Applied Machine Learning

Regularization

Reihaneh Rabbany

McGill School of Computer Science

COMP 551 (winter 2020)

Learning objectives

Basic idea of

- overfitting and underfitting
- regularization (L1 & L2)
- MLE vs MAP estimation
- bias and variance trade off
- evaluation metrics & cross validation

Previously...

Linear regression and logistic regression

is linear too simple? what if it's not a good fit?

how to increase the models expressiveness?

- create new nonlinear features
- is there a downside?

Recall: Linear regression

model

$$f_w(x) = \underbrace{\hat{y}^{(n)}}_{\in \mathbb{R}} = \underbrace{w^T}_{1 \times D} \underbrace{x^{(n)}}_{D \times 1}$$

cost

$$J(w) = \frac{1}{2} \sum_n \left(y^{(n)} - w^T x^{(n)} \right)^2$$

linear least squares **(LLS)**

Optimization

$$\sum_n (y^{(n)} - w^T x^{(n)}) x_d^{(n)} = 0 \quad \forall d$$

matrix notation

$$\underbrace{\hat{y}}_{N \times 1} = \underbrace{X}_{N \times D} \underbrace{w}_{D \times 1}$$

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

$$X^T (y - Xw) = \vec{0}$$

$$\underbrace{w^*}_{D \times 1} = (\underbrace{X^T}_{D \times N} \underbrace{X}_{N \times D})^{-1} X^T \underbrace{y}_{N \times 1}$$

Recall: Linear regression

$$\underset{D \times 1}{w^*} = (\underset{D \times N}{X^T}\,\underset{N \times D}{X})^{-1} X^T \underset{N \times 1}{y}$$

what if $X^T X$ is **not invertible**?

add a small value to the diagonals, a.k.a. regularize

what if **linear fit is not the best**?

use nonlinear basis

Recall: nonlinear basis functions

what if **linear fit is not the best**?

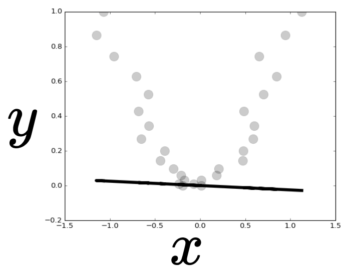
$$X = \begin{bmatrix} x_1^{(1)}, & x_2^{(1)}, & \cdots, & x_D^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(N)}, & x_2^{(N)}, & \cdots, & x_D^{(N)} \end{bmatrix}$$

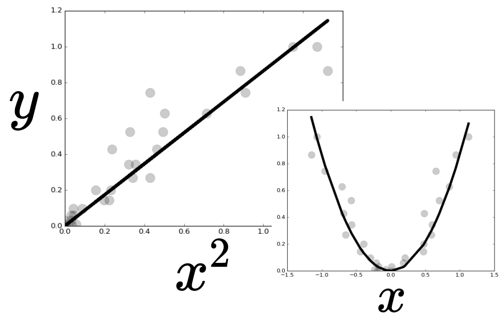
replacing X with Φ

$$\Phi = \begin{bmatrix} \phi_1(x^{(1)}), & \phi_2(x^{(1)}), & \cdots, & \phi_D(x^{(1)}) \\ \phi_1(x^{(2)}), & \phi_2(x^{(2)}), & \cdots, & \phi_D(x^{(2)}) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(x^{(N)}), & \phi_2(x^{(N)}), & \cdots, & \phi_D(x^{(N)}) \end{bmatrix}$$

a (nonlinear) feature

Example





Recall: nonlinear basis functions

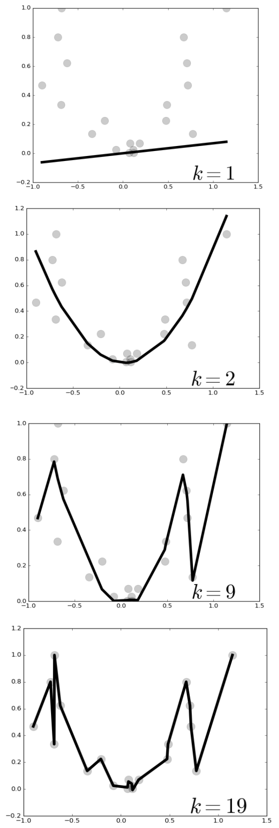
$\Phi_k(x) = x^k$

what if **linear fit is not the best**?

replace original features in $f_w(x) = \sum_d w_d x_d$

with nonlinear bases $f_w(x) = \sum_d w_d \, \phi_d(x)$

linear least squares (LLS) solution $w^* = (\Phi^T \Phi)^{-1} \Phi^T y$



How to avoid overfitting then?

regularize

Regularization serves many purposes

$$\underset{D\times 1}{w^*} = (\underset{D\times N}{X^T}\,\underset{N\times D}{X})^{-1}X^T\underset{N\times 1}{y}$$

what if X^TX is **not invertible**?

add a small value to the diagonals, a.k.a. **regularize**

what if **linear fit is not the best**?

use nonlinear basis

How to avoid **overfitting** then? **regularize**

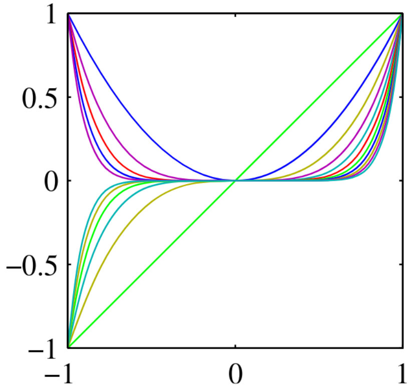
what if **we want a sparse model**?

do feature selection and only keep important parameters with **regularizing**

▶ Today, more on overfitting and regularization

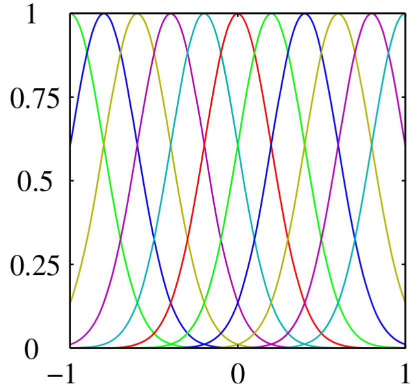
Recall: nonlinear basis functions

examples original input is scalar $x \in \mathbb{R}$



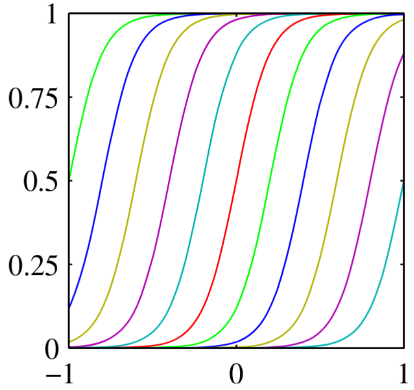
polynomial bases

$$\phi_k(x) = x^k$$



Gaussian bases

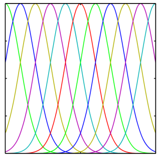
$$\phi_k(x) = e^{-\frac{(x-\mu_k)^2}{s^2}}$$



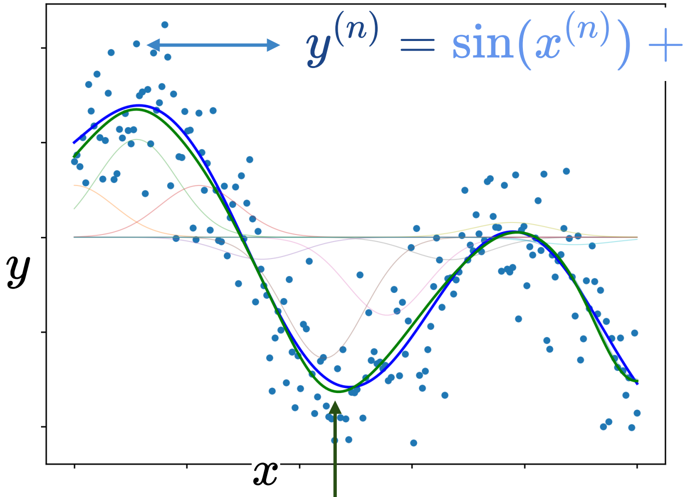
Sigmoid bases

$$\phi_k(x) = \frac{1}{1+e^{-\frac{x-\mu_k}{s}}}$$

Example: Gaussian bases



$$\phi_k(x) = e^{-\frac{(x-\mu_k)^2}{s^2}}$$



$$y^{(n)} = \sin(x^{(n)}) + \cos(\sqrt{|x^{(n)}|}) + \epsilon$$

our fit to data using 10 Gaussian bases

prediction for a new instance

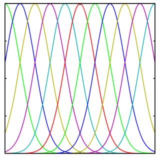
$$f(x') = \phi(x')^T \underbrace{(\Phi^T\Phi)^{-1}\Phi^T y}_{w}$$

