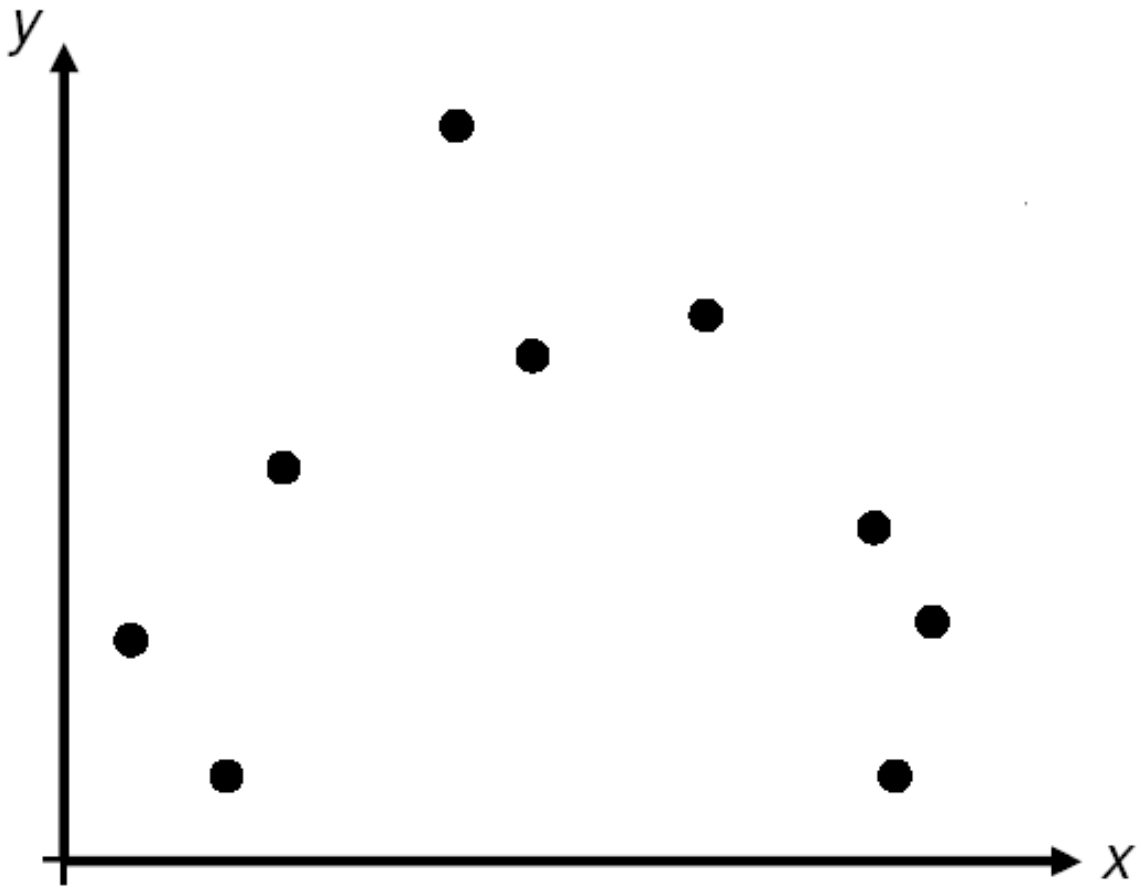
K-NN vs. Other Techniques

- Instance-based methods do not need a training phase, unlike decision trees and Bayes classifiers
- However, the nearest-neighbors-search step can be expensive for large/high-dimensional datasets
- No foolproof way to pre-select **K** … but can be selected using cross-validation

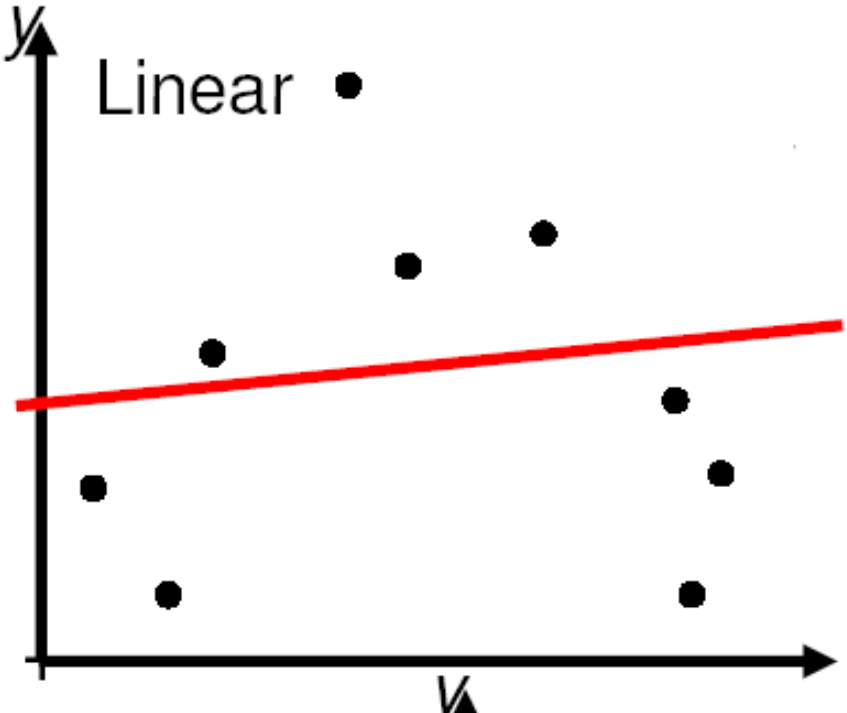
Avoiding Overfitting

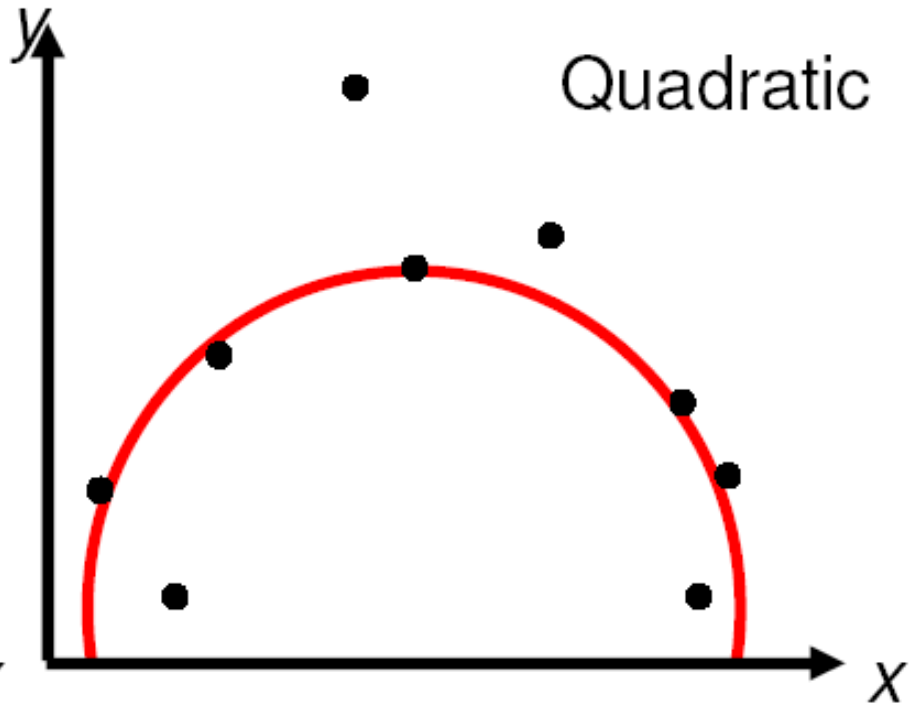
- We have a choice of different techniques:
- Decision trees, Neural Networks, Nearest Neighbors, Bayes Classifier,…
- For each we have different levels of complexity:
  - Depth of trees
  - Number of layers and hidden units
  - Number of neighbors in K-NN
  - …..
- How to choose the right one?
- Overfitting: A complex enough model (e.g., enough units in a neural network, large enough trees,..) will *always* be able to fit the training data well

