How flexible should my algorithms be?

Supervised machine learning model

We search for the model that best fits our data

$$y = f(x, \boldsymbol{w}) + \epsilon$$

Labels

Model

Input Data

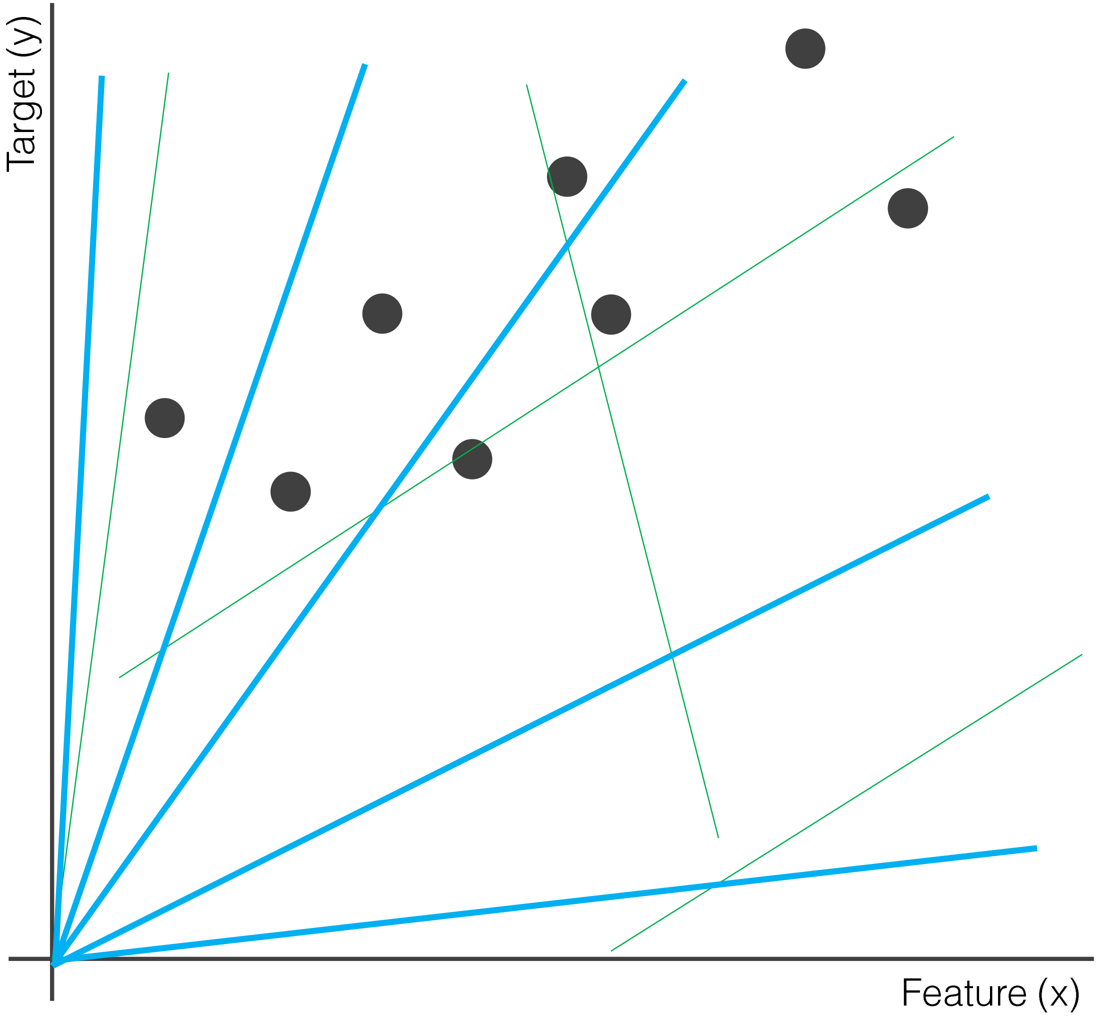
Parameter(s)

Random Noise

Hypothesis

Typically prevents perfect performance

Example: linear regression



Using any line as a hypothesis function, how many possible hypothesis functions are in the set?

Infinitely many

Using the line $y = wx$ as the family of hypothesis functions, how many possible hypothesis functions are in the set?

Infinitely many

Which set contains the better hypothesis?

Which set has more options to consider?

What is our learning algorithm?

Algorithm Development

Application

(iterate as needed)

1. **Load training data**
2. **Preprocess & extract features**
3. **Choose learning model**
4. **Evaluate performance**
5. **Make predictions on unseen data**

y

1
1
0
0
1
0

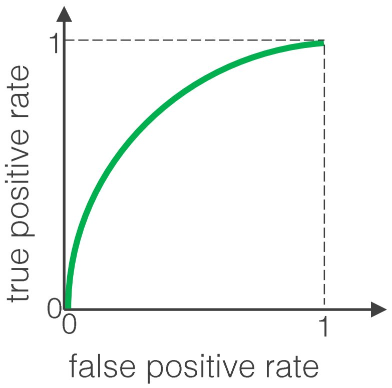
X

x_1 x_2 x_3 x_4 x_5 x_6

X'

x_1	0.38	0.39	0.85	0.78
x_2	0.81	0.91	0.97	0.53
x_3	0.65	0.59	0.91	0.11
x_4	0.94	0.05	0.40	0.26
x_5	0.27	0.19	0.03	0.64
x_6	0.02	0.98	0.36	0.11

linear discriminant
perceptron
logistic regression
decision trees
random forests
support vector machine
k nearest neighbors
neural networks

