

# COMP 551 - Applied Machine Learning

## Lecture 3 – Linear Regression (cont.)

---

William L. Hamilton

(with slides and content from Joelle Pineau)

\* Unless otherwise noted, all material posted for this course are copyright of the instructor, and cannot be reused or reposted without the instructor's written permission.

# Self-assessment/practice quizzes

---

- Quiz 0 – Attempt 1 is out and due Wednesday at 11:59pm.
- Quiz 0 – Attempt 2 will be out Thursday at 12am.
- You have one hour to complete it once you begin.
- This week's quizzes do not count towards your grade. They are for self-assessment and practice.
- If you are really struggling or confused by the quiz's this week, it might be worthwhile to wait and take this course in a later semester.
  - (Ideally you should be able to get 100%)

# Recall: Linear regression

---

- The linear regression problem:  $f_{\mathbf{w}}(\mathbf{x}) = w_0 + \sum_{j=1:m} w_j x_j$   
where  $m$  = the dimension of observation space, i.e. number of features.
- Goal: Find the **best** linear model (i.e., weights) given the data.
- Many different possible **evaluation** criteria!
- Most common choice is to find the  $\mathbf{w}$  that minimizes:

$$\text{Err}(\mathbf{w}) = \sum_{i=1:n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

(A note on notation: Here  $\mathbf{w}$  and  $\mathbf{x}$  are vectors of size  $m+1$ .)

# Is linear regression always easy to solve?

---

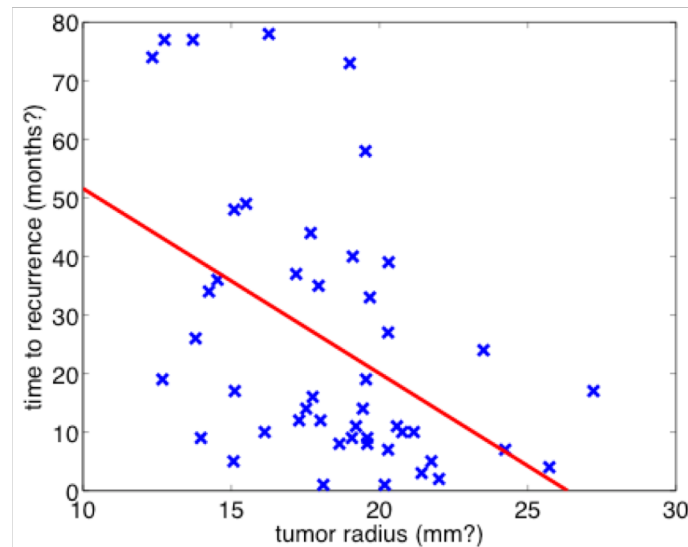
- Recall the least-square solution:

$$\hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y}$$

- Assuming for now that  $\mathbf{X}$  is reasonably small and that we can find a closed-form solution in poly-time.
  - Does this always work (well)?

# Is linear regression always easy to solve?

- Recall the least-square solution:  
 $\hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y}$
- Two main failure modes:
  - 1)  $(\mathbf{X}^T \mathbf{X})$  is singular, so the inverse is undefined.
  - 2) A simple linear function is a bad fit... (e.g., figure to the right)



# Failure mode 1: Avoiding singularities

---

- Recall the least-square solution:

$$\hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y}$$

- To have a unique solution, we need  $\mathbf{X}^T \mathbf{X}$  to be nonsingular.
- That means  $\mathbf{X}$  must have full column rank (i.e. no correlation between features).

# Failure mode 1: Avoiding singularities

---

- Recall the least-square solution:  
 $\hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y}$
- To have a unique solution, we need  $\mathbf{X}^T \mathbf{X}$  to be nonsingular.
- That means  $\mathbf{X}$  must have full column rank (i.e. no correlation between features).

Exercise: What if  $\mathbf{X}$  does not have full column rank? When would this happen?  
Design an example. Try to solve it.

# Dealing with difficult cases of $(X^T X)^{-1}$

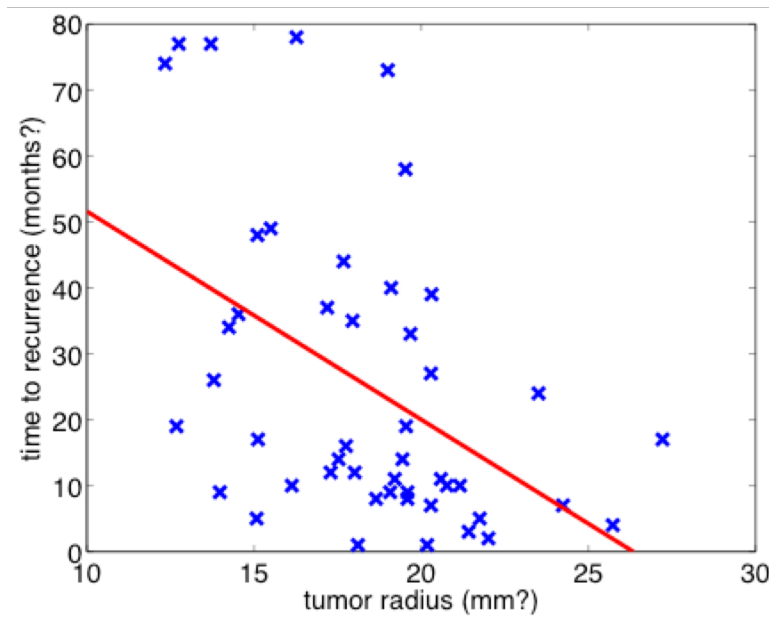
---

- **Case #1:** The weights are not uniquely defined.
  - **Example:** One feature is a linear function of the other, e.g.,  $X_m = 2X_{m-1} + 2$
  - **Solution:** Re-code or drop some redundant columns of  $X$ .
- **Case #2:** The number of features/weights ( $m$ ) exceeds the number of training examples ( $n$ ).
  - **Solution:** Reduce the number of features using various techniques (to be studied later.)



