CS5841/EE5841 Machine Learning

Lecture 3: Regression

Evan Lucas

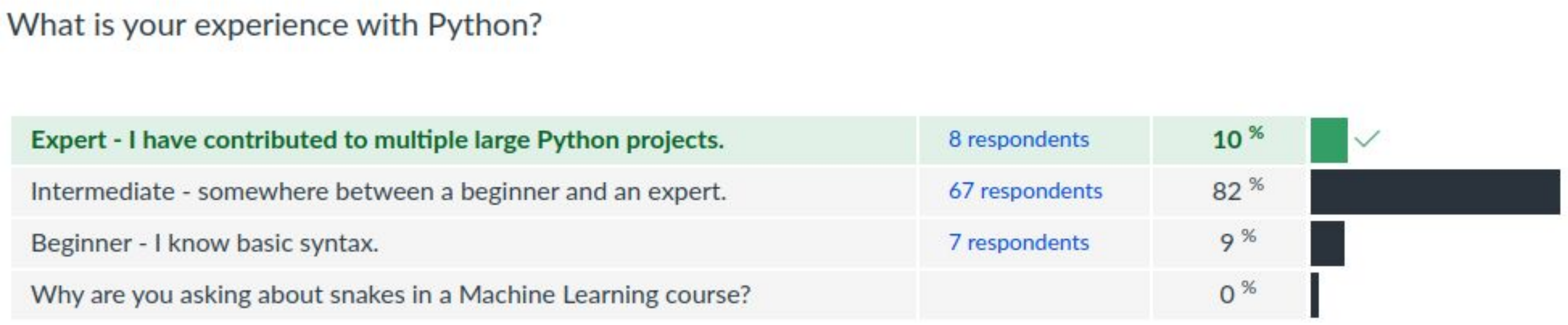
Overview

- Course updates
-

Class updates

- Julia module created
  - Extra credit quiz
  - Contains download links for Julia and Pluto
- Discord and discussion threads made

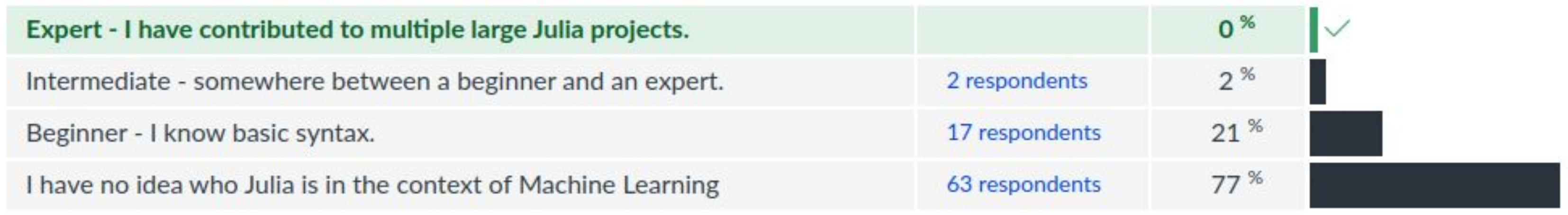
Course survey results



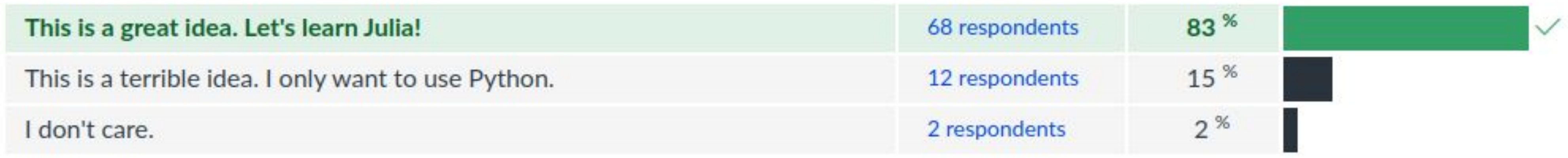
What is your experience with Python?

Expert - I have contributed to multiple large Python projects.	8 respondents	10 %	✓
Intermediate - somewhere between a beginner and an expert.	67 respondents	82 %	
Beginner - I know basic syntax.	7 respondents	9 %	
Why are you asking about snakes in a Machine Learning course?		0 %	

Course survey results

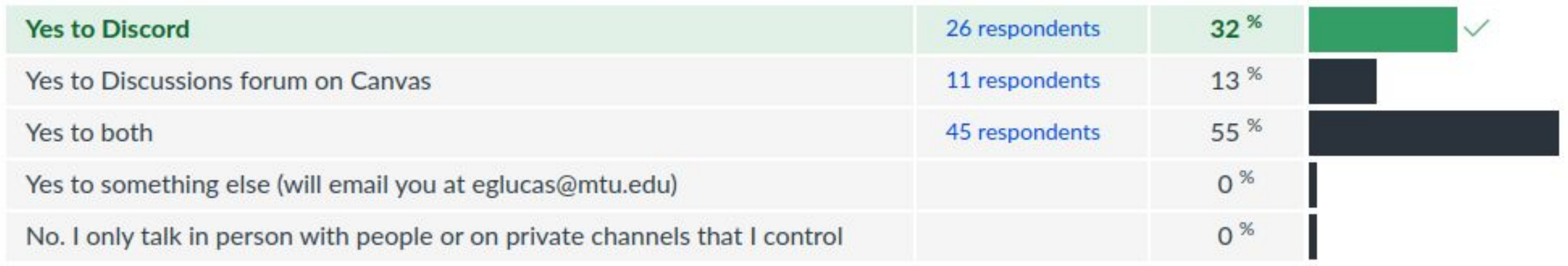


Expert - I have contributed to multiple large Julia projects.		**0 %**	✓
Intermediate - somewhere between a beginner and an expert.	2 respondents	2 %	
Beginner - I know basic syntax.	17 respondents	21 %	
I have no idea who Julia is in the context of Machine Learning	63 respondents	77 %	



This is a great idea. Let's learn Julia!	68 respondents	**83 %**	✓
This is a terrible idea. I only want to use Python.	12 respondents	15 %	
I don't care.	2 respondents	2 %	

Course survey results



Yes to Discord	26 respondents	32 %	✓
Yes to Discussions forum on Canvas	11 respondents	13 %	
Yes to both	45 respondents	55 %	
Yes to something else (will email you at eglucas@mtu.edu)		0 %	
No. I only talk in person with people or on private channels that I control		0 %	

Bonus topic question

- Lots of interest in:
  - Ethics
  - Large language models
  - NLP in general
  - Object detection
- Lecture plan to come soon.
  - Do you prefer a high level survey of a few topics or one deeper topic?
  - Do you want an extra credit assignment?
- A couple answers were generated with ChatGPT or similar…
  - Please don’t do that unless you attribute it

Related reading

- Strongly suggested
    - Bishop Chapter 3
- Additional
    - Chapter 11 in Murphy
    - Chapter 3 in ESLII

Where does linear regression fit in the world of ML?

- Supervised learning
  - We know the answer and are training the model to predict it
- Regression
  - We are predicting a numerical value

Supervised learning formal definition

- Labeled datasets are used to train an algorithm to predict an outcome
- Given

Regression vs. Classification

- Regression
  - We are trying to predict a continuous value
- Classification
  - We are trying to predict a discrete value

Why linear regression?

