Lecture 6:
Learning, Bias & Variance

Olexandr Isayev

Department of Chemistry, CMU

olexandr@cmu.edu

HW2: Clustering

1. Load the data. (We provide you with CSV file in Canvas)

2. Perform k-means clustering. Determine the optimal number of clusters using e.g. elbow method.

3. Perform clustering with different clustering method as implemented in Scikit Learn.

4. Now, try clustering with another distance metric (e.g. Cosine, Jaccard, etc). Hint: Think whether the default distance metric is appropriate for your data or not.

5. Visualize results using the dimensionality reduction (UMAP or tSNE) technique with respect to the cluster labels.

6. Compare clustering results. Try to rationalize observed commonalities or differences with respect to clustering methods and distance metrics used.

In class project

Please submit information about your projects:

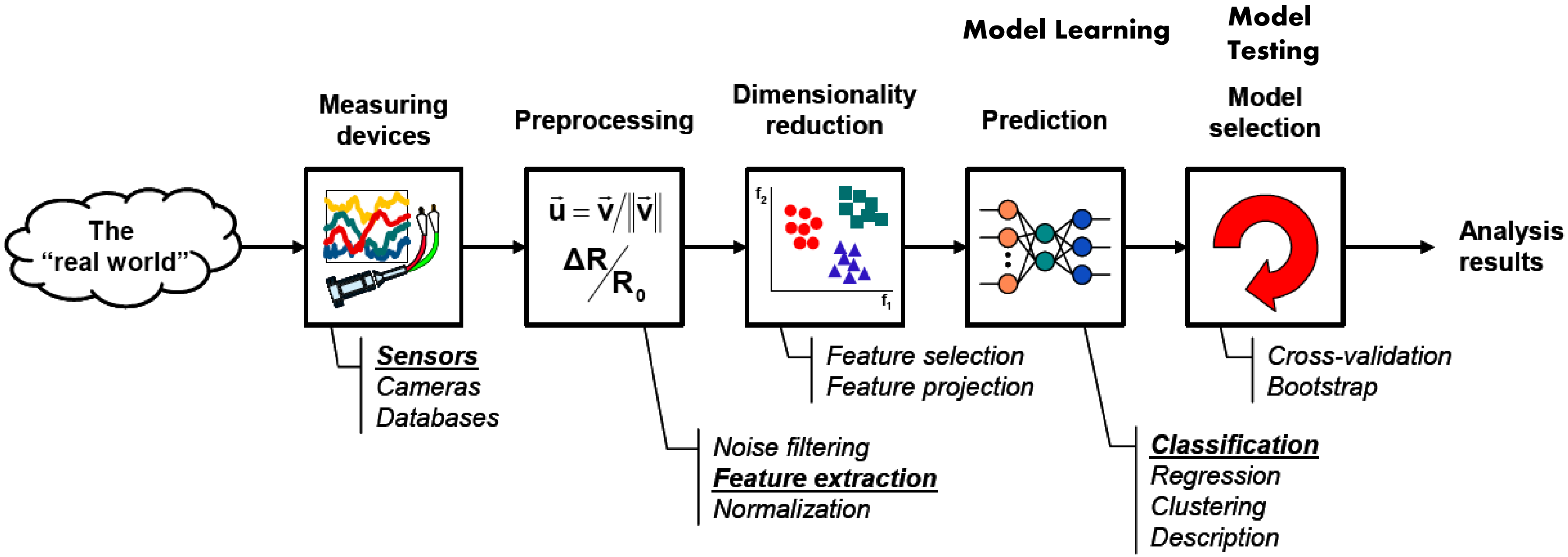
- Title
- Names of team members
- Briefly describe the problem you are trying to solve and rough size of your dataset.

Deadline: Sunday, October 8

Machine Learning Problems

	Supervised Learning	***Unsupervised Learning***
Discrete	classification or categorization	clustering
Continuous	regression	dimensionality reduction

The Learning Process



Model Learning

Model Testing

Measuring devices

Preprocessing

Dimensionality reduction

Prediction

Model selection

Analysis results

Sensors
Cameras
Databases

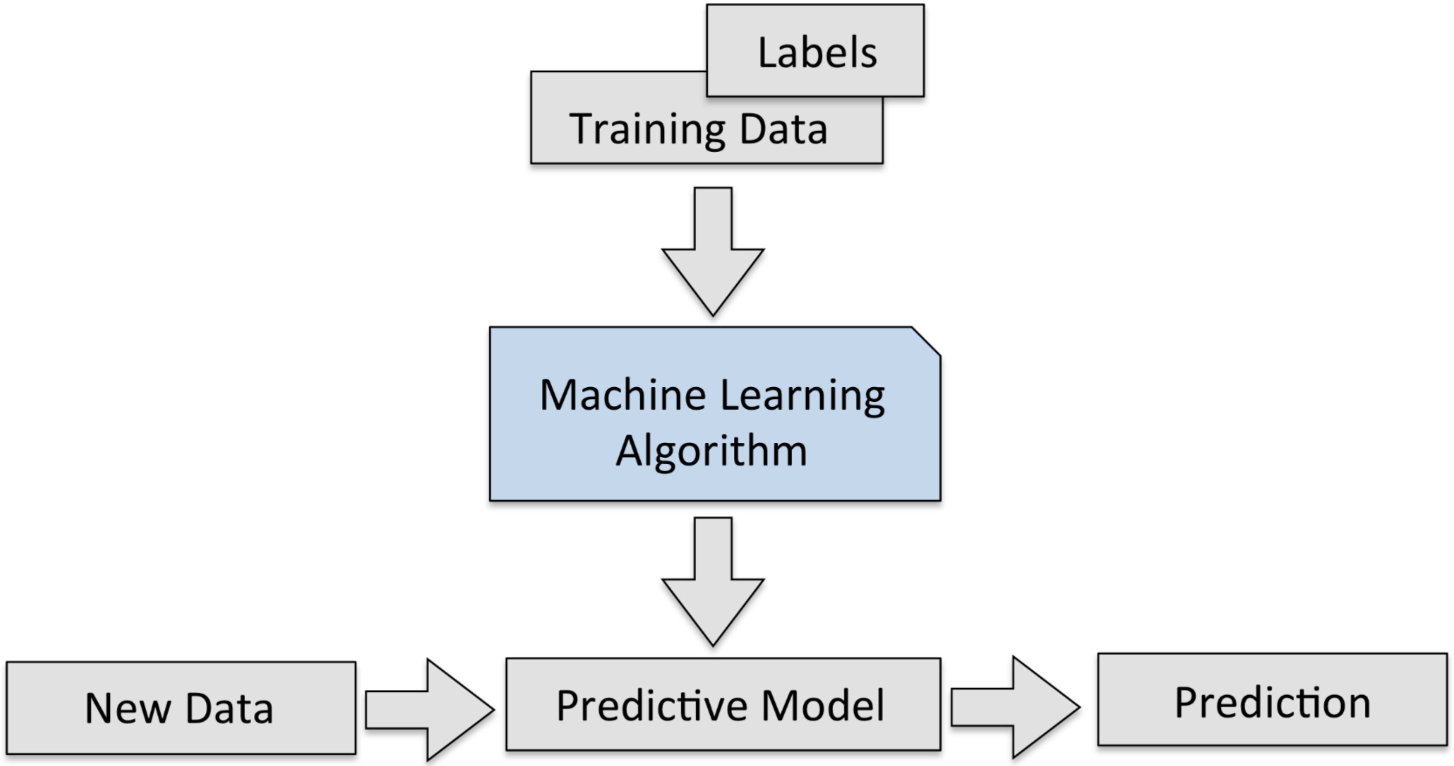
Noise filtering
Feature extraction
Normalization

Feature selection
Feature projection

Classification
Regression
Clustering
Description

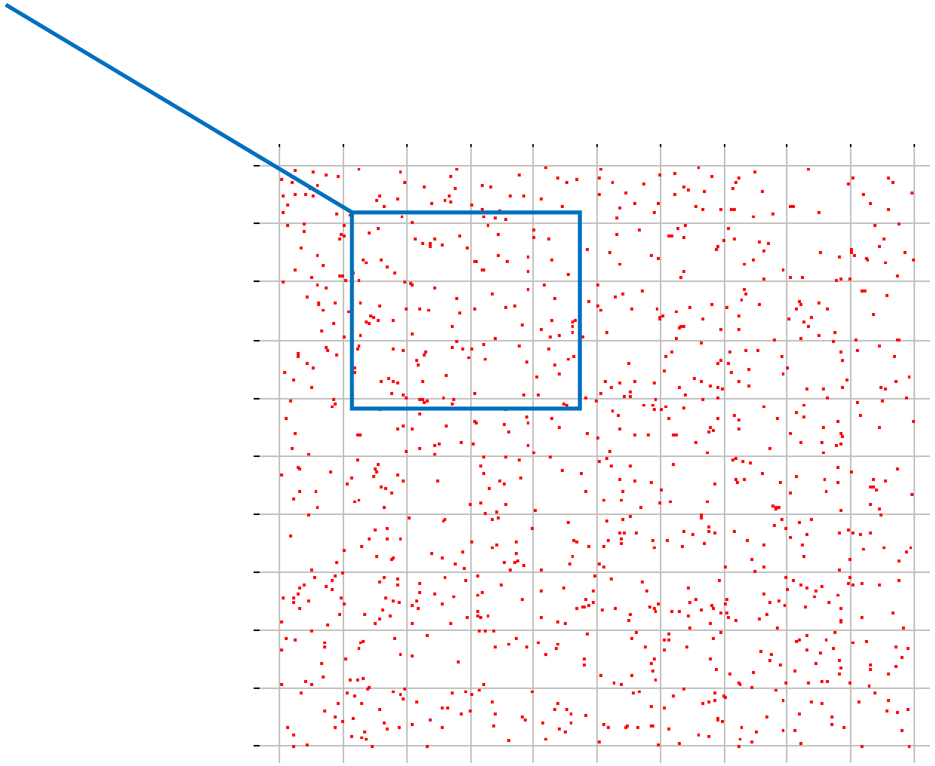
Cross-validation
Bootstrap

Making predictions about the future with supervised learning



Predicting from Samples

- Most datasets are **samples** from an **infinite population**.
- We are most interested in **models of the population**, but we have access only to a **sample** of it.



For datasets consisting of (X,y)

- features X + label y

a model is a prediction y = f(X)

We train on a training sample D and we denote the model as $f_D(X)$

Bias and Variance

Our data-generated model $f_D(X)$ is a **statistical estimate** of the true function $f(X)$.

Because of this, its subject to bias and variance:

Bias: if we train models $f_D(X)$ on many training sets D, bias is the expected difference between their predictions and the true y's.

i.e.
$$Bias = \mathrm{E}[f_D(X) - y]$$

E[] is taken over points X and datasets D

Variance: if we train models $f_D(X)$ on many training sets D, variance is the variance of the estimates:

$$Variance\ = \mathrm{E}\left[\left(f_D(X) - \bar{f}(X)\right)^2\right]$$

Where $\bar{f}(X) = \mathrm{E}[f_D(X)]$ is the average prediction on X.

Bias and Variance Tradeoff

There is usually a bias-variance tradeoff caused by model complexity.

Complex models (many parameters) usually have lower bias, but higher variance.

Simple models (few parameters) have higher bias, but lower variance.

Bias and Variance Tradeoff

The total expected error is

$$Bias^2 + Variance$$

Because of the bias-variance trade-off, we want to **balance** these two contributions.

If $Variance$ strongly dominates, it means there is too much variation between models. This is called **over-fitting**.

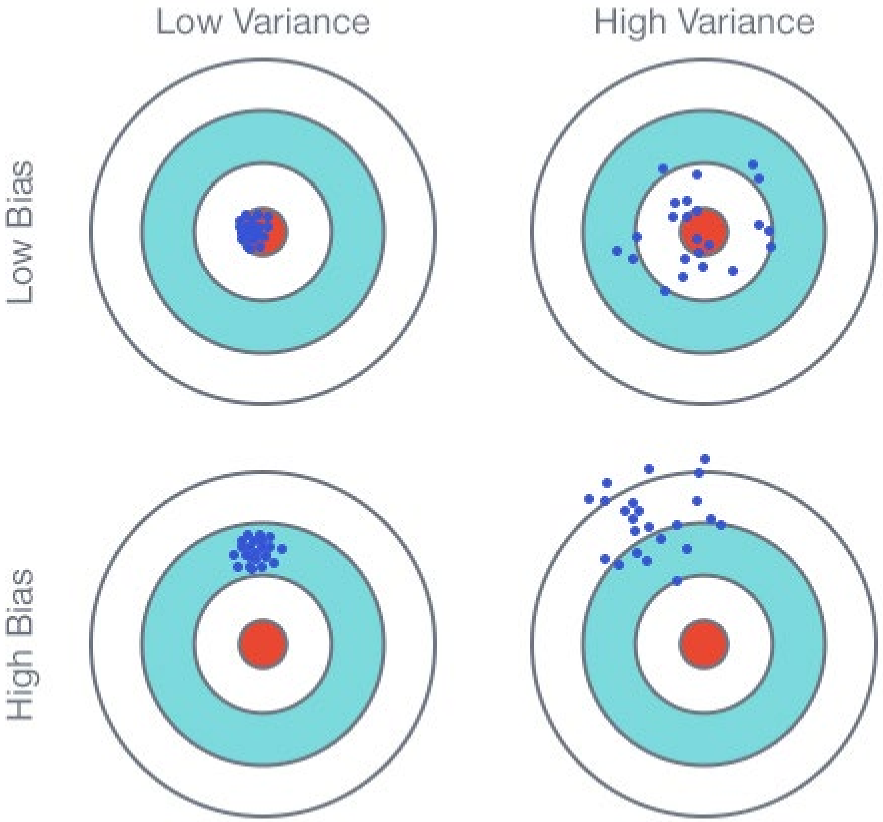
If $Bias$ strongly dominates, then the models are not fitting the data well enough. This is called **under-fitting**.

