CS 2750 Machine Learning

Lecture 2

Designing a learning system

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square, x4-8845

http://www.cs.pitt.edu/~milos/courses/cs2750/

Typical learning

Three basic steps:

- **Select a model** or a set of models (with parameters)

  E.g. $y = ax + b$

- **Select the error function** to be optimized

  E.g. $\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$

- **Find the set of parameters optimizing the error function**
  - The model and parameters with the smallest error represent the best fit of the model to the data

But there are problems one must be careful about …

Learning

Problem

- We fit the model based on past experience (past examples seen)
- 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: $\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$

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

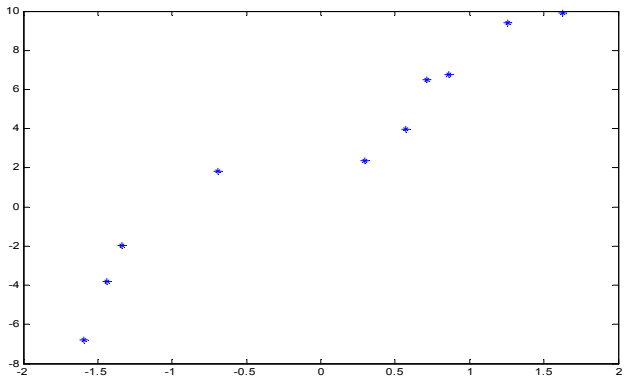
$E_{(x,y)}[(y - f(x))^2]$ Mean squared error

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

Does a good training error imply a good generalization error ?

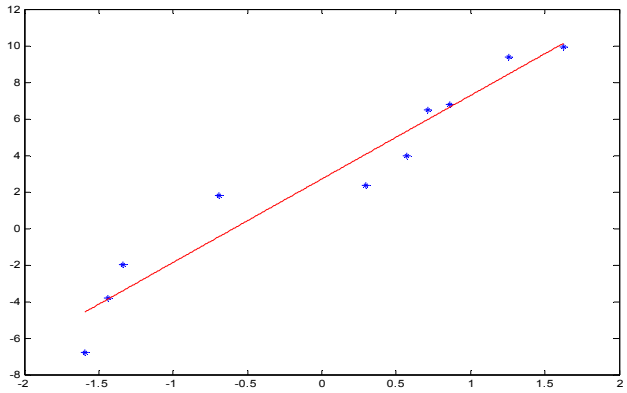
Overfitting

- Assume we have a set of 10 points and we consider polynomial functions as our possible models



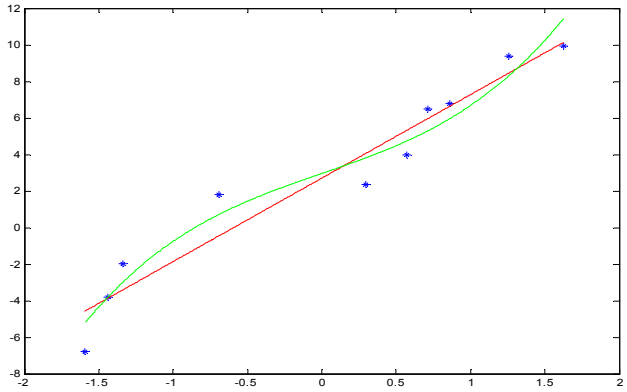
Overfitting

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



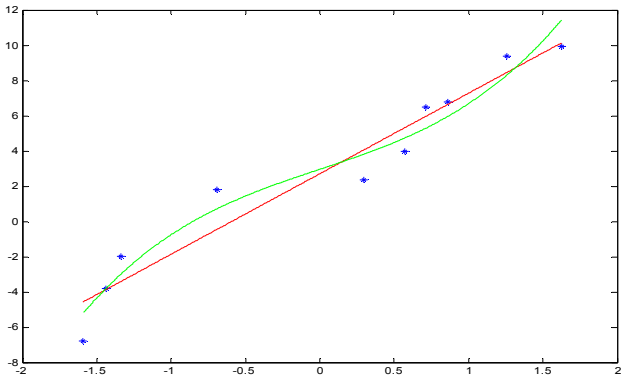
Overfitting

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



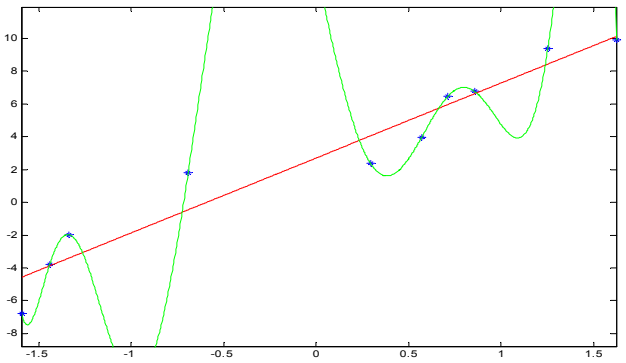
Overfitting

- Is it always good to minimize the error of the observed data?



