SVAN 2016 Mini Course: Stochastic Convex Optimization Methods in Machine Learning

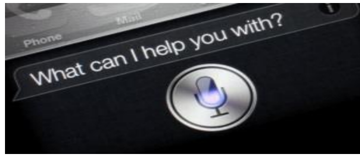
Mark Schmidt

University of British Columbia, May 2016

www.cs.ubc.ca/~schmidtm/SVAN16

Big Data Phenomenon

- We are collecting and storing data at an unprecedented rate.
- Examples:
  - News articles and blog posts.
  - YouTube, Facebook, and WWW.
  - Credit cards transactions and Amazon purchases.
  - Gene expression data and protein interaction assays.
  - Maps and satellite data.
  - Large hadron collider and surveying the sky.
  - Phone call records and speech recognition results.
  - Video game worlds and user actions.



Machine Learning

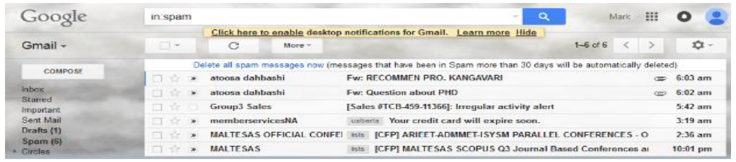
- What do you do with all this data?
  - Too much data to search through it manually.
- But there is valuable information in the data.
  - Can we use it for fun, profit, and/or the greater good?
- Machine learning: use computers to automatically detect patterns in data and make predictions or decisions.
- Most useful when:
  - Don’t have a human expert.
  - Humans can’t explain patterns.
  - Problem is too complicated.

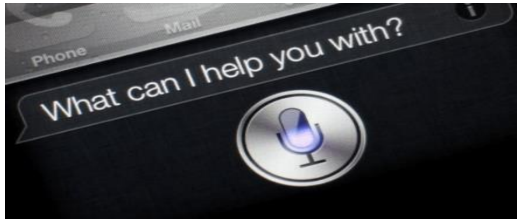
Machine Learning vs. Statistics

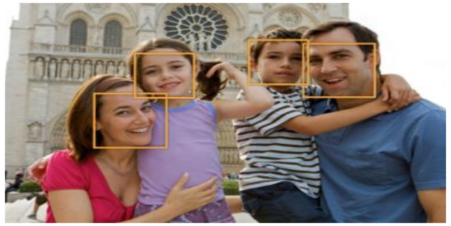
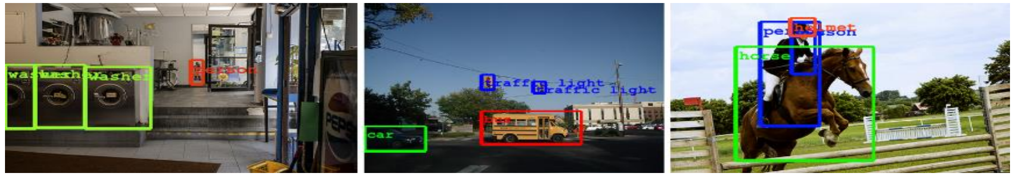
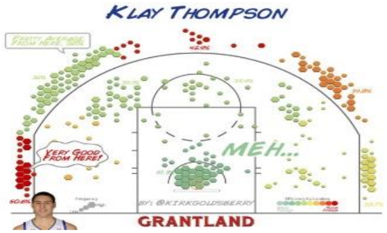
- Machine learning (ML) is very similar to statistics.
  - A lot of topics overlap.
- But ML places more emphasis on:
  1. Computation and large datasets.
  2. Predictions rather than descriptions.
  3. Non-asymptotic performance.
  4. Models that work across domains.
- The field is growing very fast:
  - ~2500 attendees at NIPS 2014, ~4000 at NIPS 2015.
  - Influence of $$$, too.

Applications

- Spam filtering.
- Credit card fraud detection.
- Product recommendation.
- Motion capture.
- Machine translation.
- Speech recognition.
- Face detection.
- Object detection.
- Sports analytics.

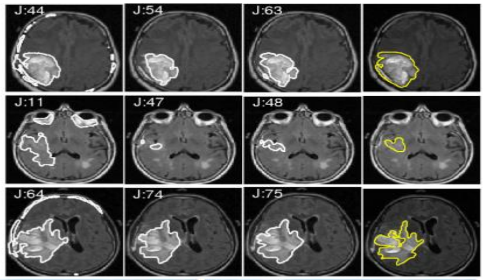


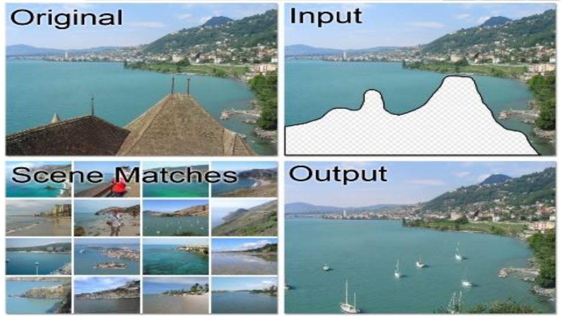



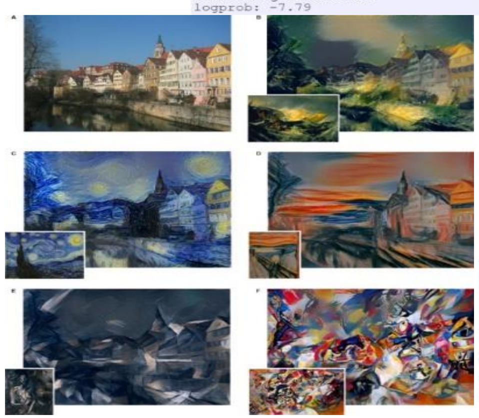




Applications

- Personal Assistants.
- Medical imaging.
- Self-driving cars.
- Scene completion.
- Image search and annotation.
- Artistic rendering.
- Your research?





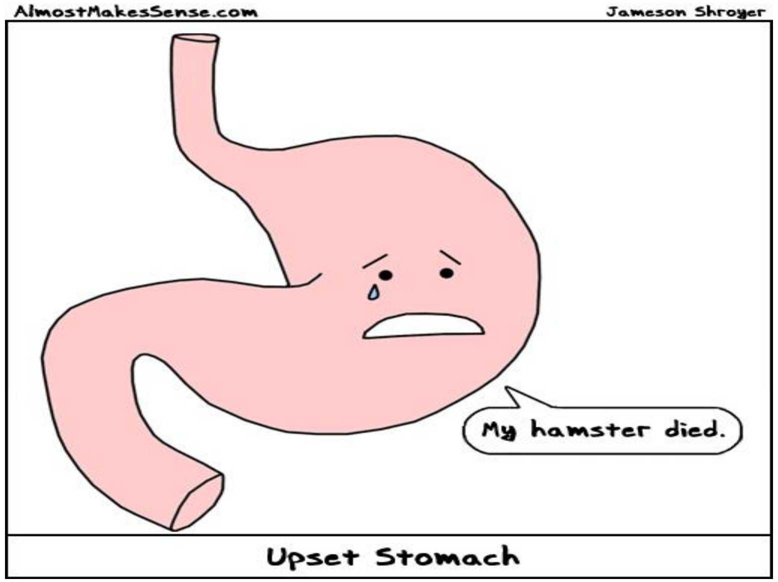


Course Outline (Approximate)

- Day 1:
  - L1: Linear regression, nonlinear bases.
  - L2: Validation, regularization.
- Day 2:
  - L3: Loss functions, convex functions
- Day 3:
  - L4: Gradient methods, L1-regularization
  - L5: Coordinate optimization, structure sparsity.
- Day 4:
  - L6: Projected-gradient, proximal-gradient
- Day 5:
  - L7: Stochastic subgradient, stochastic average gradient.
  - L8: Kernel trick, Fenchel duality.
- If time allows:
  - Alternating minimization, non-uniform sampling, parallelization, non-convex.

Motivating Example: Food Allergies

- You frequently start getting an upset stomach



- You suspect an adult-onset food allergy.

http://www.cliparthut.com/upset-stomach-clipart-cn48e5.html

Motivating Example: Food Allergies

- You start recording food and IgE levels each day:

	Egg	Milk	Fish	Wheat	Shellfish	Peanuts	…	IgE
Day 1	0	0.7	0	0.3	0	0		700
Day 2	0.3	0.7	0	0.6	0	0.01		740
Day 3	0	0	0	0.8	0	0		50
Day 4	0.3	0.7	1.2	0	0.10	0.01		950

- We want to write a program that:
  – Takes food levels for the day as an input.
  – Predicts IgE level for the day as the output.
- But foods interact: ‘formula’ mapping foods to IgE is hard to find:
  – Given the data, we could use machine learning to write this program.
  – The program will predict target (IgE levels) given features (food levels).