Bias and Variance

Bias and variance can be visualized with a classic example of a dartboard. We have four different dart throwers, each with different combinations of low/high bias and low/high variance. We represent the locations of each of their dart throws as blue dots:

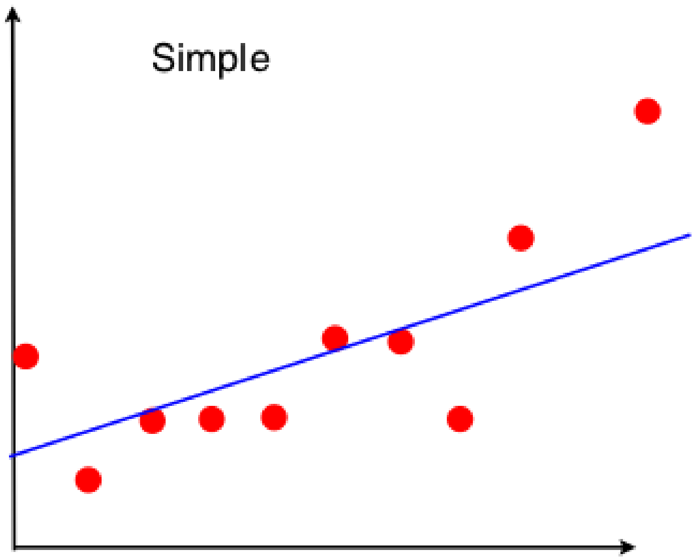
The low bias players' darts tend to be centered around the center of the dart board, while the high bias players' darts are centered in a different location. In this case, their darts are "biased" toward the top of the dart board.

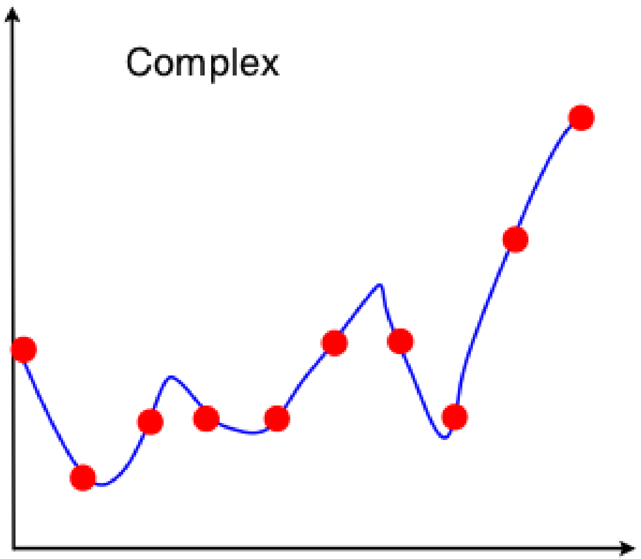
The high variance dart throwers have their darts more spread out; they are less able to successfully place the dart where they're aiming (in the case of the biased player, they are aiming at the incorrect location).



Bias and Variance Tradeoff

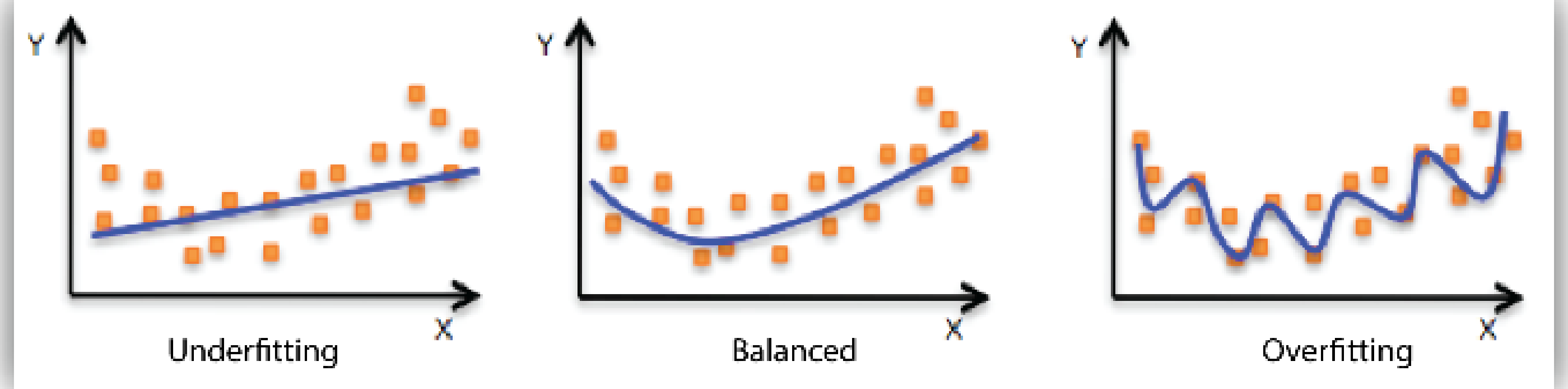
e.g. a linear model can only fit a straight line. A high-degree polynomial can fit a complex curve. But the polynomial can fit the individual sample, rather than the population. Its shape can vary from sample to sample, so it has high variance.





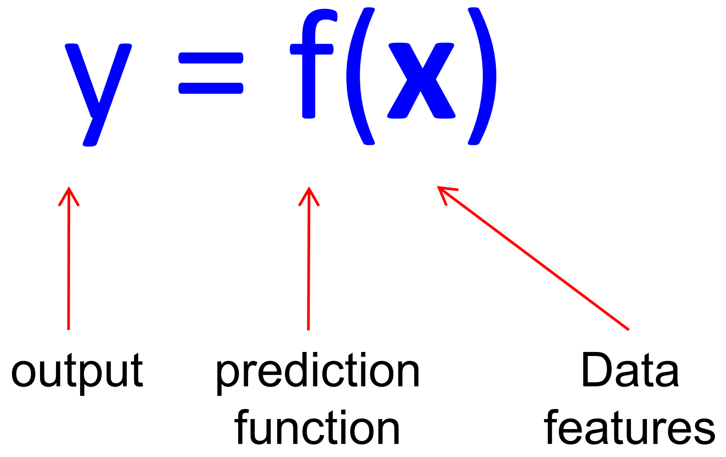
Underfitting vs. Overfitting

Understanding model fit is important for understanding the root cause for poor model accuracy. This understanding will guide you to take corrective steps. We can determine whether a predictive model is underfitting or overfitting the training data by looking at the prediction error on the training data and the evaluation data.



Your model is underfitting the training data when the model performs poorly on the training data. This is because the model is unable to capture the relationship between the input examples (often called X) and the target values (often called Y). Your model is overfitting your training data when you see that the model performs well on the training data but does not perform well on the evaluation data. This is because the model is memorizing the data it has seen and is unable to generalize to unseen examples.

Supervised Machine Learning Framework



$$y = f(\mathbf{x})$$

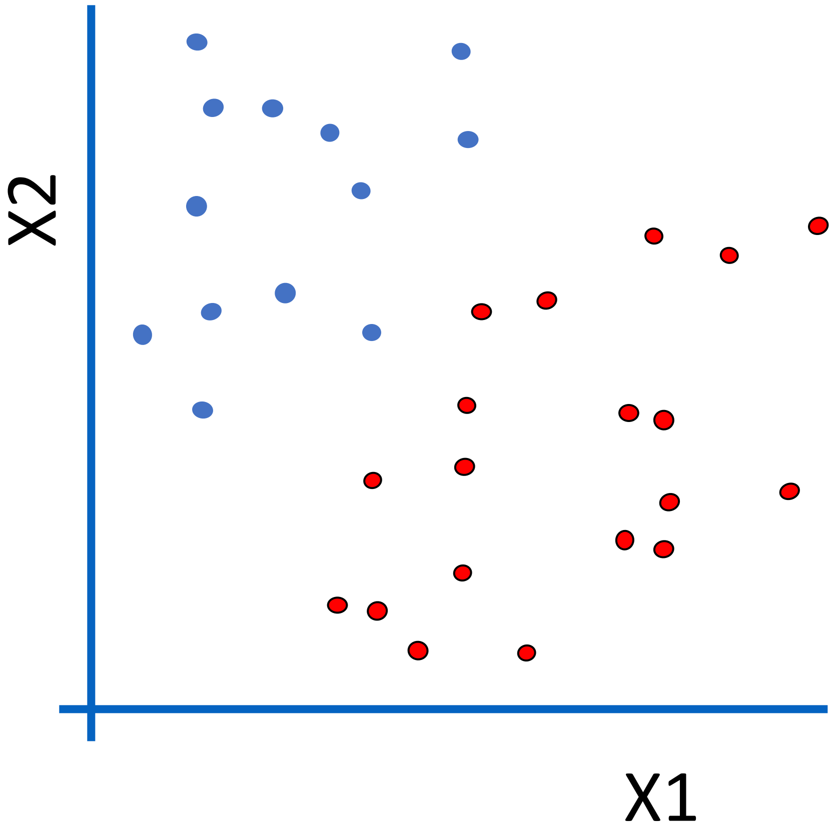
output

prediction function

Data features

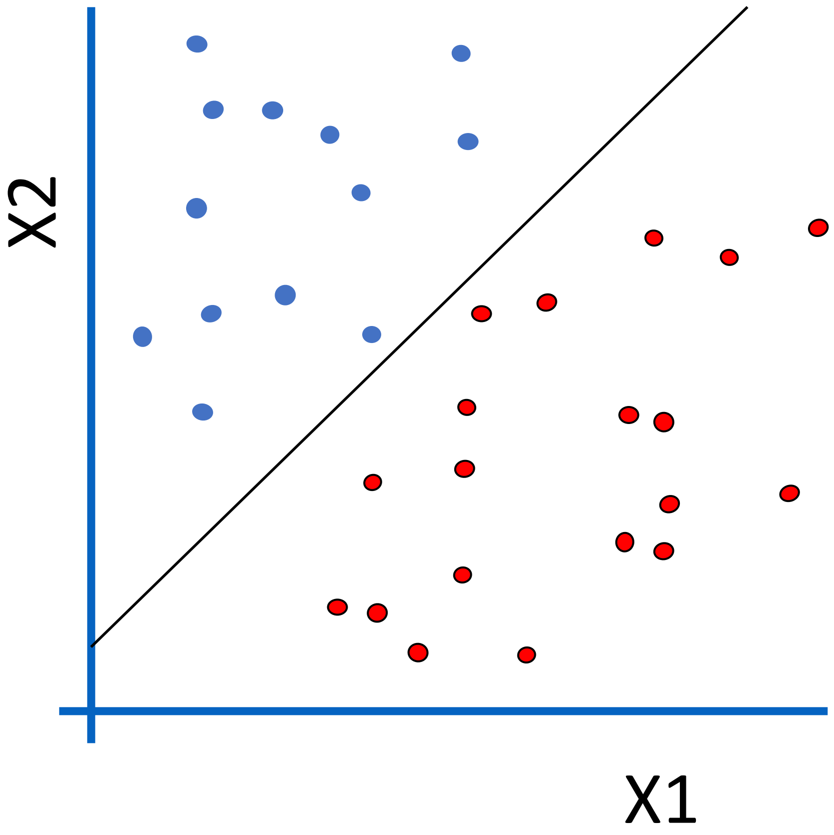
- **Training:** given a *training set* of labeled examples $\{(\mathbf{x}_1,y_1), \ldots, (\mathbf{x}_N,y_N)\}$, estimate the prediction function f by minimizing the prediction error on the training set
- **Testing:** apply f to a never before seen *test example* $\mathbf{x}$ and output the predicted value $y = f(\mathbf{x})$

Linear Classifiers



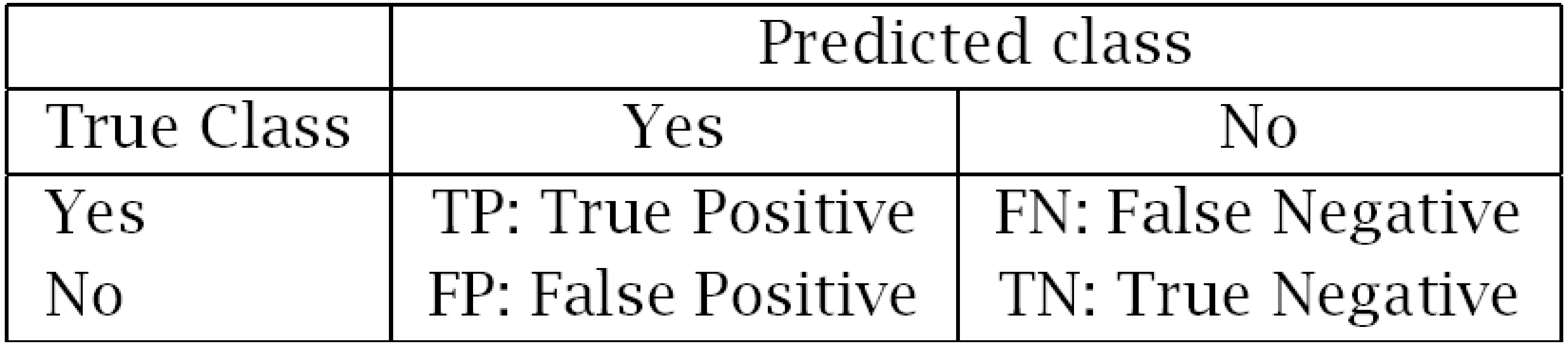
How would you classify this data?

