A learning system: basic cycle

CS-309

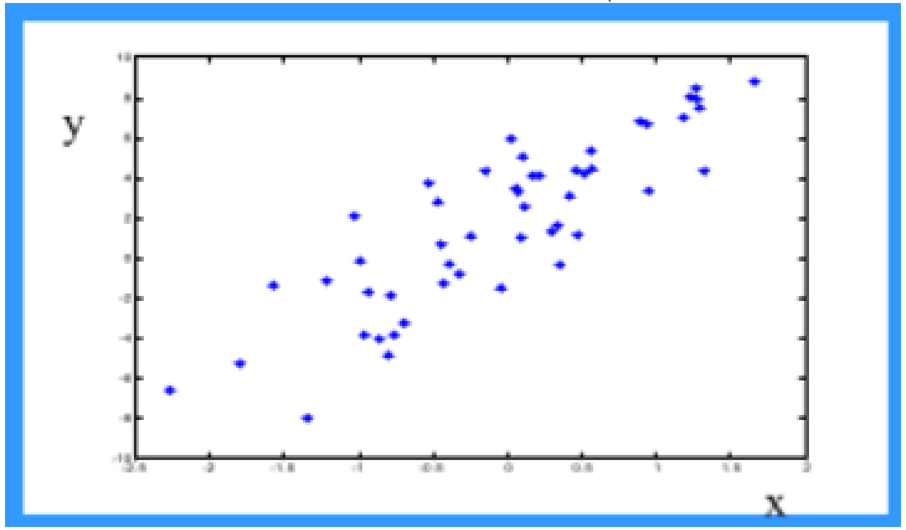
Sourav Kumar Dandapat

A learning system: basic cycle

1. **Data:**
2. **Model selection:**
3. **Choose the objective function**
4. **Learning**

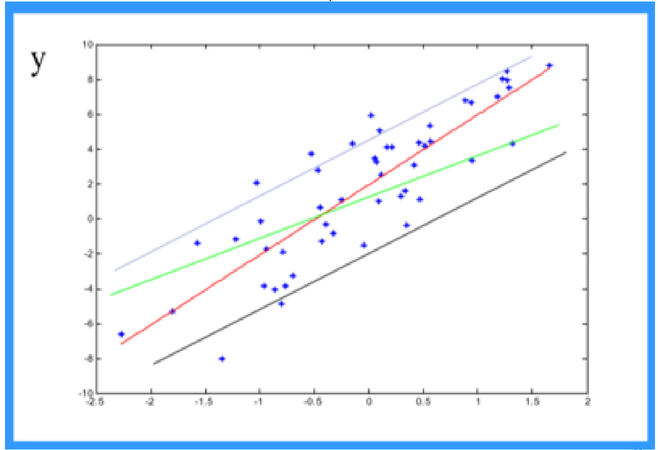
A learning system: basic cycle

1. **Data:** $D=\{d_1,d_2,\ldots,d_n\}$
2. **Model selection:**
3. **Choose the objective function**
4. **Learning**



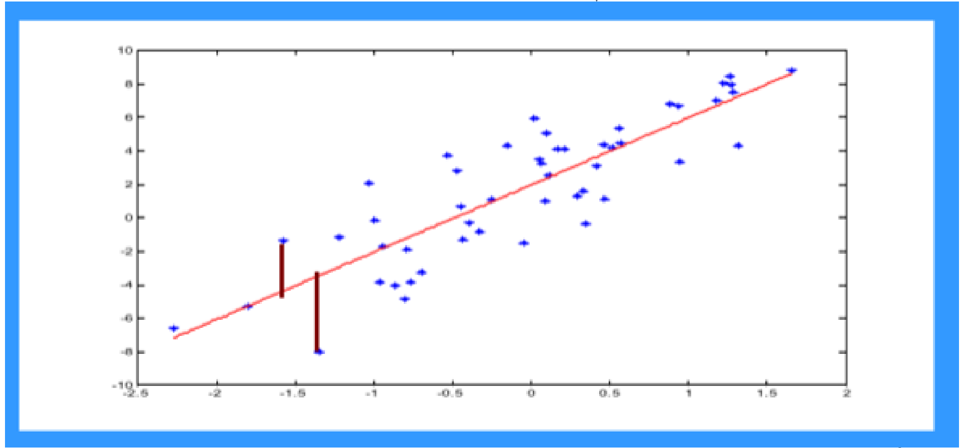
A learning system: basic cycle

1. **Data:** $D=\{d_1,d_2,\ldots,d_n\}$
2. **Model selection:**
   a. **Select a model** or a set of models (with parameters) E.g. $y = ax + b$
3. **Choose the objective function**
4. **Learning:**



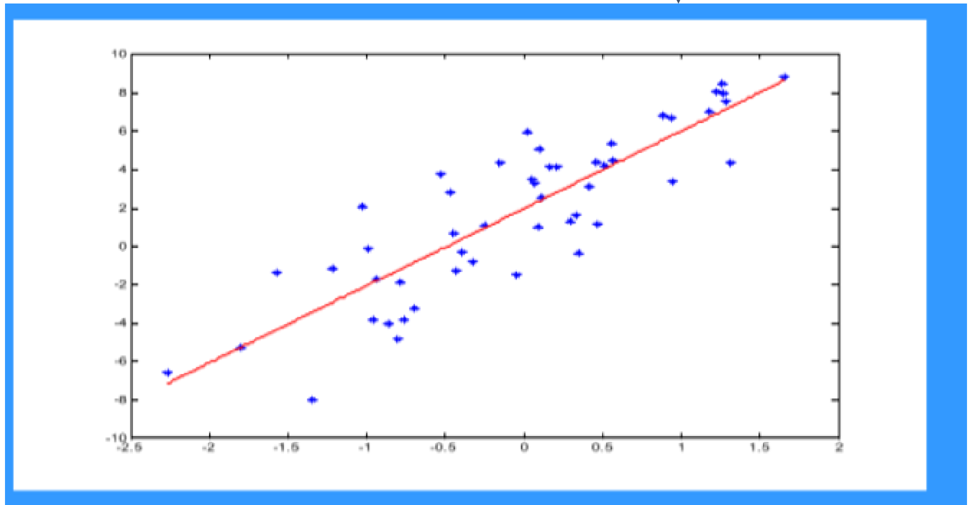
A learning system: basic cycle

1. **Data:** **D={d_1,d_2,…,d_n}**
2. **Model selection:**
   a. **Select a model** or a set of models (with parameters) E.g. *y =ax+ b*
3. **Choose the objective function**
   a. **Squared error** $\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$
4. **Learning:**



A learning system: basic cycle

1. **Data:** $D=\{d_1,d_2,\ldots,d_n\}$
2. **Model selection:**
   a. **Select a model** or a set of models (with parameters) E.g. $y = ax + b$
3. **Choose the objective function**
   a. **Squared error** $\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$
4. **Learning:**
   a. Find the set of parameters optimizing the error function
      i. The model and parameters with the smallest error



A learning system: basic cycle

1. **Data:** **D={d_1,d_2,…,d_n}**
2. **Model selection:**
   a. **Select a model** or a set of models (with parameters) E.g. $y = ax + b$
3. **Choose the objective function**
   a. **Squared error** $\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$
4. **Learning:**
   a. Find the set of parameters optimizing the error function
      i. The model and parameters with the smallest error

But there are problems one must be careful about …

Evaluation of the learned model

Problem

- We fit the model based on past examples observed in D
- But ultimately we are interested in learning the mapping that performs well on the whole population of examples

Training data: **Data used to fit the parameters of the model**

Training error: $Error(D,f) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$

True (generalization) error (over the whole population):

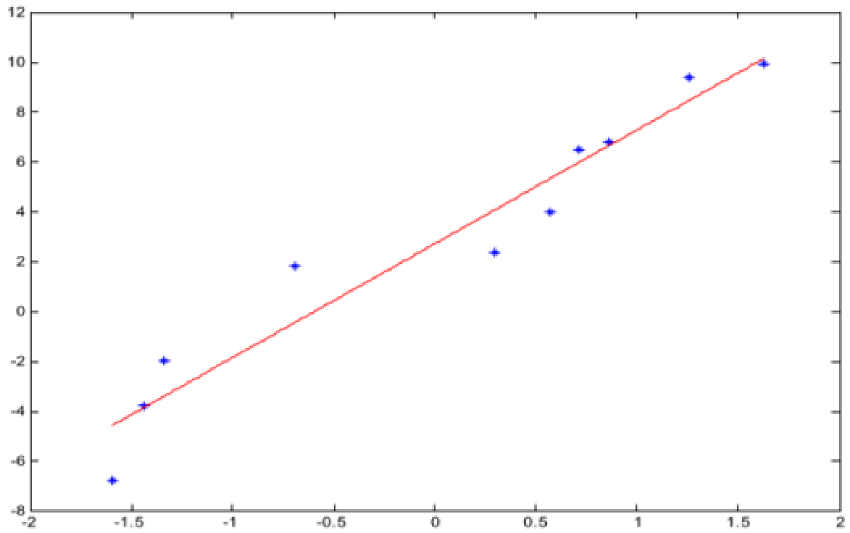
$E_{(x,y)}[(y - f(x))^2]$ Mean squared error

Training error tries to approximate the true error !!!!

Does a good training error imply a good generalization error ?

