CPSC 540: Machine Learning

Nonlinear Bases, Training/Testing, Regularization

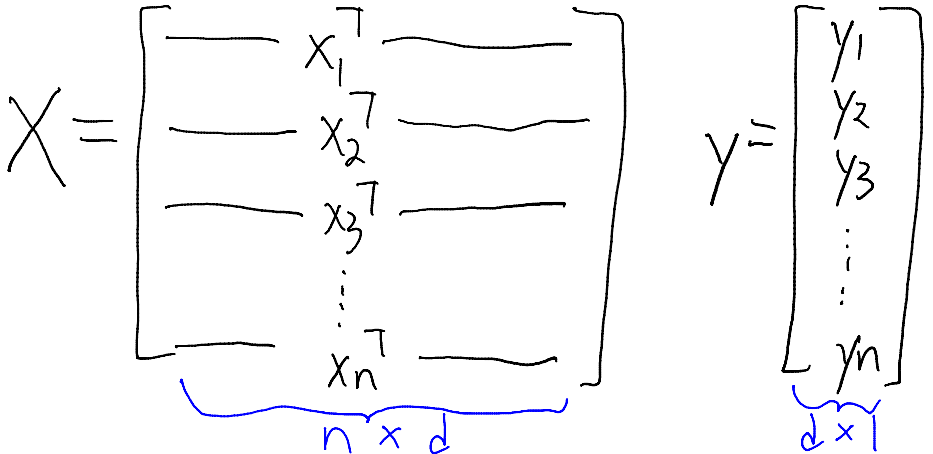
Winter 2016

Admin

- Webpage: https://www.cs.ubc.ca/~schmidtm/Courses/540-W16
- Room: Search for new room is in progress.
- Tutorials: Fridays from 3-4 and 4-5 in DMP 101, starting tomorrow.
- E-mail: I do not answer it very often, use Piazza instead.
- Auditing/enrollment forms:
  - Drop-off/pickup your forms at the end of class.
  - For enrollment, I need your prerequisite forms.
- CPSC and EECE grads:
  - submit your prereq form in class by January 14th.
- Assignment 1:
  - Posted, due January 19. Start early!

Last Time: Supervised Learning

- We discussed supervised learning:
  - We have a set of inputs x_i and a corresponding output y_i.
  - Food allergy example:
    - x_i is the quantities of food we ate on day 'i'.
    - y_i is the level of IgE we measure on day 'i'.
  - The goal is to learn a function 'f' such that $(f(x_i) - y_i)$ is small.
- We introduced standard notation for supervised learning:



Last Time: Linear Regression and Least Squares

- We considered the special case of linear regression:

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

prediction for example 'i'.

regression weights

linear combination of x_i.

- To fit this model, a classic approach is least squares:

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

- Which we can write in matrix notation as:

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

Least Squares Solution – Part 1

- Our least squares problem is:

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

(½ doesn't change minimizer)

- We'll expand:

$\|z\|^2 = z^T z$

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

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

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

Let $X^T y = v$, then use $w^T v = v^T w$.

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

$$= \frac{1}{2} w^T X^T Xw - w^T X^T y + \frac{1}{2} y^T y$$

Least Squares Solution – Part 2

- So our objective function can be written:

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

quadratic: $w^T A w$ ← (symmetric)

linear: $\rightarrow w^T b$

constant

- Using our two tedious matrix calculus exercises from last time:

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

- 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 $(X^T X)^{-1}$:

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

$A^{-1}A = I$

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

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

Least Squares Solution – Part 3

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

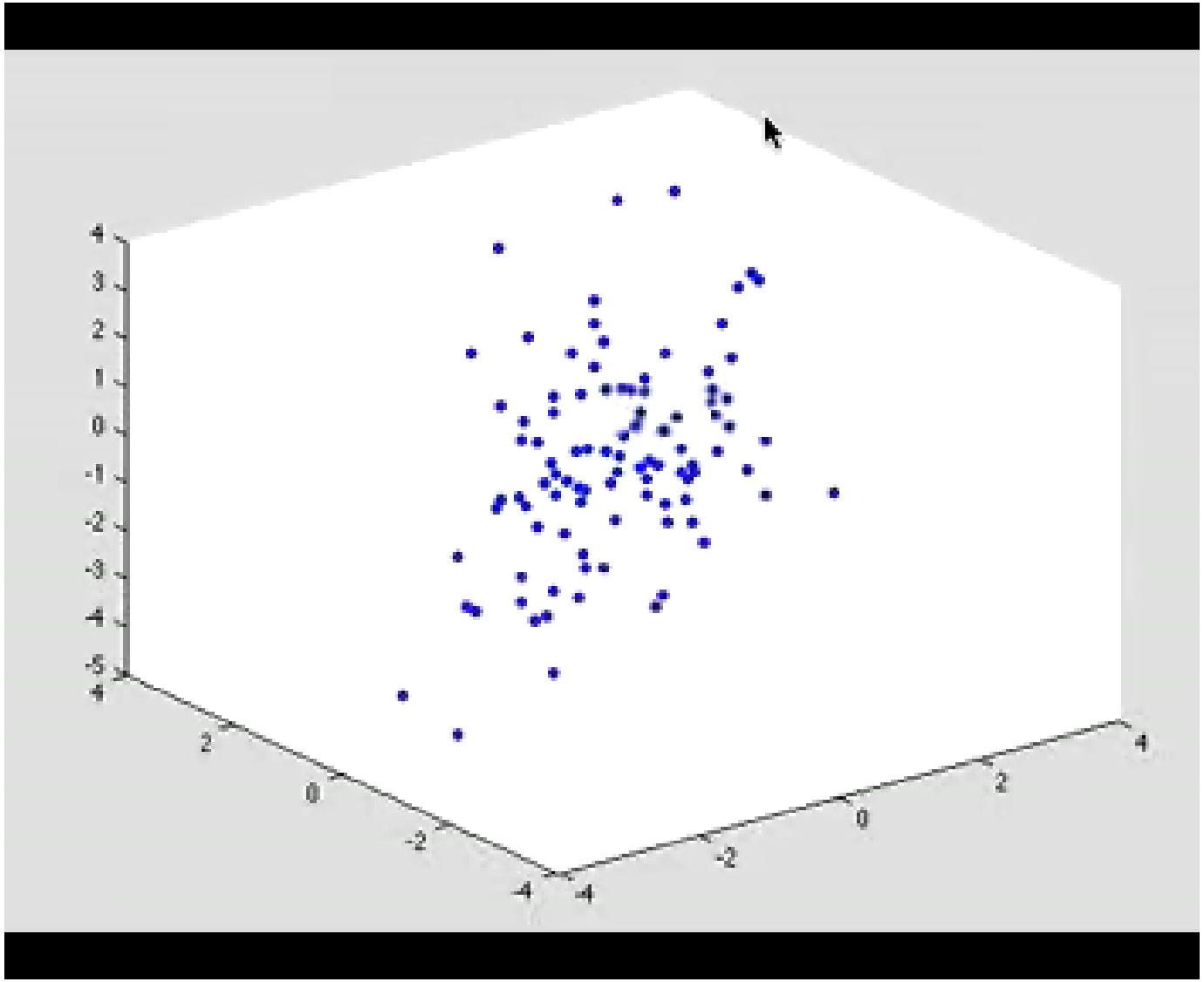
$$X^T X w = X^T y$$

In Matlab: $w = (X' * X) \backslash (X' * y)$ (x = A \ b; "solve $Ax = b$")

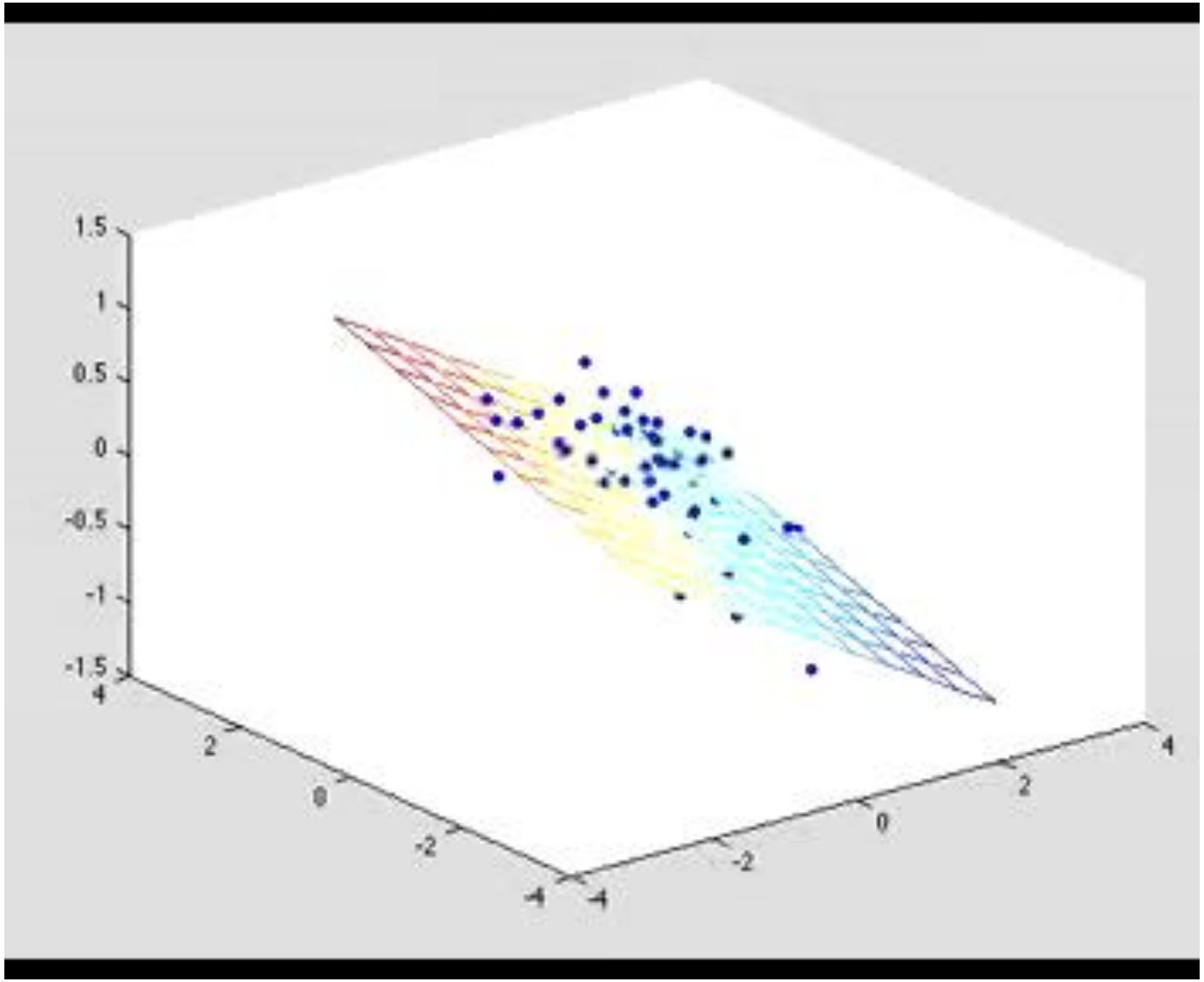
- What is the cost of computing this?
  1. Forming $X^T y$ costs $O(nd)$.
  2. Forming $X^T X$ costs $O(nd^2)$.
  3. Solving a 'd' by 'd' linear system costs $O(d^3)$.
     - If we use LU decomposition (AKA Gaussian elimination).