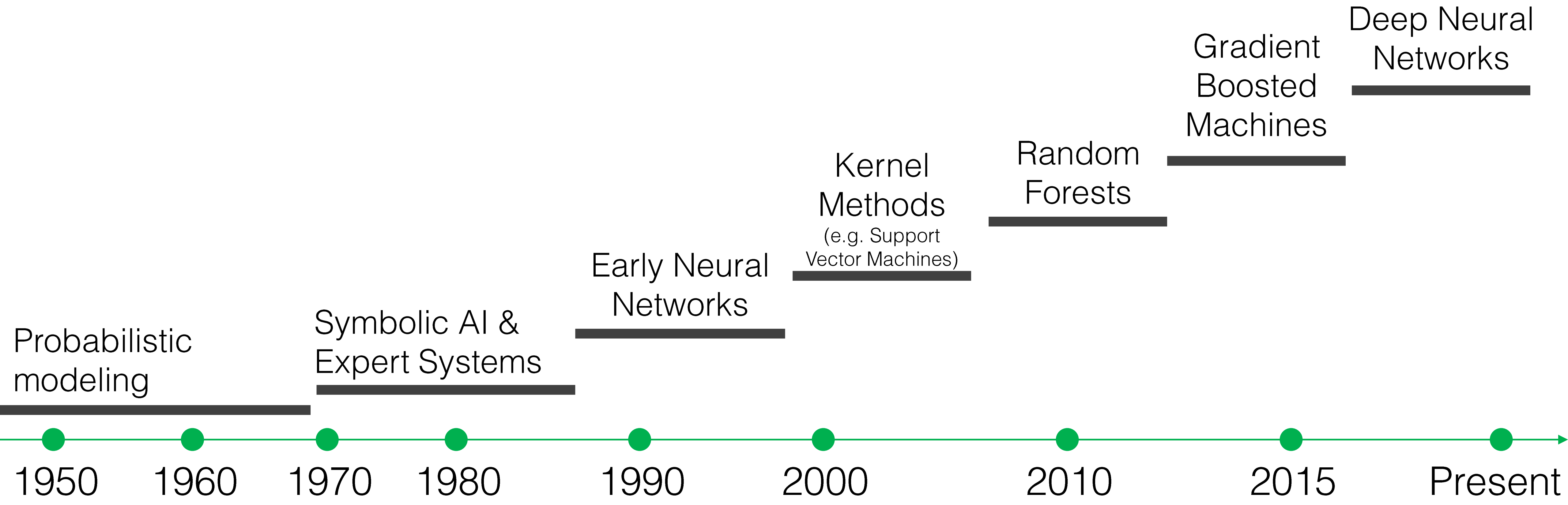


X_{new} **$\hat{y}_{new}$**

→ **1**

→ **0**

Historic Progression of Algorithms



François Chollet, *Deep Learning with Python*, 2017

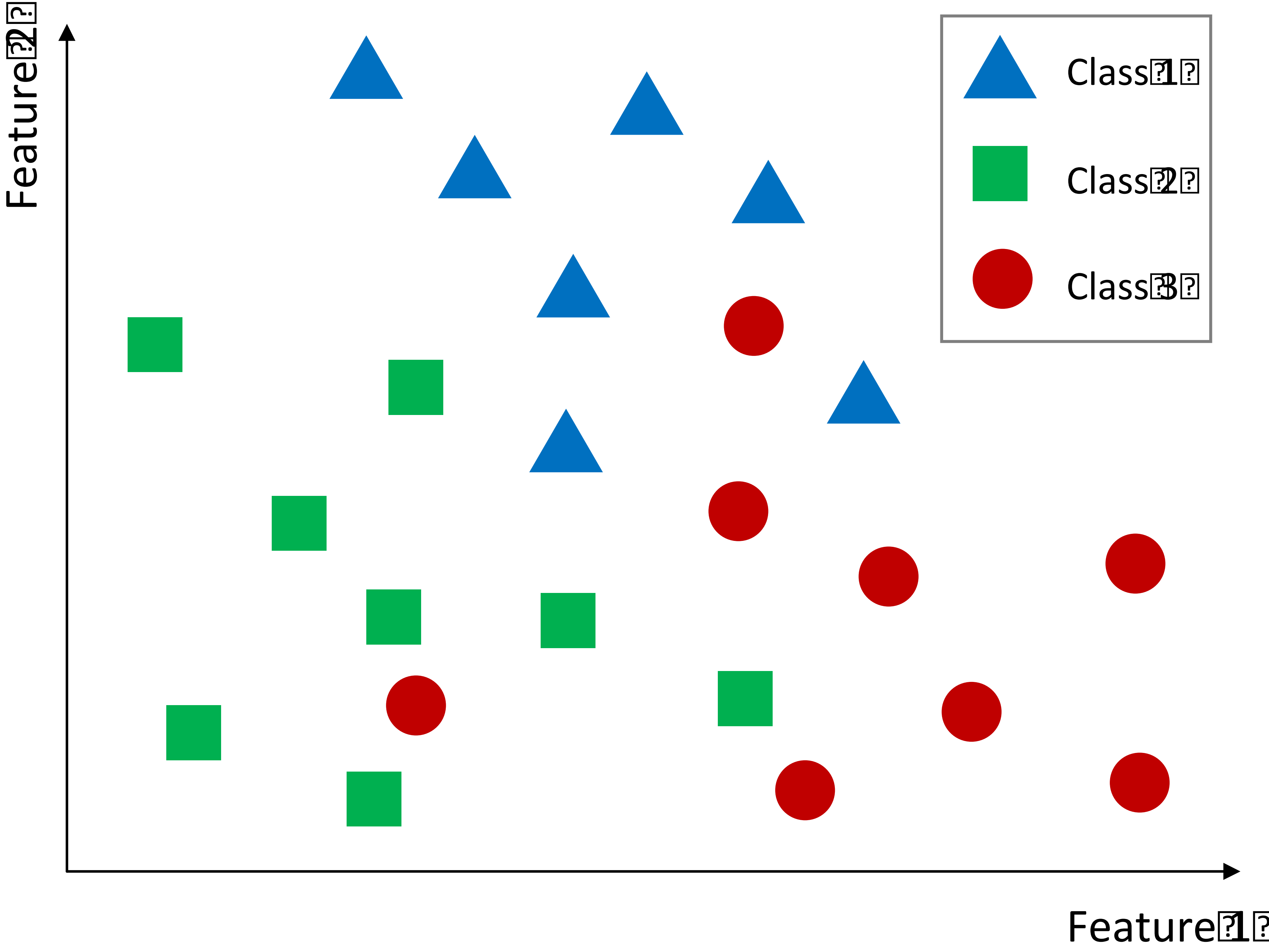
K-Nearest Neighbors

Classification and Regression

K Nearest Neighbor Classifier

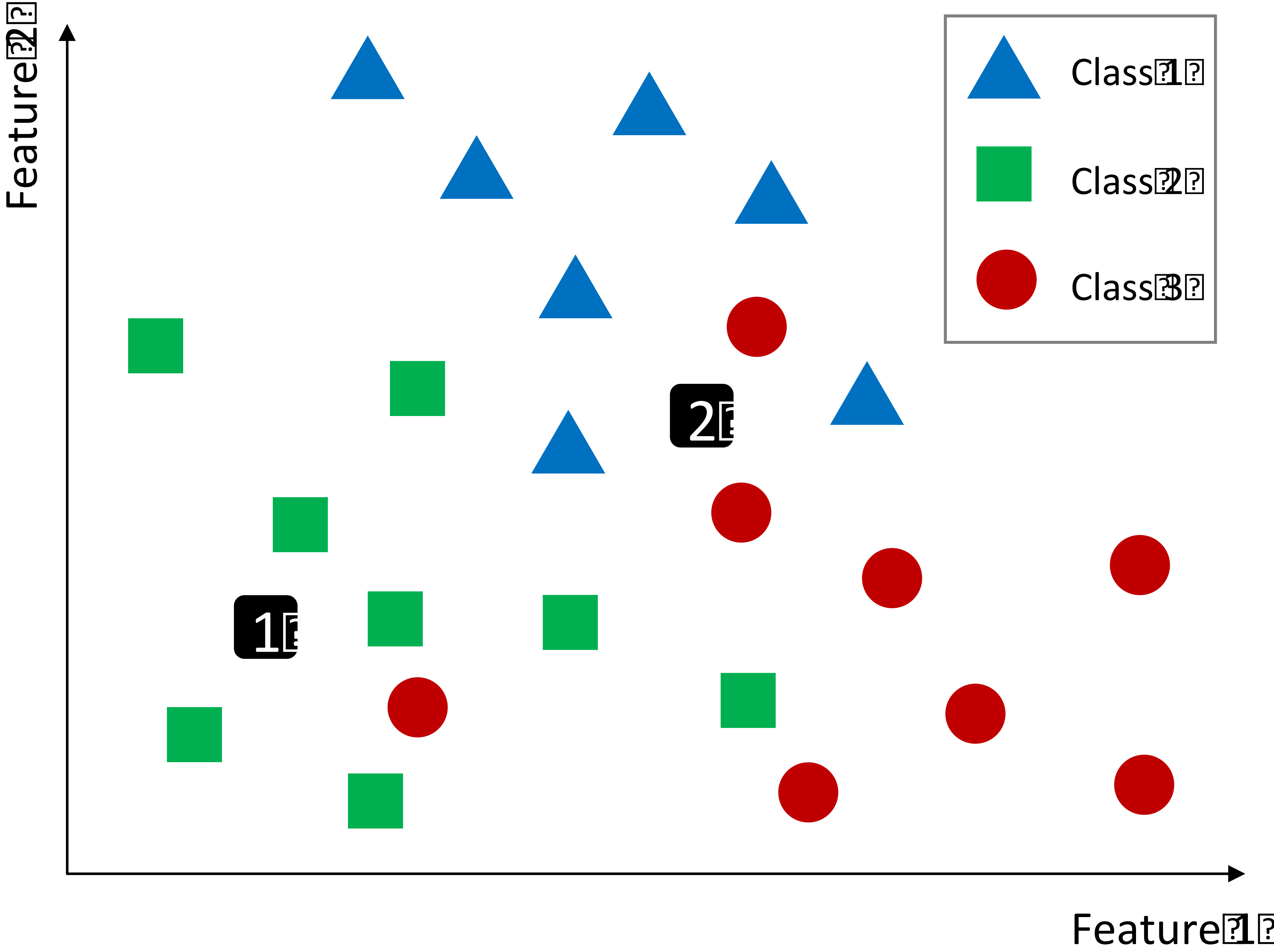
Step 1:
Training

Every new data point is a model parameter

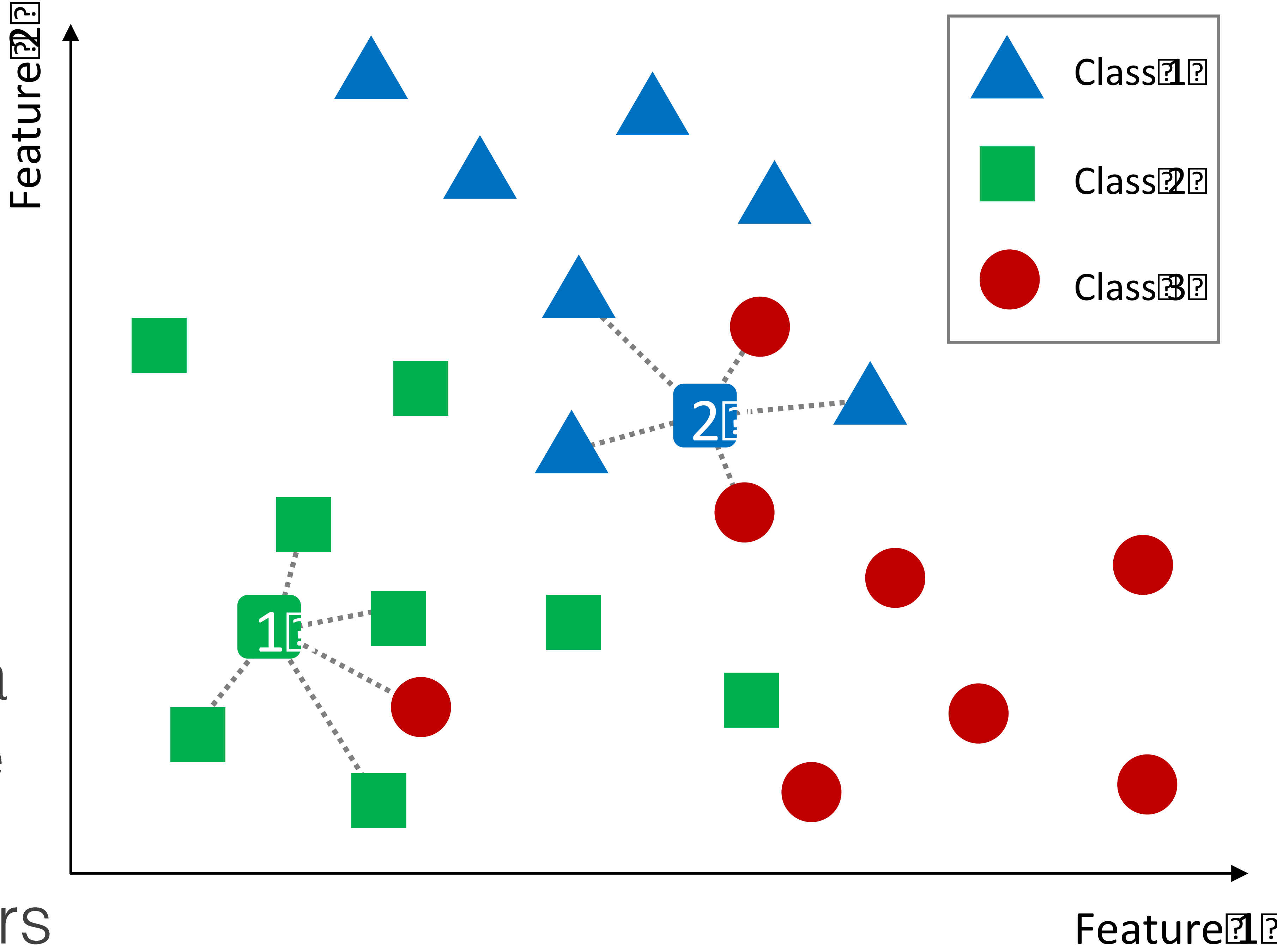


K Nearest Neighbor Classifier

Step 2:
Place new (unseen) examples in the feature space

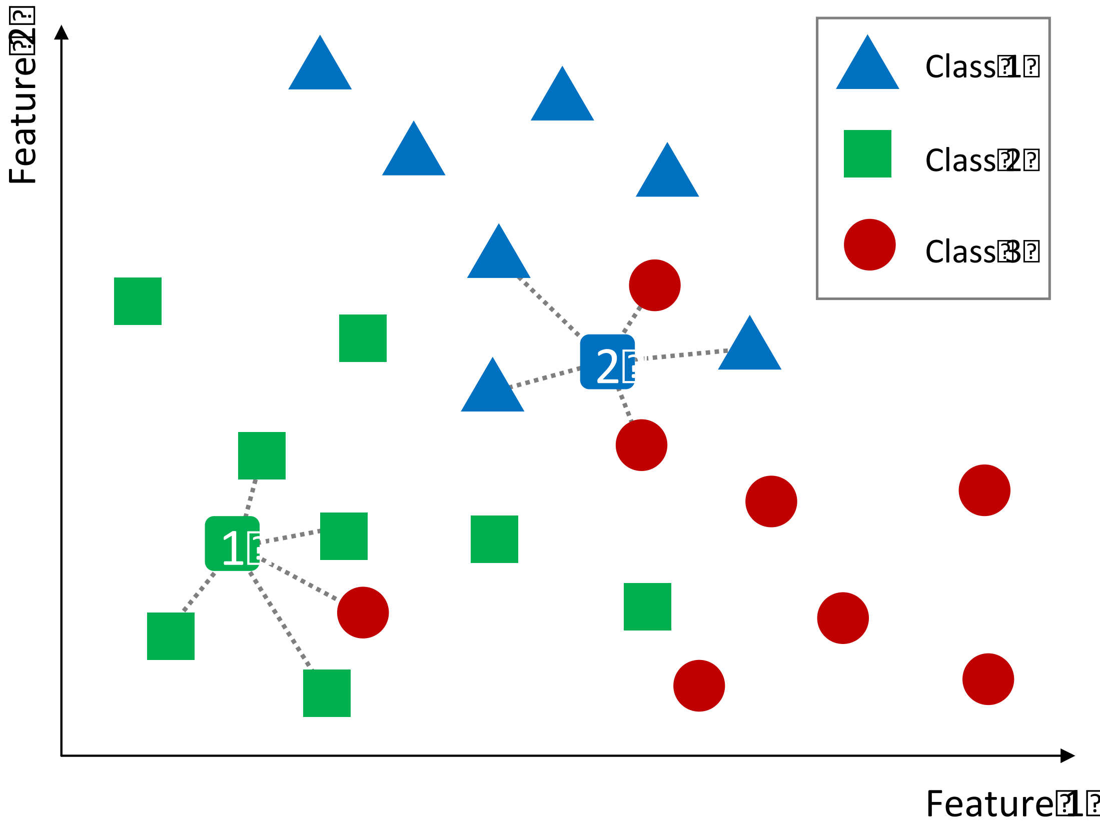


K Nearest Neighbor Classifier



Step 3: Classify the data by assigning the class of the k nearest neighbors

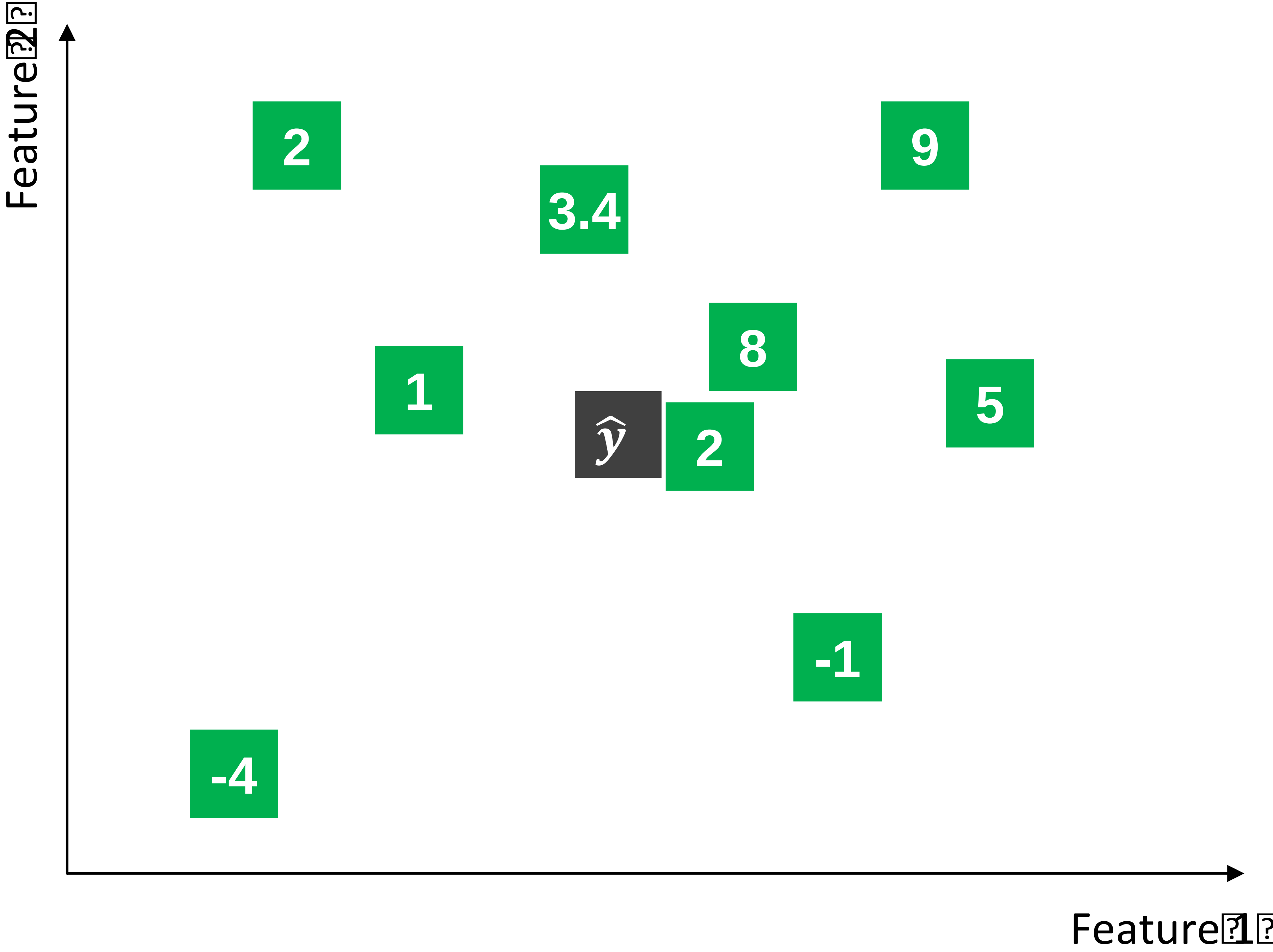
K Nearest Neighbor Classifier



Score vs Decision :
For 5-NN, the confidence score that a sample belongs to a class could be: {0,1/5,2/5,3/5,4/5,1}

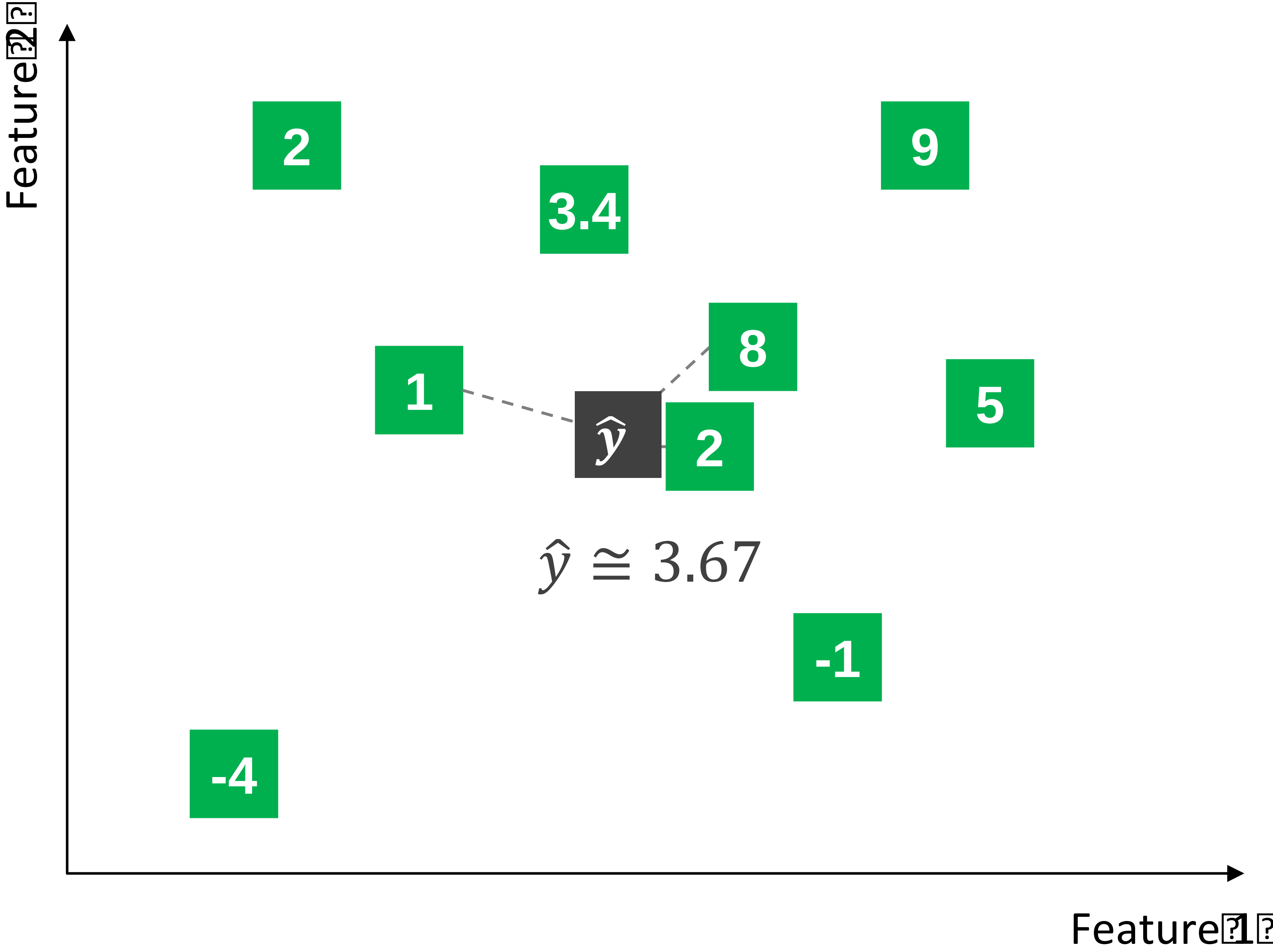
Decision Rule:
If the confidence score for a class > threshold, predict that class