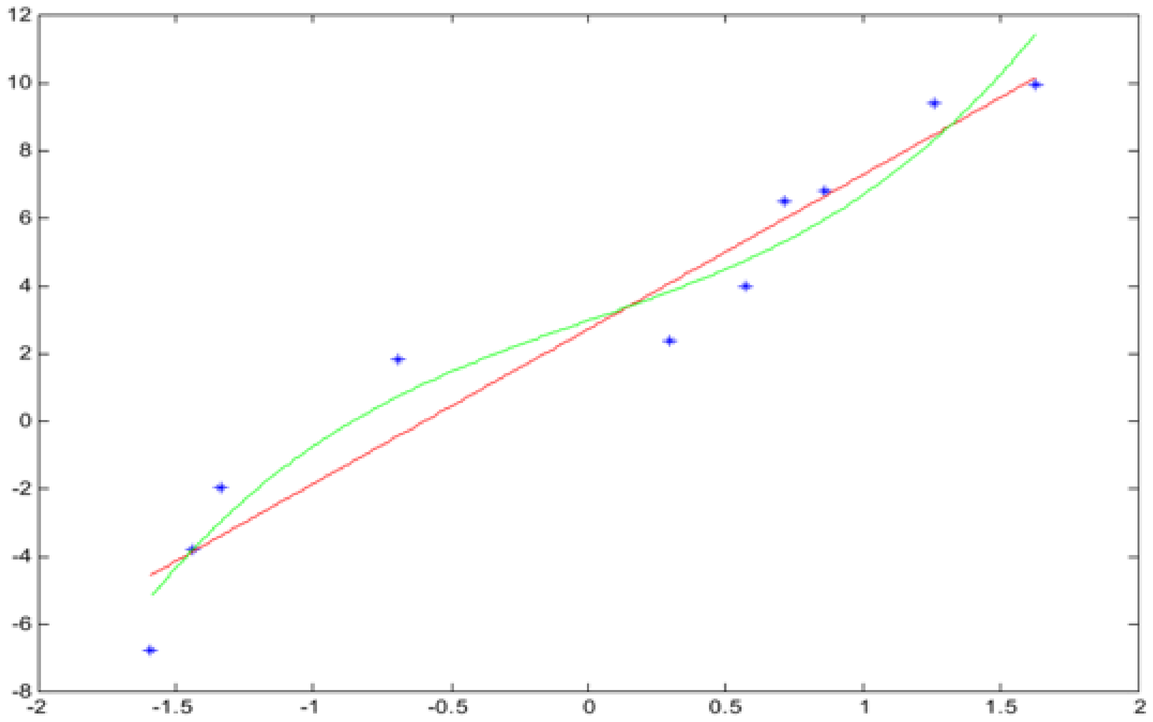
Overfitting

- Fitting a linear function with the square error
- Error is nonzero



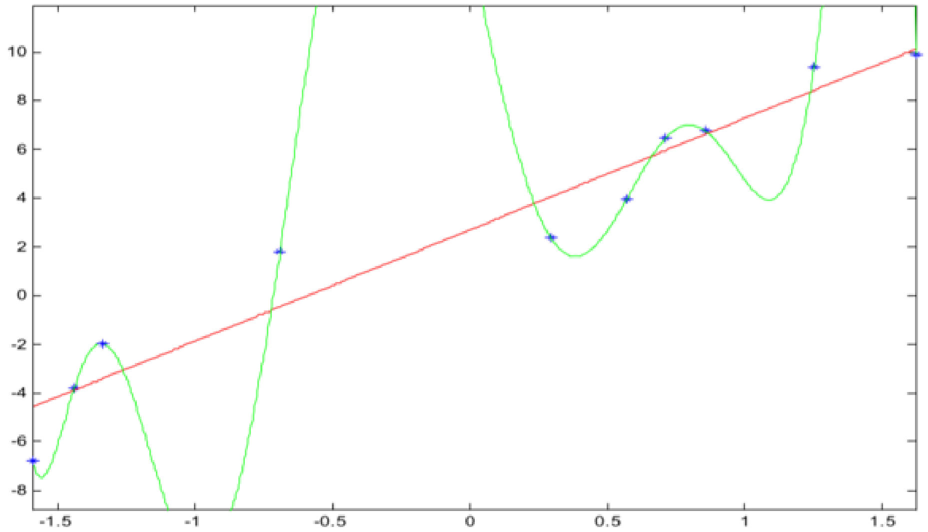
Overfitting

- Linear vs. cubic polynomial
- Higher order polynomial leads to a better fit, smaller error



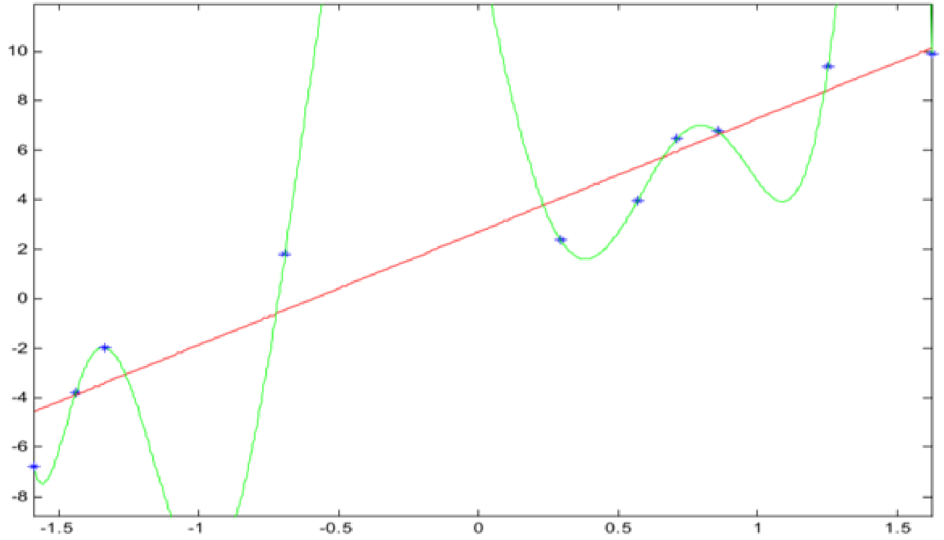
Overfitting

- For 10 data points, degree 9 polynomial gives a perfect fit (Lagrange interpolation). Error is zero.
- Is it always good to minimize the training error? NO !!



Overfitting

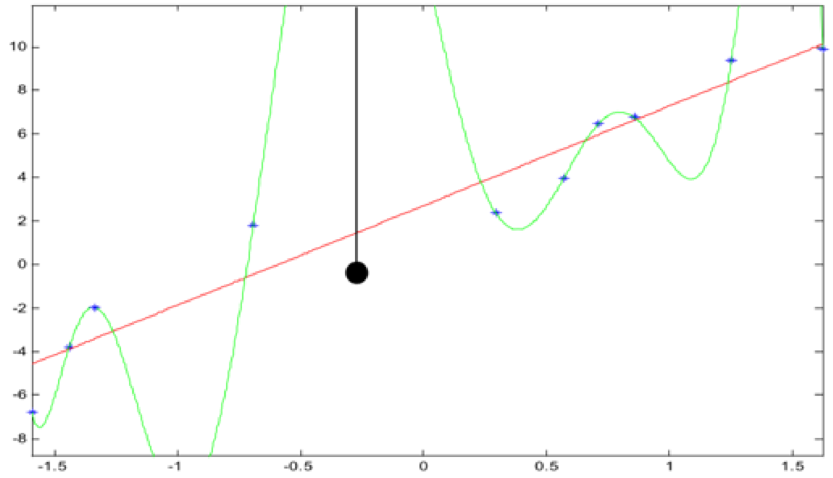
- For 10 data points, degree 9 polynomial gives a perfect fit (Lagrange interpolation). Error is zero.
- Is it always good to minimize the training error? NO !!
- **More important:** How do we perform on the unseen data?



Overfitting

Situation when the training error is low and the generalization error is high. Causes of the phenomenon:

- Model with a large number of parameters (degrees of freedom)
  - The model complexity is high, so it learns the noise within the training data.
- Small data size (as compared to the complexity of the model) to accurately represent all possible input data values.
- The training data contains large amounts of irrelevant information, called noisy data.
- The model trains for too long on a single sample set of data.



How to evaluate the learner's performance?

- **Generalization error** is the true error for the population of examples we would like to optimize

$$E_{(x,y)}[(y - f(x))^2]$$

- But it cannot be computed exactly
- **Sample mean only approximates the true mean**
- **Optimizing (mean) training error can lead to the overfit,** i.e. training error may not reflect properly the generalization error

$$\frac{1}{n}\sum_{i=1,\ldots n}(y_i - f(x_i))^2$$

- So how to test the generalization error?

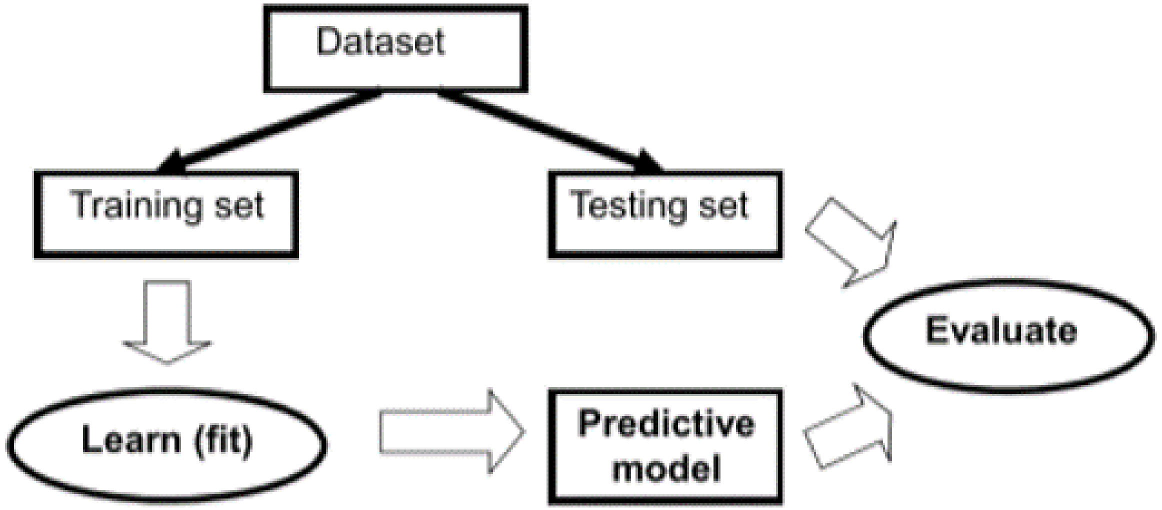
How to evaluate the learner’s performance?

- **Generalization error** is the true error for the population of examples we would like to optimize
- Sample mean only approximates it
- Two ways to assess the generalization error is:
  - **Theoretical:** Law of Large numbers
    - statistical bounds on the difference between true and sample mean errors
  - **Practical:** Use a separate data set with m data samples to test the model •
    - **(Mean) test error**

