

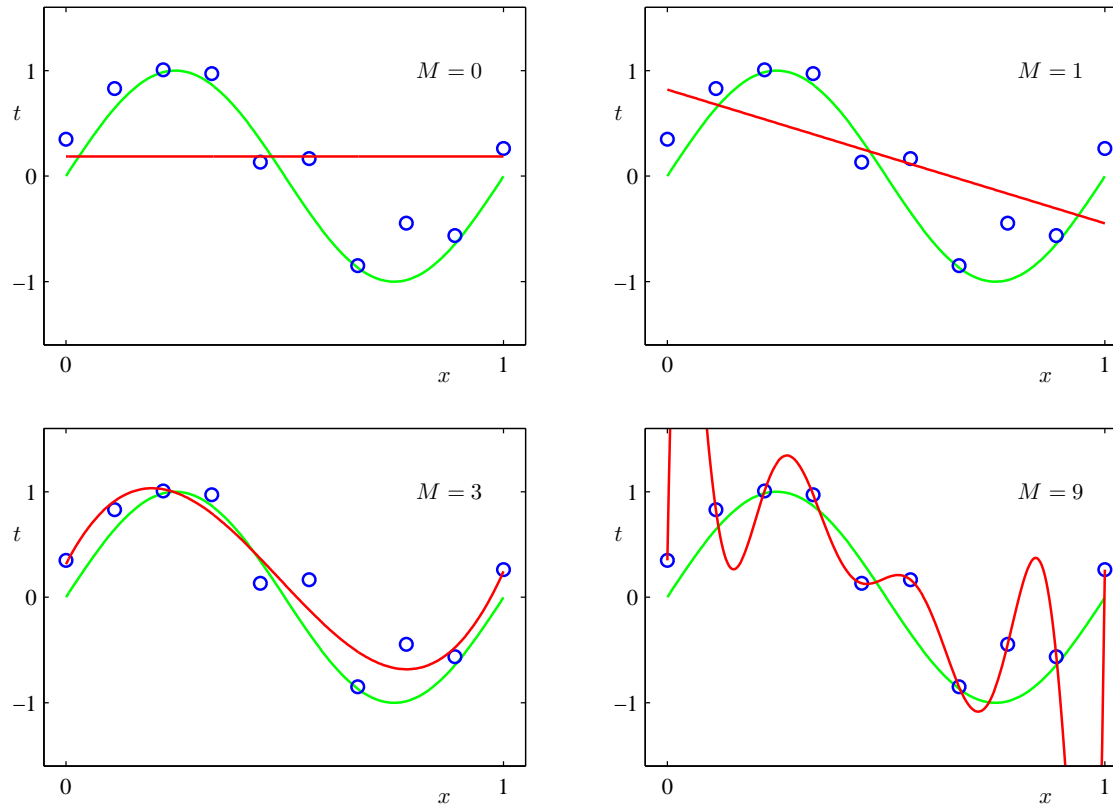
## Lecture 2: Overfitting. Regularization

- Overfitting
- Cross-validation
- L2 and L1 regularization for linear estimators
- Bias-variance trade-off

## Recall: Overfitting

- A general, *HUGELY IMPORTANT* problem for all machine learning algorithms
- We can find a hypothesis that predicts perfectly the training data but *does not generalize* well to new data
- E.g., a lookup table!

## Another overfitting example

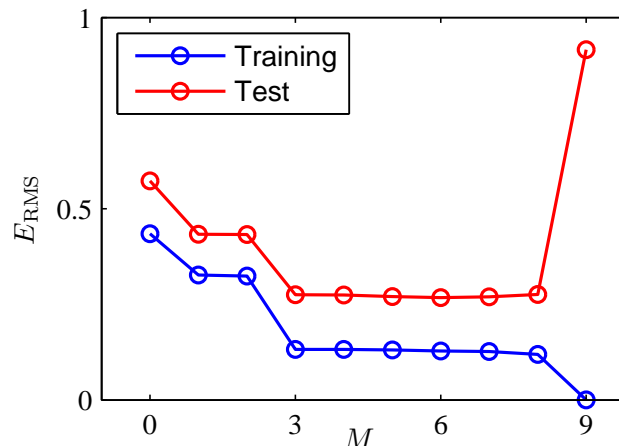


- The higher the degree of the polynomial  $M$ , the more degrees of freedom, and the more capacity to “overfit” the training data
- Typical overfitting means that error on the training data is very low, but error on new instances is high