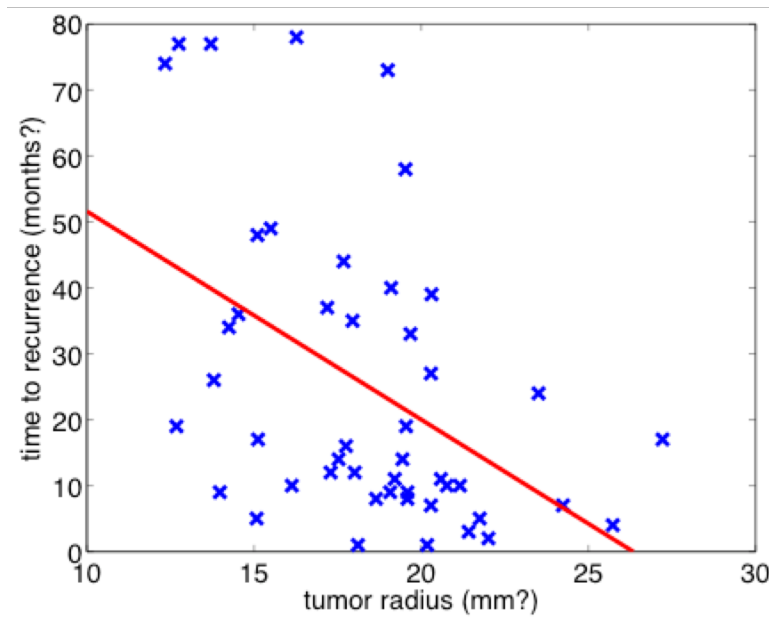
# Failure mode 2: When linear is a bad fit

- This function looks complicated, and a linear hypothesis does not seem very good.
- What should we do?



# Failure mode 2: When linear is a bad fit

- This function looks complicated, and a linear hypothesis does not seem very good.
- What should we do?
  - *Pick a better function?*
  - *Use more features?*
  - *Get more data?*



# Input variables for linear regression

---

- Original quantitative variables  $X_1, \dots, X_m$

# Input variables for linear regression

---

- Original quantitative variables  $X_1, \dots, X_m$
- Transformations of variables, e.g.  $X_{m+1} = \log(X_i)$

# Input variables for linear regression

---

- Original quantitative variables  $X_1, \dots, X_m$
- Transformations of variables, e.g.  $X_{m+1} = \log(X_i)$
- Basis expansions, e.g.  $X_{m+1} = X_i^2, X_{m+2} = X_i^3, \dots$

# Input variables for linear regression

---

- Original quantitative variables  $X_1, \dots, X_m$
- Transformations of variables, e.g.  $X_{m+1} = \log(X_i)$
- Basis expansions, e.g.  $X_{m+1} = X_i^2, X_{m+2} = X_i^3, \dots$
- Interaction terms, e.g.  $X_{m+1} = X_i X_j$

# Input variables for linear regression

---

- Original quantitative variables  $X_1, \dots, X_m$
- Transformations of variables, e.g.  $X_{m+1} = \log(X_i)$
- Basis expansions, e.g.  $X_{m+1} = X_i^2, X_{m+2} = X_i^3, \dots$
- Interaction terms, e.g.  $X_{m+1} = X_i X_j$
- Numeric coding of qualitative variables, e.g.  $X_{m+1} = 1$  if  $X_i$  is true and  $0$  otherwise.

# Input variables for linear regression

---

- Original quantitative variables  $X_1, \dots, X_m$
- Transformations of variables, e.g.  $X_{m+1} = \log(X_i)$
- Basis expansions, e.g.  $X_{m+1} = X_i^2, X_{m+2} = X_i^3, \dots$
- Interaction terms, e.g.  $X_{m+1} = X_i X_j$
- Numeric coding of qualitative variables, e.g.  $X_{m+1} = 1$  if  $X_i$  is true and 0 otherwise.

In all cases, we can add  $X_{m+1}, \dots, X_{m+k}$  to the list of original variables and perform the linear regression.



# Variable coding is not always straightforward!

---

- Suppose we have a categorical variable with four possible values:  $X_{\text{cat}} = \{\text{blue, red, green, yellow}\}$ 
  - How do we encode this?

# Variable coding is not always straightforward!

---

- Suppose we have a categorical variable with four possible values:  $X_{\text{cat}} = \{\text{blue, red, green, yellow}\}$ 
  - How do we encode this?
  - Solution 1: Four separate binary variables (e.g.,  $X_{\text{blue}} = 1$  if  $X_{\text{cat}} = \text{blue}$  and  $X_{\text{blue}} = 0$  otherwise).

# Variable coding is not always straightforward!

---

- Suppose we have a categorical variable with four possible values:  $X_{\text{cat}} = \{\text{blue, red, green, yellow}\}$ 
  - How do we encode this?
  - Solution 1: Four separate binary variables (e.g.,  $X_{\text{blue}} = 1$  if  $X_{\text{cat}} = \text{blue}$  and  $X_{\text{blue}} = 0$  otherwise).
    - Careful... this can lead to singularities, since these four variables will be perfectly correlated. If we know  $X_{\text{cat}}$  is not  $\{\text{red, green, yellow}\}$  then it must be **blue**!