Linear Classifiers



How would you classify this data?

Measuring Classification Error

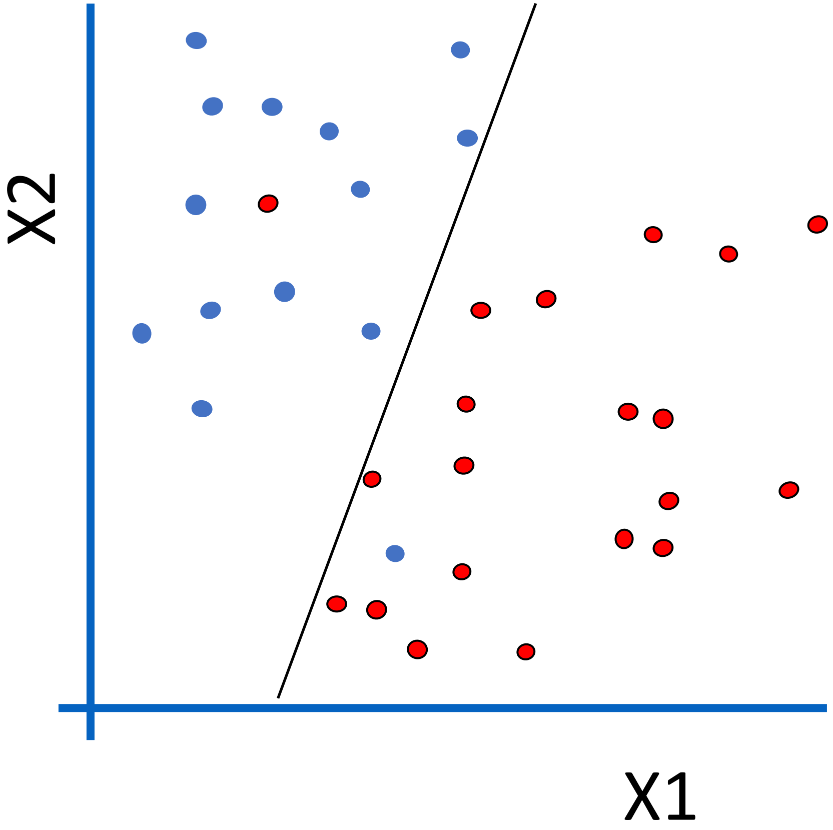


	Predicted class	
True Class	Yes	No
Yes	TP: True Positive	FN: False Negative
No	FP: False Positive	TN: True Negative

- Error rate = # of errors / # of instances = (FN+FP) / N
- Recall = # of found positives / # of positives
  = TP / (TP+FN) = sensitivity = hit rate
- Precision = # of found positives / # of found
  = TP / (TP+FP)
- Specificity = TN / (TN+FP)
- False alarm rate = FP / (FP+TN) = 1 - Specificity

Evaluating What’s Been Learned

1. We randomly select a portion of the data to be used for training (the training set)
2. Train the model on the training set.
3. Once the model is trained, we run the model on the remaining instances (the test set) to see how it performs



Confusion Matrix

Classified As

Actual

	Blue	**Red**
Blue	7	1
Red	0	5

Prediction: Regression

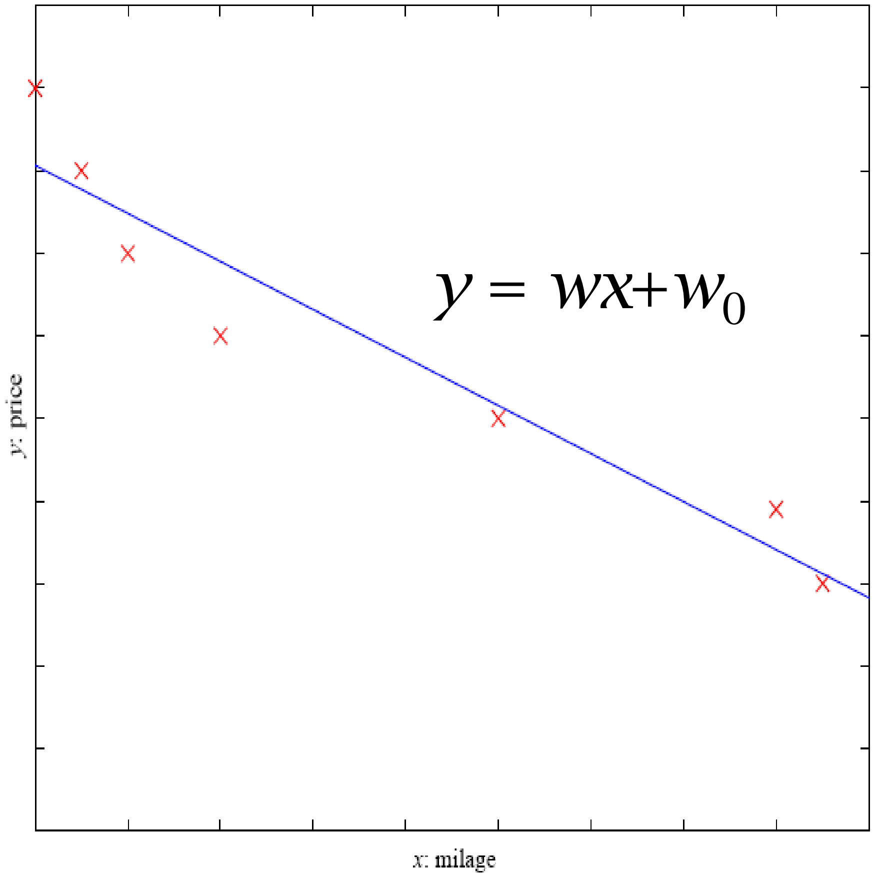
- Example: Price of a used car
- *x* : car attributes

  y : price

  $$y = g\,(x \mid \vartheta\,)$$

  g ( ) model,

  ϑ parameters

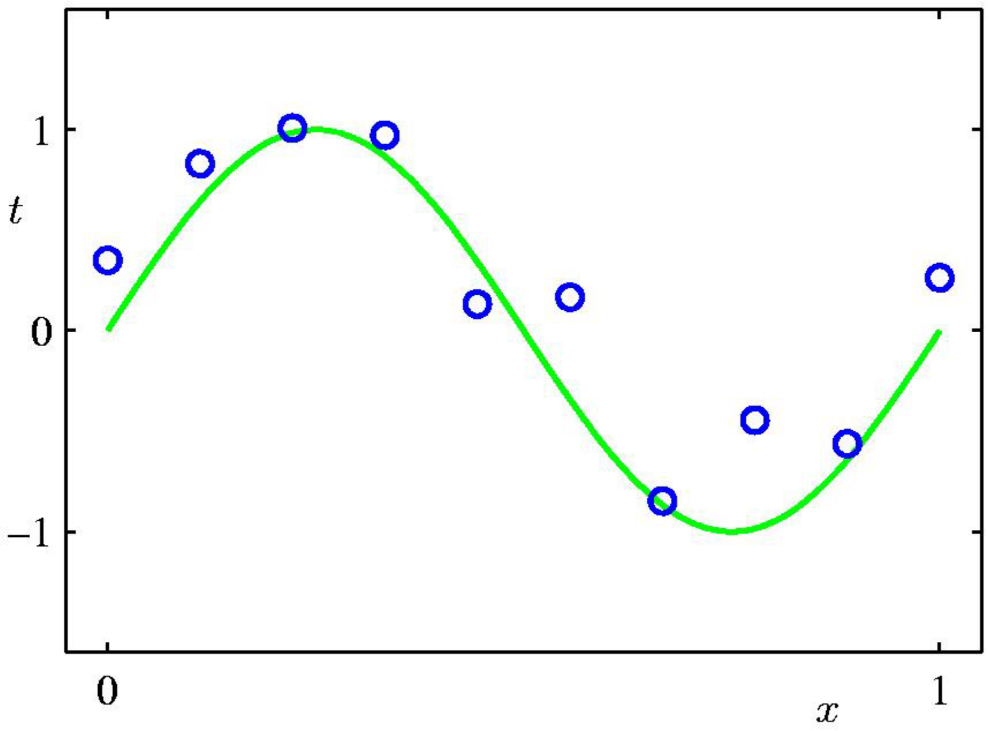


Measuring Regression Error

- Mean absolute error (MAE)
- Mean squared error (MSE)
- Root mean squared error (RMSE)

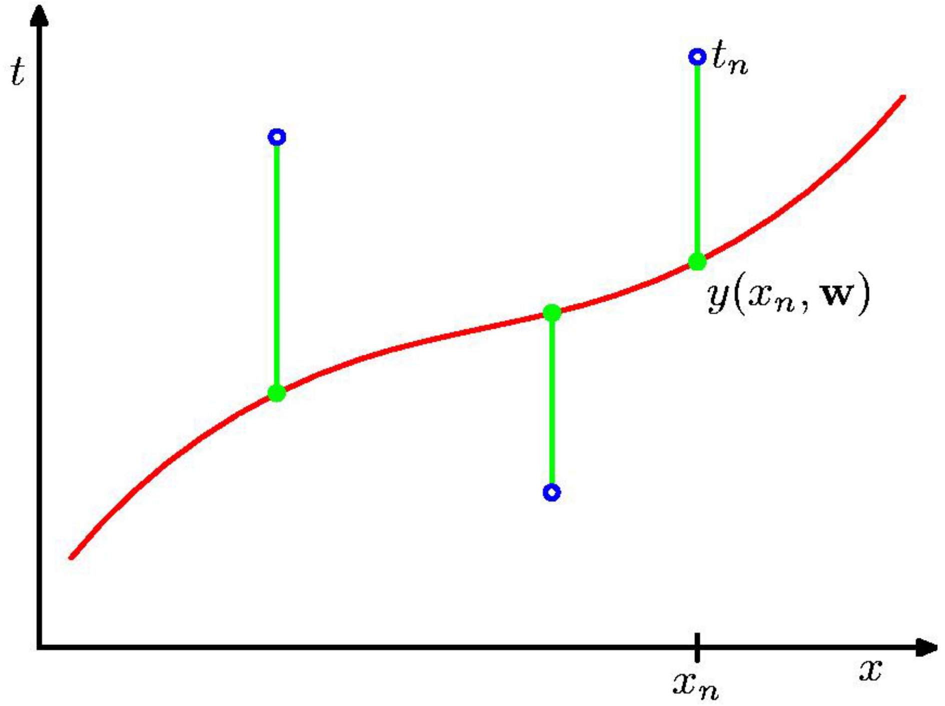
- Explained variance (%)
- Correlation coefficient (R)
- R^2 - coefficient of determination or correlation coefficient squared

Polynomial Curve Fitting



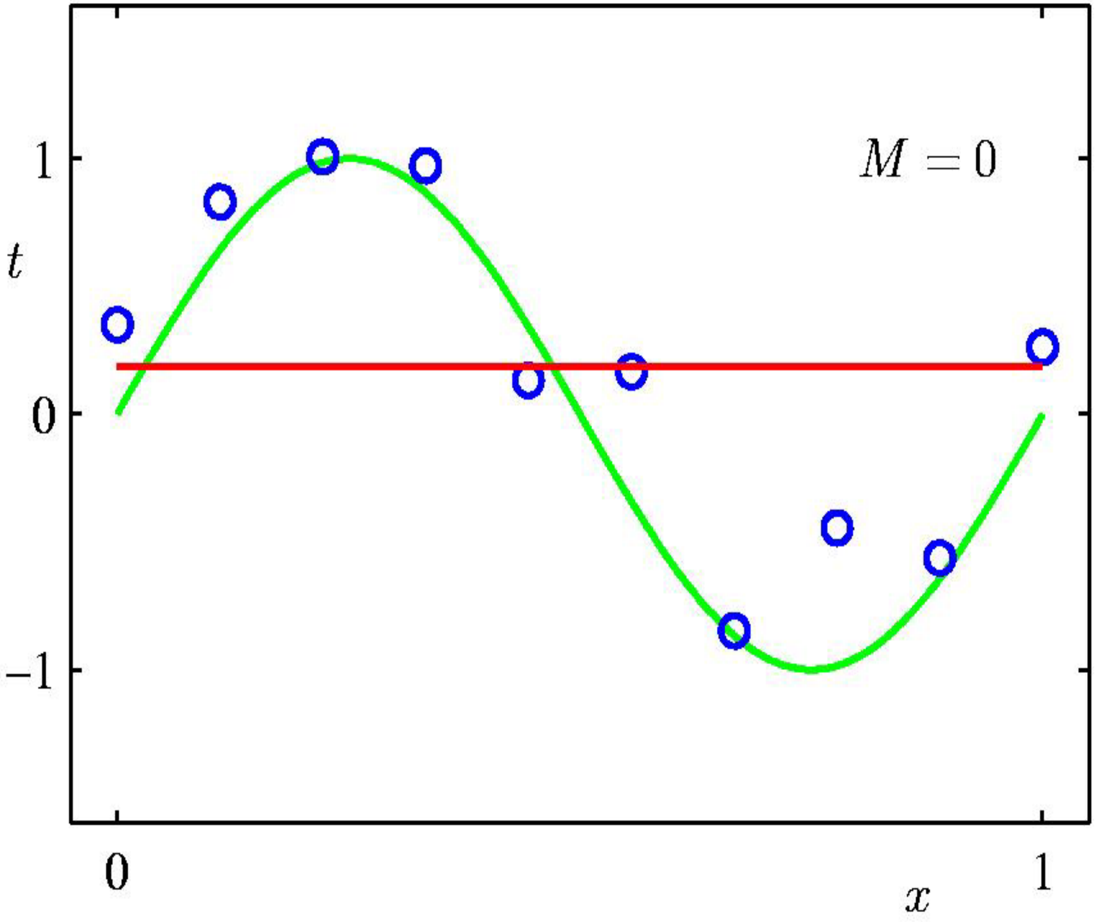
$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