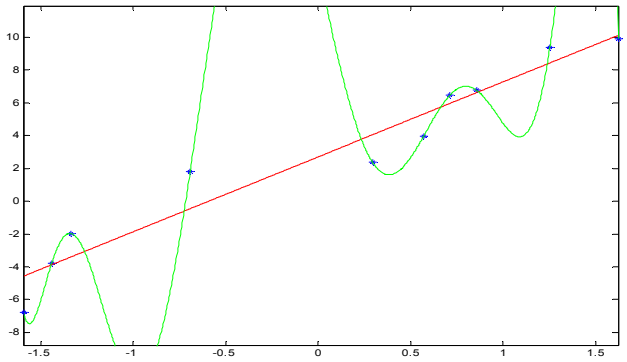
Overfitting

- For 10 data points, the degree 9 polynomial gives a perfect fit (Lagrange interpolation). Error is zero.
- Is it always good to minimize the training error?



Overfitting

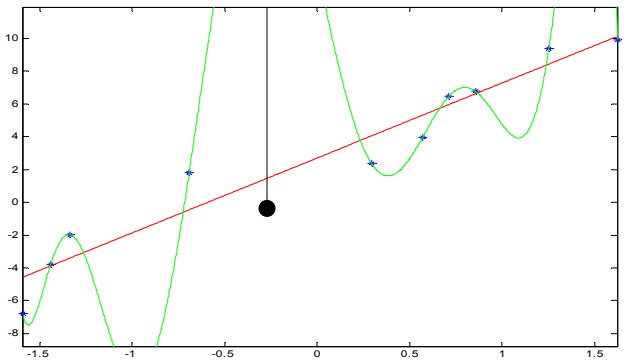
- For 10 data points, degree 9 polynomial gives a perfect fit (Lagrange interpolation). Error is zero.
- Is it always good to minimize the training error? NO !!
- **More important:** How do we perform on the unseen data?



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)
- Small data size (as compared to the complexity of the model)



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

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

- 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

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

- **Sample mean only approximates it**
- How to measure the generalization error?
- **Two ways:**
  - **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
    - **(Mean) test error** $\frac{1}{m}\sum_{j=1,..m}(y_j - f(x_j))^2$

Basic experimental setup to test the learner's performance

1. Take a dataset D and divide it into:

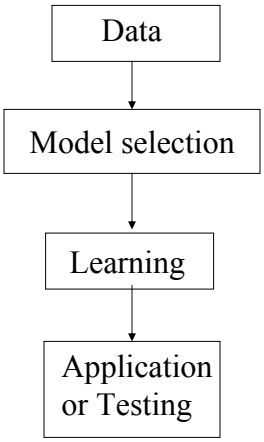
- **Training data set**
- **Testing data set**

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 of a learning system (first view)



Design of a learning system.

1. Data: $D = \{d_1, d_2, .., d_n\}$

2. Model selection:

- **Select a model** or a set of models (with parameters)