# Variable coding is not always straightforward!

---

- Suppose we have a categorical variable with four possible values:  $X_{\text{cat}} = \{\text{blue, red, green, yellow}\}$ 
  - How do we encode this?
  - Solution 1: Four separate binary variables (e.g.,  $X_{\text{blue}} = 1$  if  $X_{\text{cat}} = \text{blue}$  and  $X_{\text{blue}} = 0$  otherwise).
    - Careful... this can lead to singularities, since these four variables will be perfectly correlated. I.e., if we know  $X_{\text{cat}}$  is not  $\{\text{red, green, yellow}\}$  then it must be **blue**!
  - Solution 2: Three separate binary variables for  $\{\text{red, green, yellow}\}$  with the **blue** category represented by the intercept.
    - This is a better solution!

# Variable coding is not always straightforward!

---

- What about if our input is just raw text (or images)...?
- We need to get creative! E.g.,
  - word counts
  - average word length
  - number of words per sentence
  - etc...

# Variable coding is not always straightforward!

---

- What about if our input is just raw text (or images)...?
- We need to get creative! E.g.,
  - word counts
  - average word length
  - number of words per sentence
  - etc...
- This is called feature design, and it will be a recurring theme throughout the course!

# Input variables for linear regression

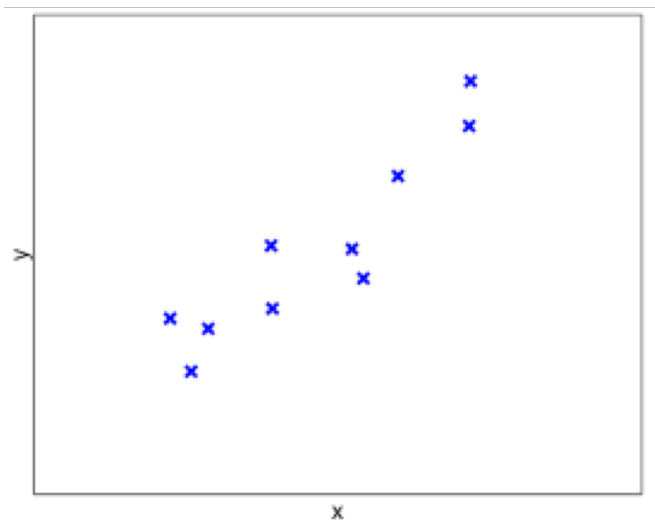
---

- Original quantitative variables  $X_1, \dots, X_m$
- Transformations of variables, e.g.  $X_{m+1} = \log(X_i)$
- Basis expansions, e.g.  $X_{m+1} = X_i^2, X_{m+2} = X_i^3, \dots$
- Interaction terms, e.g.  $X_{m+1} = X_i X_j$
- Numeric coding of qualitative variables, e.g.  $X_{m+1} = 1$  if  $X_i$  is true and 0 otherwise.

In all cases, we can add  $X_{m+1}, \dots, X_{m+k}$  to the list of original variables and perform the linear regression.

# Ex: Linear regression with polynomial terms

$$f_w(x) = w_0 + w_1 x + w_2 x^2$$



$$X = \begin{bmatrix} x^2 & x & 1 \\ 0.75 & 0.86 & 1 \\ 0.01 & 0.09 & 1 \\ 0.73 & -0.85 & 1 \\ 0.76 & 0.87 & 1 \\ 0.19 & -0.44 & 1 \\ 0.18 & -0.43 & 1 \\ 1.22 & -1.10 & 1 \\ 0.16 & 0.40 & 1 \\ 0.93 & -0.96 & 1 \\ 0.03 & 0.17 & 1 \end{bmatrix} \quad Y = \begin{bmatrix} 2.49 \\ 0.83 \\ -0.25 \\ 3.10 \\ 0.87 \\ 0.02 \\ -0.12 \\ 1.81 \\ -0.83 \\ 0.43 \end{bmatrix}$$



# Data matrices

$$\begin{aligned} X^T X &= \begin{bmatrix} 0.75 & 0.01 & 0.73 & 0.76 & 0.19 & 0.18 & 1.22 & 0.16 & 0.93 & 0.03 \\ 0.86 & 0.09 & -0.85 & 0.87 & -0.44 & -0.43 & -1.10 & 0.40 & -0.96 & 0.17 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 0.75 & 0.86 & 1 \\ 0.01 & 0.09 & 1 \\ 0.73 & -0.85 & 1 \\ 0.76 & 0.87 & 1 \\ 0.19 & -0.44 & 1 \\ 0.18 & -0.43 & 1 \\ 1.22 & -1.10 & 1 \\ 0.16 & 0.40 & 1 \\ 0.93 & -0.96 & 1 \\ 0.03 & 0.17 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 4.11 & -1.64 & 4.95 \\ -1.64 & 4.95 & -1.39 \\ 4.95 & -1.39 & 10 \end{bmatrix} \end{aligned}$$

