CS 2750 Machine Learning

Lecture 2

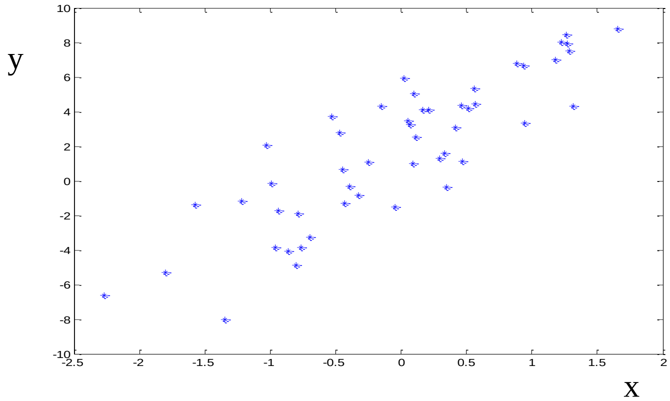
Designing a learning system

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

people.cs.pitt.edu/~milos/courses/cs2750-Spring2020/

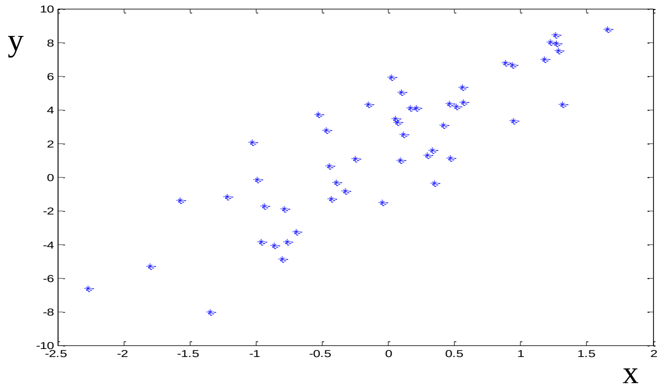
Learning: first look

- Assume we see examples of pairs ($\mathbf{x}$, y) in D and we want to learn the mapping $f : X \rightarrow Y$ to predict y for some future **x**
- We get the data D - what should we do?



Learning: first look

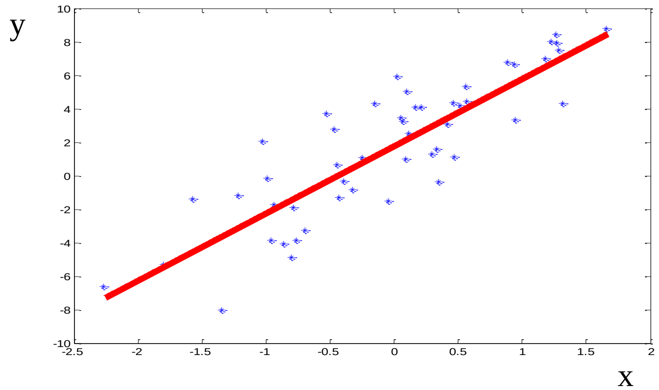
- **Problem:** many possible functions $f : X \rightarrow Y$ exists for representing the mapping between **x** and y
- Which one to choose? Many examples still unseen!



Learning: first look

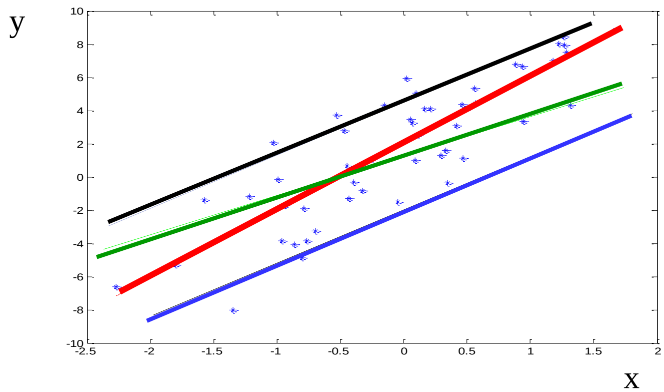
- **Solution:** make an assumption about the model, say,

$$f(x) = ax + b$$



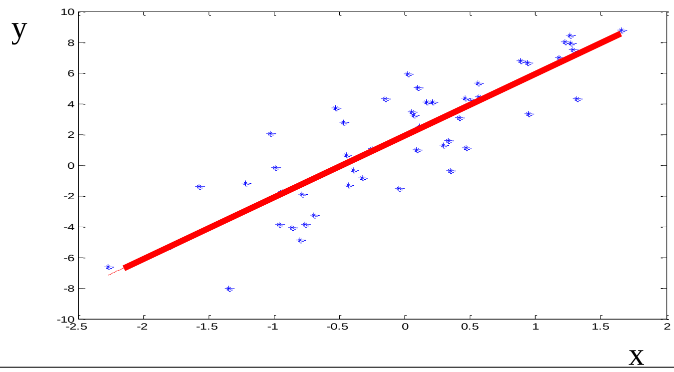
Learning: first look

- Choosing a parametric model or a set of models is not enough
  Still too many functions $f(x) = ax + b$
  - One for every pair of parameters *a*, *b*



