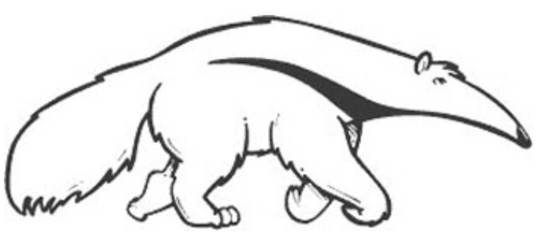
+

CS178: Machine Learning and Data Mining

Complexity & Nearest Neighbor Methods

Prof. Erik Sudderth



Some materials courtesy Alex Ihler & Sameer Singh

BREN:ICS INFORMATION AND COMPUTER SCIENCES

UNIVERSITY of CALIFORNIA IRVINE

CS178 Zoom Lectures

CS178 zoom lectures are recorded by the instructor (the recording feature is disabled for students). Recordings are posted to YuJa, and only available to CS178 students and staff. To ask questions during lecture, you may:

- Use the **Raise Hand** feature. Prof. Sudderth will then call on you by name, unmute your microphone, and let you ask a question. *Your question will be recorded. Please be respectful of your instructor and classmates.*
- Use the **Q&A Window** to type a question. Prof. Sudderth will read your question to the class before answering it, but *will not personally identify you.*

Machine Learning

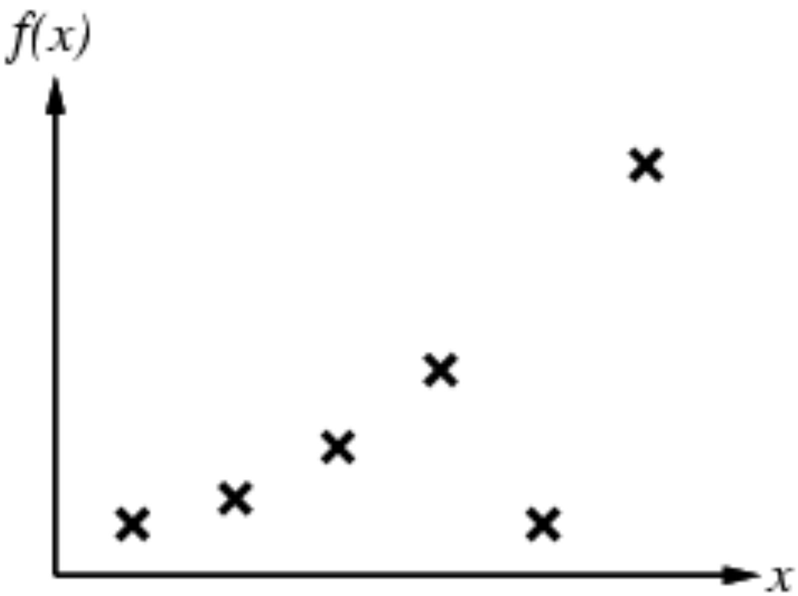
Complexity and Overfitting

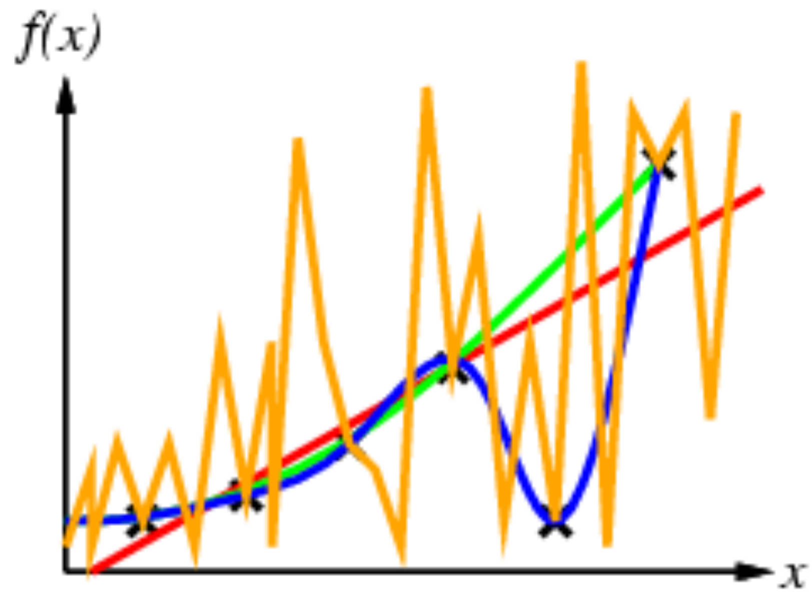
Nearest Neighbors

K-Nearest Neighbors

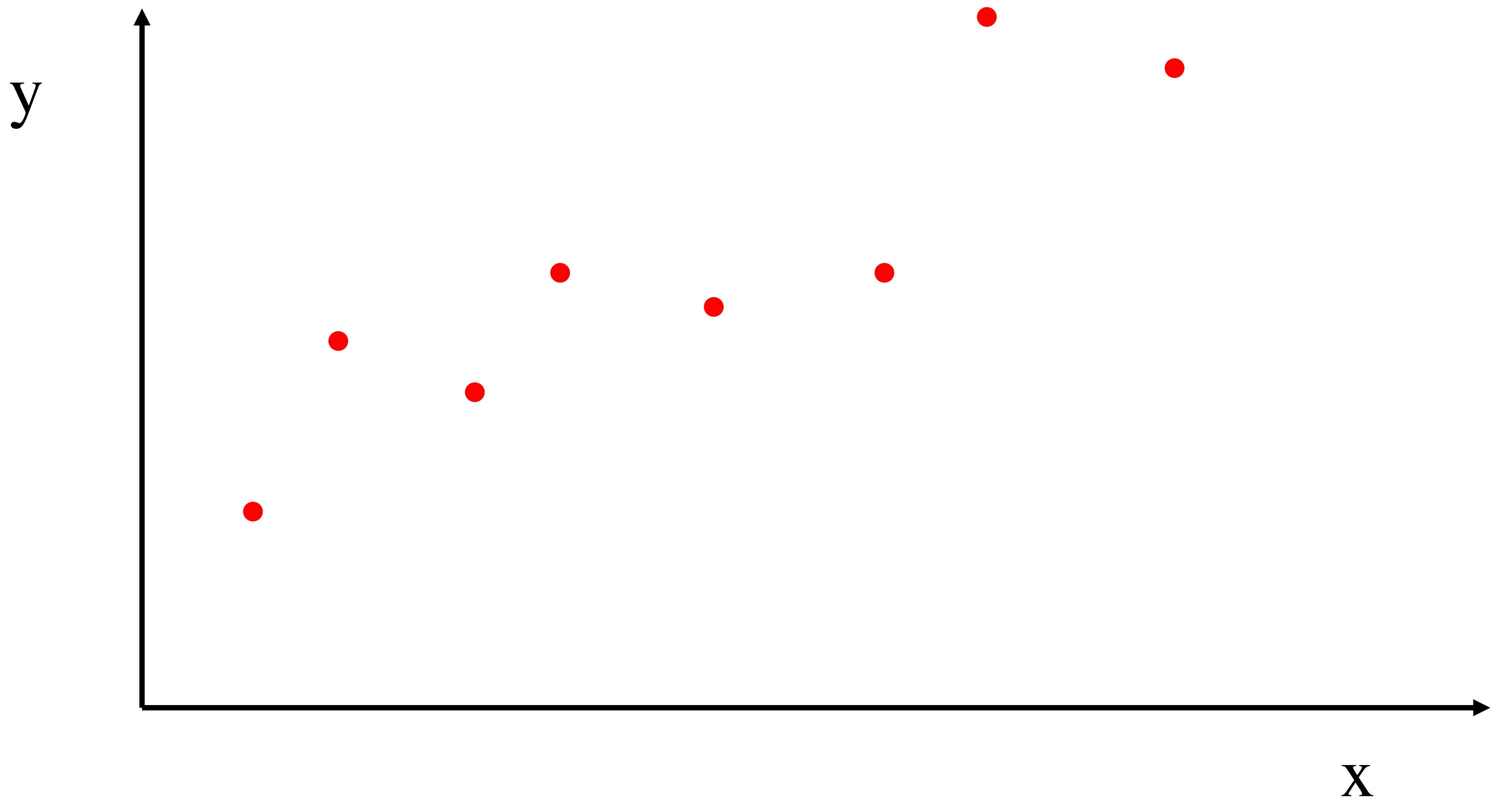
Inductive bias

- “Extend” observed data to unobserved examples
  - “Interpolate” / “extrapolate”
- What kinds of functions to expect? Prefer these (“bias”)
  - Usually, let data pull us away from assumptions only with evidence!