Sum-of-Squares Error Function

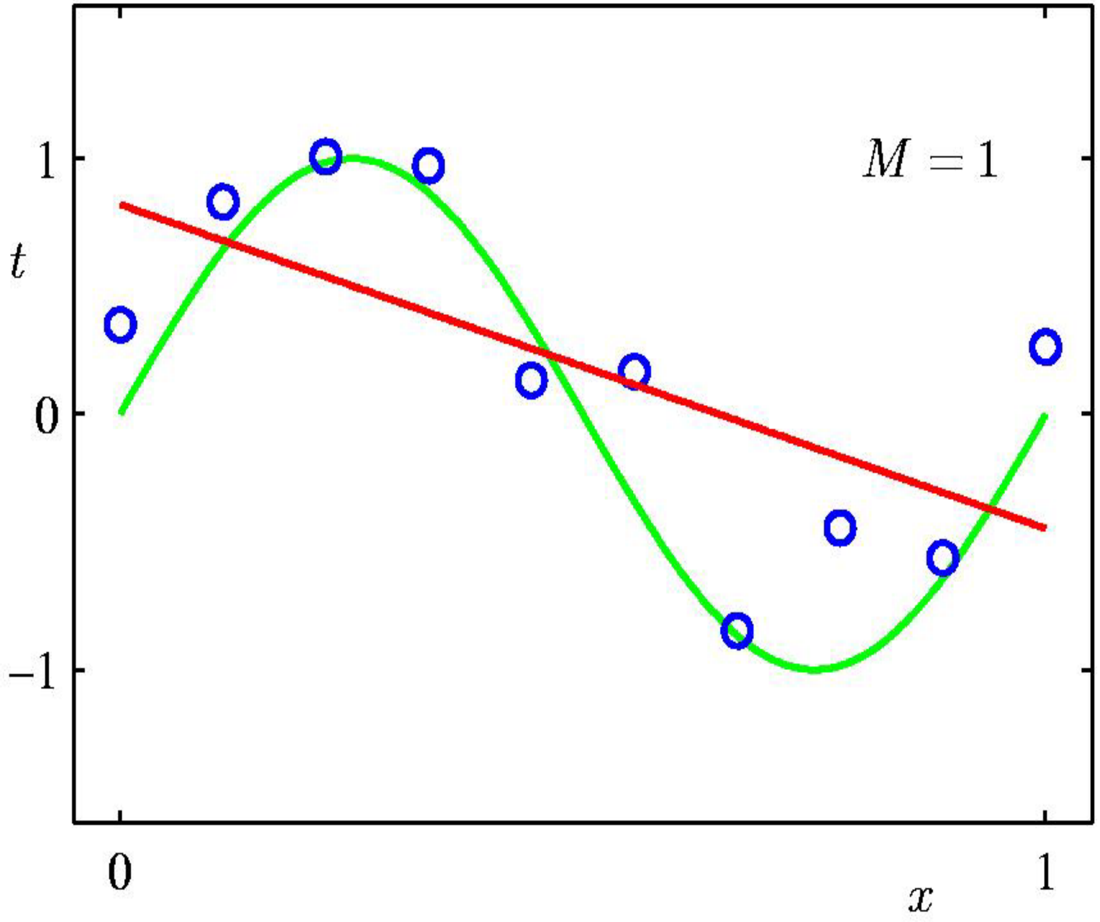


$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2$$

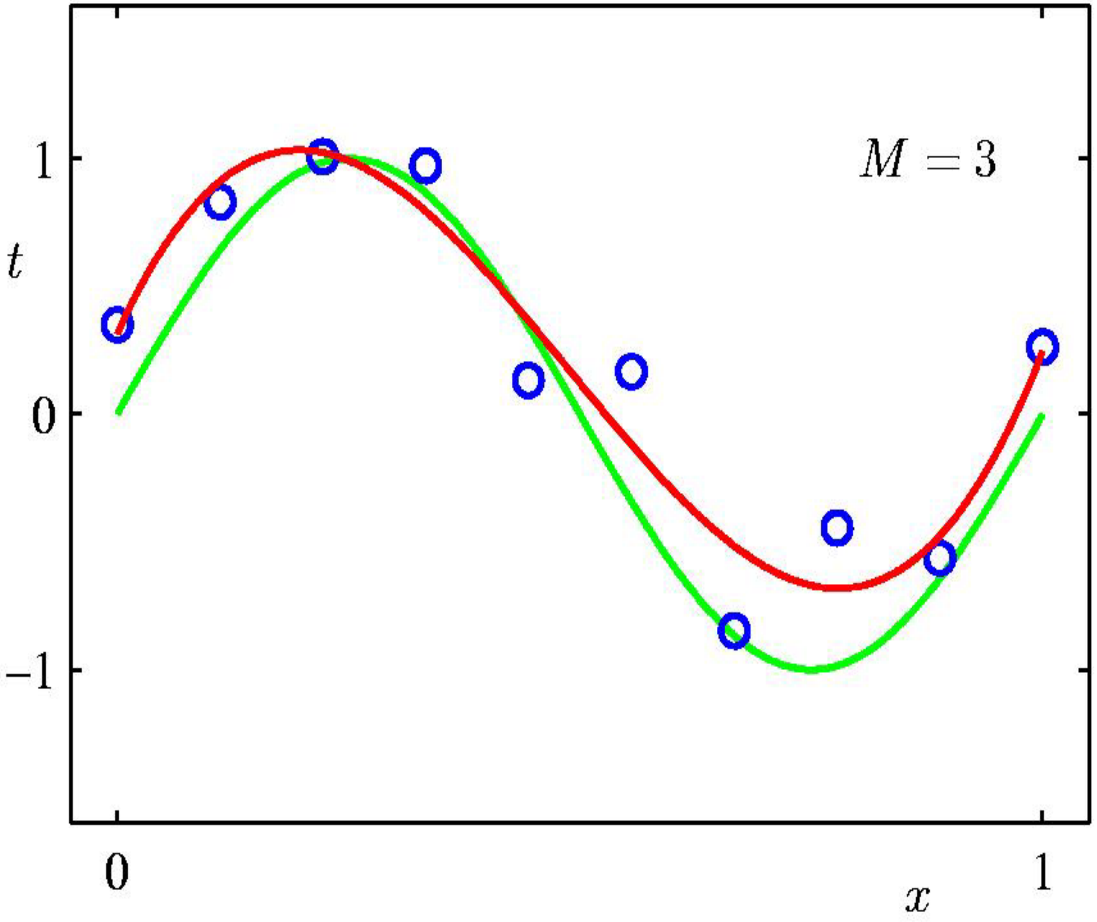
0th Order Polynomial



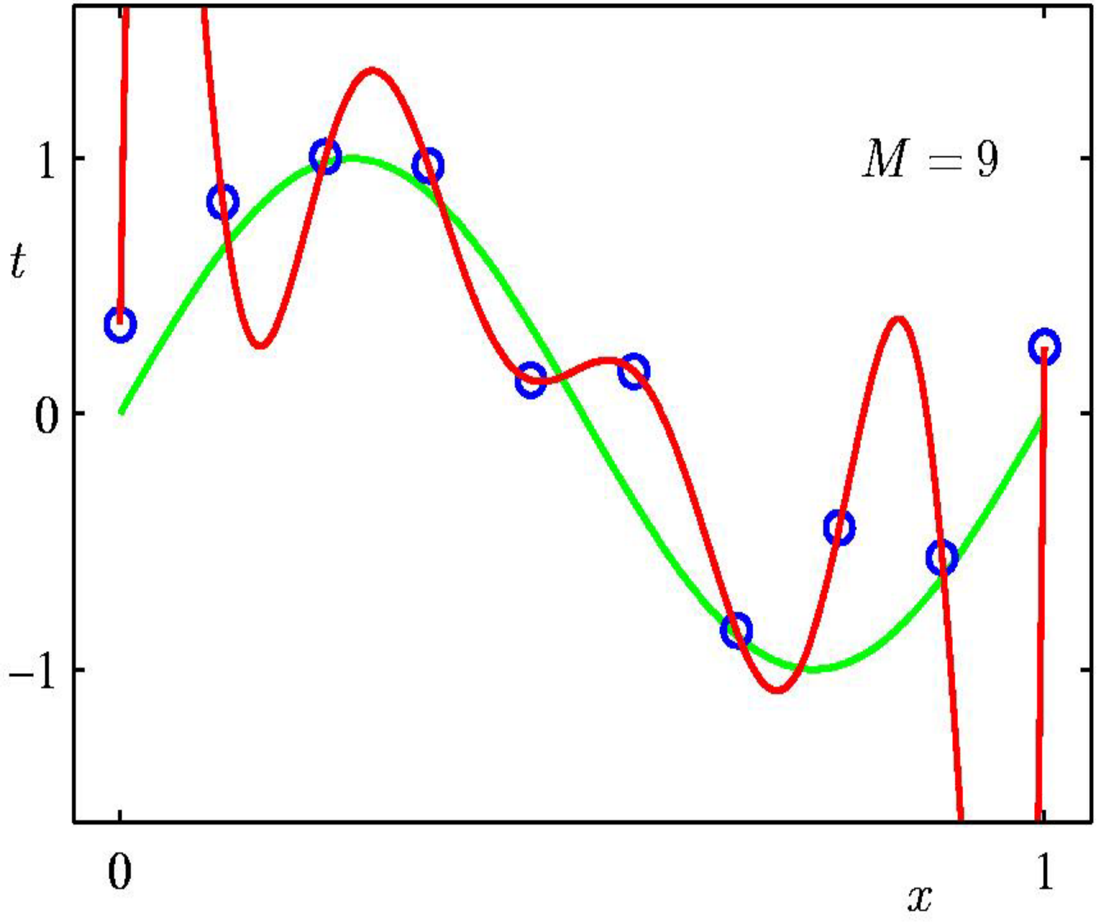
1st Order Polynomial



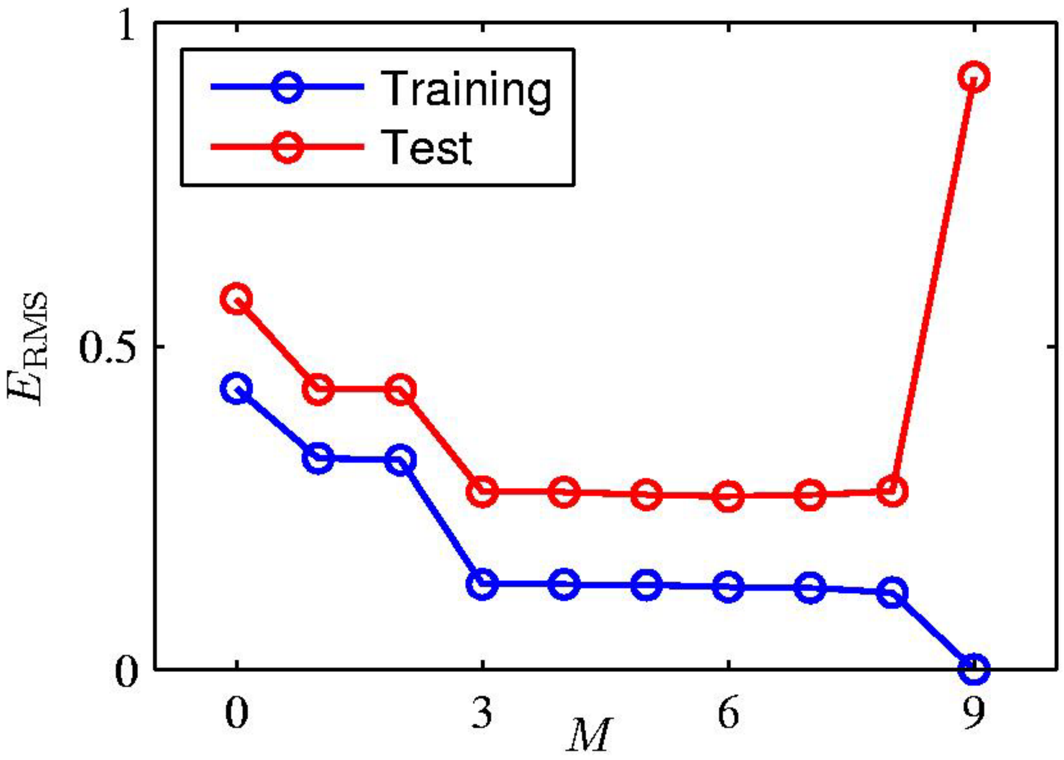
3^{rd} Order Polynomial



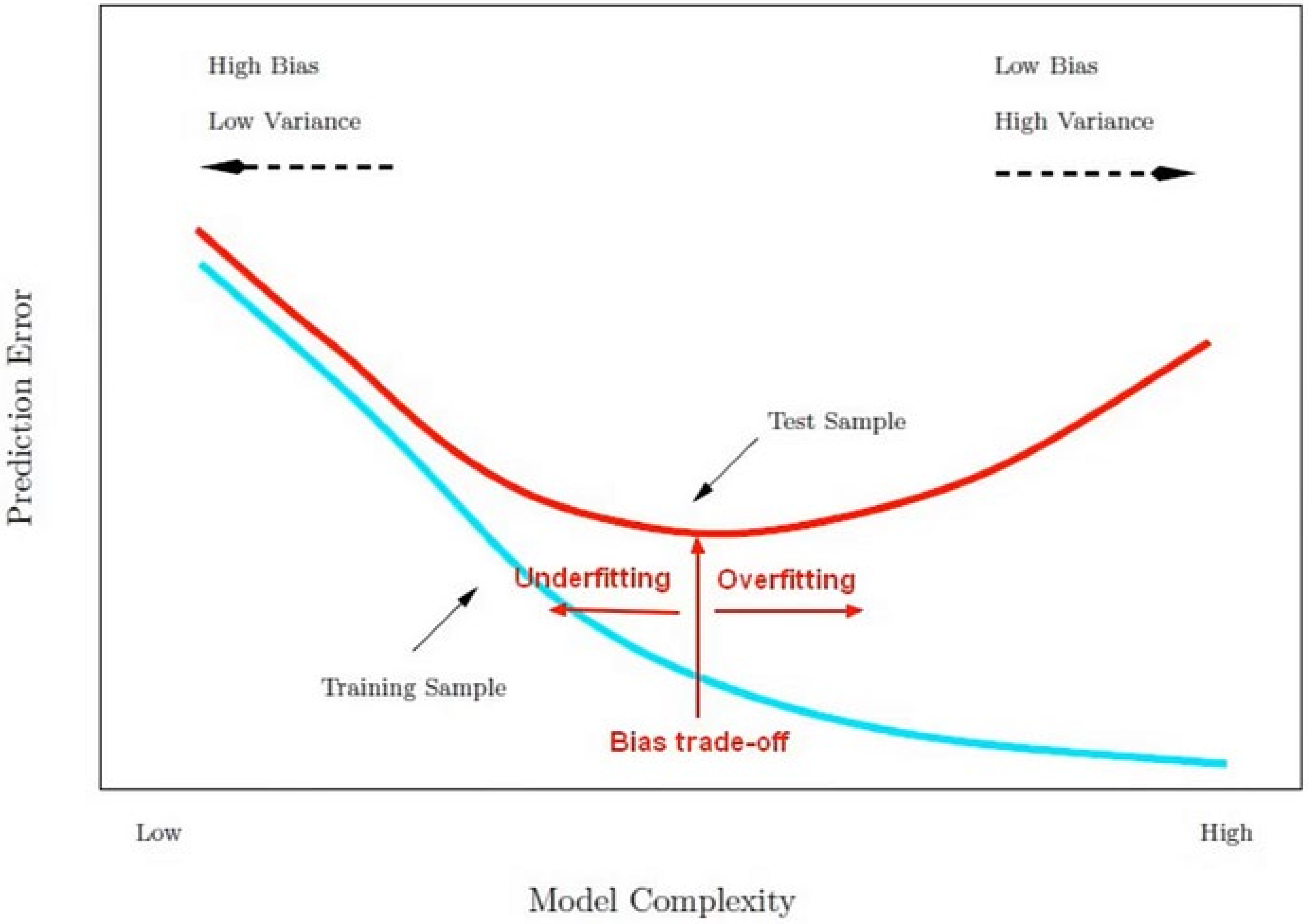
9th Order Polynomial



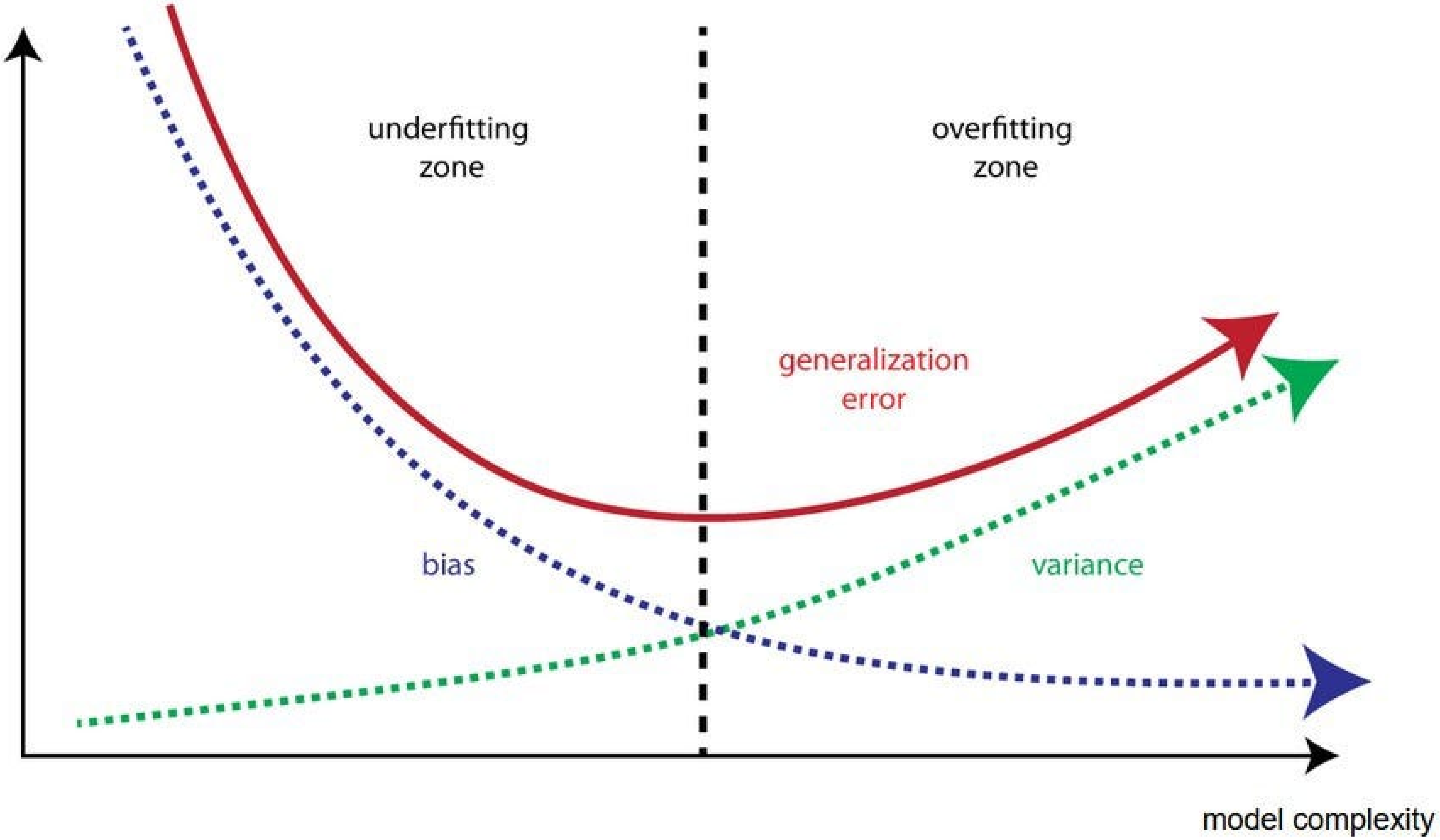
Over-fitting



Root-Mean-Square (RMS) Error: $E_{\mathrm{RMS}} = \sqrt{2E(\mathbf{w}^\star)/N}$


High Bias
Low Variance
Low Bias
High Variance
Prediction Error
Test Sample
Underfitting
Overfitting
Training Sample
Bias trade-off
Low
High
Model Complexity

the bias vs. variance trade-off