new instance

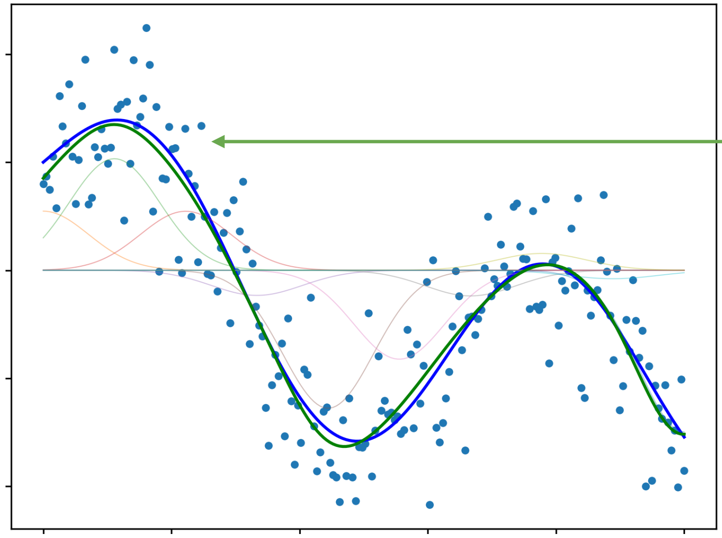
features evaluated for the new point

w found using LLS

Example: Gaussian bases

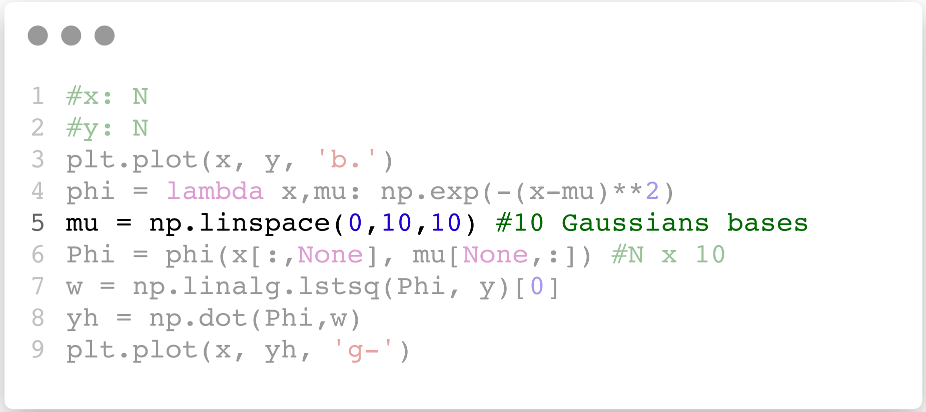


$$\phi_k(x) = e^{-\frac{(x-\mu_k)^2}{s^2}}$$



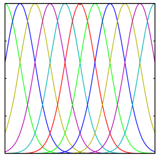
our fit to data using **10 Gaussian bases**

why not more?



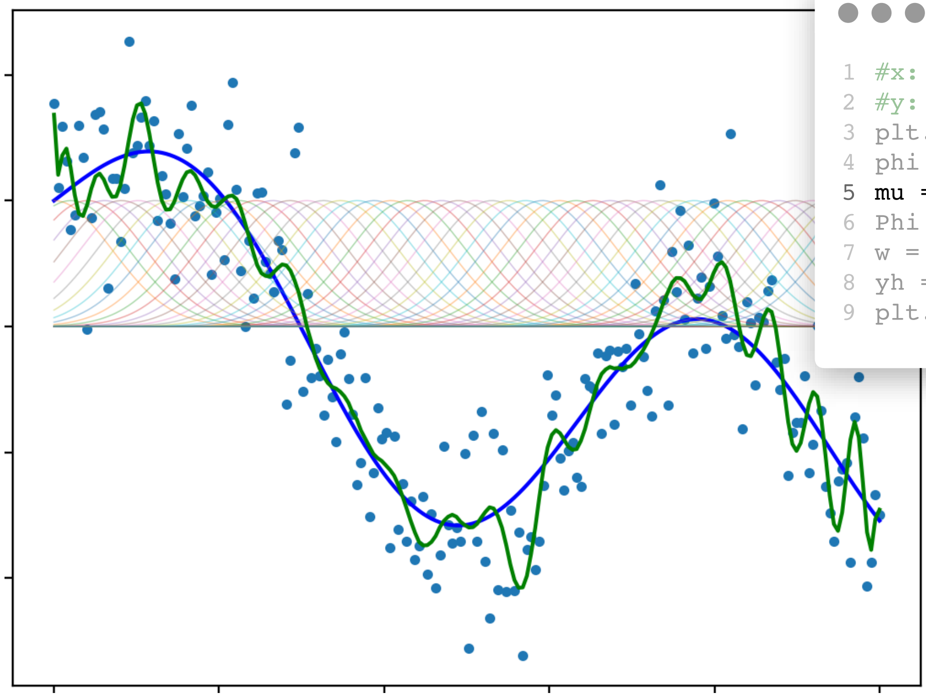
```
#x: N
#y: N
plt.plot(x, y, 'b.')
phi = lambda x,mu: np.exp(-(x-mu)**2)
mu = np.linspace(0,10,10) #10 Gaussians bases
Phi = phi(x[:,None], mu[None,:]) #N x 10
w = np.linalg.lstsq(Phi, y)[0]
yh = np.dot(Phi,w)
plt.plot(x, yh, 'g-')
```

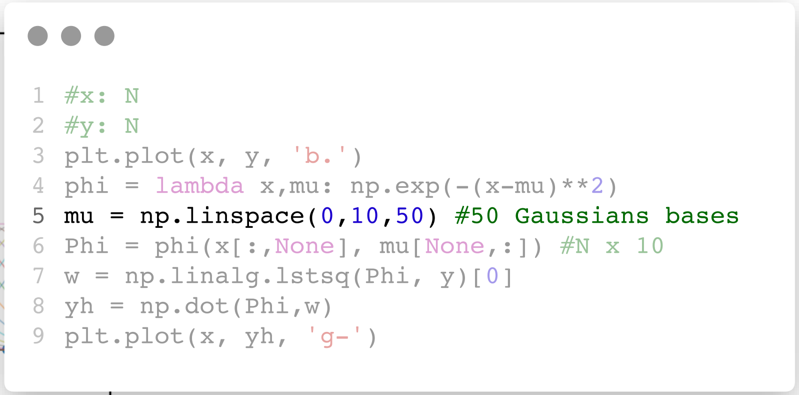
Example: Gaussian bases



$$\phi_k(x) = e^{-\frac{(x-\mu_k)^2}{s^2}}$$

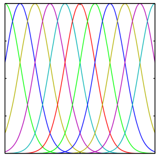
using 50 bases





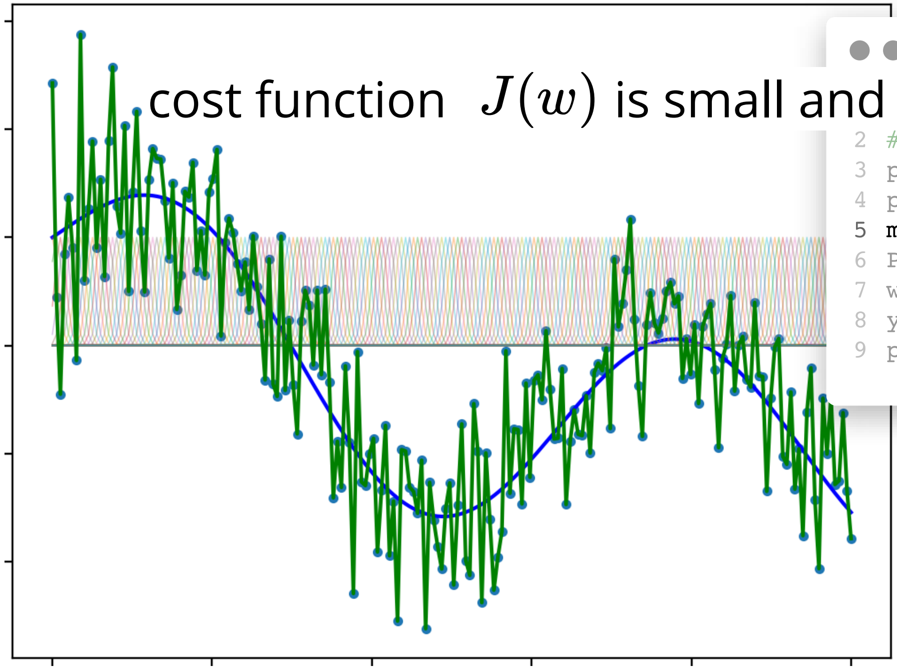
```
#x: N
#y: N
plt.plot(x, y, 'b.')
phi = lambda x,mu: np.exp(-(x-mu)**2)
mu = np.linspace(0,10,50) #50 Gaussians bases
Phi = phi(x[:,None], mu[None,:]) #N x 10
w = np.linalg.lstsq(Phi, y)[0]
yh = np.dot(Phi,w)
plt.plot(x, yh, 'g-')
```