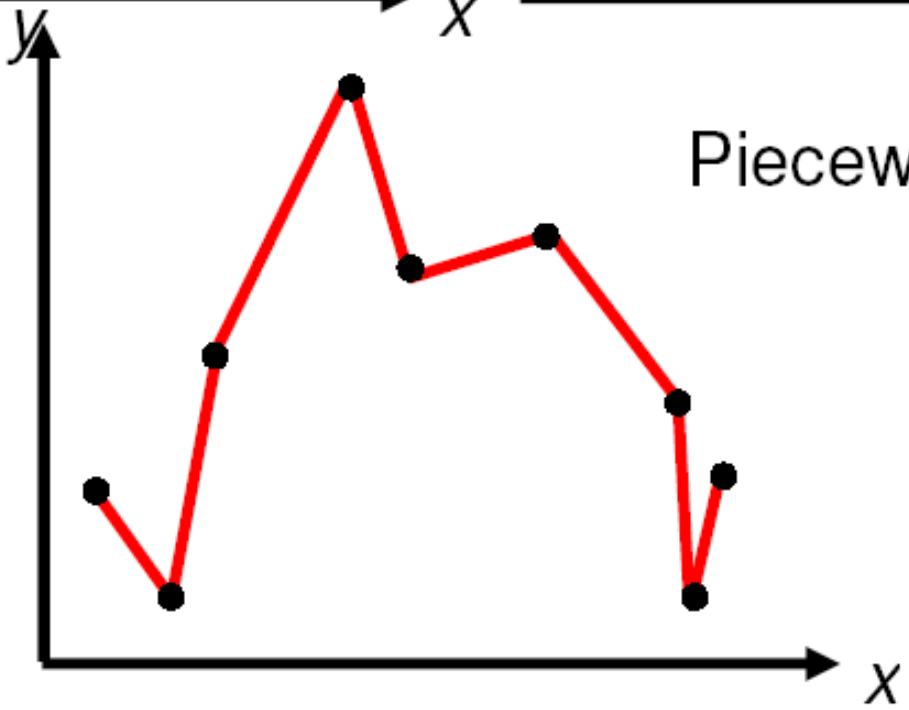
Example



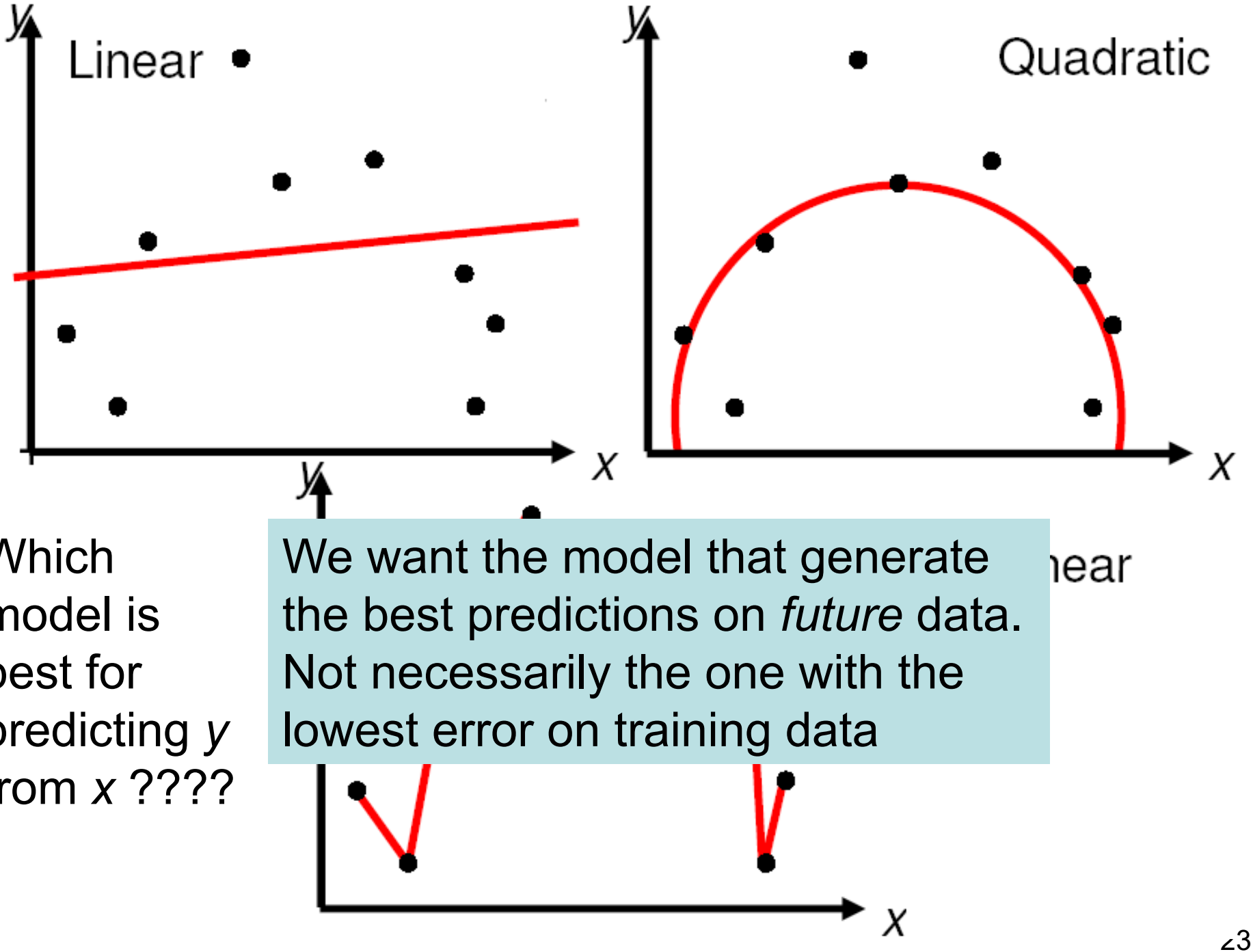
- Construct a predictor of *y* from *x* given this training data