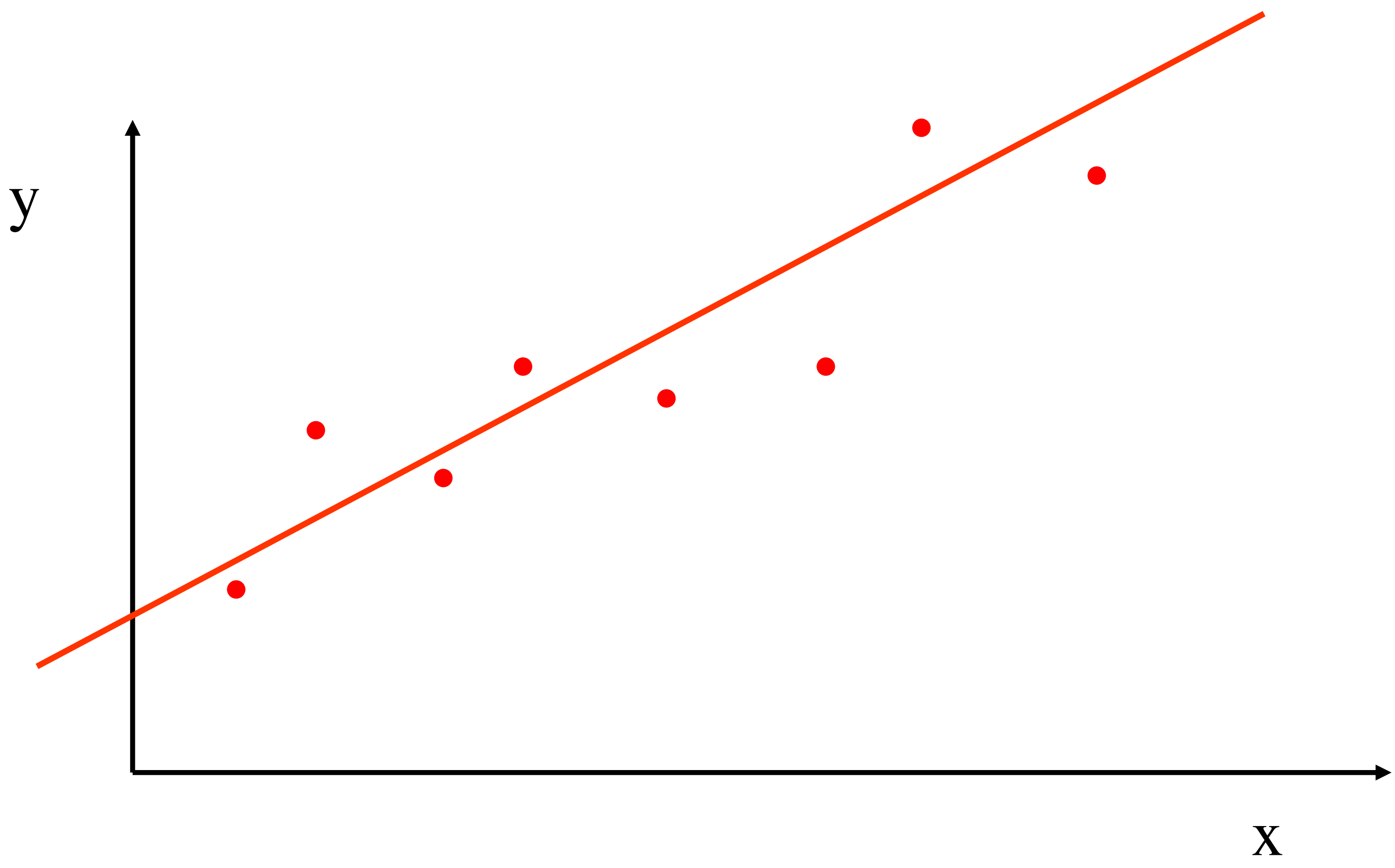


Overfitting and complexity

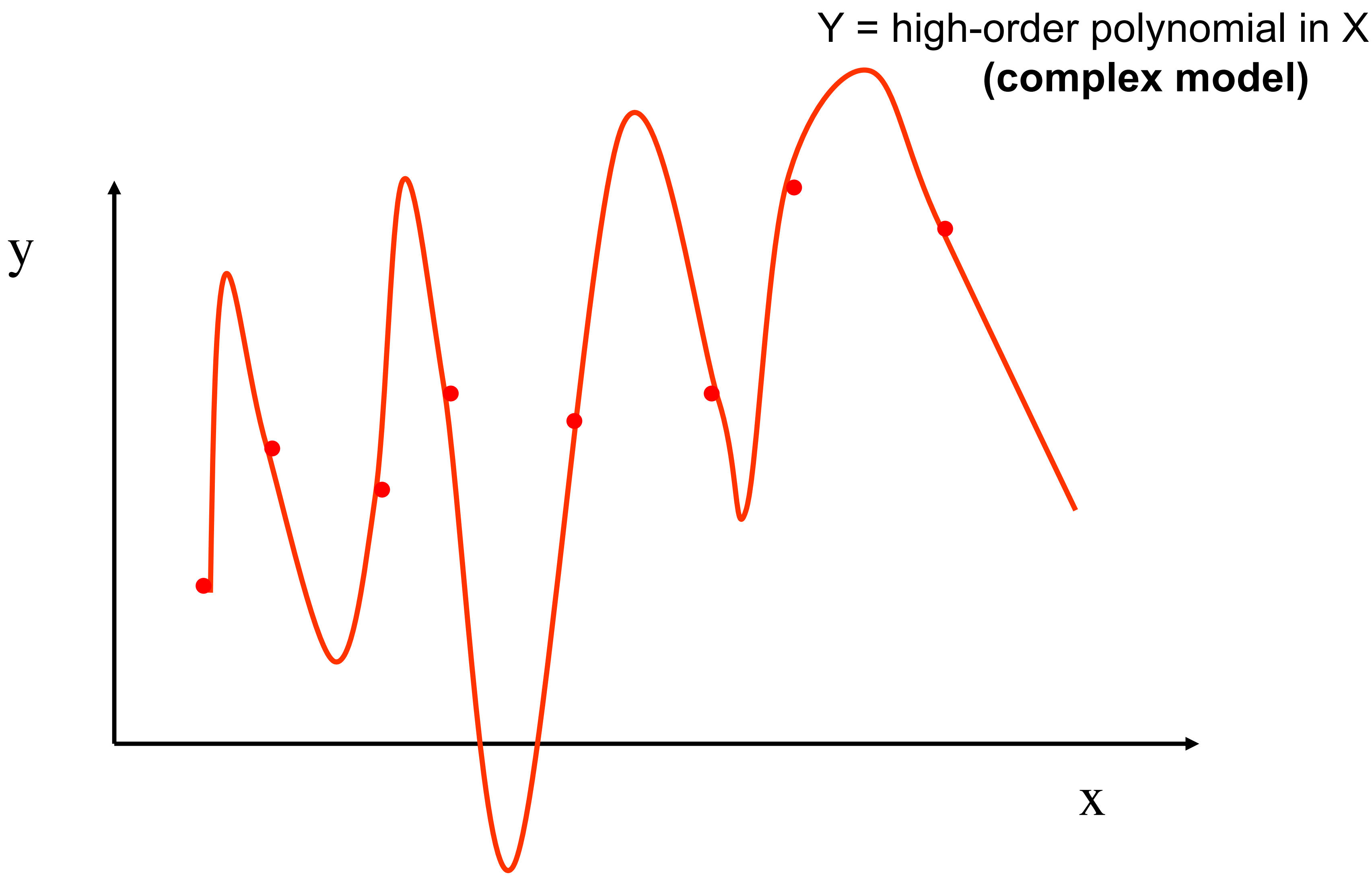


Overfitting and complexity

Simple model: Y= aX + b + e

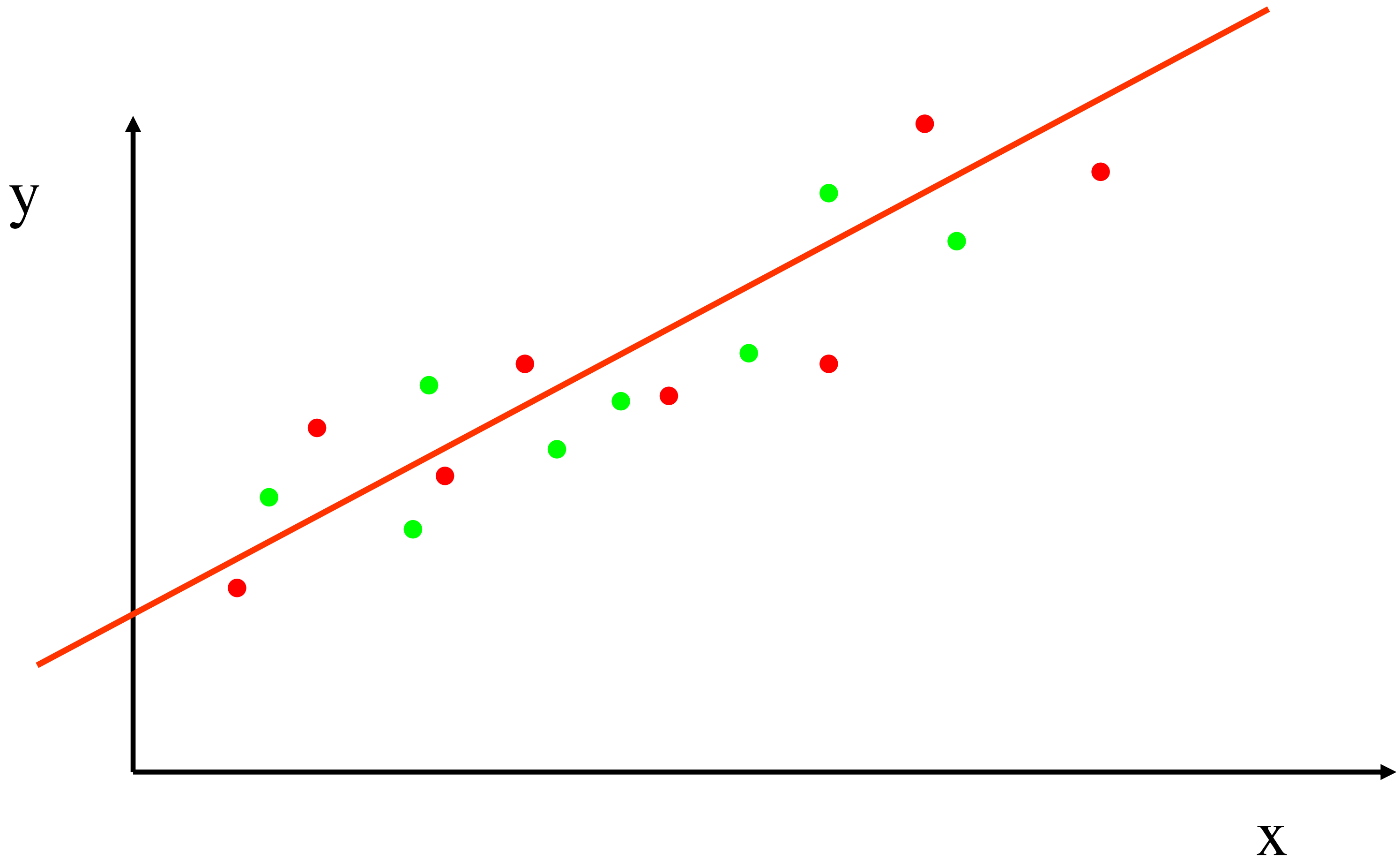


Overfitting and complexity

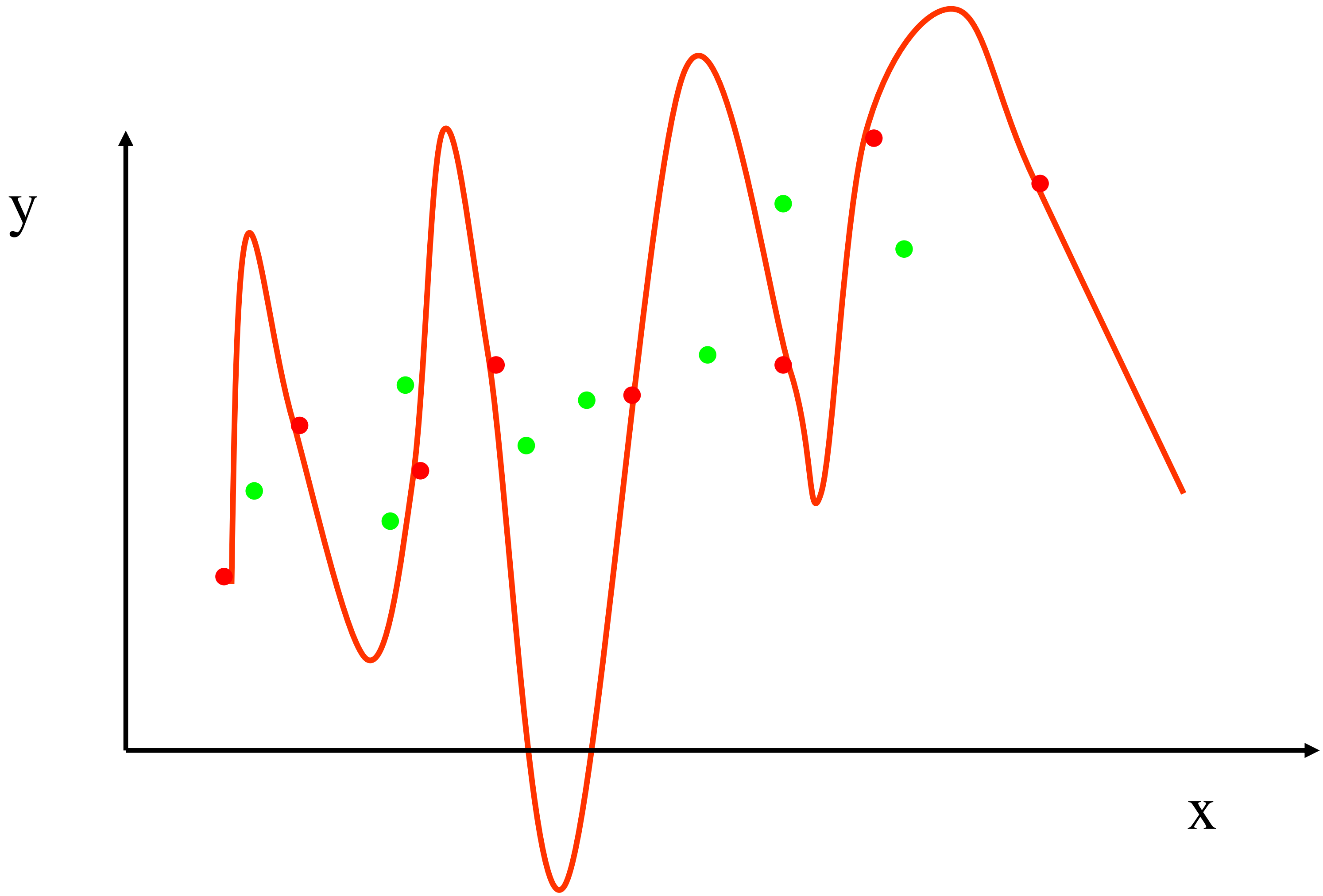


Overfitting and complexity

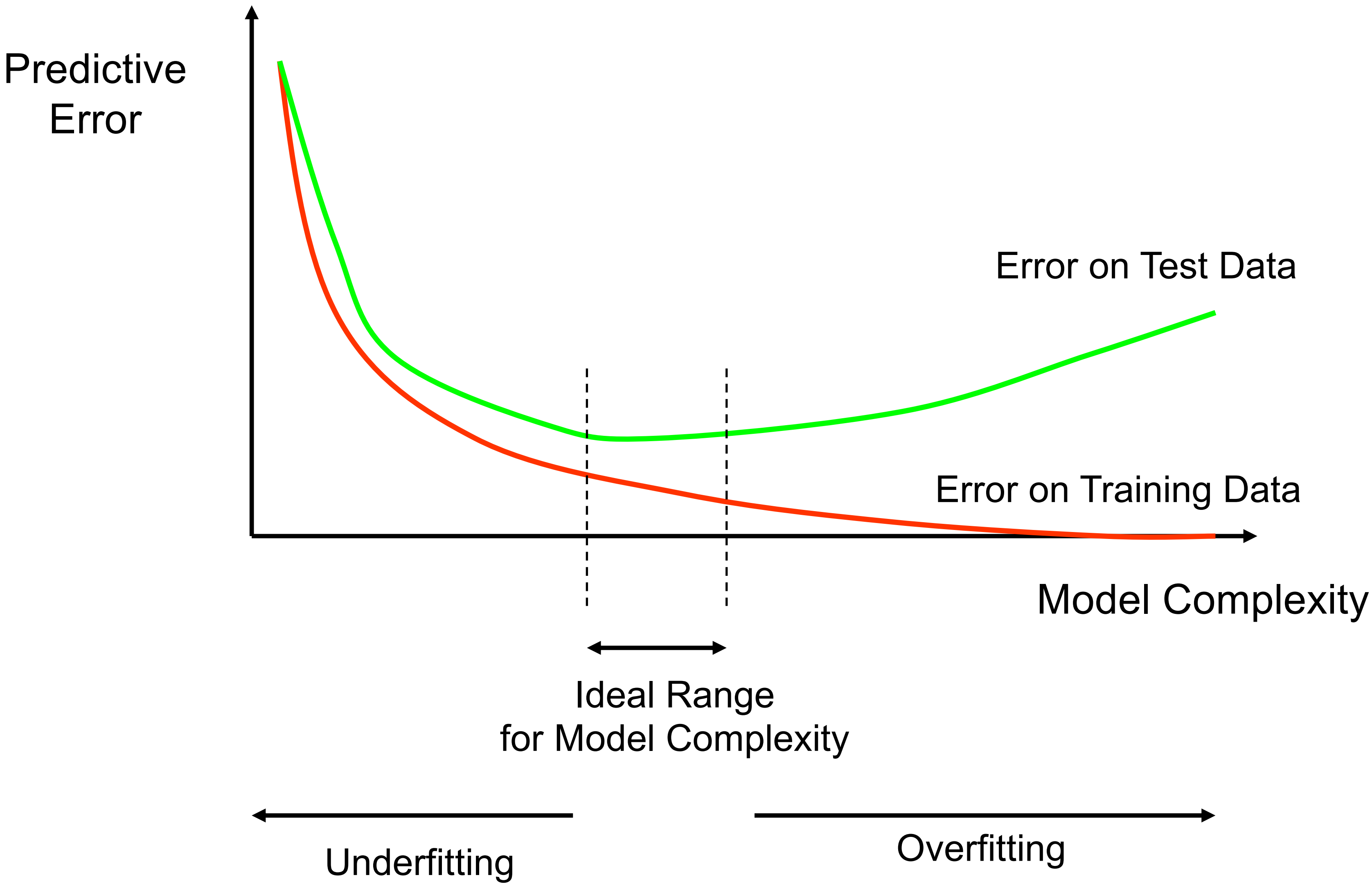
Simple model: Y= aX + b + e



Overfitting and complexity



How Overfitting affects Prediction



Training and Test Data

Data

- Several candidate learning algorithms or models, each of which can be fit to data and used for prediction
- How can we decide which is best?

Approach 1: Split into train and test data

Training Data	Test Data

- Learn parameters of each model from training data
- Evaluate all models on test data, and pick best performer

Problem:

- Over-estimates test performance ("lucky" model)
- Learning algorithms should *never* have access to test data

Training, Validation, and Test Data

Data

- Several candidate learning algorithms or models, each of which can be fit to data and used for prediction
- How can we decide which is best?

Approach 2: Reserve some data for validation

Training Data	Validation	Test Data

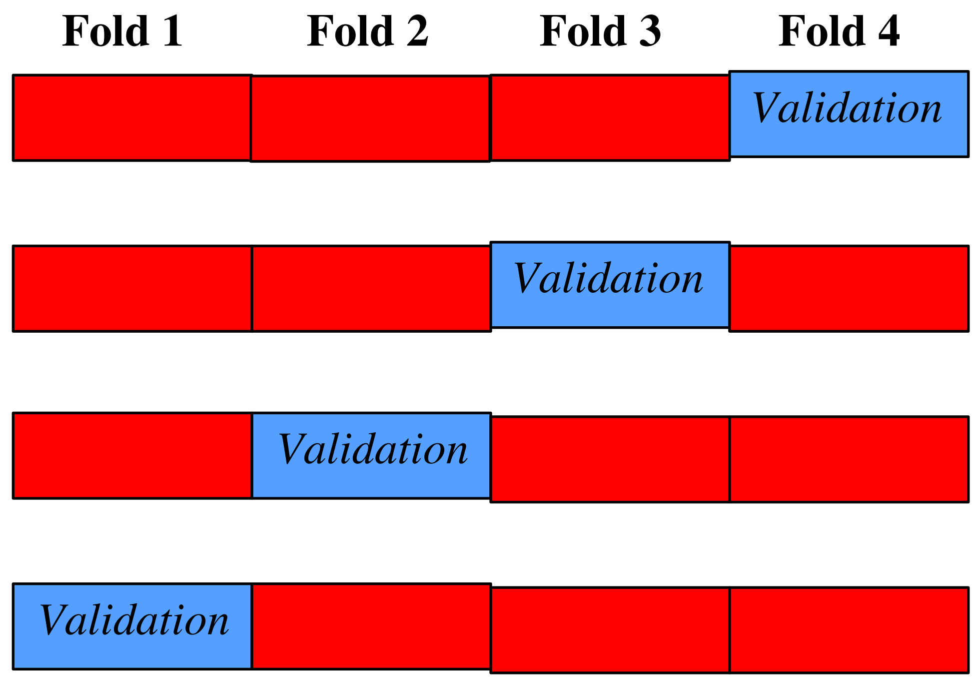
- Learn parameters of each model from training data