K Nearest Neighbor Regression



K Nearest Neighbor Regression

$$\hat{y} = \frac{1}{k} \sum_{y_i \in \{\text{k nearest}\}} y_i$$



KNN Pros and Cons

Pros

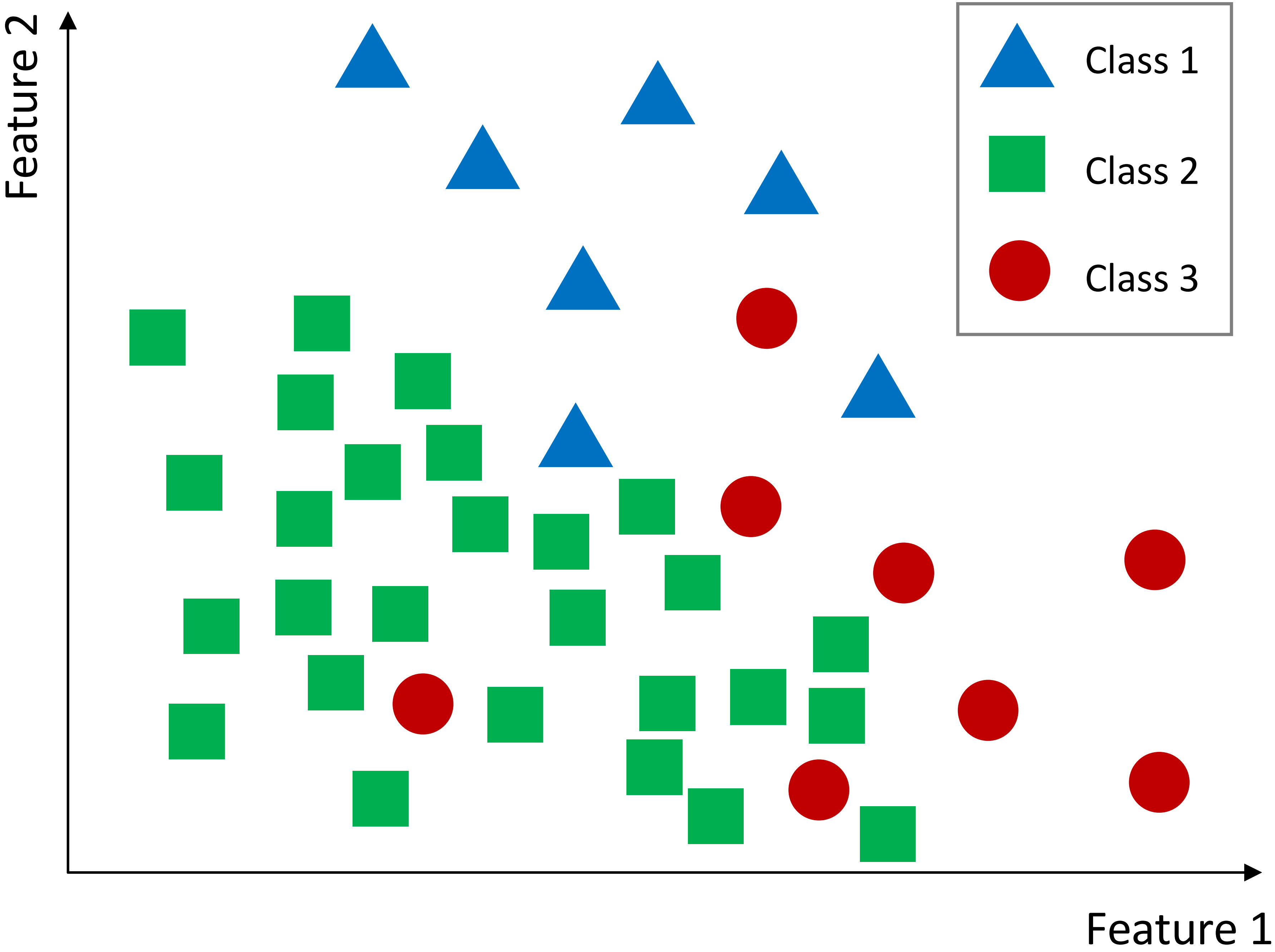
- Simple to implement and interpret
- Minimal training time
- Naturally handles multiclass data

Cons

- Computationally expensive to find nearest neighbors
- Requires **all** of the training data to be stored in the model
- Suffers if classes are imbalanced
- Performance may suffer in high dimensions

K Nearest Neighbor Classifier

What happens if the data aren't balanced?



How flexible should my model be?

the bias-variance tradeoff and learning to generalize

bias
consistently incorrect prediction

error from poor model assumptions
(high bias results in underfit)

variance
inconsistent prediction

error from sensitivity to small changes in the training data
(high variance results in overfit)

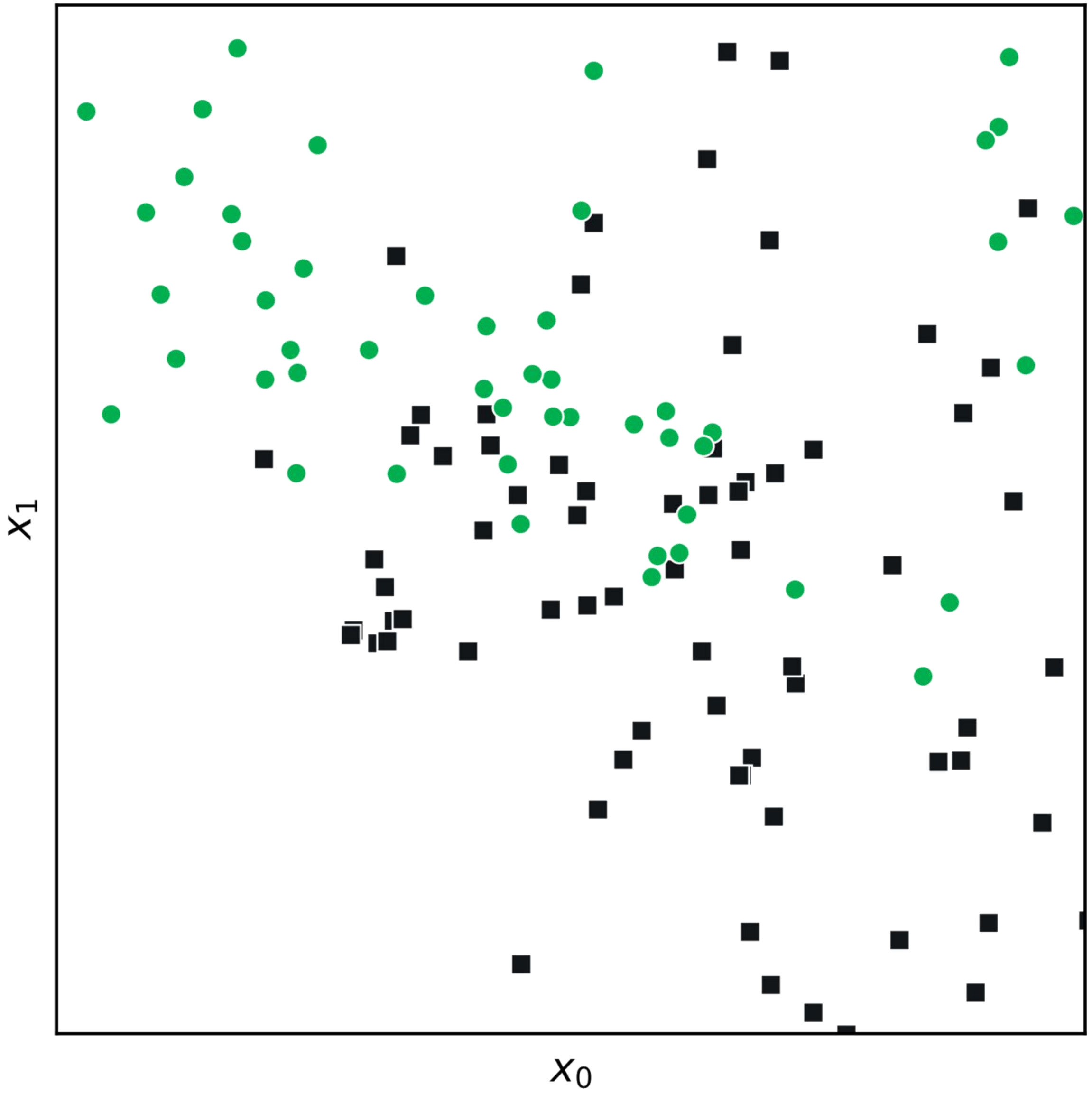
noise
lower bound on generalization error

irreducible error inherent to the problem
(e.g. you cannot predict the outcome of a flip of a fair coin any more than 50% of the time)

Bias-Variance Tradeoff

generalization error = bias^2 + variance + noise

Classification feature space



What’s the lowest classification error we can achieve for binary classification?

If we fully know the probability distribution of the data…

The Bayes decision rule

Bayes' Rule

$$\underbrace{P(C|\boldsymbol{X})}_{\text{Posterior}} = \frac{\overbrace{P(\boldsymbol{X}|C)}^{\text{Likelihood}}\overbrace{P(C)}^{\text{Prior}}}{\underbrace{P(\boldsymbol{X})}_{\text{Evidence}}}$$

$\boldsymbol{X}$ Features

C Class label

i.e. $C \in \{c_0, c_1\}$ for the binary case

Bayes' Decision Rule:

choose the most probable class given the data

If $P(C_i = c_1|\boldsymbol{X}_i) > P(C_i = c_0|\boldsymbol{X}_i)$ then $\hat{y} = c_1$

otherwise $\hat{y} = c_0$

- If the distributions are correct, this decision rule is **optimal**
- Rarely do we have enough information to use this in practice

Bayes' Rule Biased Coin Example

Two types of coins:

c_0 Coin with probability of heads: 0.5 (fair)

c_1 Coin with probability of heads: 0.7 (unfair)

$$\underbrace{P(C|\boldsymbol{X})}_{\text{Posterior}} = \frac{\overbrace{P(\boldsymbol{X}|C)}^{\text{Likelihood}}\overbrace{P(C)}^{\text{Prior}}}{\underbrace{P(\boldsymbol{X})}_{\text{Evidence}}}$$

You want to classify a coin as fair or unfair

You draw a coin from a bag that has 50/50 fair/unfair coins

$$P(c_0) = P(c_1) = 0.5$$

You flip the coin 5 times and it lands on heads 5 times ($\mathbf{X} = \{\text{flip 5 heads}\}$)