  – 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

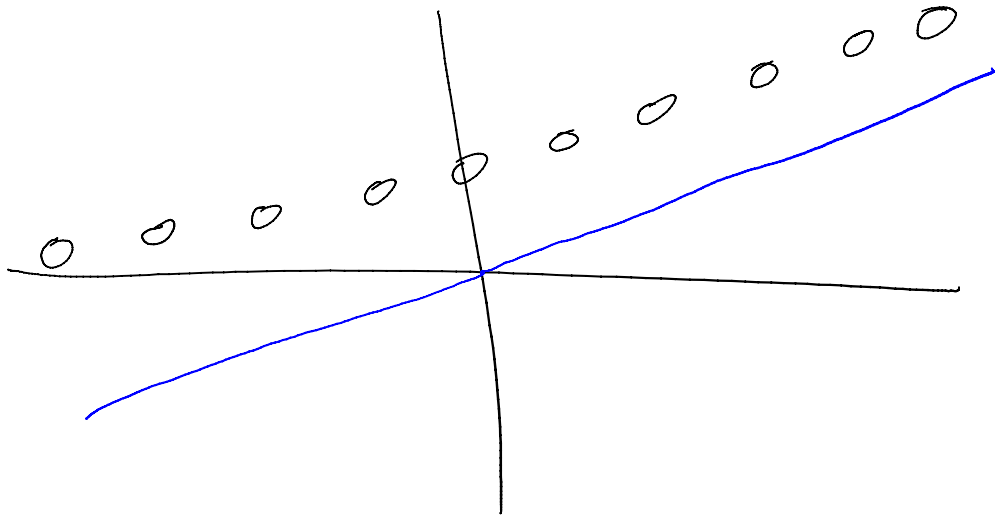


Problem with Linear Least Squares

- 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'll spend first few lectures fixing these issues…

First Problem: y-intercept

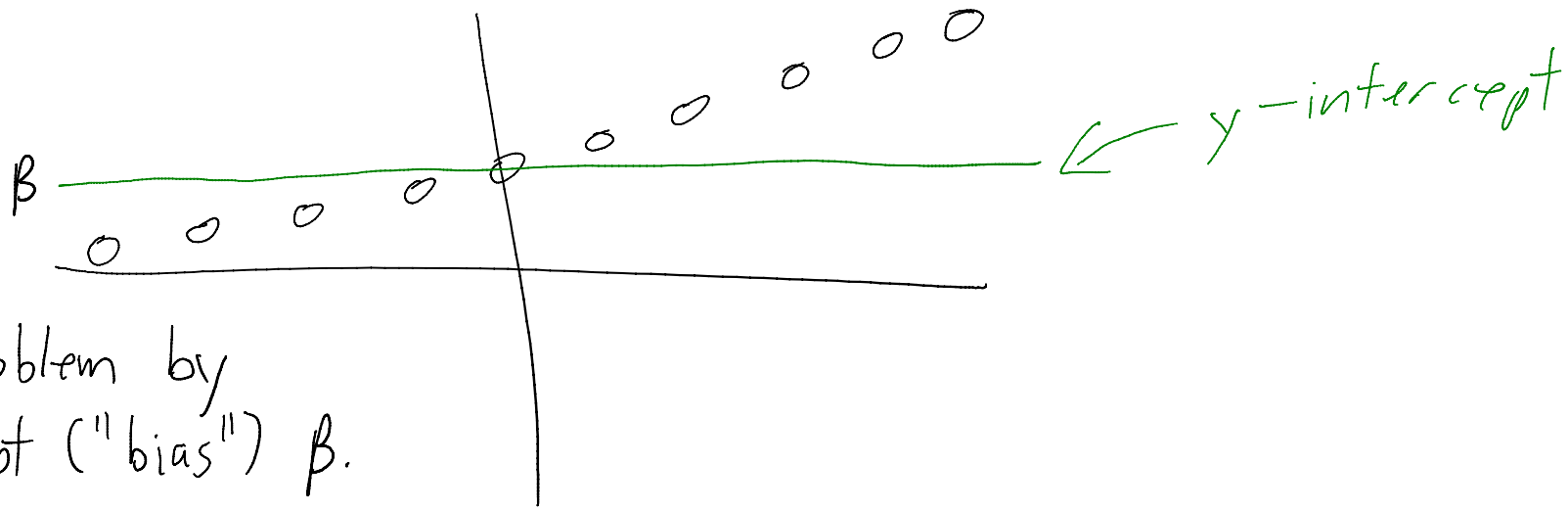
Since we predict $\hat{y}_i = w^T x_i$, we must predict $y_i = 0$ if $x_i = 0$.



"least squares" prediction must go through origin.

First Problem: y-intercept

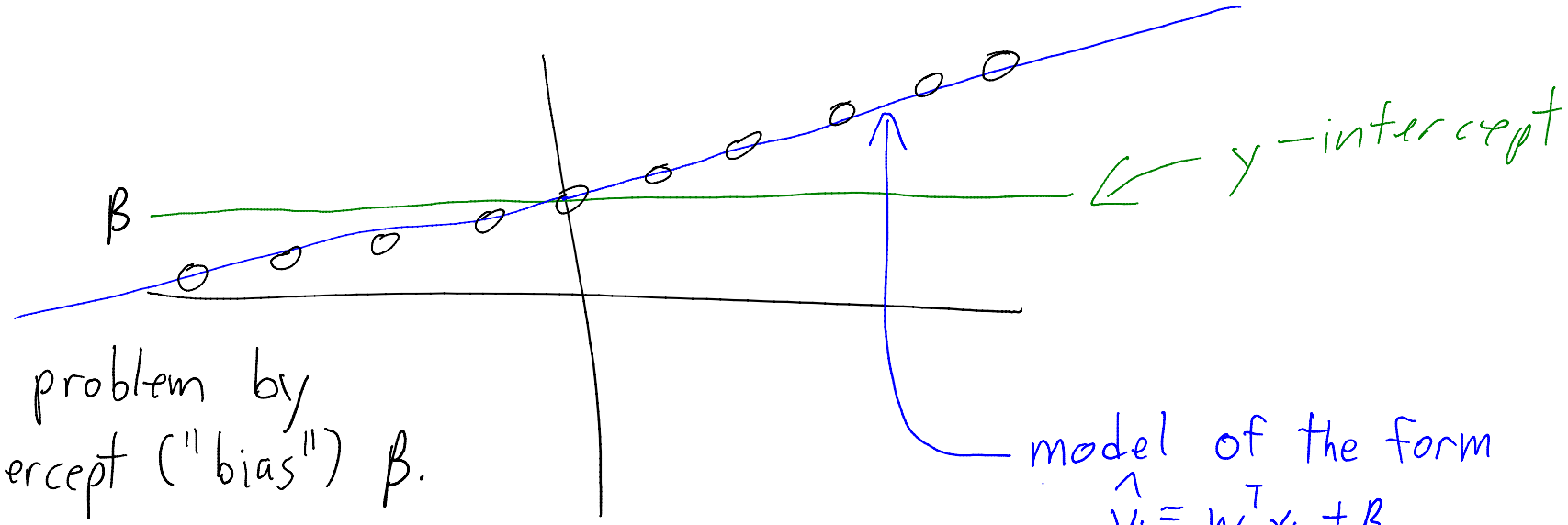
Since we predict $\hat{y}_i = w^T x_i$, we must predict $y_i = 0$ if $x_i = 0$.