## Overfitting more formally

- Assume that the data is drawn from some fixed, unknown probability distribution
- Every hypothesis has a "true" error  $J^*(h)$ , which is the expected error when data is drawn from the distribution.
- Because we do not have all the data, we measure the error on the training set  $J_D(h)$
- Suppose we compare hypotheses  $h_1$  and  $h_2$  on the training set, and  $J_D(h_1) < J_D(h_2)$
- If  $h_2$  is "truly" better, i.e.  $J^*(h_2) < J^*(h_1)$ , our algorithm is overfitting.
- We need theoretical and empirical methods to guard against it!

## Typical overfitting plot

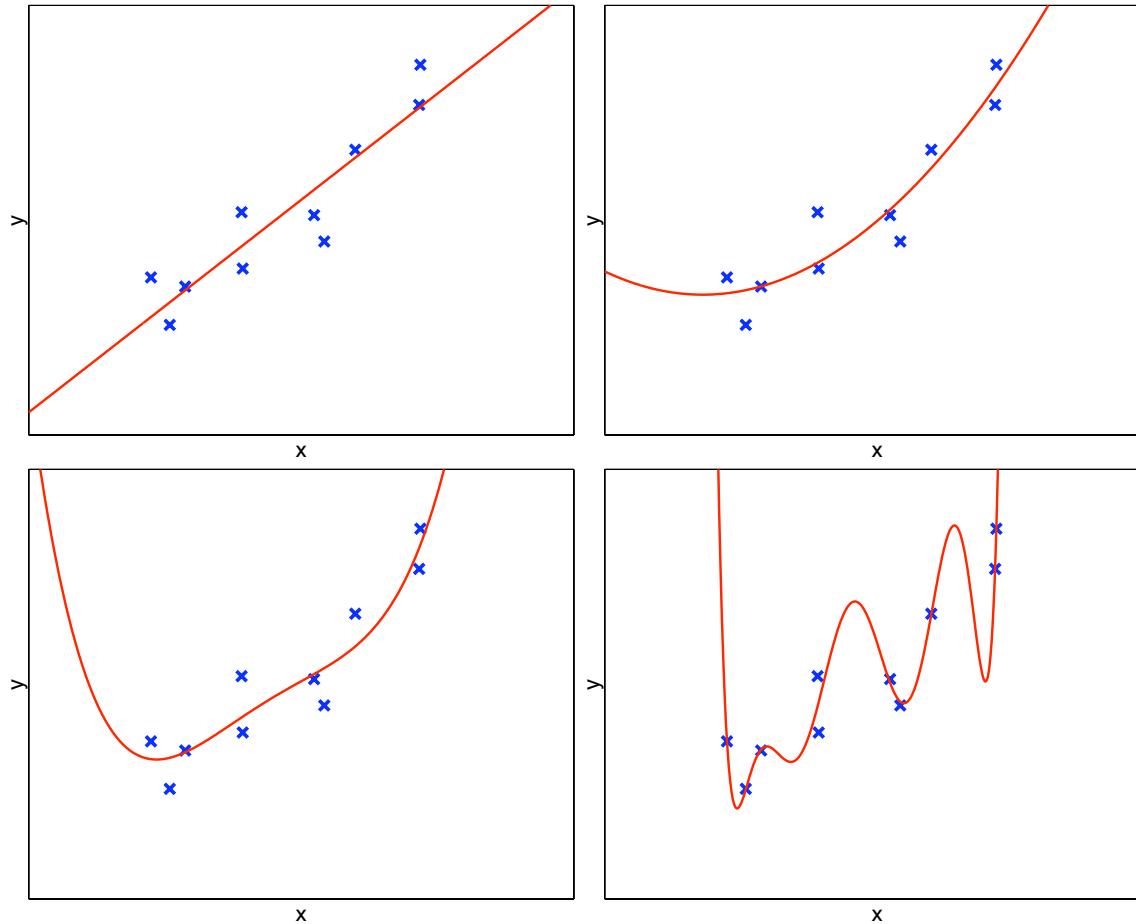


- The training error decreases with the degree of the polynomial  $M$ , i.e. *the complexity of the hypothesis*
- The testing error, measured on independent data, decreases at first, then starts increasing
- Cross-validation helps us:
  - Find a good hypothesis class ( $M$  in our case), using a *validation set of data*
  - Report unbiased results, using a *test set*, untouched during either parameter training or validation

# Cross-validation

- A general procedure for estimating the true error of a predictor
- The available data is split into two subsets:
  - A *training and validation set* used only to find the right predictor
  - A *test set* used to report the prediction error of the algorithm
- These sets *must be disjoint!*
- The process is repeated several times, and the results are averaged to provide error estimates.