$$P(\boldsymbol{X}|c_0) = (0.5)^5 \approx 0.03$$

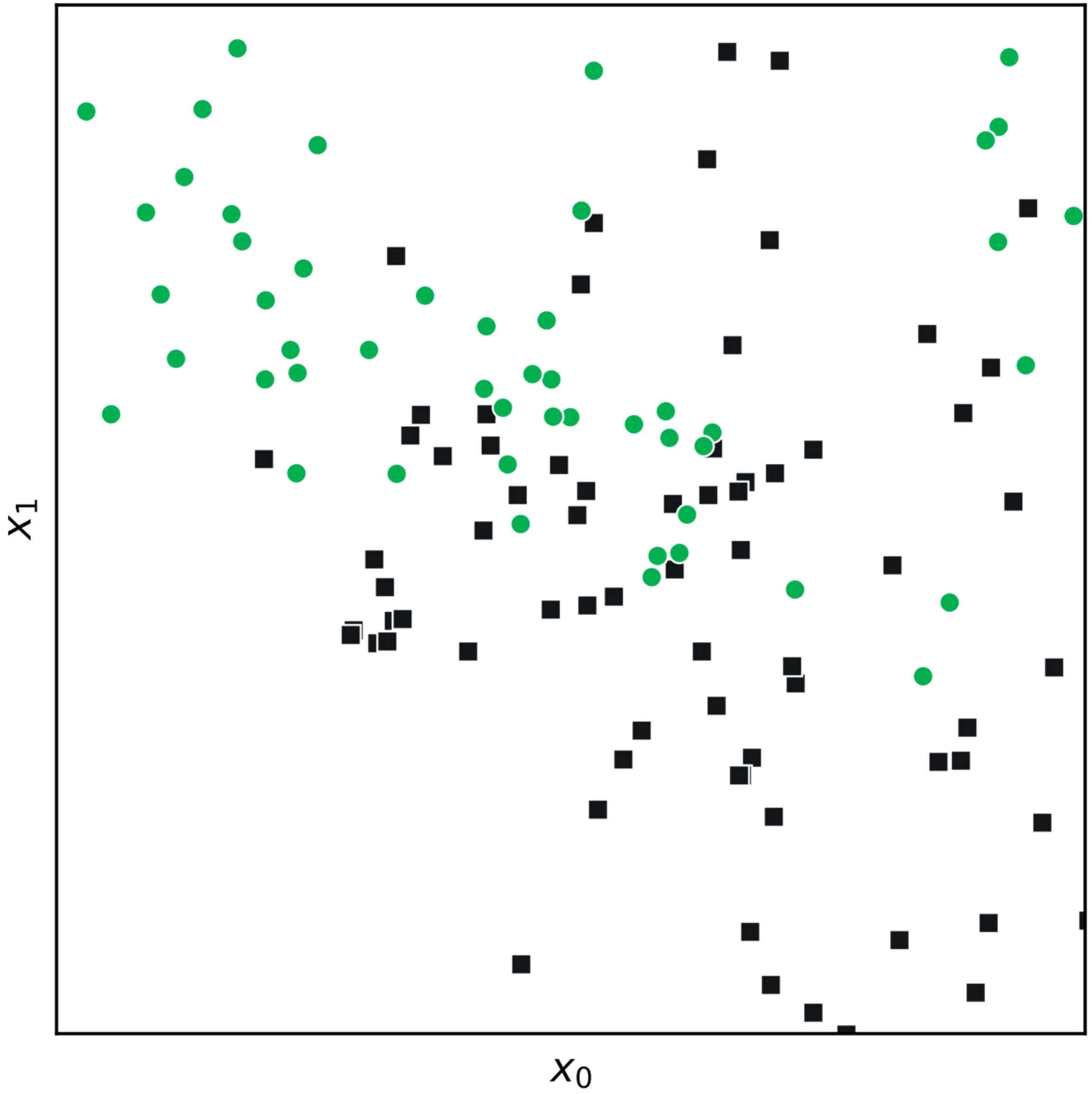
$$P(\boldsymbol{X}|c_1) = (0.7)^5 \approx 0.17$$

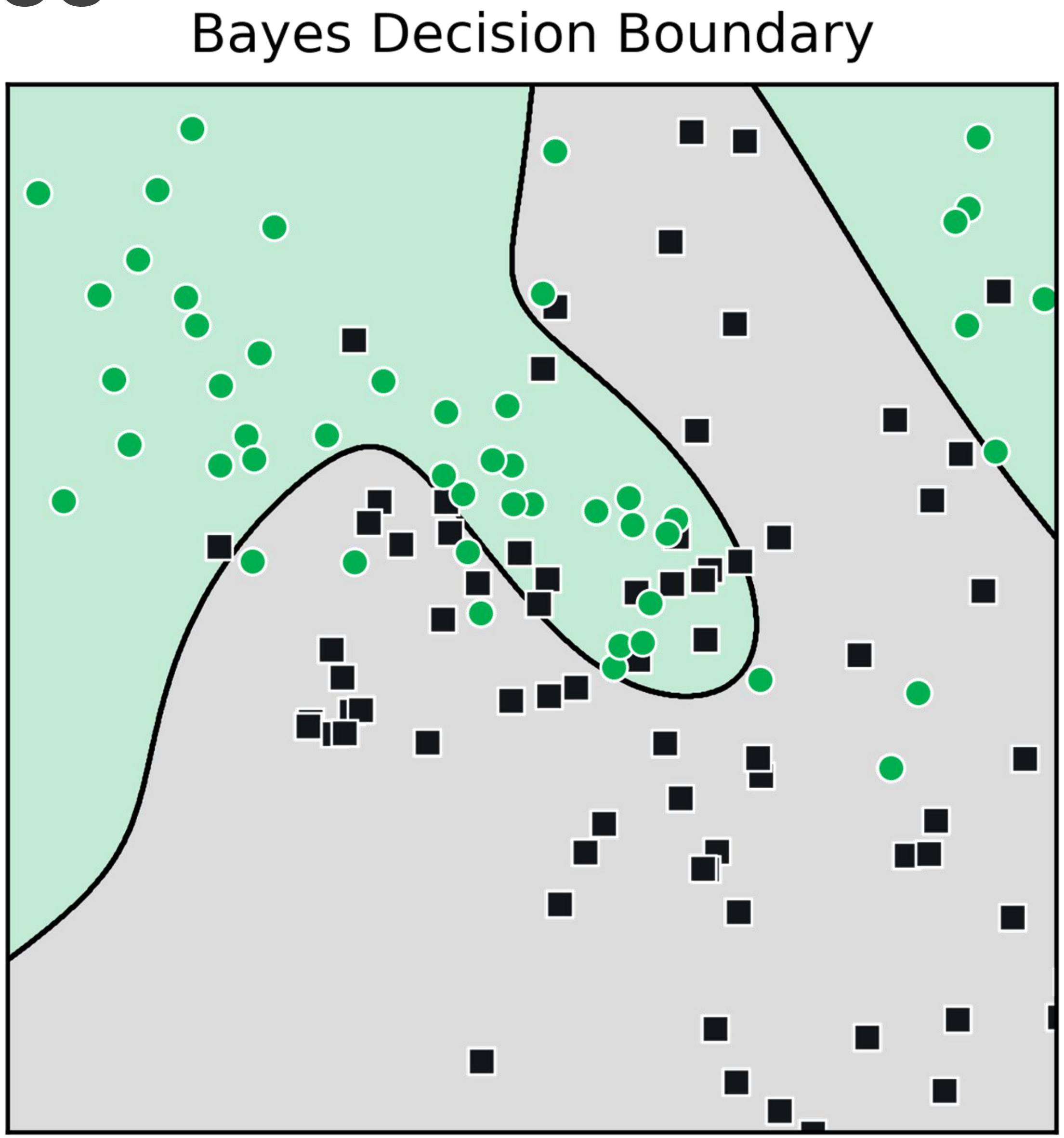
Decision rule: if $P(c_1|\boldsymbol{X}) > P(c_0|\boldsymbol{X})$, then the coin is unfair, otherwise the coin is fair

$$\frac{P(\boldsymbol{X}|c_1)P(c_1)}{P(\boldsymbol{X})} > \frac{P(\boldsymbol{X}|c_0)P(c_0)}{P(\boldsymbol{X})} \rightarrow \frac{(0.17)(0.5)}{\cancel{P(\boldsymbol{X})}} > \frac{(0.03)(0.5)}{\cancel{P(\boldsymbol{X})}}$$

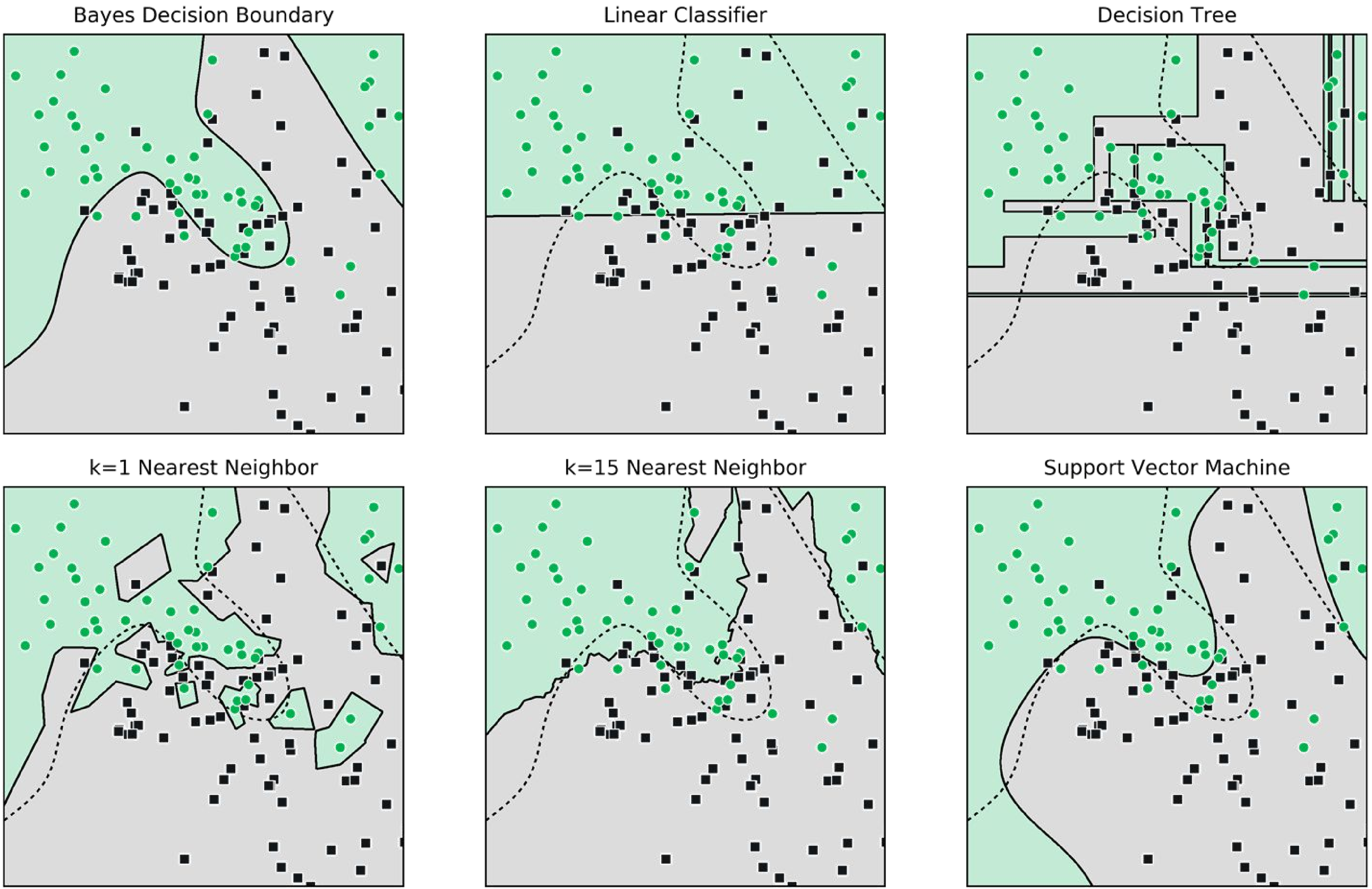
$\rightarrow$ We predict the coin is **unfair**