$$\frac{1}{m}\sum_{j=1,..m}(y_j - f(x_j))^2$$

Testing of learning models

- Simple holdout method
  - Divide the data to the training and test data



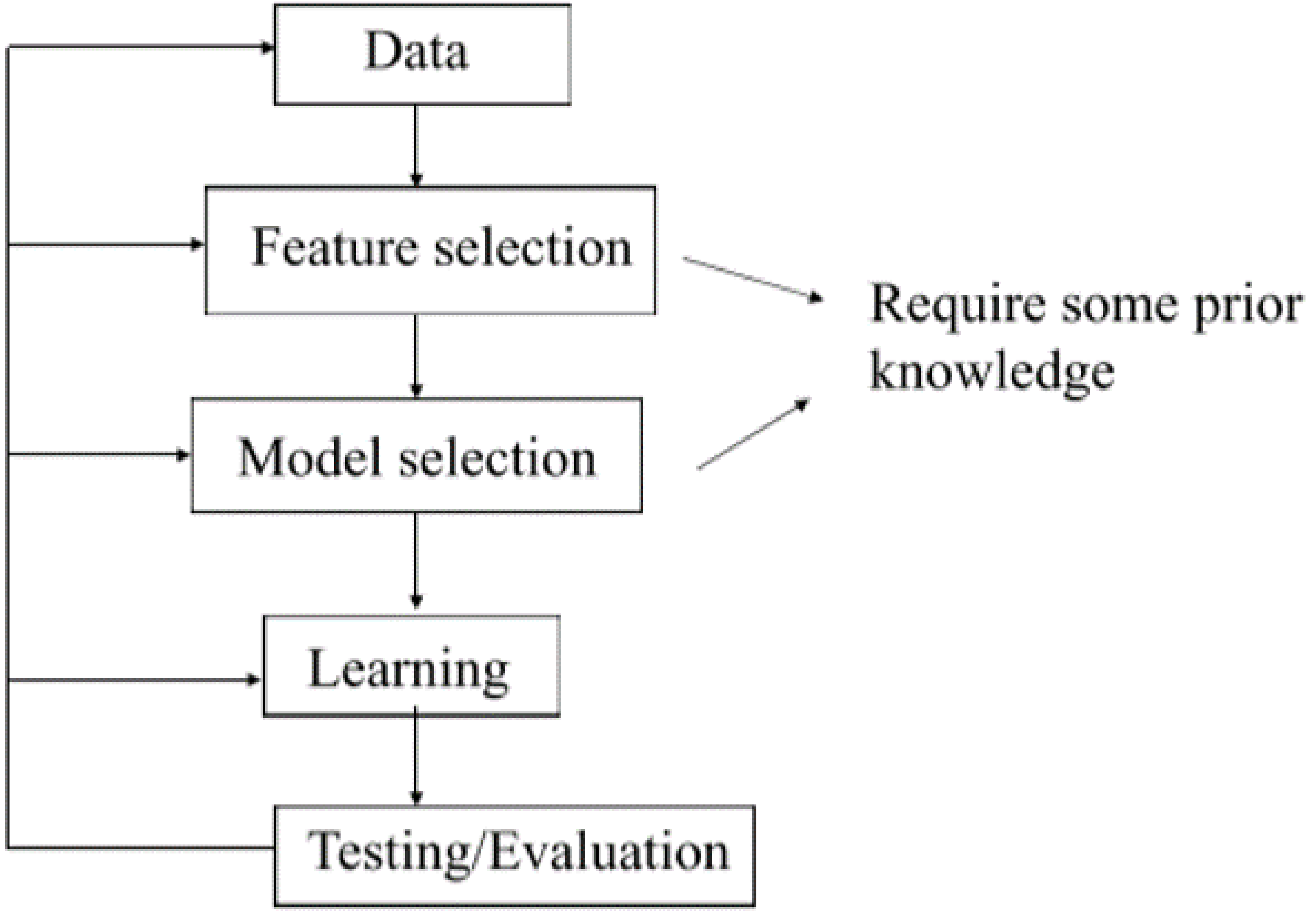
- Typically 2/3 training and 1/3 testing

Basic experimental setup to test the learner's performance

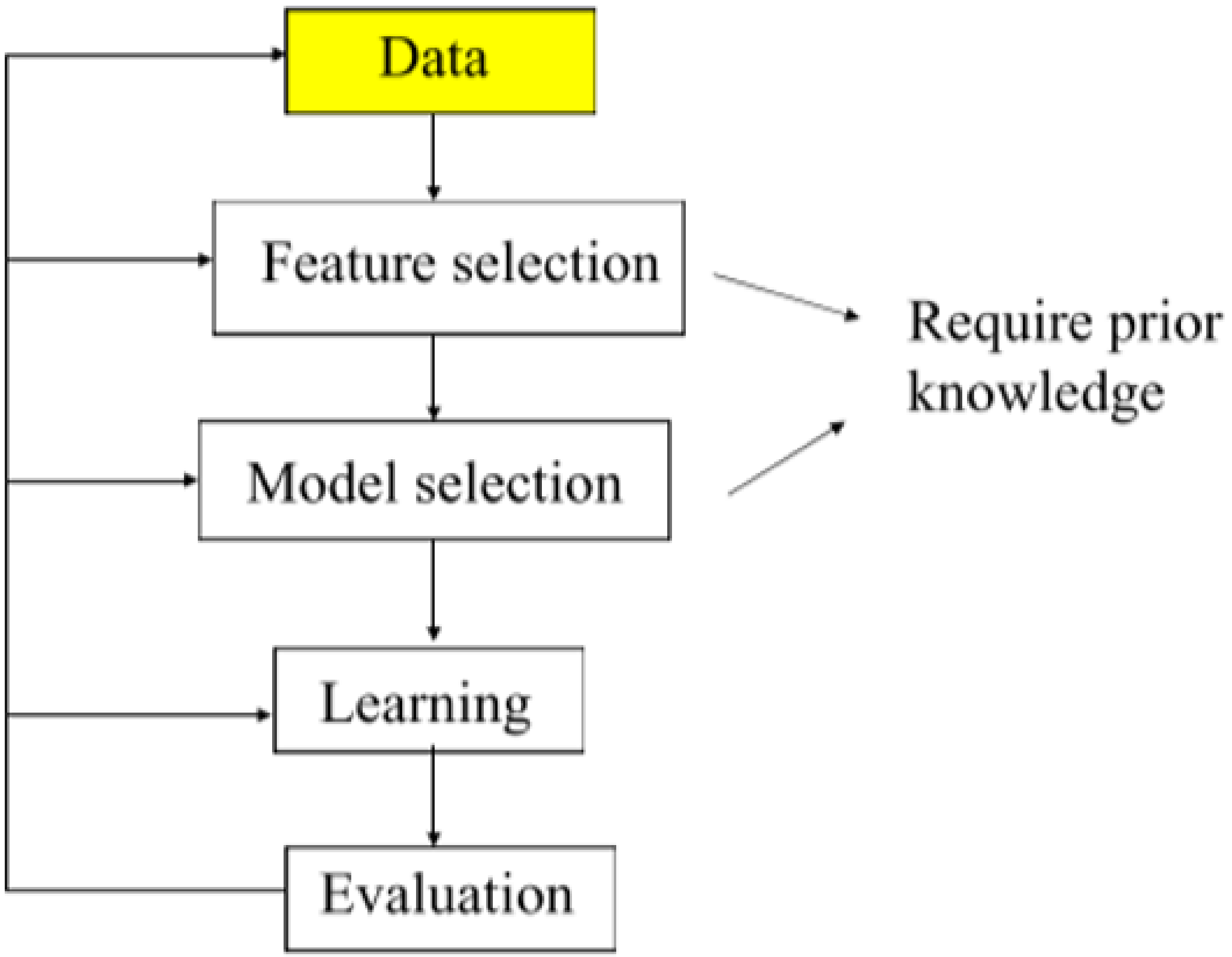
1. Take a dataset D and divide it into:
   a. Training data set
   b. Testing data set
2. Use the training set and your favorite ML algorithm to train the learner
3. Test (evaluate) the learner on the testing data set

The results on the testing set can be used to compare different learners powered with different models and learning algorithms

Design cycle



Design cycle



Data

Data may need a lot of:

- Cleaning
- Preprocessing

Cleaning:

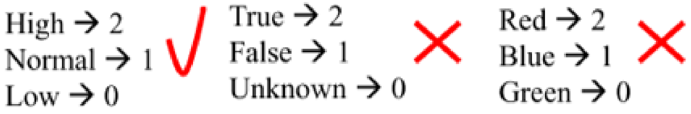
→ Get rid of errors, noise
→ Removal of redundancies

Preprocessing:

→ Renaming
→ Rescaling (normalization)
→ Discretization
→ Abstraction
→ Aggregation
→ New attributes

Data preprocessing

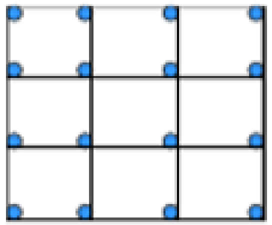
- **Renaming (relabeling)** categorical values to numbers
  - dangerous in conjunction with some learning methods
  - numbers will impose an order that is not warranted



https://www.analyticsvidhya.com/blog/2015/11/easy-methods-deal-categorical-variables-predictive-modeling/

- **Rescaling (normalization):** continuous values transformed to some range, typically [-1, 1] or [0,1].

- **Discretizations (binning):** continuous values to a finite set of discrete values



- **Abstraction:** Keep essential elements and discard the rest
- **Aggregation:** summary or aggregation operations, such minimum value, maximum value, average etc.
- **New attributes:**
  - example: obesity-factor = weight/height

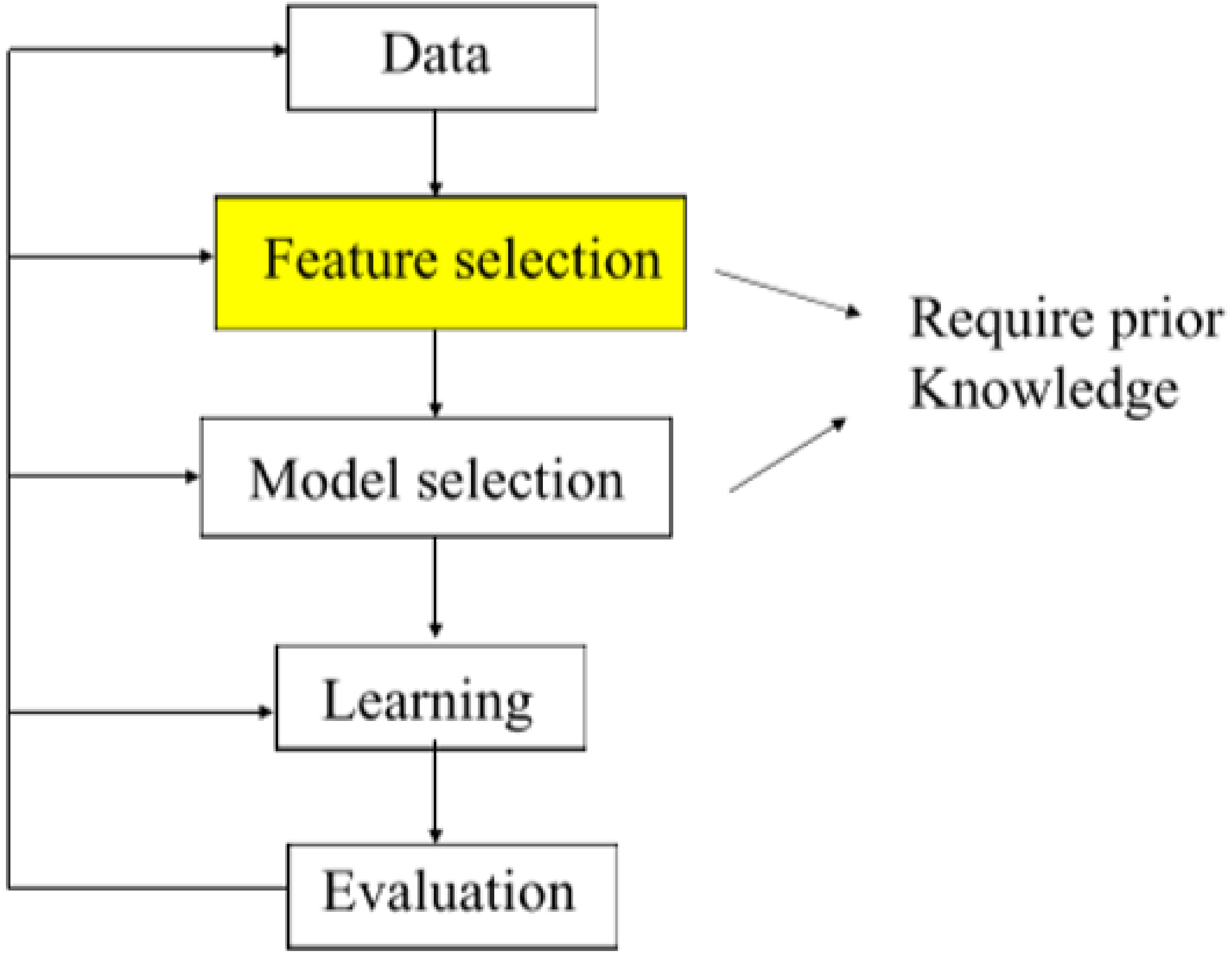
Data biases

- **Watch out for data biases:**
  - Try to understand the data source
  - Make sure the data we make conclusions on are the same as data we used in the analysis
  - It is very easy to derive "unexpected" results when data used for analysis and learning are biased (pre-selected)
- Results (conclusions) derived for a biased dataset do not hold in general !!!