# Model selection: Polynomial regression



# Model selection with leave-one-out cross-validation

1. For each order of polynomial,  $d$ :
  - (a) Repeat the following procedure  $m$  times:
    - i. Leave out  *$i$ th instance* from the training set, to estimate the true prediction error; we will put it in a *validation set*
    - ii. Use all the other instances to find best parameter vector,  $\mathbf{w}_{d,i}$
    - iii. Measure the error in predicting the label on the instance left out, for the  $\mathbf{w}_{d,i}$  parameter vector; call this  $J_{d,i}$
    - iv. This is a *(mostly) unbiased estimate of the true prediction error*
  - (b) Compute the average of the estimated errors:  $J_d = \frac{1}{m} \sum_{i=1}^m J_{d,i}$
2. Choose the  $d$  with lowest average estimated error:  $d^* = \arg \min_d J(d)$



## Estimating true error for $d = 1$

$D = \{(0.86, 2.49), (0.09, 0.83), (-0.85, -0.25), (0.87, 3.10), (-0.44, 0.87), (-0.43, 0.02), (-1.10, -0.12), (0.40, 1.81), (-0.96, -0.83), (0.17, 0.43)\}.$

Iter	$D_{train}$	$D_{valid}$	Error <sub>train</sub>	Error <sub>valid</sub> ( $J_{1,i}$ )
1	$D - \{(0.86, 2.49)\}$	(0.86, 2.49)	0.4928	0.0044
2	$D - \{(0.08, 0.83)\}$	(0.09, 0.83)	0.1995	0.1869
3	$D - \{(-0.85, -0.25)\}$	(-0.85, -0.25)	0.3461	0.0053
4	$D - \{(0.87, 3.10)\}$	(0.87, 3.10)	0.3887	0.8681
5	$D - \{(-0.44, 0.87)\}$	(-0.44, 0.87)	0.2128	0.3439
6	$D - \{(-0.43, 0.02)\}$	(-0.43, 0.02)	0.1996	0.1567
7	$D - \{(-1.10, -0.12)\}$	(-1.10, -0.12)	0.5707	0.7205
8	$D - \{(0.40, 1.81)\}$	(0.40, 1.81)	0.2661	0.0203
9	$D - \{(-0.96, -0.83)\}$	(-0.96, -0.83)	0.3604	0.2033
10	$D - \{(0.17, 0.43)\}$	(0.17, 0.43)	0.2138	1.0490
mean:			0.2188	0.3558

## Leave-one-out cross-validation results

$d$	Error <sub>train</sub>	Error <sub>valid</sub> ( $J_d$ )
1	0.2188	0.3558
2	0.1504	0.3095
3	0.1384	0.4764
4	0.1259	1.1770
5	0.0742	1.2828
6	0.0598	1.3896
7	0.0458	38.819
8	0.0000	6097.5
9	0.0000	6097.5

- Typical overfitting behavior: as  $d$  increases, the training error decreases, but the validation error decreases, then starts increasing again
- Optimal choice:  $d = 2$ . Overfitting for  $d > 2$

## Estimating both hypothesis class and true error

- Suppose we want to compare polynomial regression with some other algorithm
- We chose the hypothesis class (i.e. the degree of the polynomial,  $d^*$ ) based on the estimates  $J_d$
- Hence  $J_{d^*}$  is *not* unbiased - our procedure was aimed at optimizing it
- If we want to have both a hypothesis class *and* an unbiased error estimate, we need to tweak the leave-one-out procedure a bit