Example: Gaussian bases

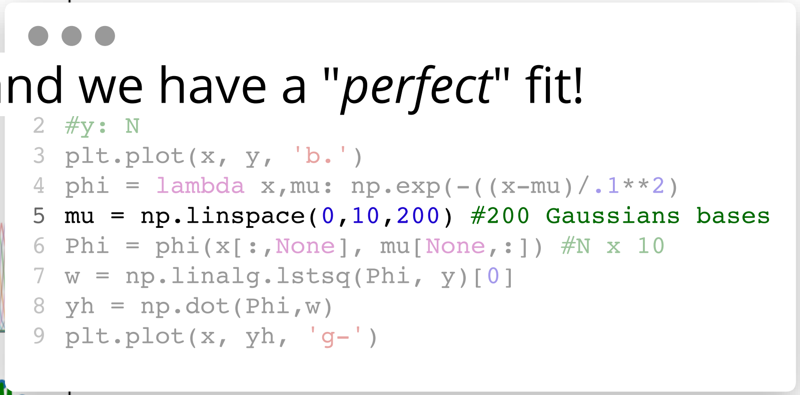


$$\phi_k(x) = e^{-\frac{(x-\mu_k)^2}{s^2}}$$

using 200, thinner bases (s=.1)

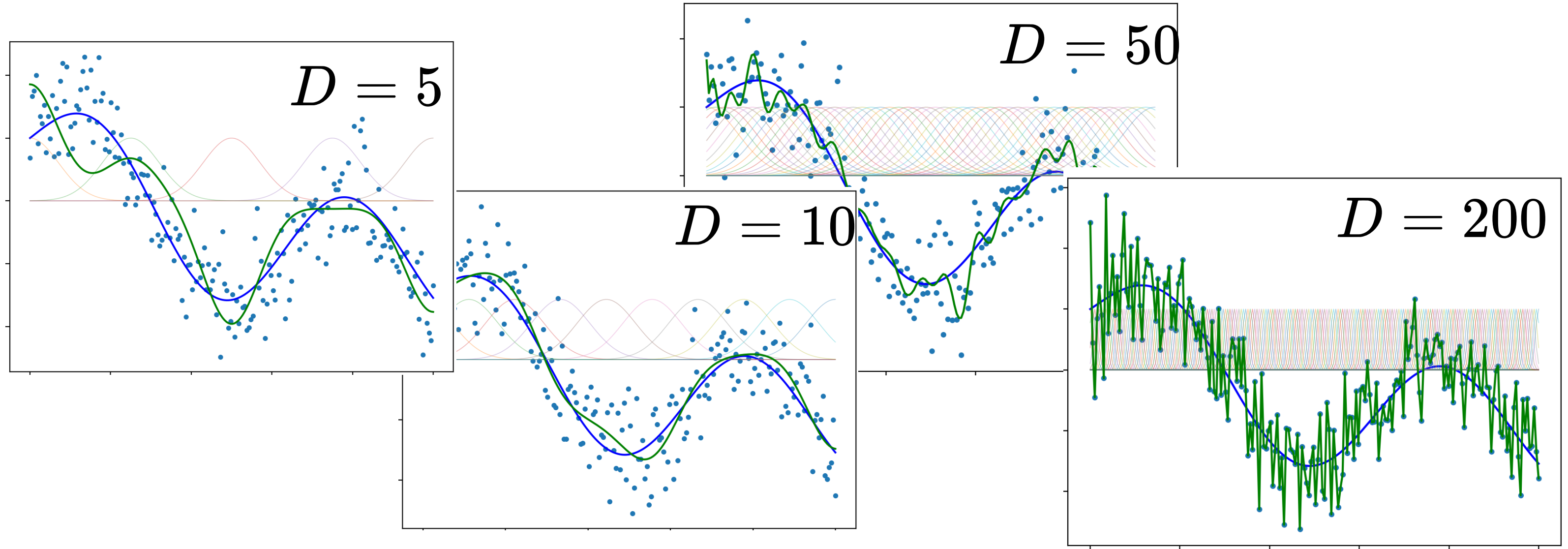


cost function $J(w)$ is small and we have a "*perfect*" fit!



```
#y: N
plt.plot(x, y, 'b.')
phi = lambda x,mu: np.exp(-((x-mu)/.1**2)
mu = np.linspace(0,10,200) #200 Gaussians bases
Phi = phi(x[:,None], mu[None,:]) #N x 10
w = np.linalg.lstsq(Phi, y)[0]
yh = np.dot(Phi,w)
plt.plot(x, yh, 'g-')
```

Generalization

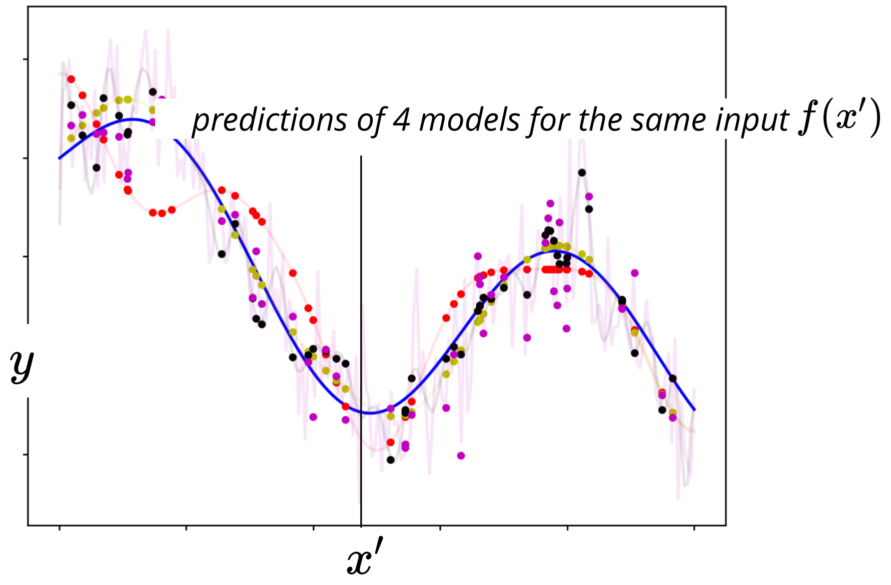


lower training error

which one of these models performs better at test time?

Overfitting

which one of these models performs better at test time?



