Assignment 1: Recommender Systems Tips and Tricks

Wojtek Kowalczyk

To Do:

- Fetch Movielens 1M data set (http://grouplens.org/datasets/movielens/)

- Implement:
  - 4 Naive Approaches: the first 4 formulas from slide 17
  - Matrix factorization with Gradient Descent as described in the paper gravity-Tikk.pdf (the beginning of section 3.1)
  - *(optionally) Implement the ALS algorithm (ALS.pdf)*

- Estimate accuracy of your models (5-fold cross-validation)

- Document your findings (in the notebook)

Naïve models

- R_{global}(User, Item)=mean(all ratings) : TRIVIAL
- R_{item}(User, Item)=mean(all ratings for Item) : TRIVIAL
- R_{user}(User, Item)=mean(all ratings for User) : TRIVIAL
- **$R_{user\text{-}item}$(User, Item)=α*R_{user}(User, Item) + β*R_{item}(User, Item) + γ**
  (parameters α, β, γ estimated with Linear Regression)

- The data consits of triplets: <user_id, movie_id, rating> (three columns)
- We want to approximate the 'rating column', as a linear combination of 2 columns:
  R_user, R_item and a constant term γ.
  In matrix notation: **INPUT= <R_user, R_item, 1>**, (3 columns) ; we want to find **A=[α, β, γ]' ,**
  such that the norm (RMSE): **|INPUT*A - rating|** is minimized.
- To solve this "least squares regression problems" use Numpy package np.linalg.lstsq

Matrix factorization with Gradient Descent

- Implement the “double loop” algorithm (page 24, gravity-Tikk):

```
initialize all parameters
for i=0:number of iterations
    for j=0:number of records in the training set
        update parameters “responsible” for the j-th record
    end
end
```

- Experiment with a few settings (to see that the suggested setting is good)
- Document your findings , comparing results to the naïve algorithms
- Be aware that a single iteration may take a few seconds, so a single run of the algorithm may take an hour (or so)!

Matrix Factorization: Gradient Descent with Regularization (prevents overfitting) [page 29, gravity-Tikk.pdf]

1. Initialize all variables at random
2. Iterate over all records (user, item, rating):

```
    calculate the error:
        $err = rating – u_{user}*v_{item}$
    update parameters $u_{user}$ and $v_{item}$:
        $u_{user}=u_{user}+lrate*(err*v_{item} - lambda*u_{user})$
        $v_{item}=v_{item}+lrate*(err*u_{user} - lambda*v_{item})$
```

until convergence.

Typical values: lrate=0.001; lambda=0.01

General Advice

• Keep in mind that there are “gaps” in numbering of users and items. (Dictionaries? Renumber everything? …)

• When running cross-validation, the samples might have gaps in numbering, even if the gaps in original data set are already “fixed” (e.g. some user in the test fold might not be present in the training set) – how will you handle such cases?

• Remember about post-processing: squeeze output to [1,5] and implement fall-back rules

• Avoid nested loops! They are slow!
They can be avoided with operations on matrices

Writing the report: general tips

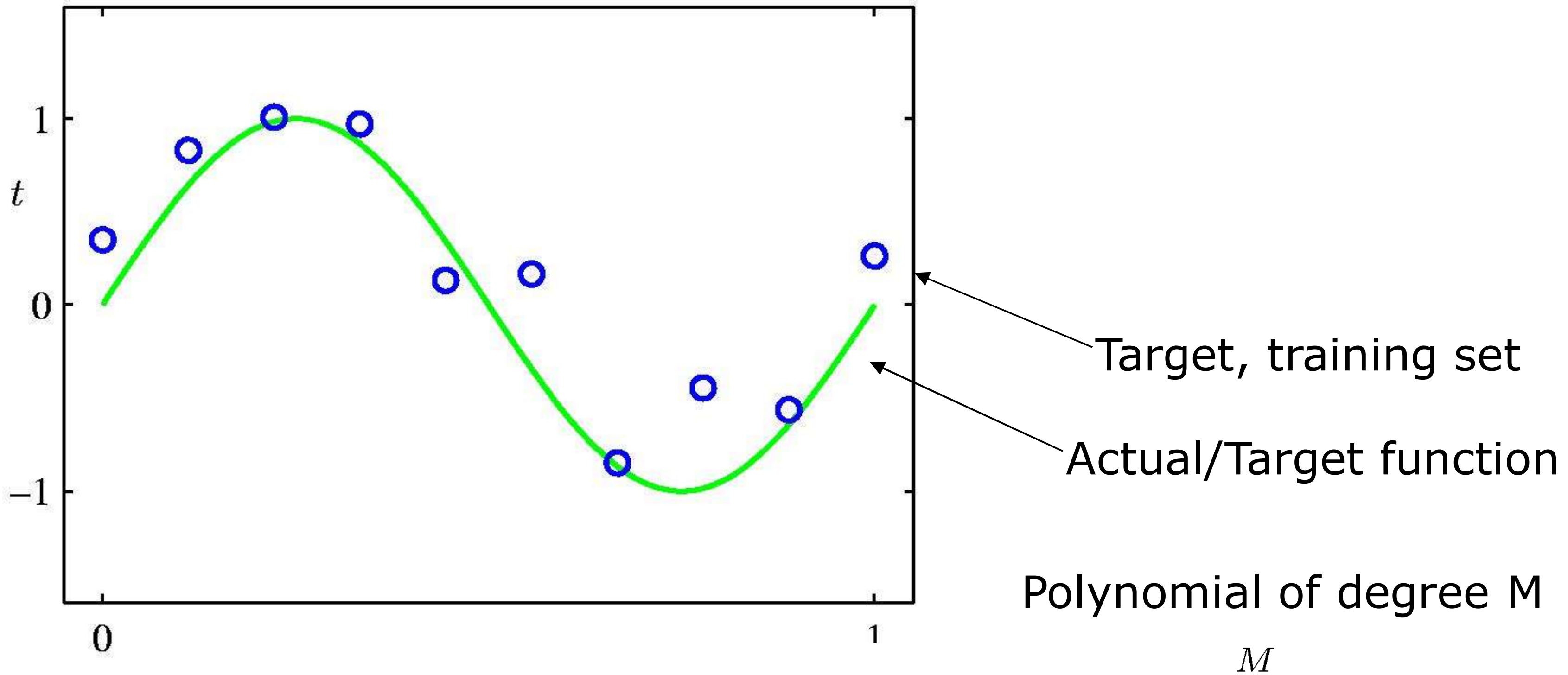
1. The notebooks (one per task) should be self-contained: the reader/TA should be able to understand the whole story without looking into your code, reading instructions, or checking other papers.
2. The notebooks should be well-structured: for each task explain what you were supposed to do, what you actually did, what are the results and, most important, draw some conclusions. When explaining your experiments, specify the setup (values of parameters you used, algorithms, options, etc.), so the reader could get a fair picture of what you did, without checking your code.
3. In case you see something strange in your results, write something about it! Most important: try to explain possible reasons for the strange things you've observed!
4. Make sure that your notebook looks well: spell-checked, tables and figures contain informative captions, your name(s) and e-mail address(es) are provided, etc.
5. Make your notebooks is concise.