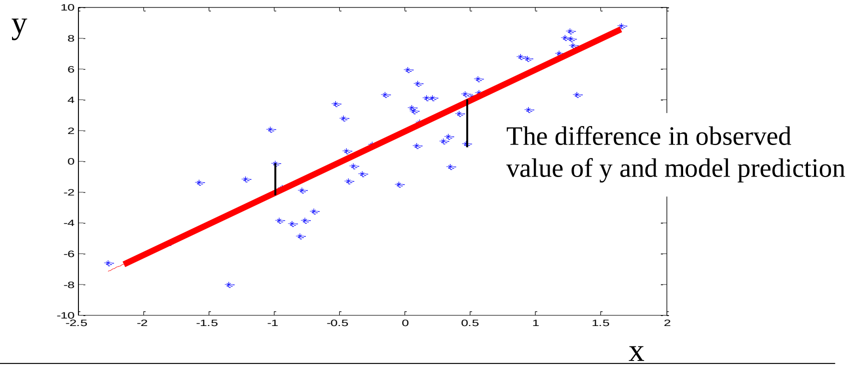
Learning: first look

- We want the **best set** of model parameters
  - reduce the misfit between the model **M** and observed data ***D***
  - Or, (in other words) explain the data the best
- **How to measure the misfit?**



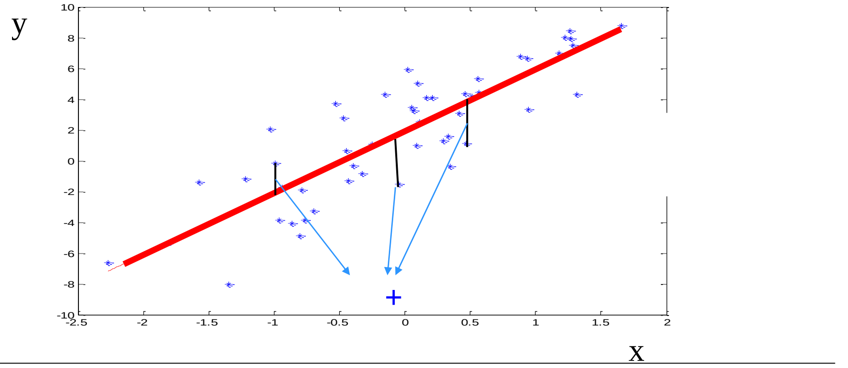
Learning: first look

- We want the **best set** of model parameters
  - reduce the misfit between the model **M** and observed data ***D***
  - Or, (in other words) explain the data the best
- **How to measure the misfit?**



Learning: first look

- We want the **best set** of model parameters
  - reduce the misfit between the model **M** and observed data ***D***
  - Or, (in other words) explain the data the best
- **How to measure the misfit?**



Learning: first look

- We want the **best set** of model parameters
  - reduce the misfit between the model **M** and observed data ***D***
  - Or, (in other words) explain the data the best
- **How to measure the misfit?**

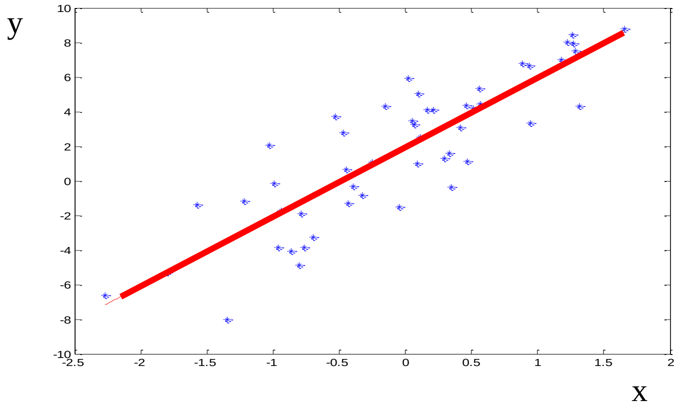
Objective function:

- **Error function: Measures the misfit between *D* and M**
- **Examples of error functions:**
  - Average Square Error $\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$
  - Average Absolute Error $\frac{1}{n}\sum_{i=1}^{n}|y_i - f(x_i)|$

Learning: first look

- **Linear regression problem**
  - Minimizes the squared error function for the linear model

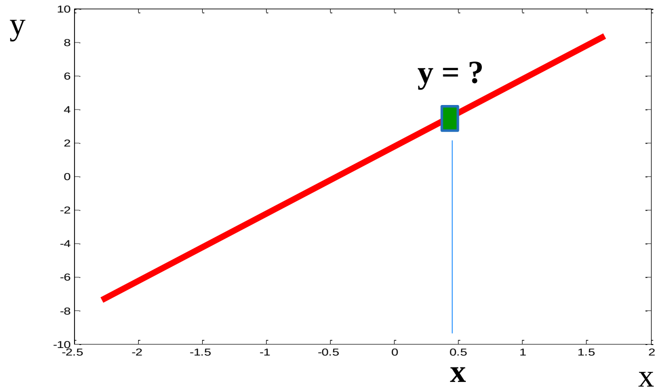
$$\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$$



Learning: first look

- **Application:** A new example **x** with unknown value y is checked against the model, and y is calculated

$$y = f(x) = ax + b$$

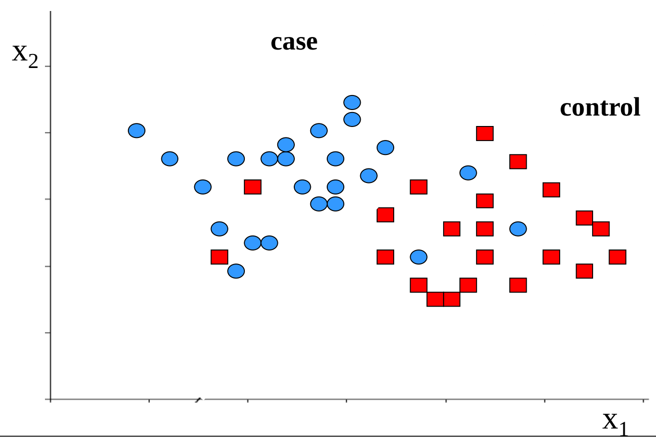


Supervised learning: Classification

- **Data** D: pairs (**x** , *y*) where y is a class label:

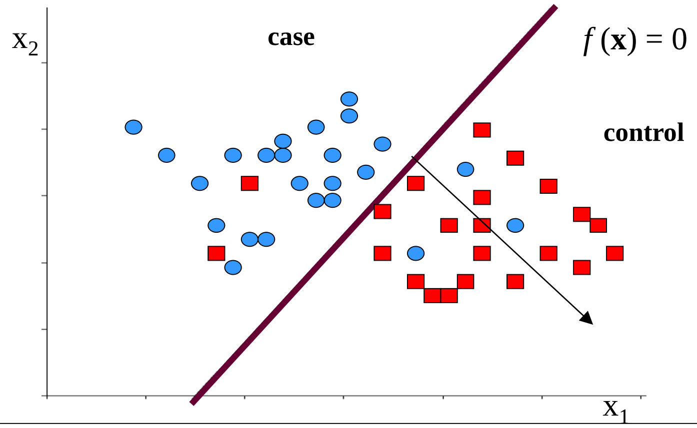
  y examples: patient will be readmitted or no,

  has disease (case) or no (control)



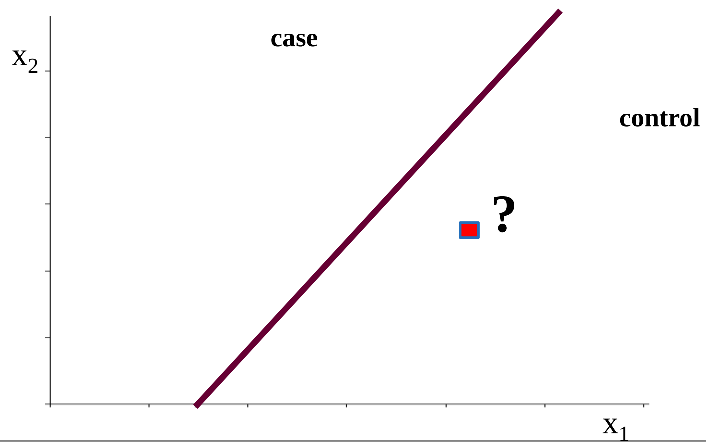
Supervised learning: Classification

- Find a model f: X → R, say $f(x) = ax_1 + bx_2 + c$ that defines a decision boundary $f(\mathbf{x}) = 0$ that separates well the two classes
  - **Note that some examples are not correctly classified**