🟥	$D = 5$	underfitting
🟨	$D = 10$	● lowest test error
⬛	$D = 50$	
🟪	$D = 200$	overfitting

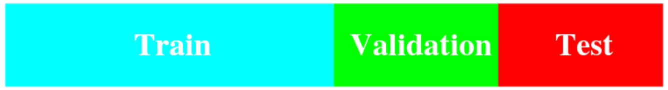
Model selection

how to pick the model with lowest expected loss / test error?

regularization bound the test error by bounding

- training error
- model complexity

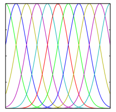
use a **validation set** (and a separate test set for final assessment)



use for model selection use for final model assessment

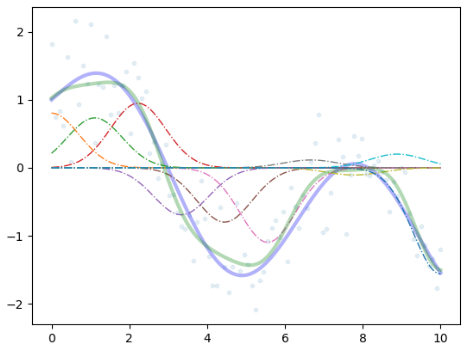
An observation

when overfitting, we often see large weights

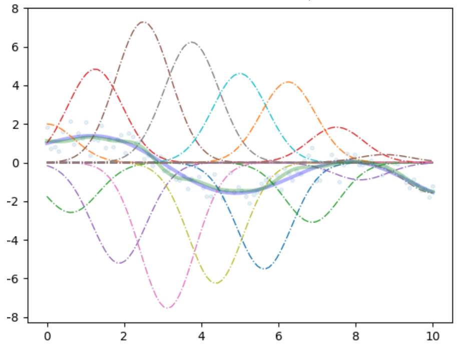


dashed lines are $w_d \phi_d(x) \quad \forall d$

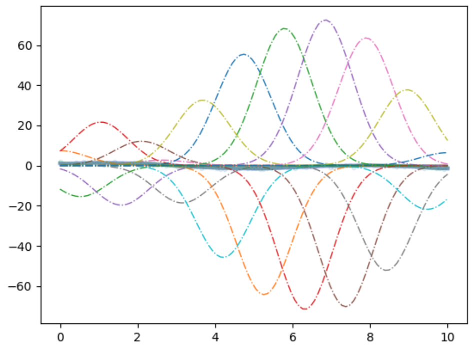
$D = 10$



$D = 17$



$D = 20$



idea: penalize large parameter values

Ridge regression

L2 regularized linear least squares regression:

$$J(w) = \underbrace{\frac{1}{2}||Xw - y||_2^2}_{\text{sum of squared error}} + \frac{\lambda}{2}\underbrace{||w||_2^2}_{\text{(squared) L2 norm of w}}$$

sum of squared error

$\frac{1}{2}\sum_n (y^{(n)} - w^T x)^2$

(squared) L2 norm of w

$w^T w = \sum_d w^2$

regularization parameter $\lambda > 0$ controls the strength of regularization

a good practice is to not penalize the intercept $\lambda(||w||_2^2 - w_0^2)$

Ridge regression

we can set the derivative to zero $J(w) = \frac{1}{2}(Xw-y)^T(Xw-y) + \frac{\lambda}{2}w^Tw$

$\frac{\partial Xw}{\partial w} = X^T$ Using matrix differentiation

$\frac{\partial w^T Xw}{\partial w} = 2Xw$

$$\nabla J(w) = X^T(Xw-y) + \lambda w = 0$$

when using gradient descent, this term reduces the weights at each step **(weight decay)**

$$(X^TX + \lambda \mathbf{I})w = X^Ty$$

$$w^* = (X^TX + \lambda \mathbf{I})^{-1}X^Ty$$

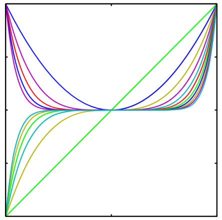
the only part different due to regularization

λI makes it invertible!

we can have linearly dependent features (e.g., D > N)

the solution will be unique!

Example: polynomial bases

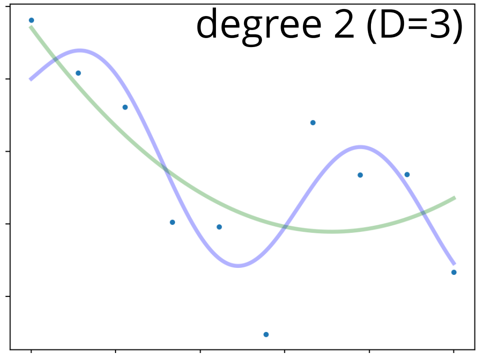


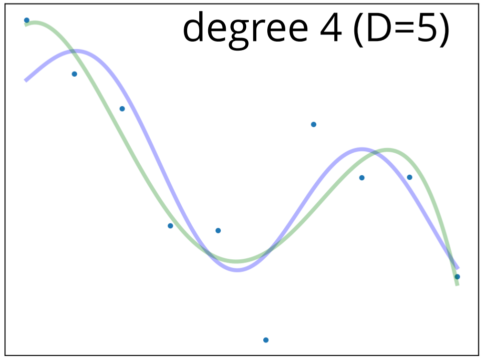
polynomial bases

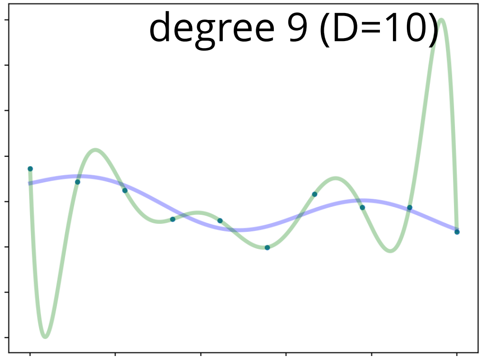
$$\phi_k(x) = x^k$$

Without regularization:

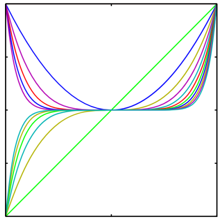
- using D=10 we can perfectly fit the data (high test error)







Example: polynomial bases

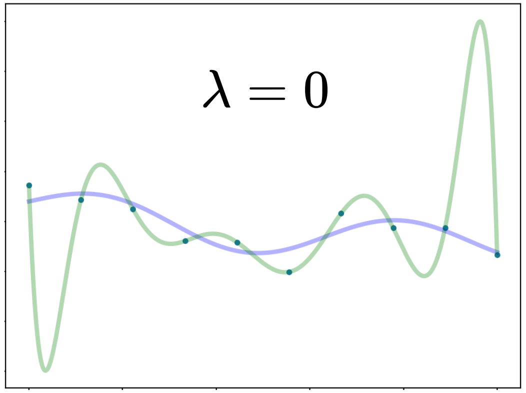


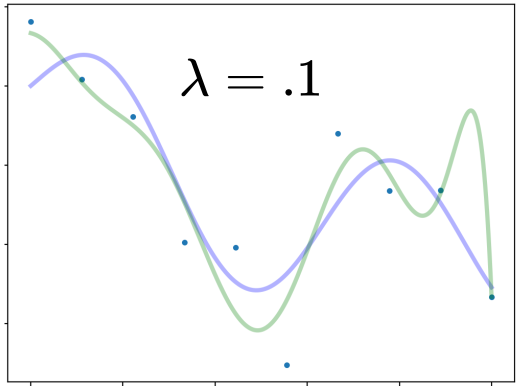
polynomial bases

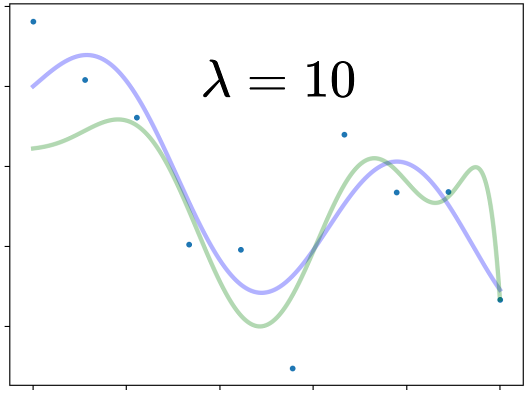
$$\phi_k(x) = x^k$$

with regularization:

- fixed D=10, changing the amount of regularization







Data normalization

what if we scale the input features, using different factors $\tilde{x}^{(n)} = \gamma_d x^{(n)} \forall d, n$

if we have no regularization: $\tilde{w}_d = \frac{1}{\gamma_d} w_d \forall d$

everything remains the same because: $||Xw - y||_2^2 = ||\tilde{X}\tilde{w} - y||_2^2$

with regularization: $||\tilde{w}||_2 \neq ||w||_2^2$ so the optimal **w** will be different!

features of different mean and variance will be penalized differently

normalization $\begin{cases} \mu_d = \frac{1}{N} x_d^{(n)} \\ \sigma_d^2 = \frac{1}{N-1} (x_d^{(n)} - \mu_d)^2 \end{cases}$

makes sure all features have the same mean and variance $x_d^{(n)} \leftarrow \frac{x_d^{(n)} - \mu_d}{\sigma_d}$

Maximum likelihood

previously: linear regression & logistic regression maximize log-likelihood

linear regression

$$w^* = \arg\max p_w(y|x)$$

$$w^* = \arg\max p(y|x, w)$$

$$= \arg\max_w \prod_{n=1}^N \mathcal{N}\left(y; \Phi w, \sigma^2\right)$$

$$\equiv \arg\min \sum_n L_2(y^{(n)}, w^T \phi(x^{(n)}))$$

logistic regression

$$w^* = \arg\max p_w(y|x)$$

$$w^* = \arg\max p(y|x, w)$$

$$= \arg\max_w \prod_{n=1}^N \text{Bernoulli}\left(y; \sigma(\Phi w)\right)$$

$$\equiv \arg\min \sum_n L_{CE}(y^{(n)}, \sigma(w^T \phi(x^n)))$$

idea: maximize the posterior instead of likelihood

$$p(w|y) = \frac{p(w)p(y|w)}{p(y)}$$

Maximum a Posteriori (MAP)

use the Bayes rule and find the parameters with max posterior prob.

$$p(w|y) = \frac{p(w)p(y|w)}{p(y)}$$

the same for all choices of w (ignore)

MAP estimate

$$w^* = \arg\max_w p(w)p(y|w)$$

$$\equiv \arg\max_w \underbrace{\log p(y|w)}_{\text{likelihood: original objective}} + \underbrace{\log p(w)}_{\text{prior}}$$

even better would be to estimate the posterior distribution $p(w|y)$

- more on this later in the course!