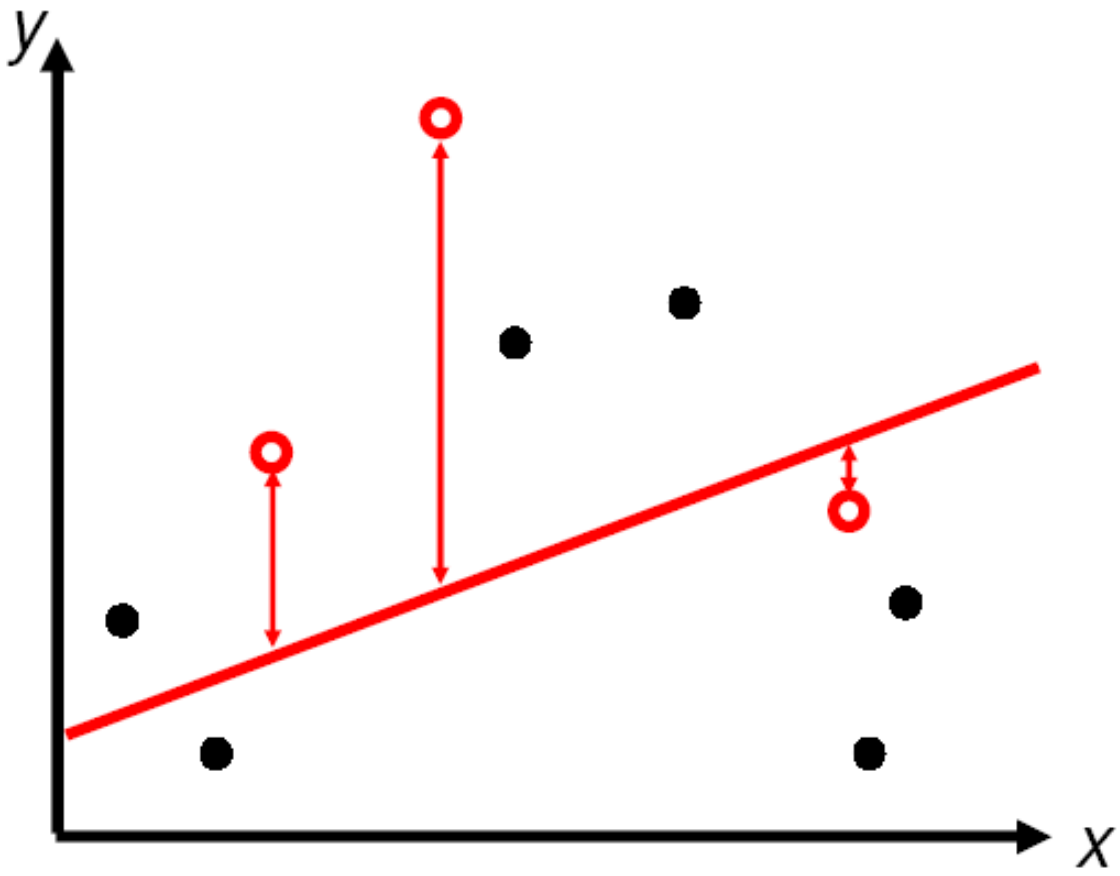
Which model is best for predicting *y* from *x* ????



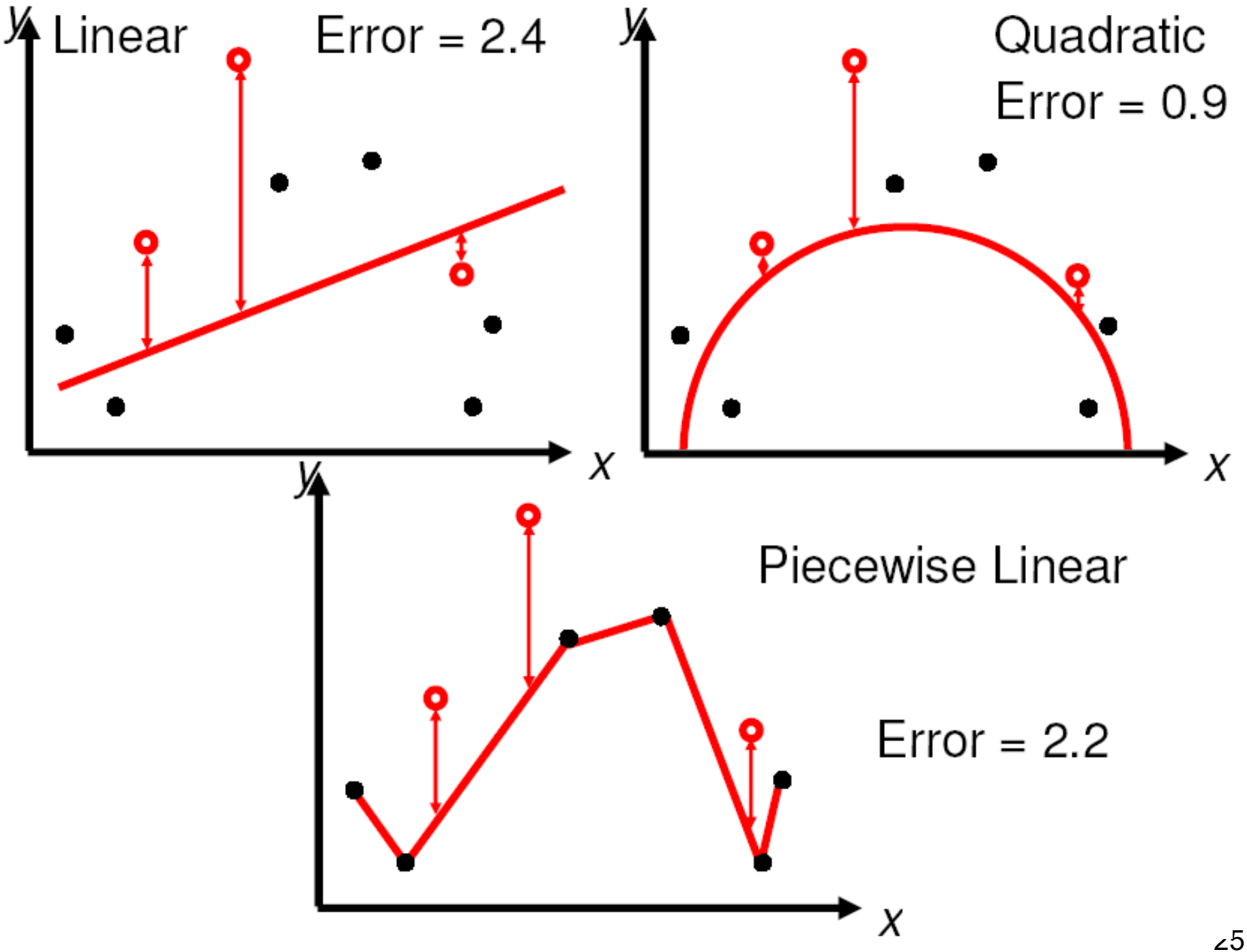
Which model is best for predicting *y* from *x* ????