Classification feature space





Decision Boundary Examples



Bias Variance Tradeoff

higher bias

underfit

higher variance

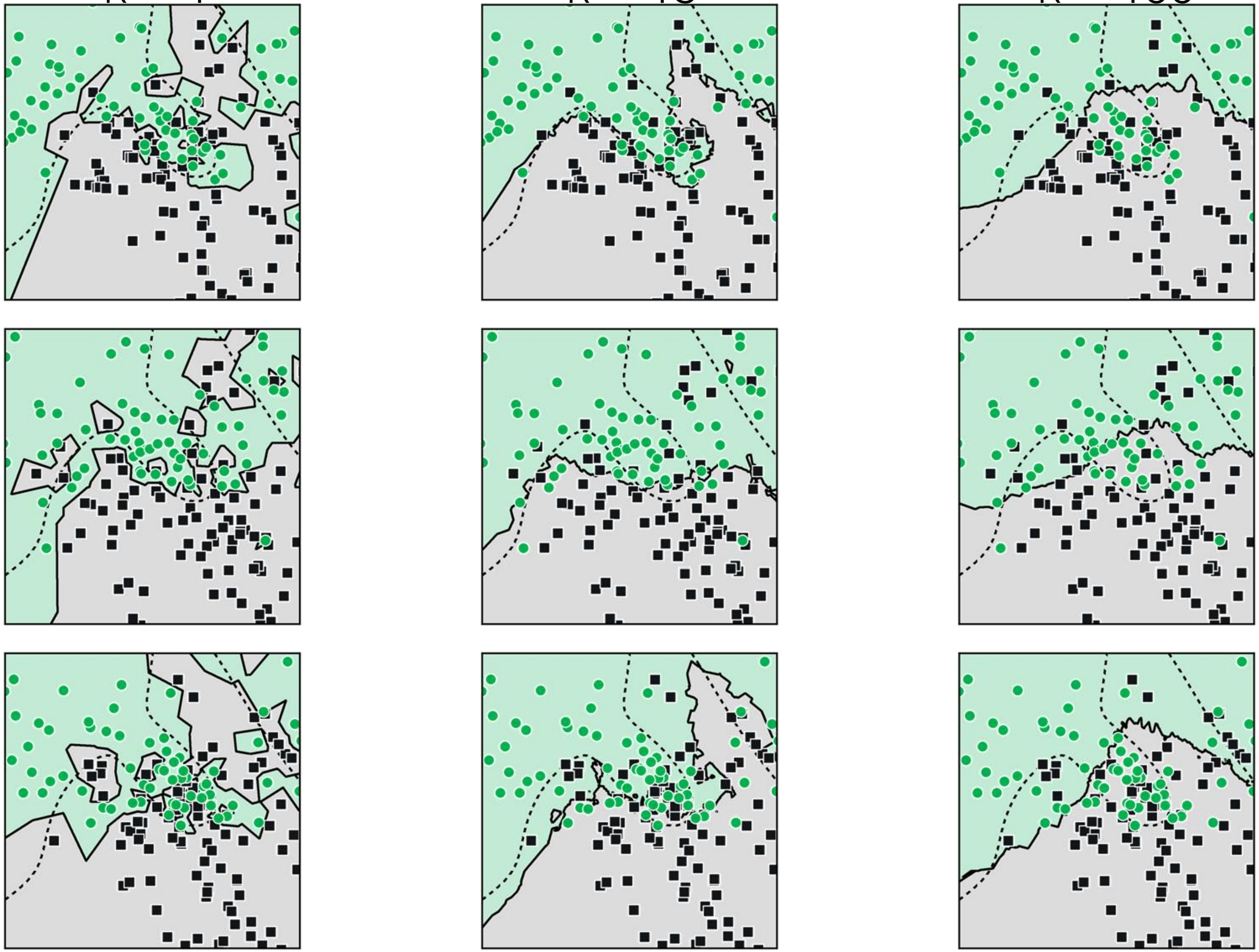
overfit

k = 1 | k = 15 | k = 100

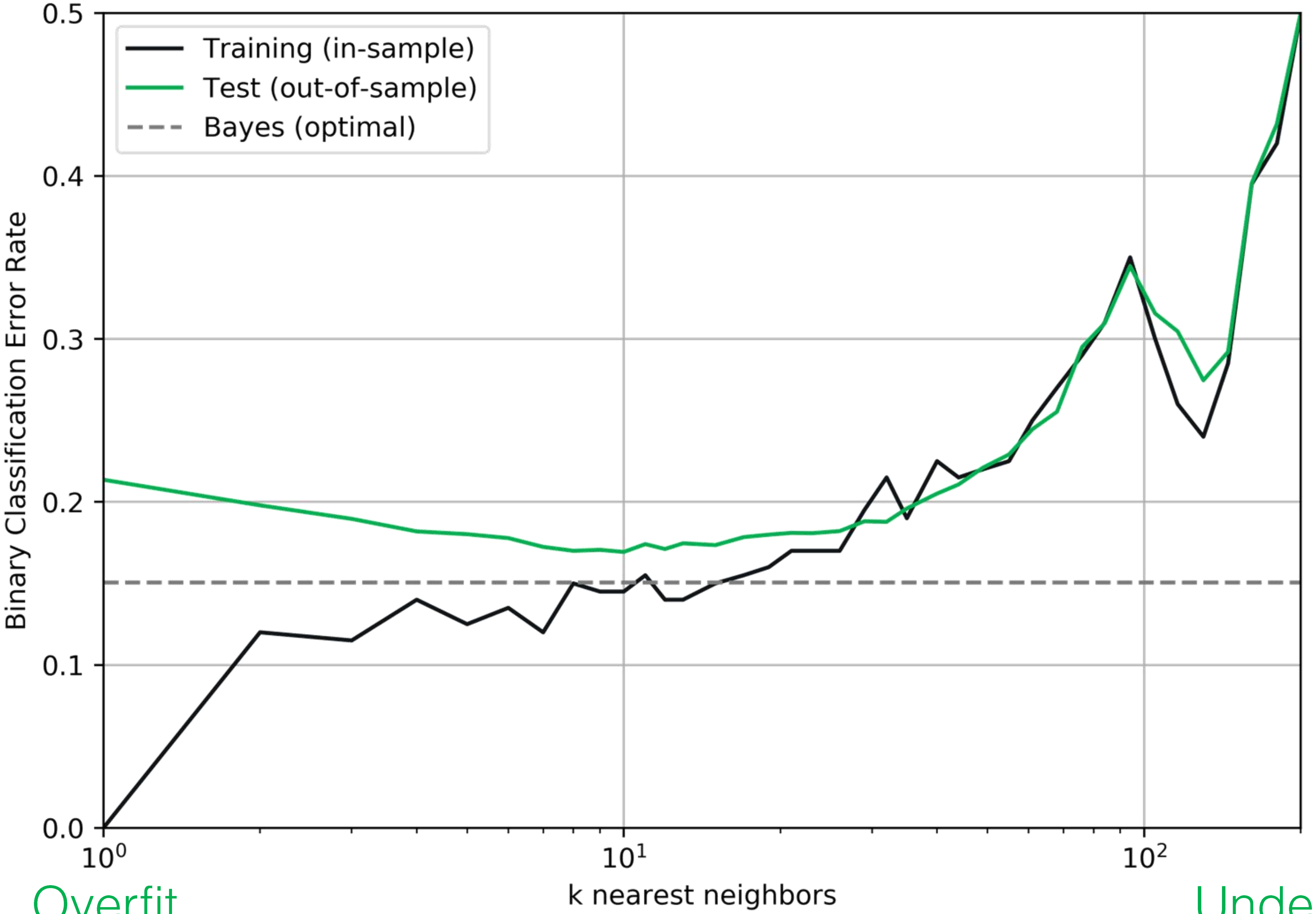
Sample 1

Sample 2

Sample 3



Bias Variance Tradeoff



Overfit
High Variance
Low Bias

more flexible

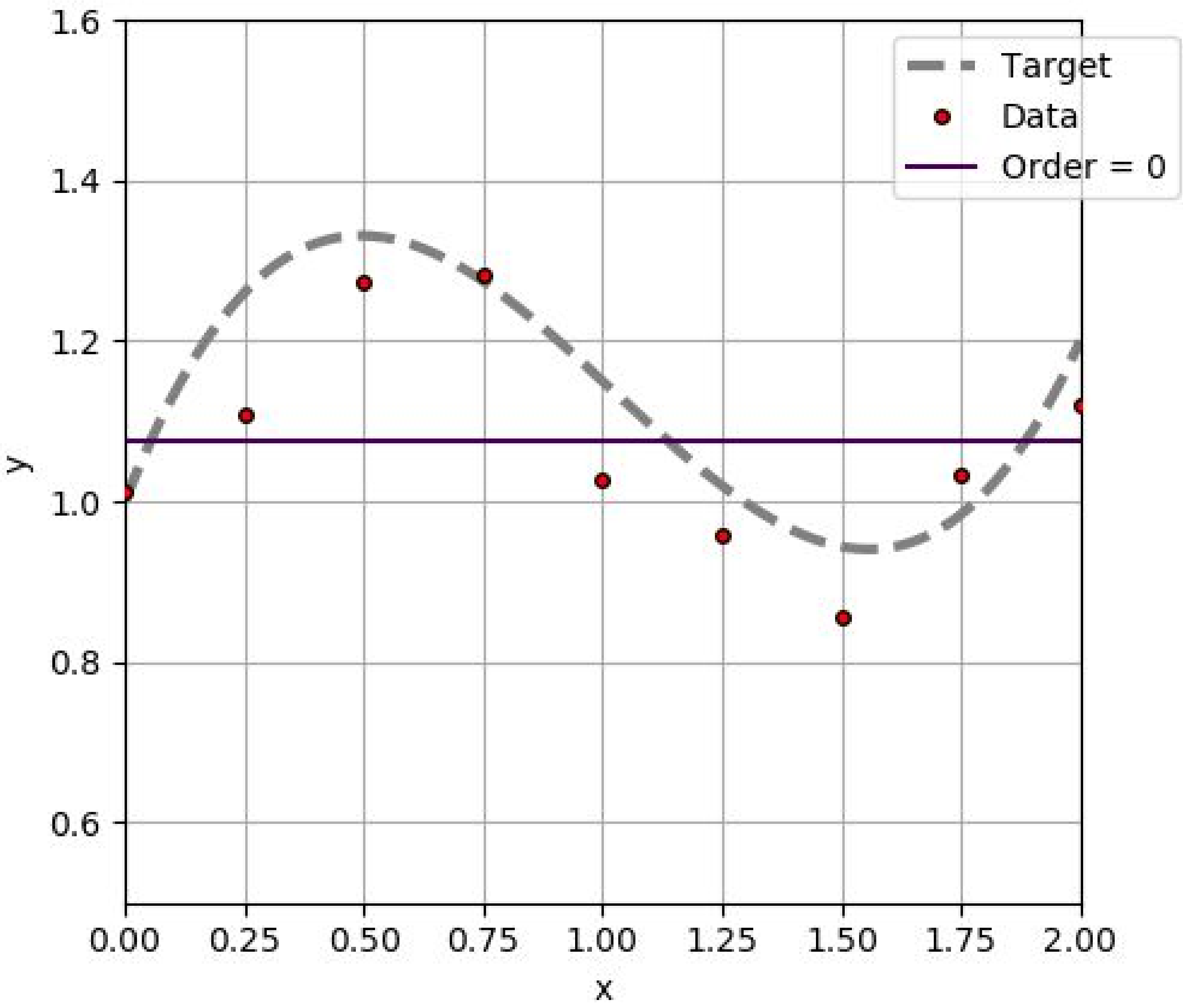
Underfit
Low Variance
High Bias

This tradeoff is equally challenging for regression