- Simple models are useful
  - Always good to benchmark with a simple model!
  - Simple models are good placeholders in system backend prototyping
- Interpretable!
- A component of neural networks!
  - Called a linear, dense, or fully connected layer
- Introduction of concepts without a complicated model
  - Loss metrics
  - Probabilistic model training (gradient descent family)
  - Regularization

Linear regression

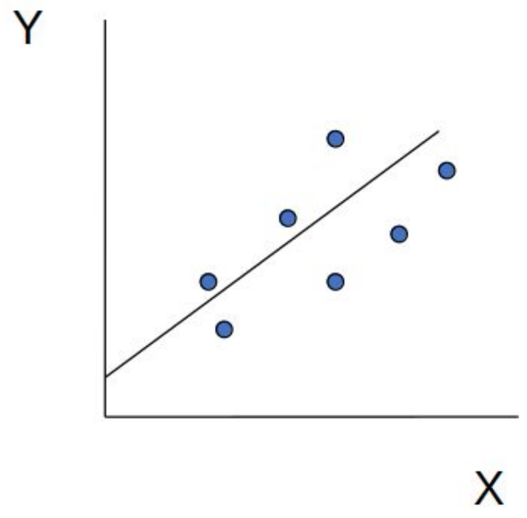
- $y = m*x + b$
  - Given two points, can find an exact solution for a line
  - How do we handle multiple data points?
- $\{y\} = [w][x]$
  - Matrix definition
  - x_0 is 1

Some notes on notation

- Instead of algebraic standard form (*x^2+x+...*), we typically list things in ascending order
  - ie: $w_0, w_1, \ldots, w_n$
  - This sets up the bias term as being the first column (or row in some references)

Solving linear regression

- Minimize difference between estimate and true value by adjusting weights
  - $\text{argmin}_w (y-y_{est})^2$
  - $\text{argmin}_w (y-wx)^2$
- Why squared error?
  - Always positive
  - Easy to differentiate



Solving linear regression

$$\arg\min_w \sum_w (y_i - wx_i)^2$$

$$\frac{\delta}{\delta w} \sum_i (y_i - wx_i)^2 = 2 \sum_i -x_i(y_i - wx_i)$$

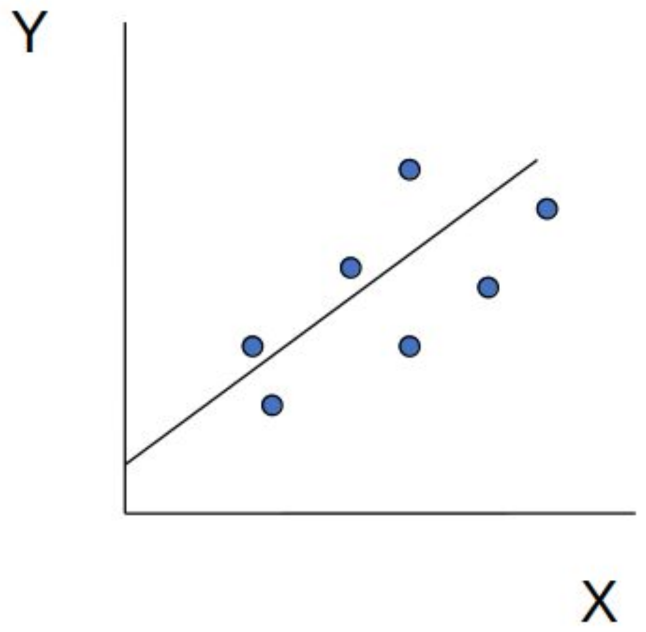
$$2 \sum_i x_i(y_i - wx_i) = 0$$

$$2 \sum_i x_i y_i - 2 \sum_i w x_i x_i = 0$$

$$w = \frac{\sum_i x_i y_i}{\sum_i x_i^2}$$

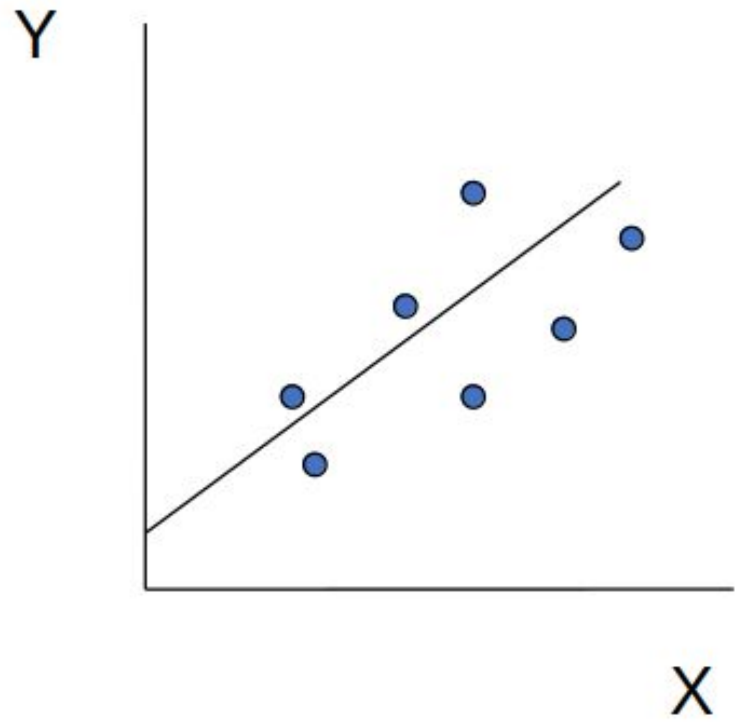
Michigan Tech

Adding a bias term



$$E = \sum_i (y_i - w_0 - w_1 x_i)^2$$

Adding a bias term



$$w_0 = \bar{y} - w_1 \bar{x}$$

$$w_1 = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sum_i (x_i - \bar{x})^2}$$

Multiple linear regression

$$y = w_0 + w_1x_1 + \ldots + w_kx_k$$

Basis functions

- We can use the concept of basis functions to map non-linear spaces into our linear model

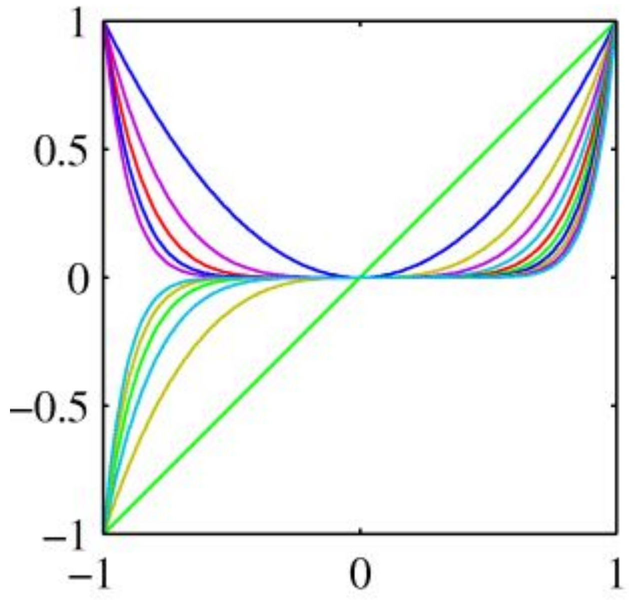
$$y(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}) = \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x})$$