Example 1: Risks in facial recognition

if facial recognition algorithms are primarily trained on data from a specific racial group, they may perform poorly on individuals from other racial backgrounds.

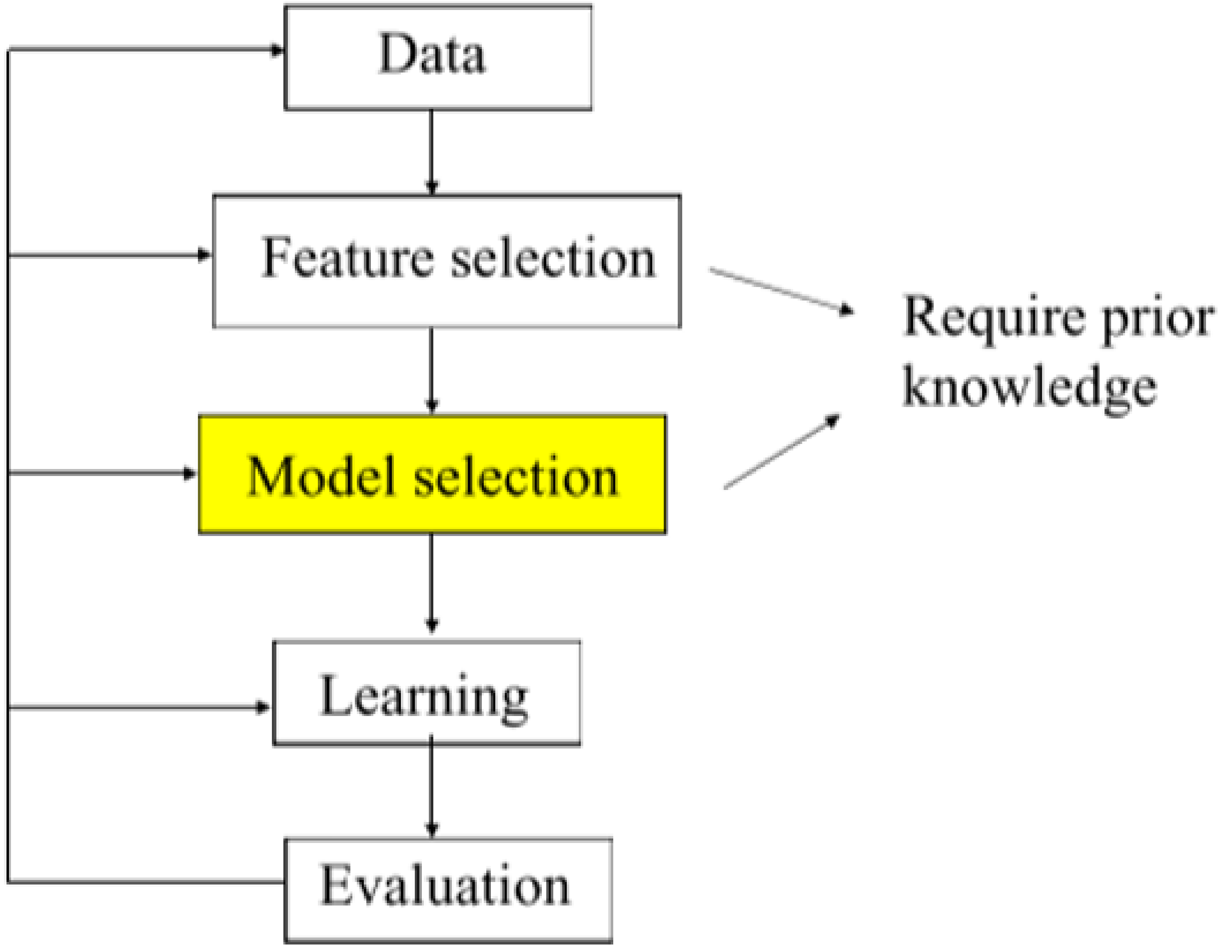
Design cycle



Feature selection

- **The size (dimensionality) of a sample** can be enormous d
- **Example: document classification**
  - **thousands of documents**
  - 10,000 different words
  - **Features/Inputs:** counts of occurrences of different words
  - Overfit threat - too many parameters to learn, not enough samples to justify the estimates the parameters of the model
- **Feature selection: reduces the feature sets**
  - **Methods for removing input features**

Design cycle



Model selection

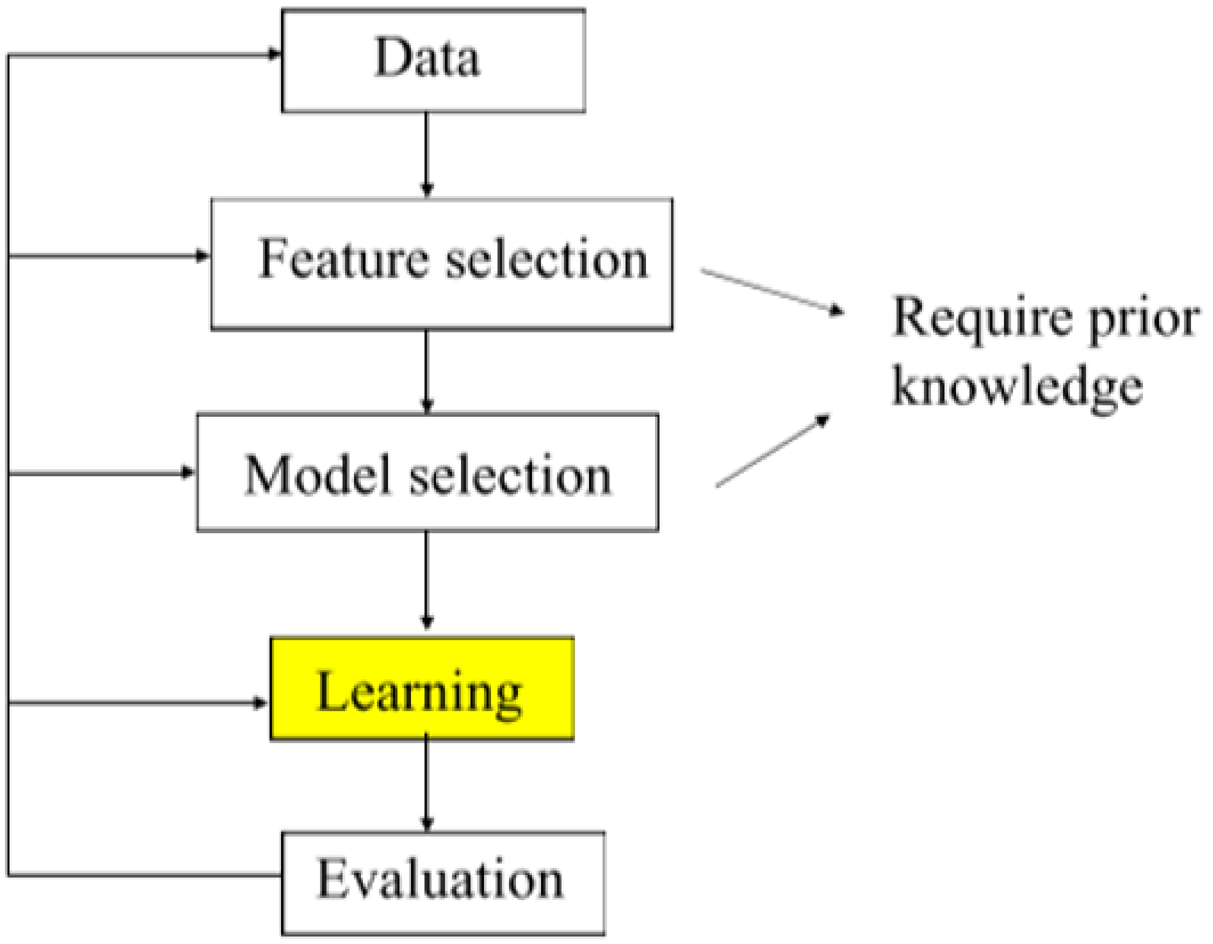
- **What is the right model to learn?**
  - A prior knowledge helps a lot, but still a lot of guessing
  - Initial data analysis and visualization
    - We can make a good guess about the form of the distribution, shape of the function
- **Overfitting problem**
  - Take into account the **bias and variance** of error estimates
    - https://www.geeksforgeeks.org/bias-vs-variance-in-machine-learning/

Solutions for overfitting

How to make the learner avoid the overfit?

- **Assure sufficient number of samples** in the training set
    - May not be possible (small number of examples)
- **Hold some data out of the training set = validation set**
    - Train (fit) on the training set (w/o data held out);
    - Check for the generalization error on the validation set, choose the model based on the validation set error (random re-sampling validation techniques)
- **Regularization (Occam's Razor)**
    - Explicit preference towards simple models
    - Penalize for the model complexity (number of parameters) in the objective function

Design cycle



Learning

- **Learning = optimization problem**. Various criteria:
  - **Mean square error**

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} Error(\mathbf{w}) \qquad Error(\mathbf{w}) = \frac{1}{N}\sum_{i=1,..N}(y_i - f(x_i,\mathbf{w}))^2$$

  - **Maximum likelihood (ML) criterion**

$$\Theta^* = \arg\max_{\Theta} P(D \mid \Theta) \qquad Error(\Theta) = -\log P(D \mid \Theta)$$

  - **Maximum posterior probability (MAP)**

$$\Theta^* = \arg\max_{\Theta} P(\Theta \mid D) \qquad P(\Theta \mid D) = \frac{P(D \mid \Theta)P(\Theta)}{P(D)}$$

Learning

Learning = optimization problem

- Optimization problems can be hard to solve. Right choice of a model and an error function makes a difference.
- **Parameter optimizations (continuous space)**
  - Linear programming, Convex programming
  - Gradient methods: grad. descent, Conjugate gradient
  - Newton-Rhapson (2nd order method)
  - Levenberg-Marquard

  Some can be carried **on-line** on a sample by sample basis

- **Combinatorial optimizations (over discrete spaces):**
  - Hill-climbing
  - Simulated-annealing
  - Genetic algorithms

Parametric optimizations

- Sometimes can be solved directly but this depends on the objective function and the model
  - **Example:** squared error criterion for linear regression
- Very often the error function to be optimized is not that nice.

$$Error(\mathbf{w}) = f(\mathbf{w}) \qquad \mathbf{w} = (w_0, w_1, w_2 \ldots w_k)$$

- a complex function of weights (parameters)