Overfitting, cross-validation, regularization, ...

1. What is overfitting?
2. What is cross-validation and do we use it?
3. What is regularization and how do we work with it?
4. Tuning learning algorithms (learning rate, initialization, batch mode)

5. *What is time and memory complexity? O(n) notation*
6. *Internals of a computer*
7. *Numpy vs Python: efficiency comparison (e.g., matrix multiplication)*

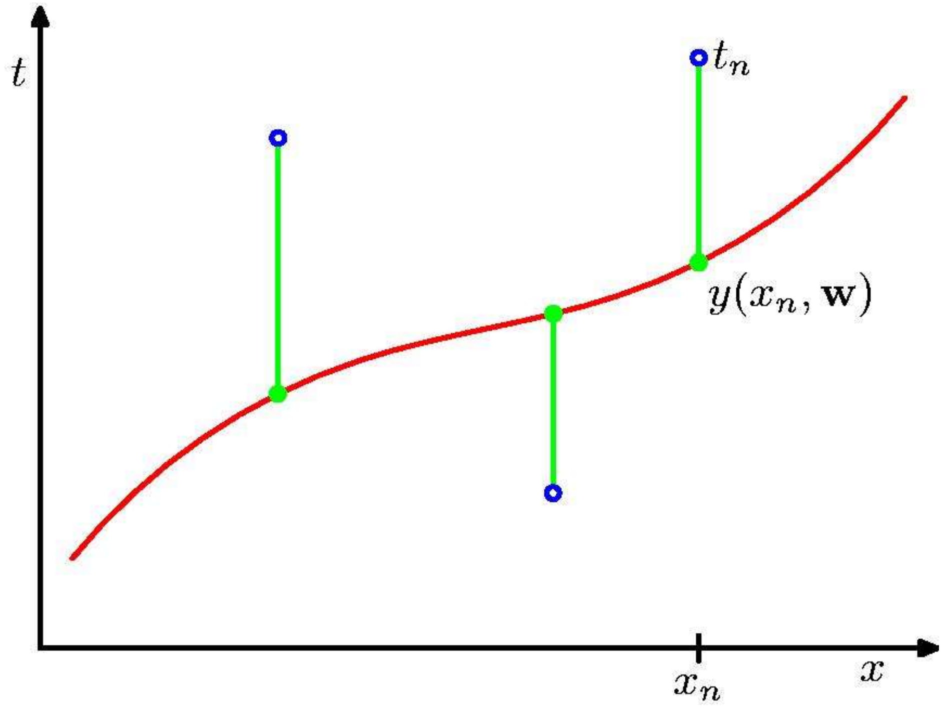
Example: Polynomial Curve Fitting



Polynomial of degree M

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

Sum-of-Squares Error Function



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

Through optimization,
we need to find the model (red)
y(x,w) that minimizes the
error function.