Supervised learning: Classification

- A new example x with unknown class label is checked against the model, the class label is assigned



Learning: first look

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

2. Model selection:

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

E.g. $y = ax + b$

3. Choose the objective function

– **Squared error** $\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$

4. Learning:

- **Find the set of parameters optimizing the error function**

– The model and parameters with the smallest error

5. Application

– **Apply the learned model to new data**

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

Learning: first look

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

2. Model

– **Selec**

E.g.

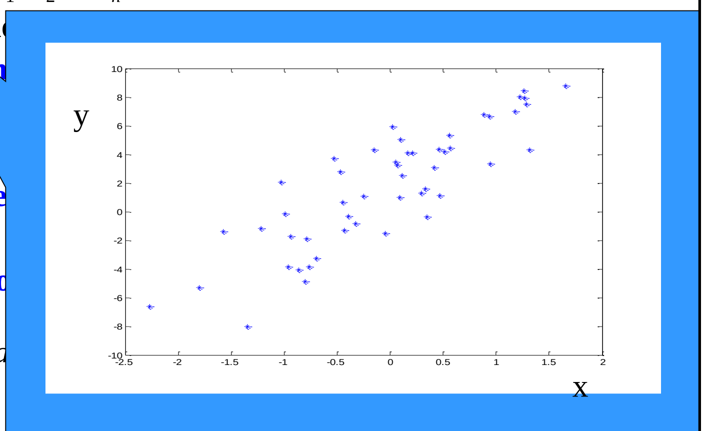
3. Choose the

– **Squared e**

4. Learning:

- **Find the set o**
function

(a



5. Application

– **Apply the learned model to new data** $f(x) = a\ x + b$

– E.g. predict ys for the new input x

Learning: first look

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

2. Model selection:

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

  E.g. $y = ax + b$

3. Choose the

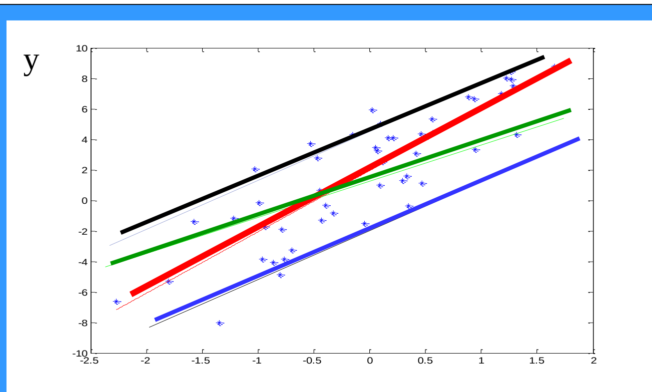
- **Squar**

4. Learning:

- **Find the s**
  function

5. Applicatio

- **Apply t**
- E.g. pre



Learning: first look

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

2. Model selection:

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

  E.g. $y = ax + b$

3. Choose the objective (error) function

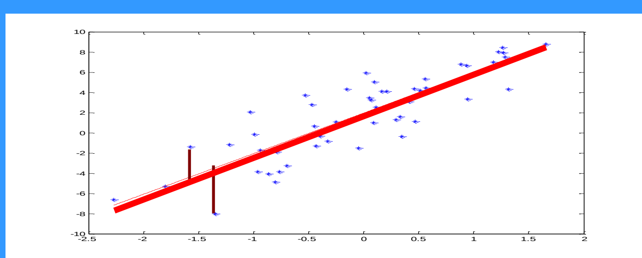
- **Squared error** $Error(D,a,b) = \frac{1}{n}\sum_{i=1}^{n}(y_i - ax_i - b))^2$