- Φ_j is our basis function
  - $\Phi_d(x) = x_d$ is the linear basis function
  - $\Phi_0(x) = 1$, typically, to give us a bias term

Michigan Tech

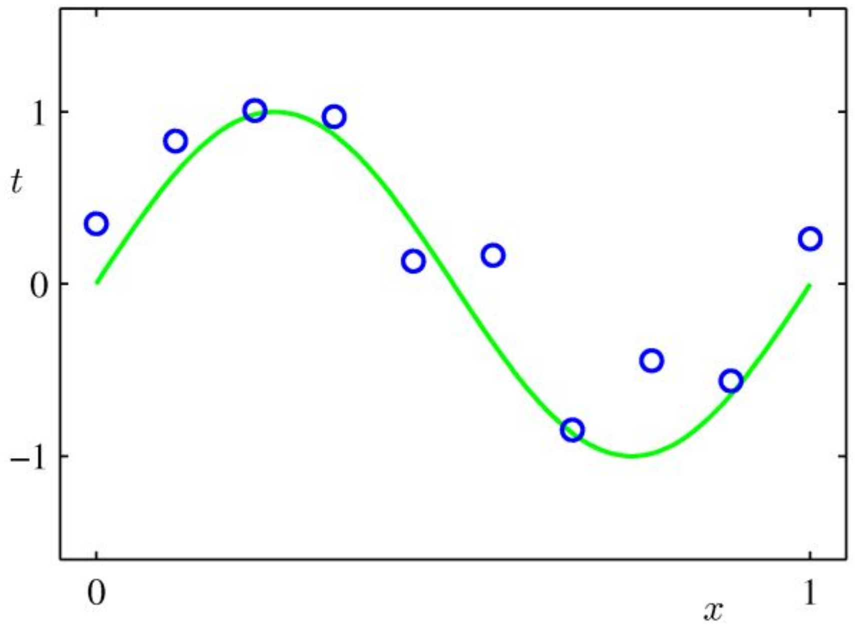
Polynomial basis functions

- Global!
    - Small change in **x** affects all basis functions



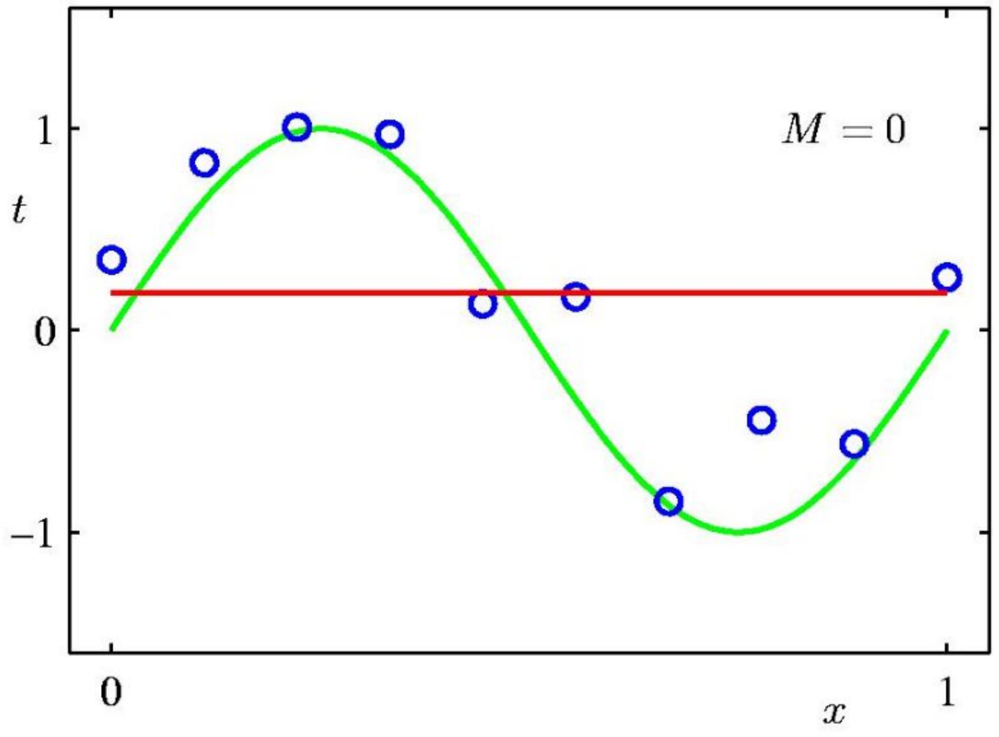