Poor performance on the training data could be because the model is too simple (the input features are not expressive enough) to describe the target well. Performance can be improved by increasing model flexibility. To increase model flexibility, try the following:

- Add new features or change the types of feature processing used
- Decrease the amount of regularization used

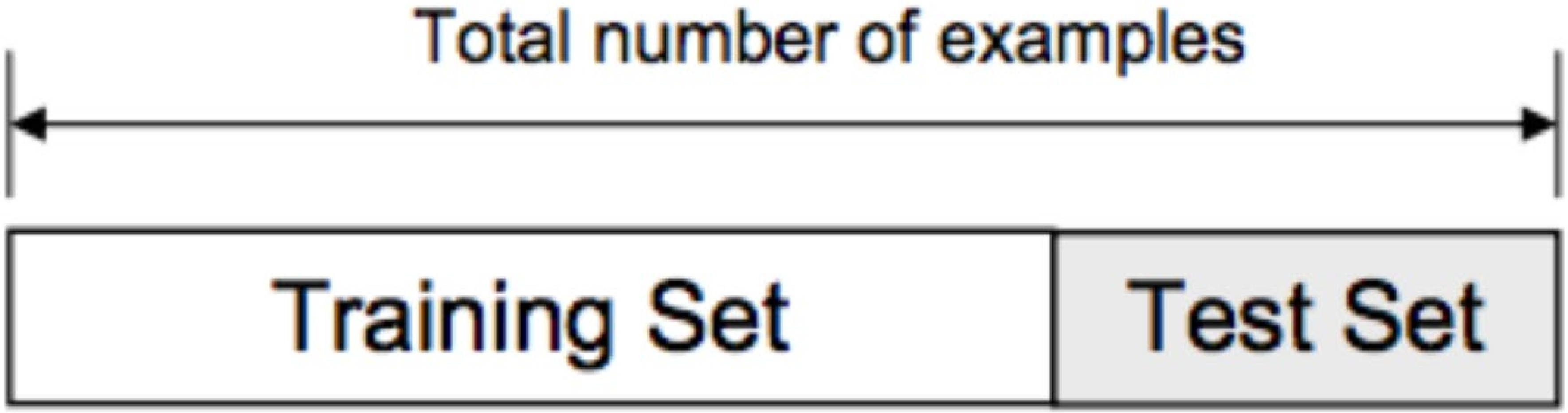
If your model is overfitting the training data, it makes sense to take actions that reduce model flexibility. To reduce model flexibility, try the following:

- Feature selection: consider using fewer feature combinations.
- Increase the amount of regularization used.

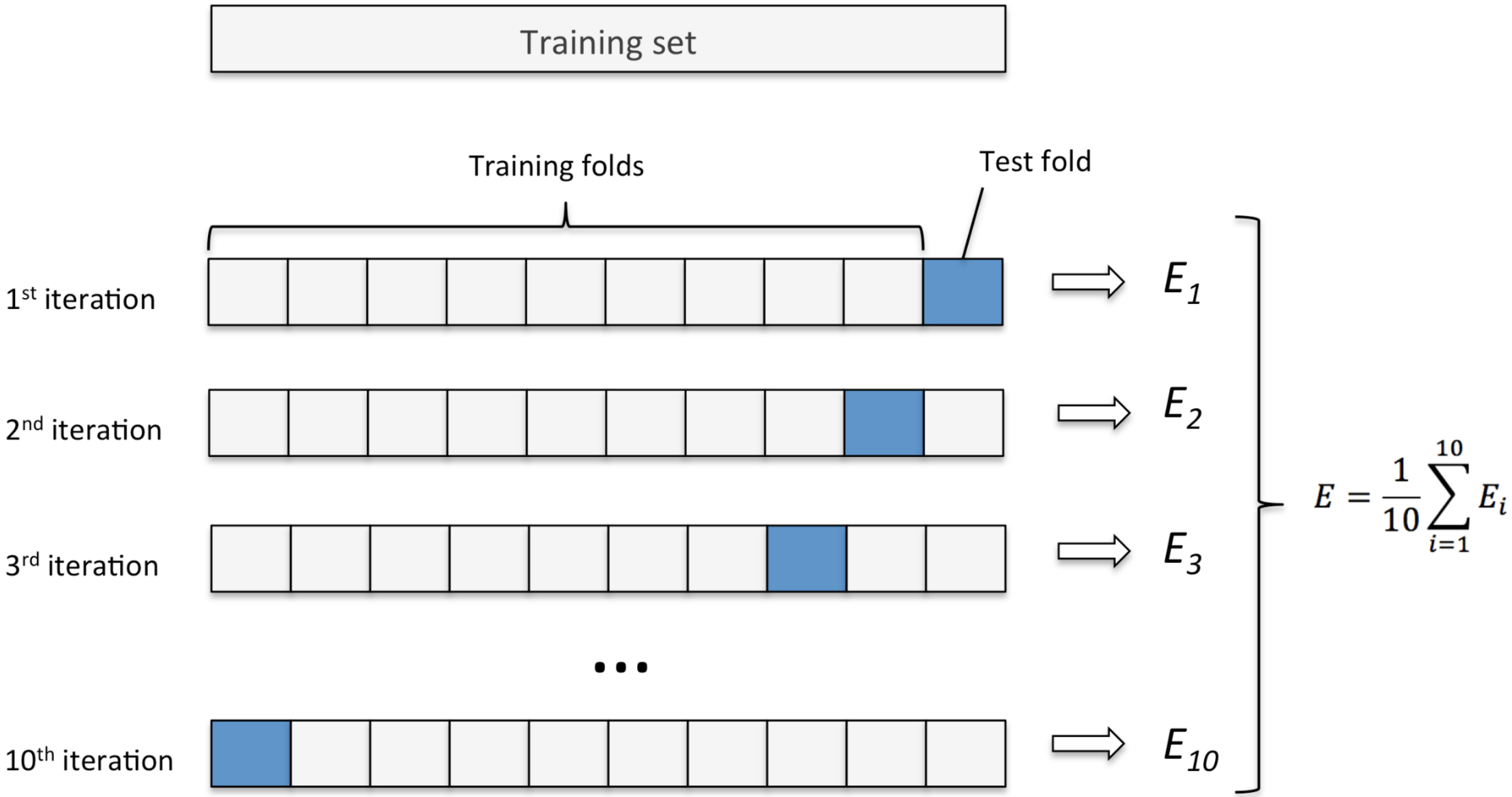
Accuracy on training and test data could be poor because the learning algorithm did not have enough data to learn from. You could improve performance by doing the following:

- Increase the amount of training data examples.