Gaussian prior

Gaussian likelihood and **Gaussian** prior

$$\mathcal{N}(\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

$$w^* = \arg\max_w p(w)p(y|w) \quad \equiv \arg\max_w \log p(y|w) + \log p(w)$$

$$\equiv \arg\max_w \log \mathcal{N}(y|w^T x, \sigma^2) + \sum_{d=1}^{D} \log \mathcal{N}(w_d, 0, \tau^2)$$

assuming independent Gaussian (one per each weight)

$$\equiv \arg\max_w \frac{-1}{2\sigma^2}(y - w^T x)^2 - \sum_{d=1}^{D} \frac{1}{2\tau^2} w_d^2 \quad \equiv \arg\min_w \frac{1}{2}(y - w^T x)^2 + \sum_{d=1}^{D} \frac{\sigma^2}{2\tau^2} w_d^2$$

multiple data-points

$$\lambda = \frac{\sigma^2}{\tau^2}$$

$$\downarrow$$

$$\equiv \arg\min_w \frac{1}{2} \sum_n (y^{(n)} - w^T x^{(n)})^2 + \sum_{d=1}^{D} \frac{\lambda}{2} w_d^2$$

L2 regularization

L2- regularization is assuming a Gaussian prior on weights

the same is true for logistic regression (or any other cost function)

Laplace prior

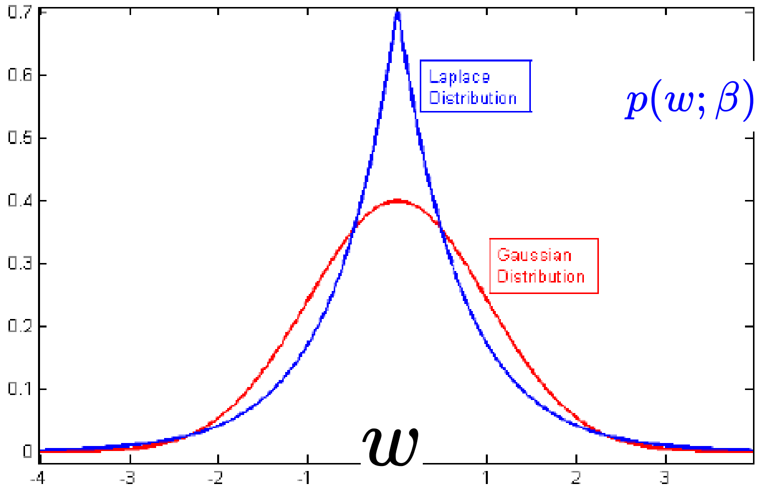
another notable choice of prior is the Laplace distribution

minimizing negative log-likelihood $\rightarrow -\sum_d \log p(w_d) = \sum_d \frac{1}{2\beta}|w_d| = \frac{1}{2\beta}||w||_1$

L1 norm of w

L1 regularization: $J(w) \leftarrow J(w) + \lambda||w||_1$ also called lasso

(least absolute shrinkage and selection operator)



$p(w;\beta) = \frac{1}{2\beta}e^{-\frac{|w|}{\beta}}$ notice the peak around zero

image:https://stats.stackexchange.com/questions/177210/why-is-laplace-prior-producing-sparse-solutions

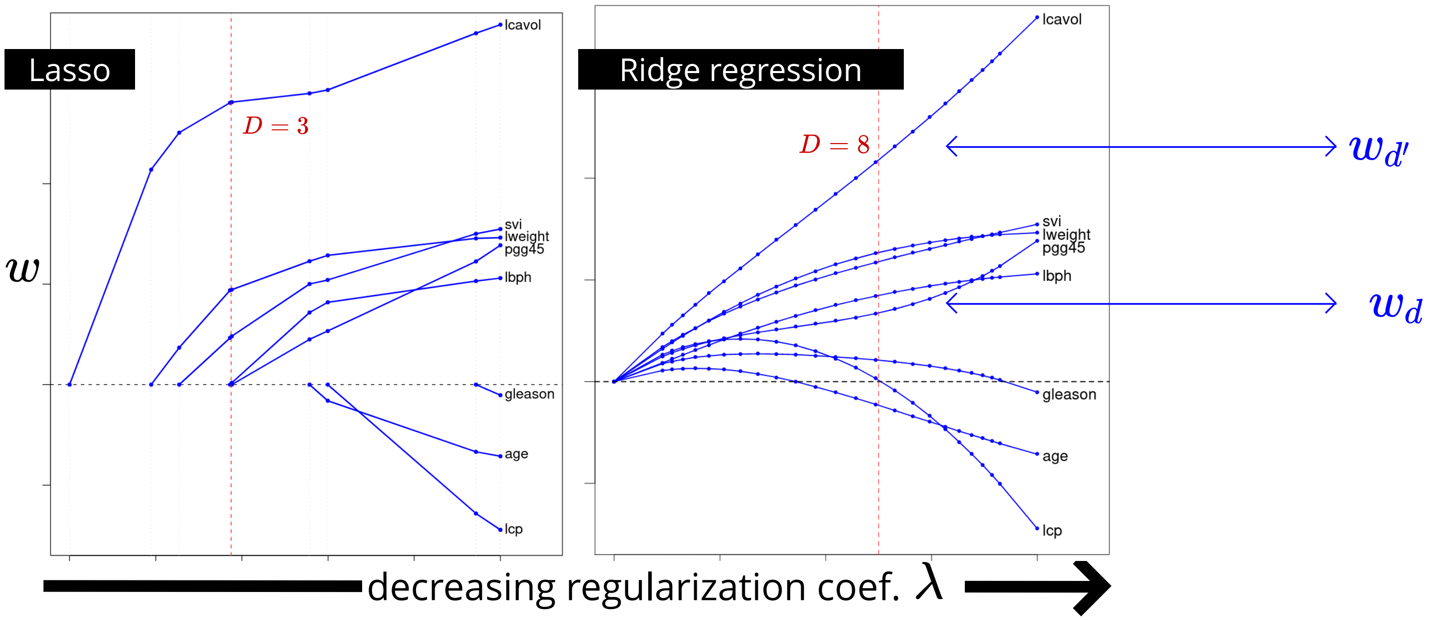
L_1 vs L_2 regularization

regularization path shows how $\{w_d\}$ change as we change λ

Lasso produces sparse weights (many are zero, rather than small)