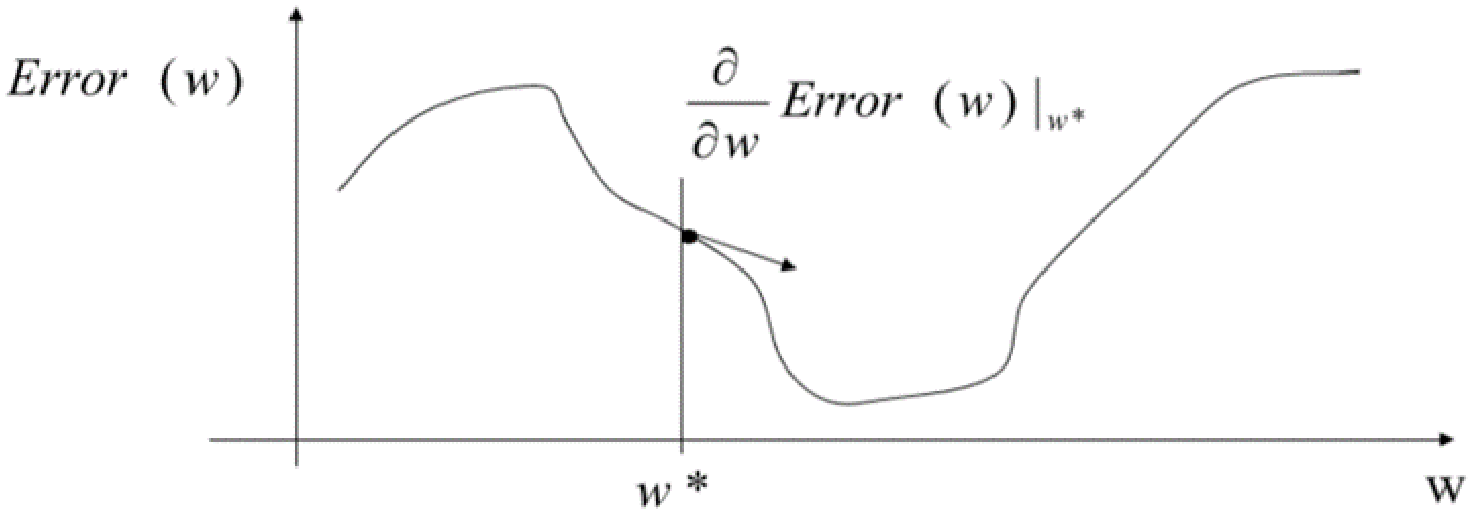
Goal: $$\mathbf{w}^* = \arg\min_{\mathbf{w}} f(\mathbf{w})$$

- **Example of a possible method: Gradient-descent method**

Idea: move the weights (free parameters) gradually in the error decreasing direction

Gradient descent method

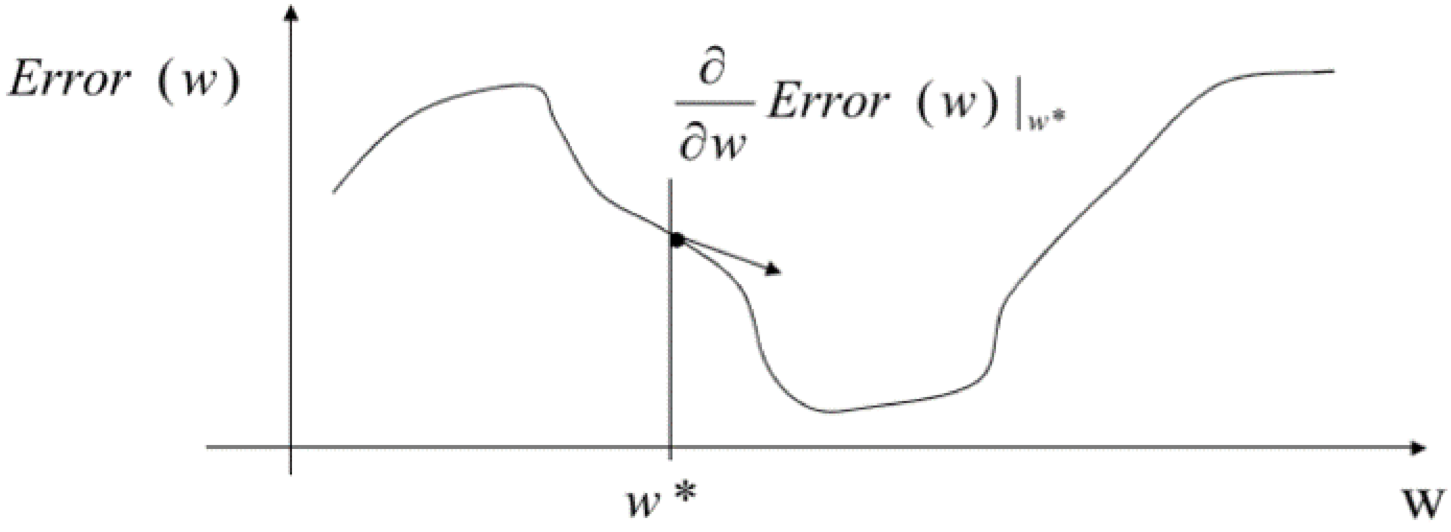
- Descend to the minimum of the function using the gradient information



- Change the parameter value of w according to the gradient

$$w \leftarrow w^* - \alpha \frac{\partial}{\partial w} Error\ (w)|_{w^*}$$

Gradient descent method



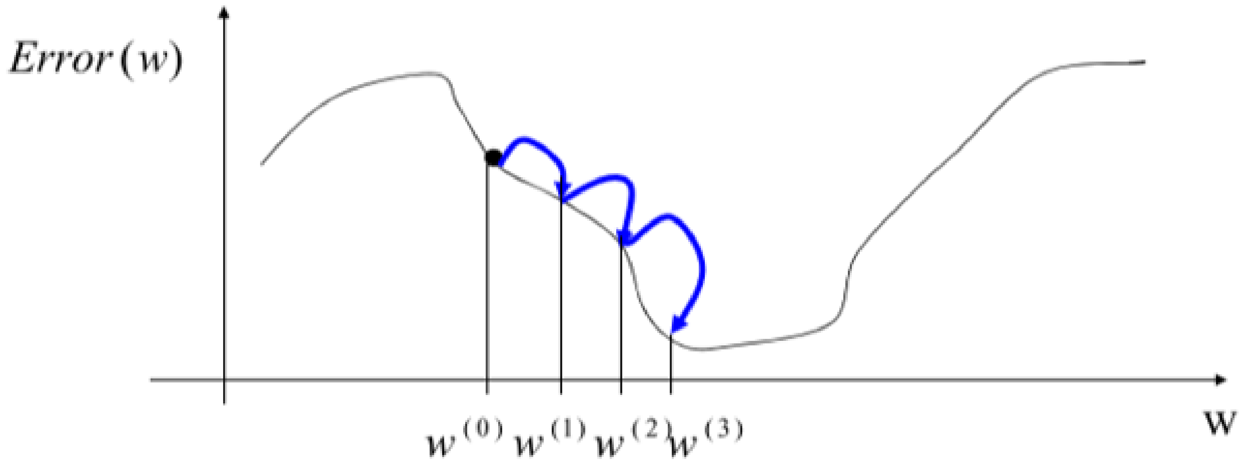
- New value of the parameter

$$w \leftarrow w^* - \alpha \frac{\partial}{\partial w} Error\ (w)|_{w*}$$

$\alpha > 0$ - a learning rate (scales the gradient changes)

Gradient descent method

- To get to the function minimum repeat (iterate) the gradient based update few times
- **Problems:** local optima, saddle points, slow convergence • More complex optimization techniques use additional information (e.g. second derivatives)



On-line learning (optimization)

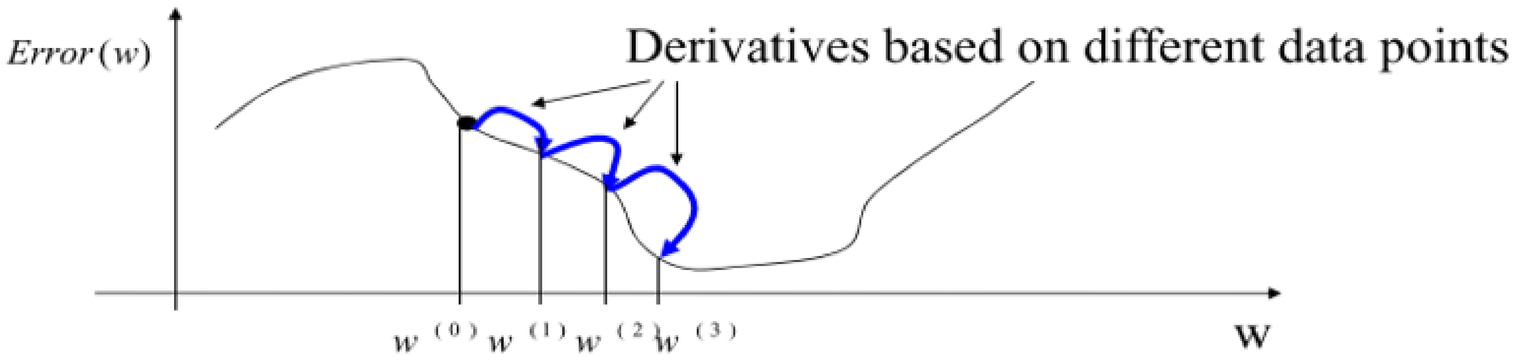
- Error function looks at all data points at the same time

  $$\text{E.g.} \quad Error(\mathbf{w}) = \frac{1}{n} \sum_{i=1,\dots n} (y_i - f(x_i, \mathbf{w}))^2$$