Training-Test Split



K-fold cross-validation



Leave One Out Cross Validation (LOOCV)

In this type of cross validation, the number of folds equals to the number of observations we have in the dataset.

We then average ALL of these folds and build our model with the average. We then test the model against the last fold.

Because we would get a big number of training sets (equals to the number of samples), this method is very computationally expensive and should be used on small datasets.

If the dataset is big, it would most likely be better to use a different method, like k-fold.

Practical Tips

Plenty of data: training-test split

Medium data: 3,5,10-fold CV

Small datasets (< ~100): LOOCV

Model **Hyperparameters**

(Hyper-) Parameters to the model

Training – validation- test split

More complex strategies

Worst ML crimes

Test on training data

Dropping outliers

Use high variance model when n<<p (number of samples << number of features)

L1/L2/... regularization without data standardization

Use linear models for non-linear interaction

KNN: k-nearest neighbors (kNN)

- Simplest learning algorithm!

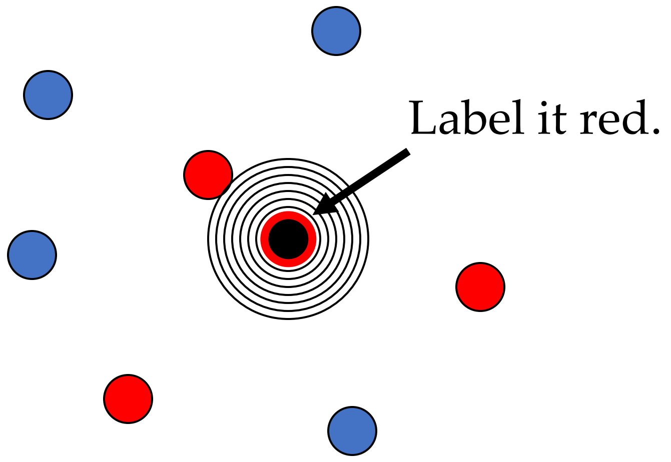
- Actually there is NO LEARNIG

Instance-Based Learning

- Idea:
  - Similar examples have similar label.
  - Classify new examples like similar training examples.
- Algorithm:
  - Given some new example x for which we need to predict its class y
  - Find most similar training examples
  - Classify *x* “like” these most similar examples
- Questions:
  - How to determine similarity?
  - How many similar training examples to consider?
  - How to resolve inconsistencies among the training examples?

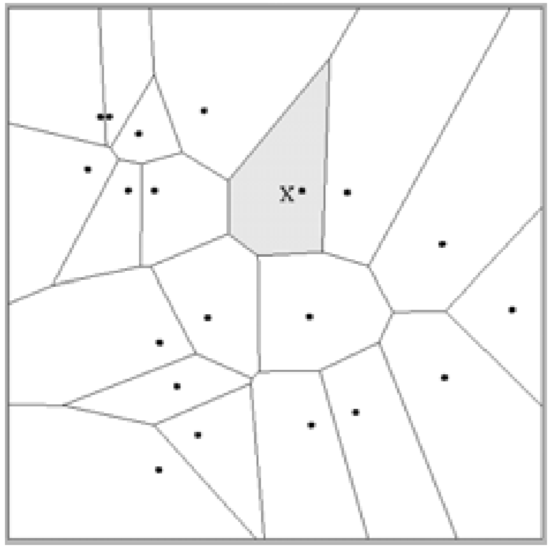
1-Nearest Neighbor

- One of the simplest of all machine learning classifiers
- Simple idea: label a new point the same as the closest known point



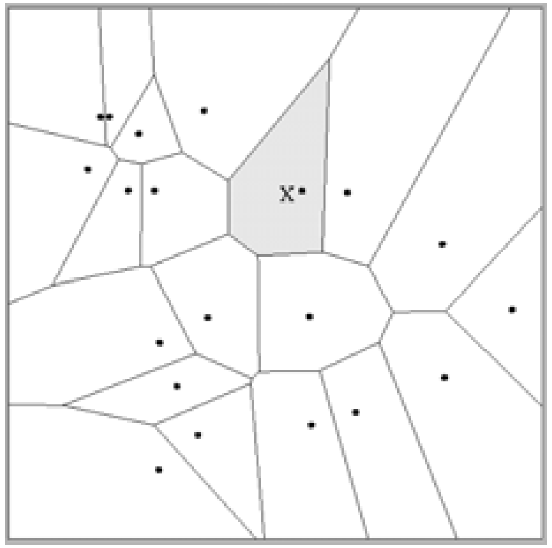
1-Nearest Neighbor

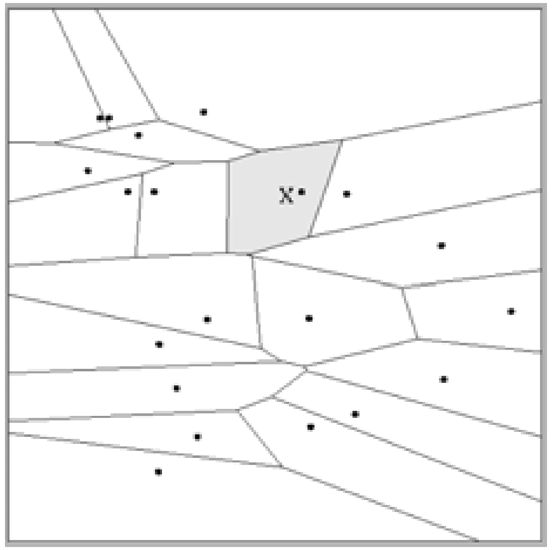
- A type of instance-based learning
  - Also known as “memory-based” learning
- Forms a Voronoi tessellation of the instance space



Distance Metrics

- Different metrics can change the decision surface





$\text{Dist}(\mathbf{a,b}) = (a_1 - b_1)^2 + (a_2 - b_2)^2$ $\text{Dist}(\mathbf{a,b}) = (a_1 - b_1)^2 + (3a_2 - 3b_2)^2$

- Standard Euclidean distance metric:
  - Two-dimensional: Dist(a,b) = sqrt($(a_1 - b_1)^2 + (a_2 - b_2)^2$)
  - Multivariate: Dist(a,b) = sqrt($\sum (a_i - b_i)^2$)

Adapted from "Instance-Based Learning"
lecture slides by Andrew Moore, CMU.

1-NN's Aspects as an Instance-Based Learner:

A distance metric

- Euclidean
- When different units are used for each dimension

  → normalize each dimension by standard deviation
- For discrete data, can use hamming distance

  → D(x1,x2) =number of features on which x1 and x2 differ
- Others (e.g., normal, cosine)

How many nearby neighbors to look at?

- One

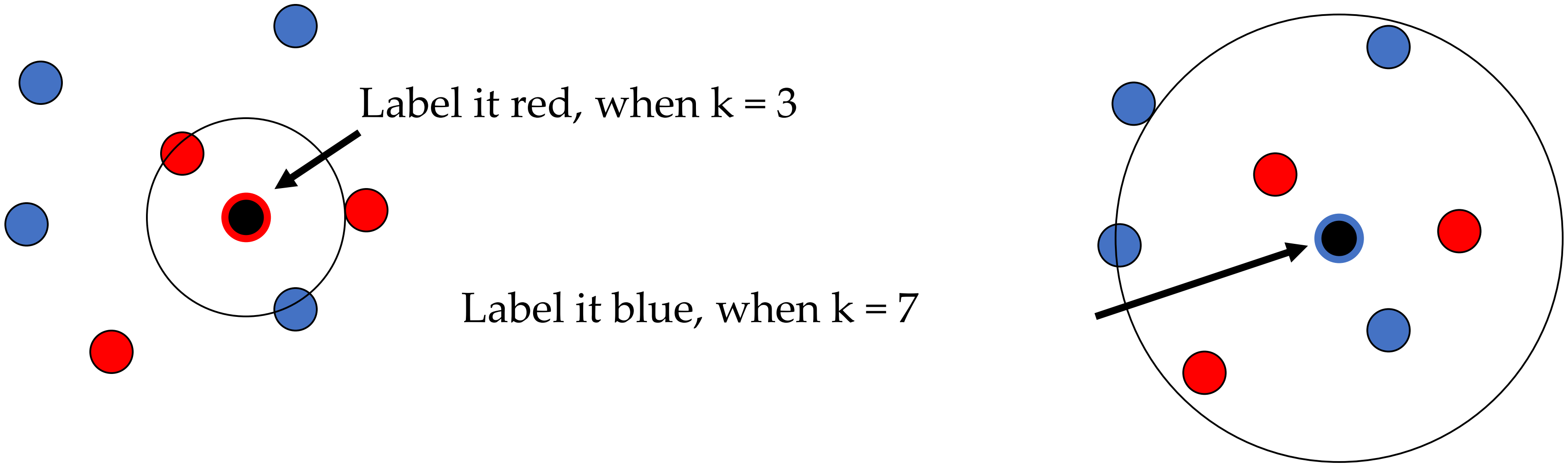
How to fit with the local points?

- Just predict the same output as the nearest neighbor.

Adapted from "Instance-Based Learning"
lecture slides by Andrew Moore, CMU.

k – Nearest Neighbor

- Generalizes 1-NN to smooth away noise in the labels
- A new point is now assigned the most frequent label of its *k* nearest neighbors



Label it red, when k = 3

Label it blue, when k = 7

k-NN issues

The Data is the Model

- No training needed.
- Accuracy generally improves with more data.
- Matching is simple and fairly fast if data fits in memory.
- Usually need data in memory, but can be run off disk.

Minimal Configuration:

- Only parameter is k (number of neighbors)
- But two other choices are important:
  - Weighting of neighbors (e.g. inverse distance)
  - Similarity metric

k-NN Flavors

Classification:

- Model is y = f(X), y is from a discrete set (labels).
- Given X, compute y = majority vote of the k nearest neighbors.
- Can also use a weighted vote* of the neighbors.

Regression:

- Model is y = f(X), y is a real value.
- Given X, compute y = average value of the k nearest neighbors.
- Can also use a weighted average* of the neighbors.

* Weight function is usually the inverse distance.

K-NN distance measures

- **Euclidean Distance:** Simplest, fast to compute

$$d(x, y) = ||x - y||$$

- **Cosine Distance:** Good for documents, images, etc.

$$d(x, y) = 1 - \frac{x \cdot y}{||x||\,||y||}$$

- **Jaccard Distance:** For set data:

$$d(X, Y) = 1 - \frac{|X \cap Y|}{|X \cup Y|}$$

- **Hamming Distance:** For string data:

$$d(x, y) = \sum_{i=1}^{n} (x_i \neq y_i)$$

K-NN metrics

- **Manhattan Distance:** Coordinate-wise distance

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

- **Edit Distance:** for strings, especially genetic data.