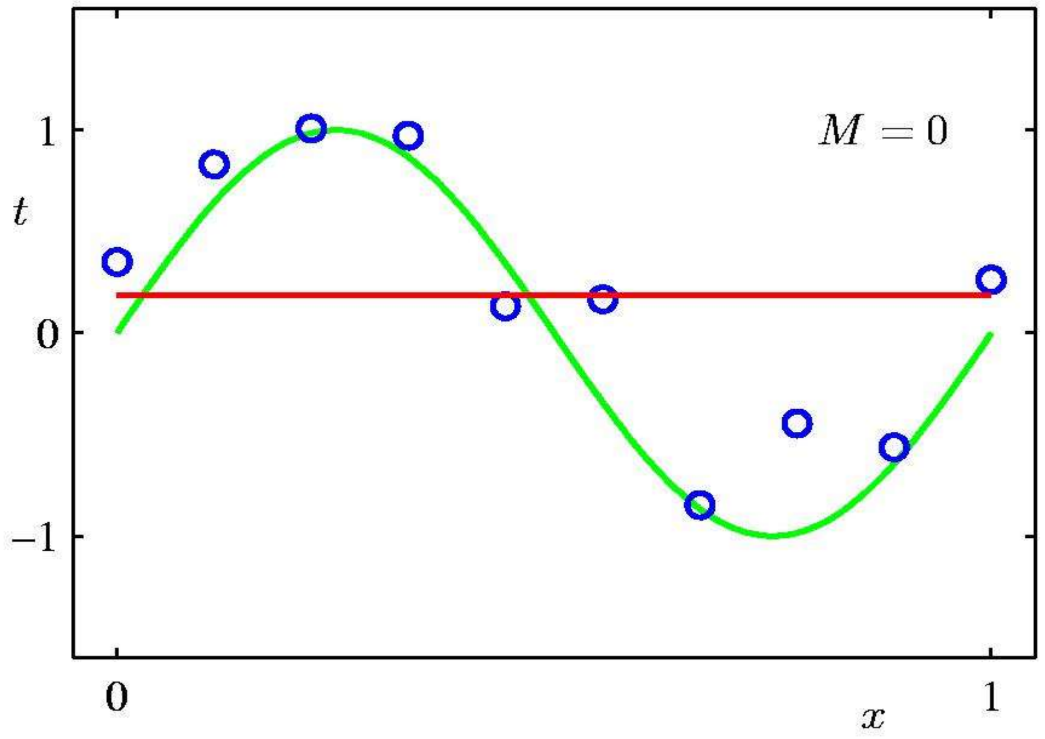
E_{SSE}(w): Sum squared error: 2E(w)

E_{RMS}(w): Root mean squared error

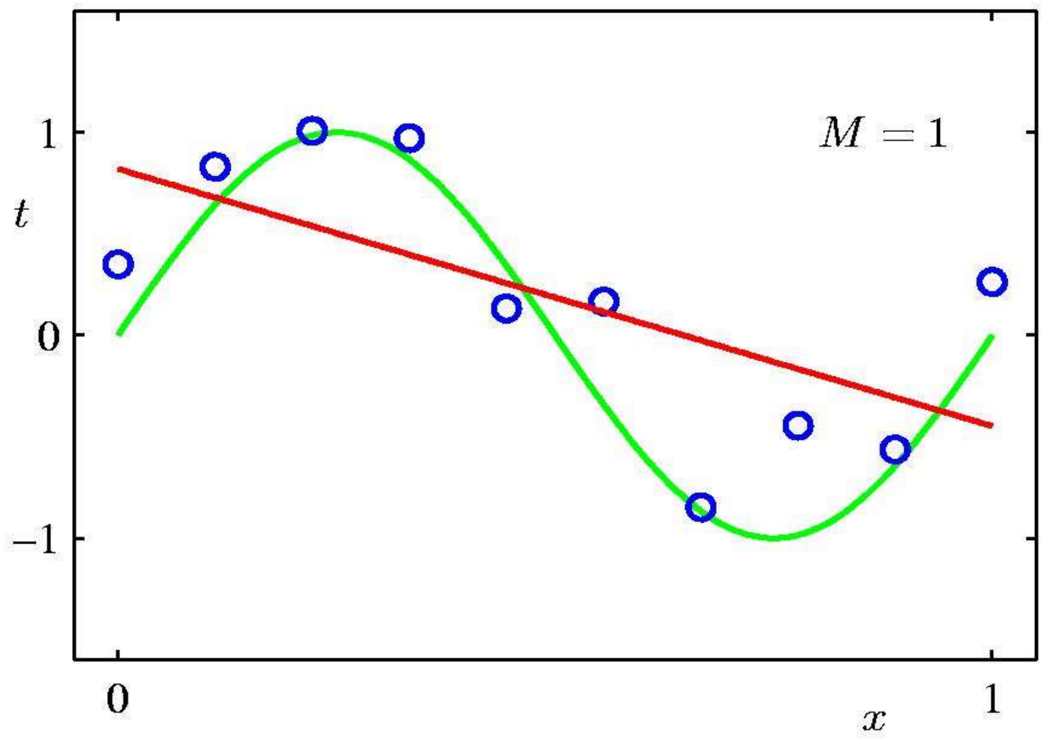
$$E_{\mathrm{RMS}} = \sqrt{2E(\mathbf{w}^\star)/N}$$

Finding optimal coefficients:
the ***polyfit.m*** function.