4. Learning:

- **Find the se**
  function

5. Applicatio

- **Apply t**
- E.g. pre



Learning: first look

1. Data: $D =$

2. Model sele

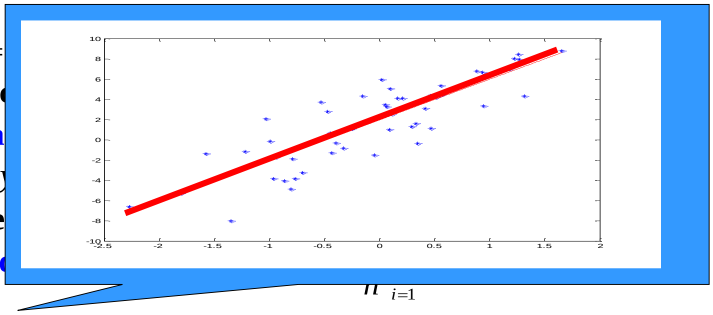
– **Select a**

E.g. y

3. Choose the

– **Square**

4. Learning:



$n \quad i=1$

- **Find the set of parameters (*a*,*b*) optimizing the error function**

$$(a^*, b^*) = \arg\max_{(a,b)} Error(D, a, b)$$

5. Application

– **Apply the learned model to new data** $f(x) = a^* x + b^*$

– E.g. predict ys for the new input x

Learning: first look

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

2. Model selection:

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

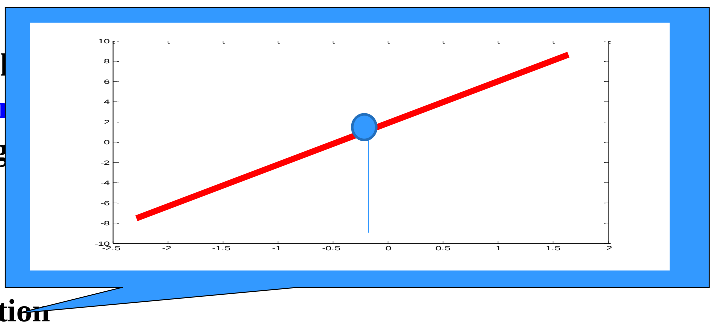
E.g.

3. Choose t

– **Squa**

4. Learning

- **Find the function**



5. Application

– **Apply the learned model to new data** $f(x) = a^* x + b^*$

– E.g. predict ys for the new input x

Learning: first look

1. Data: $D = \{d_1, d_2, .., d_n\}$
2. Model selection:
- **Select a model** or a set of models (with parameters)
  E.g. $y = ax + b$

3. Choose the objective (error) function
- **Squared error** $Error(D,a,b) = \frac{1}{n}\sum_{i=1}^{n}(y_i - ax_i - b))^2$

4. Learning:
- **Find the set of parameters (*a*,*b*) optimizing the error function** $(a^*, b^*) = \arg\max_{(a,b)} Error(D,a,b)$

5. Application
- **Apply the learned model to new data** $f(x) = a^* x + b^*$

Looks straightforward, but there are problems ….

–

Learning: generalization error

We fit the model based on past examples observed in ***D***

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

Training error:

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

Problem: Ultimately we are interested in learning the mapping that performs well on the whole population of examples

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

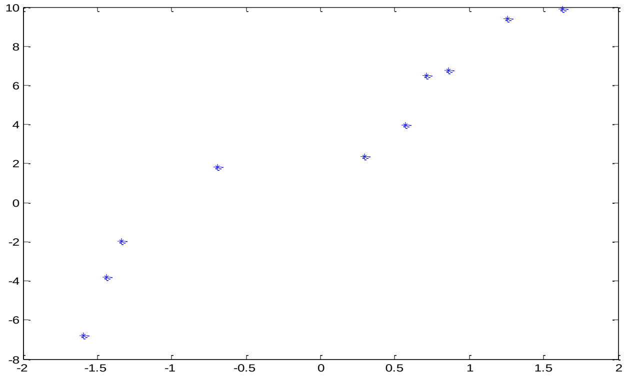
$$Error(a,b) = E_{(x,y)}[(y - f(x))^2]$$ Mean squared error

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

Does a good training error imply a good generalization error ?