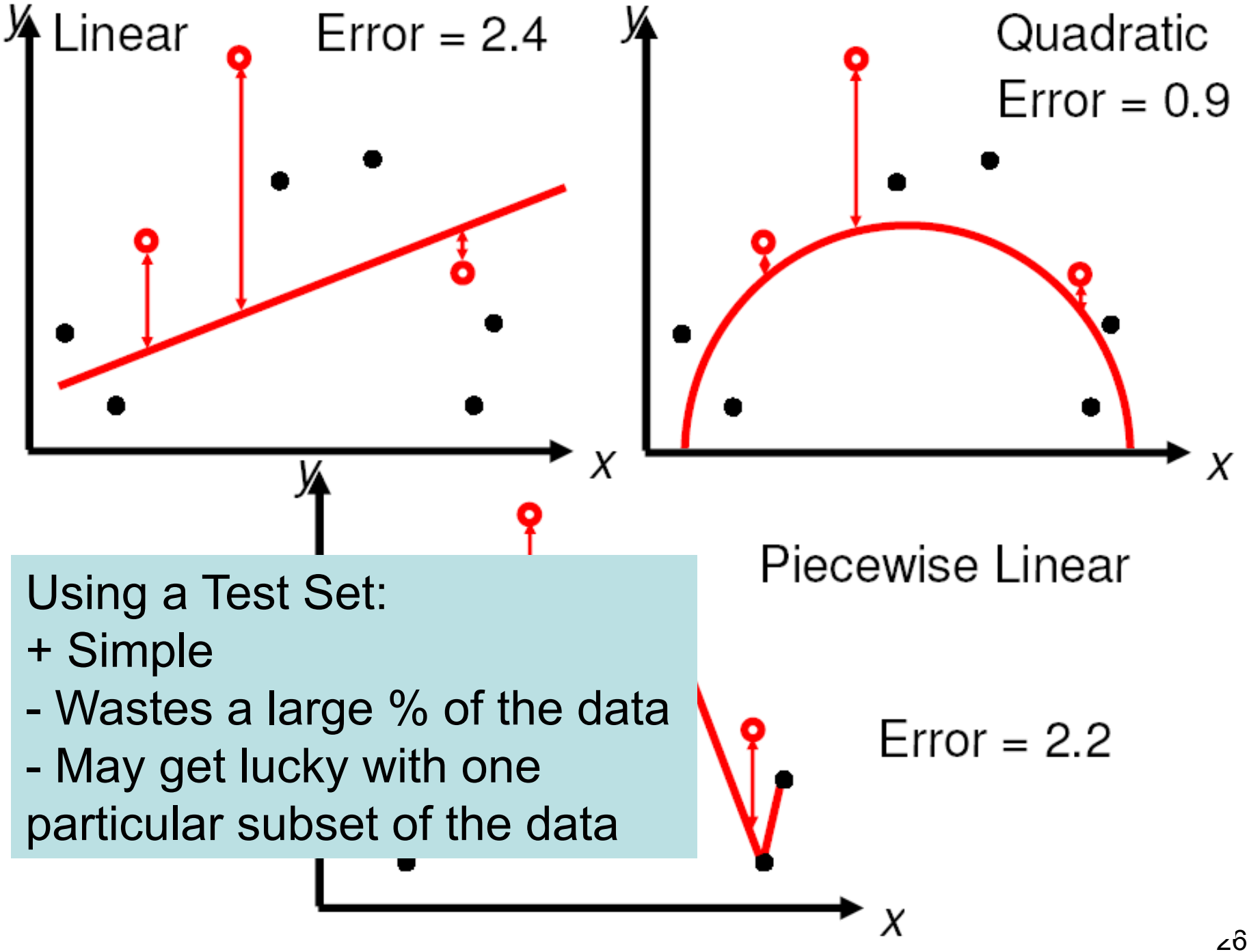
We want the model that generate the best predictions on *future* data. Not necessarily the one with the lowest error on training data

Using a Test Set



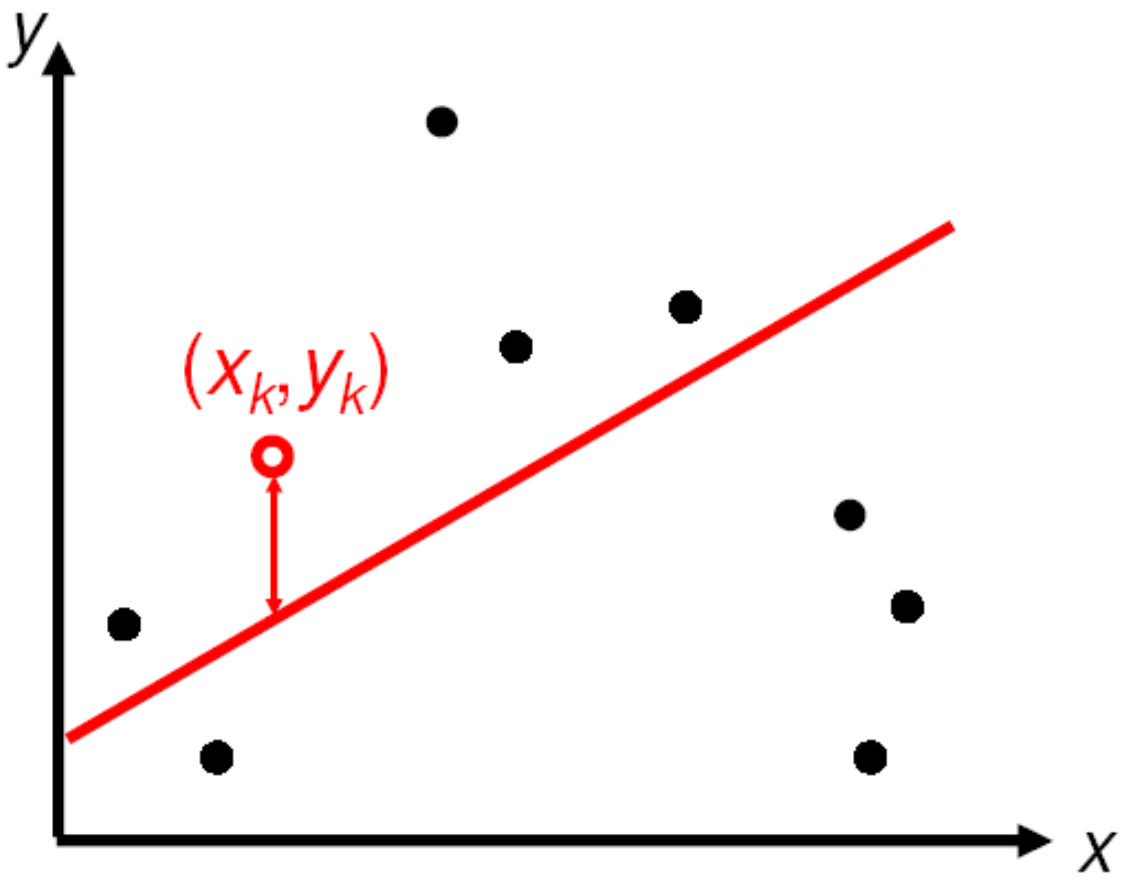
1. Use a portion (e.g., 30%) of the data as *test data*
2. Fit a model to the remaining training data
3. Evaluate the error on the *test data*


y
Linear
Error = 2.4
x
y
Quadratic
Error = 0.9
x
y
Piecewise Linear
Error = 2.2
x