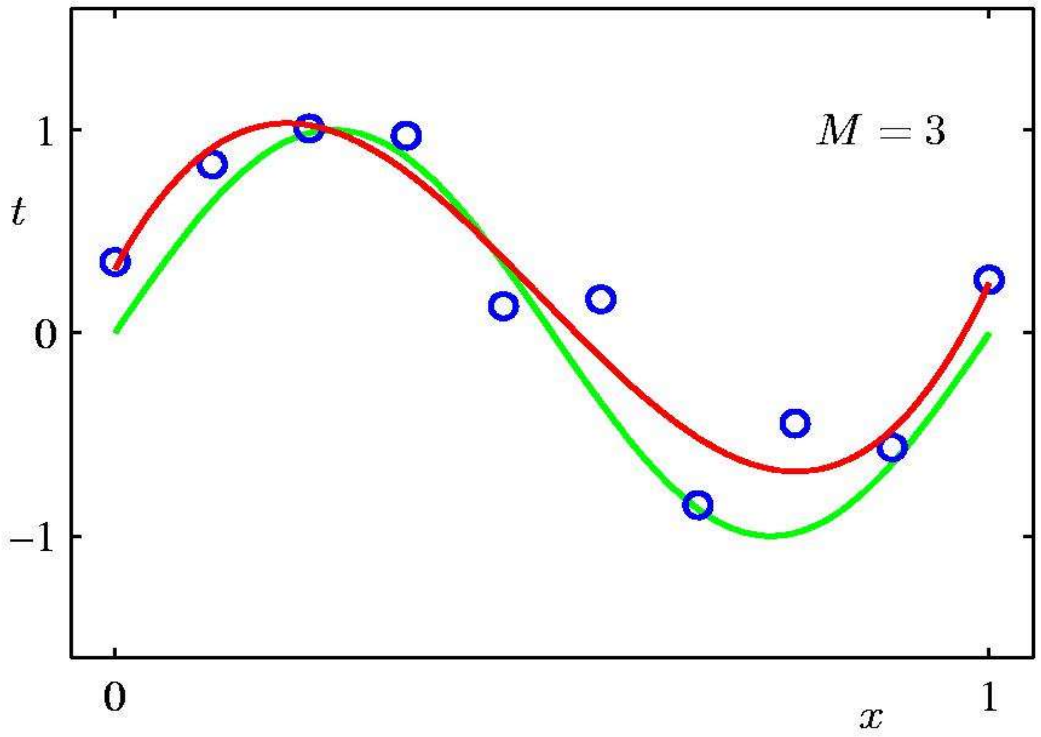
0th Order Polynomial



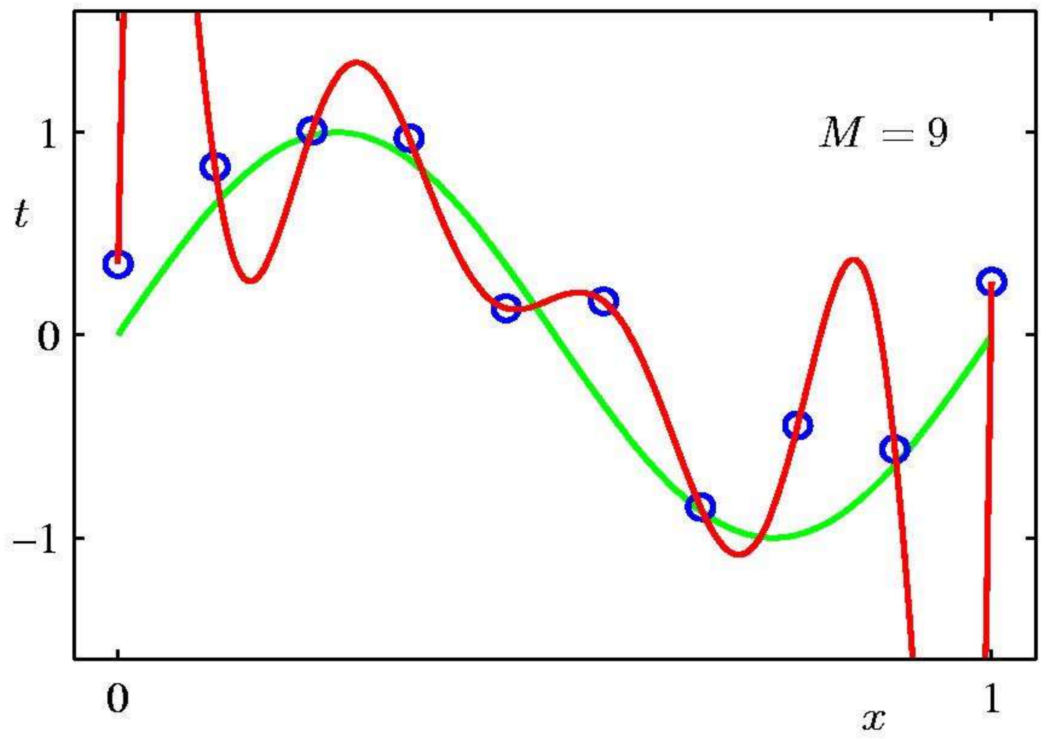
1st Order Polynomial



3rd Order Polynomial



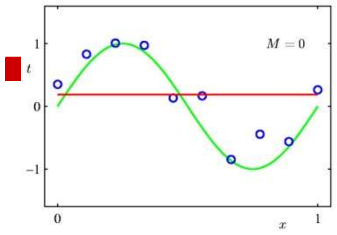
9th Order Polynomial: a disaster!!!