red-line is the optimal λ from cross-validation

$D = 8$



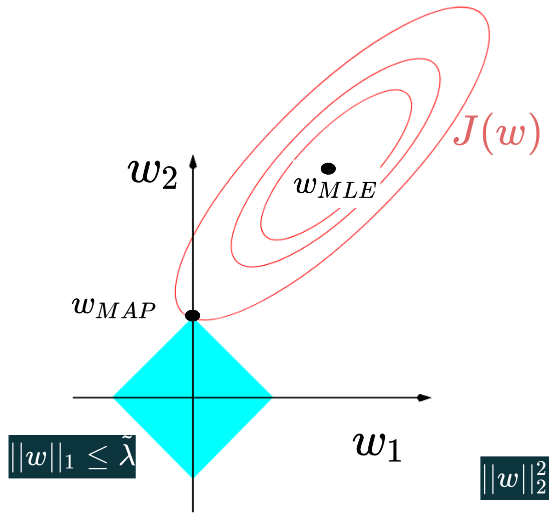
L_1 vs L_2 regularization

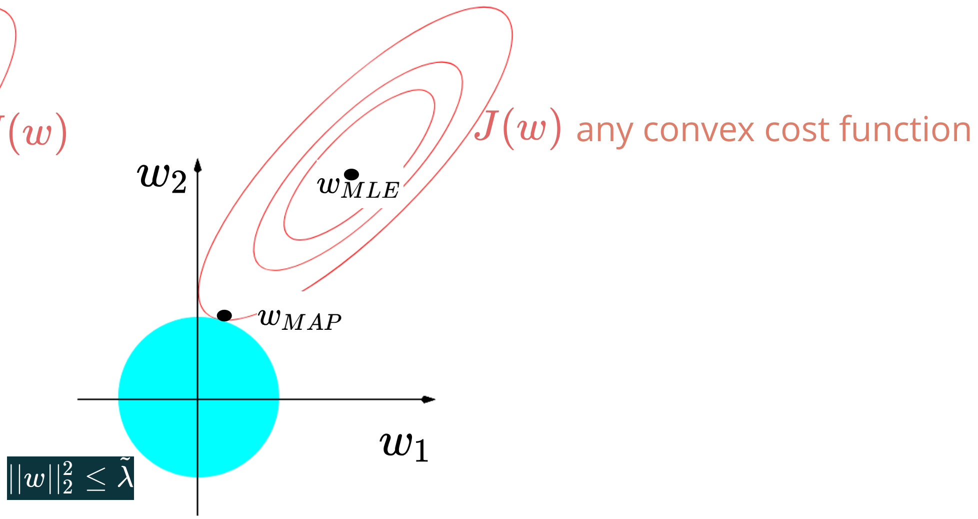
$\min_w J(w) + \lambda ||w||_p^p$

is equivalent to $\min_w J(w)$ *subject to* $||w||_p^p \leq \tilde{\lambda}$ for an appropriate choice of $\tilde{\lambda}$

figures below show the constraint and the isocontours of $J(w)$

optimal solution with L1-regularization is more likely to have zero components

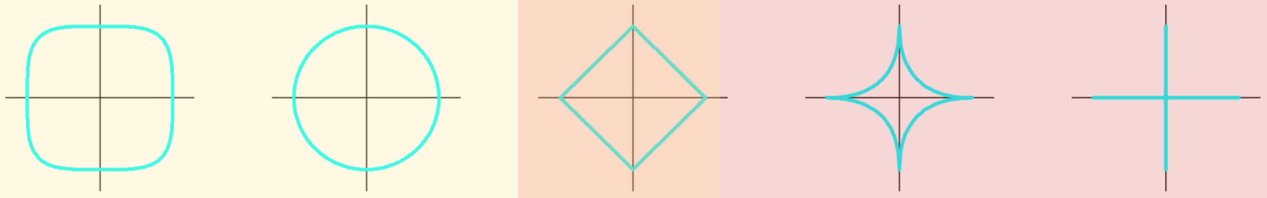



$J(w)$ any convex cost function
w_2
w_{MLE}
w_{MAP}
w_1
$||w||_2^2 \leq \tilde{\lambda}$

Subset selection

p-norms with $p \geq 1$ are convex (easier to optimize)

p-norms with $p \leq 1$ induces sparsity



$\sum_d w_d^4$ $\sum_d w_d^2$ $\sum_d |w_d|$ $\sum_d |w_d|^{\frac{1}{2}}$ $\sum_d |w_d|^{\frac{1}{10}}$

L_0 norm closer to **0-norm**

penalizes the **number of** non-zero features

$$J(w) + \lambda ||w||_0 = J(w) + \lambda \sum_d \mathbb{I}(w_d \neq 0)$$

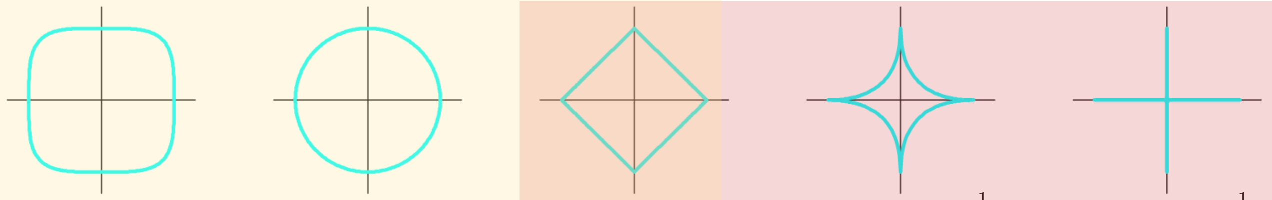
a penalty of λ for each feature

performs feature selection

Subset selection

p-norms with $p \geq 1$ are convex (easier to optimize)

p-norms with $p \leq 1$ induces sparsity



$\sum_d w_d^4$ $\sum_d w_d^2$ $\sum_d |w_d|$ $\sum_d |w_d|^{\frac{1}{2}}$ $\sum_d |w_d|^{\frac{1}{10}}$

L_0 norm ——— closer to **0-norm** →

optimizing this is a difficult *combinatorial problem*:

- search over all 2^D subsets

L1 regularization is a viable alternative to L0 regularization

Bias-variance decomposition

for L2 loss

assume a true distribution $p(x, y)$

the regression function is $f(x) = \mathbb{E}_p[y|x]$

assume that a dataset $\mathcal{D} = \{(x^{(n)}, y^{(n)})\}_n$ is sampled from $p(x, y)$

let $\hat{f}_\mathcal{D}$ be our model based on the dataset

what we care about is the expected loss (*aka risk*)

$$\mathbb{E}[(\hat{f}_\mathcal{D}(x) - y)^2]$$

all blue items are random variables

Bias-variance decomposition

for L2 loss

what we care about is the expected loss (*aka risk*)

$$\mathbb{E}[(\underbrace{\hat{f}_{\mathcal{D}}(x)}_{\hat{f}_{\mathcal{D}}(x) + \mathbb{E}_{\mathcal{D}}[\hat{f}_{\mathcal{D}}(x)] - \mathbb{E}_{\mathcal{D}}[\hat{f}_{\mathcal{D}}(x)]} - \underbrace{y}_{f(x)+\epsilon})^2] \quad = \mathbb{E}[(\hat{f}_{\mathcal{D}}(x) - \mathbb{E}_{\mathcal{D}}[\hat{f}_{\mathcal{D}}(x)] - y + \mathbb{E}_{\mathcal{D}}[\hat{f}_{\mathcal{D}}(x)])^2]$$

add and subtract a term

the remaining terms evaluate to zero (check for yourself!)

$$= \underbrace{\mathbb{E}[(\hat{f}_{\mathcal{D}}(x) - \mathbb{E}_{\mathcal{D}}[\hat{f}_{\mathcal{D}}(x)])^2]}_{\text{variance}} + \underbrace{\mathbb{E}[(f(x) - \mathbb{E}_{\mathcal{D}}[\hat{f}_{\mathcal{D}}(x)])^2]}_{\text{bias}} + \underbrace{\mathbb{E}[\epsilon^2]}_{\text{unavoidable noise error}}$$

Bias-variance decomposition

for L2 loss

the expected loss is decomposed to:

$$\mathbb{E}[(f(x) - \mathbb{E}_{\mathcal{D}}[\hat{f}_{\mathcal{D}}(x)])^2]$$

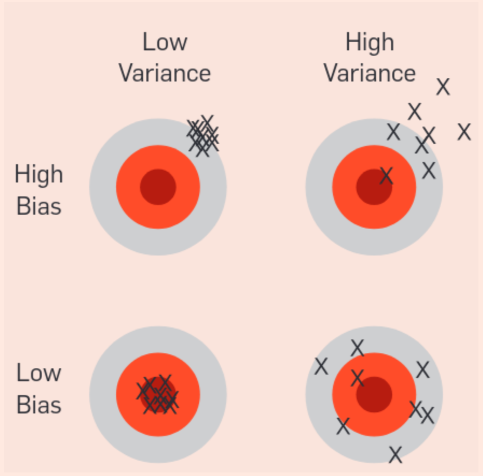
bias: how average over all datasets differs from the regression function

$$\mathbb{E}[(\hat{f}_{\mathcal{D}}(x) - \mathbb{E}_{\mathcal{D}}[\hat{f}_{\mathcal{D}}(x)])^2]$$