- **On-line error** - separates the contribution from a data point

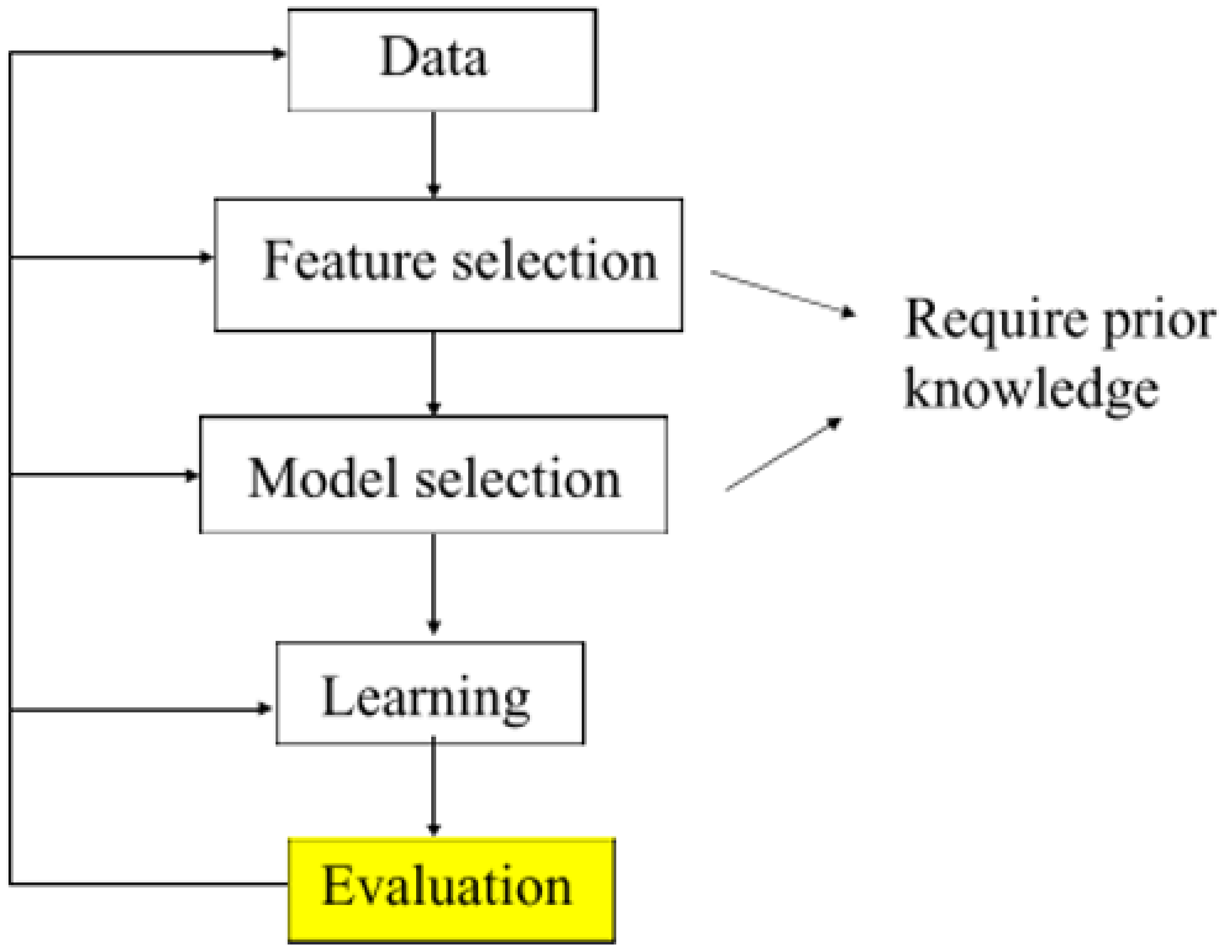
  $$Error_{\text{ON-LINE}}(\mathbf{w}) = (y_i - f(x_i, \mathbf{w}))^2$$

- **Example: On-line gradient descent**



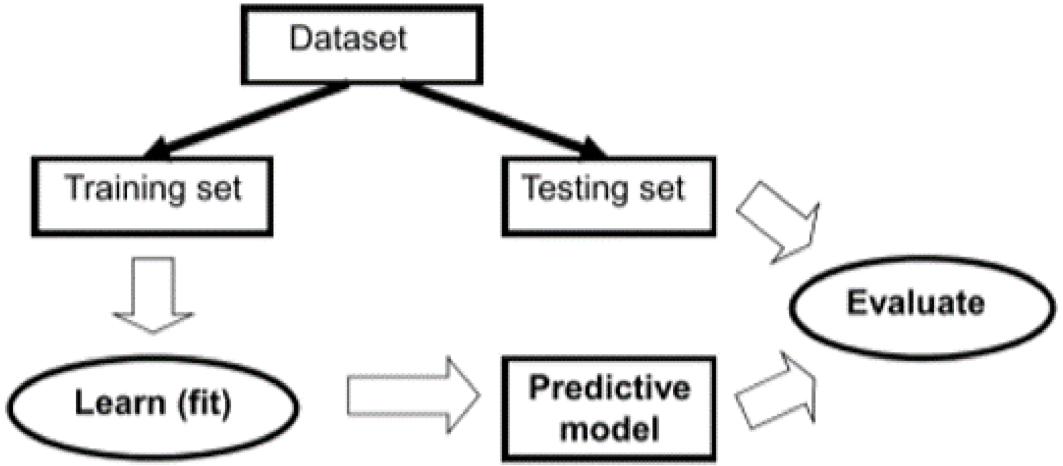
- **Advantages:**
  - simple learning algorithm
  - no need to store data (on-line data streams)

Design cycle



Evaluation of learning models

- Simple holdout method
  - Divide the data to the training and test data

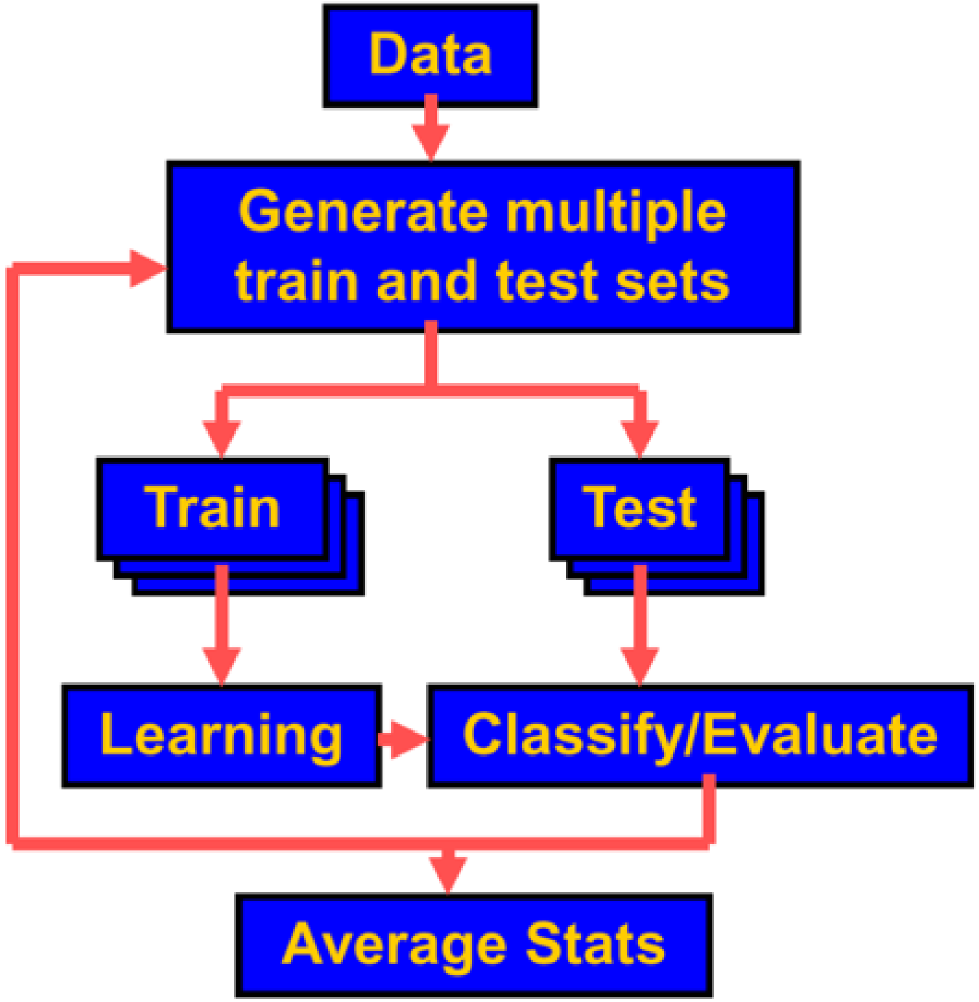


- Typically 2/3 training and 1/3 testing

Evaluation

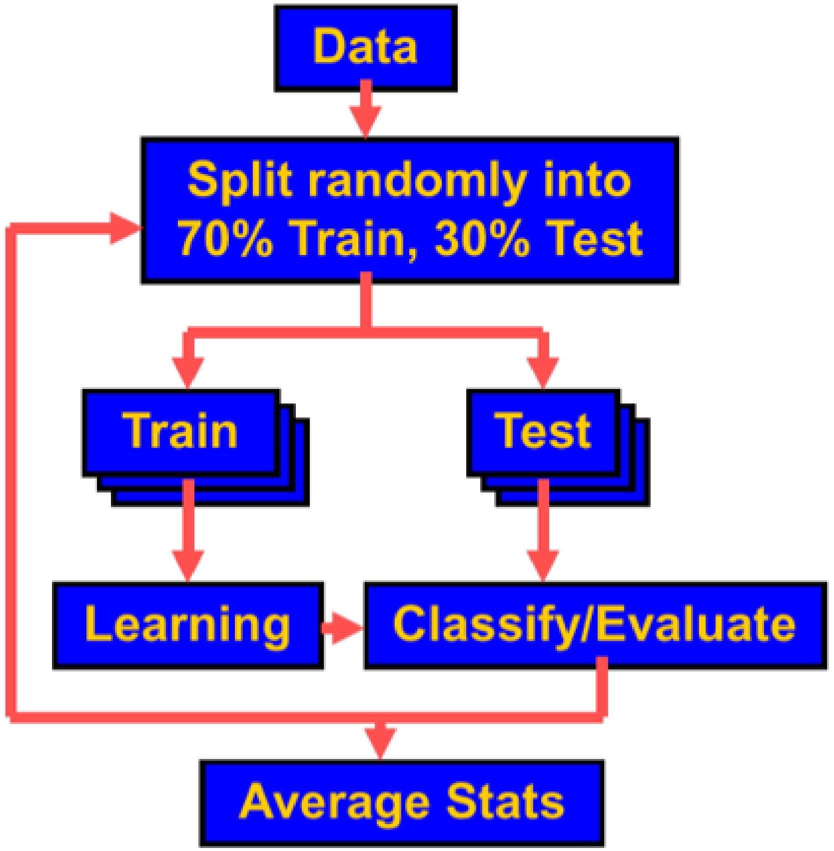
Other more complex methods

- Use multiple train/test sets
- Based on various random re-sampling schemes:
    - Random sub-sampling
    - Cross-validation
    - Bootstrap



Evaluation

- Random sub-sampling
  - Repeat a simple
  - holdout method k times

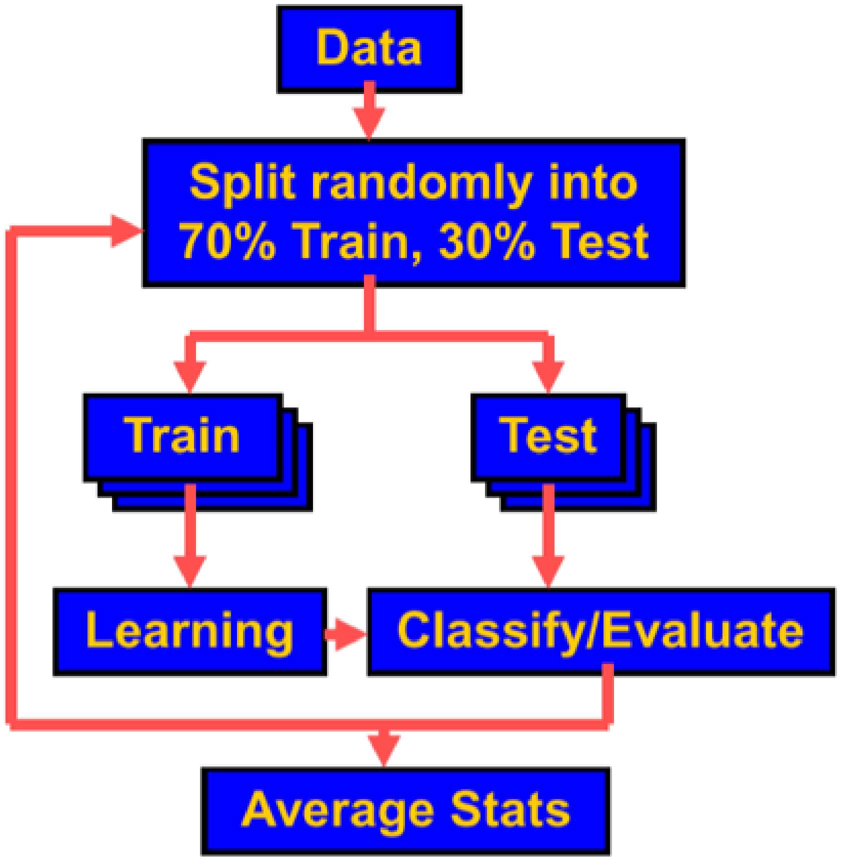


Evaluation

Cross-validation (k-fold)