We can fix 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 $x_i = 0$.



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:

$$\bar{w} = (\bar{X}^T \bar{X})^{-1} (\bar{X}^T y), \qquad \bar{w} = \begin{bmatrix} \beta \\ w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} \} w$$

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

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):

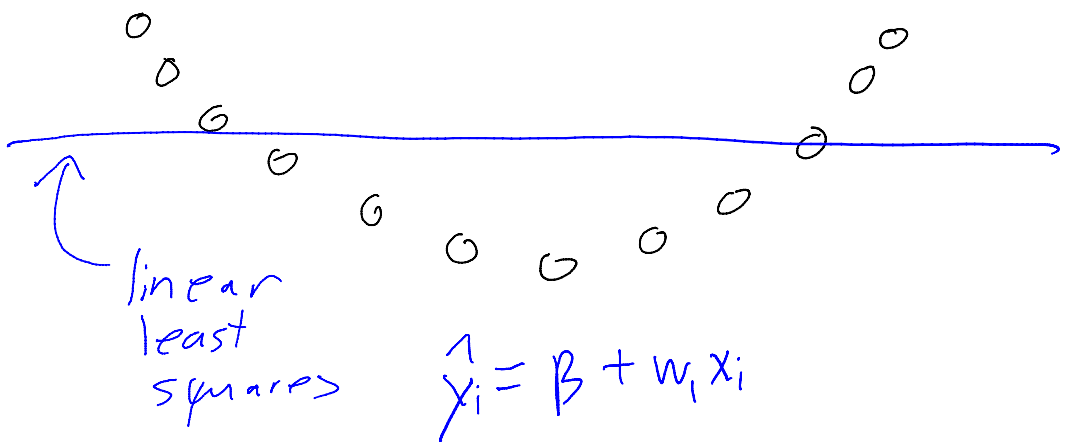
$$X = \begin{bmatrix} 0.2 \\ -0.5 \\ 1 \\ 4 \end{bmatrix} \qquad X_{poly} = \begin{bmatrix} 1 & 0.2 & (0.2)^2 \\ 1 & -0.5 & (-0.5)^2 \\ 1 & 1 & (1)^2 \\ 1 & 4 & (4)^2 \end{bmatrix}$$

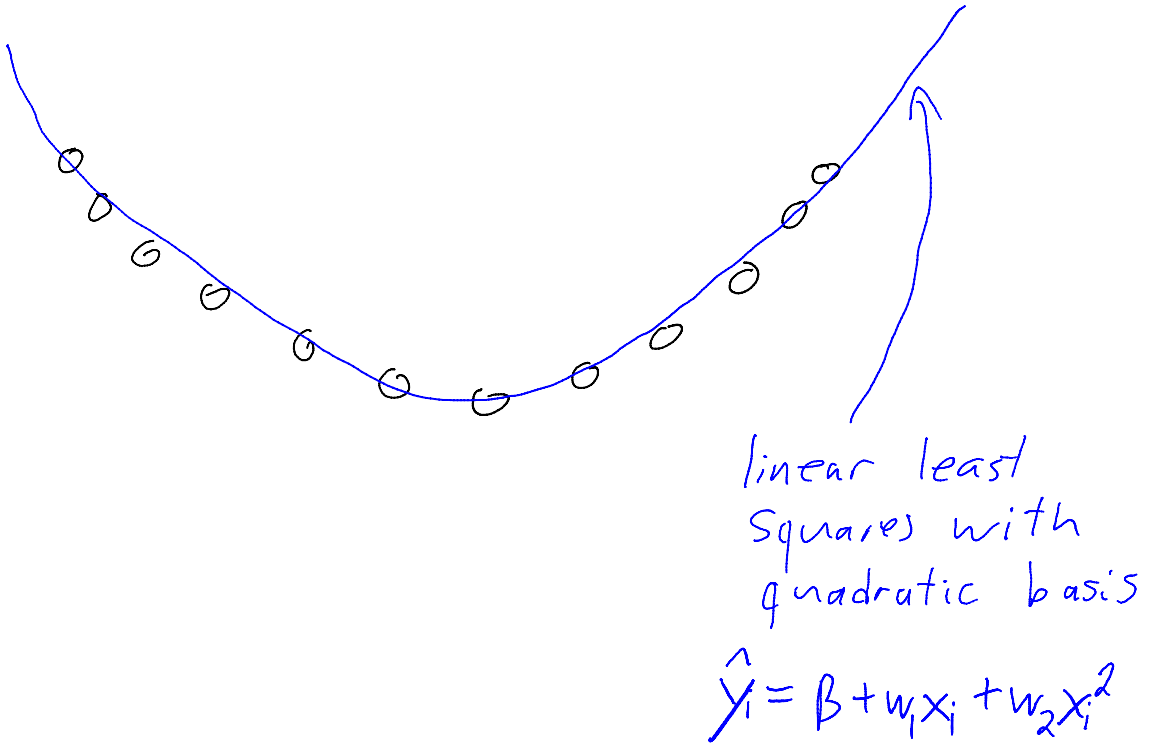
- Now fit least squares with this matrix:

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

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

Change of Basis

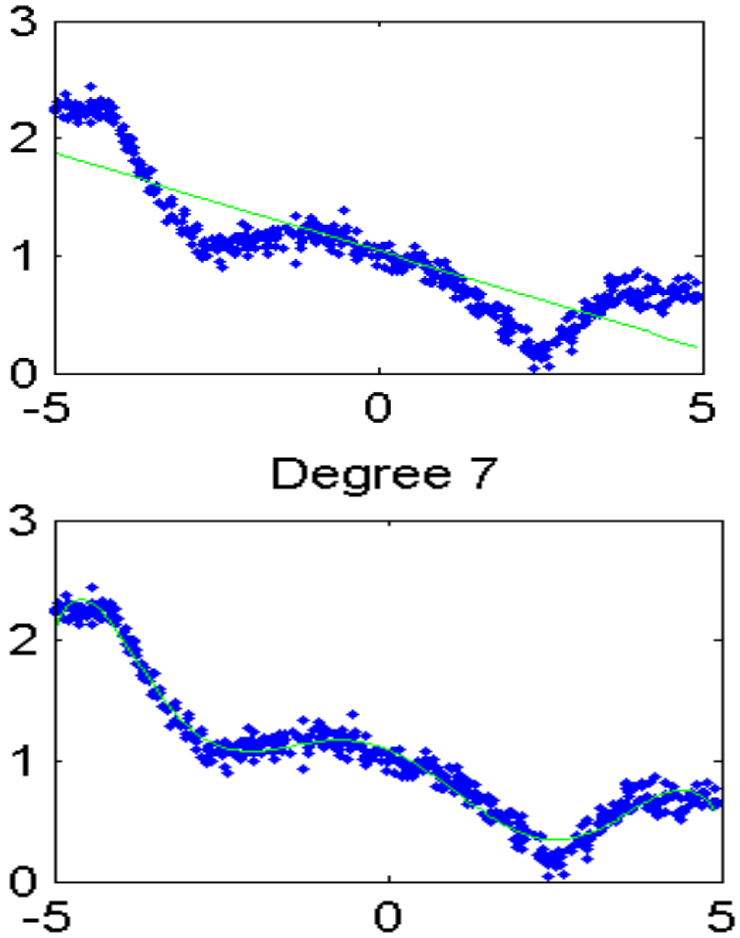




General Polynomial Basis

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

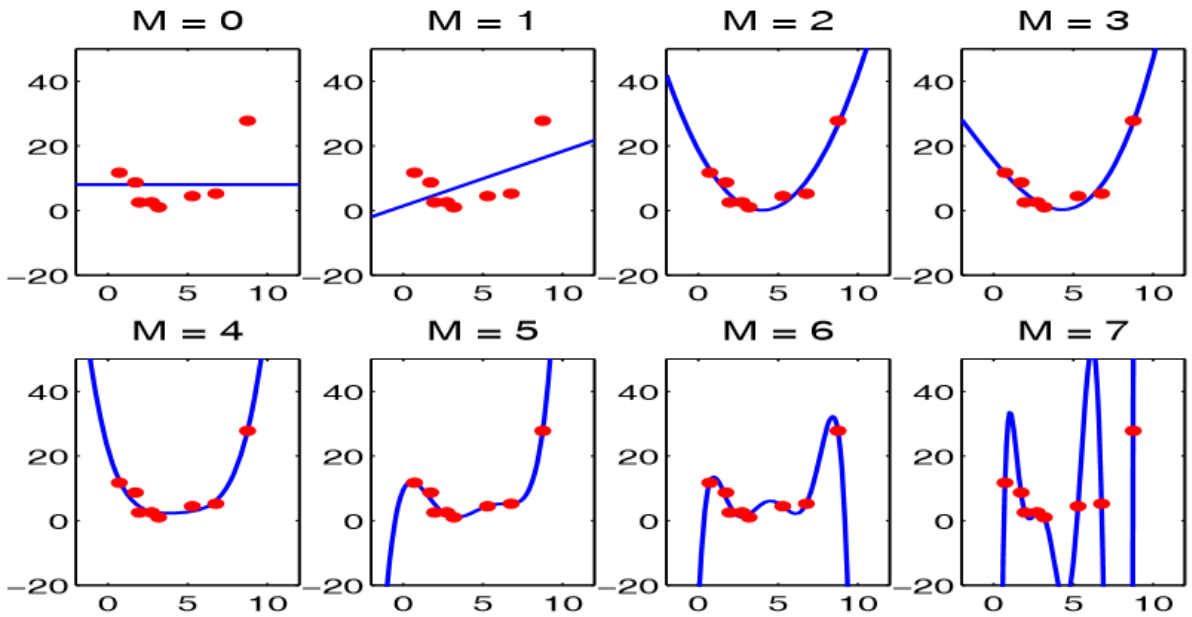
$$X_{poly} = \begin{bmatrix} 1 & x_1 & (x_1)^2 & \cdots & (x_1)^d \\ 1 & x_2 & (x_2)^2 & \cdots & (x_2)^d \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n & (x_n)^2 & \cdots & (x_n)^d \end{bmatrix}$$