# Data matrices

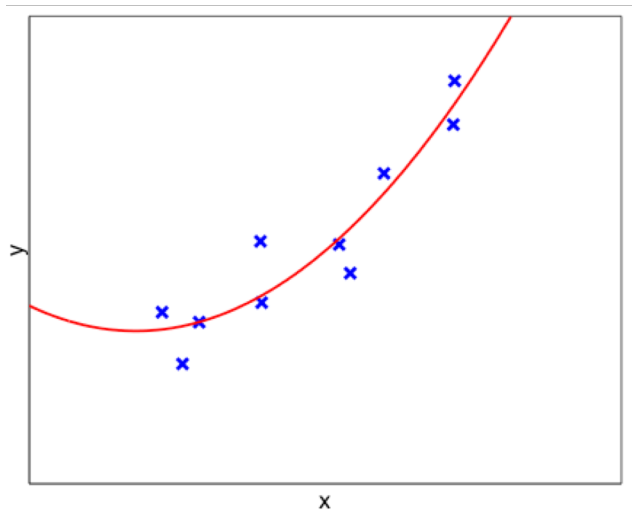
---

$$X^T Y =$$
$$\begin{bmatrix} 0.75 & 0.01 & 0.73 & 0.76 & 0.19 & 0.18 & 1.22 & 0.16 & 0.93 & 0.03 \\ 0.86 & 0.09 & -0.85 & 0.87 & -0.44 & -0.43 & -1.10 & 0.40 & -0.96 & 0.17 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 2.49 \\ 0.83 \\ -0.25 \\ 3.10 \\ 0.87 \\ 0.02 \\ -0.12 \\ 1.81 \\ -0.83 \\ 0.43 \end{bmatrix}$$
$$= \begin{bmatrix} 3.60 \\ 6.49 \\ 8.34 \end{bmatrix}$$

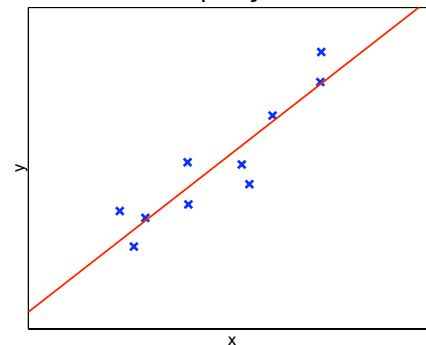
# Solving the problem

$$\mathbf{w} = (X^T X)^{-1} X^T Y = \begin{bmatrix} 4.11 & -1.64 & 4.95 \\ -1.64 & 4.95 & -1.39 \\ 4.95 & -1.39 & 10 \end{bmatrix}^{-1} \begin{bmatrix} 3.60 \\ 6.49 \\ 8.34 \end{bmatrix} = \begin{bmatrix} 0.68 \\ 1.74 \\ 0.73 \end{bmatrix}$$

So the best order-2 polynomial is  $y = 0.68x^2 + 1.74x + 0.73$ .

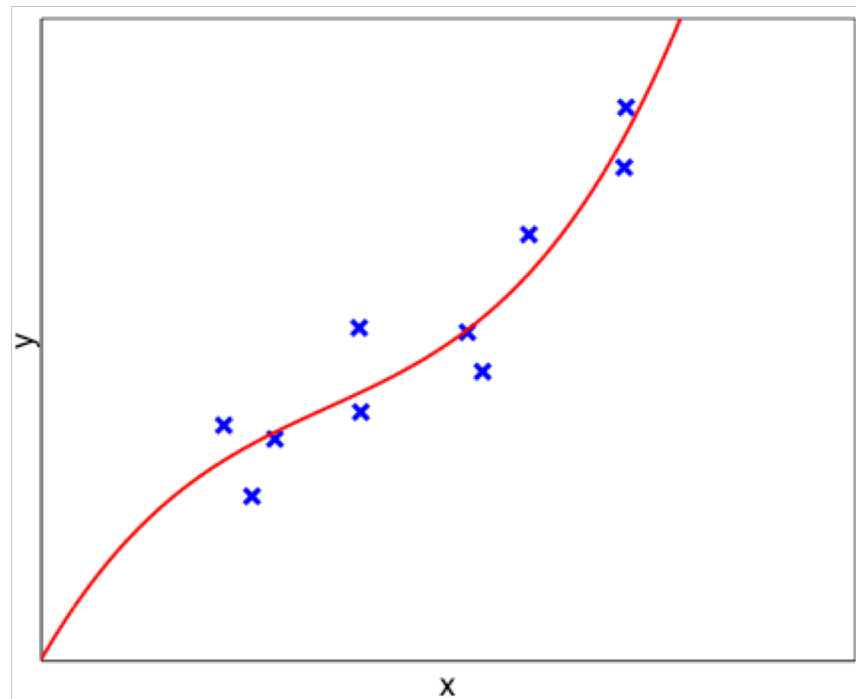


Compared to  $y = 1.6x + 1.05$  for the order-1 polynomial.



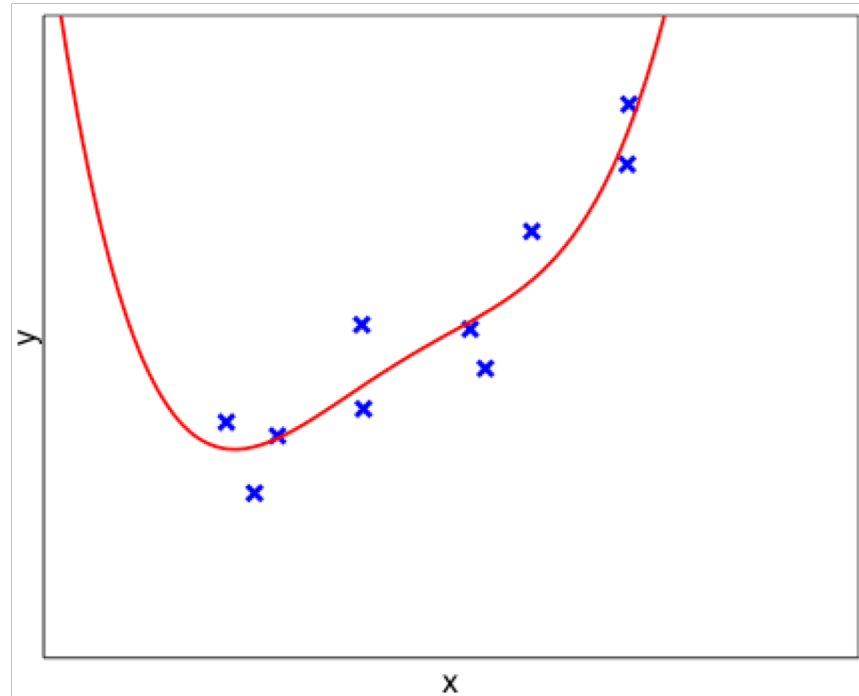
# Order-3 fit: Is this better?