- Divide data into k disjoint groups, test on k-th group/train on the rest
- Typically 10-fold cross-validation
- Leave one out cross-validation

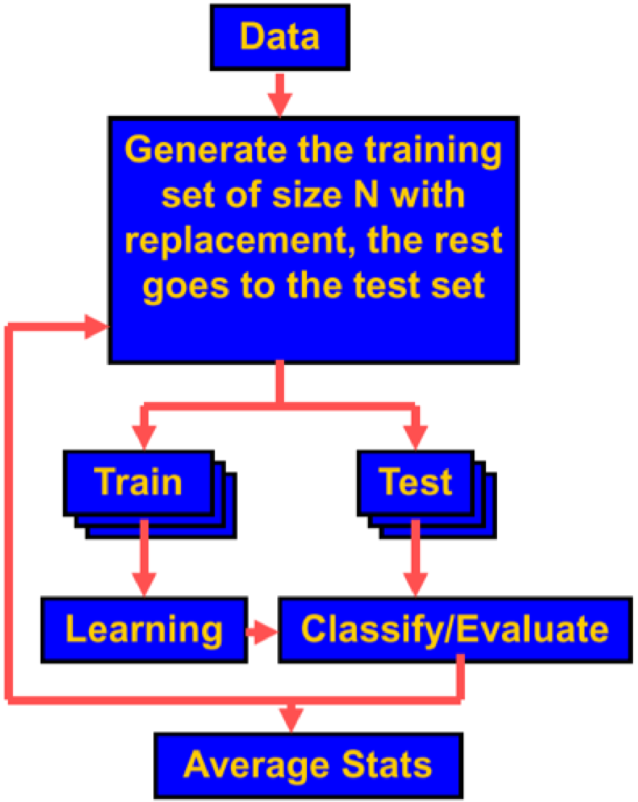
(k = size of the data D)



Evaluation

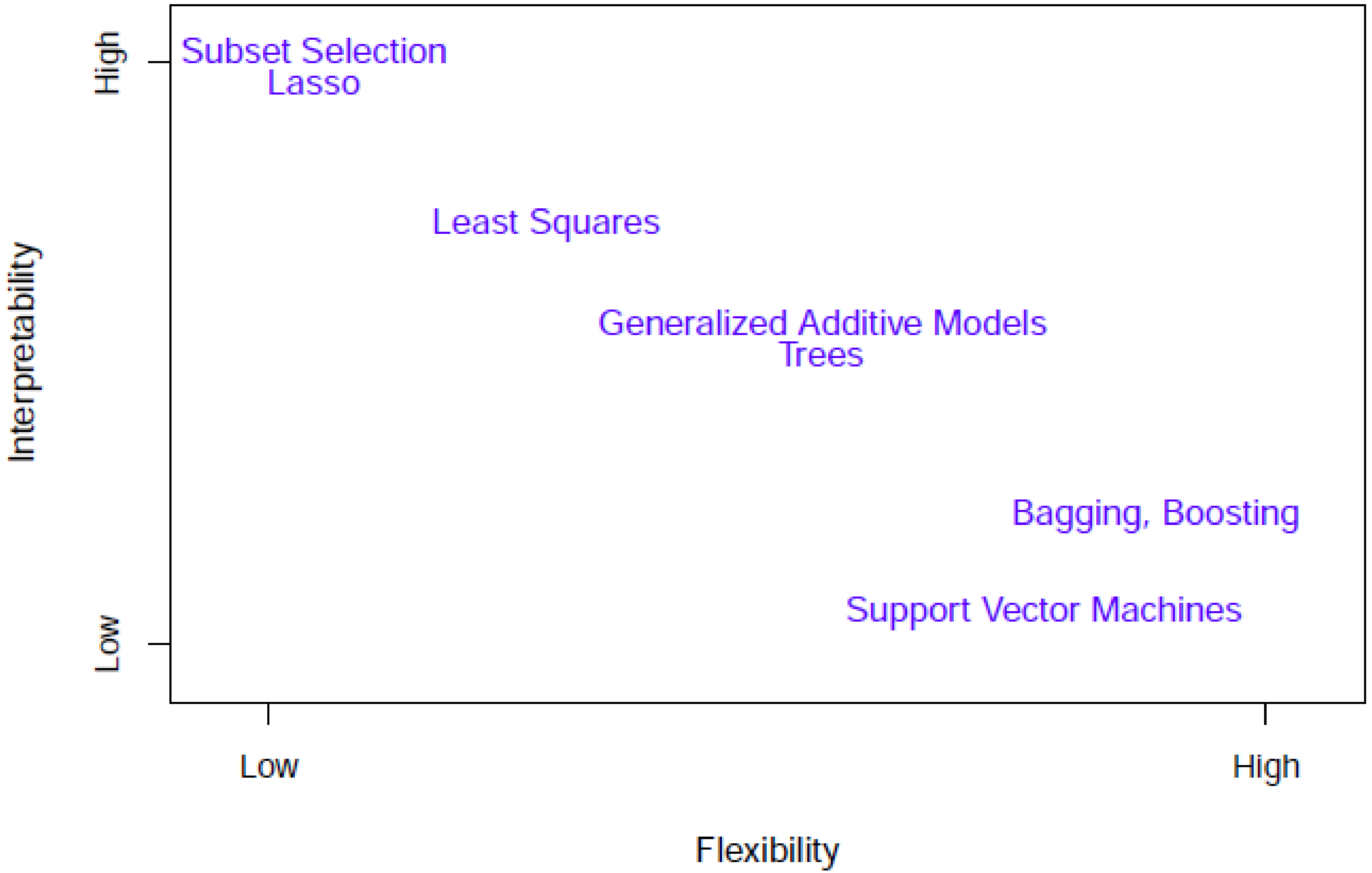
Bootstrap

- The training set of size N = size of the data D
- Sampling with the replacement



Some trade-offs

- Prediction accuracy versus interpretability.
  — Linear models are easy to interpret; thin-plate splines are not.
- Good fit versus over-fit or under-fit.
  — How do we know when the fit is just right?
- Parsimony versus black-box.
  — We often prefer a simpler model involving fewer variables over a black-box predictor involving them all.


Subset Selection
Lasso
Least Squares
Generalized Additive Models
Trees
Bagging, Boosting
Support Vector Machines
High
Low
Interpretability
Low
High
Flexibility

Assessing Model Accuracy

Suppose we fit a model $\hat{f}(x)$ to some training data $\mathsf{Tr} = \{x_i, y_i\}_1^N$, and we wish to see how well it performs.

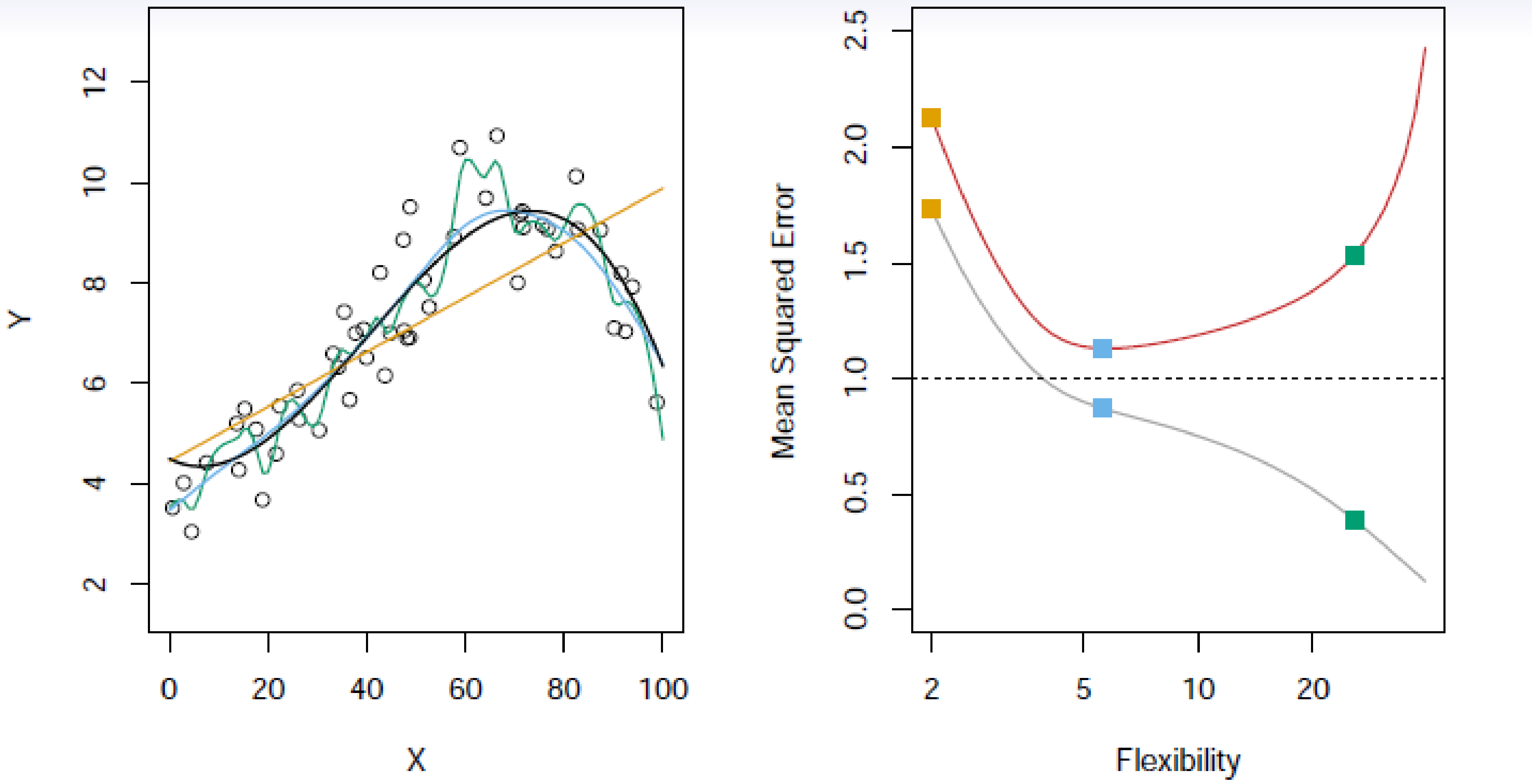
- We could compute the average squared prediction error over Tr:

$$\mathrm{MSE}_{\mathsf{Tr}} = \mathrm{Ave}_{i \in \mathsf{Tr}}[y_i - \hat{f}(x_i)]^2$$