- 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 =

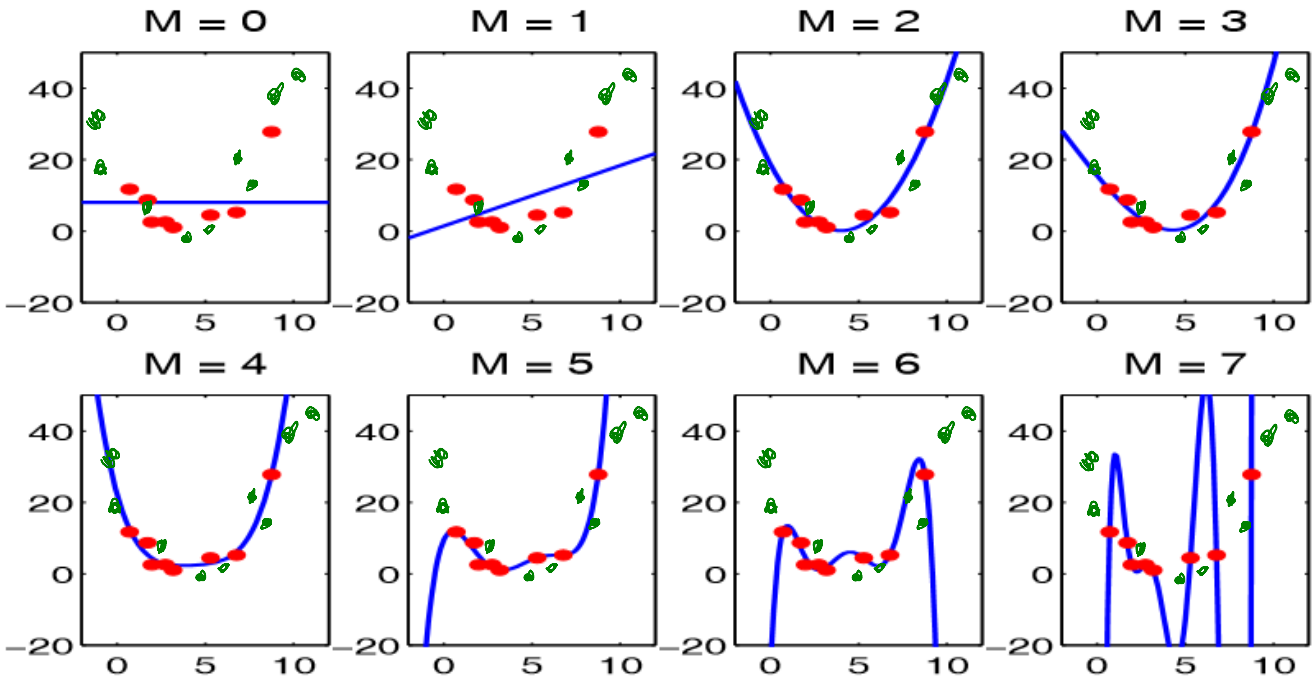
Sick?
?
?

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 company Baidu announced that its image recognition software had inched ahead of Google's on a standardized

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.

Bias-Variance Decomposition

- Analysis of expected test error of any learning algorithm:

Assume $y_i = f(x_i) + \varepsilon$, for some function 'f' and random error ε with a mean of 0 and a variance of σ^2.

Assume we have a "learner" that can take a training set $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ and use these to make predictions $\hat{f}(x_i)$.

Then for a new example (x_i, y_i) the error averaged over training sets is

$$E[(y_i - \hat{f}(x_i))^2] = \text{Bias}[\hat{f}(x_i)]^2 + \text{Var}[\hat{f}(x_i)] + \sigma^2$$

"Irreducible error": best we can hope for given the noise level.

Expected error due to having wrong model.

Where $\text{Bias}[\hat{f}(x_i)] = E[\hat{f}(x_i)] - f(x_i)$,

How sensitive is the model to the particular training set?

$$\text{Var}[\hat{f}(x_i)] = E[(\hat{f}(x_i) - E[\hat{f}(x_i)])^2]$$

Discussion of Bias-Variance Decomposition

- Polynomial basis with high degree:
  - Very likely to fit data well, so bias is low.
  - But model changes a lot if you change the data, so variance is high.
- Polynomial basis with low degree:
  - Less likely to fit data well, so bias is high.
  - But model doesn't change much you change data, so variance is low.
- And degree does not affect irreducible error.
- Bias-variance is a bit weird:
  - Considers expectation over possible training set.
  - But doesn't say anything about test error with *your* training set.
- There are other ways to estimate test error:
  - VC dimension bounds test error based on training error and model complexity.
  - Learning theory is the last (planned) topic for this course.

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.

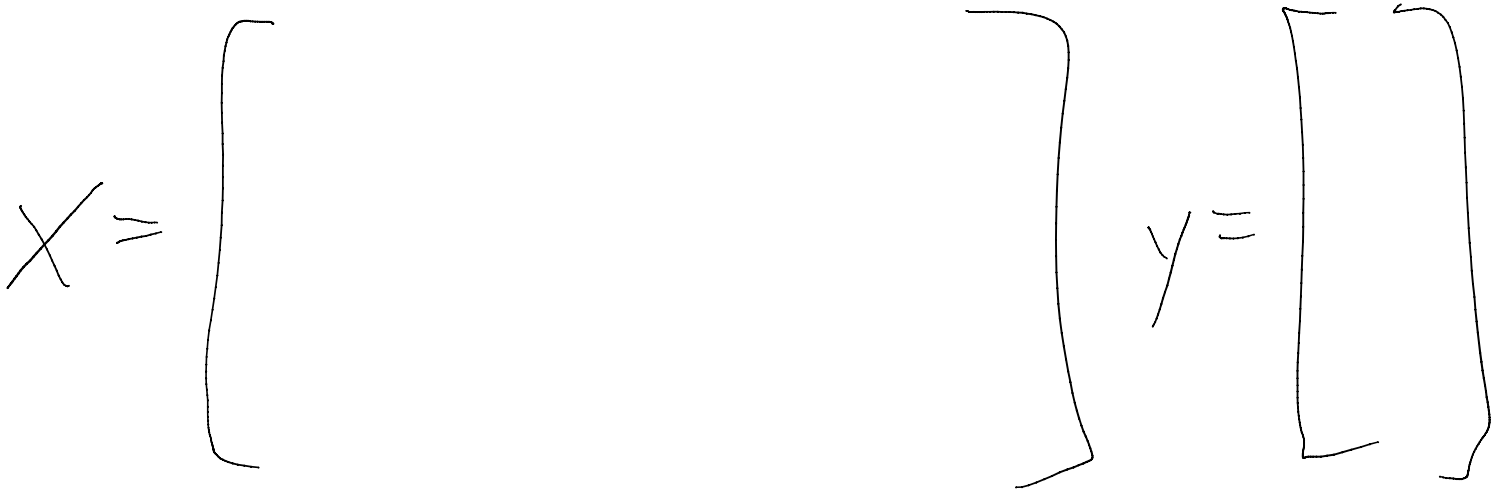
- Different models make different trade-offs.
- Simple models (low-degree polynomials):
  - Training error is good approximation of test error:
    - Not very sensitive to the particular training set you have.
  - But don't fit training data well.
- Complex models (high-degree polynomials):
  - Fit training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