---



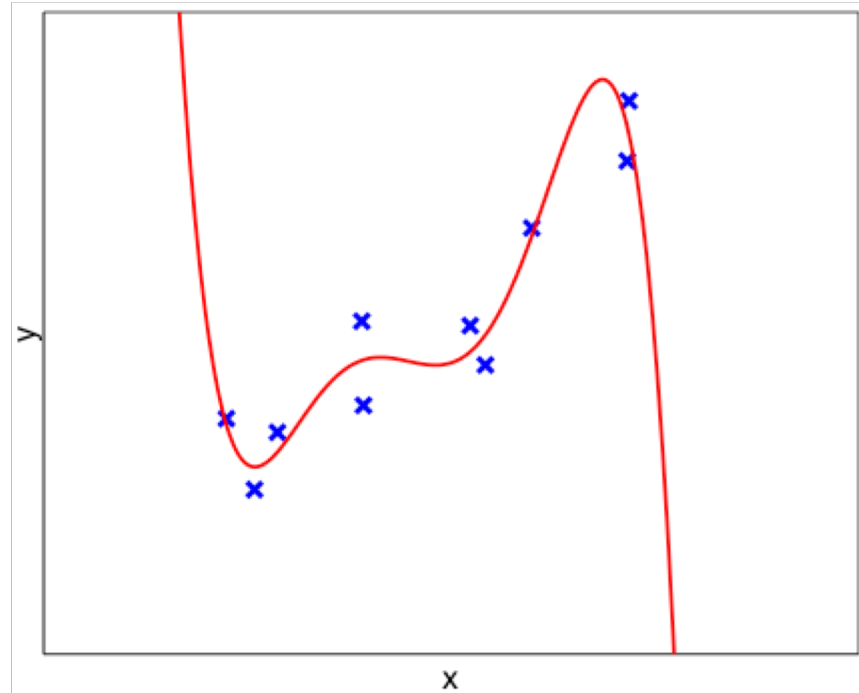
# Order-4 fit

---



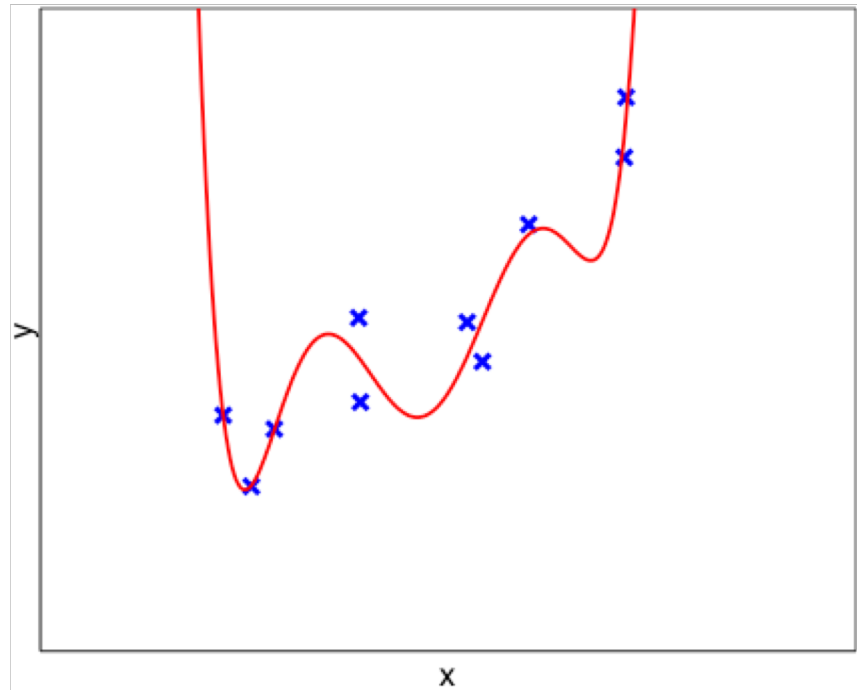
# Order-5 fit

---



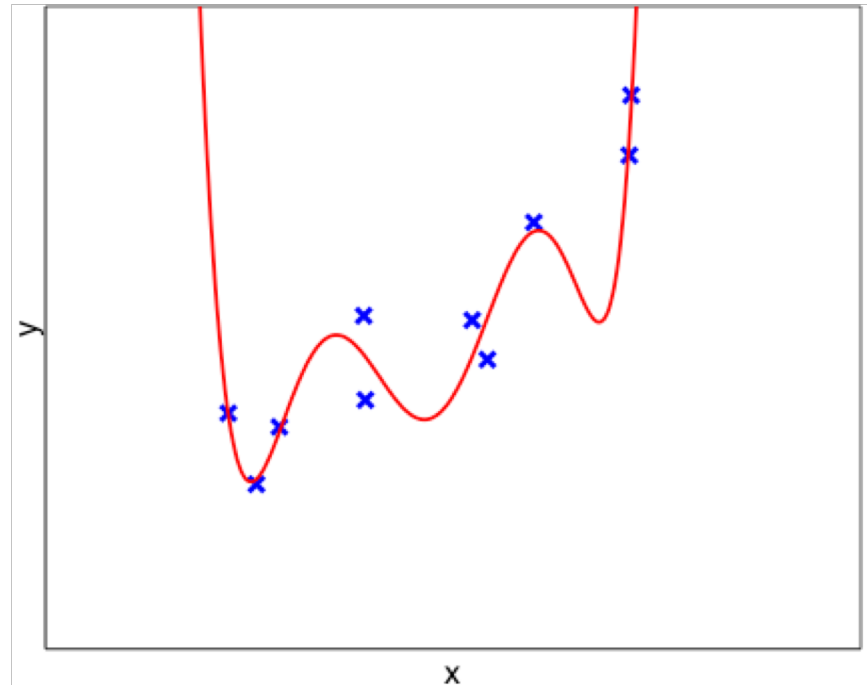
# Order-6 fit

---



# Order-7 fit

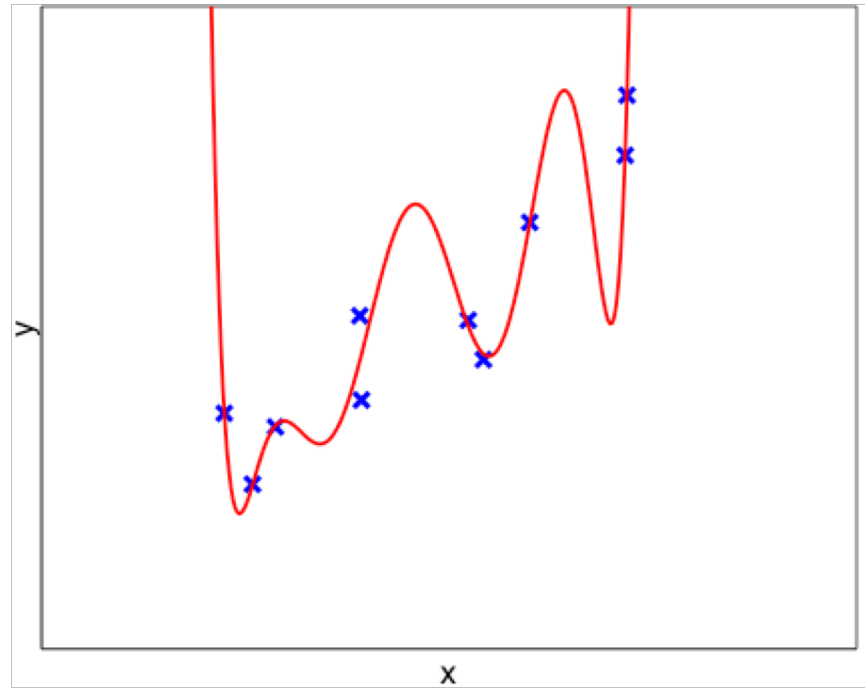
---



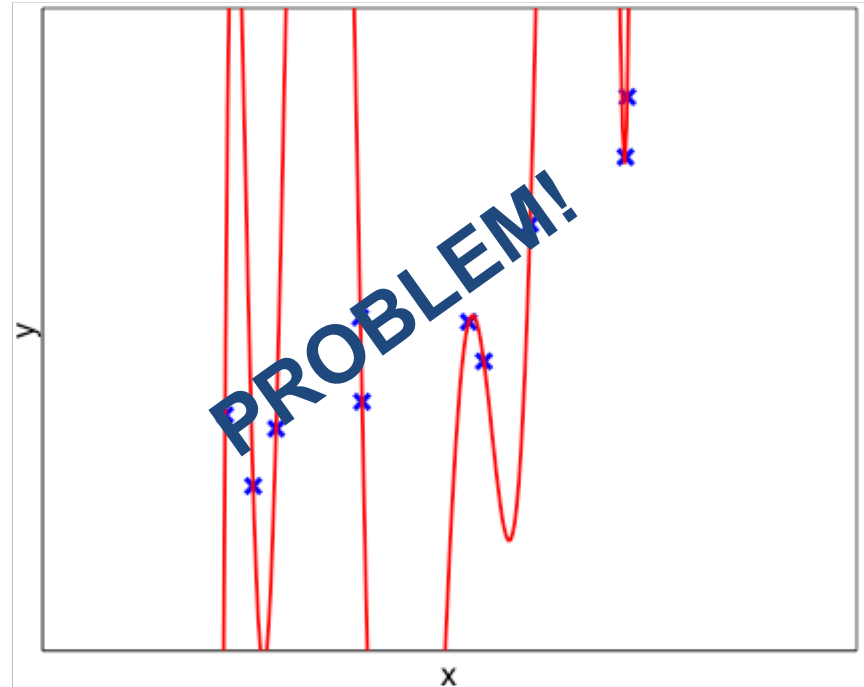


# Order-8 fit

---



# Order-9 fit



# This is overfitting!

---

- We can find a hypothesis that explains perfectly the training data, but does not generalize well to new data.
- In this example: we have a lot of parameters (weights), so the model matches the data points exactly,  
but is wild everywhere else.
- A very important problem in machine learning.

# Overfitting

---

- Every hypothesis has a **true error** measured on **all possible data items** we could ever encounter (e.g.  $f_w(\mathbf{x}_i) - y_i$  ).
- Since we don't have all possible data, in order to decide what is a good hypothesis, we **measure error over the training set**.
- Formally: Suppose we compare hypotheses  $f_1$  and  $f_2$ .
  - Assume  $f_1$  has lower error on the training set.
  - If  $f_2$  has lower true error, then our algorithm is overfitting.