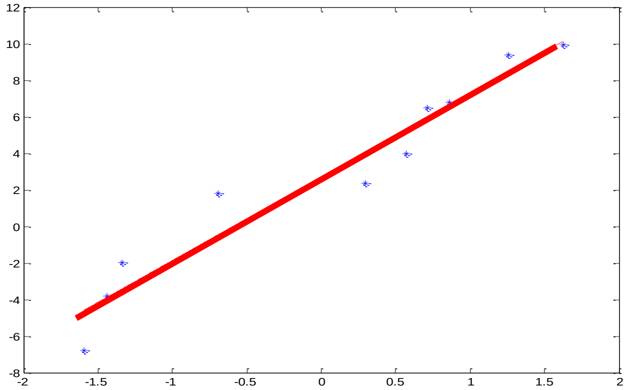
Overfitting

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



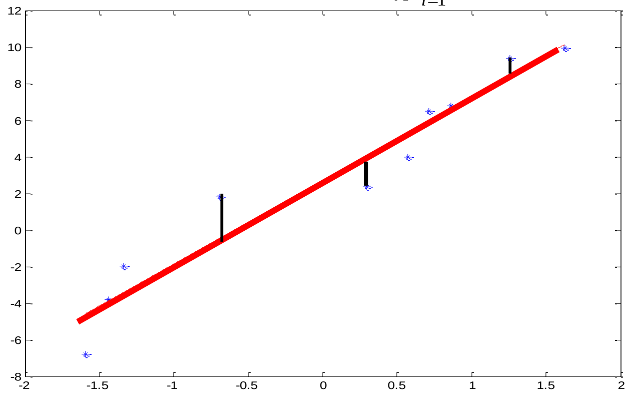
Overfitting

- Fitting a linear function with the square error
- **Error is nonzero. Why?**



Overfitting

- Fitting a linear function with the square error
- **Error is nonzero:** $Error(D, f) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$

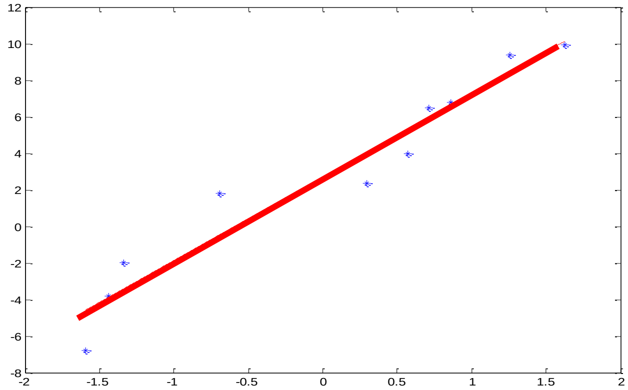


Overfitting

Assume in addition to a linear model: $y = f(x) = ax + b$

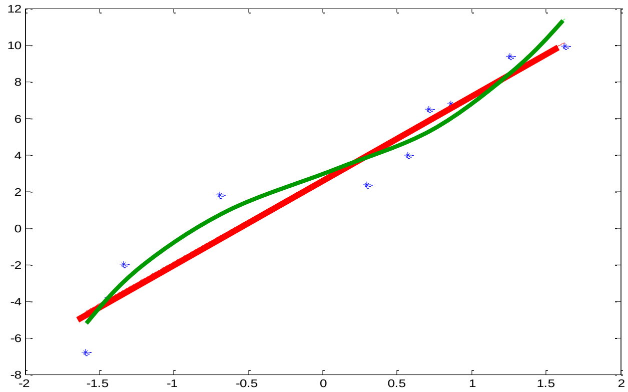
also: $y = f(x) = a_3x^3 + a_2x^2 + a_1x + b$

Which model would give us a smaller error for the least squares fit?