$$\phi_j(x) = x^j$$

Example of polynomial curve fitting

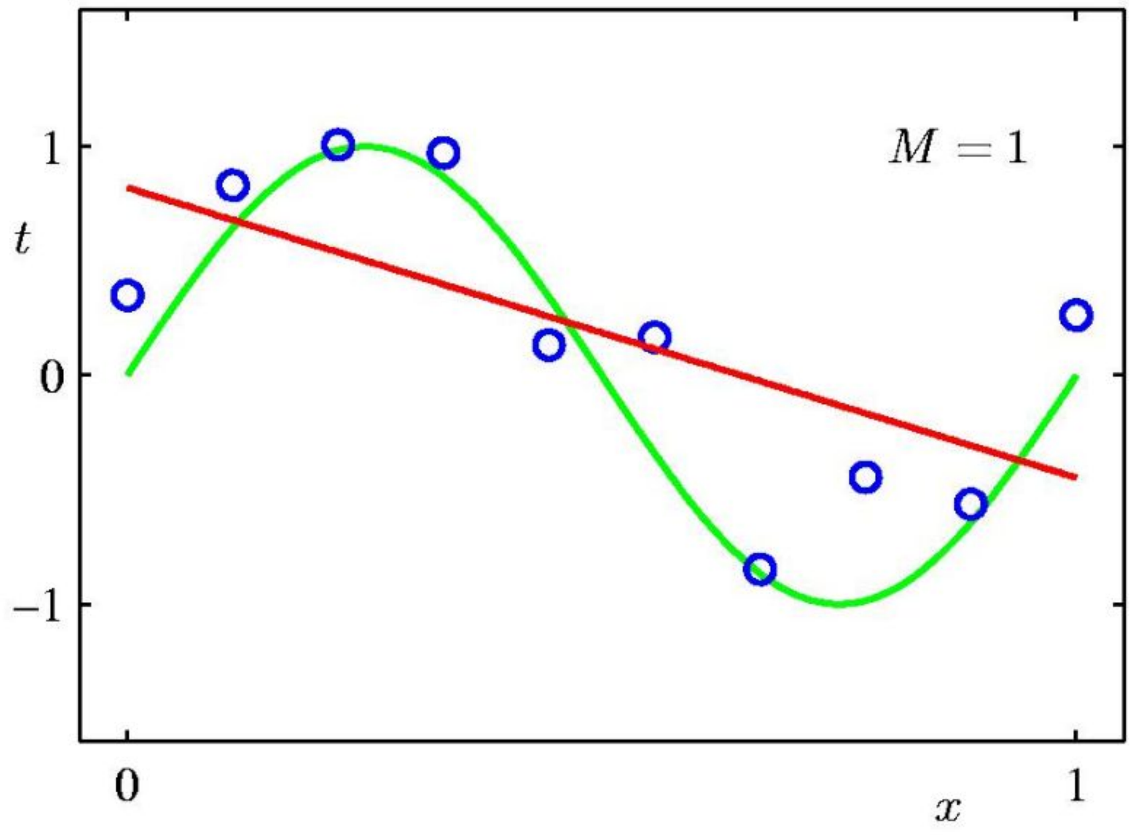


$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

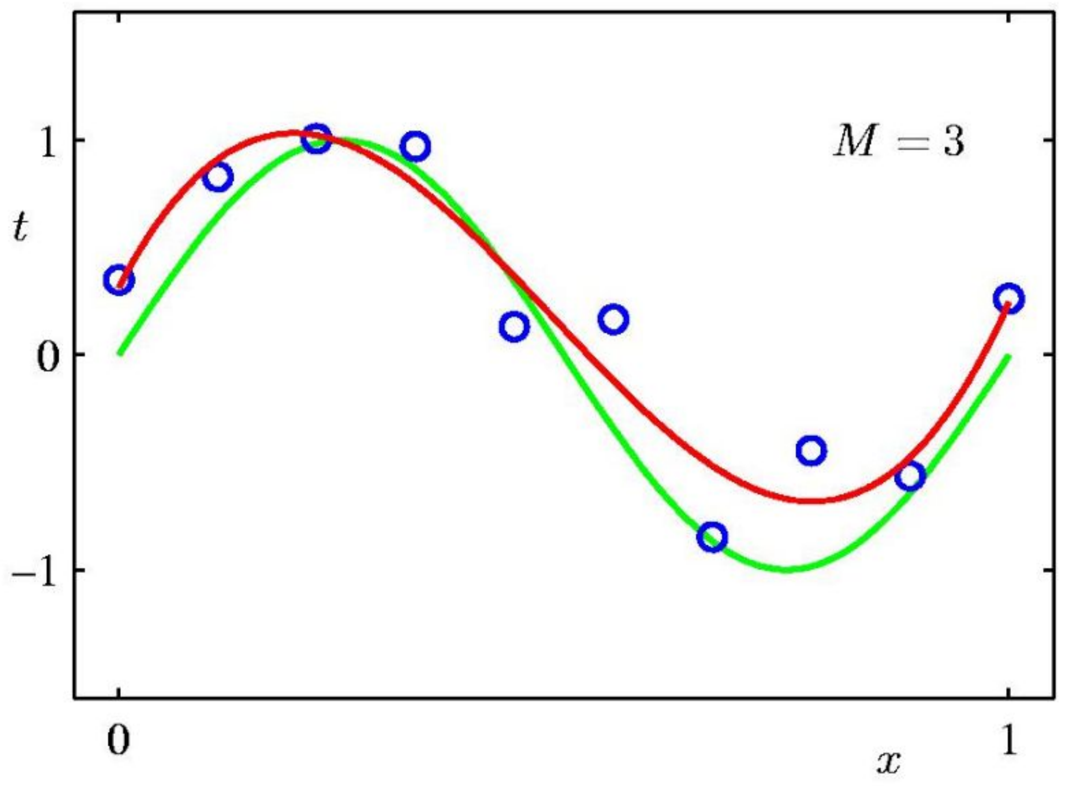
0th Order Polynomial



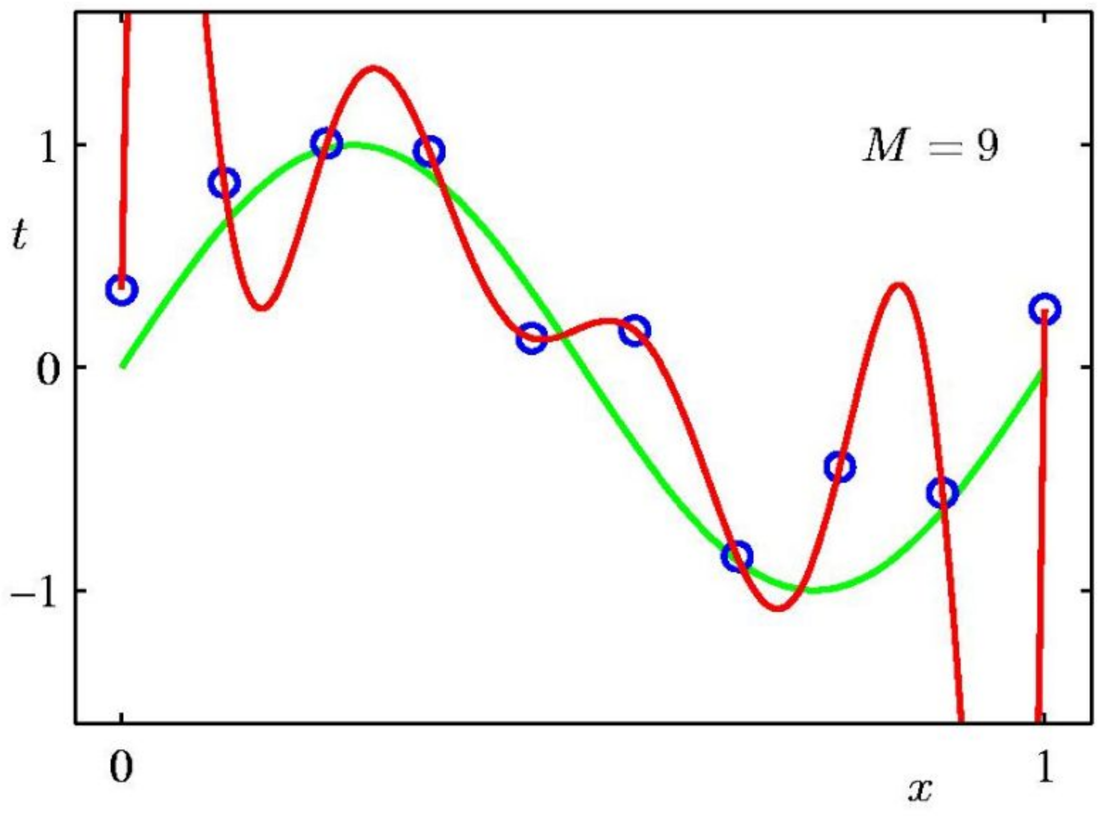
1st Order Polynomial



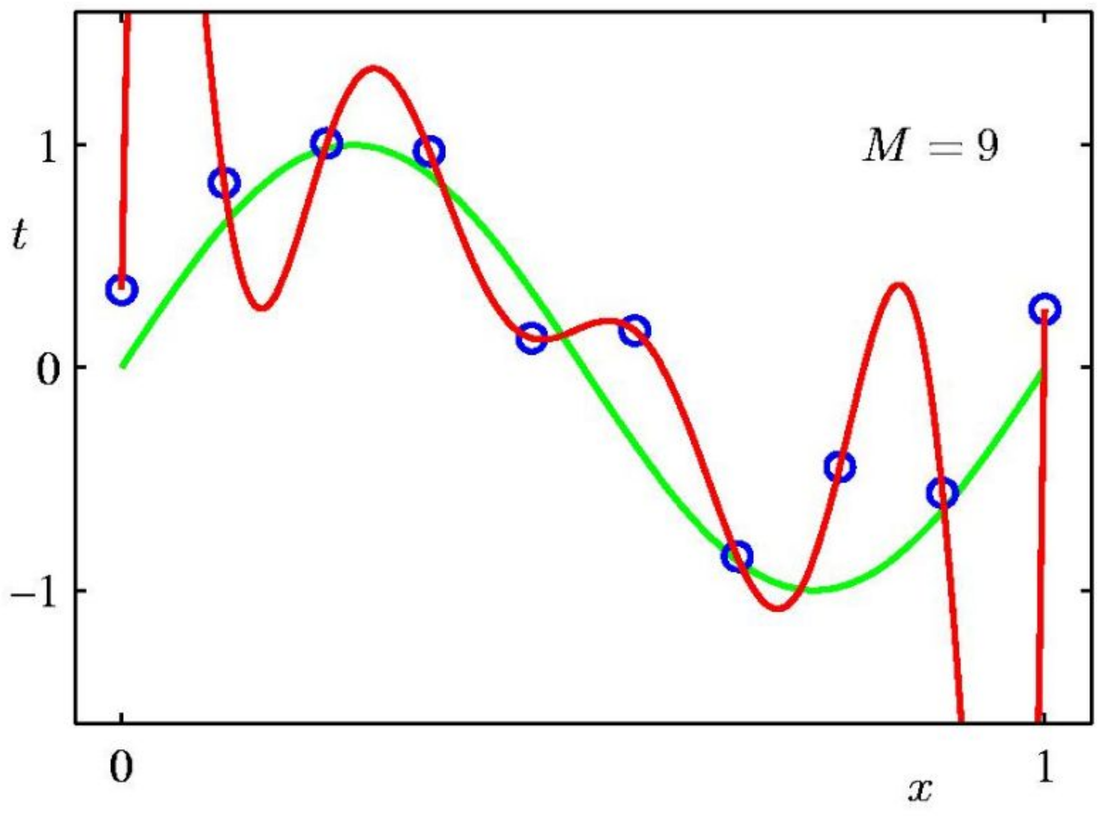