Supervised Learning

- This is an example of supervised learning:
  - Input is 'n' training examples (x_i, y_i).

$$
n\left\{
\begin{array}{l}
x_1 = \overbrace{[\ 0 \quad 0.7 \quad 0 \quad 0.3 \quad 0 \quad 0\ ]}^{d} \\
x_2 = [\ 0.3 \quad 0.7 \quad 0 \quad 0.6 \quad 0 \quad 0.01\ ] \\
x_3 = [\ 0 \quad 0 \quad 1.2 \quad 0 \quad 0.1 \quad 0.01\ ] \\
\quad \vdots
\end{array}
\right.
\qquad
\begin{array}{l}
y_1 = 700 \\
y_2 = 740 \\
y_3 = 50 \\
\quad \vdots
\end{array}
$$

  - x_i is the features for example 'i' (we'll use 'd' as the number of features).
    - In this case, the quantities of food eaten on day 'i'.
  - y_i is target for example 'i'.
    - In this case, the level of IgE.
  - Output is a function mapping from x_i space to y_i space.

$f(\text{egg}, \text{milk}, \text{fish}, \text{wheat}, \text{shellfish}, \text{peanuts}) = \text{IgE}.$ $\quad f(x_1) = f(0, 0.7, 0, 0.3, 0, 0) = 700 = y_1$

Supervised Learning

- Supervised learning is most successful ML method:
  - Spam filtering, Microsoft Kinect, speech recognition, object detection, etc.
- Most useful when:
  - You don't know how to map from inputs to outputs.
  - But you have a lot of input-to-output examples.

If $w_6 > 0$

more peanuts ⇒ more IgE

- When y_i is continuous, it's called regression.
- Today, we consider the special case of linear regression:

feature

$$\hat{IgE} = w_1(\text{eggs}) + w_2(\text{milk}) + w_3(\text{fish}) + w_4(\text{wheat}) + w_5(\text{shellfish}) + w_6(\text{peanuts})$$

predicted value

weight for 'eggs'

weight for 'fish'

Linear Regression

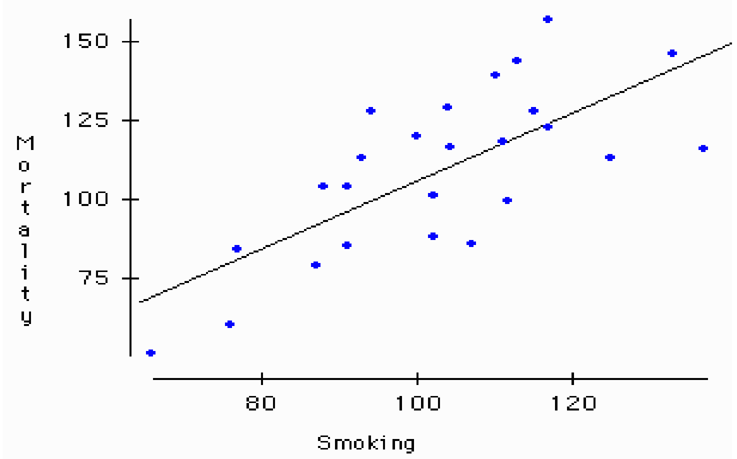
- Linear regression:
  - Prediction is weighted sum of features:

$$\hat{y}_i = w_1 x_{i1} + w_2 x_{i2} + w_3 x_{i3} + \cdots + w_d x_{id}$$

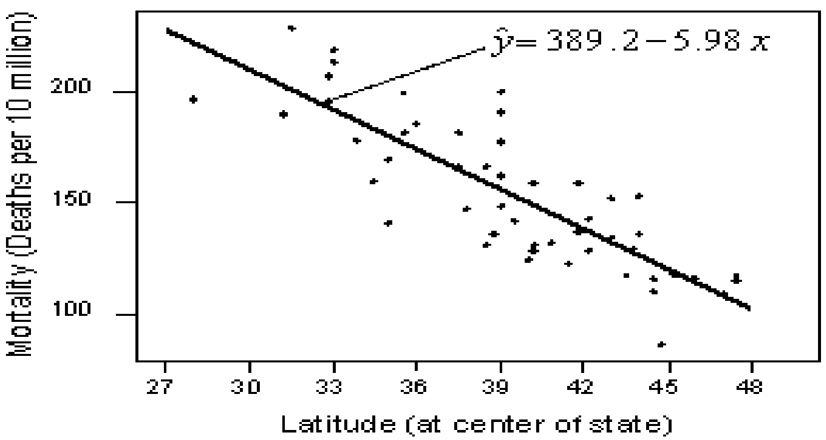
prediction for example 'i'

"weight" of feature 2

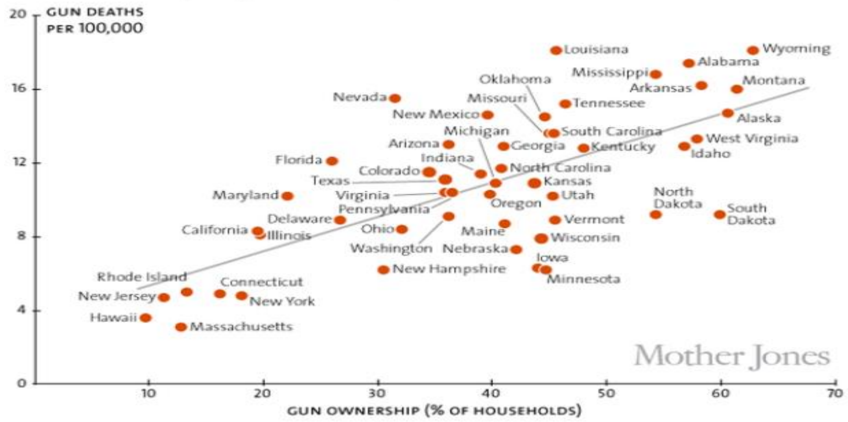
feature 2 for example 'i'



Skin cancer mortality versus State latitude



Gun ownership vs. gun deaths, by state



http://www.cvgs.k12.va.us:81/digstats/main/inferant/d_regrs.html
https://onlinecourses.science.psu.edu/stat501/node/11
http://www.vox.com/2015/10/3/9444417/gun-violence-united-states-america

Least Squares

- Supervised learning goal:

If we have $x_i = [\overset{\text{egg}}{0} \;\; \overset{\text{milk}}{0.7} \;\; \overset{\text{fish}}{0} \;\; \overset{\text{wheat}}{0.3} \;\; \overset{\text{shellfish}}{0} \;\; \overset{\text{peanut}}{0}]$ and $y_i = 700$,
then we have

$$\hat{y}_i = w_1(0) + w_2(0.7) + w_3(0) + w_4(0.3) + w_5(0) + w_6(0),$$

and we want $(\hat{y}_i - 700)$ to be small.

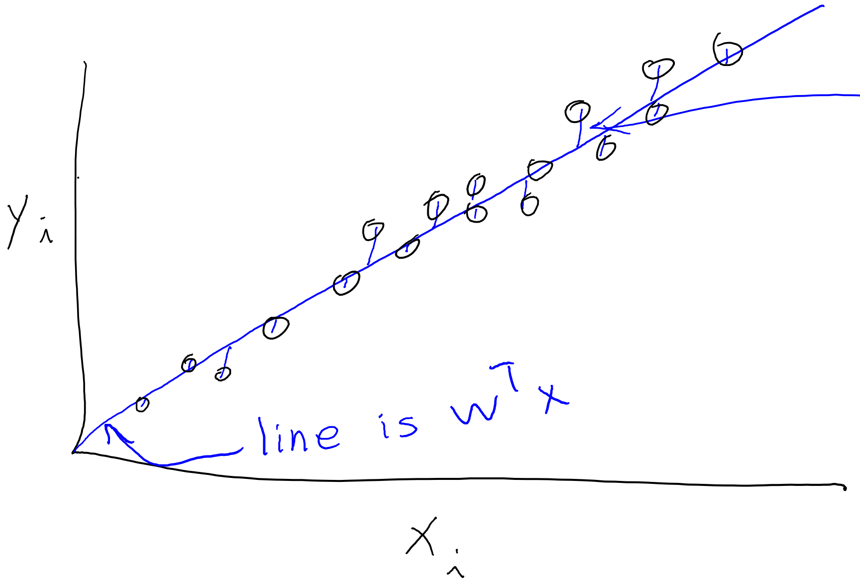
- Can we choose weights to make this happen?
- The classic way to do is minimize square error:

Find $\hat{w}$ that minimizes $(\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 + (\hat{y}_3 - y_3)^2 + \cdots$

This should make the average $(\hat{y}_i - y_i)$ small.