Regression

$$\hat{y}_i = \sum_{j=0}^{m} a_j x_i^j$$

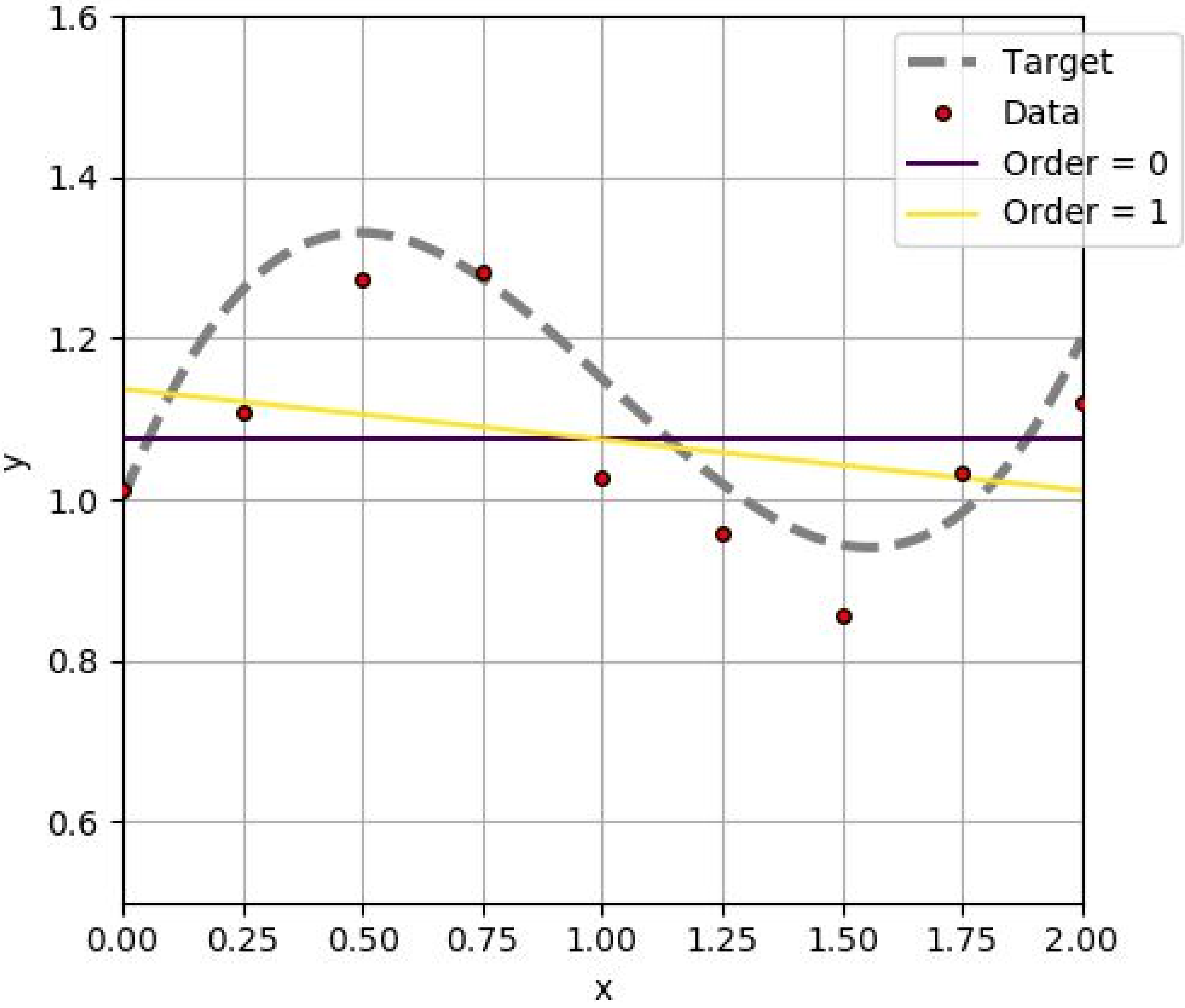
m is the model order



Regression

$$\hat{y}_i = \sum_{j=0}^{m} a_j x_i^j$$

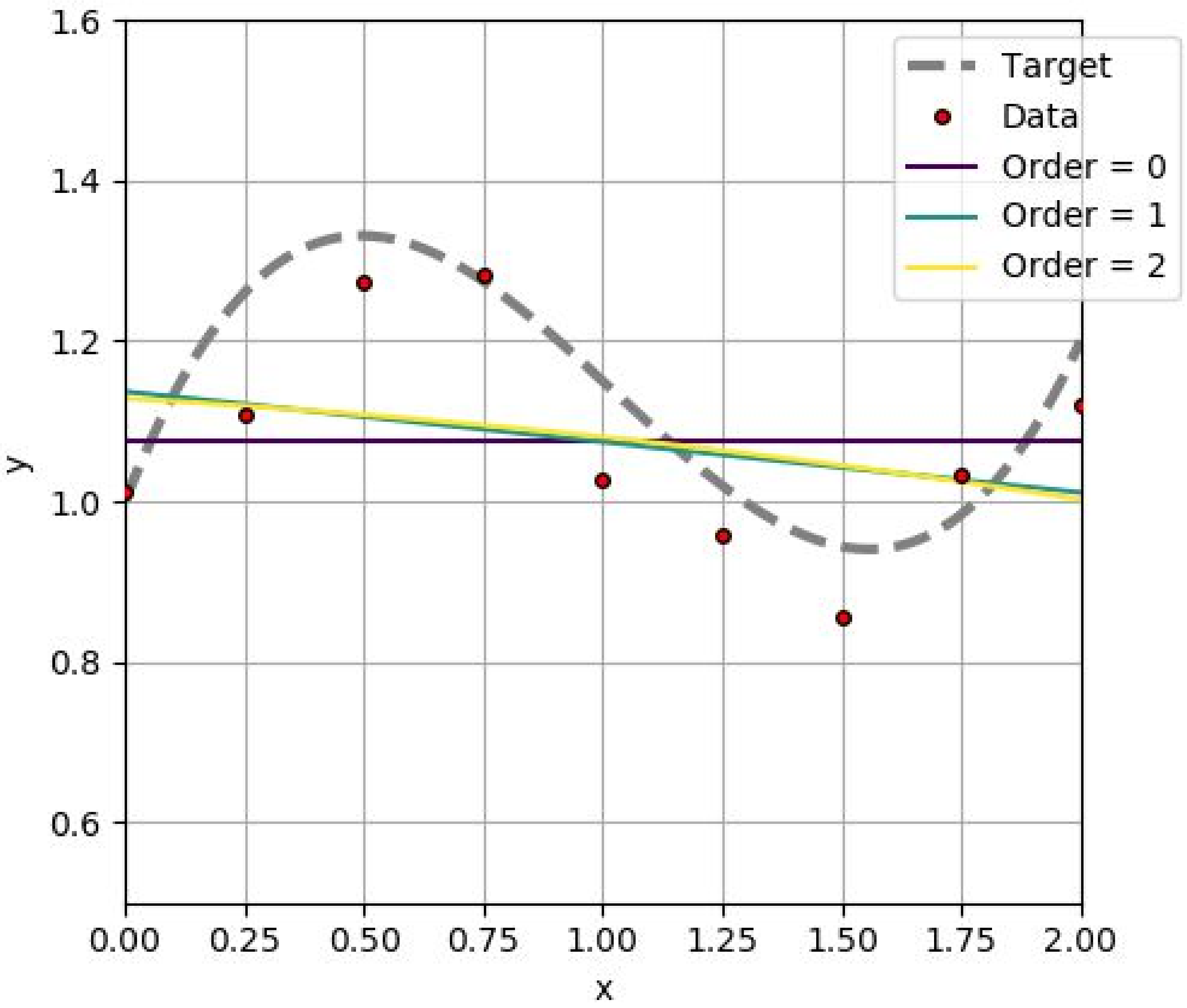
m is the model order



Regression

$$\hat{y}_i = \sum_{j=0}^{m} a_j x_i^j$$

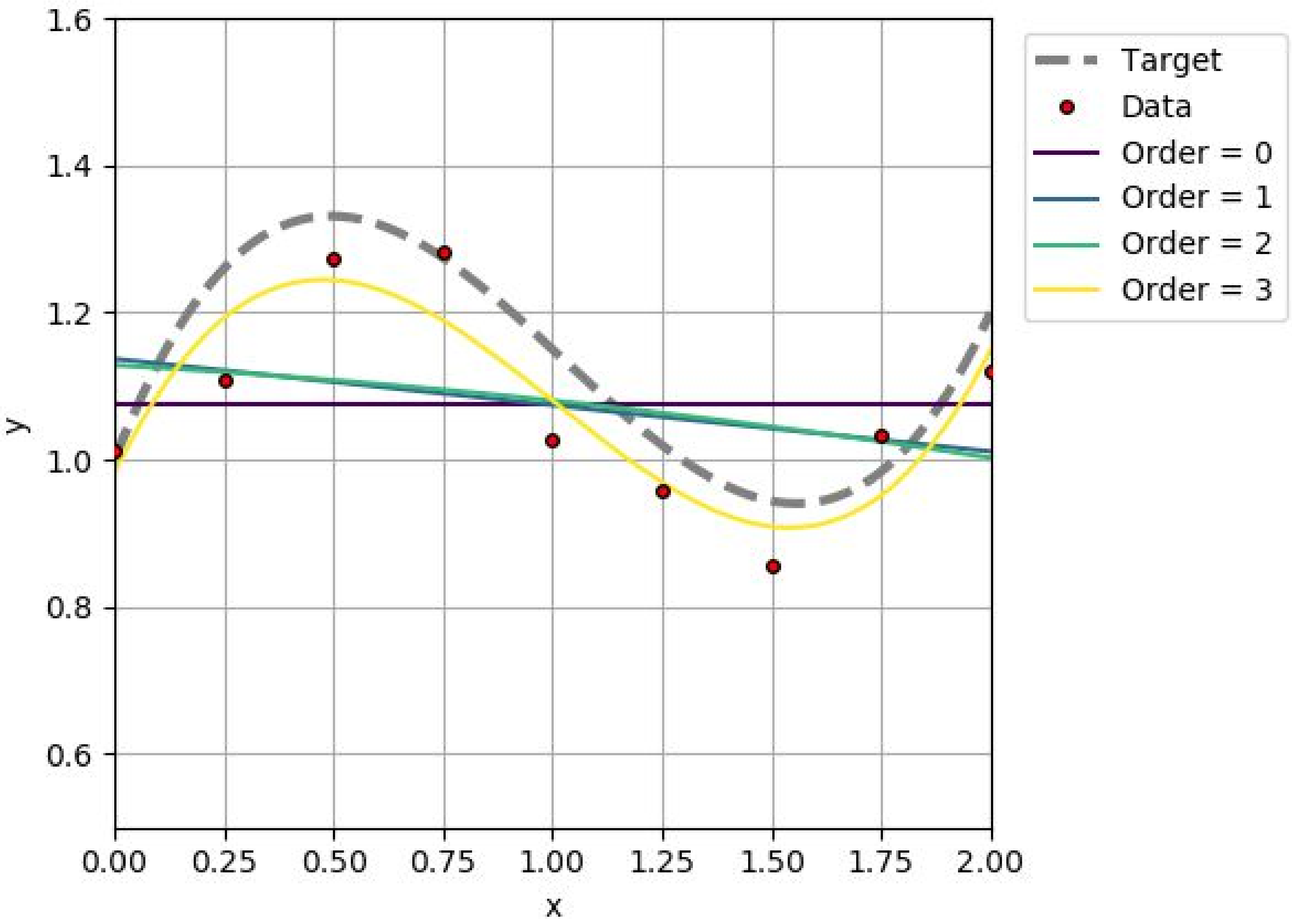
m is the model order



Regression

$$\hat{y}_i = \sum_{j=0}^{m} a_j x_i^j$$

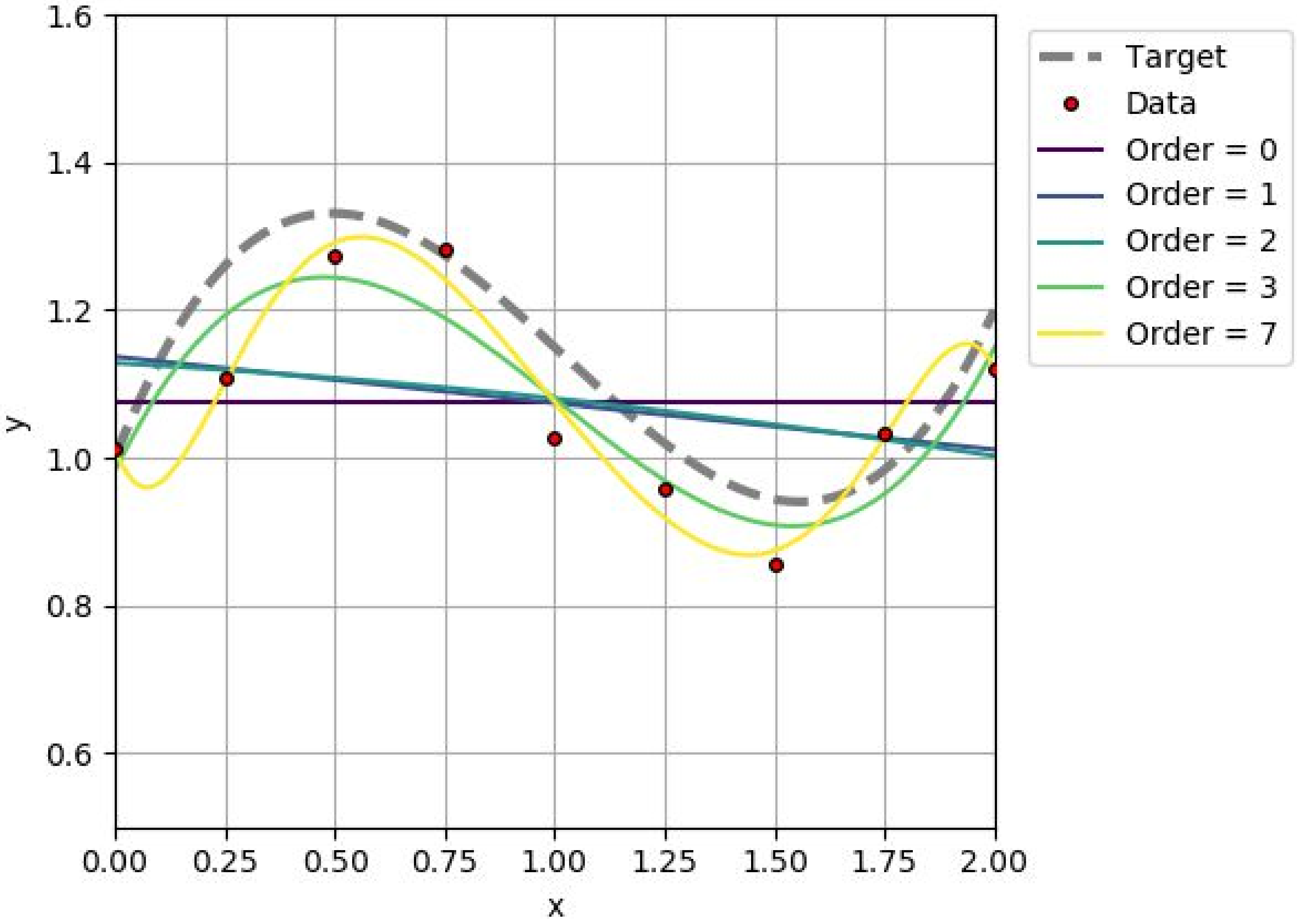
m is the model order



Regression