3rd Order Polynomial



9th Order Polynomial



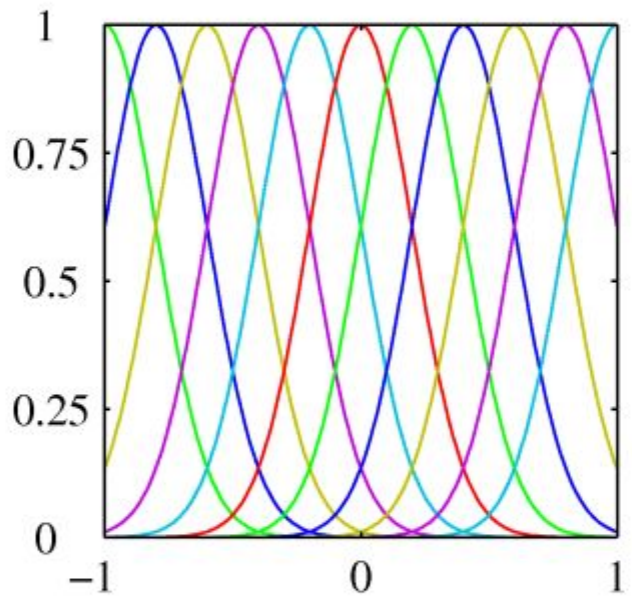
Concept: Overfitting!



Michigan Tech

Gaussian basis functions

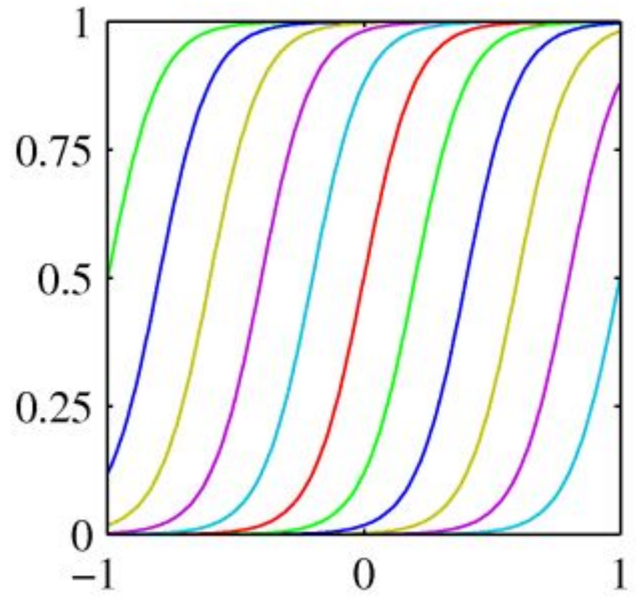
- Local!
  - Small changes in **x** only affect nearby basis functions
  - Parameters control location and scale (width)