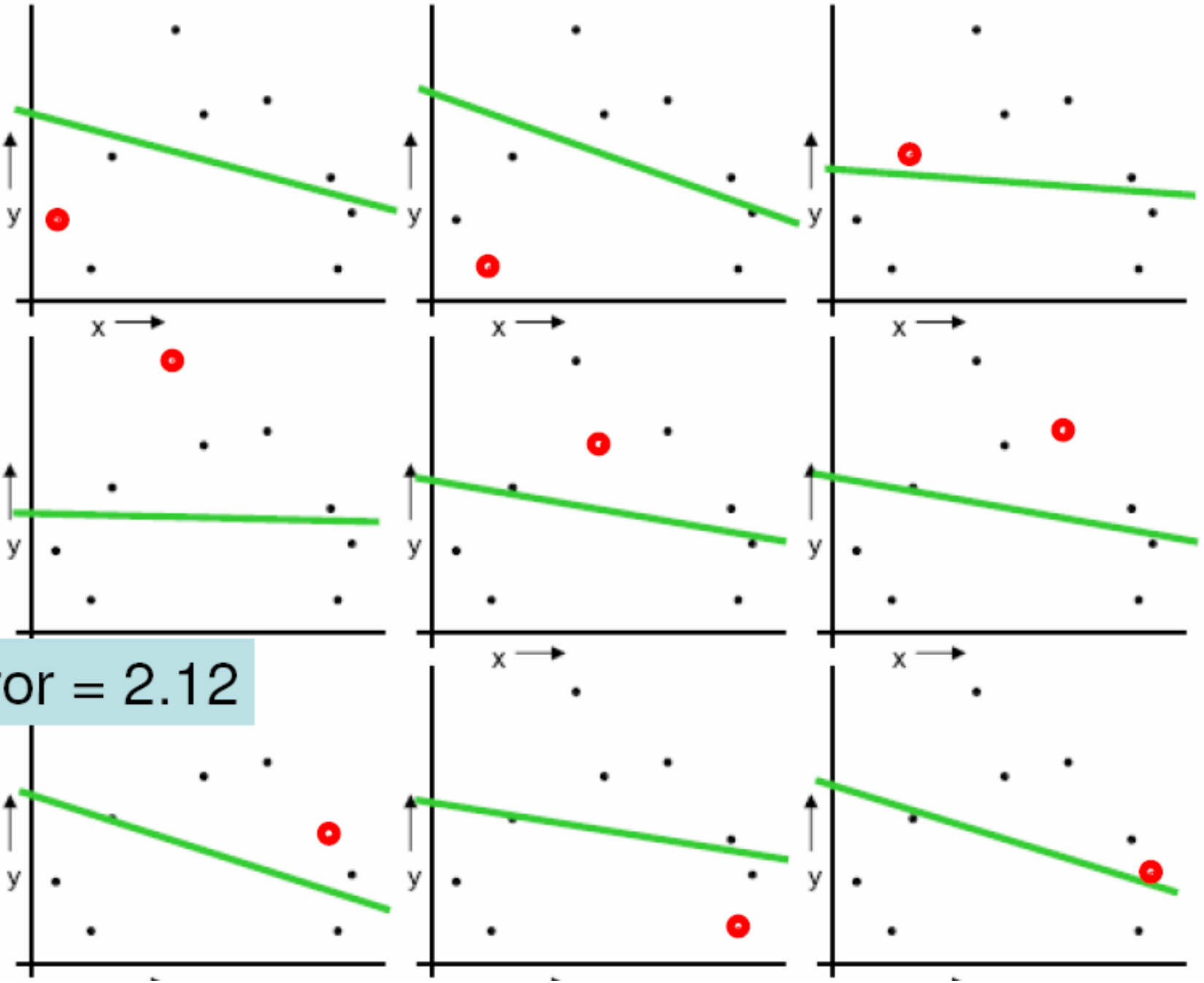


Using a Test Set:
+ Simple
- Wastes a large % of the data
- May get lucky with one particular subset of the data