  E.g. $y = ax + b + \varepsilon \qquad \varepsilon = N(0, \sigma)$

- **Select the error function** to be optimized

  E.g. $\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$

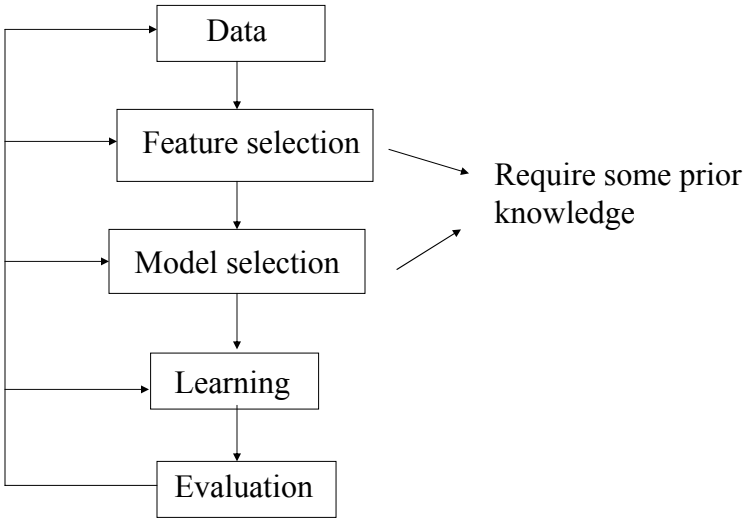
3. Learning:

- **Find the set of parameters optimizing the error function**
  - The model and parameters with the smallest error

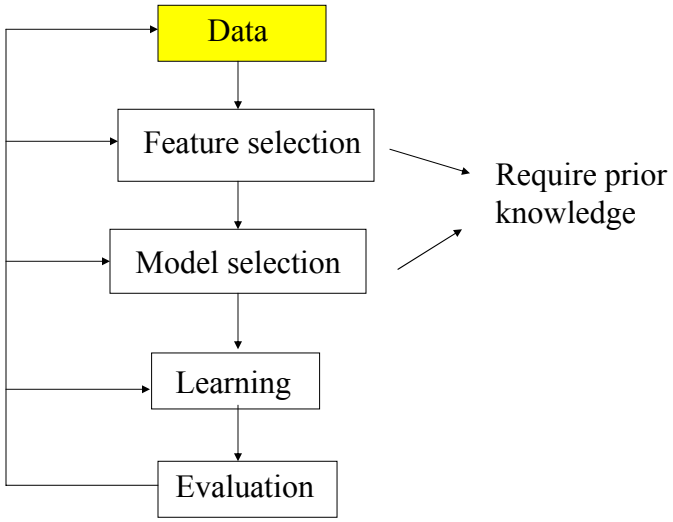
4. Application:

- **Apply the learned model**
  - E.g. predict ys for new inputs **x** using learned $f(\mathbf{x})$

Design cycle



Design cycle



Data

Data may need a lot of:

- **Cleaning**
- **Preprocessing (conversions)**

Cleaning:

- **Get rid of errors, noise**
- **Removal of redundancies**

Preprocessing:

- **Renaming**
- **Rescaling (normalization)**
- **Discretizations**
- **Abstraction**
- **Agreggation**
- **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 warrantied
- **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:** merge together categorical values
- **Aggregation:** summary or aggregation operations, such minimum value, maximum value etc.
- **New attributes:**
  - example: obesity-factor = weight/height

Data biases

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

Data biases

Example 1: Risks in pregnancy study

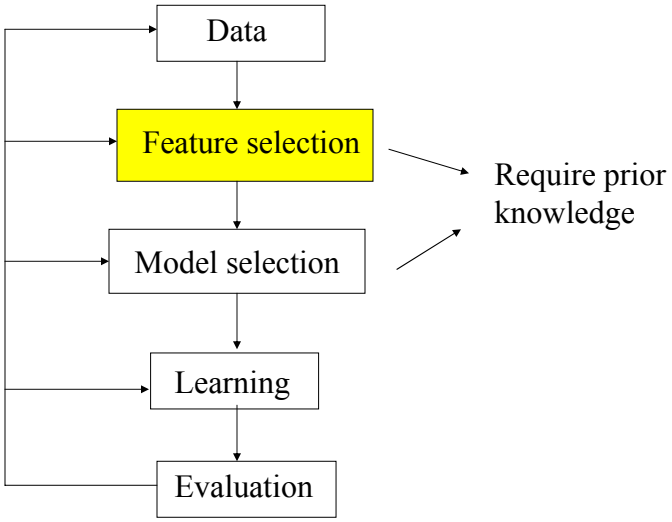
- Sponsored by DARPA at military hospitals
- Study of a large sample of pregnant woman
- **Conclusion:** the factor with the largest impact on reducing risks during pregnancy (statistically significant) is a pregnant woman being single
- Single woman -> the smallest risk
- What is wrong?

Data

Example 2: Stock market trading (example by Andrew Lo)

- Data on stock performances of companies traded on stock market over past 25 year
- **Investment goal:** pick a stock to hold long term
- **Proposed strategy**: invest in a company stock with an IPO corresponding to a Carmichael number
- **Evaluation result:** excellent return over 25 years
- Where the magic comes from?