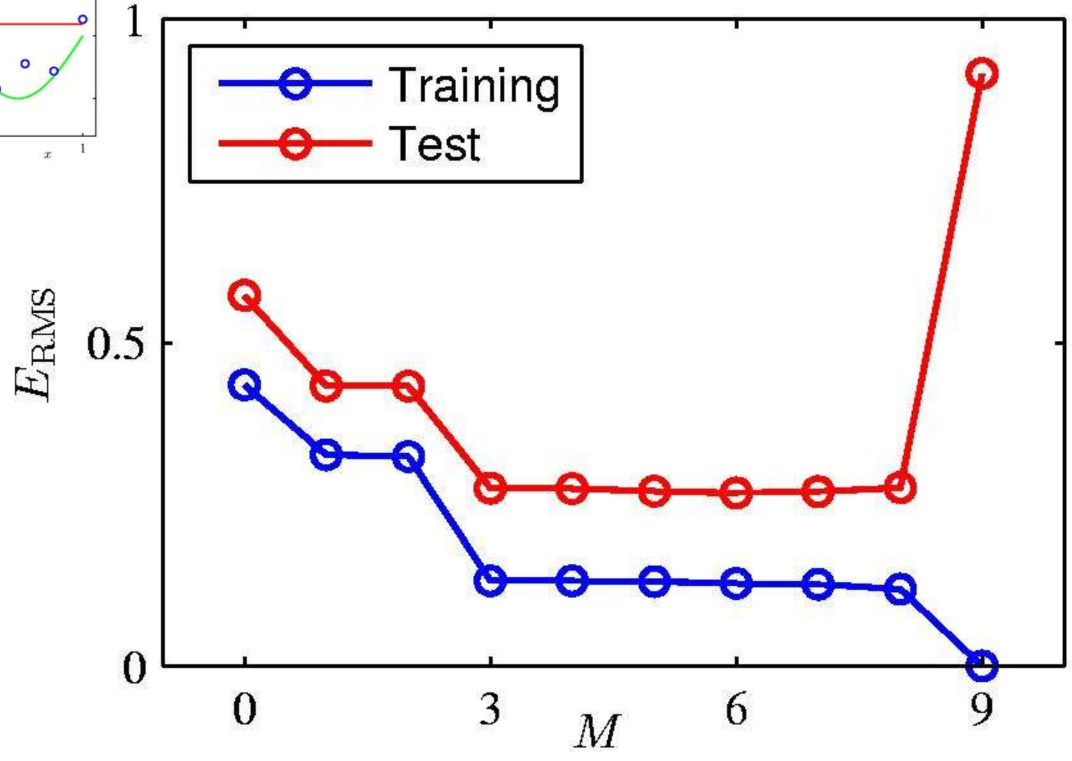


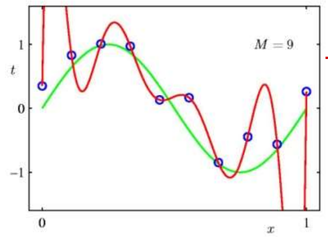
Polynomial Coefficients

	$M = 0$	$M = 1$	$M = 3$	$M = 9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

Fighting Over-fitting: train and test sets







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

Better approach: N-fold cross-validation

N-fold cross-validation:

1. split your data into N parts (equal size);
2. develop N models on all combinations of (N-1) parts;
3. test each model on the remaining parts (test sets);
4. average the errors over these N test sets;
5. the average error is a realistic estimator
   of the error made by your model on the fresh data

In practice:

- N=5 (5-fold cross-validation) or N=10 (10-fold cross-validation)
- errors also measured on the training sets/folds
- standard deviation of errors says something about "model stability"

Importance of cross-validation

Cross-validation is a common method for:

- estimating performance of the model on fresh (future, new) data
- finding optimal model parameters (e.g., degree of the polynomial, depth of the decision tree, architecture of a neural network, etc.)
- distribution (e.g., stdev) of errors on the test folds says something about "model stability"

Cross validation can be easily implemented on parallel computers

Regularization

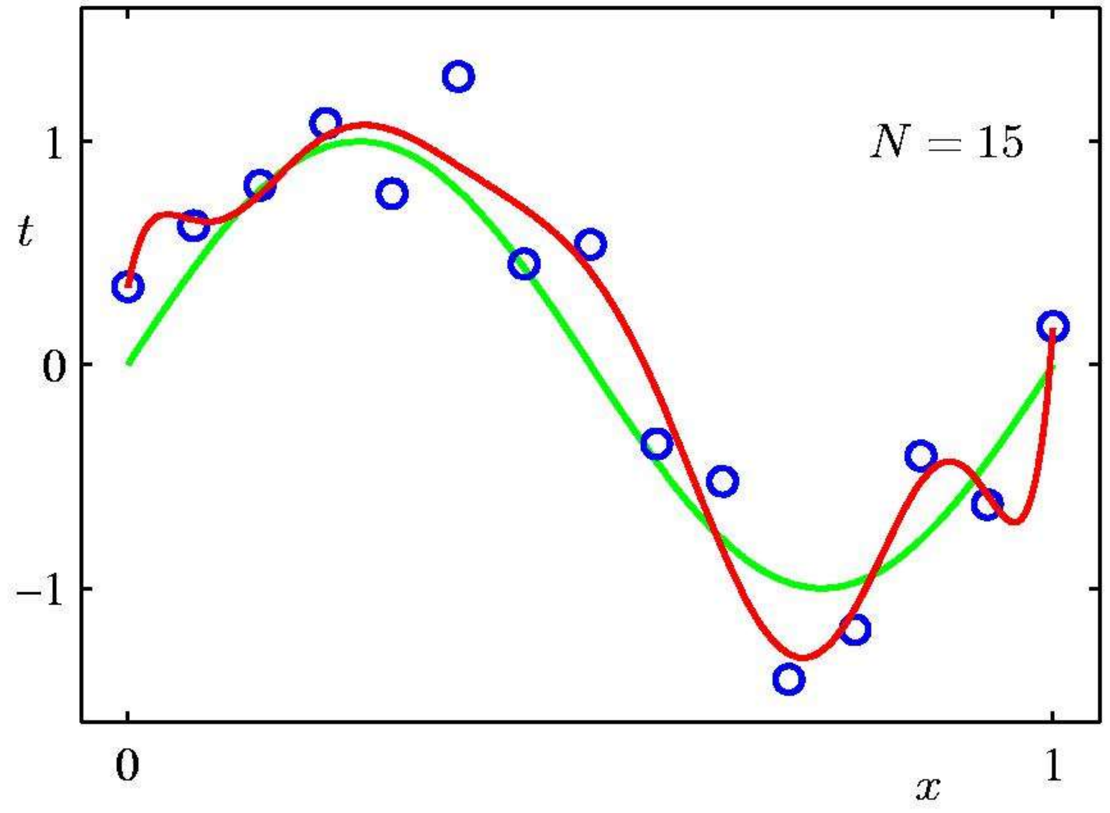
- Usually we overfit the data when the number of model parameters is relatively big, when compared to the size of the training set