Least Squares Objective

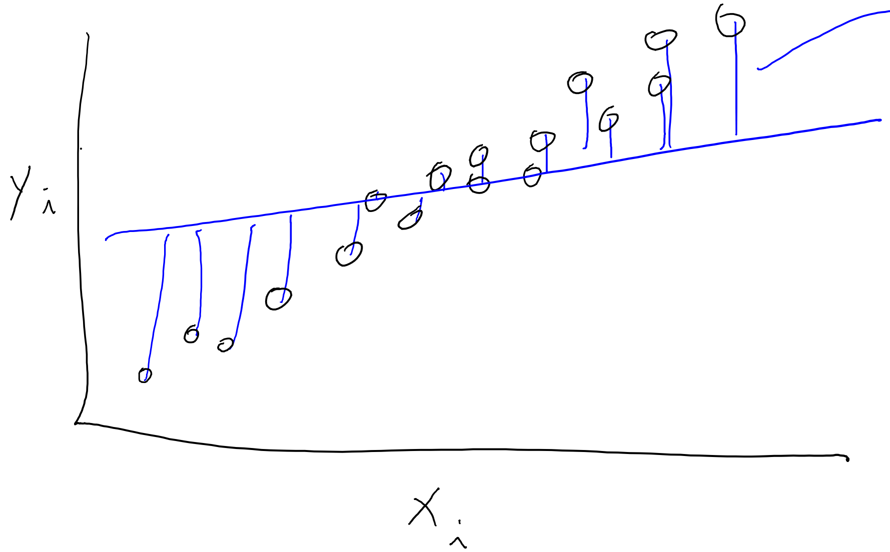
- Classic procedure: minimize squared error:



"errors" $(y_i - \hat{y}_i)$,
if these are <u>small</u>
then the line predicts
y_i accurately.

Least Squares Objective

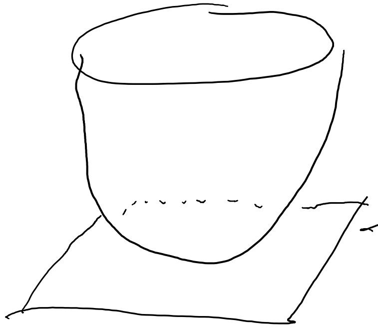
- Classic procedure: minimize squared error:



If these vertical distances, $(y_i - \hat{y}_i)$ are large, then the line does a bad job of predicting y_i.

Least Squares Objective

- Why squared error?
  - There is some theory justifying this choice:
    - Errors follow a normal distribution.
    - Central limit theorem.
  - Computation: quadratic function, so easy to minimize.
- How do we calculate the optimal 'w'?
  - The error is a convex quadratic function of 'w'.
  - We can take the gradient and set it to zero.



at minimizer, gradient is (0,0).

Least Squares (Vector Notation)

- So our objective is to minimize

$$(\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 + (\hat{y}_3 - y_3)^2 + \cdots + (\hat{y}_n - y_n)^2$$

in terms of 'w', where we have:

$$\hat{y}_i = w_1 x_{i1} + w_2 x_{i2} + w_3 x_{i3} + \cdots + w_d x_{id}$$

$$\overset{d \times 1}{w = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_d \end{bmatrix}}$$

- Written with summation notation:

$$\underset{w \in \mathbb{R}^d}{\text{minimize}} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 \quad \text{with} \quad \hat{y}_i = \sum_{j=1}^{d} w_j x_{ij} = w^T x_i$$

- Observe that prediction is inner product.
- This lets us write objective as: $\underset{w \in \mathbb{R}^d}{\text{minimize}} \sum_{i=1}^{n} (w^T x_i - y_i)^2$

Least Squares (Matrix Notation)

- So least squares using vectors 'w' and 'x_i' is:

$$\underset{w \in \mathbb{R}^d}{\text{minimize}} \sum_{i=1}^{n} (w^T x_i - y_i)^2$$

- To derive solution, need matrix notation:

$$\underset{w \in \mathbb{R}^d}{\text{minimize}} \; \|Xw - y\|^2$$

- Where:
  - Each row of feature matrix 'X' is an 'x_i^T'.
  - Element 'i' of target vector 'y' is 'y_i'.
  - $||z||$ is the Euclidean norm.

$$\begin{array}{c} n \times d \\ X = \begin{bmatrix} \text{——} & x_1^T & \text{——} \\ \text{——} & x_2^T & \text{——} \\ \text{——} & x_3^T & \text{——} \\ & \vdots & \\ \text{——} & x_n^T & \text{——} \end{bmatrix} \end{array} \qquad \begin{array}{c} n \times 1 \\ y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} \end{array}$$

$$\|z\| = \sqrt{\sum_{j=1}^{d} z_j^2}$$

observe

$$\|z\|^2 = \sum_{j=1}^{d} z_j^2 = \sum_{j=1}^{d} z_j z_j = z^T z$$

Least Squares (Matrix Notation)

$$\underset{w \in \mathbb{R}^d}{\text{minimize}} \sum_{i=1}^{n} (w^T x_i - y_i)^2$$

$$\Updownarrow$$

$$\underset{w \in \mathbb{R}^d}{\text{minimize}} \sum_{i=1}^{n} r_i^2 \quad \text{where } r_i = w^T x_i - y_i$$

"residual"

$$\Updownarrow$$

$$\underset{w \in \mathbb{R}^d}{\text{minimize}} \|r\|^2 \Longleftrightarrow \boxed{\underset{w \in \mathbb{R}^d}{\text{minimize}} \|Xw - y\|^2}$$

$$r = \begin{bmatrix} w^T x_1 - y_1 \\ w^T x_2 - y_2 \\ w^T x_3 - y_3 \\ \vdots \\ w^T x_n - y_n \end{bmatrix} = \underbrace{\begin{bmatrix} x_1^T w \\ x_2^T w \\ x_3^T w \\ \vdots \\ x_n^T w \end{bmatrix}}_{Xw} - \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix}}_{y} = Xw - y$$

Observe that $w^T x_1 = x_1^T w$ (scalar)

- Where:
  - Each row of feature matrix 'X' is an 'x_i^T'.
  - Element 'i' of target vector 'y' is 'y_i'.
  - ||z|| is the Euclidean norm.

$$X = \begin{bmatrix} \text{———} & x_1^T & \text{———} \\ \text{———} & x_2^T & \text{———} \\ \text{———} & x_3^T & \text{———} \\ & \vdots & \\ \text{———} & x_n^T & \text{———} \end{bmatrix} \qquad y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix}$$

$$\|z\| = \sqrt{\sum_{j=1}^{d} z_j^2}$$

observe

$$\|z\|^2 = \sum_{j=1}^{d} z_j^2 = \sum_{j=1}^{d} z_j z_j = z^T z$$

Gradient Vector

- Deriving least squares solution:
  - Set gradient equal to zero and solve for ‘w’.
- Recall gradient is vector of partial derivatives:

$$\nabla f(w) = \begin{bmatrix} \frac{\partial f}{\partial w_1} \\ \frac{\partial f}{\partial w_2} \\ \frac{\partial f}{\partial w_3} \\ \vdots \\ \frac{\partial f}{\partial w_d} \end{bmatrix}$$

- Gradients appear a lot in ML.

Digression: Linear Functions

- A linear function of 'w' is a function of the form:

$$f(w) = \underbrace{a}_{\text{vector}}^T w + \underbrace{\beta}_{\text{scalar}}$$

- Gradient of linear function in matrix notation:
  1. Convert to summation notation: $f(w) = \sum_{i=1}^{d} a_i w_i + \beta$
  2. Take generic partial derivative: $\frac{\partial}{\partial w_i} f(w) = a_i$
  3. Assemble into vector and simplify: $\nabla f(w) = \begin{bmatrix} \frac{\partial}{\partial w_1} f(w) \\ \frac{\partial}{\partial w_2} f(w) \\ \vdots \\ \frac{\partial}{\partial w_d} f(w) \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_d \end{bmatrix} = a$

Digression: Quadratic Functions

- A quadratic function 'w' is a function of the form:

$$f(w) = \frac{1}{2} w^T \underbrace{A}_{d \times d \text{ matrix}} w + \underbrace{b^T}_{\text{vector}} w + \underbrace{\gamma}_{\text{scalar}}$$

Generalizes $\frac{d}{dw}\left[aw^2\right] = 2aw$

- Gradient of quadratic function in matrix notation:

Let $f(w) = w^T A w$.