- Evaluate models on validation data, pick best performer
- Reserve test data to benchmark chosen model

Problem:

- Wasteful of training data (learning can't use validation)
- May bias selection towards overly simple models

Cross-Validation

- Divide training data into K equal-sized *folds*
- Train on K-1 folds, evaluate on remainder
- Pick model with best average performance across K trials



How many folds?

- *Bias:* Too few, and effective training dataset much smaller
- *Variance:* Too many, and test performance estimates noisy
- *Cost:* Must run training algorithm once per fold (parallelizable)
- *Practical rule of thumb:* 5-fold or 10-fold cross-validation
- *Theoretically troubled:* Leave-one-out cross-validation, K=N

Competitions

- Training data
  - Used to build your model(s)
- Validation data
  - Used to assess, select among, or combine models
  - Personal validation; leaderboard; …
- Test data
  - Used to estimate “real world” performance

#	Δ1w	Team Name * in the money	Score	Entries	Last Submission UT
1	-	BrickMover *	1.21251	40	Sat, 31 Aug 2013 23:
2	new	vsu *	1.21552	13	Sat, 31 Aug 2013 20:
3	↑2	Merlion	1.22724	29	Sat, 31 Aug 2013 23:
4	↓2	Sergey	1.22856	15	Sat, 31 Aug 2013 23:
5	new	liuyongqi	1.22980	13	Sat, 31 Aug 2013 13:

Machine Learning

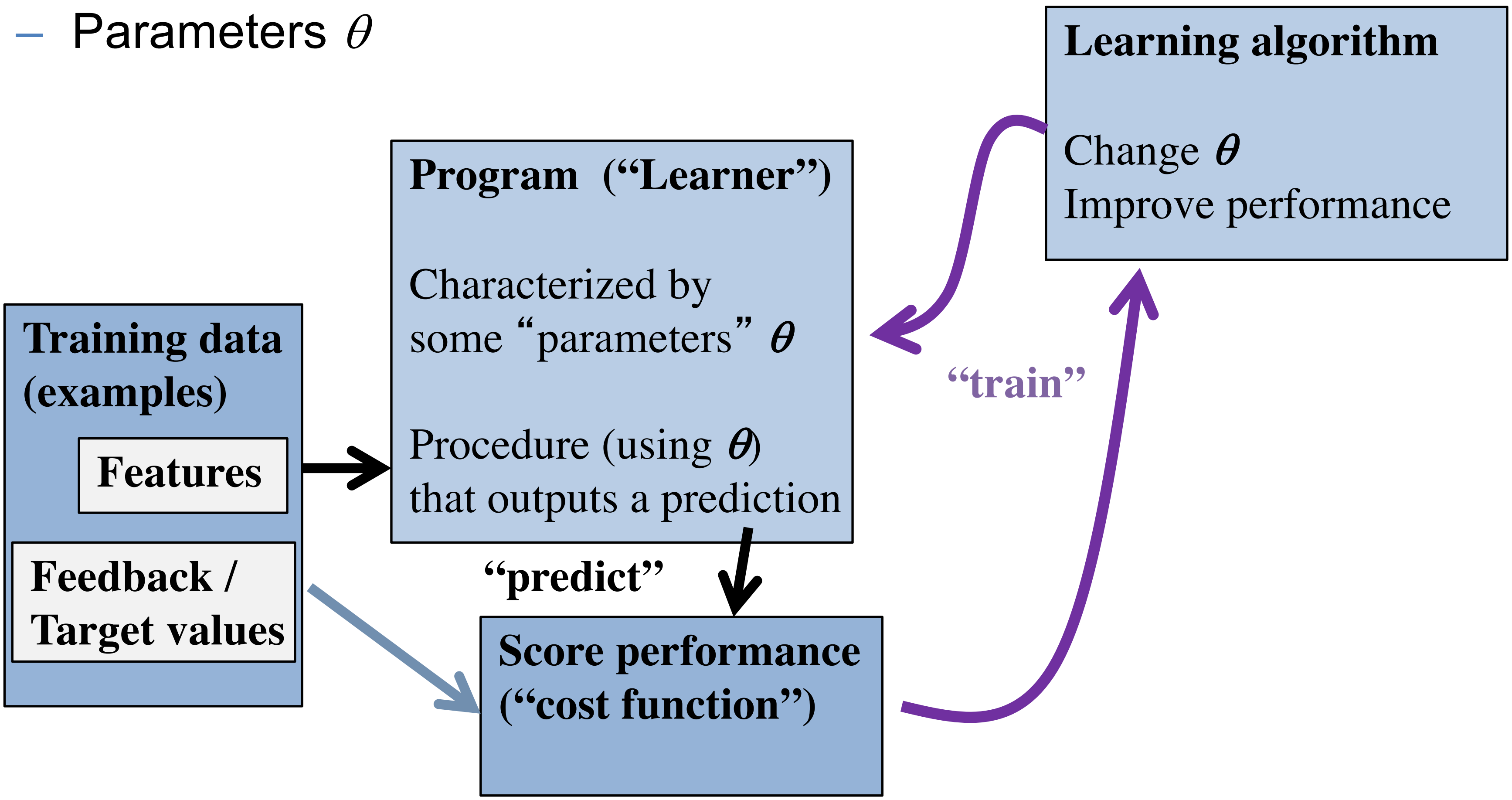
Complexity and Overfitting

Nearest Neighbors

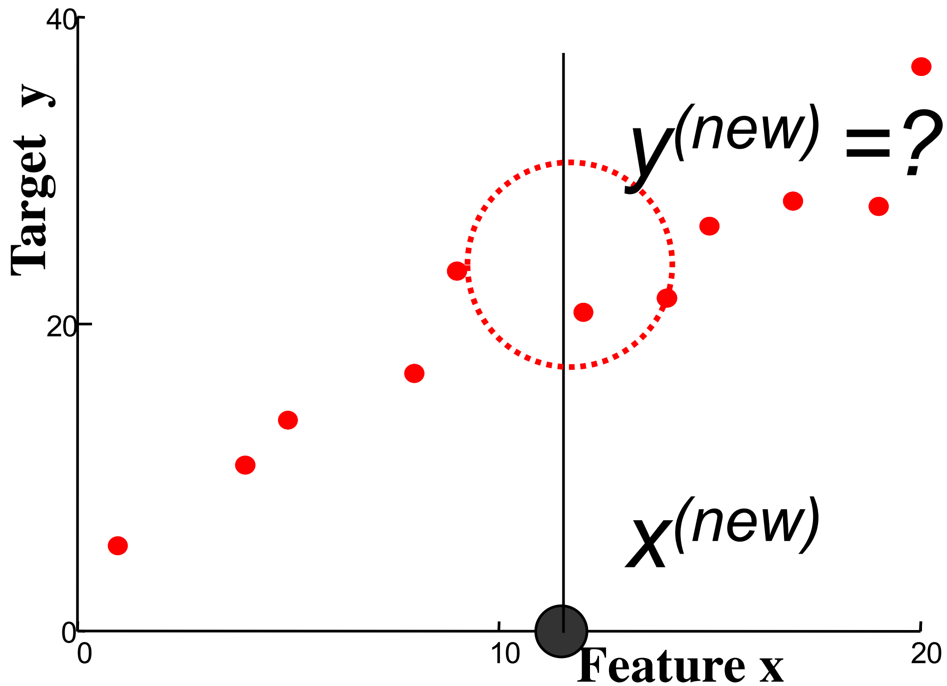
K-Nearest Neighbors

Supervised learning

- Notation
  - Features x
  - Targets y
  - Predictions $\hat{y} = f(x\,;\,\theta)$
  - Parameters θ

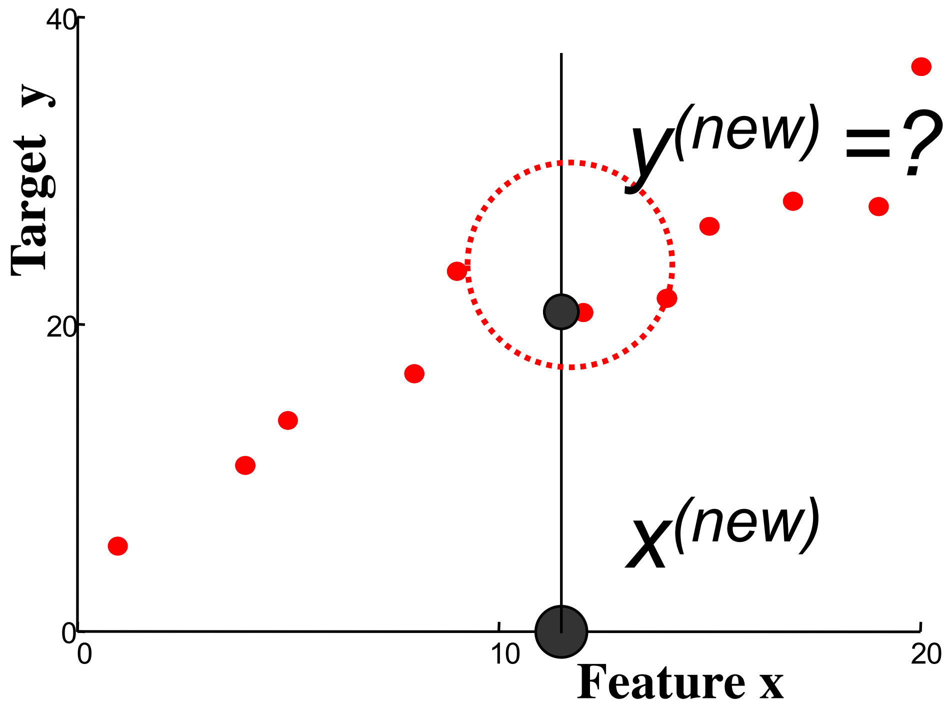


Regression; Scatter plots



- Suggests a relationship between x and y
- Regression: given new observed $x^{(new)}$, estimate $y^{(new)}$