Back to reality…

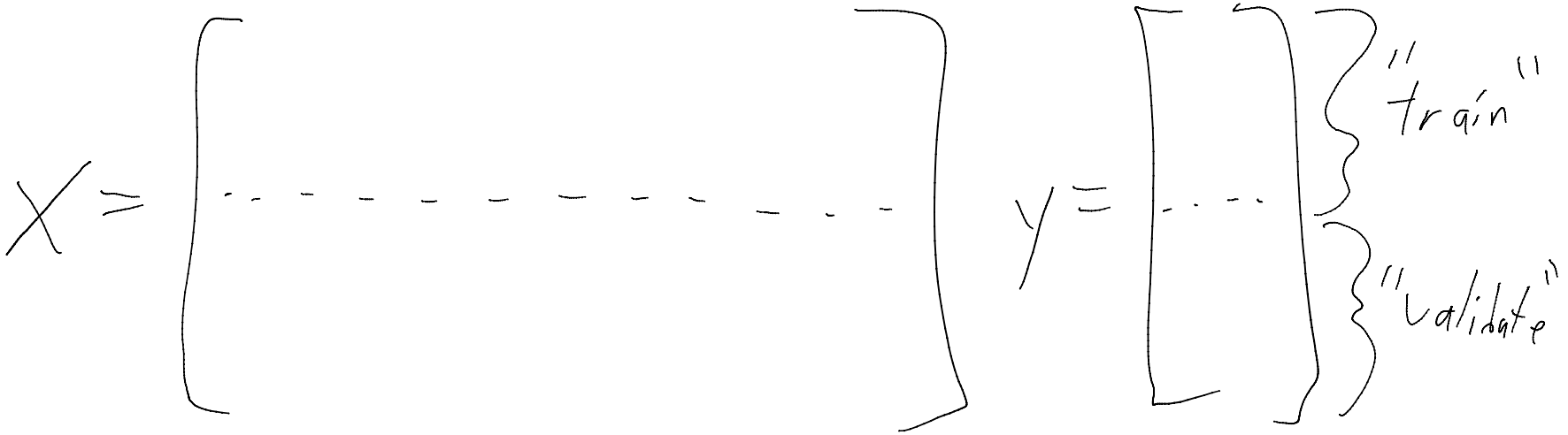
- How do we decide polynomial degree in practice?
- We care about the test error.
- But we can't look at the test data.
- So what do we do?????

- 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.