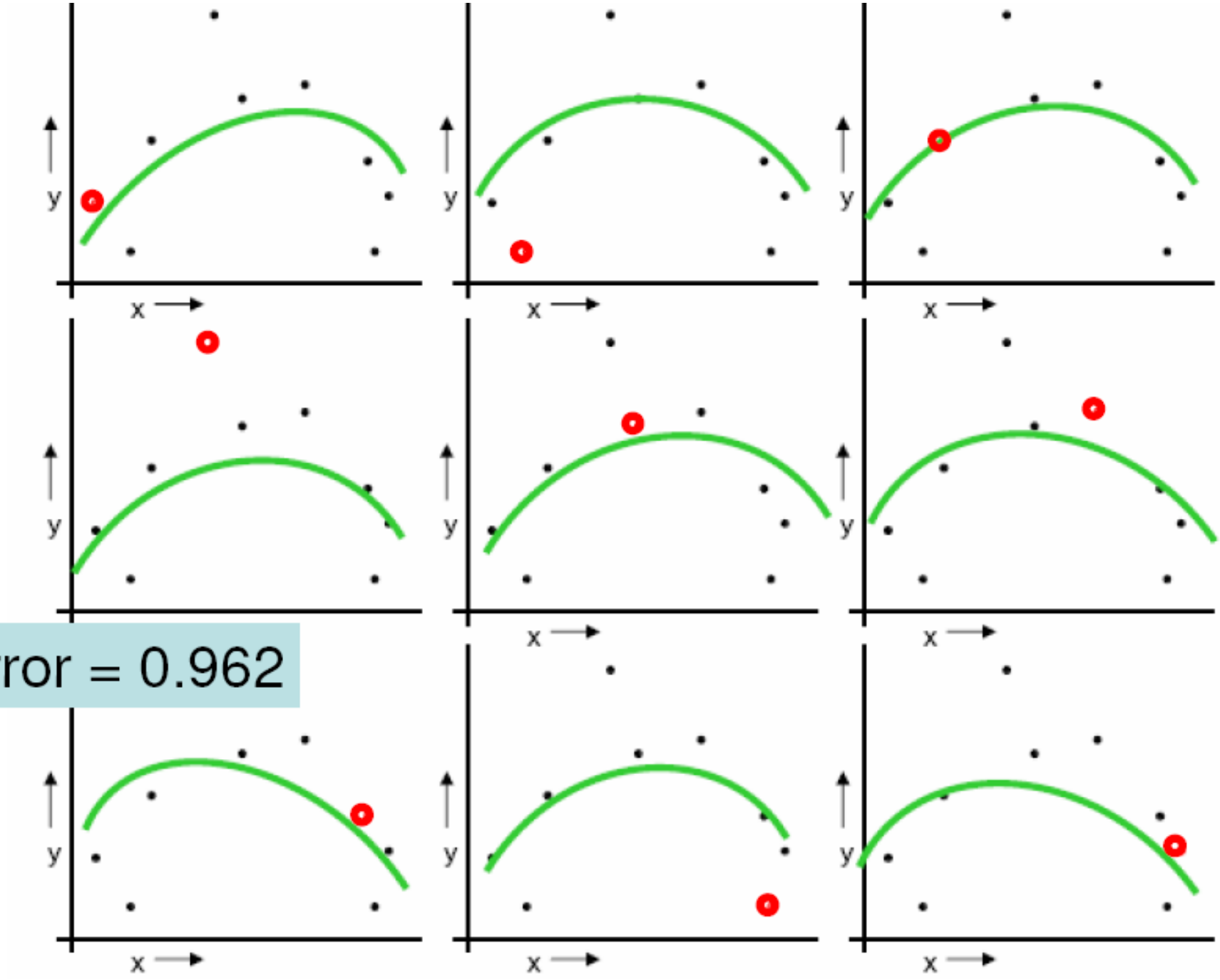
“Leave One Out” Cross-Validation



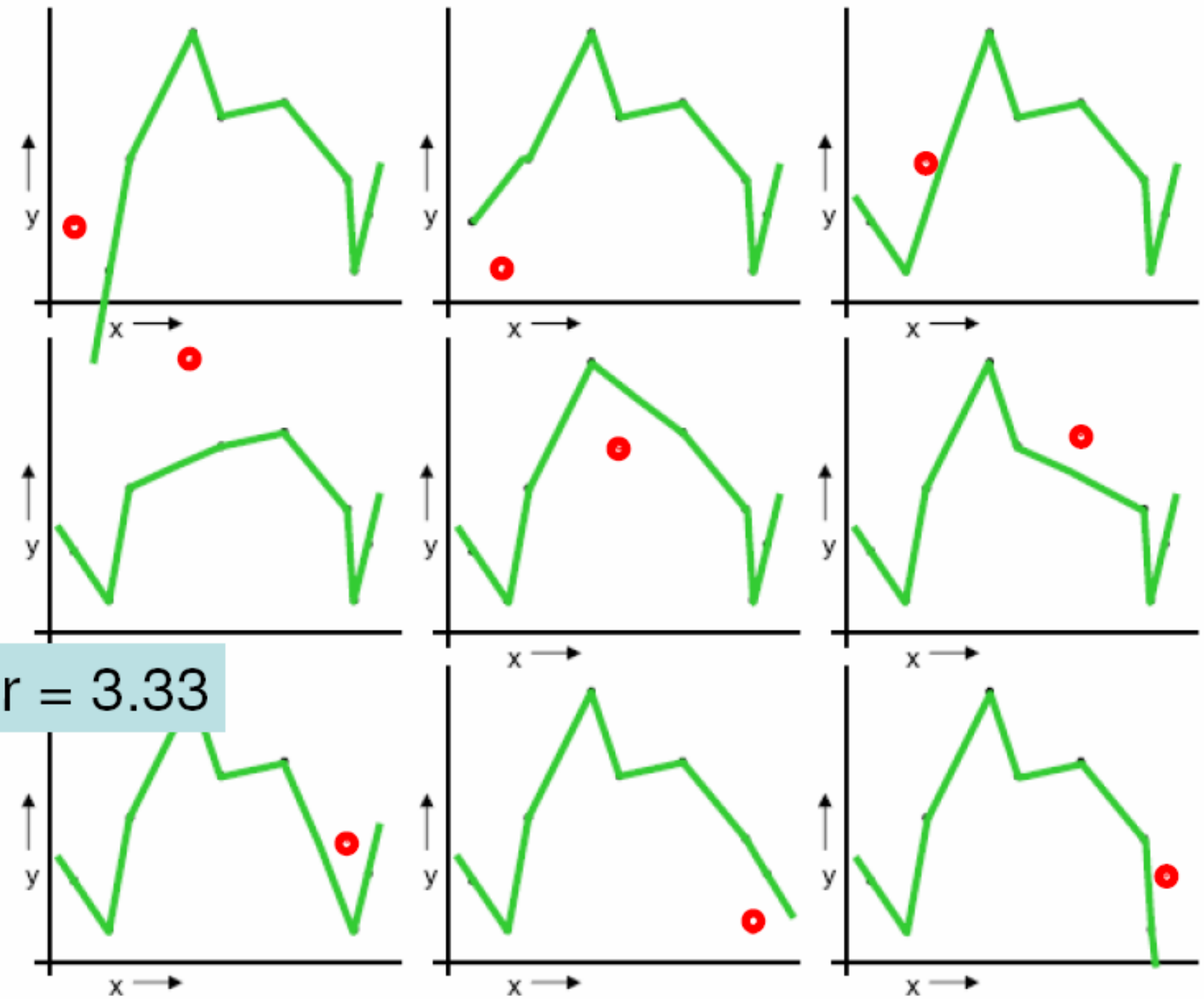
- For k=1 to R
  - Train on all the data leaving out (x_k,y_k)
  - Evaluate error on (x_k,y_k)
- Report the average error after trying *all* the data points



Error = 2.12



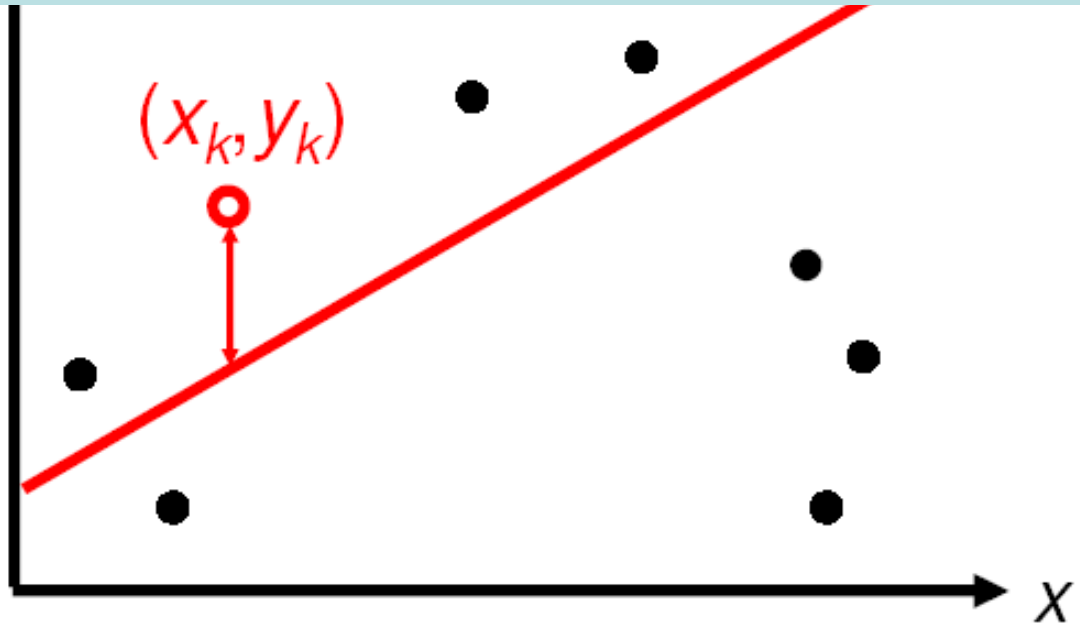
Error = 0.962



Error = 3.33

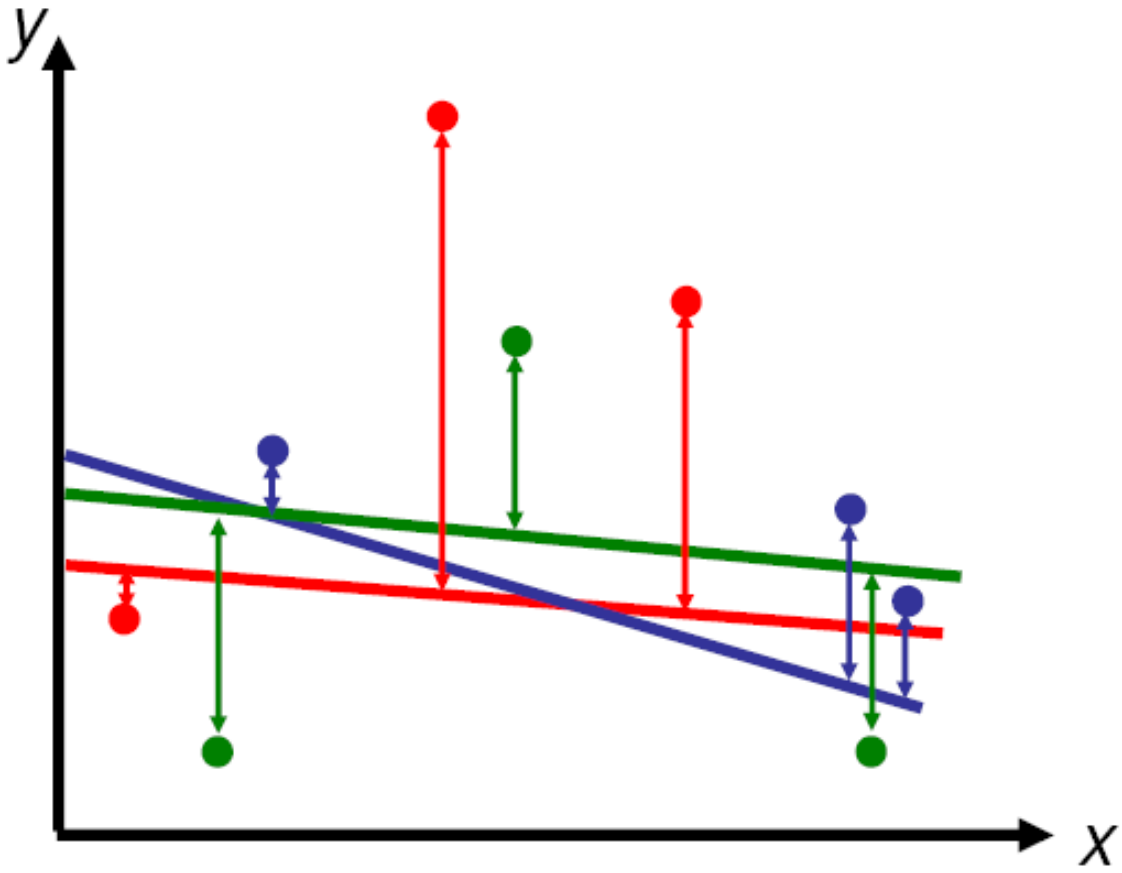
“Leave One Out” Cross-Validation

“Leave One Out” Cross-Validation:
+ Does not waste data
+ Average over large number of trials
- Expensive