$$\phi_j(x) = \exp\left\{-\frac{(x-\mu_j)^2}{2s^2}\right\}$$

Sigmoidal basis functions

- Local!
- Scale parameter affects slope

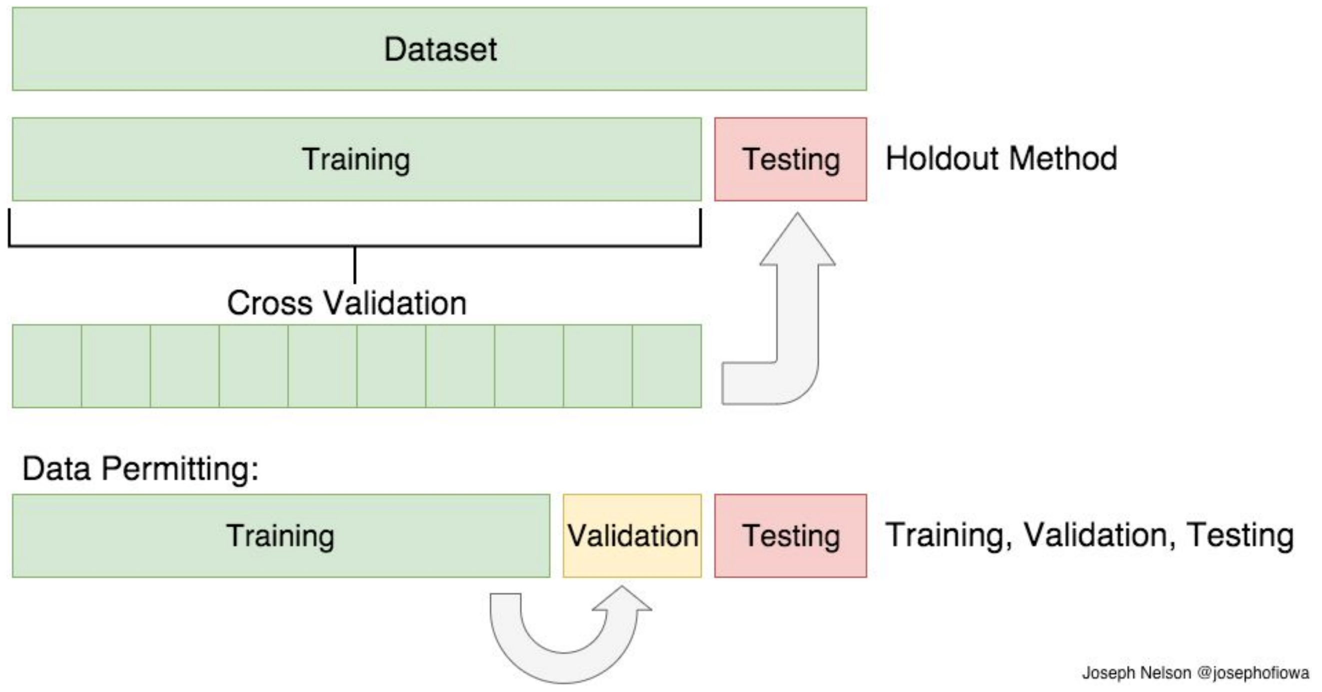


$$\phi_j(x) = \sigma\left(\frac{x-\mu_j}{s}\right)$$

$$\sigma(a) = \frac{1}{1+\exp(-a)}.$$

Data splits

- We want a fair assessment of our model!
  - This requires testing on data we didn't use for training
- Different strategies for test/train split
  - Test/train
  - Test/train/validation
  - Cross validation with separate test split



Test/train/validation splits

- Many big datasets contain test/train/validation splits
  - Some are just test/train
  - Common test split ensures fair comparison
  - Long training time precludes cross validation
- Split uses
  - Test - used to measure final model performance (usually 10-20%)
  - Train - used to train model (usually 70-90%)
  - Validation - used to evaluate model for model tuning (usually 10-20%)

Cross validation

- *K-fold cross validation*
  - Partition data X_{train} into K separate sets of equal size
    - $X_{train} = (X_{train,1}, X_{train,2}, \ldots X_{train,K})$
  - Common K are K=5 and K=10
- For each k=1,2,…,K
  - Fit the model *$y(w,\lambda)$* to the training set excluding kth fold $X_{train,k}$
  - Compute values for $X_{train,k}$ and compute error
  - Repeat for each

Special cases of cross validation

- What if we do n-fold cross validation, where n is the dataset size?
    - *Leave one out cross validation (LOO CV)*
    - Not very data efficient
    - Good for estimating model performance with all available data
    - Bad for estimating model generalization with unseen data

Data leakage

- Extra data is available during training that is not available during testing/inference
- Ex: this paper (Learning with Signatures [1]) scored 100% on several common benchmark datasets
    - Why is this suspicious?
    - How did they do it?
        - They created different classifiers for each class and only used that classifier for that class!
- Simpler example: predicting yearly salary and including a monthly_salary variable

[1] https://arxiv.org/abs/2204.07953v1

How does data leakage happen?

- Improper featurization
  - Data pre-processing that shares information is fit on test and train splits instead of just test split
- Duplicate datapoints
  - Easy to do when oversampling or augmenting
- Group leakage
  - Ex: dataset includes 1000 patients with 10 x-rays from each, not splitting by patient could cause model to learn patient instead of pathology
- Time leakage
  - Time series data is especially challenging!
  - Generally (not always), you split with older data in training and new data in test

Following slides are directly from Bishop Ch 3

Derivation will be gone over in class, but extra slides are here for your reference. The Bishop ch. 3 should also be a good reference

- Assume observations from a deterministic function with added Gaussian noise:

$$t = y(\mathbf{x}, \mathbf{w}) + \epsilon \qquad \text{where} \qquad p(\epsilon|\beta) = \mathcal{N}(\epsilon|0, \beta^{-1})$$

- which is the same as saying,

$$p(t|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}).$$

- Given observed inputs, $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, and targets, $\mathbf{t} = [t_1, \ldots, t_N]^{\mathrm{T}}$, we obtain the likelihood function

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}(t_n|\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n), \beta^{-1}).$$

- Taking the logarithm, we get

$$\begin{aligned}\ln p(\mathbf{t}|\mathbf{w},\beta) &= \sum_{n=1}^{N} \ln \mathcal{N}(t_n|\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n),\beta^{-1}) \\ &= \frac{N}{2}\ln\beta - \frac{N}{2}\ln(2\pi) - \beta E_D(\mathbf{w})\end{aligned}$$

- where

$$E_D(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\}^2$$

- is the sum-of-squares error.

- Computing the gradient and setting it to zero yields

$$\nabla_{\mathbf{w}} \ln p(\mathbf{t}|\mathbf{w},\beta) = \beta \sum_{n=1}^{N} \left\{ t_n - \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_n) \right\} \boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}} = \mathbf{0}.$$

- Solving for w, we get

$$\mathbf{w}_{\mathrm{ML}} = \overbrace{\left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}}^{\text{The Moore-Penrose pseudo-inverse, } \boldsymbol{\Phi}^{\dagger}.}\mathbf{t}$$

- where

$$\boldsymbol{\Phi} = \begin{pmatrix} \phi_0(\mathbf{x}_1) & \phi_1(\mathbf{x}_1) & \cdots & \phi_{M-1}(\mathbf{x}_1) \\ \phi_0(\mathbf{x}_2) & \phi_1(\mathbf{x}_2) & \cdots & \phi_{M-1}(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\mathbf{x}_N) & \phi_1(\mathbf{x}_N) & \cdots & \phi_{M-1}(\mathbf{x}_N) \end{pmatrix}.$$

- Maximizing with respect to the bias, w_0, alone, we see that

$$w_0 = \bar{t} - \sum_{j=1}^{M-1} w_j \overline{\phi_j}$$

We can also maximize with respect to the noise precision parameter, □

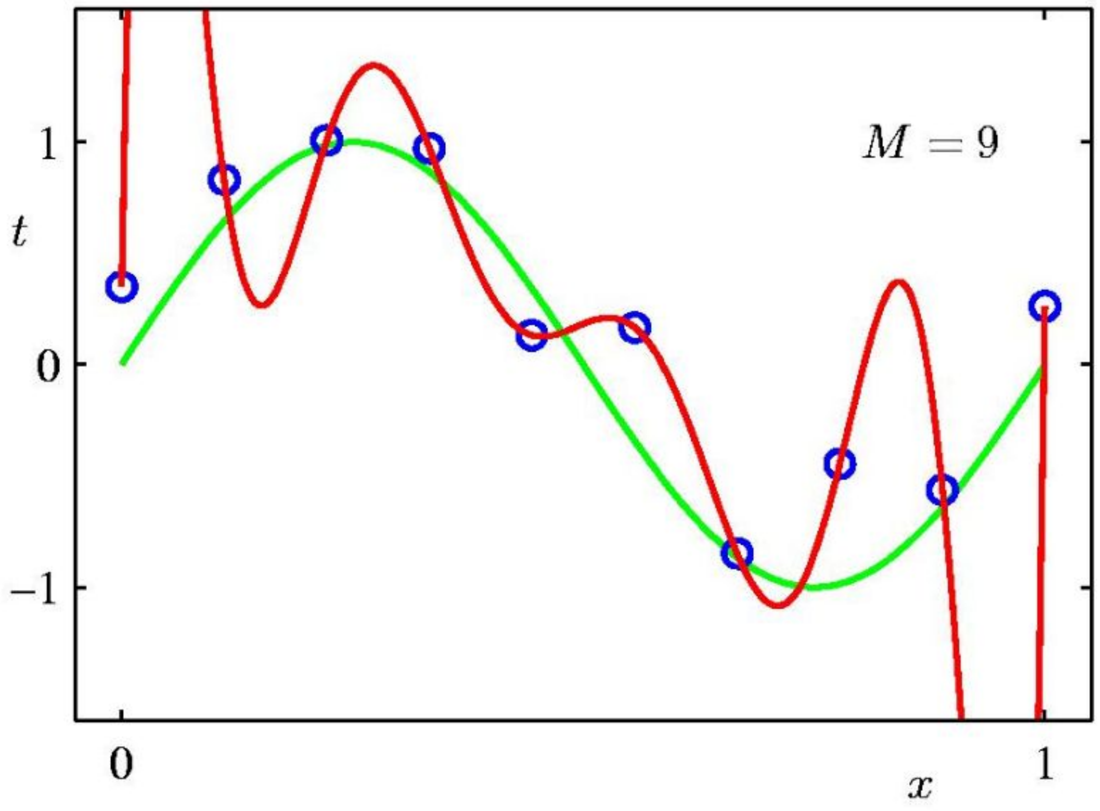
$$\frac{1}{\beta_{\mathrm{ML}}} = \frac{1}{N} \sum_{n=1}^{N} \{t_n - \mathbf{w}_{\mathrm{ML}}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_n)\}^2$$