## Cross-validation with validation and testing sets

1. For each example  $j$ :
  - (a) Create a *test set* consisting just of the  $j$ th example,  $D_j = \{(\mathbf{x}_j, y_j)\}$  and a *training and validation set*  $\bar{D}_j = D - \{(\mathbf{x}_j, y_j)\}$
  - (b) Use the leave-one-out procedure from above on  $D_j$  to find a hypothesis,  $h_j^*$ 
    - Note that this will split the data internally, in order to both train and validate!
    - Typically, only one such split is used, rather than all possible splits
  - (c) Evaluate the error of  $h_j^*$  on  $D_j$  (call it  $J(h_j^*)$ )
2. Report the average of the  $J(h_j^*)$ , as a measure of performance of *the whole algorithm*

## Summary of leave-one-out cross-validation

- A very easy to implement algorithm
- Provides a great estimate of the true error of a predictor
- Computational cost scales with the number of instances (examples), so it can be prohibitive, especially if finding the best predictor is expensive
- Alternatives:
  - Leave- $k$ -out generalizes LOO, computationally very expensive.
  - $k$ -fold cross-validation: split the data set into  $k$  parts, then proceed as above.

# Regularization

- Remember the intuition: complicated hypotheses lead to overfitting
- Idea: change the error function to *penalize hypothesis complexity*:

$$J(\mathbf{w}) = J_D(\mathbf{w}) + \lambda J_{pen}(\mathbf{w})$$

This is called *regularization* in machine learning and *shrinkage* in statistics

- $\lambda$  is called *regularization coefficient* and controls how much we value fitting the data well, vs. a simple hypothesis

## $L_2$ regularization for linear models

- $L_2$  regularization (or *weight decay* in neural networks): add a squared penalty on the weights:

$$J_\lambda(\mathbf{w}) = \frac{1}{2}(\Phi\mathbf{w} - \mathbf{y})^\top(\Phi\mathbf{w} - \mathbf{y}) + \frac{\lambda}{2}\mathbf{w}^\top\mathbf{w}$$

- Why?
  - A simple hypothesis should not be too sensitive to small perturbation of the input
  - Math works out nicely
  - Resolve the issue of  $\Phi^\top\Phi$  not being invertible in linear regression, e.g. large  $n$  small  $m$  (original motivation)...

## $L_2$ regularization: closed form solution

$$J_\lambda(\mathbf{w}) = \frac{1}{2}(\Phi\mathbf{w} - \mathbf{y})^\top(\Phi\mathbf{w} - \mathbf{y}) + \frac{\lambda}{2}\mathbf{w}^\top\mathbf{w}$$

- By re-grouping terms, we get:

$$J_\lambda(\mathbf{w}) = \frac{1}{2}(\mathbf{w}^\top(\Phi^\top\Phi + \lambda\mathbf{I})\mathbf{w} - \mathbf{w}^\top\Phi^\top\mathbf{y} - \mathbf{y}^\top\Phi\mathbf{w} + \mathbf{y}^\top\mathbf{y})$$

- Optimal solution (obtained by solving  $\nabla J_\lambda(\mathbf{w}) = 0$ )

$$\mathbf{w} = (\Phi^\top\Phi + \lambda\mathbf{I})^{-1}\Phi^\top\mathbf{y}$$

(observe that  $\Phi^\top\Phi + \lambda\mathbf{I}$  is invertible)



## Side note: data centering

- For linear regression, we incorporated the bias term in  $\mathbf{w} \in \mathbb{R}^{n+1}$  by adding a 1 to each input vector  $\mathbf{x} \in \mathbb{R}^n$ . Why cannot we do the same for regression with  $L_2$  regularization?
  - Instead we center the data, i.e. remove the means from inputs and outputs:
    - Let  $\bar{\mathbf{x}} = \frac{1}{m} \sum_{i=1}^m \mathbf{x}_i$  and  $\bar{y} = \frac{1}{m} \sum_{i=1}^m y_i$ .
    - Consider the new data set  $\{(\mathbf{x}_i - \bar{\mathbf{x}}, y_i - \bar{y})\}_{i=1}^m$ .
    - Solving the regularized regression problem on this new dataset is *equivalent* to solving the original problem
    - You can check that the bias term of the solution on this new data set is 0...
- ↪ exercise

## What $L_2$ regularization does

$$\arg \min_{\mathbf{w}} \frac{1}{2}(\Phi \mathbf{w} - \mathbf{y})^\top (\Phi \mathbf{w} - \mathbf{y}) + \frac{\lambda}{2} \mathbf{w}^\top \mathbf{w} = (\Phi^\top \Phi + \lambda \mathbf{I})^{-1} \Phi^\top \mathbf{y}$$