1. Summation notation:

$$f(w) = w^T \underbrace{\begin{bmatrix} \sum_{j=1}^d a_{1j} w_j \\ \sum_{j=1}^d a_{2j} w_j \\ \vdots \\ \sum_{j=1}^d a_{dj} w_j \end{bmatrix}}_{Aw} = \sum_{i=1}^d \sum_{j=1}^d w_i a_{ij} w_j$$

$$= \sum_{i=1}^d \left(a_{ii} w_i^2 + \sum_{j \neq i} w_i a_{ij} w_j\right)$$

2. Generic partial:

$$\frac{\partial f}{\partial w_i}\left[w^T A w\right] = 2 a_{ii} w_i + \sum_{j \neq i} w_j a_{ji} + \sum_{j \neq i} a_{ij} w_j$$

$$= \sum_{j=1}^d w_j a_{ji} + \sum_{j=1}^d a_{ij} w_j$$

3. Assemble/simplify:

$$\nabla f(w) = \begin{bmatrix} \sum_{j=1}^d w_j a_{j1} + \sum_{j=1}^d a_{1j} w_j \\ \sum_{j=1}^d w_j a_{j2} + \sum_{j=1}^d a_{2j} w_j \\ \vdots \\ \sum_{j=1}^d w_j a_{jd} + \sum_{j=1}^d a_{dj} w_j \end{bmatrix}$$

$$= A^T w + A w$$

$$= (A^T + A) w$$

If symmetric:

$$= (A + A) w$$

$$= 2 A w$$

Least Squares Solution – Part 1

- Our least squares problem is:

$$\underset{w \in \mathbb{R}^d}{\text{minimize}} \ \frac{1}{2}\|Xw - y\|^2$$

($\frac{1}{2}$ doesn't change minimizer)

- We'll expand:

$$f(w) = \frac{1}{2}\|Xw - y\|^2 = \frac{1}{2}(Xw - y)^T(Xw - y)$$

$$= \frac{1}{2}(w^TX^T - y^T)(Xw - y)$$

$$= \frac{1}{2}(w^TX^T(Xw - y) - y^T(Xw - y))$$

$$= \frac{1}{2}(w^TX^TXw - w^TX^Ty - y^TXw + y^Ty)$$

$$= \frac{1}{2}w^TX^TXw - w^TX^Ty + \frac{1}{2}y^Ty$$

Let $X^Ty = v$, then use $w^Tv = v^Tw$.

Least Squares Solution – Part 2

- So our objective function can be written:

$$f(w) = \underbrace{\frac{1}{2} w^T X^T X w}_{\substack{\text{quadratic} \\ \text{(symmetric)}}} - \underbrace{w^T X^T y}_{\text{linear}} + \underbrace{\frac{1}{2} y^T y}_{\text{constant}}$$

$w^T A w \leftarrow$ (symmetric)   linear $\rightarrow w^T b$

- Using our two tedious matrix calculus exercises:

$$\nabla f(w) = X^T X w - X^T y$$

- Setting the gradient equal to zero:

$$0 = X^T X w - X^T y \quad \text{or} \quad \boxed{X^T X w = X^T y}$$

This is a linear system like $Ax = b$ from linear algebra.

Any solution gives a least squares solution.

If $X^T X$ is invertible, pre-multiply by $(X^T X)^{-1}$:

$$(X^T X)^{-1} (X^T X) w = (X^T X)^{-1} (X^T y)$$

$$I w = (X^T X)^{-1} (X^T y)$$

$$w = (X^T X)^{-1} (X^T y)$$

Least Squares Solution – Part 3

- So finding a least solution means finding a 'w' satisfying:

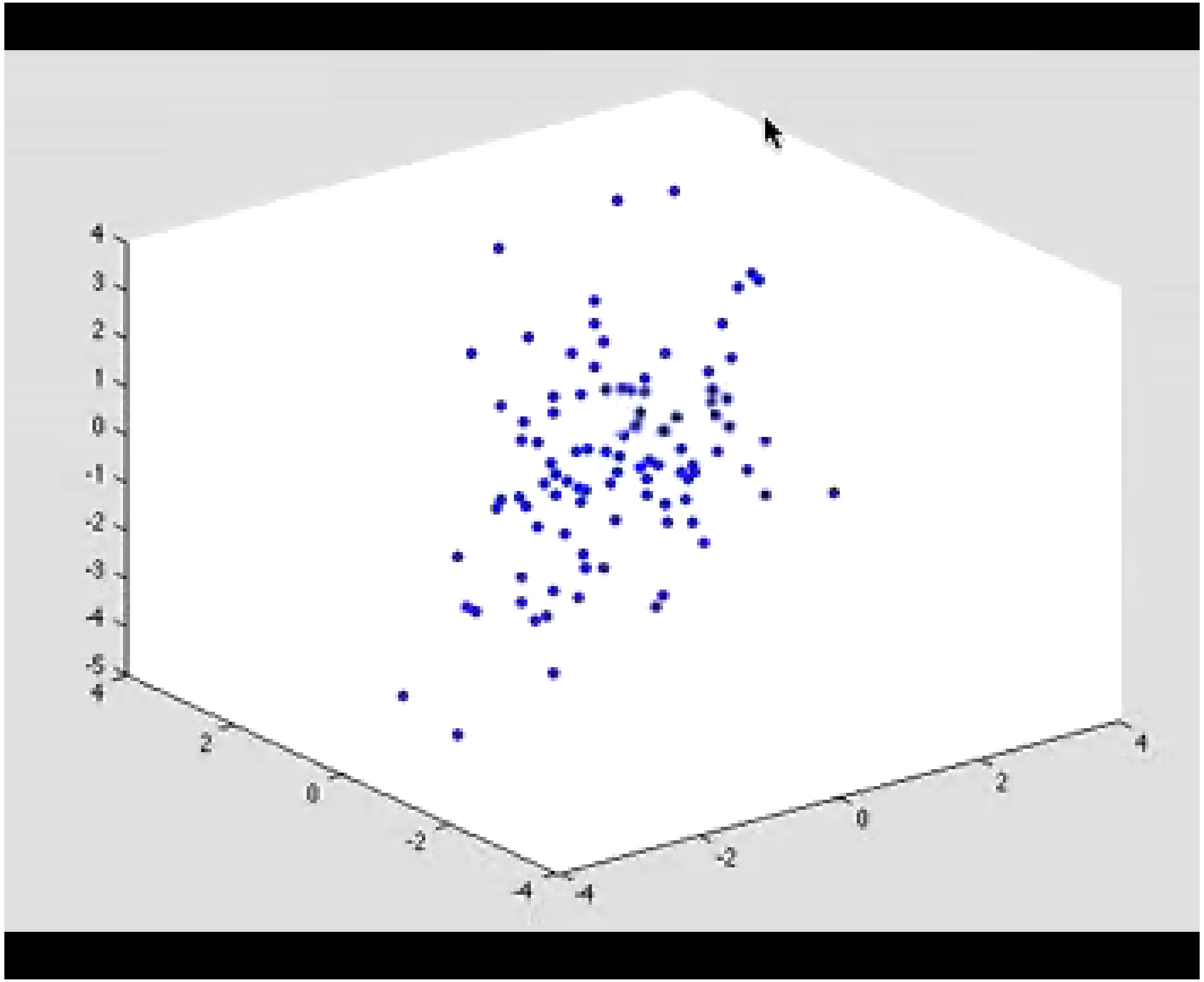
$$X^T X w = X^T y$$

In Matlab: $\overset{x}{w} = \overset{A}{(X'*X)} \backslash \overset{b}{(X'*y)}$

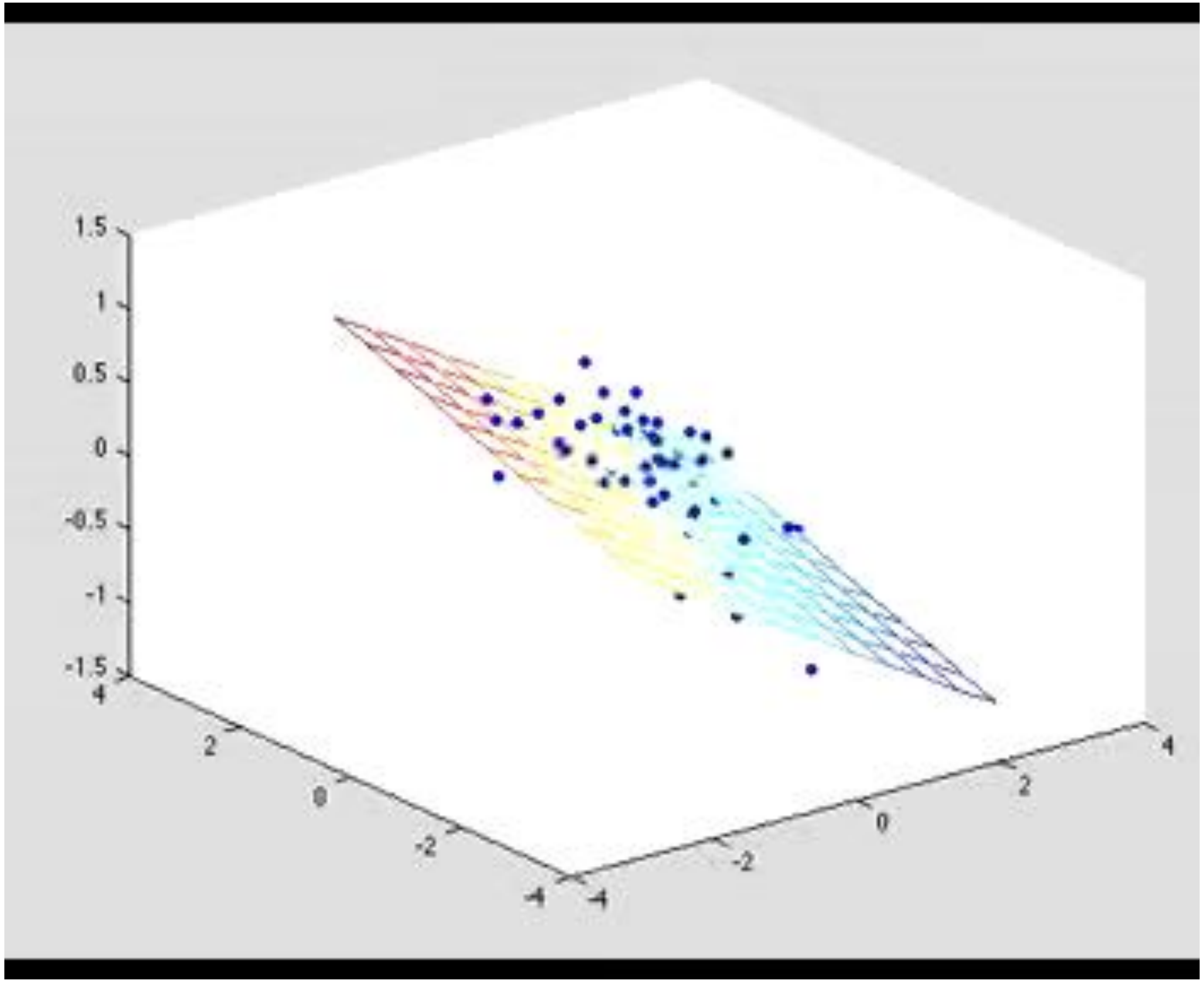
"solve $Ax = b$"

- What is the cost of computing this?
  1. Forming $X^T y$ costs $O(nd)$.
  2. Forming $X^T X$ costs $O(nd^2)$.
  3. Solving a 'd' by 'd' linear system costs $O(d^3)$.
     - If we use LU decomposition (AKA Gaussian elimination).
  - Total cost: $O(nd^2 + d^3)$.
    - We can solve "medium-size" problems ($n = 10k$, $d = 1000$).
    - We can't solve "large" problems ($n = 100k$, $d = 10m$).

Least Squares in 2-Dimensions



Least Squares in 2-Dimensions

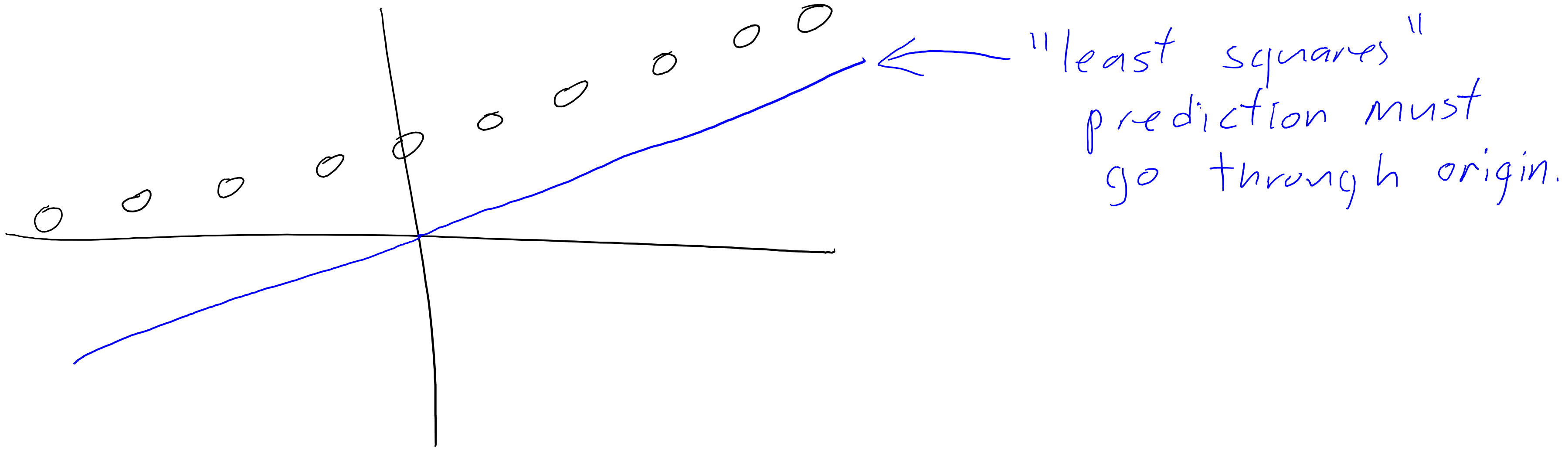


Problem with Linear Least Squares

- Least squares is very old and widely-used.
  - But it usually works terribly.
- Issues with least squares model:
  - It assumes a linear relationship between x_i and y_i.
  - It might predict poorly for *new* values of x_i.
  - X^TX might not be invertible.
  - It is sensitive to outliers.
  - It might predict outside known range of y_i values.
  - It always uses all features.
  - 'd' might be so big we can't store X^TX.
- We're going to start fixing these problems.

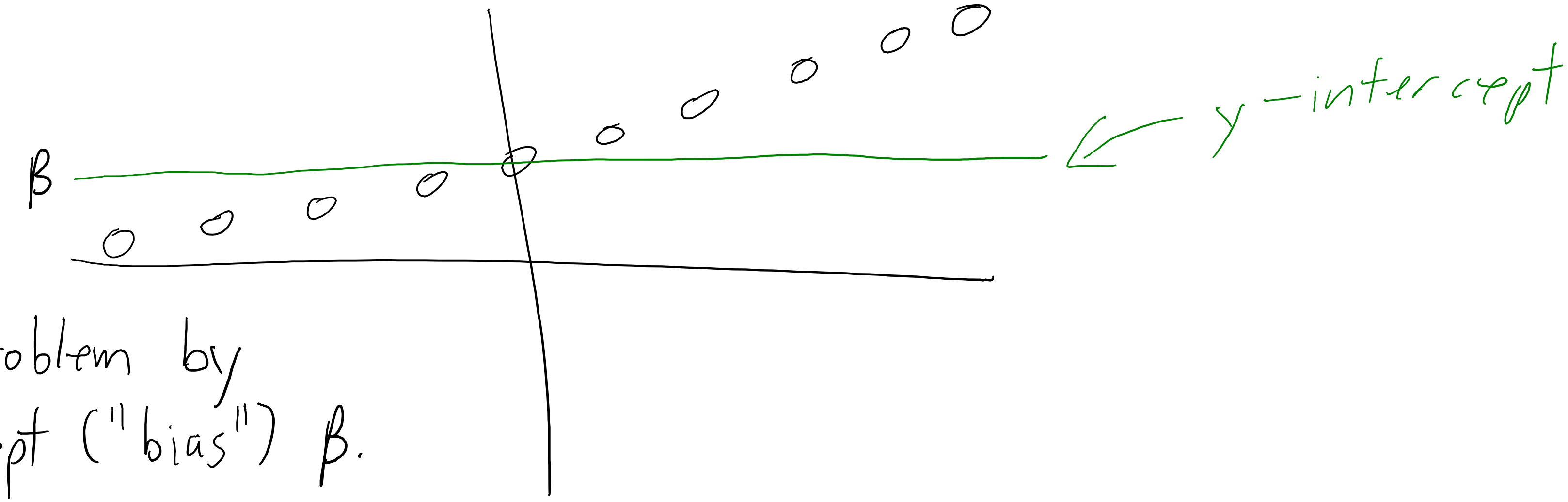
First Problem: y-intercept

Since we predict $\hat{y}_i = w^T x_i$, we must predict $y_i = 0$ if $x_i = 0$.