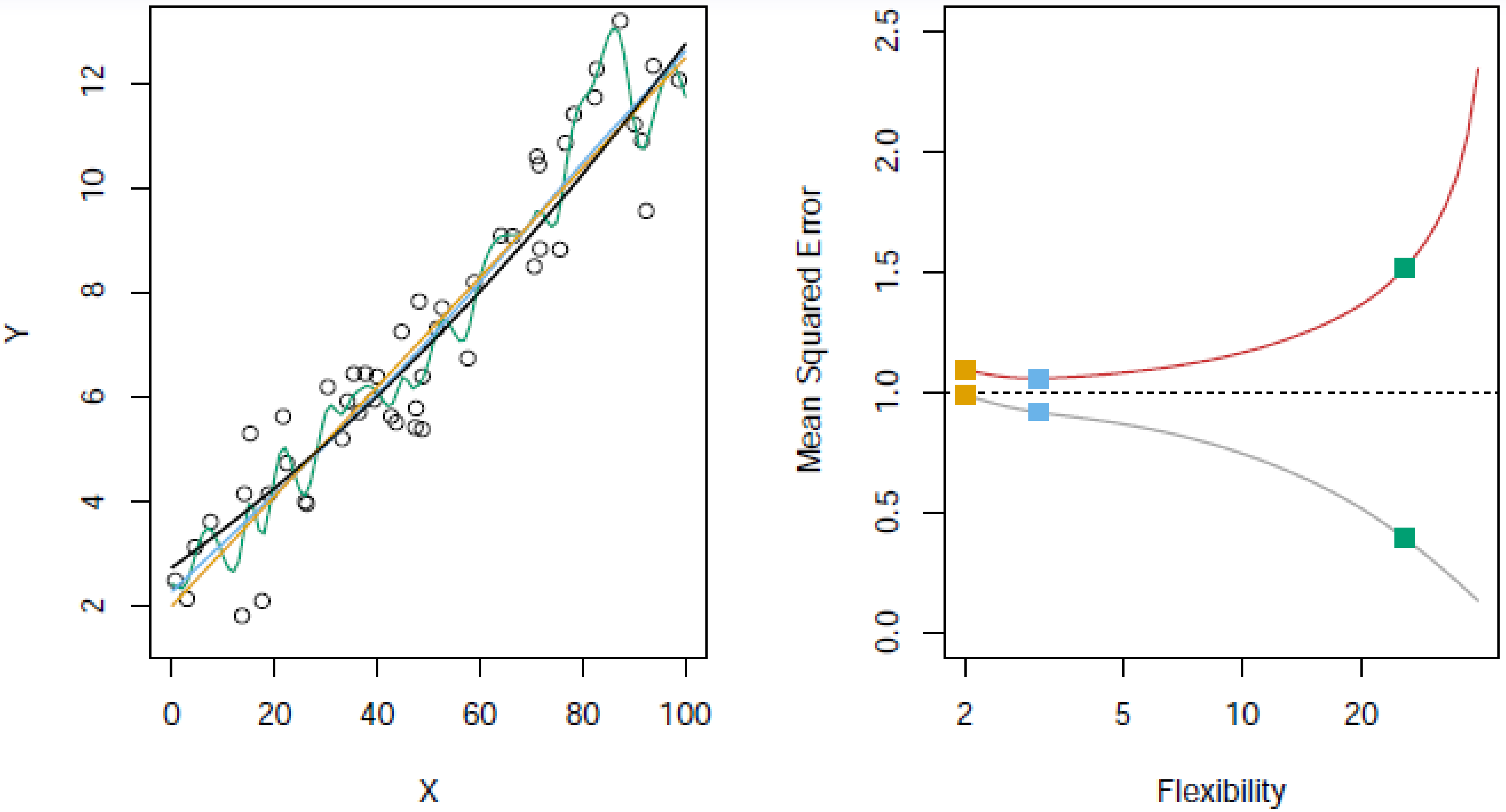
This may be biased toward more overfit models.

- Instead we should, if possible, compute it using fresh *test* data $\mathsf{Te} = \{x_i, y_i\}_1^M$:

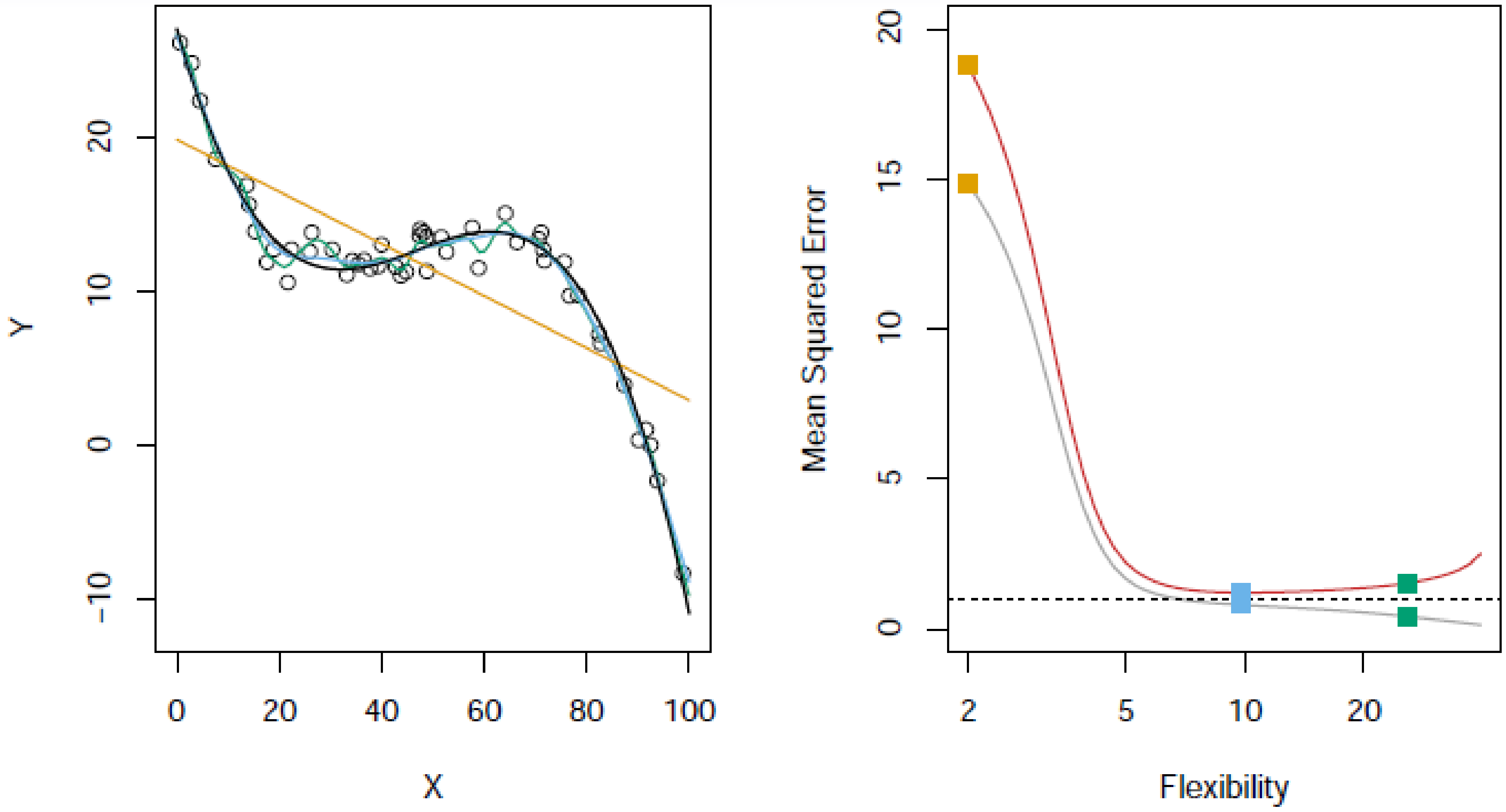
$$\mathrm{MSE}_{\mathsf{Te}} = \mathrm{Ave}_{i \in \mathsf{Te}}[y_i - \hat{f}(x_i)]^2$$



Black curve is truth. Red curve on right is $\mathrm{MSE}_{\mathrm{Te}}$, grey curve is $\mathrm{MSE}_{\mathrm{Tr}}$. Orange, blue and green curves/squares correspond to fits of different flexibility.



Here the truth is smoother, so the smoother fit and linear model do really well.



Here the truth is wiggly and the noise is low, so the more flexible fits do the best.

- Bias: *Bias* refers to the error that is introduced by approximating a real-life problem, which may be extremely complicated, by a much simpler model.

For example, linear regression assumes that there is a linear relationship between Y and $X_1, X_2, \ldots, X_p$. It is unlikely that any real-life problem truly has such a simple linear relationship, and so performing linear regression will undoubtedly result in some bias in the estimate of f.

Variance:

- *Variance* refers to the amount by which $\hat{f}$ would change if we estimated it using a different training data set.
- Different training data sets will result in a different $\hat{f}$.
- But ideally the estimate for *f* should not vary too much between training sets.
- However, if a method has high variance then small changes in the training data can result in large changes in $\hat{f}$.
- In general, more flexible statistical methods have higher variance.
- For example, the flexible green curve is following the observations very closely.
- It has high variance because changing any one of these data points may cause the estimate $\hat{f}$ to change considerably.

Bias-Variance Trade-off

- As a general rule, as we use more flexible methods, the variance will increase and the bias will decrease.
- The relative rate of change of these two quantities determines whether the test MSE increases or decreases.
- As we increase the flexibility of a class of methods, the bias tends to initially decrease faster than the variance increases.
- Consequently, the expected test MSE declines.
- However, at some point increasing flexibility has little impact on the bias but starts to significantly increase the variance.
- When this happens the test MSE increases. Note that we observed this pattern of decreasing test MSE followed by increasing test MSE in the right-hand panels of last 3 figures.

Bias-Variance Trade-off

Suppose we have fit a model $\hat{f}(x)$ to some training data Tr, and let (x_0, y_0) be a test observation drawn from the population. If the true model is $Y = f(X) + \epsilon$ (with $f(x) = E(Y|X = x)$), then

$$E\left(y_0 - \hat{f}(x_0)\right)^2 = \mathrm{Var}(\hat{f}(x_0)) + [\mathrm{Bias}(\hat{f}(x_0))]^2 + \mathrm{Var}(\epsilon).$$

Note that $\mathrm{Bias}(\hat{f}(x_0))] = E[\hat{f}(x_0)] - f(x_0)$.

Typically as the *flexibility* of $\hat{f}$ increases, its variance increases, and its bias decreases. So choosing the flexibility based on average test error amounts to a *bias-variance trade-off*.

Bias-variance trade-off for the three examples