variance: how change of dataset affects the prediction

$$\mathbb{E}[\epsilon^2]$$

noise error: the error even if we used the true model $f(x)$

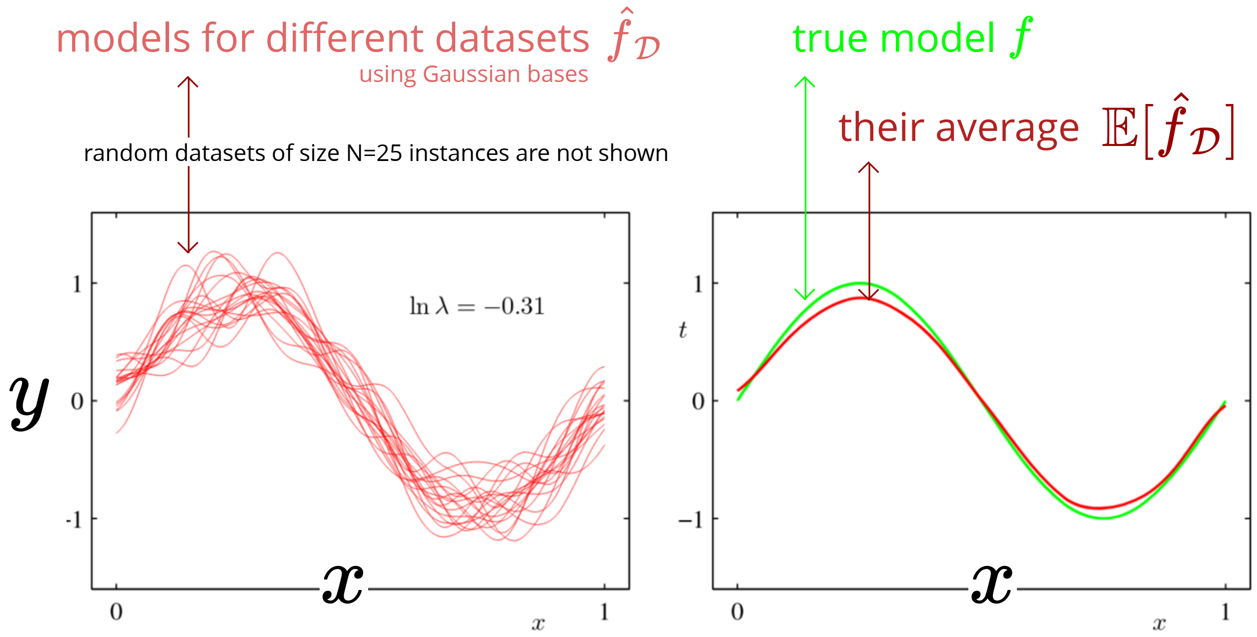


different models vary in their trade off between error due to bias and variance

- simple models: often more biased
- complex models: often have more variance

image: P. Domingos' posted article

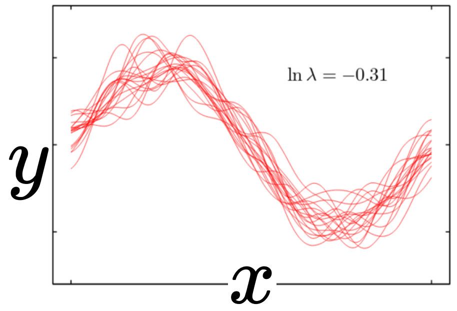
Example: bias vs. variance

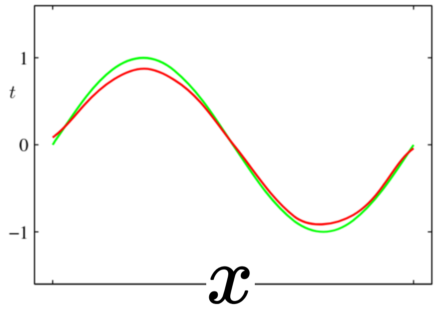


variance is the average difference (in squared L2 norm) between these curves and their average

bias is the difference (in L2 norm) between two curves

Example: bias vs. variance



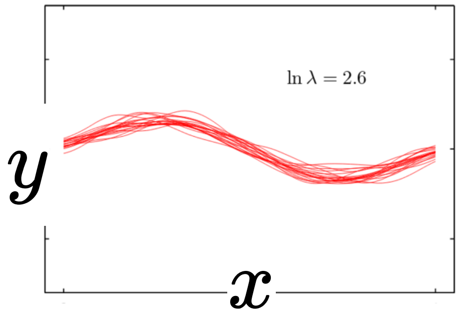


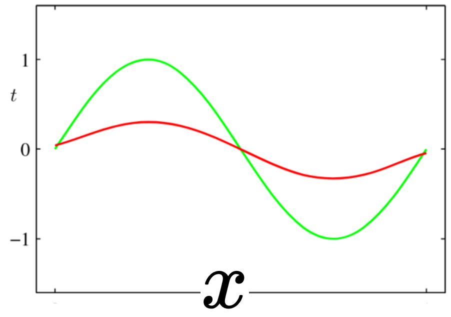
side note

the average fit is very good, despite high variance

model averaging: uses "average" prediction of expressive models to prevent overfitting

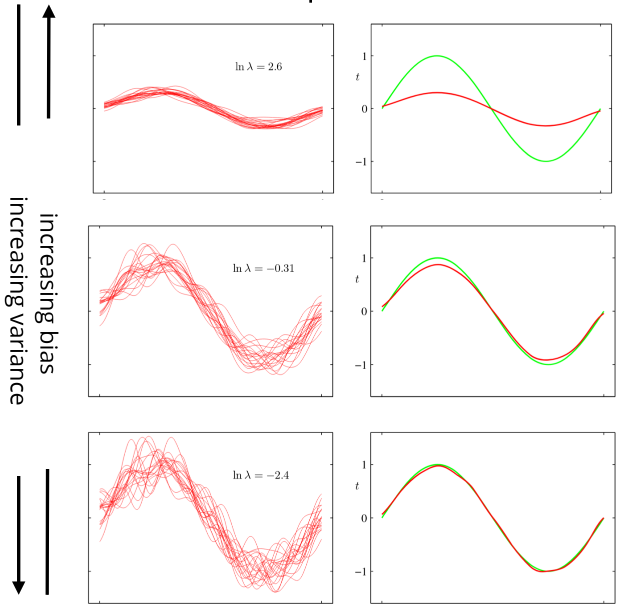
using larger regularization penalty: higher bias - lower variance