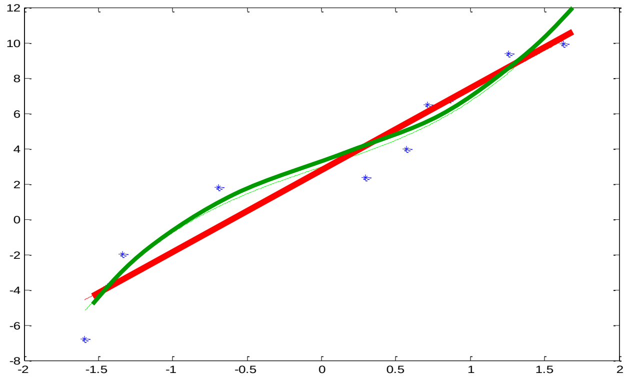
Overfitting

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



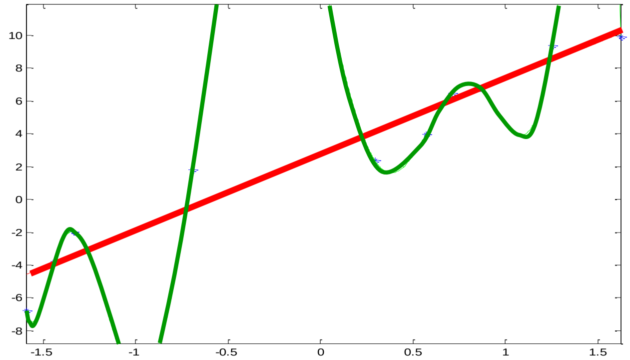
Overfitting

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



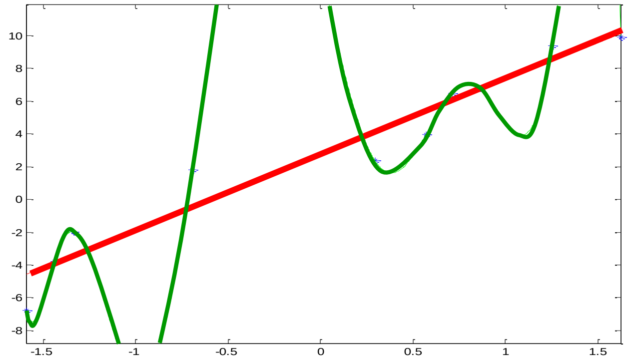
Overfitting

- 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

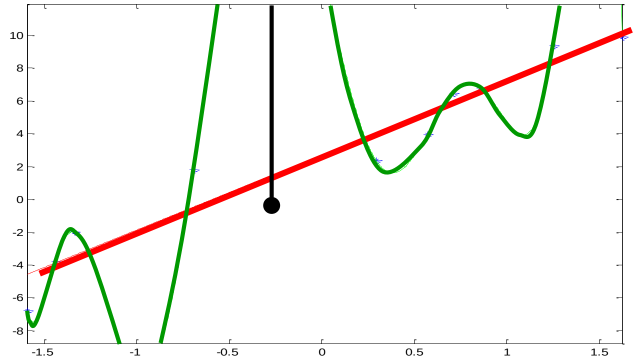
- For 10 data points, degree 9 polynomial gives a perfect fit (Lagrange interpolation). Error is zero.
- Is it always good to minimize the training error? NO !!
- **More important:** How do we perform on the unseen data?



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 the 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 assess the generalization error?

How to evaluate the learner's performance?

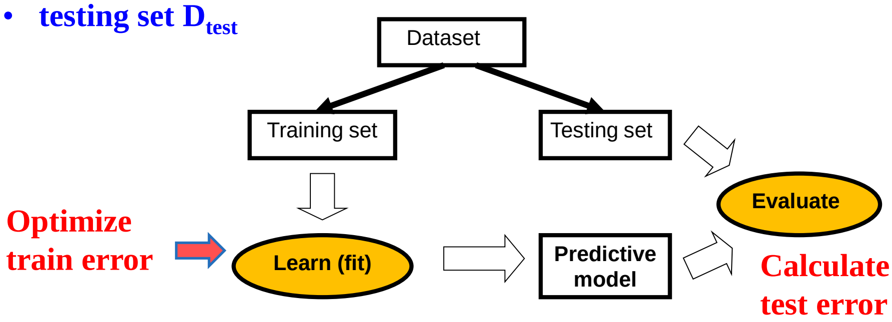
- **Generalization error** is the true error for the population of examples we would like to optimize
- **Sample mean only approximates it**
- **Two ways to assess the generalization error is:**
  - **Theoretical: Law of Large numbers**
    - statistical bounds on the difference between true and sample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

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

Evaluation of the generalization performance

Split available data D into two disjoint sets:

- **training set D_{train}**
- **testing set D_{test}**



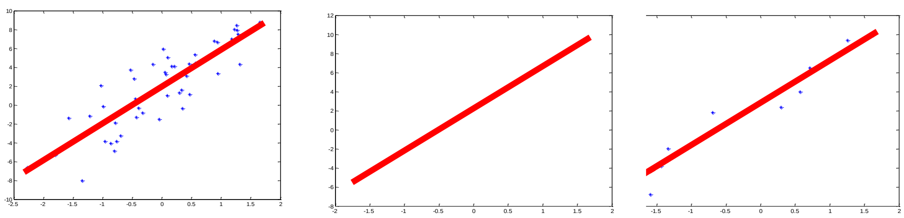
Also called: Simple holdout method

- Typically 2/3 training and 1/3 testing

Testing of models: regression

Data set

Training set | **Test set**

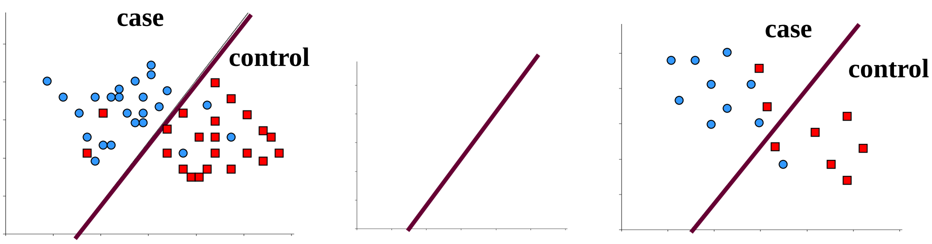


Learn on the training set → **The model** → **Evaluate on the test set**

Testing of models: classification

Data set

Training set | **Test set**



Learn on the training set → **The model** → **Evaluate on the test set**

Assessment of model performance

Assessment of the generalization performance of the model:

Basic rule:

- **Never ever touch the test data during the learning/model building process**
- **Test data should be used for the final evaluation only**

Evaluation measures

Easiest way to evaluate the model:

- **Error function used in the optimization is adopted also in the evaluation**
- **Advantage: may help us to see model overfitting. Simply compare the error on the training and testing data.**

Evaluation of the models often considers:

- **Other aspects or statistics of the model and its performance**
- Moreover the Error function used for the optimization may be a convenient approximation of the quality measure we would really like to optimize

Evaluation measures

Classification:

		Actual: Case	Actual: Control
Prediction	Case	**TP** 0.3	**FP** 0.1
	Control	**FN** 0.2	**TN** 0.4

Misclassification error:

$$E = FP + FN$$

Sensitivity:

$$SN = \frac{TP}{TP + FN}$$

Specificity:

$$SP = \frac{TN}{TN + FP}$$

A learning system: basic cycle

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

2. Model selection:

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

  E.g. $y = ax + b$

3. Choose the objective function

- **Squared error** $\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$

4. Learning:

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

5. Testing/validation:

- **Evaluate on the test data**

6. Application

- **Apply the learned model to new data** $f(\mathbf{x})$

A learning system: basic cycle

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

2. Model selection:

– **Select**

E.g.

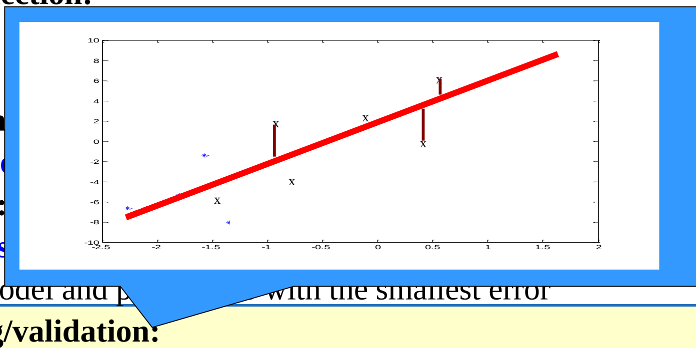
3. Choose th

– **Square**

4. Learning:

- **Find the s**

– The model and p ... with the smallest error



5. 5. Testing/validation:

– **Evaluate on the test data**

6. Application

– **Apply the learned model to new data** $f(\mathbf{x})$

A learning system: basic cycle

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

2. Model selection:

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

E.g. $y = ax + b$

3. Choose the objective function

– **Squared error** $\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$

4. Learning:

- **Find the set of parameters optimizing the error function**

– The model and parameters with the smallest error

5. Testing/validation:

– **Evaluate on the test data**

6. Application

– **Apply the learned model to new data** $f(\mathbf{x})$

Steps taken when designing an ML system

1. Data
2. Model selection
3. Choice of Error function
4. Learning/optimization
5. Evaluation
6. Application

Add some complexity

1. Data
2. Data cleaning/preprocessing
3. Feature selection/dimensionality reduction
4. Model selection
5. Choice of Error function
6. Learning/optimization
7. Evaluation
8. Application

Designing an ML solution

Data

Data cleaning/preprocessing

Feature selection/dimensionality reduction

Model selection

Choice of Error function

Learning/optimization

Evaluation

Application

Designing an ML solution

Data

Data cleaning/preprocessing

Feature selection/dimensionality reduction

Model selection

Choice of Error function

Learning/optimization

Evaluation

Application

Data source and data biases

- **Understand the data source**
- **Understand the data your models will be applied to**
- **Watch out for data biases:**
  - 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

- **Results (conclusions) derived for a biased dataset do not hold in general !!!**

Data biases

Example: Assume you want to build an ML program for predicting the stock behavior and for choosing your investment strategy

Data extraction:

- pick companies that are traded on the stock market on January 2017
- Go back 30 years and extract all the data for these companies
- Use the data to build an ML model supporting your future investments

Question:

- **Would you trust the model?**
- **Are there any biases in the data?**