- Regularization is a method reducing overfitting by imposing some constraints on the number or the magnitude of model parameters

- Usually, regularization is controlled by a parameter "λ"

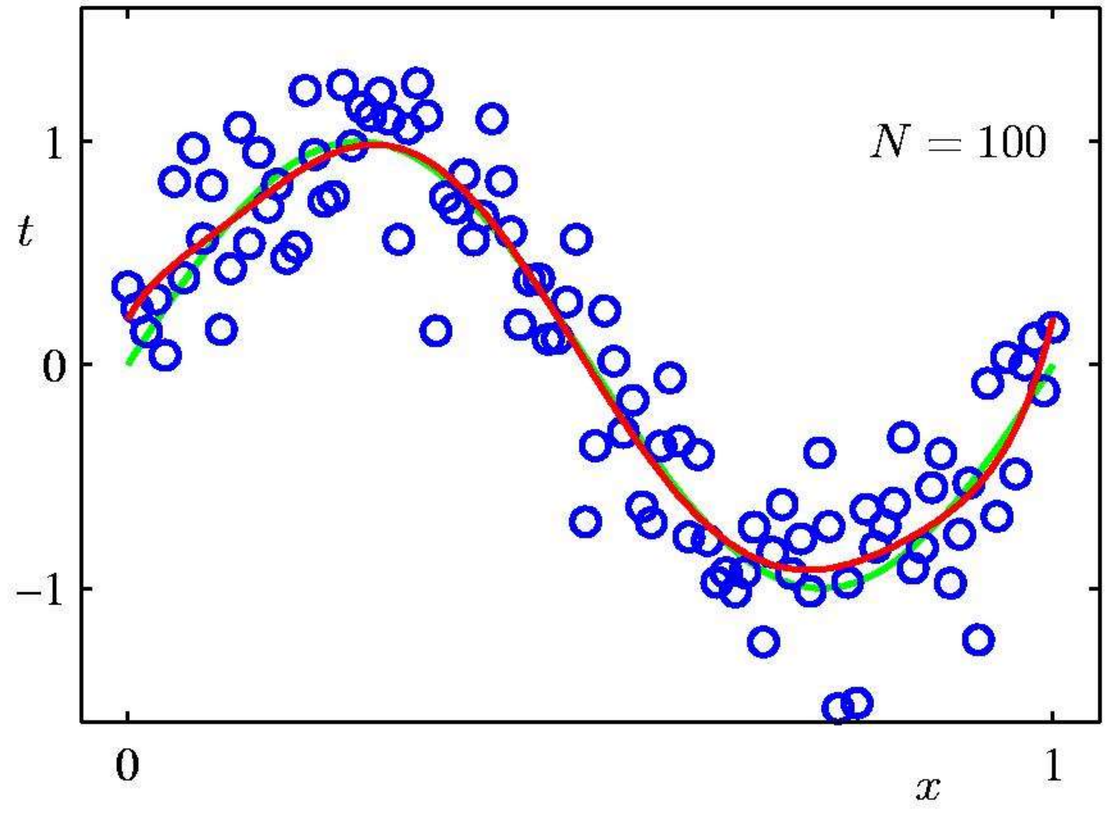
What about bigger a training set?

9th Order Polynomial for 15 data points:



Data Set Size: N=100

9th Order Polynomial



Regularization

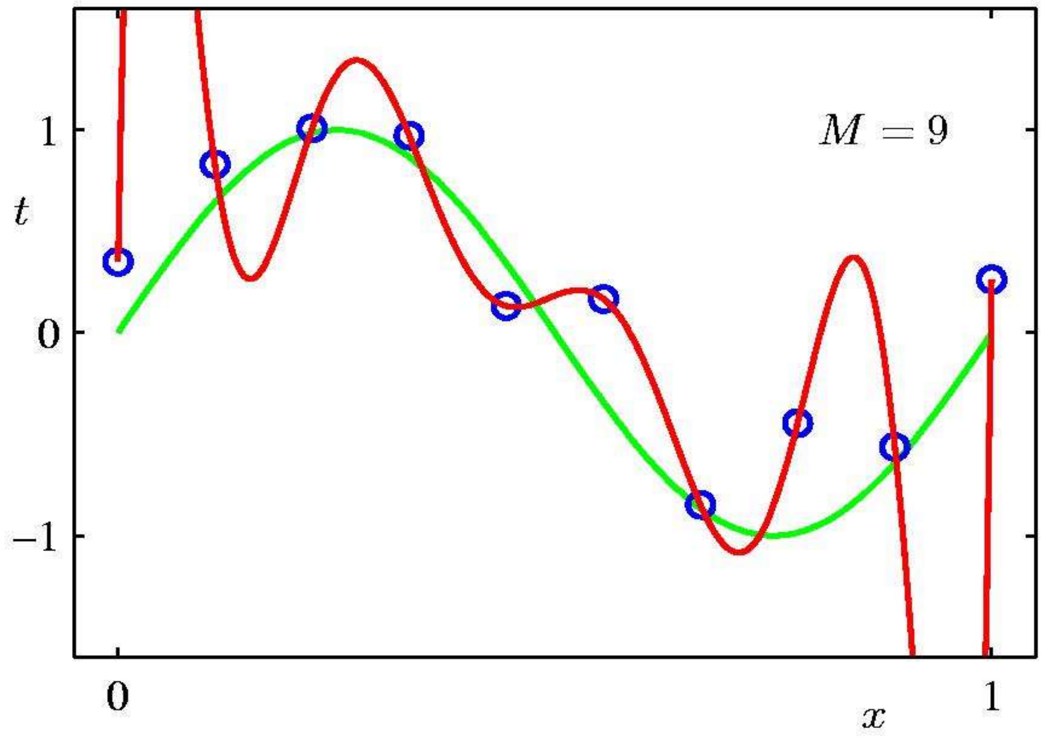
- **Penalize large coefficient values:**