- Data items considered one at a time (a.k.a. online learning); use stochastic (sequential) gradient descent:

$$
\begin{aligned}
\mathbf{w}^{(\tau+1)} &= \mathbf{w}^{(\tau)} - \eta \nabla E_n \\
&= \mathbf{w}^{(\tau)} + \eta (t_n - \mathbf{w}^{(\tau)\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_n)) \boldsymbol{\phi}(\mathbf{x}_n).
\end{aligned}
$$

- This is known as the *least-mean-squares (LMS) algorithm*. Issue: how to choose η?

Concept: Overfitting!



- Consider the error function:

$$E_D(\mathbf{w}) + \lambda E_W(\mathbf{w})$$

Data term + Regularization term

- With the sum-of-squares error function and a quadratic regularizer, we get

$$\frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\}^2 + \frac{\lambda}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$$

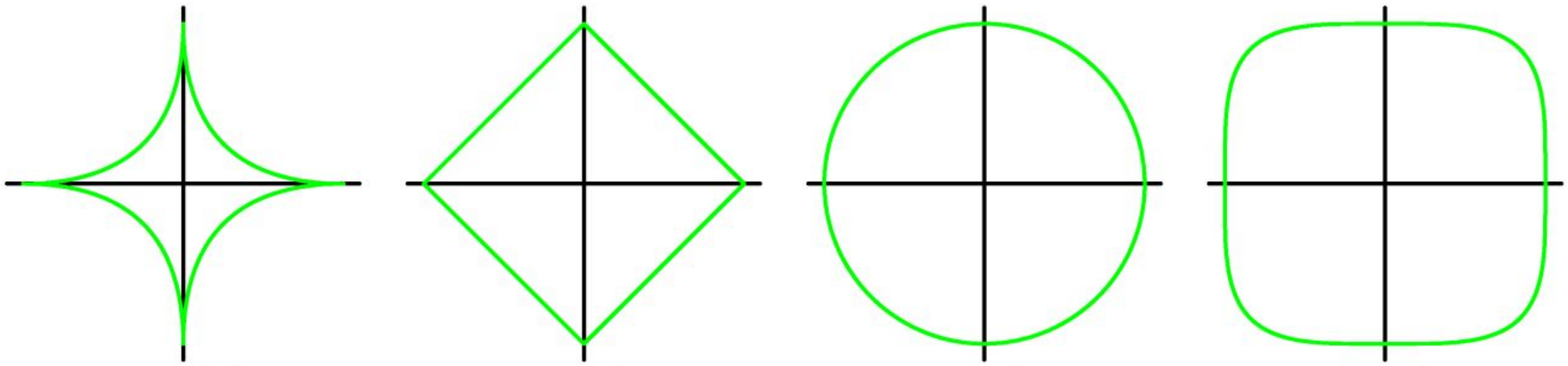
- which is minimized by

$$\mathbf{w} = \left(\lambda\mathbf{I} + \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{t}.$$

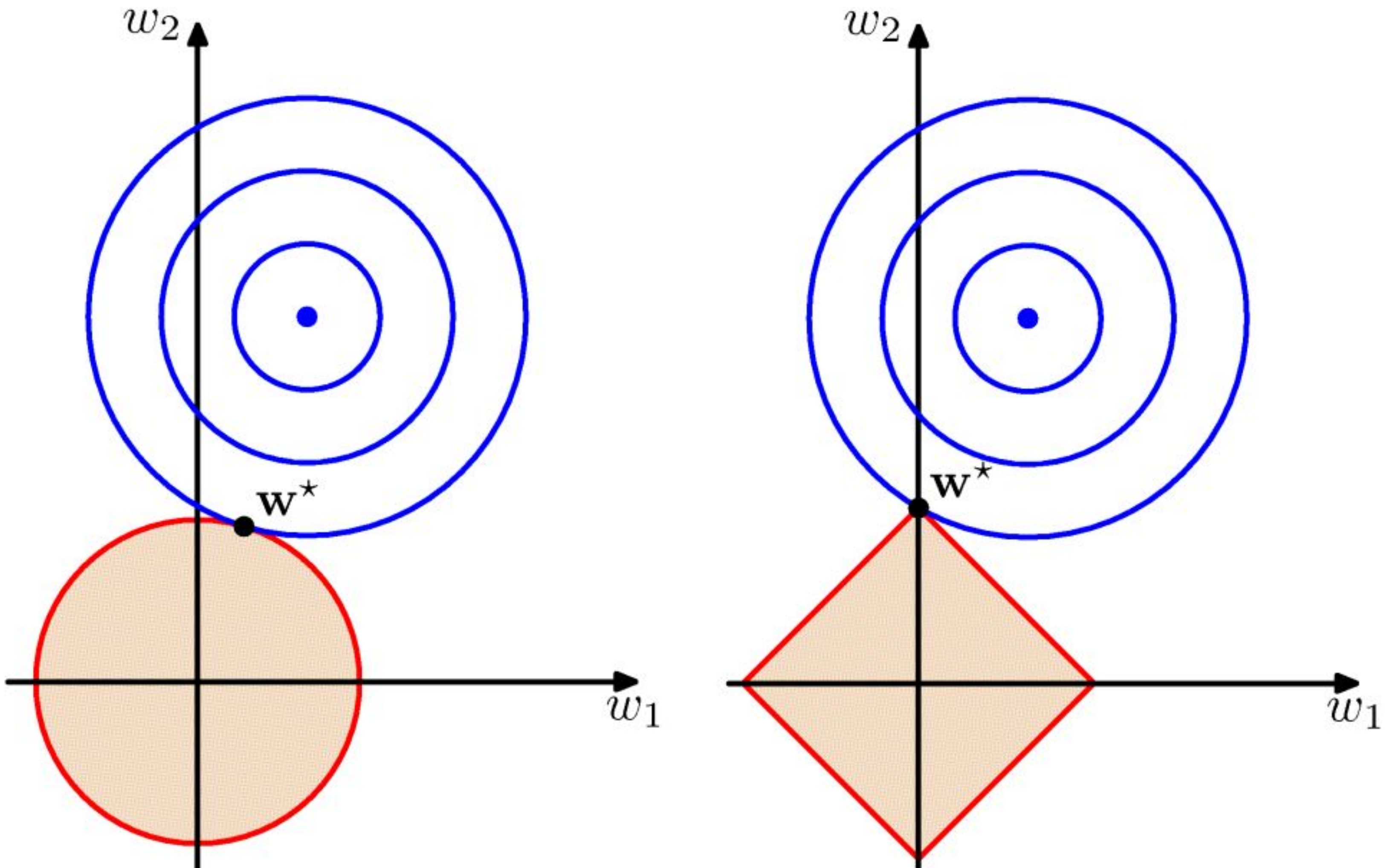
λ is called the regularization coefficient.