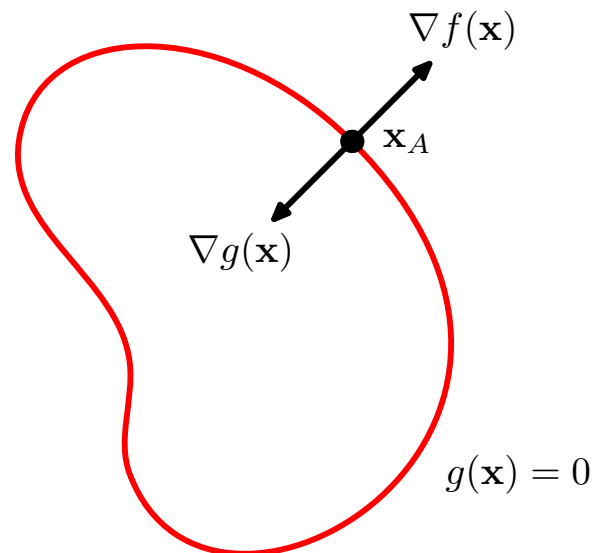
- If  $\lambda = 0$ , the solution is the same as in regular least-squares linear regression
- If  $\lambda \rightarrow \infty$ , the solution  $\mathbf{w} \rightarrow 0$
- Positive  $\lambda$  will cause the magnitude of the weights to be smaller than in the usual linear solution
- This is also called *ridge regression*, and it is a special case of Tikhonov regularization (more on that later)
- A different view of regularization: we want to optimize the error while keeping the  $L_2$  norm of the weights,  $\mathbf{w}^\top \mathbf{w}$ , bounded.

## Detour: Constrained optimization

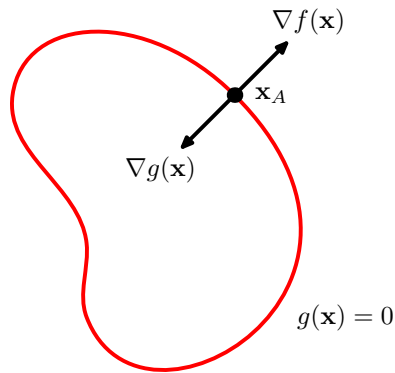
Suppose we want to find

$$\max_{\mathbf{x}} f(\mathbf{x})$$

such that  $g(\mathbf{x}) = 0$



## Detour: Lagrange multipliers



$\nabla g$  is orthogonal to the constraint surface (red curve)

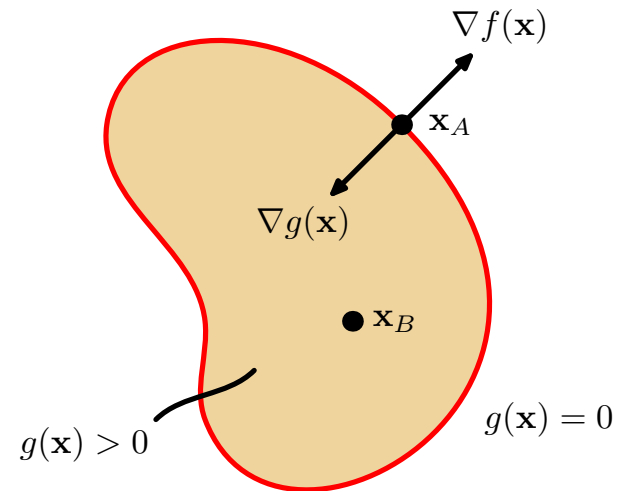
- At the optimum:
  - $\nabla f$  must be orthogonal to the surface (otherwise we could move along the curve to get a higher value of  $f$ )
  - $\Rightarrow \nabla f$  and  $\nabla g$  have to be parallel
  - $\Rightarrow$  There must exist some  $\lambda \in \mathbb{R}$  such that  $\nabla f + \lambda \nabla g = 0$
- **Lagrangian function:**  $L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x})$  ( $\lambda$  is called **Lagrange multiplier**)
- We obtain the solution to our optimization problem by setting both  $\nabla_{\mathbf{x}} L = 0$  and  $\frac{\partial L}{\partial \lambda} = 0$