First Problem: y-intercept

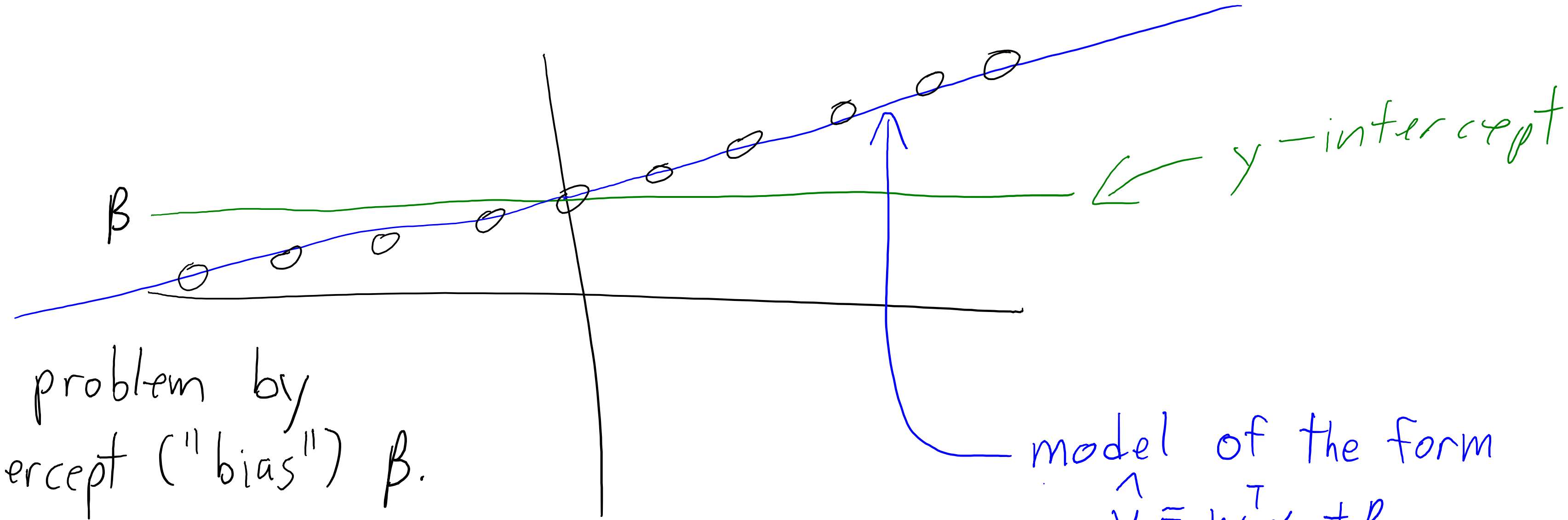
Since we predict $\hat{y}_i = w^T x_i$, we <u>must</u> predict $y_i = 0$ if $x_i = 0$.



We can <u>fix</u> this problem by adding a y-intercept ("bias") β.

First Problem: y-intercept

Since we predict $\hat{y}_i = w^T x_i$, we must predict $y_i = 0$ if $x_i = 0$.



model of the form
$\hat{y}_i = w^T x_i + \beta$

We can fix this problem by adding a y-intercept ("bias") β.

Simple Trick to Incorporate Bias Variable

- Simple way to add y-intercept is adding column of ‘1’ values:

$$X = \begin{bmatrix} 0.1 & 1.2 \\ -1 & 1.3 \\ 0.8 & 1.1 \end{bmatrix} \Longrightarrow \bar{X} = \begin{bmatrix} 1 & 0.1 & 1.2 \\ 1 & -1 & 1.3 \\ 1 & 0.8 & 1.1 \end{bmatrix}$$

- The first element of least squares now represents the bias β:

$$\bar{w} = (\bar{X}^T \bar{X})^{-1} (\bar{X}^T y),$$

$$\bar{w} = \begin{bmatrix} \beta \\ \left.\begin{matrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{matrix}\right\} w \end{bmatrix}$$

$$\begin{aligned} \hat{y}_i &= \bar{w}^T \bar{x}_i \\ &= \bar{w}_0 \bar{x}_{i1} + \bar{w}_1 \bar{x}_{i2} + \bar{w}_2 \bar{x}_{i3} + \cdots + \bar{w}_d \bar{x}_{i(d+1)} \\ &= \beta(1) + w_1 x_{i1} + w_2 x_{i2} + \cdots + w_d x_{id} \\ &= \beta + w^T x_i \end{aligned}$$

Change of Basis

- This "change the features" trick also allows us to fit non-linear models.
- For example, instead of linear we might want a quadratic function:

$$\hat{y}_i = \beta + w_1 x_i + w_2 x_i^2$$

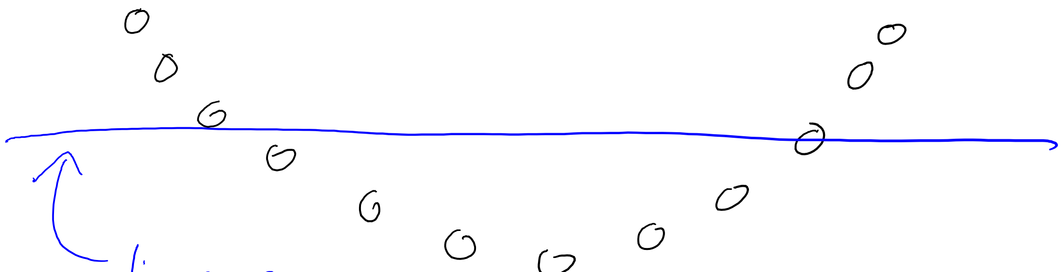
- We can do this by changing X (change of basis):

$$X = \begin{bmatrix} 0.2 \\ -0.5 \\ 1 \\ 4 \end{bmatrix} \qquad X_{poly} = \begin{bmatrix} 1 & 0.2 & (0.2)^2 \\ 1 & -0.5 & (-0.5)^2 \\ 1 & 1 & (1)^2 \\ 1 & 4 & (4)^2 \end{bmatrix}$$

- Now fit least squares with this matrix: $w = (X_{poly}^T X_{poly})^{-1} X_{poly}^T y$

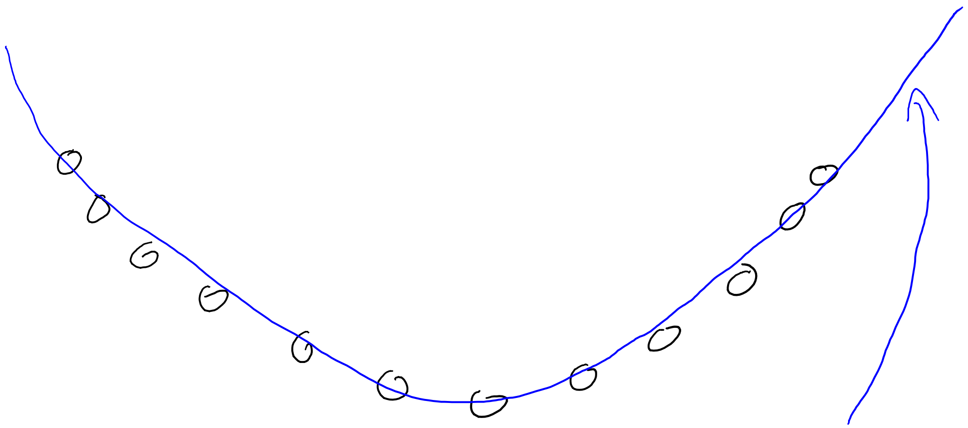
- Model is a linear function of w, but a quadratic function of x_i.