- With a more general regularizer, we have

$$\frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\}^2 + \frac{\lambda}{2}\sum_{j=1}^{M}|w_j|^q$$



Lasso tends to generate sparser solutions than a quadratic regularizer.



- Recall the *expected squared loss,*

$$\mathbb{E}[L] = \int \{y(\mathbf{x}) - h(\mathbf{x})\}^2 p(\mathbf{x})\,\mathrm{d}\mathbf{x} + \underbrace{\iint \{h(\mathbf{x}) - t\}^2 p(\mathbf{x}, t)\,\mathrm{d}\mathbf{x}\,\mathrm{d}t}$$

- where

$$h(\mathbf{x}) = \mathbb{E}[t|\mathbf{x}] = \int t p(t|\mathbf{x})\,\mathrm{d}t.$$

- The second term of E[*L*] corresponds to the noise inherent in the random variable *t*.

- What about the first term?

- Suppose we were given multiple data sets, each of size **N**. Any particular data set, **D**, will give a particular function **y(x;D)**. We then have

$$\begin{aligned}
&\{y(\mathbf{x};\mathcal{D}) - h(\mathbf{x})\}^2 \\
&\quad = \{y(\mathbf{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})] + \mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})] - h(\mathbf{x})\}^2 \\
&\quad = \{y(\mathbf{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})]\}^2 + \{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})] - h(\mathbf{x})\}^2 \\
&\qquad +2\{y(\mathbf{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})]\}\{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})] - h(\mathbf{x})\}.
\end{aligned}$$

- Taking the expectation over D yields

$$\begin{aligned}
&\mathbb{E}_{\mathcal{D}}\left[\{y(\mathbf{x};\mathcal{D}) - h(\mathbf{x})\}^2\right] \\
&= \underbrace{\{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})] - h(\mathbf{x})\}^2}_{(\text{bias})^2} + \underbrace{\mathbb{E}_{\mathcal{D}}\left[\{y(\mathbf{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})]\}^2\right]}_{\text{variance}}.
\end{aligned}$$

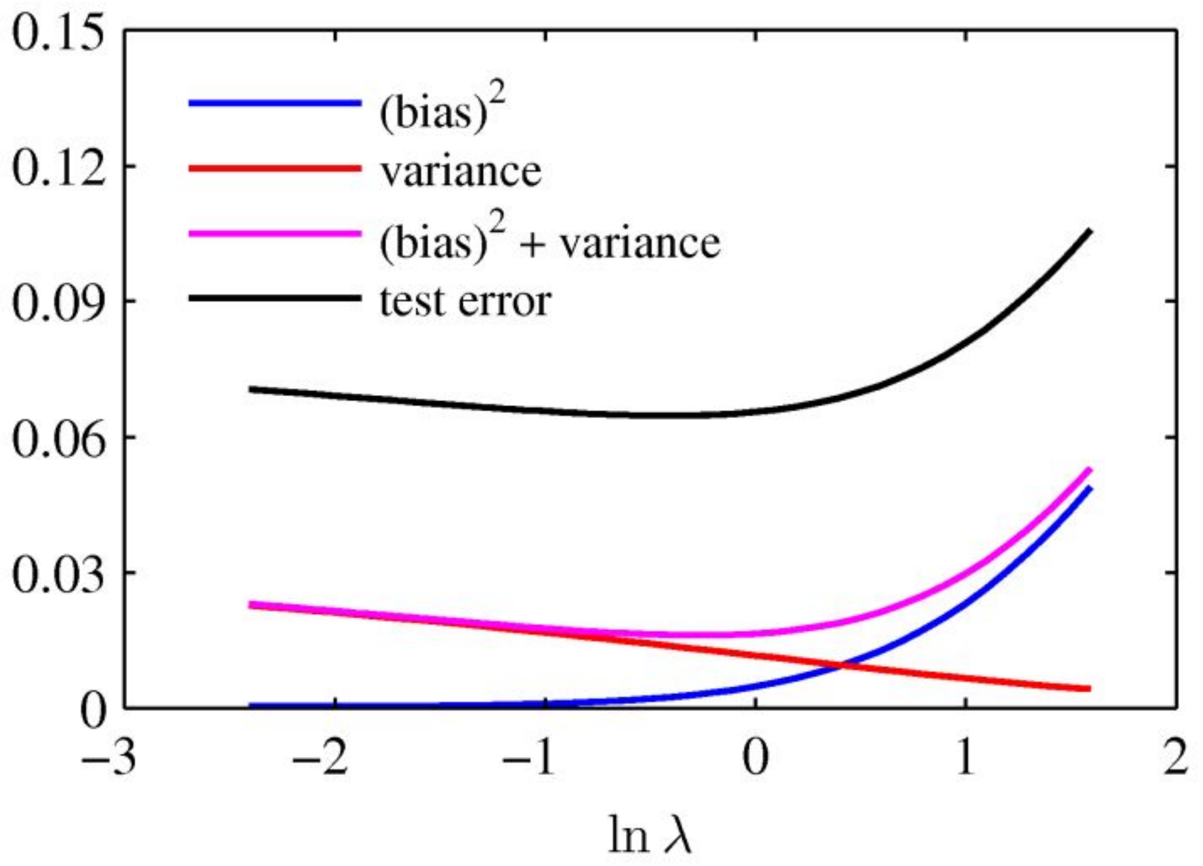
- Thus we can write

$$\text{expected loss} = (\text{bias})^2 + \text{variance} + \text{noise}$$

- where

$$\begin{aligned}
(\text{bias})^2 &= \int \{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})] - h(\mathbf{x})\}^2 p(\mathbf{x})\,\mathrm{d}\mathbf{x} \\
\text{variance} &= \int \mathbb{E}_{\mathcal{D}}\left[\{y(\mathbf{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})]\}^2\right] p(\mathbf{x})\,\mathrm{d}\mathbf{x} \\
\text{noise} &= \iint \{h(\mathbf{x}) - t\}^2 p(\mathbf{x},t)\,\mathrm{d}\mathbf{x}\,\mathrm{d}t
\end{aligned}$$

From these plots, we note that an over-regularized model (large λ) will have a high bias, while an under-regularized model (small λ) will have a high variance.



Questions + Comments?