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

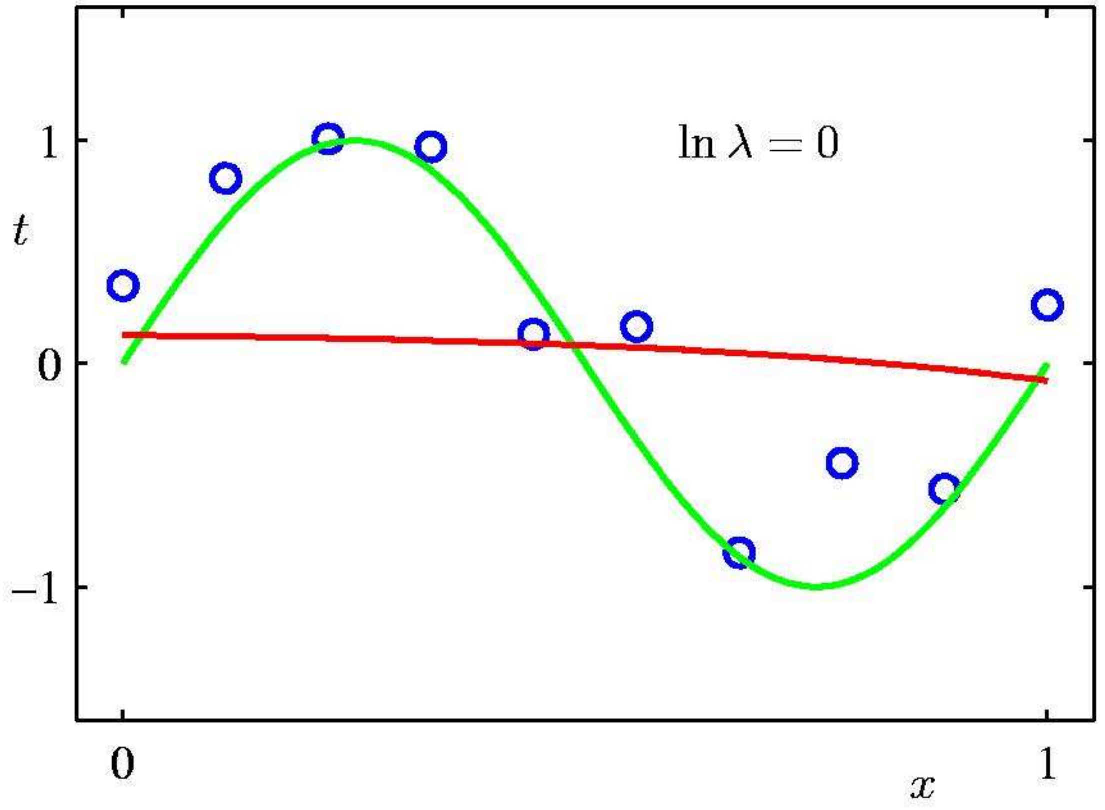
*"a simple function of **w**"!*

- **Minimize training error while keeping the weights small. This is also known as:**
  - shrinkage,
  - ridge regression,
  - weight decay (neural networks)