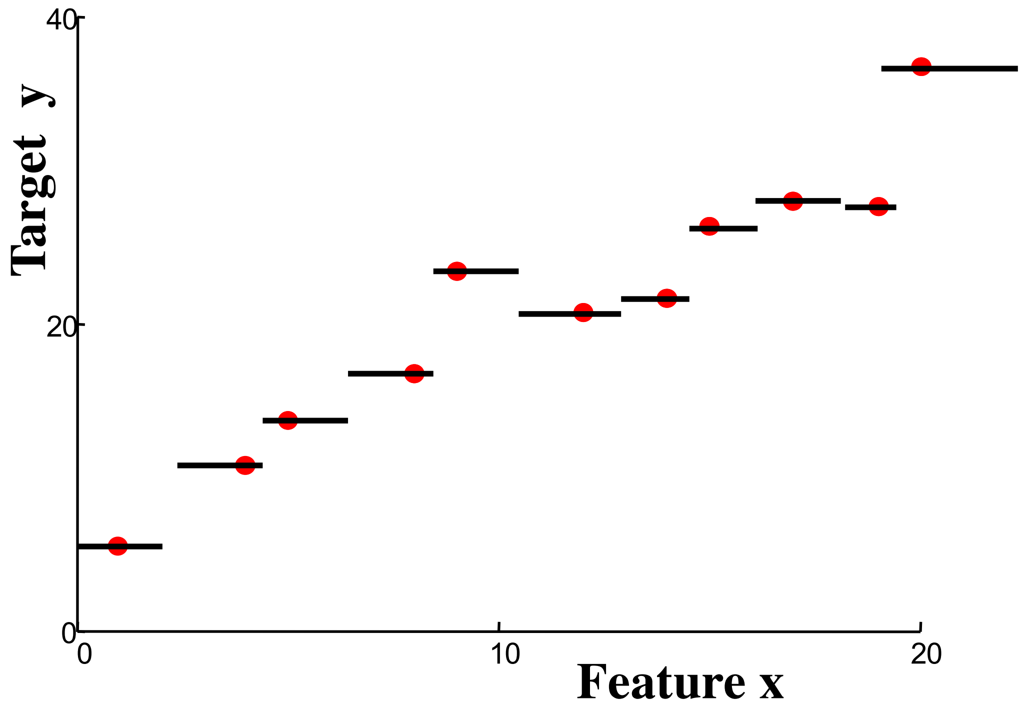
Nearest neighbor regression



"Predictor":
Given new features:
Find nearest example
Return its value

- Find training datum $x^{(i)}$ closest to $x^{(new)}$; predict $y^{(i)}$

Nearest neighbor regression

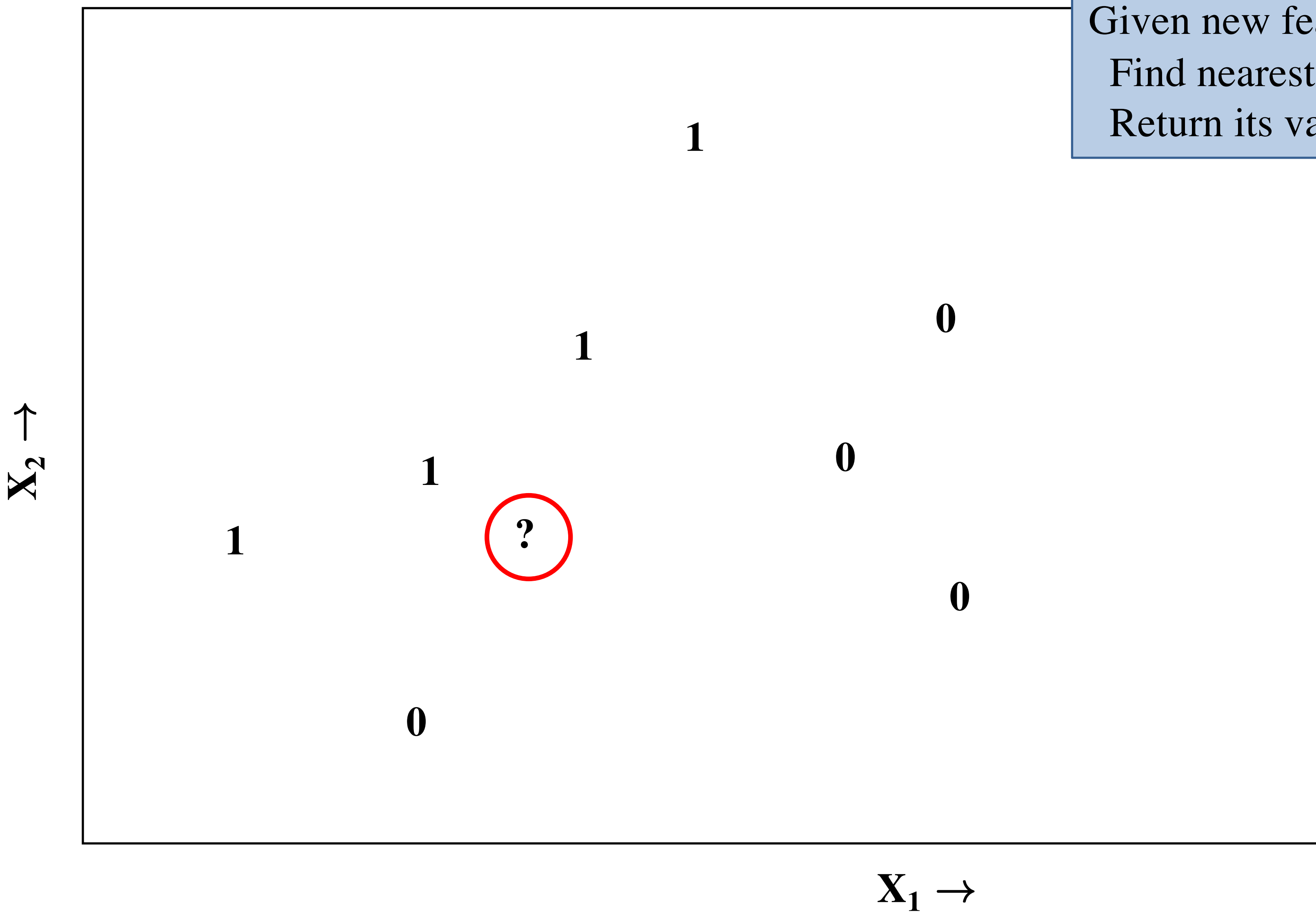


"Predictor":
Given new features:
Find nearest example
Return its value

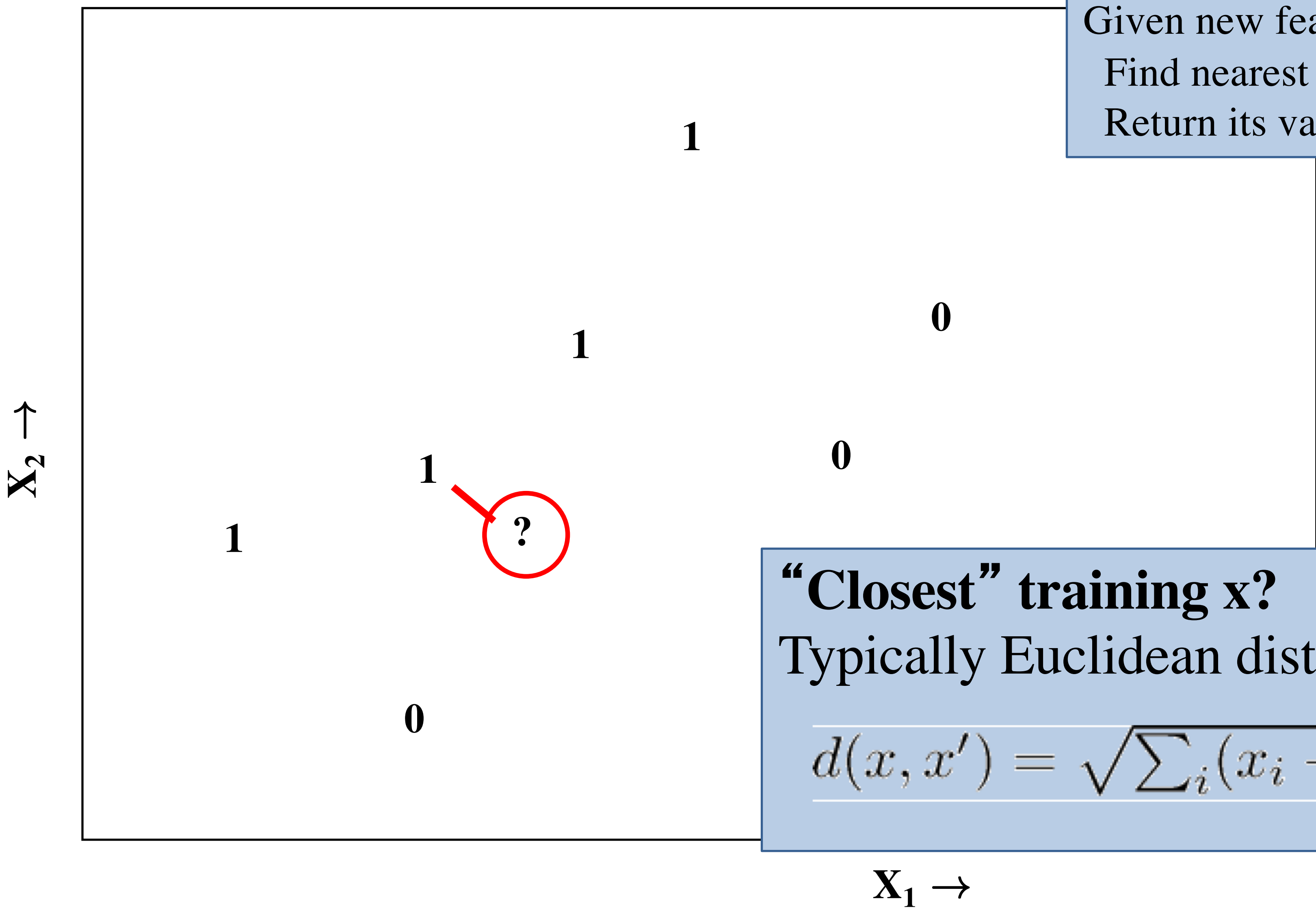
- Find training datum $x^{(i)}$ closest to $x^{(new)}$; predict $y^{(i)}$
- Defines an (implict) function f(x)
- "Form" is piecewise constant

Nearest neighbor classifier

"Predictor":
Given new features:
Find nearest example
Return its value



Nearest neighbor classifier

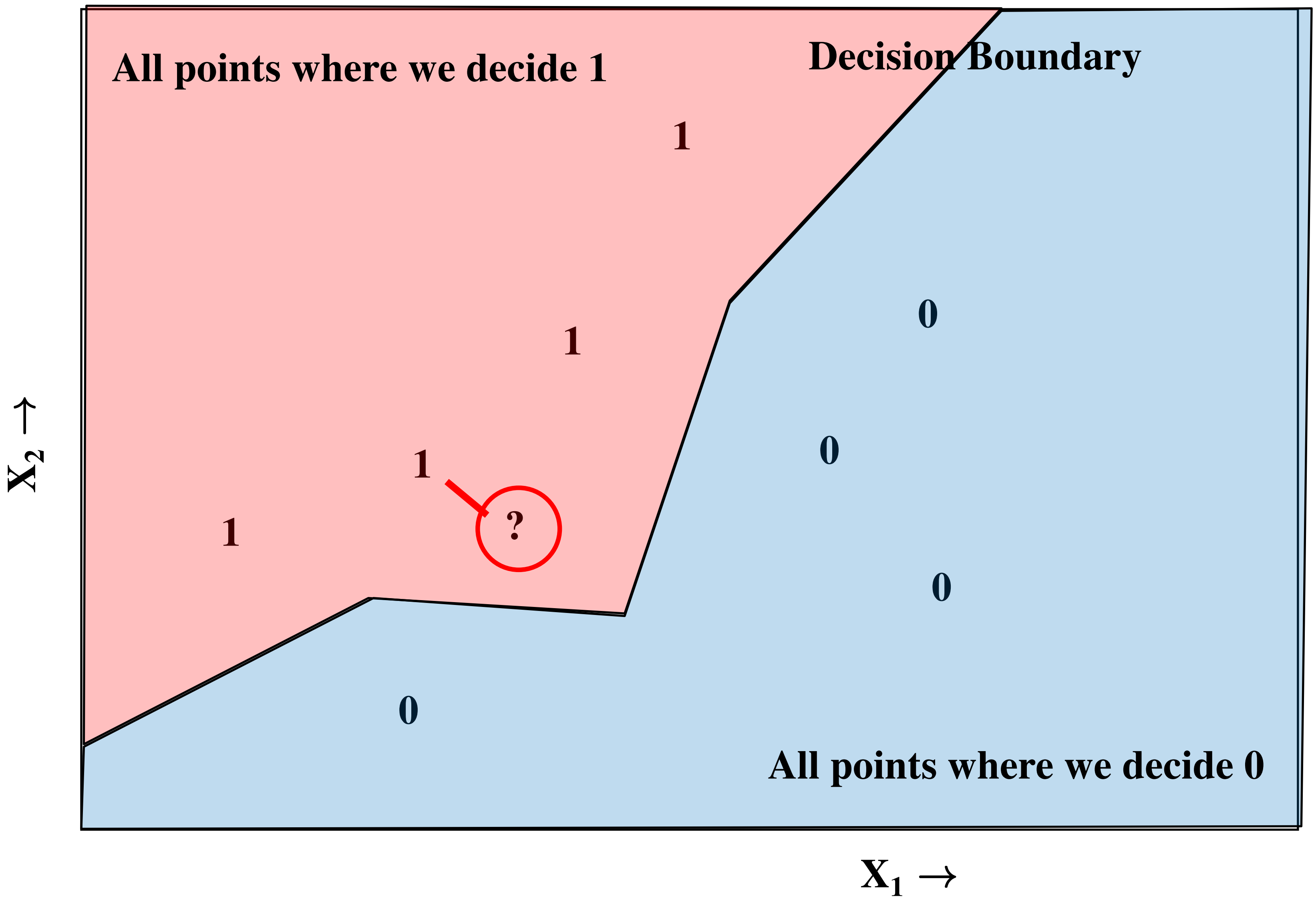


"Predictor":
Given new features:
Find nearest example
Return its value

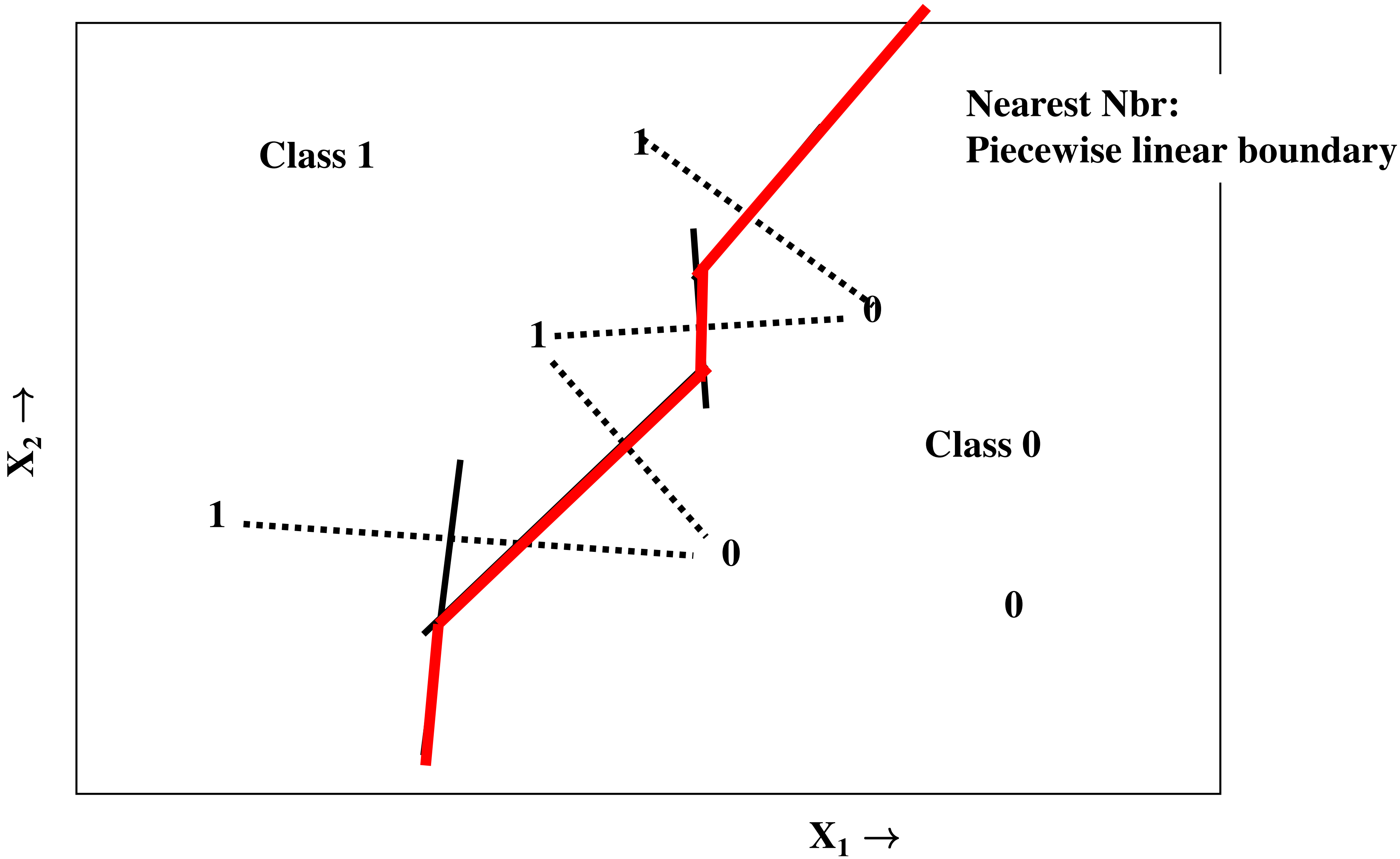
"Closest" training x?
Typically Euclidean distance:

$$d(x, x') = \sqrt{\sum_i (x_i - x'_i)^2}$$

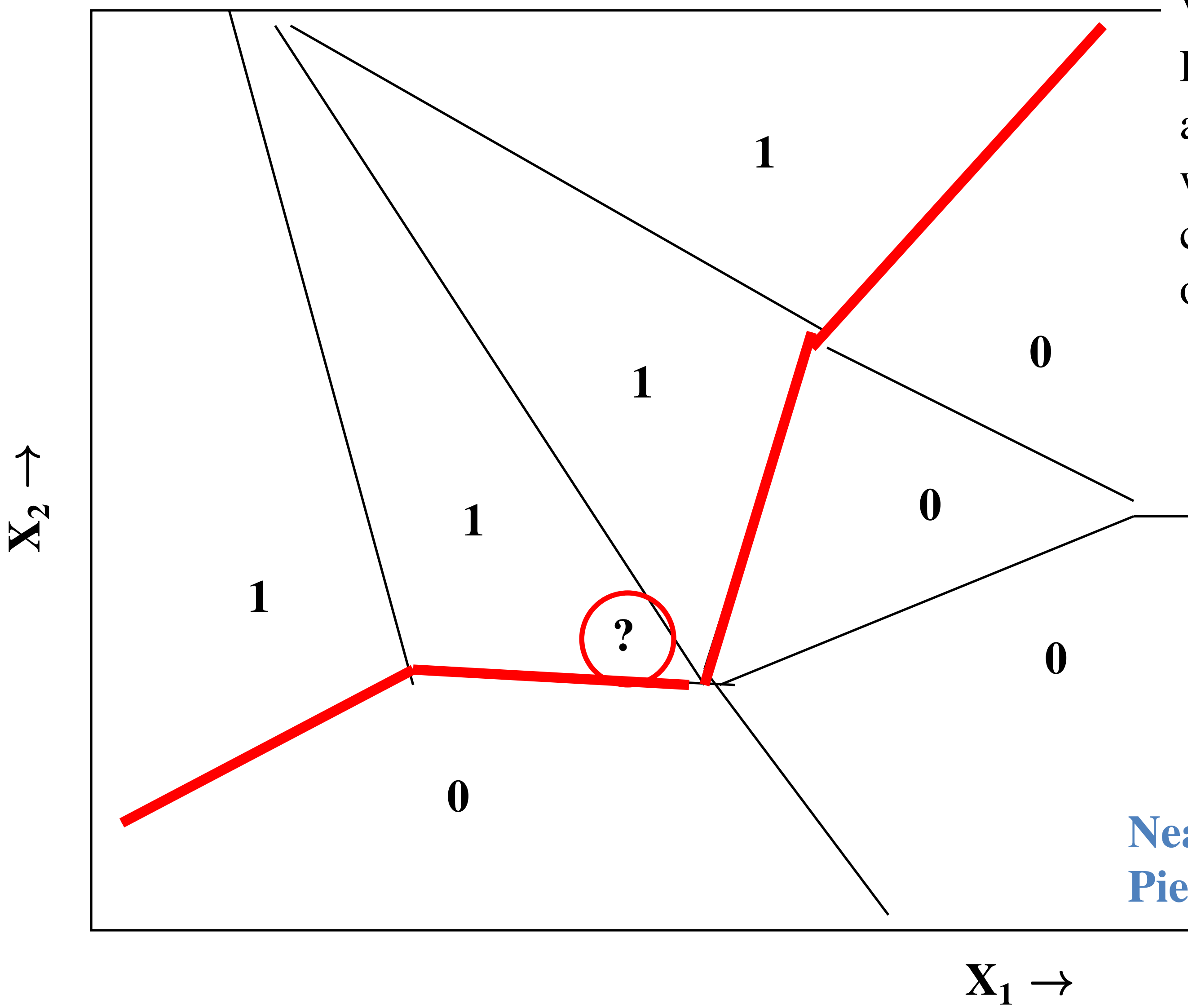
Nearest neighbor classifier



Nearest neighbor classifier



Nearest neighbor classifier

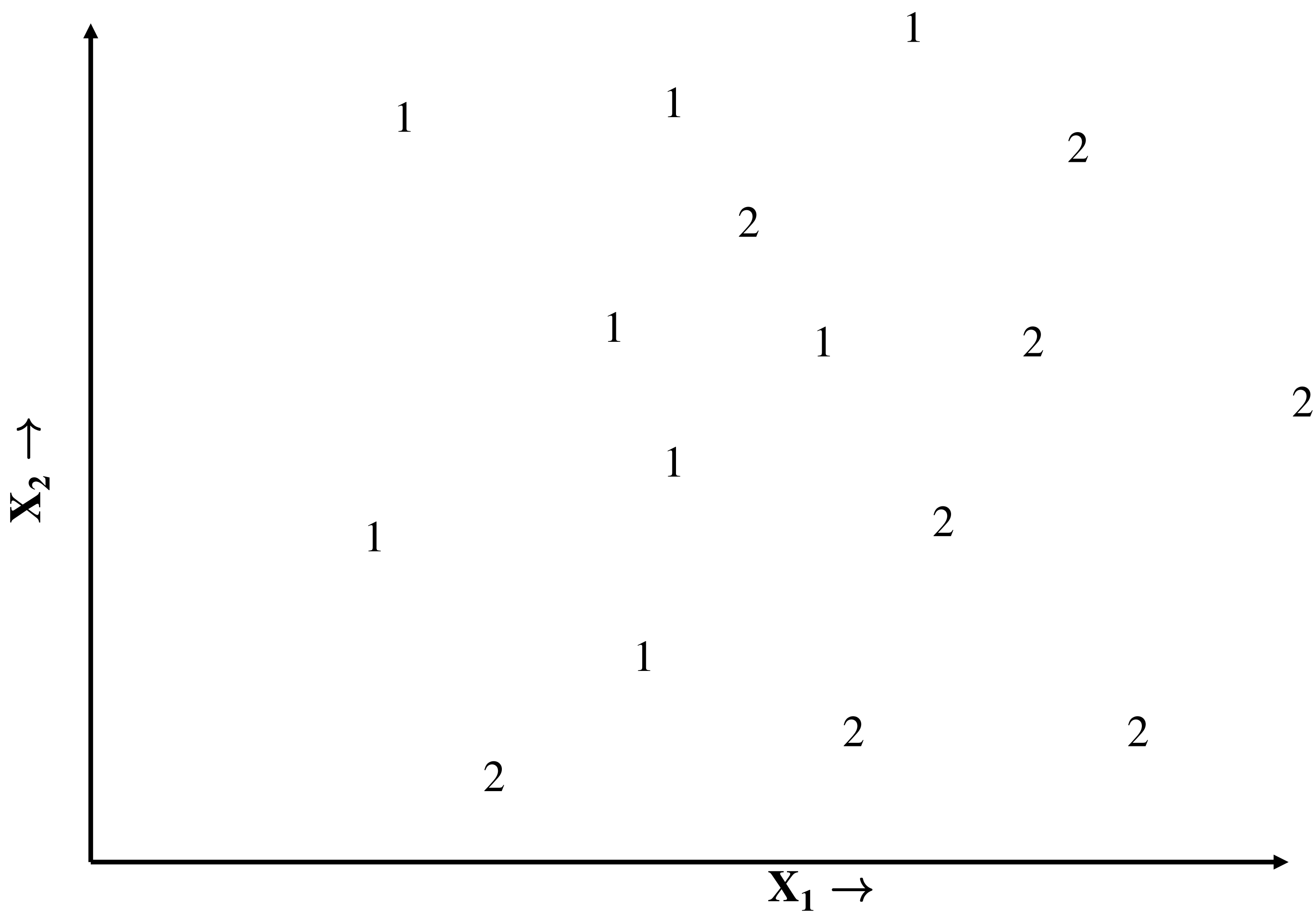


Voronoi tessellation: Each datum is assigned to a region, in which all points are closer to it than any other datum

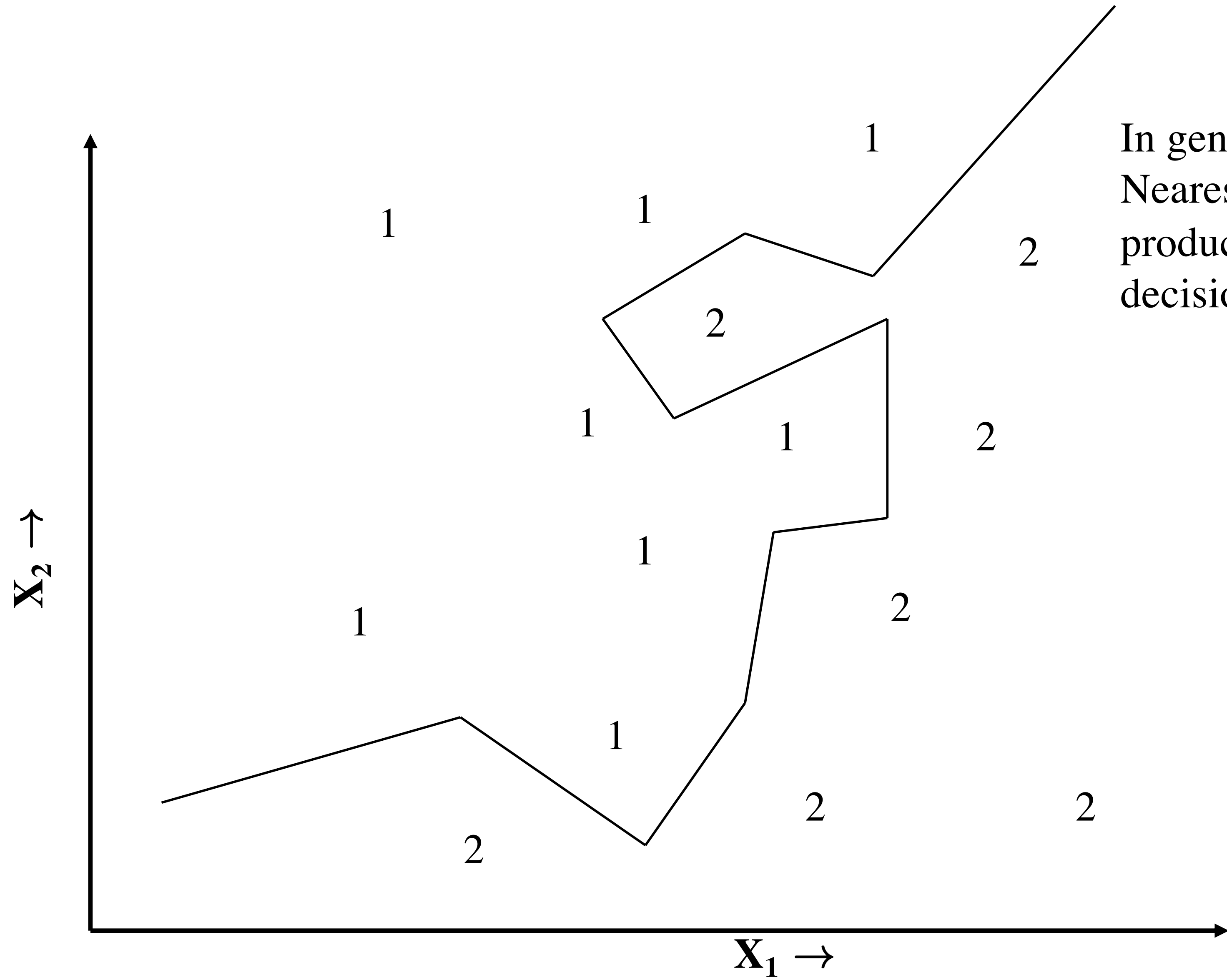
Decision boundary: Those edges across which the decision (class of nearest training datum) changes