$$\hat{y}_i = \sum_{j=0}^{m} a_j x_i^j$$

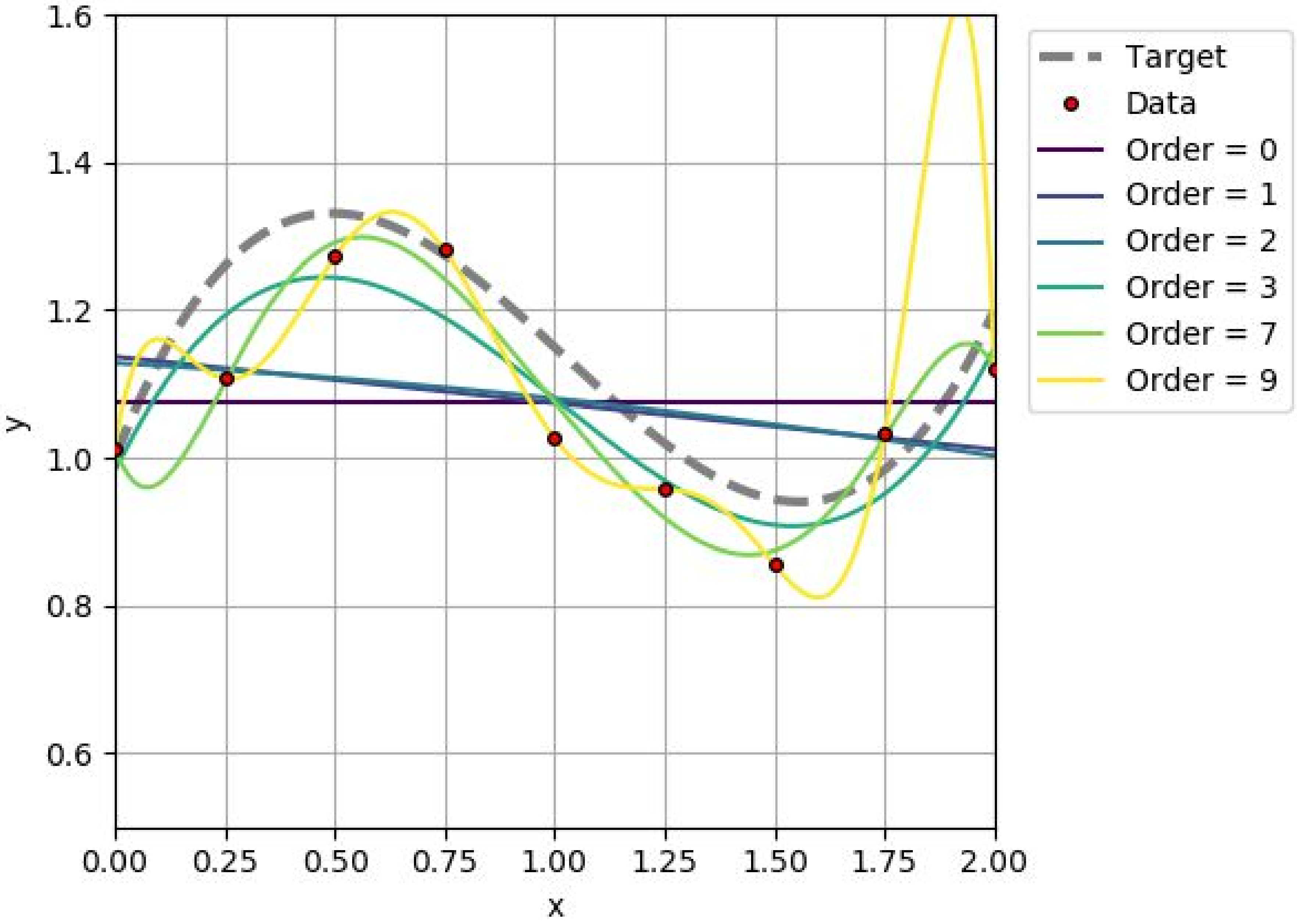
m is the model order



Regression

$$\hat{y}_i = \sum_{j=0}^{m} a_j x_i^j$$

m is the model order



Problem

Too much flexibility leads to **overfit**

Too little flexibility leads to **underfit**

Over/underfit **hurts generalization** performance

Solutions for overfitting

1. Add **more data** for training
2. Constrain model flexibility through **regularization**
3. Use model **ensembles**