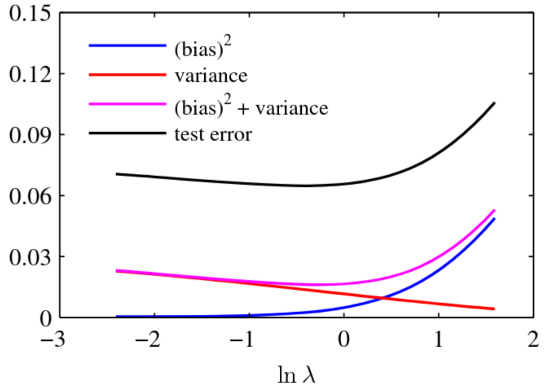




Example: bias vs. variance

the lowest expected loss (test error) is somewhere between the two extremes



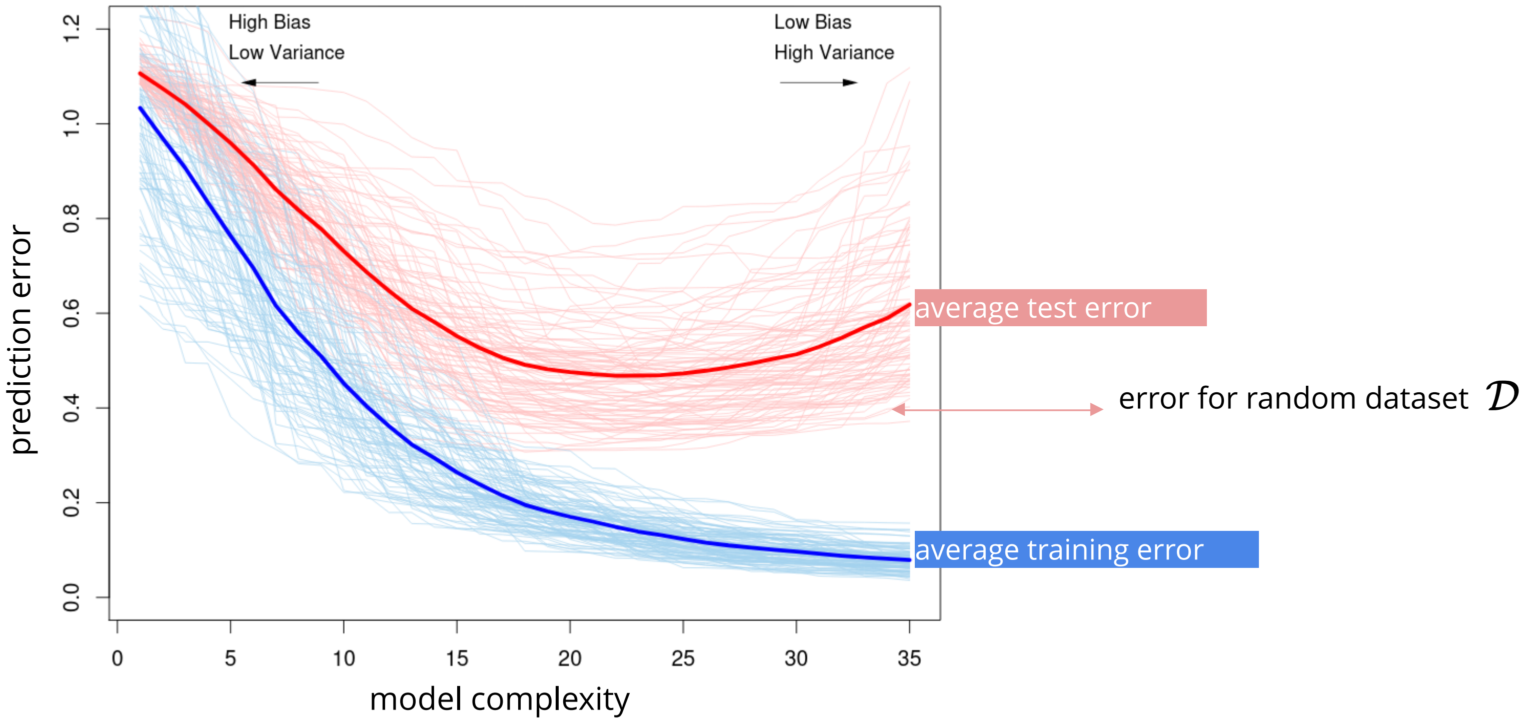


in reality, we don't have access to the true model

how to decide which model to use?

Big picture!

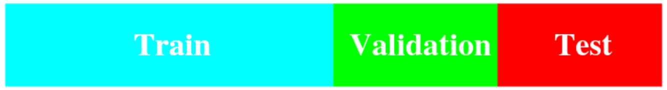
high variance in more complex models means that test and training error can be very different
high bias in simplistic models means that training error can be high



Model selection

how to pick the model with lowest expected loss / test error?

use a **validation set** (and a separate test set for final assessment)



use for model selection   use for final model assessment

regularization   bound the test error by bounding

- training error
- model complexity

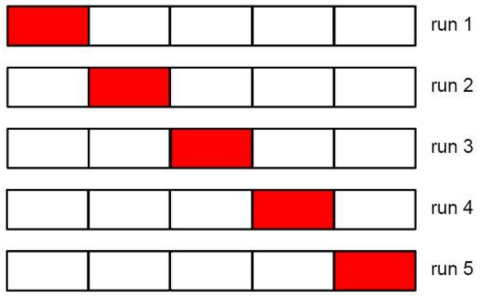
in the end we may have to use a validation set to find the right amount of regularization

Cross validation

getting a more reliable estimate of test error using validation set

K-fold cross validation(CV)

- randomly partition the data into K *folds*
- use K-1 for training, and 1 for validation
- report average/std of the validation error over all folds



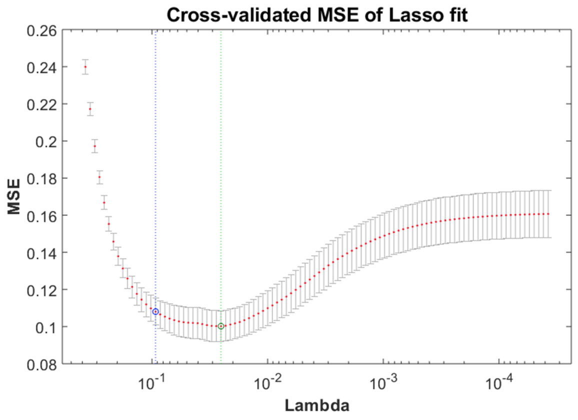
leave-one-out CV: extreme case of k=N

Cross validation

getting a more reliable estimate of test error using validation set

K-fold cross validation(CV)

- randomly partition the data into k *folds*
- use k-1 for training, and 1 for validation
- report average/std of the validation error over all folds



once the hyper-parameters are selected, we can use the whole set for training

use test set for the **final** assessment

image credit: Thanh Nguyen et al'19

Evaluation

evaluation metric can be different from the optimization objective

confusion matrix is a CxC table that compares truth-vs-prediction

for **binary classification**:

	Truth		Σ
Result	TP	FP	RP
	FN	TN	RN
Σ	P	N	

some **evaluation metrics**
(based on the confusion table)

$$Accuracy = \frac{TP+TN}{P+N}$$

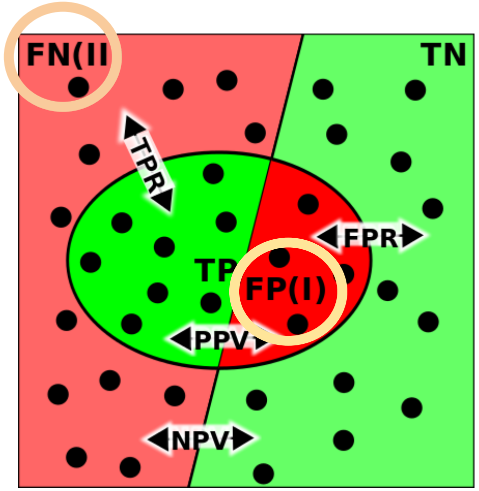
$$Error\ rate = \frac{FP+FN}{P+N}$$

$$Precision = \frac{TP}{RP}$$

$$Recall = \frac{TP}{P}$$

$$F_1 score = 2\frac{Precision \times Recall}{Precision + Recall}$$

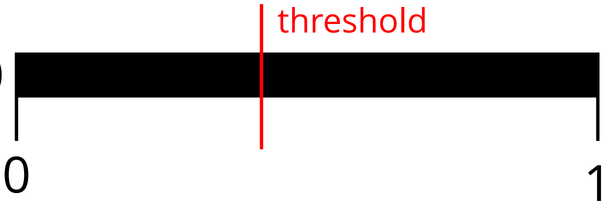
type I vs type II error



Evaluation

if we produce class score (probability)
we can trade-off between type I & type II error

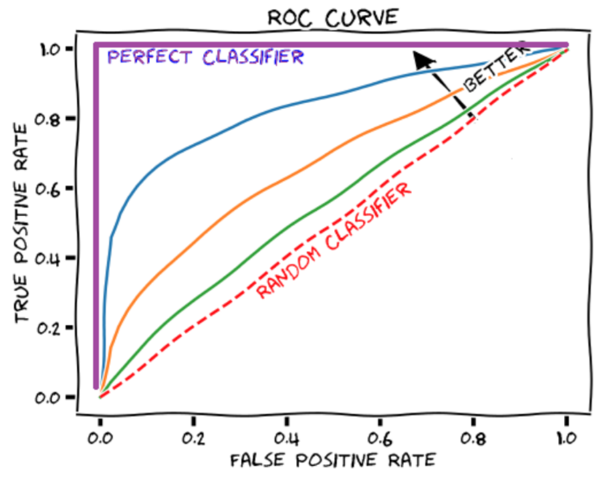
$p(y=1|x)$



goal: evaluate class scores/probabilities (independent of choice of threshold)

Receiver Operating Characteristic **ROC curve**

TPR = TP/P (**recall**, sensitivity)
FPR = FP/N (**fallout**, false alarm)



Summary

- complex models can have very different training and test error (*generalization gap*)
- regularization bounds this gap by penalizing model complexity
  - L1 & L2 regularization
  - probabilistic interpretation: different priors on weights
  - L1 produces sparse solutions (useful for feature selection)
- bias-variance trade off:
  - formalizes the relation between
    - training error (bias)
    - complexity (variance) and
    - and the test error (bias + variance)
  - not so elegant beyond L2 loss
- (cross) validation for model selection