## Detour: Inequality constraints

- Suppose we want to find

$$\max_{\mathbf{x}} f(\mathbf{x})$$

such that  $g(\mathbf{x}) \geq 0$



- In the interior,  $g(\mathbf{x}) > 0$ : simply find  $\nabla f(\mathbf{x}) = 0$  (in which case  $\lambda = 0$  in the Lagrangian)
- On the boundary,  $g(\mathbf{x}) = 0$ : same situation as before, but the sign matters this time
- For maximization, we want  $\nabla f$  pointing in the direction opposite to  $\nabla g$

## Detour: KKT conditions

- Consider again the Lagrangian

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x})$$

- Based on the previous observations, we minimize  $L$  wrt  $\mathbf{x}$  subject to the following constraints:

$$\lambda \geq 0$$

$$g(\mathbf{x}) \geq 0$$

$$\lambda g(\mathbf{x}) = 0$$

- These are called *Karush-Kuhn-Tucker (KKT) conditions*
- For minimization, simply flip the sign of  $\lambda$  in the Lagrangian:

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) - \lambda g(\mathbf{x})$$

## $L_2$ Regularization for linear models revisited

- Optimization problem: minimize error while keeping norm of the weights bounded

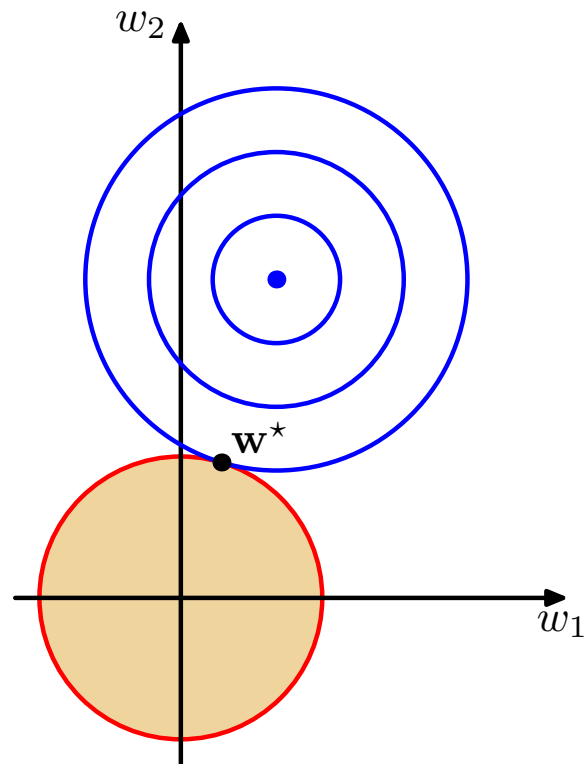
$$\begin{aligned} \min_{\mathbf{w}} J(\mathbf{w}) &= \min_{\mathbf{w}} (\Phi \mathbf{w} - \mathbf{y})^\top (\Phi \mathbf{w} - \mathbf{y}) \\ \text{such that } \mathbf{w}^\top \mathbf{w} &\leq \eta \end{aligned}$$

- The Lagrangian is:

$$L(\mathbf{w}, \nu) = J(\mathbf{w}) - \nu(\eta - \mathbf{w}^\top \mathbf{w})$$

- Fix some  $\lambda$  and let  $\mathbf{w}_\lambda$  be the solution of the penalized problem with parameter  $\lambda$ .
- You can check that if  $\eta = \|\mathbf{w}_\lambda\|^2$  and  $\nu = \lambda$ , then  $\mathbf{w}_\lambda$  satisfy the KKT conditions, hence the two problems have the same solution.

## Visualizing regularization (2 parameters)



$$\mathbf{w}^* = (\Phi^\top \Phi + \lambda \mathbf{I})^{-1} \Phi \mathbf{y}$$



## Pros and cons of $L_2$ regularization

- If  $\lambda$  is at a “good” value, regularization helps to avoid overfitting
- Choosing  $\lambda$  may be hard: cross-validation is often used
- If there are irrelevant features in the input (i.e. features that do not affect the output),  $L_2$  will give them small, but non-zero weights.
- Ideally, irrelevant input should have weights exactly equal to 0.