Change of Basis



linear least squares

$$\hat{y}_i = \beta + w_1 x_i$$



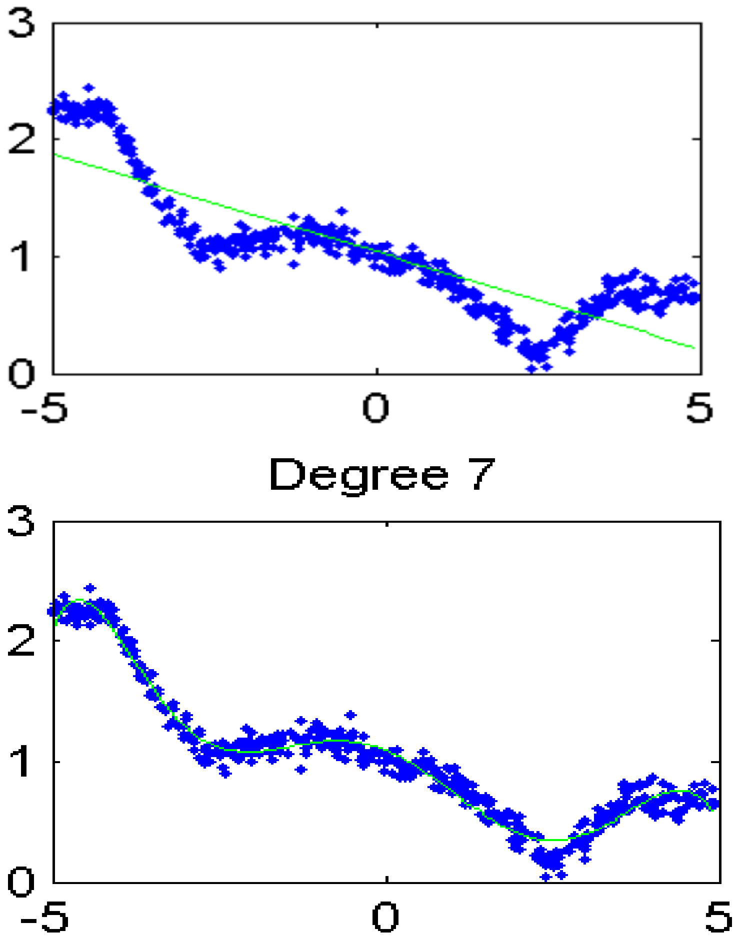
linear least squares with quadratic basis

$$\hat{y}_i = \beta + w_1 x_i + w_2 x_i^2$$

General Polynomial Basis

- We can have polynomial of degree ‘p’ by using a basis:

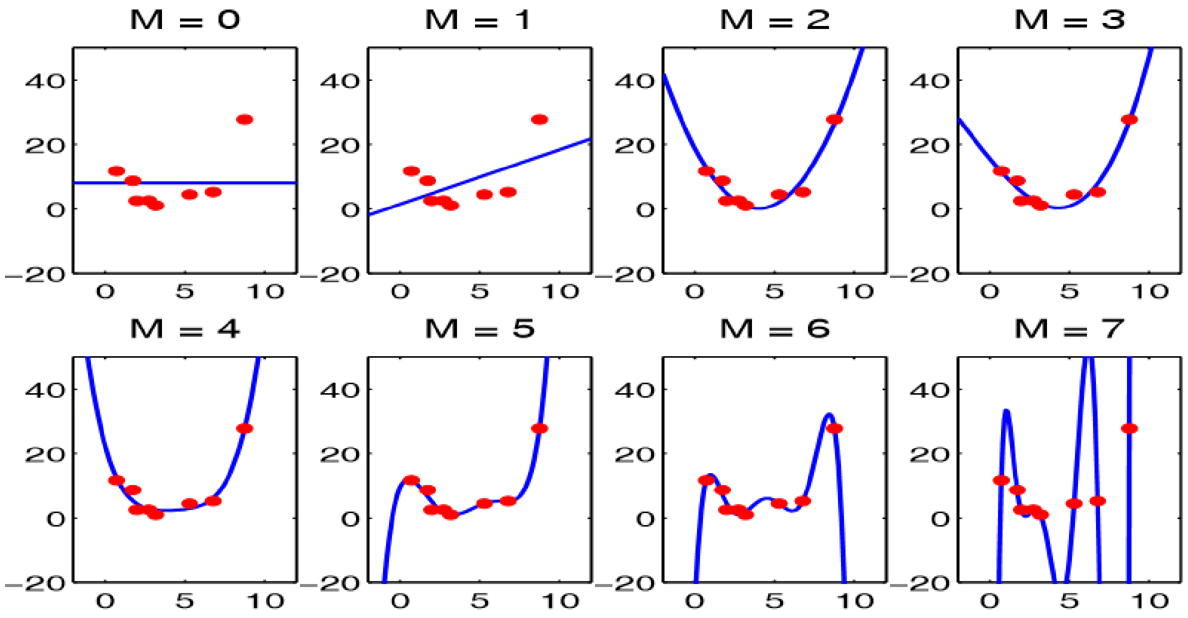
$$X_{poly} = \begin{bmatrix} 1 & x_1 & (x_1)^2 & \cdots & (x_1)^p \\ 1 & x_2 & (x_2)^2 & \cdots & (x_2)^p \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n & (x_n)^2 & \cdots & (x_n)^p \end{bmatrix}$$



- Numerically-nicer polynomial bases exist:
  - E.g., Lagrange polynomials.

Error vs. Degree of Polynomial

- Note that polynomial bases are nested:
  - I.e., model with basis of degree 7 has degree 6 as a special case.
- This means that as the degree 'd' increases, the error goes down.



- So does higher-degree always mean better model?

Training vs. Testing

- We fit our model using training data where we know y_i:

X =

Egg	Milk	Fish	Wheat	Shellfish	Peanuts	…
0	0.7	0	0.3	0	0	
0.3	0.7	0	0.6	0	0.01	
0	0	0	0.8	0	0	

y =

IgE
700
450
175

- But we aren't interested performance on this training data.
- Our goal is accurately predicts y_i on new test data:

Xtest =

Egg	Milk	Fish	Wheat	Shellfish	Peanuts	…
0.5	0	1	0.6	2	1	
0	0.7	0	1	0	0	

ytest =

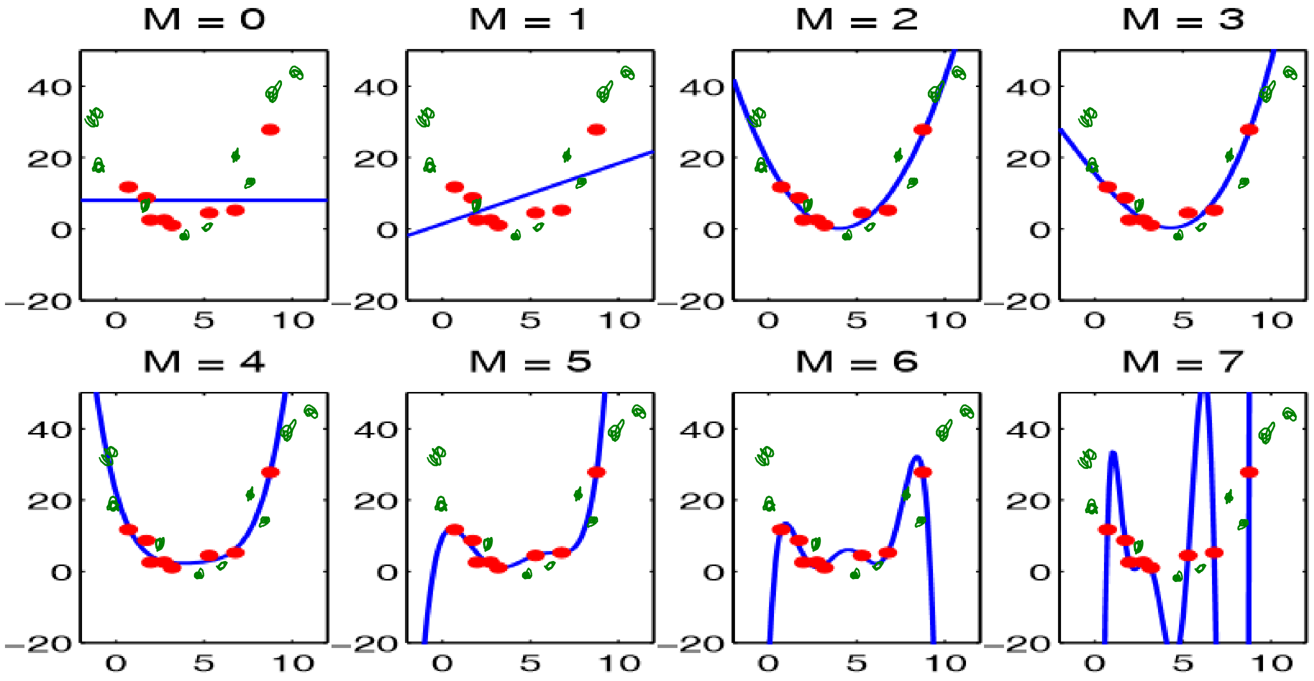
Sick?
?
?