Nearest Nbr: Piecewise linear boundary

More Data Points

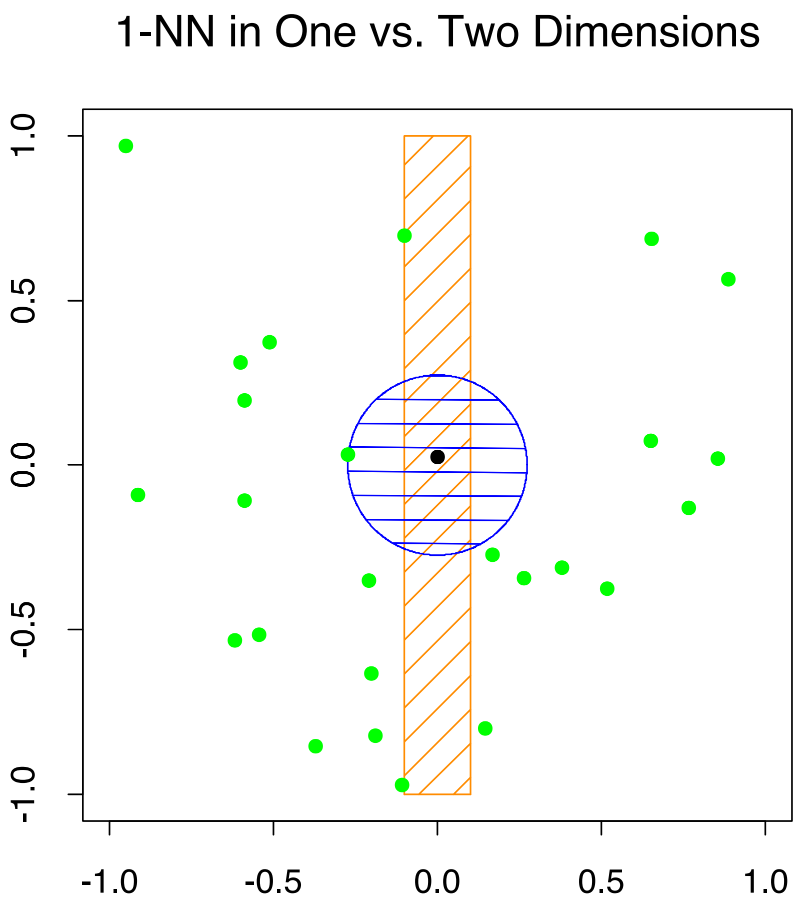


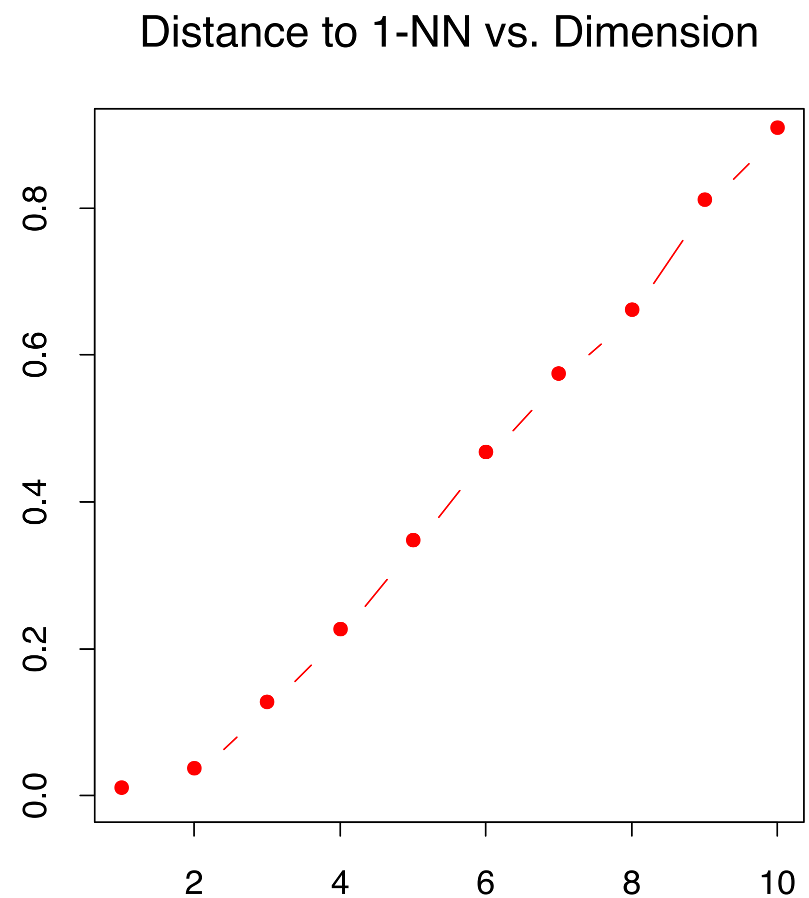
More Complex Decision Boundary



In general:
Nearest-neighbor classifier produces piecewise linear decision boundaries

Issue: Neighbor Distance & Dimension





For m=1000 data in unit hypercube

Machine Learning

Complexity and Overfitting

Nearest Neighbors

K-Nearest Neighbors

K-Nearest Neighbor (kNN) Classifier

- Find the k-nearest neighbors to <u>x</u> in the data
  - i.e., rank the feature vectors according to Euclidean distance
  - select the k vectors which are have smallest distance to <u>x</u>

- Regression
  - Usually just average the y-values of the k closest training examples

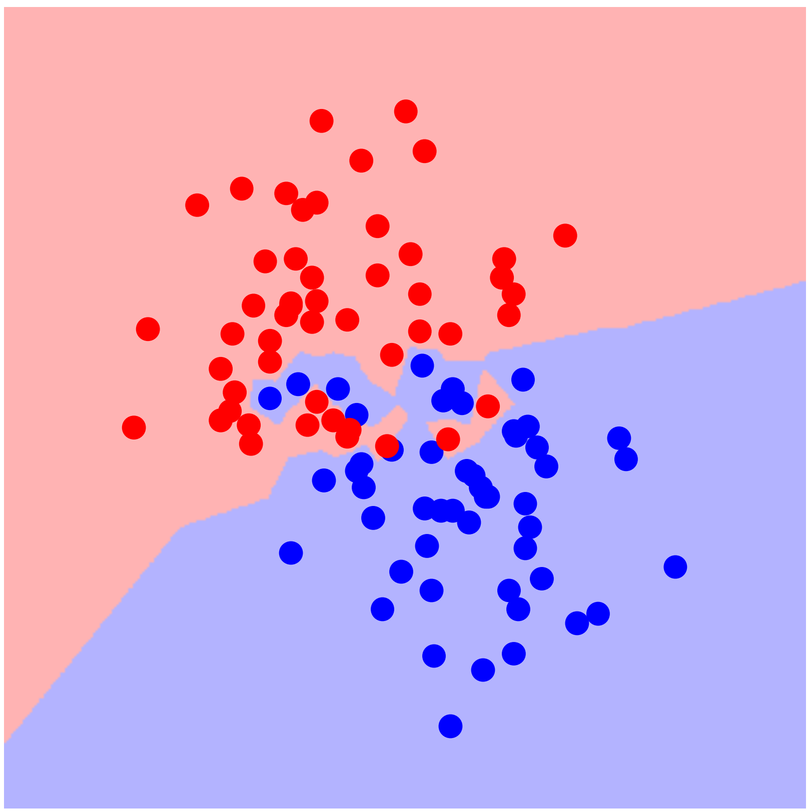
- Classification
  - ranking yields k feature vectors and a set of k class labels
  - pick the class label which is most common in this set ("vote")
  - classify <u>x</u> as belonging to this class
  - Note: for two-class problems, if k is odd (k=1, 3, 5, …) there will never be any "ties"; otherwise, just use (any) tie-breaking rule

- "Training" is trivial: just use training data as a lookup table, and search to classify a new datum
- "Testing" can be computationally expensive: must find nearest neighbors within a potentially large training set

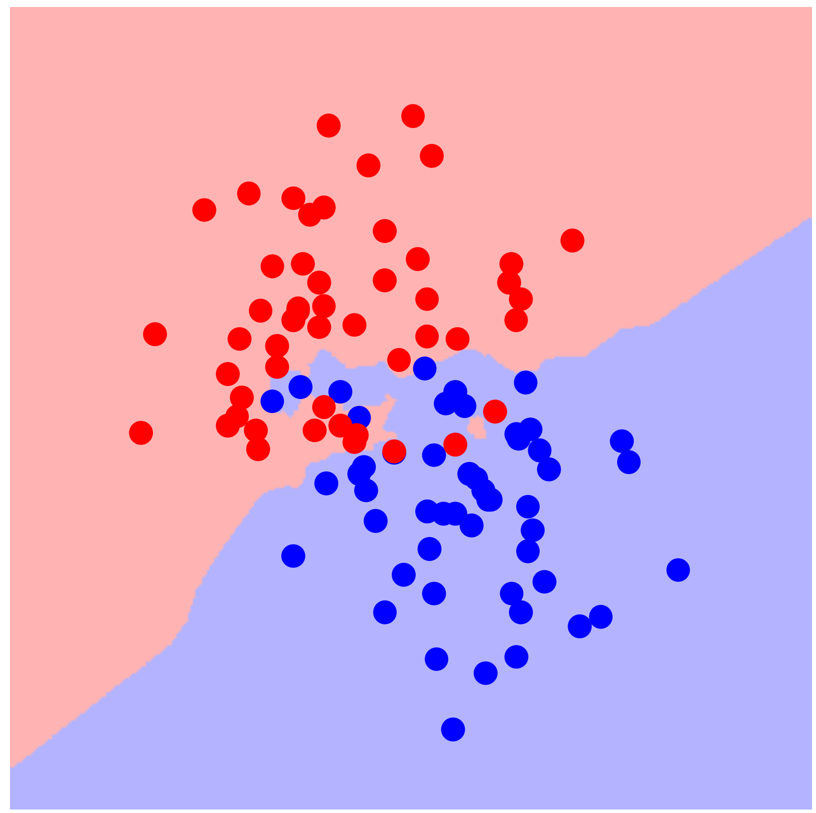
kNN Decision Boundary

- Piecewise linear decision boundary
- Increasing k “simplifies” decision boundary
  - Majority voting means less emphasis on individual points

K = 1



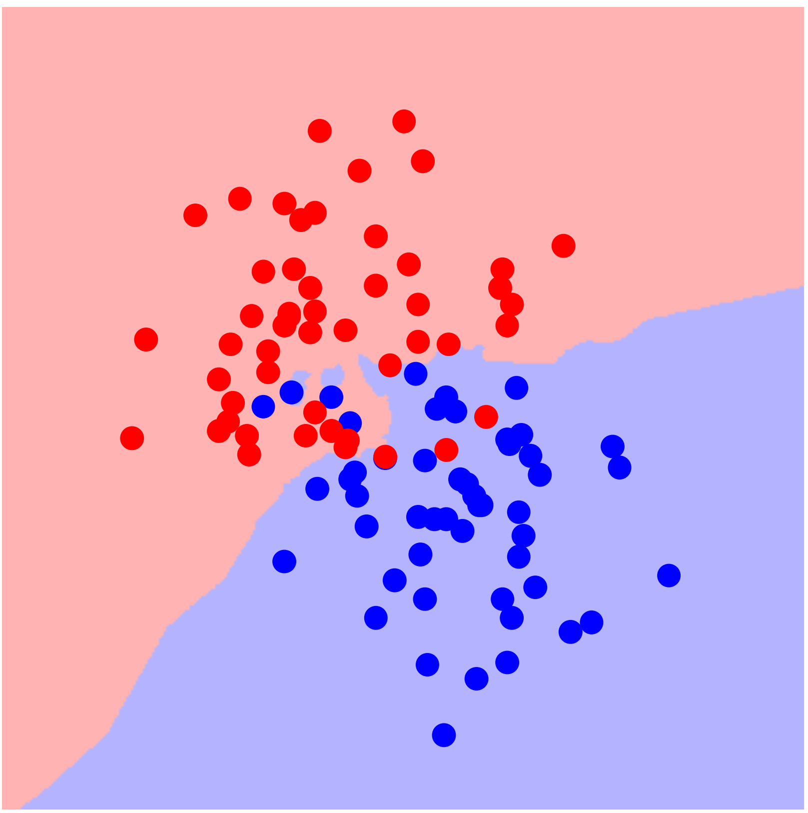
K = 3



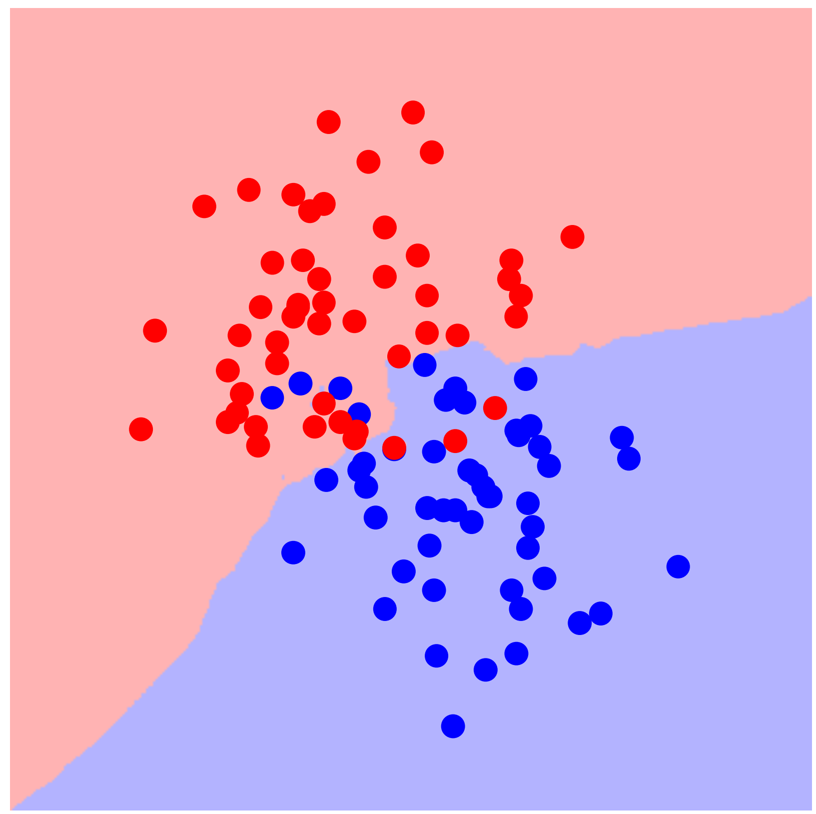
kNN Decision Boundary

- Piecewise linear decision boundary
- Increasing k “simplifies” decision boundary
  - Majority voting means less emphasis on individual points