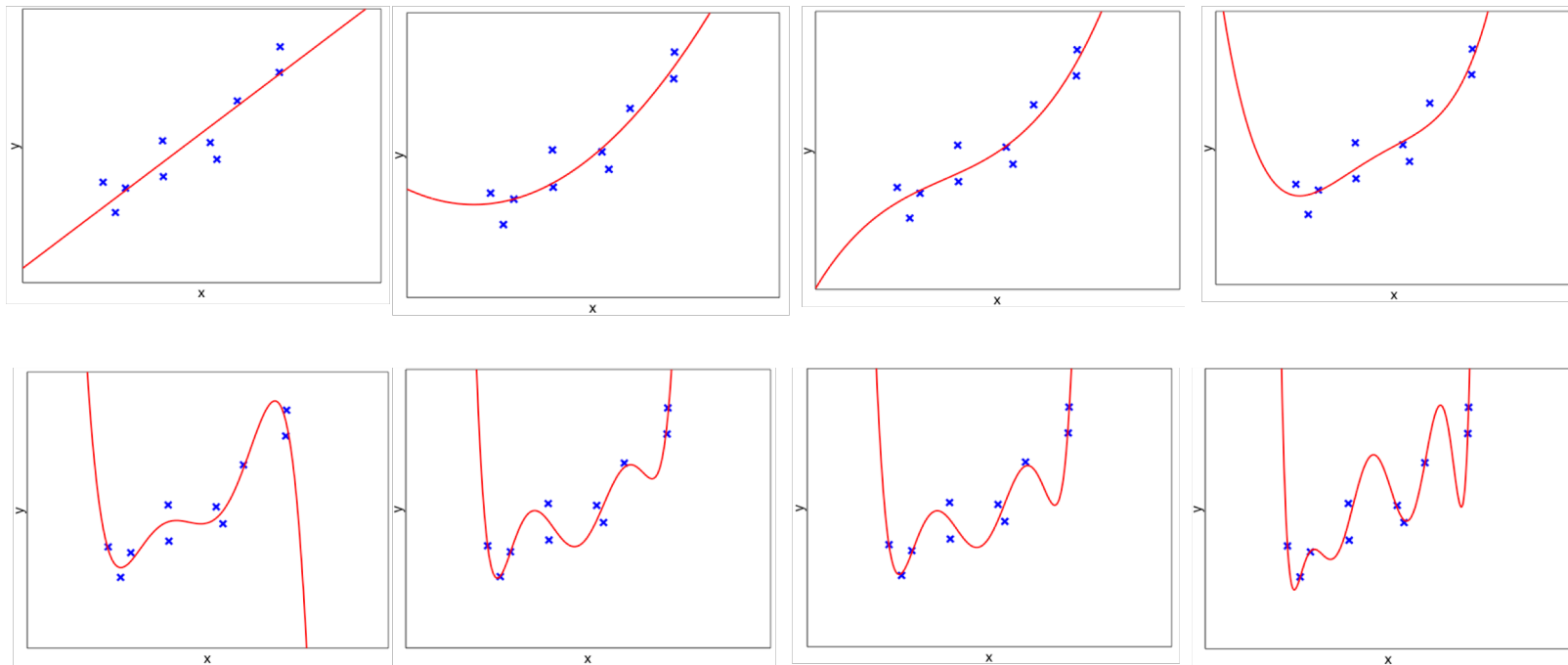
# Generalization

---

- In machine learning our goal is to develop models that **generalize**.
- With lots of parameters it is easy to perfectly fit a training set!
  - For example, if my model has more parameters than there are training points, then the model can just memorize the training data!
- Performing well on **previously unseen data** (i.e., generalization) is the real challenge.
- But how do we evaluate generalization?

# Overfitting

Which hypothesis has the lowest **true error**?



# Evaluating on held out data

---

- Partition your data into a **training set**, **validation set**, and **test set**.
  - The proportions in each set can vary.
- **Training set** is used to **fit a model** (find the best hypothesis in the class).
- **Validation set** is used for **model selection**, i.e., to estimate true error and compare hypothesis classes. (E.g., compare different order polynomials).
- **Test set** is what you report the final accuracy on.

# Evaluating on held out data

Training set =	$X = \begin{bmatrix} 0.75 & 0.86 & 1 \\ 0.01 & 0.09 & 1 \\ 0.73 & -0.85 & 1 \\ 0.76 & 0.87 & 1 \\ 0.19 & -0.44 & 1 \\ 0.18 & -0.43 & 1 \end{bmatrix}$	$Y = \begin{bmatrix} 2.49 \\ 0.83 \\ -0.25 \\ 3.10 \\ 0.87 \\ 0.02 \end{bmatrix}$
Validation set =	$\begin{bmatrix} 1.22 & -1.10 & 1 \\ 0.16 & 0.40 & 1 \end{bmatrix}$	$\begin{bmatrix} -0.12 \\ 1.81 \end{bmatrix}$
Test set =	$\begin{bmatrix} 0.93 & -0.96 & 1 \\ 0.03 & 0.17 & 1 \end{bmatrix}$	$\begin{bmatrix} -0.83 \\ 0.43 \end{bmatrix}$