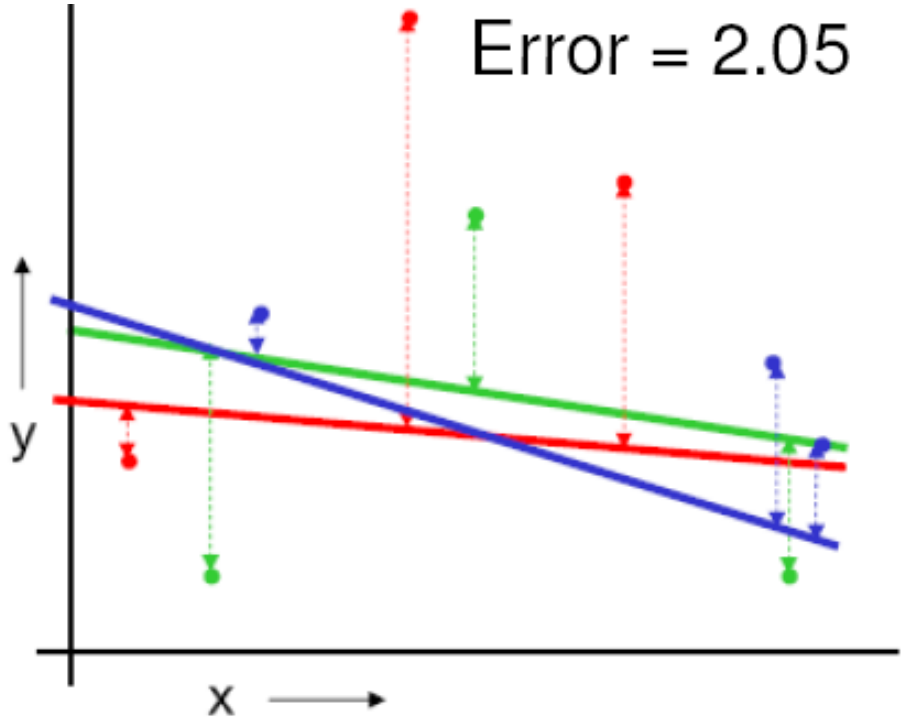
- For k=1 to R
  - Train on all the data leaving out (x_k,y_k)
  - Evaluate error on (x_k,y_k)
- Report the average error after trying *all* the data points

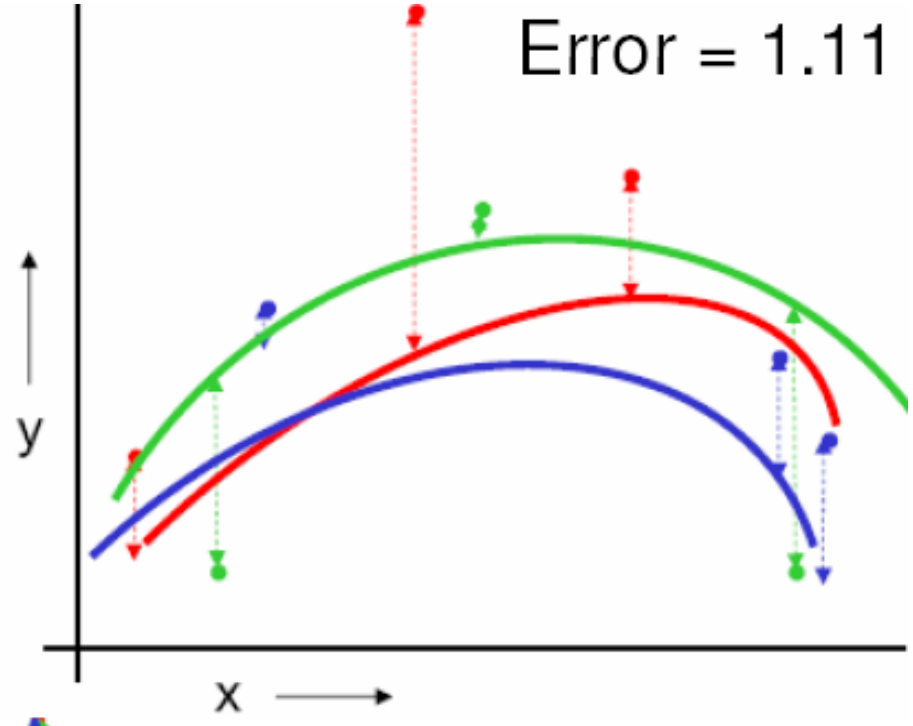
K-Fold Cross-Validation

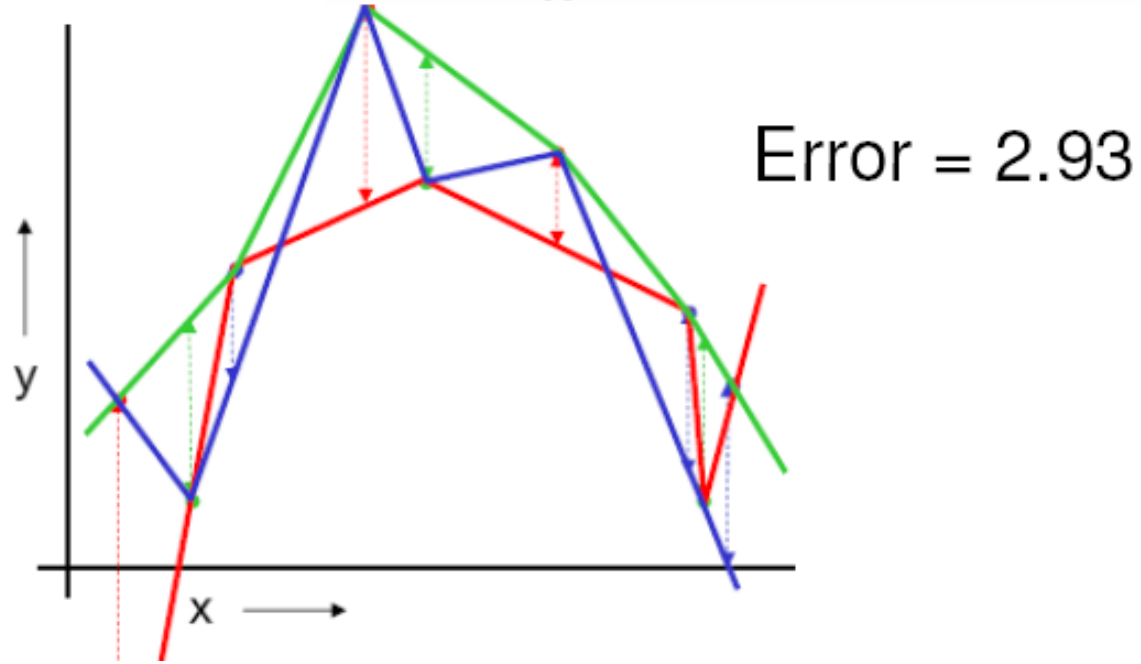


- Randomly divide the data set into *K* subsets
- For each subset *S*:
  - Train on the data *not in S*
  - Test on the data *in S*
- Return the average error over the *K* subsets