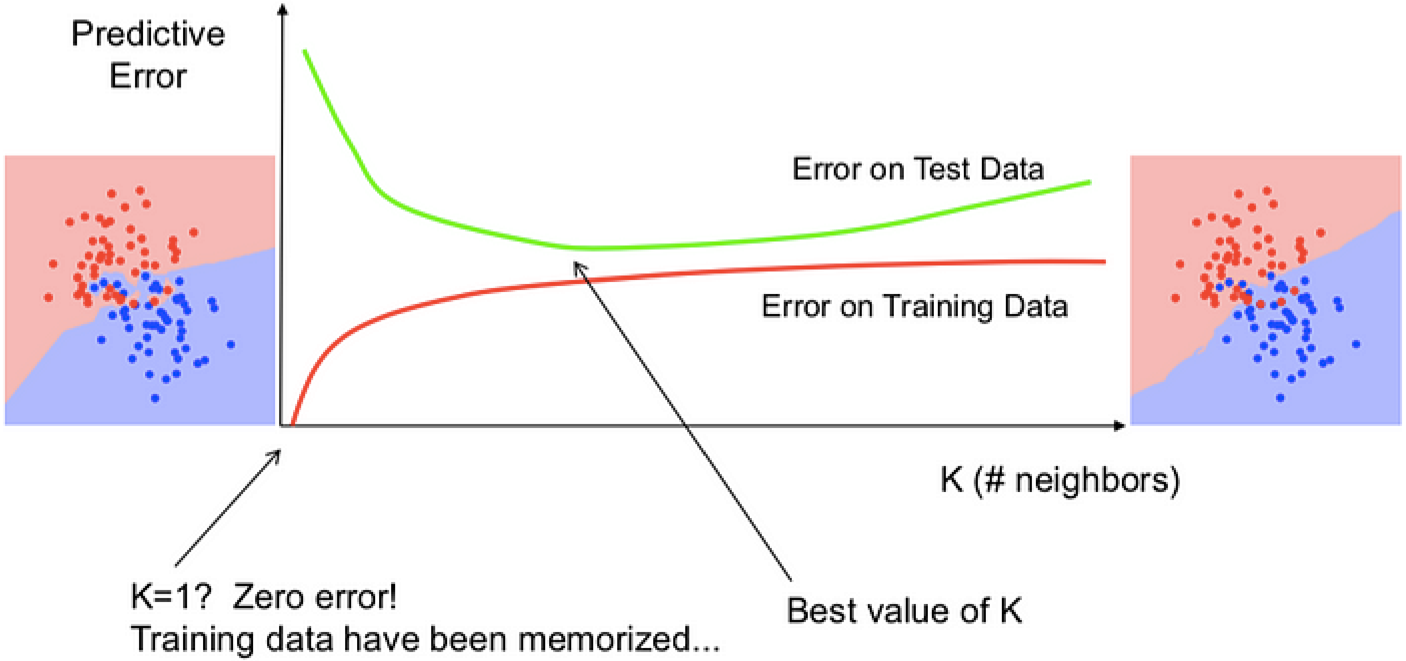
- **Mahalanobis Distance:** Normalized by the sample covariance matrix – unaffected by coordinate transformations.

Choosing k

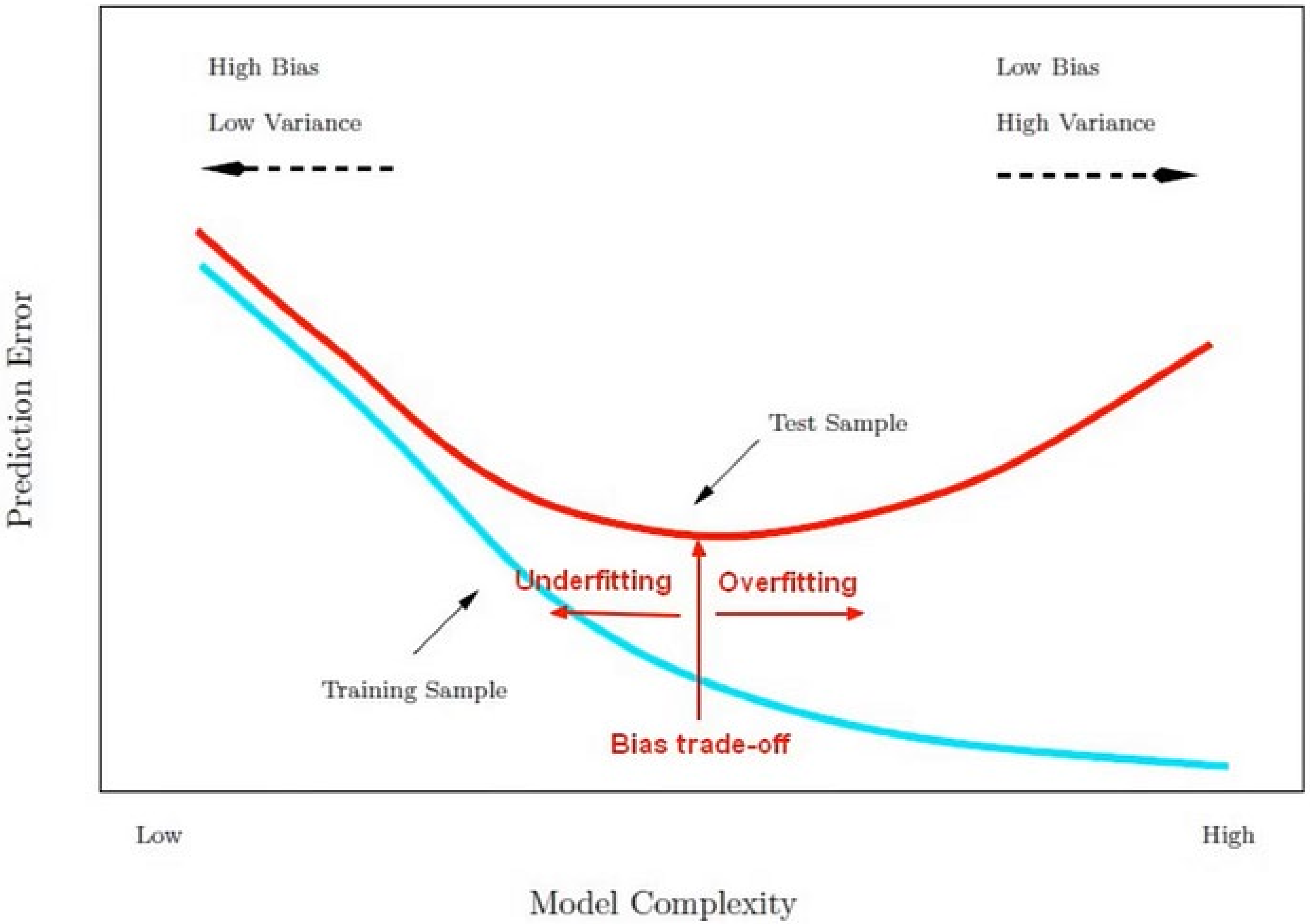
We have a bias/variance tradeoff:

- Small k → ?
- Large k → ?

Error rates and K

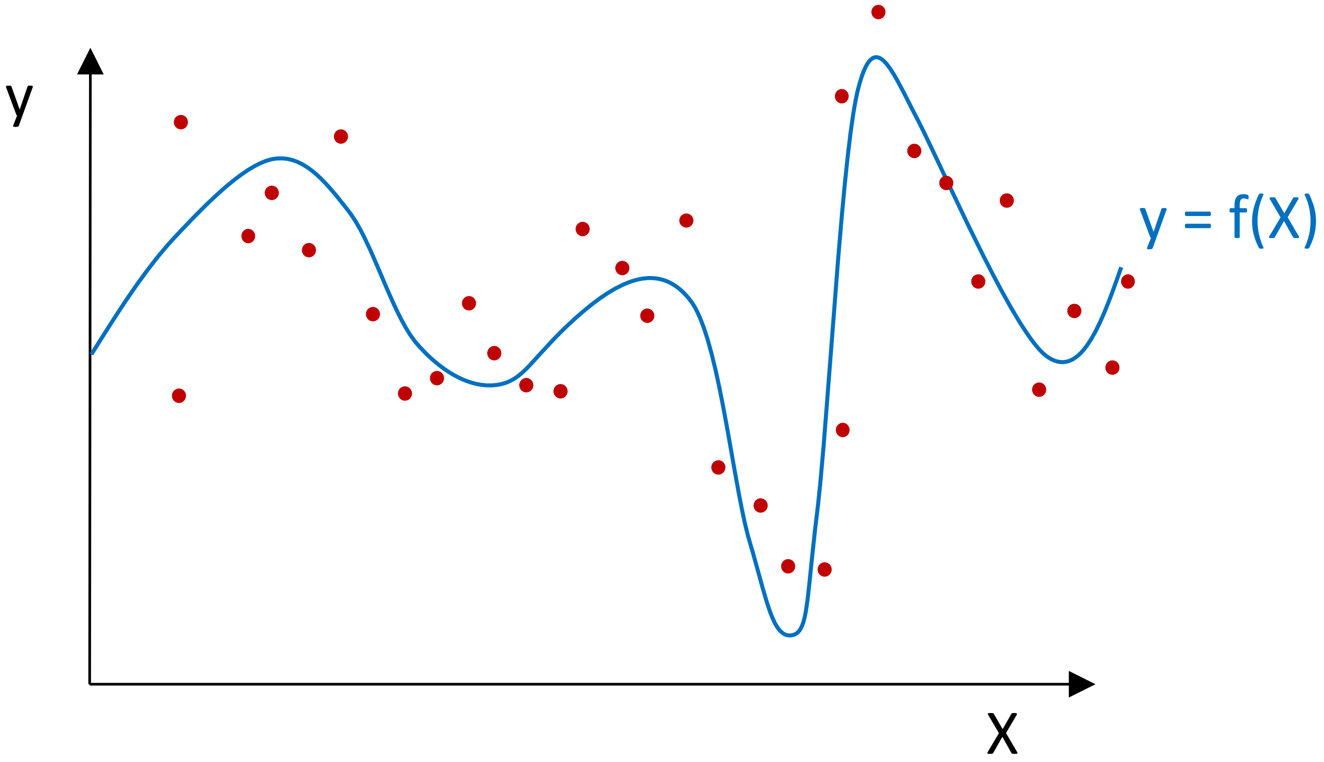


The bias will be 0 when K=1, however, when it comes to new data (in test set), it has higher chance to be an error, which causes high variance. When we increase K, the training error will increase (increase bias), but the test error may decrease at the same time (decrease variance). We can think that when K becomes larger, since it has to consider more neighbors, its model is more complex.


High Bias
Low Variance
Low Bias
High Variance
Prediction Error
Test Sample
Underfitting
Overfitting
Training Sample
Bias trade-off
Low
High
Model Complexity

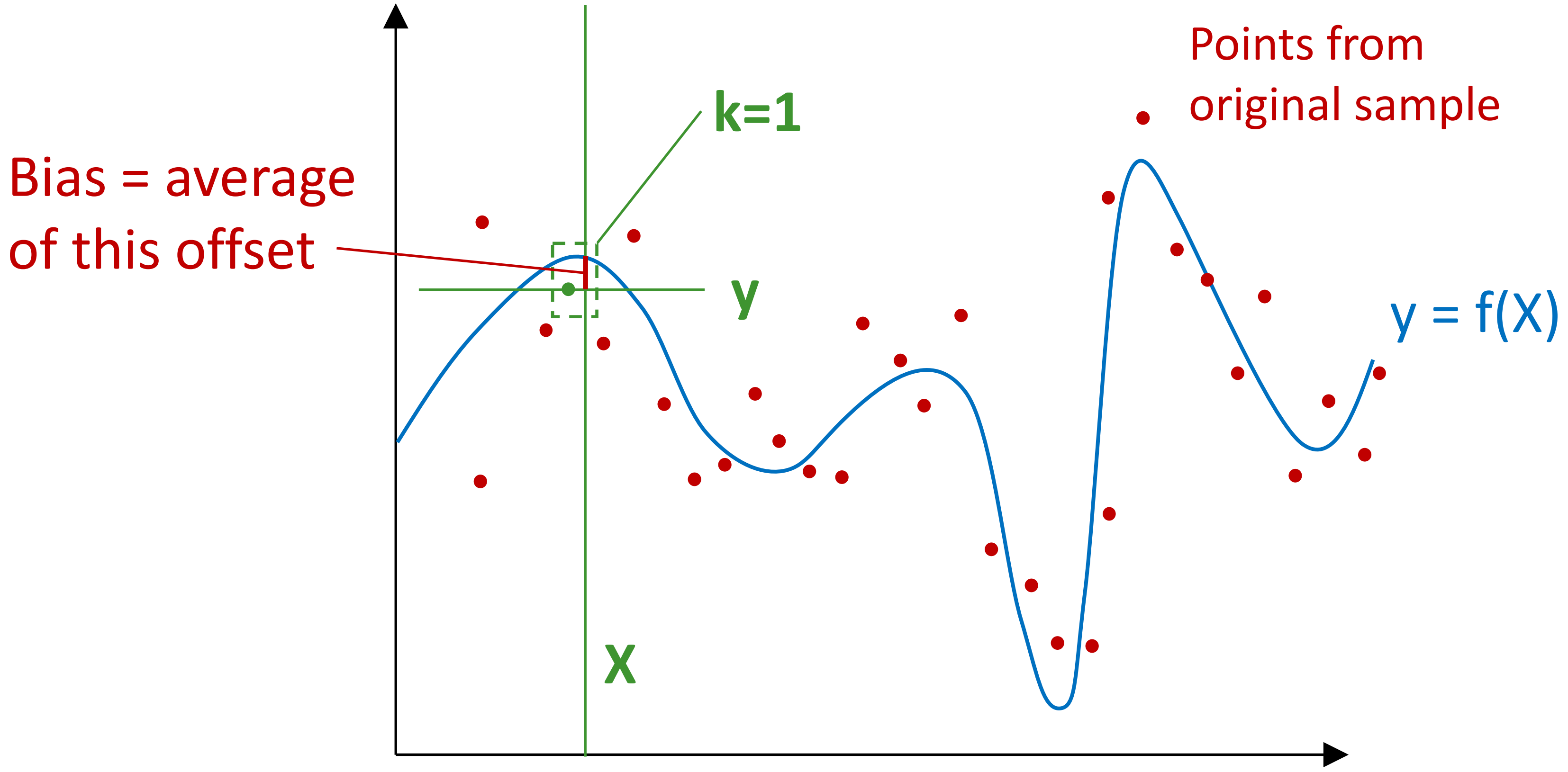
Choosing k

- Small k → low bias, high variance
- Large k → high bias, low variance
- Assume the real data follows the blue curve, with some mean-zero additive noise. Red points are a data sample.