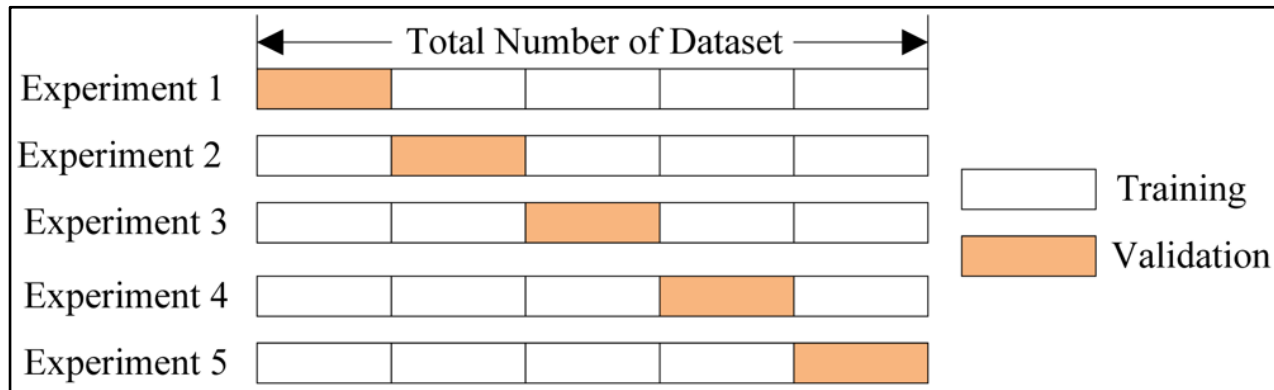
# Validation set vs. test set

---

- **Validation set** is used **compare** different models during development.
  - Compare hypothesis/model classes. E.g., should I use a first- or second-order polynomial fit?
  - Compare hyperparameters (i.e., a parameter that is **not** learned but that could impact performance). E.g., what learning rate should I use?
- **Test set** is used **evaluate the final performance of your model** (e.g., compare between research groups).
- Why separate them? If you try too many models or hyperparameter settings, you might start to overfit the validation set! (A kind of “meta”-overfitting).

# k-fold cross validation

- Instead of just one validation set, we can evaluate on many splits!
  - Consider  $k$  partitions of the training/non-test data (usually of equal size).
  - Train with  $k-1$  subsets, validate on  $k^{\text{th}}$  subset. Repeat  $k$  times.
  - Average the prediction error over the  $k$  rounds/folds.



(increases  
computation time  
by a factor of  $k$ )

Source: <http://stackoverflow.com/questions/31947183/how-to-implement-walk-forward-testing-in-sklearn>

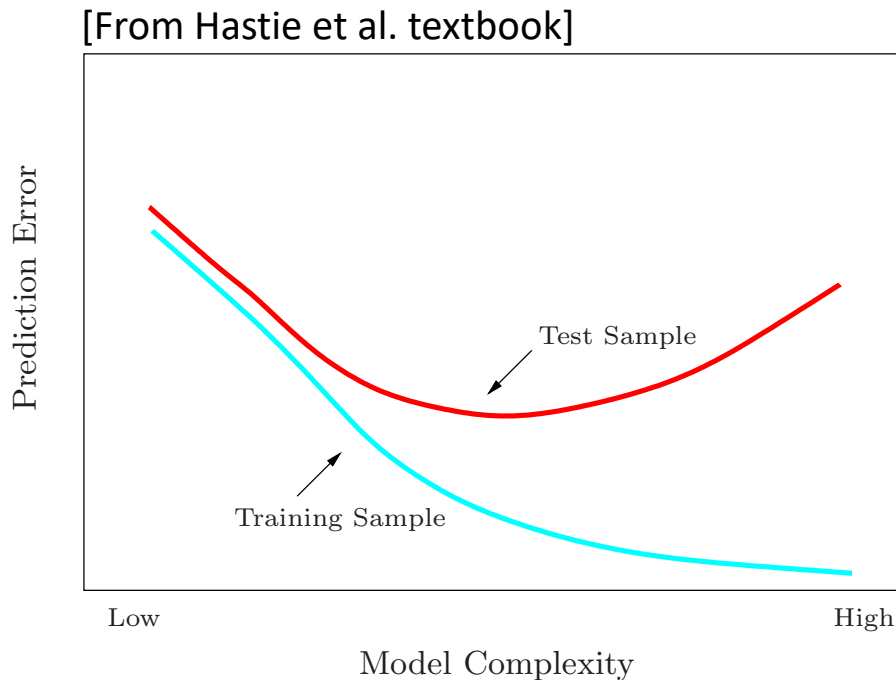
# Leave-one-out cross validation

---

- Let  $k = n$ , the size of the training set
- For each model / hyperparameter setting,
  - Repeat  $n$  times:
    - Set aside one instance  $\langle \mathbf{x}_i, y_i \rangle$  from the training set.
    - Use all other data points to find  $\mathbf{w}$  (optimization).
    - Measure prediction error on the held-out  $\langle \mathbf{x}_i, y_i \rangle$ .
  - Average the prediction error over all  $n$  subsets.
- Choose the setting with lowest estimated true prediction error.

# Generalization: test vs. train error

- Overly simple model:
  - High training error and high test error.
- Overly complex model:
  - Low training error but high test error.



**FIGURE 2.11.** *Test and training error as a function of model complexity.*

# Traditional statistics vs. machine learning

---

- Traditional statistics: how can we accurately estimate the effect of  $x$  on  $y$ ?
- Machine learning: how can we design a function that predicts  $x$  from  $y$  and generalizes to unseen data?

# Recap: Evaluating on held out data

---

- Partition your data into a **training set**, **validation set**, and **test set**.
  - The proportions in each set can vary.
- **Training set** is used to **fit a model** (find the best hypothesis in the class).
- **Validation set** is used for **model selection**, i.e., to estimate true error and compare hypothesis classes. (E.g., compare different order polynomials).
- **Test set** is what you report the final accuracy on.

# What you should know

---

- Linear regression (hypothesis class, error function, algorithm).
- How linear regression can fail
- Gradient descent method (algorithm, properties).
- Train vs. validation vs. test sets.
- The notion of **generalization**.