Design cycle

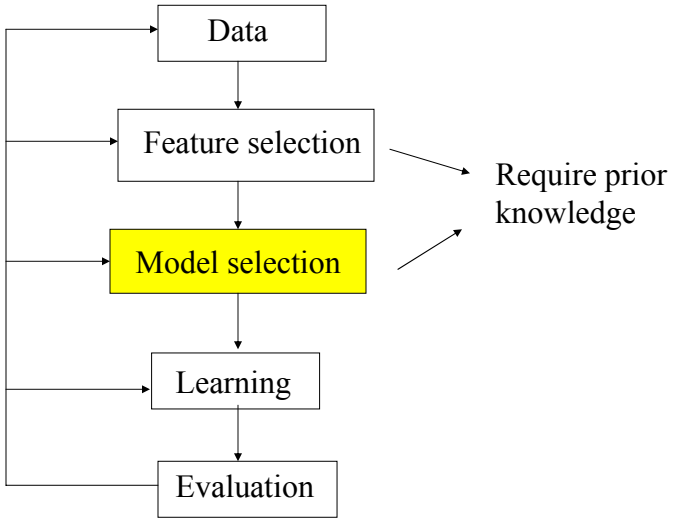


Feature selection

- **The dimensionality of a sample** can be enormous

  $x_i = (x_i^1, x_i^2, .., x_i^d)$ $\quad d$ - very large
- **Example: document classification**
  - 10,000 different words
  - Inputs: counts of occurrences of different words
  - Too many parameters to learn (not enough samples to justify the estimates the parameters of the model)
- **Dimensionality reduction: replace inputs with features**
  - **Choose relevant inputs** (e.g. differentially expressed features in DNA microarray data)
  - **Global features (**e.g. **PCA** – principal component analysis)
  - **Group (cluster) similar data points** (based on a similarity measure) and **replace a data point with the corresponding group label**

Design cycle



Model selection

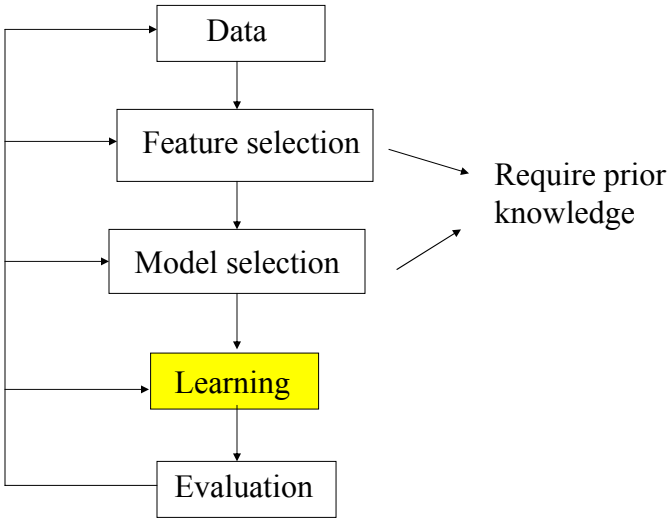
- **What is the right model to learn?**
  - A prior knowledge helps a lot, but still a lot of guessing
  - Initial data analysis and visualization
    - We can make a good guess about the form of the distribution, shape of the function
  - Independences and correlations
- **Overfitting problem**
  - Take into account the **bias and variance** of error estimates
  - Simpler (more biased) model – parameters can be estimated more reliably (smaller variance of estimates)
  - Complex model with many parameters – parameter estimates are less reliable (large variance of the estimate)

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 (cross-validation techniques)
- **Regularization (Occam's Razor)**
  - Penalize for the model complexity (number of parameters)
  - Explicit preference towards simple models

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**

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

  - **Maximum posterior probability (MAP)**

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

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**
  - Gradient descent, Conjugate gradient
  - Newton-Rhapson
  - 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 error function and the model
  - **Example:** squared error criterion for linear regression
- Very often the error function to be optimized is not that nice.

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

  - a complex function of weights (parameters)

  Goal: $\mathbf{w}^* = \arg\min_{\mathbf{w}} f(\mathbf{w})$
- One solution: **iterative optimization methods**
- **Example: Gradient-descent method**

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