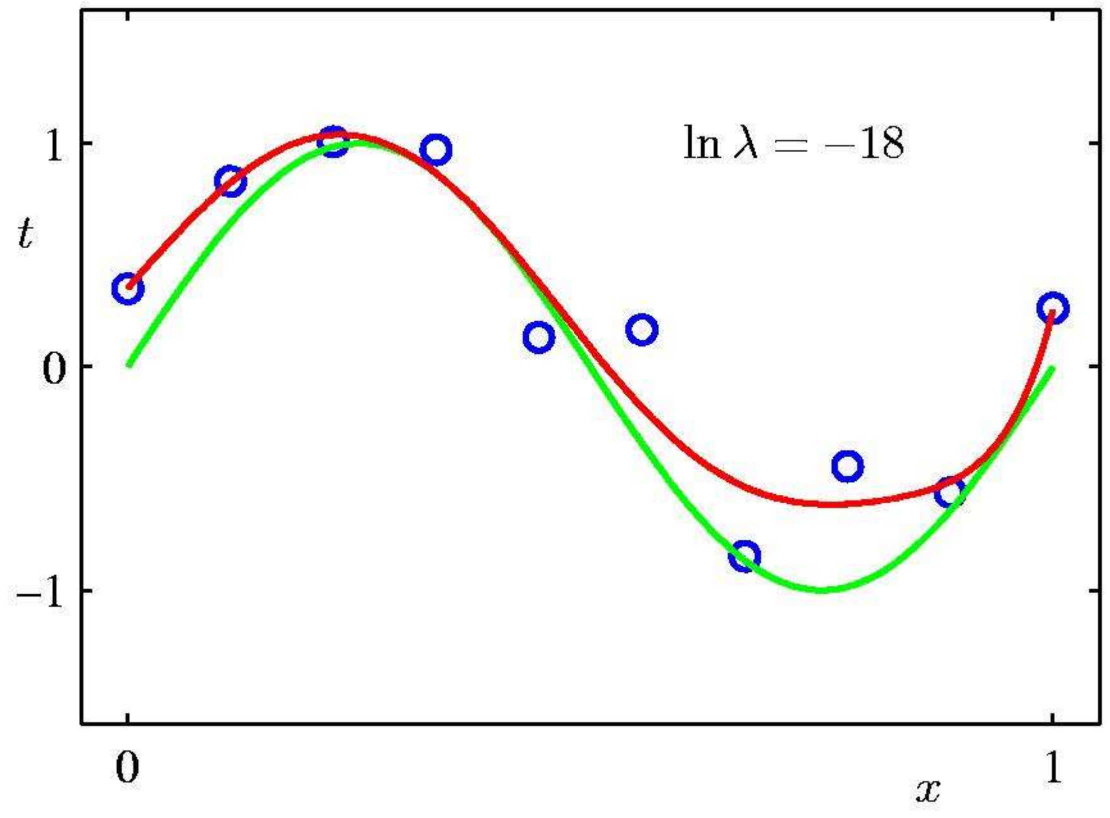
Regularization (d=9; λ=0)



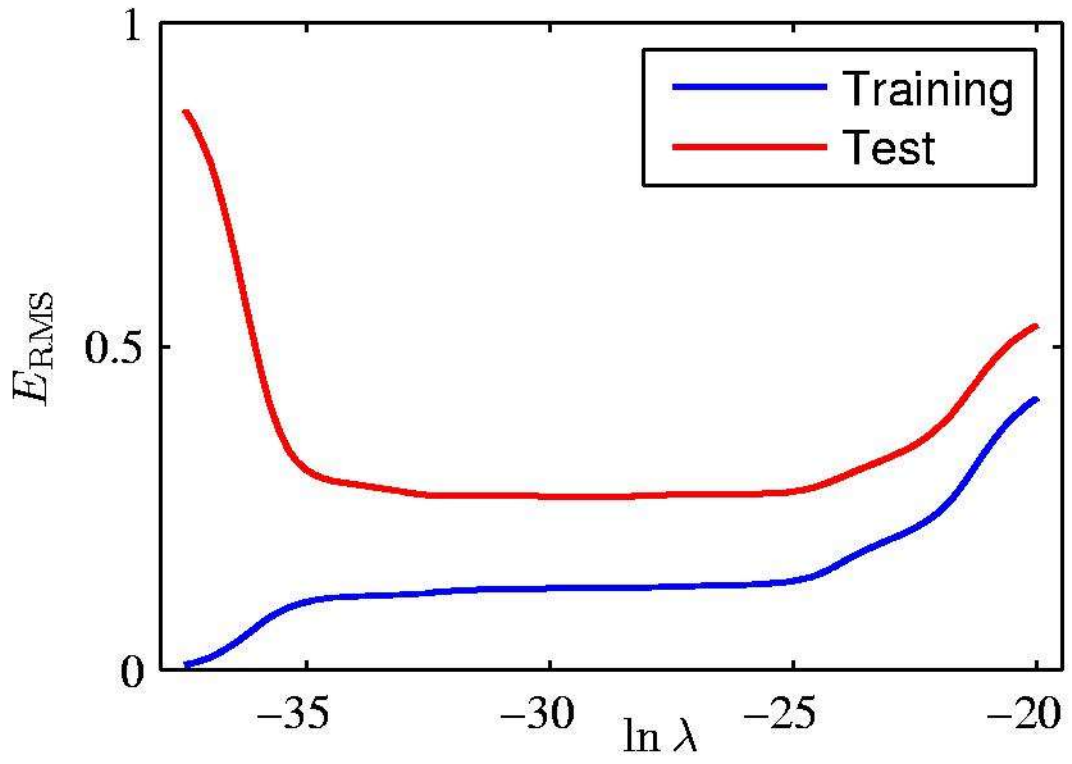
Regularization (d=9; λ=1)



Regularization (d=9; λ=1.5230e-08)



Regularization: error vs. lambda



Polynomial Coefficients

	$\ln \lambda = -\infty$	$\ln \lambda = -18$	$\ln \lambda = 0$
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18	-45.95	-0.00
$w_8^\star$	-557682.99	-91.53	0.00
$w_9^\star$	125201.43	72.68	0.01

Choice of training parameters

- The "optimal" values of η or $\boldsymbol{\lambda}$ are usually determined by a trial-and-error.

- A good heuristic (Andrew Ng) is to start "somewhere" (e.g., at 0.01) and experiment with increasing or decreasing values, using geometric progressions:

  0.01, 0.033, 0.1, 0.33, 1.0, ...
  0.01, 0.0033, 0.001, 0.00033, ...

  changing one parameter (η or $\boldsymbol{\lambda}$) at a time