Example: *K* = 3, each color corresponds to a subset


Error = 2.05
y
x

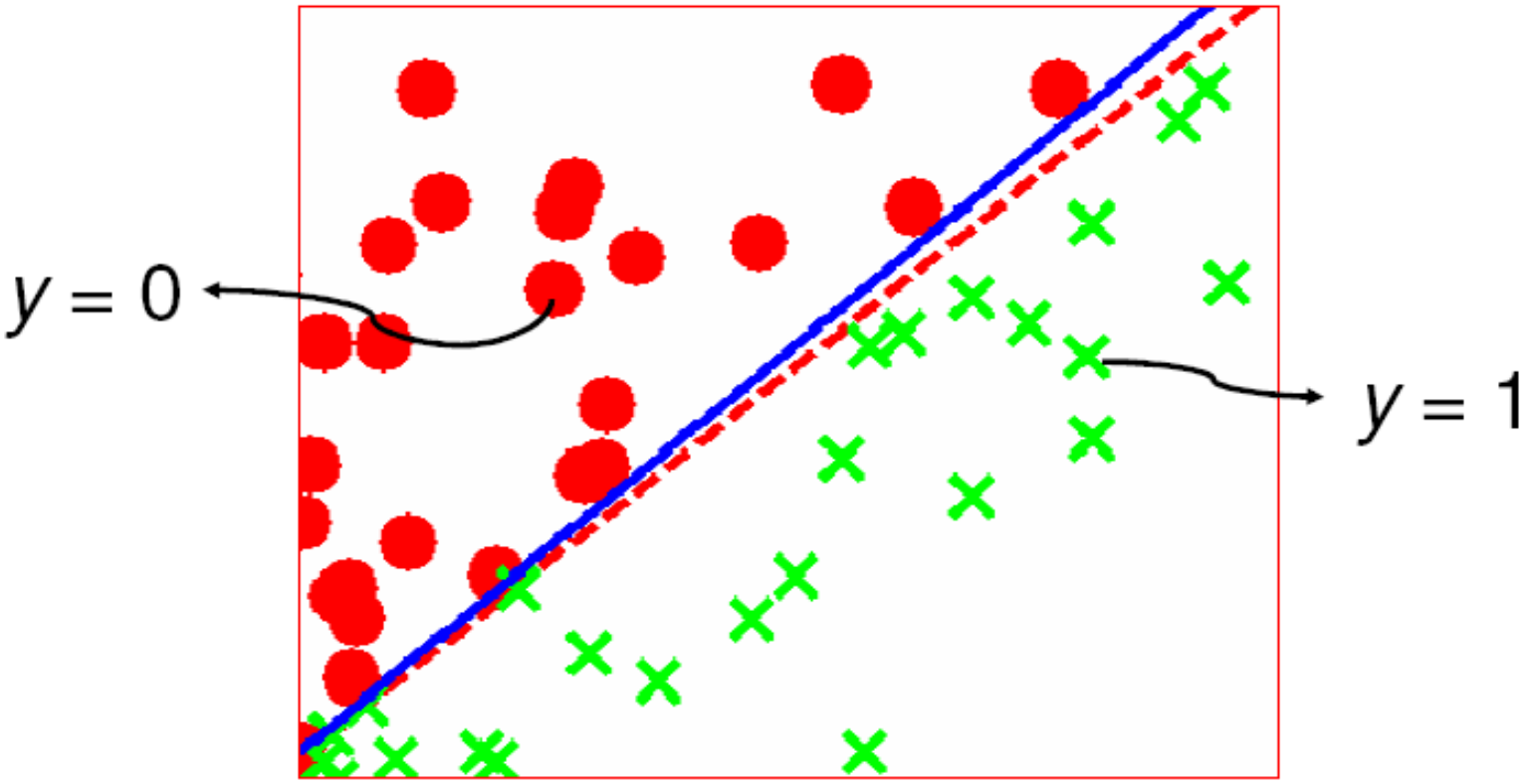

Error = 1.11
y
x


Error = 2.93
y
x

Cross-Validation Summary

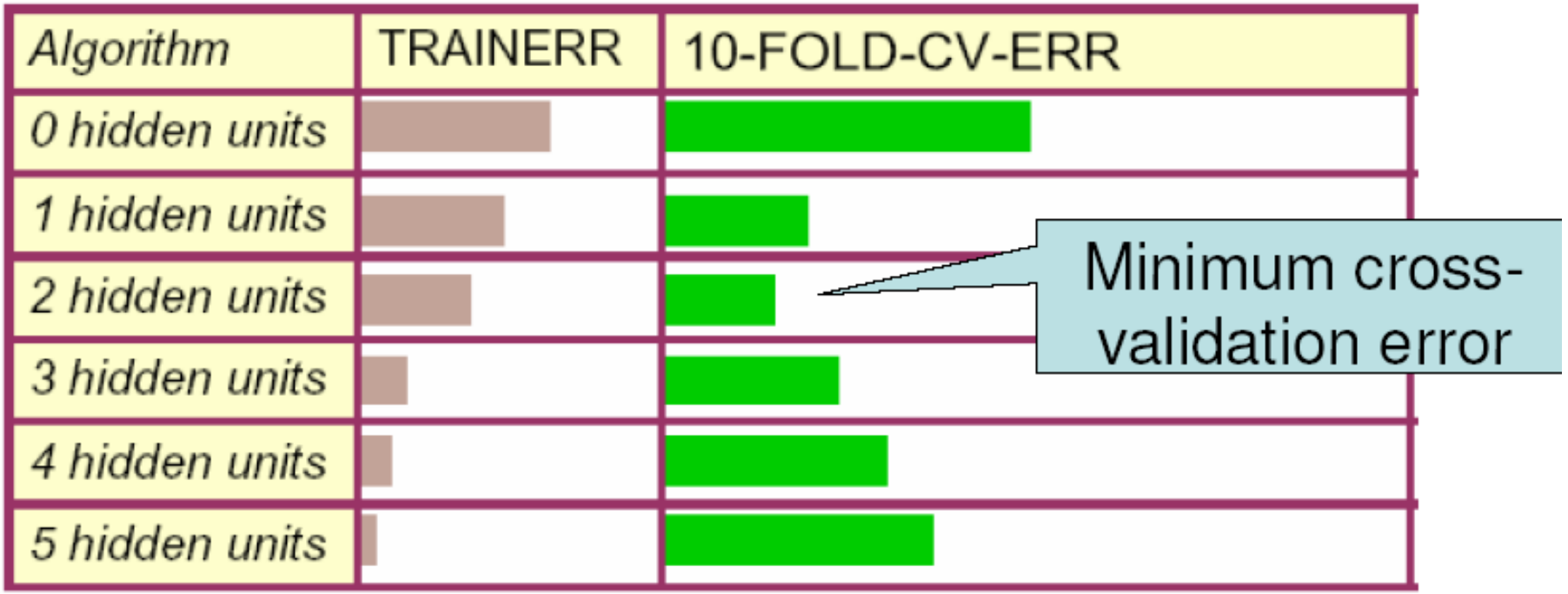
	-	+
Test Set	Wastes a lot of data Poor predictor of future performance	Simple/Efficient
Leave One Out	Inefficient	Does not waste data
K-Fold	Wastes 1/*K* of the data *K* times slower than Test Set	Wastes only 1/*K* of the data! Only *K* times slower than Test Set!

Classification Problems



- The exact same approaches apply for cross-validation except that the error is the number of data points that are misclassified.

Example: Training a Neural Net



- Train neural nets with different numbers of hidden units (more and more complex NNs)
- For each NN, evaluate the error using K-fold Cross-Validation
- Choose the one with the minimum cross-validation error