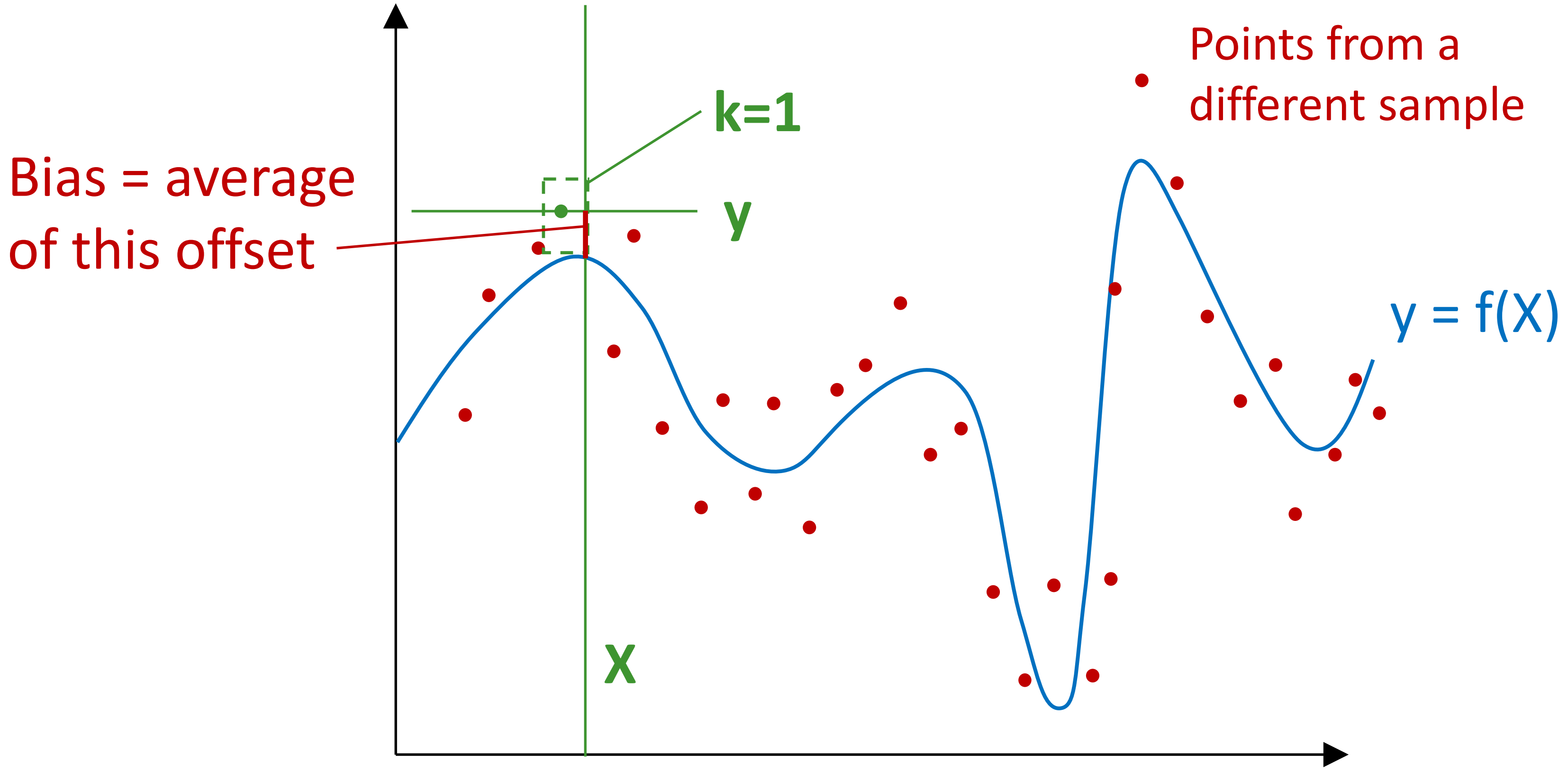
Choosing k

- **Small k** → low bias, high variance
- Large k → high bias, low variance



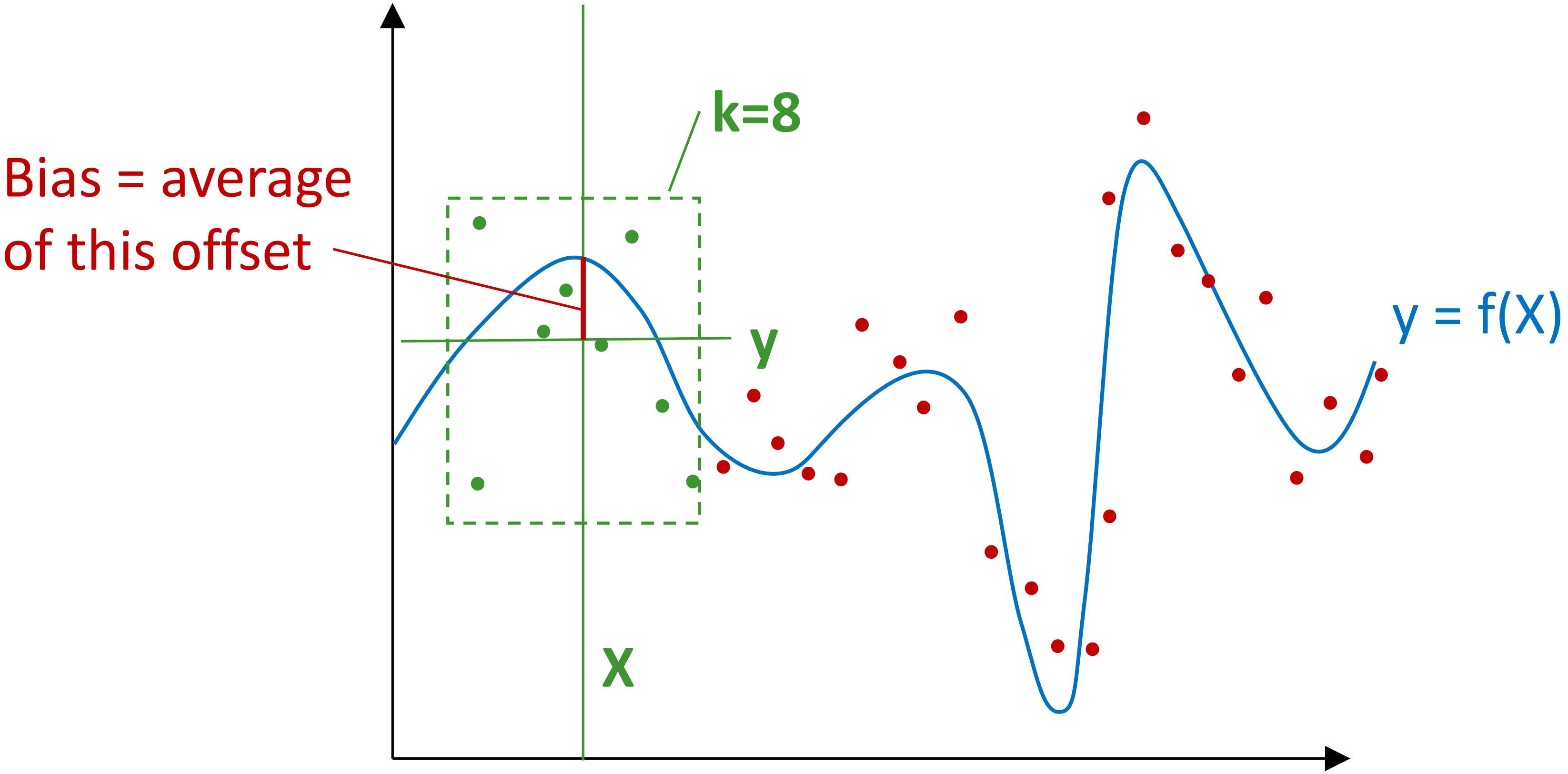
Choosing k

- **Small k** → low bias, high variance
- Large k → high bias, low variance



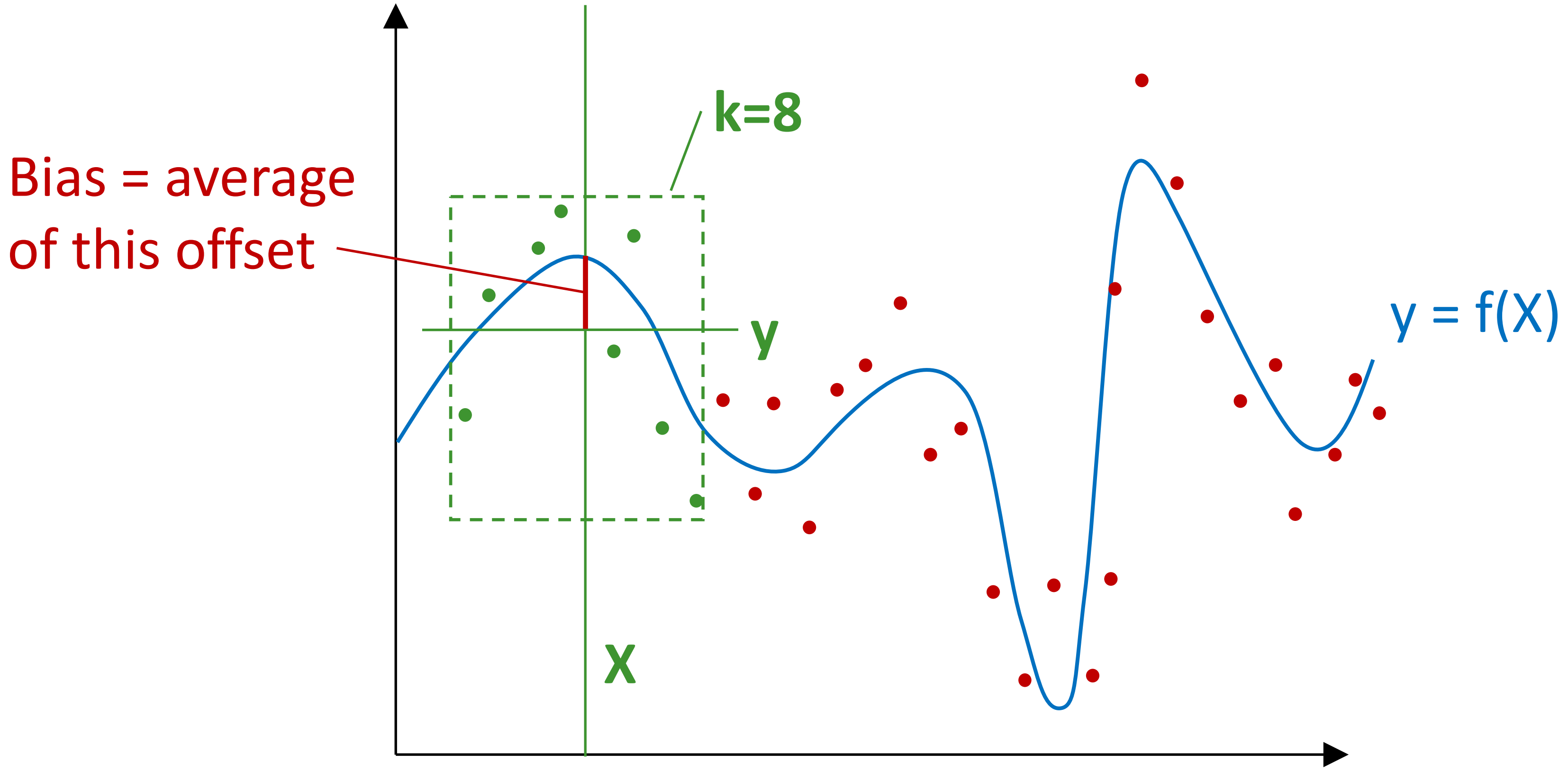
Choosing k

- Small k → low bias, high variance
- **Large k** → high bias, low variance



Choosing k

- Small k → low bias, high variance
- **Large k** → high bias, low variance



Choosing k in practice

Use cross-validation! Break data into train, validation and test subsets, e.g. 80-10-10 % random split.

Predict: For each point in the validation set, predict using the k-Nearest neighbors from the training set. Measure the error rate (classification) or squared error (regression).

Tune: try different values of k, and use the one that gives minimum error on the validation set.

Evaluate: test on the test set to measure performance.

kNN and the curse of dimensionality

The curse of dimensionality refers to phenomena that occur in high dimensions (100s to millions) that do not occur in low-dimensional (e.g. 3-dimensional) space.

In particular data in high dimensions are much sparser (less dense) than data in low dimensions.

For kNN, that means there are less points that are very close in feature space (very similar), to the point we want to predict.

kNN and the curse of dimensionality

From this perspective, its surprising that kNN works at all in high dimensions.

Luckily real data are not like random points in a high-dimensional cube. Instead they live in **dense clusters** and near **much lower-dimensional surfaces**.

Finally, points can be very “similar” even if their euclidean distance is large