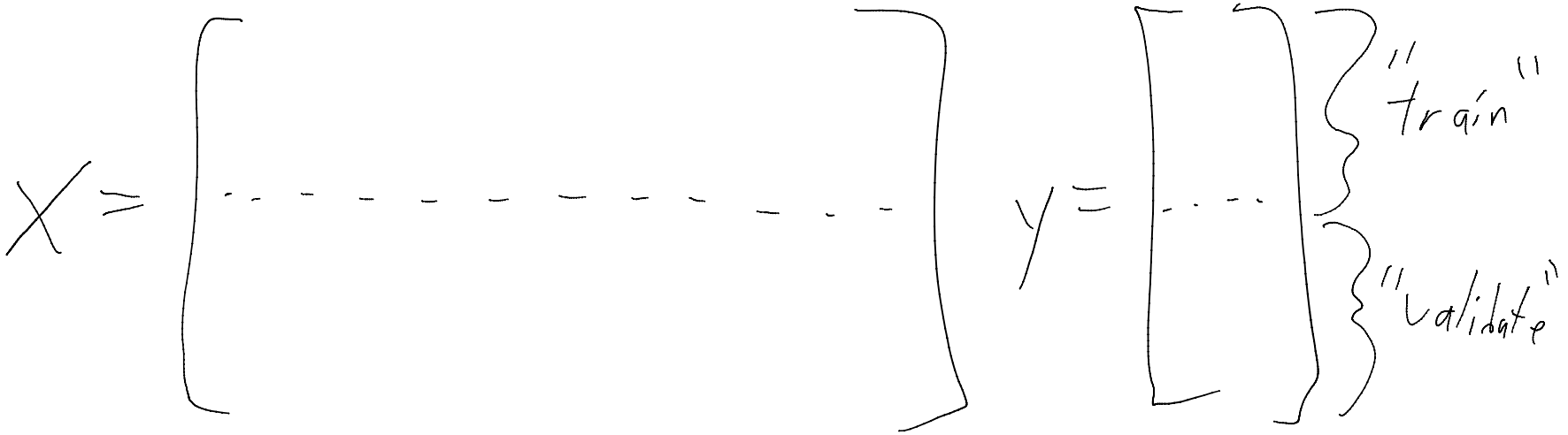
Validation Error



Validation Error



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

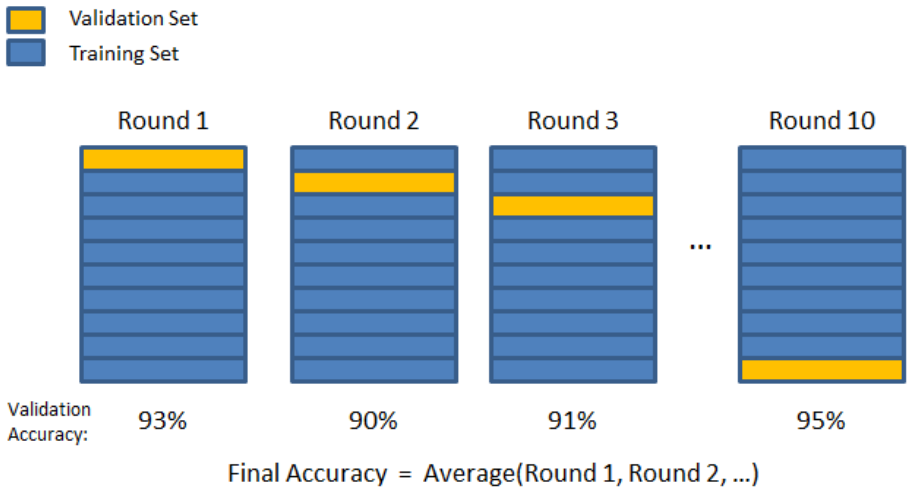
- If training data is IID, validation set is gives IID samples from test set:
  - Unbiased test error approximation.
- But in practice we evaluate the validation error multiple times:

```
for degree = 0:D
    model = fit(X, y, degree)
    ŷ = model.predict(X_val)
    error(degree) = mean((ŷ − y_val)^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.

Controlling Complexity

- We know that complex models can overfit.
- But usually the "true" mapping from x_i to y_i is complex.
- So what do we do???

- There are many possible answers:
  - Model averaging: average over multiple models to decrease variance.
  - Regularization: add a penalty on the complexity of the model.

L2-Regularization

- Our standard least squares formulation:

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

- Standard regularization strategy is to add a penalty on the L2-norm:

$$\underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \ \frac{1}{2}\|Xw - y\|^2 + \frac{\lambda}{2}\|w\|^2$$

- Regularization parameter λ controls 'strength' of regularization:
  - If λ is large then it forces 'w' to be very small: low complexity.
  - If λ is tiny then 'w' can be get huge: high complexity.
- Has been re-invented several times:
  - Tikhonov regularization, ridge regression, etc.

L2-Regularization

- In terms of bias-variance:
  - Regularization increases bias: training error will be higher.
  - Regularization decreases variance: training error better approximates test error.
- How should you choose λ?
  - Theory: as 'n' grows λ should be in the range O(1) to $O(n^{-1/2})$.
  - Practice: optimize validation set or cross-validation error.
    - This almost always decreases the test error.
- How do you compute 'w'?

Ridge Regression Calculation

Objective: $f(w) = \frac{1}{2}(y - Xw)^T(y - Xw) + \frac{\lambda}{2} w^T w.$

Gradient: $\nabla f(w) = X^T X w - X^T y + \lambda w$

Setting to zero: $X^T X w + \lambda w = X^T y$, or

$(X^T X + \lambda I) w = X^T y.$ → using $Iw = w$

Pre-multiply by $(X^T X + \lambda I)^{-1}$, which always exists:

$$\boxed{w = (X^T X + \lambda I)^{-1} X^T y.}$$

In Matlab:

```
w = (X'*X + lambda*eye(d))^-1 (X'*y)
```

Cost: same as least squares.

Why use L2-Regularization?

- Mark says: “You should always use regularization.”
- “Almost always improves test error” should already convince you.
- But here are more reasons:
  1. Solution ‘w’ is unique.
  2. Does not require X’X to be invertible.
  3. Solution ‘w’ is less sensitive to changes in X or y.
  4. You can use Cholesky factorization instead of LU factorization.
  5. Makes large-scale methods for computing ‘w’ run faster.
  6. Stein’s paradox: if $d \geq 3$, regularization moves us closer to ‘true’ w.
  7. In the worst case you just set λ small and get the same performance.

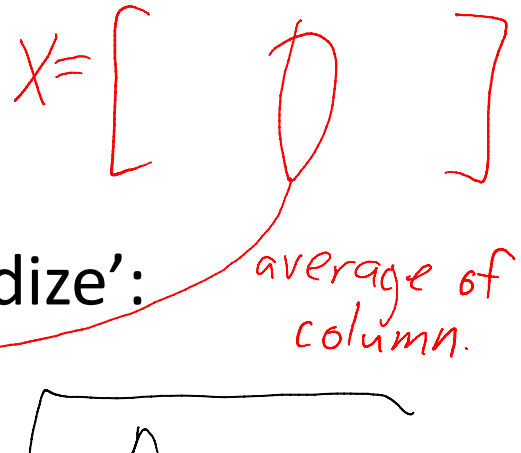
Features with Different Scales

- Consider features with different scales:

Egg (#)	Milk (mL)	Fish (g)	Pasta (cups)
0	250	0	1
1	250	200	1
0	0	0	0.5
2	250	150	0

- Should we convert to some standard 'unit'?
  - For least squares, it doesn't matter:
    - w_j*(100 mL) gives the same model as w_j*(0.1 L), w_j will just be 1000 times smaller.
  - With regularization, it matters:
    - Penalization $|w_j|$ means different things if features 'j' are on different scales.

Standardizing Features



- To put features on a similar scale, it is common to 'standardize':
  - For each feature:
    - Compute mean and standard deviation:

$$\mu_j = \frac{1}{n}\sum_{i=1}^{n} x_{ij} \qquad \sigma_j = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{ij}-\mu_j)^2}$$

    - Subtract mean and divide by standard deviation:

$$x_{ij} \leftarrow \frac{x_{ij}-\mu_j}{\sigma_j}$$

  - Means that change in 'w_j' have similar effect for any feature 'j'.
- Should we regularize the bias?
  - No! We don't want to penalize a global shift up or down.
  - Yes! Regularizing all variables makes it easier to compute 'w'.
  - Compromise: regularize the bias by a smaller amount than other variables.

Standardizing Target

- In regression, we also often standardize the targets y_i.
  - Puts targets on the same standard scale as standardized features:

$$y_i \leftarrow \frac{y_i - \mu_y}{\sigma_y}$$

- With standardized target, setting w = 0 predicts average y_i:
  - High regularization makes us predict closer to the average value.
- Other common transformations of y_i are logarithm/exponent:

$$y_i \leftarrow \log(y_i) \qquad y_i \leftarrow \exp(y_i)$$

  - Makes sense for geometric/exponential processes.

Summary

- Least squares solution involves solving a linear system.
- 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.
- Regularization: allows complicated models by penalizing complexity.

- Next time: dealing with e-mail spam and crazy y_i values.