Training vs. Testing

- We usually think of supervised learning in two phases:
  1. Training phase:
     - Fit a model based on the training data X and y.
  2. Testing phase:
     - Evaluate the model on new data that was not used in training.
- In machine learning, what we care about is the test error!
- Memorization vs learning:
  – Can do well on training data by memorizing it.
  – You’ve only “**learned**” if you can do well in new situations.

Error vs. Degree of Polynomial

- As the polynomial degree increases, the training error goes down.



- The test error also goes down initially, then starts going up.
  - Overfitting: test error is higher than training error.

Golden Rule of Machine Learning

- Even though what we care about is test error:
  - YOU CANNOT USE THE TEST DATA DURING TRAINING.
- Why not?
  - Finding the model that minimizes the test error is the goal.
  - But we’re only using the test error to gauge performance on new data.
  - Using it during training means it doesn’t reflect performance on new data.
- If you violate golden rule, you can overfit to the test data:


Tom Simonite
June 4, 2015

Why and How Baidu Cheated an Artificial Intelligence Test

Machine learning gets its first cheating scandal.

The sport of training software to act intelligently just got its first cheating scandal. Last month Chinese search

http://www.technologyreview.com/view/538111/why-and-how-baidu-cheated-an-artificial-intelligence-test/

Is Learning Possible?

- Does training error say anything about test error?
  - In general, NO!
  - Test data might have nothing to do with training data.
- In order to have any hope of learning we need assumptions.
- A standard assumption is that training and test data are IID:
  - "Independent and identically distributed".
  - New examples will behave like the existing objects.
  - The order of the examples doesn't matter.
  - Rarely true in practice, but often a good approximation.
- Field of learning theory examines learnability.

Fundamental Trade-Off

- Learning theory results tend to lead to a fundamental trade-off:
  1. How small you can make the training error.

     vs.

  2. How well training error approximates the test error.

- Different models make different trade-offs.
- Simple models (low-degree polynomials):
  - Training error is good approximation of test error:
    - Not very sensitive to the particular training set you have.
  - But don't fit training data well.
- Complex models (high-degree polynomials):
  - Fit training data well.
  - Training error is poor approximation of test error:
    - Very sensitive to the particular training set you have.

Back to reality…

- How do we decide polynomial degree in practice?
- We care about the test error.
- But we can't look at the test data.
- So what do we do?????

- One answer:
  - Validation set: *save part of your dataset to approximate the test error.*
- Randomly split training examples into 'train' and 'validate':
  - Fit the model based on the 'train' set.
  - Test the model based on the 'validate' set.

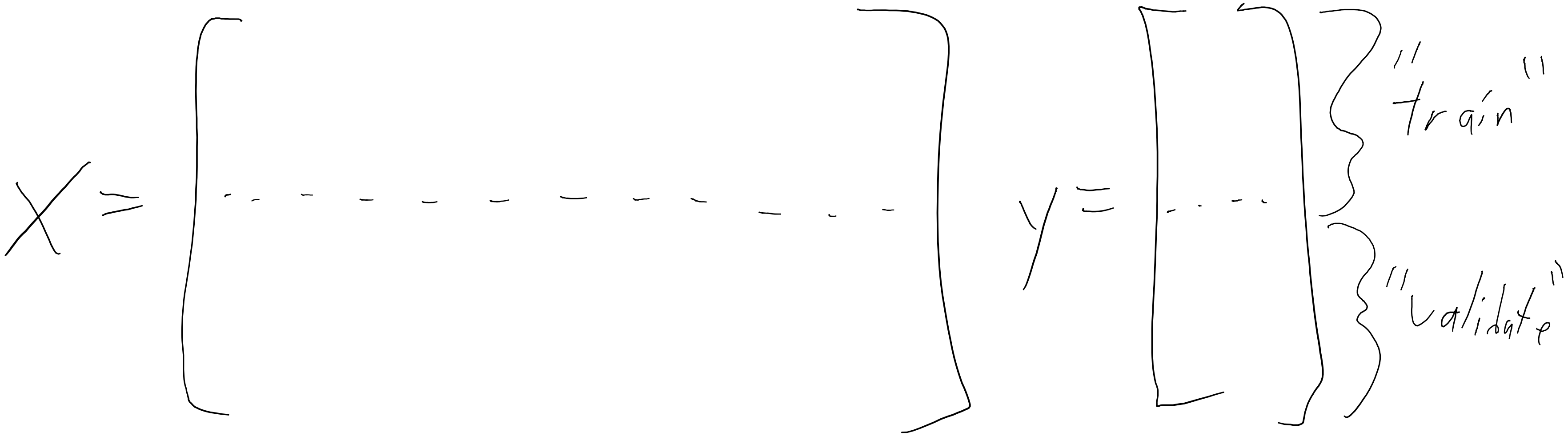
Validation Error

$$X = \begin{bmatrix} \quad \end{bmatrix} \quad y = \begin{bmatrix} \quad \end{bmatrix}$$

Validation Error

$$X = \begin{bmatrix} \\ \cdots\cdots\cdots \\ \\ \end{bmatrix} \qquad y = \begin{bmatrix} \\ \cdots \\ \\ \end{bmatrix} \begin{matrix} \left.\begin{matrix} \\ \\ \end{matrix}\right\} \text{"train"} \\ \left.\begin{matrix} \\ \\ \end{matrix}\right\} \text{"validate"} \end{matrix}$$

Validation Error



1. model = fit(X_{train}, y_{train})
2. $\hat{y}$ = predict(model, $X_{validate}$)
3. error = mean(($\hat{y} - y_{validate})^2$)

Validation Error

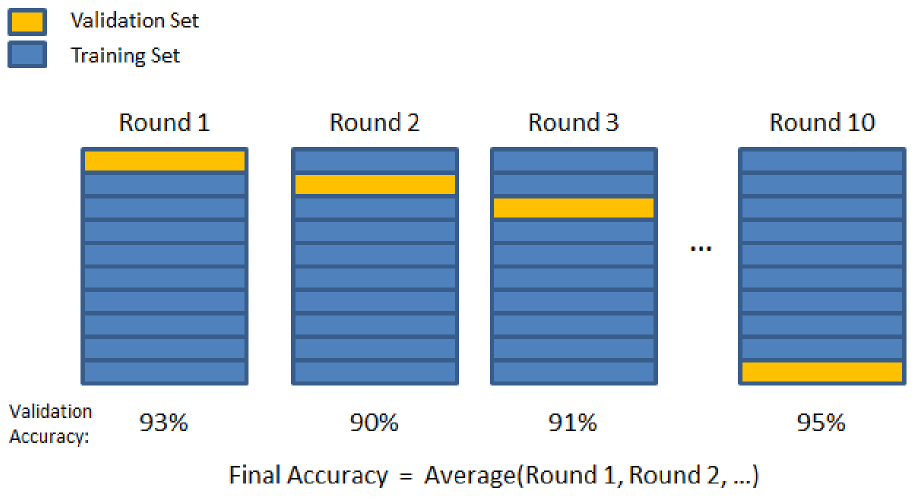
- If training data is IID, validation set is gives IID samples from test set:
  – Unbiased test error approximation.
- But in practice we evaluate the validation error multiple times:

```
for degree = 0:D
    model = fit(X, y, degree)
    ŷ = model.predict(Xval)
    error(degree) = mean((ŷ - yval)^2)
degree = argmin(error)
now fix degree and train on full dataset.
```

- In this setting, it is no longer unbiased:
  – We have violated the golden rule, so we can overfit.
  – However, often a reasonable approximation if you only evaluate it few times.

Cross-Validation

- Is it wasteful to only use part of your data to select degree?
  - Yes, standard alternative is cross-validation:
- 10-fold cross-validation:
  - Randomly split your data into 10 sets.
  - Train on 9/10 sets, and validate on the remaining set.
  - Repeat this for all 10 sets and average the score:



https://chrisjmccormick.wordpress.com/2013/07/31/k-fold-cross-validation-with-matlab-code/

Cross-Validation Theory

- Cross-validation uses more of the data to estimate train/test error.
- Does CV give unbiased estimate of test error?
  - Yes: each data point is only used once in validation.
  - But again, assuming you only compute CV score once.
- What about variance of CV?
  - Hard to characterize.
    - Variance of CV on ‘n’ examples is worse than variance if with‘n’ new examples.
    - But we believe it is close.

Summary

- Supervised learning: using data to learn input:output map.
- Least squares: classic approach to linear regression.
- Nonlinear bases can be used to relax linearity assumption.
- Test error is what we want to optimize in machine learning.
- Golden rule: you can’t use test data during training!
- Fundamental trade-off: Complex models improve training error, but training error is a worse approximation of test error.
- Validation and cross-validation: practical approximations to test error.

- Next session: dealing with e-mail spam.