K = 5



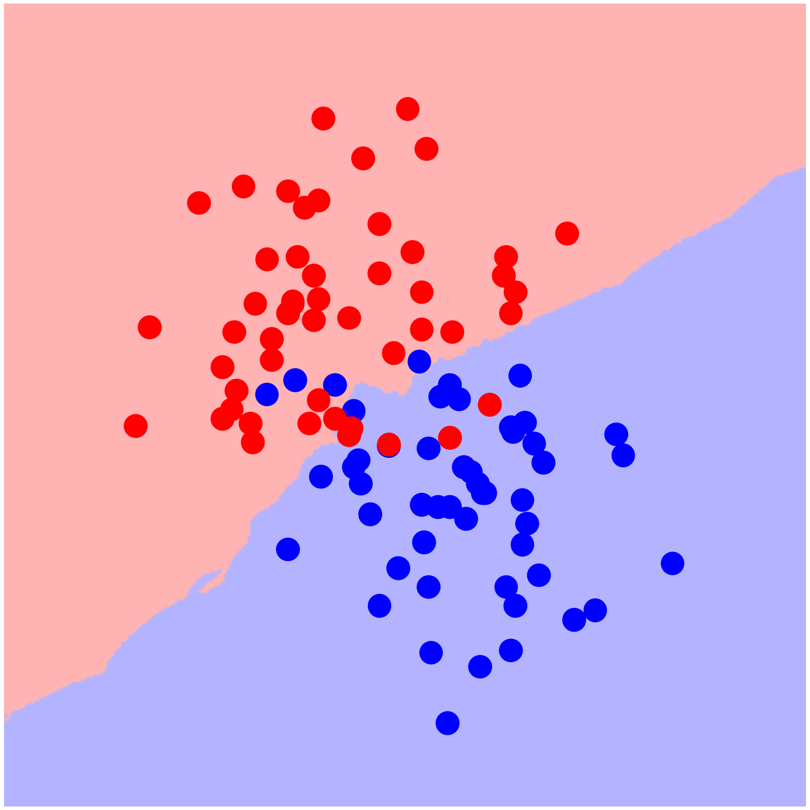
K = 7



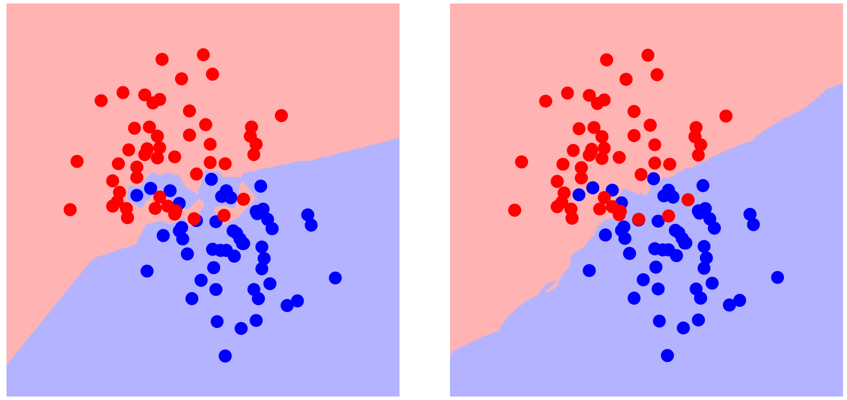
kNN Decision Boundary

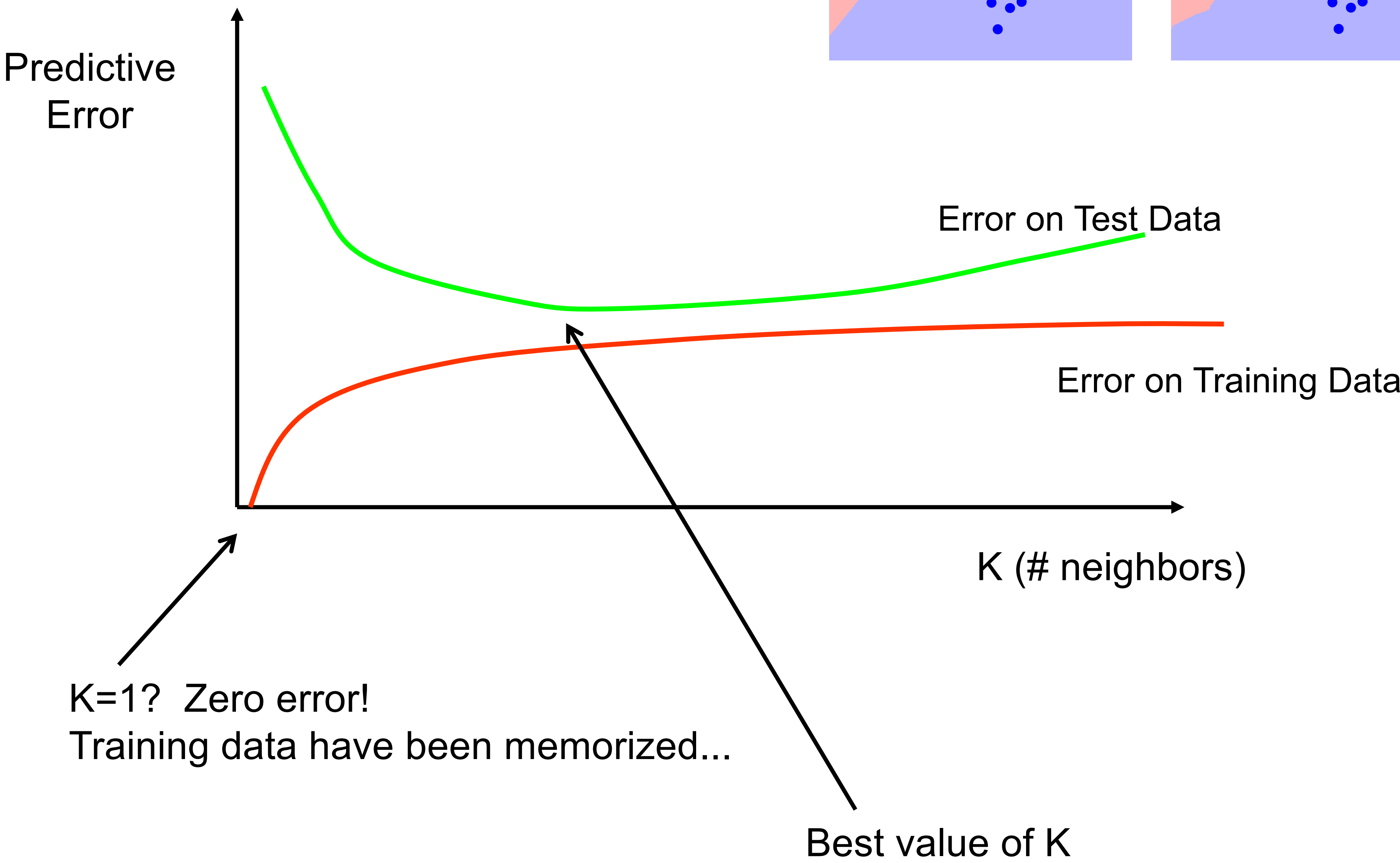
- Piecewise linear decision boundary
- Increasing k “simplifies” decision boundary
  - Majority voting means less emphasis on individual points

K = 25



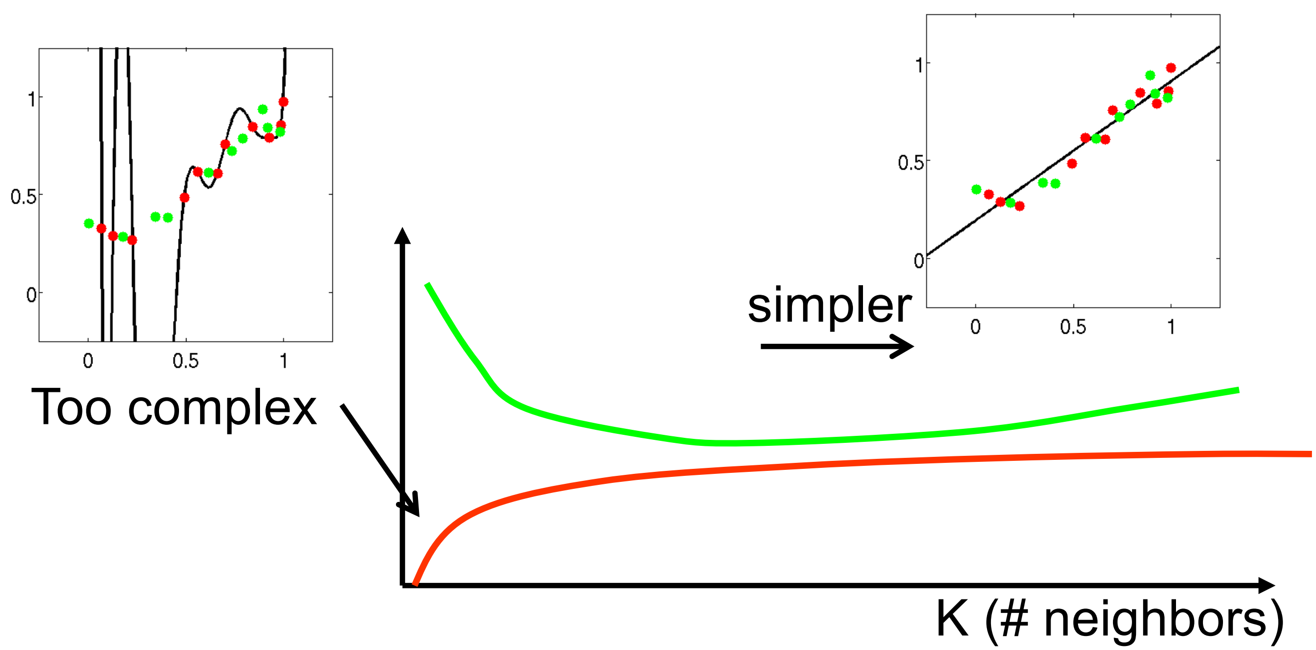
Error rates and K





Complexity & Overfitting

- Complex model predicts all training points well
- Doesn’t generalize to new data points
- k = 1 : perfect memorization of examples (complex)
- k = m : always predict majority class in dataset (simple)
- Can select k using validation data, etc.



K-Nearest Neighbor (kNN) Classifier

- Theoretical Considerations
  - as k increases
    - we are averaging over more neighbors
    - the effective decision boundary is more "smooth"
  - as m increases, the optimal k value tends to increase
  - k=1, m increasing to infinity : error < 2x optimal

- Extensions of the Nearest Neighbor classifier
  - Weighted distances $d(x, x') = \sqrt{\sum_i w_i (x_i - x'_i)^2}$
    - e.g., some features may be more important; others may be irrelevant
  - Fast search techniques (indexing) to find k-nearest points in d-space
  - Weighted average / voting based on distance

Digit & Hand Gesture Recognition

Athitsos et al., CVPR 2004 & PAMI 2008

Tricks to build efficient & accurate nearest-neighbor classifiers:

- **Gather large training sets (possibly by generating synthetic data)**
- **Engineer clever distance functions that are invariant to aspects of the data unrelated to the class label**
- **Use algorithms to find (approximate) nearest neighbors in sub-linear time (locality sensitive hashing, class-sensitive embeddings, KD-trees, etc.)**

Summary

- K-nearest neighbor models
  - Classification (vote)
  - Regression (average or weighted average)
- Piecewise linear decision boundary
  - How to calculate
- Test data and overfitting
  - Model "complexity" for knn
  - Use validation data to estimate test error rates & select k