## $L_1$ Regularization for linear models

- Instead of requiring the  $L_2$  norm of the weight vector to be bounded, make the requirement on the  $L_1$  norm:

$$\min_{\mathbf{w}} J_D(\mathbf{w}) = \min_{\mathbf{w}} (\Phi \mathbf{w} - \mathbf{y})^\top (\Phi \mathbf{w} - \mathbf{y})$$

such that  $\sum_{i=1}^n |w_i| \leq \eta$

- This yields an algorithm called Lasso (Tibshirani, 1996)

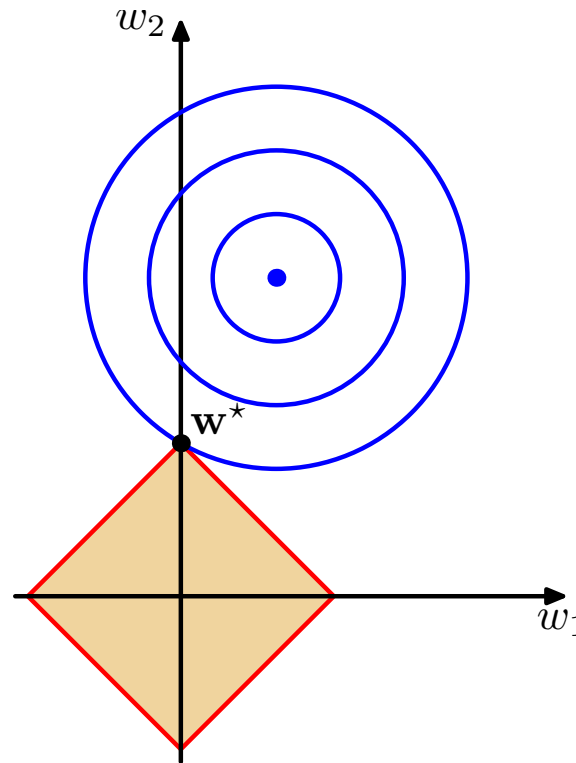
## Solving $L_1$ regularization

- The optimization problem is a quadratic program
- There is one constraint for each possible sign of the weights ( $2^n$  constraints for  $n$  weights)
- For example, with two weights:

$$\begin{aligned} \min_{w_1, w_2} \quad & \sum_{j=1}^m (y_j - w_1 x_1 - w_2 x_2)^2 \\ \text{such that } & w_1 + w_2 \leq \eta \\ & w_1 - w_2 \leq \eta \\ & -w_1 + w_2 \leq \eta \\ & -w_1 - w_2 \leq \eta \end{aligned}$$

- Solving this program directly can be done for problems with a small number of inputs

## Visualizing $L_1$ regularization

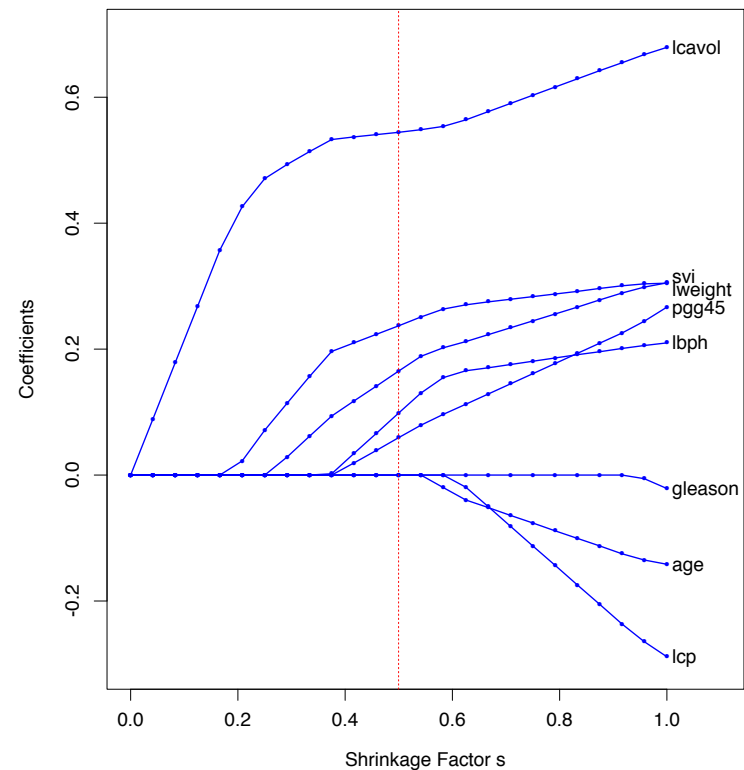
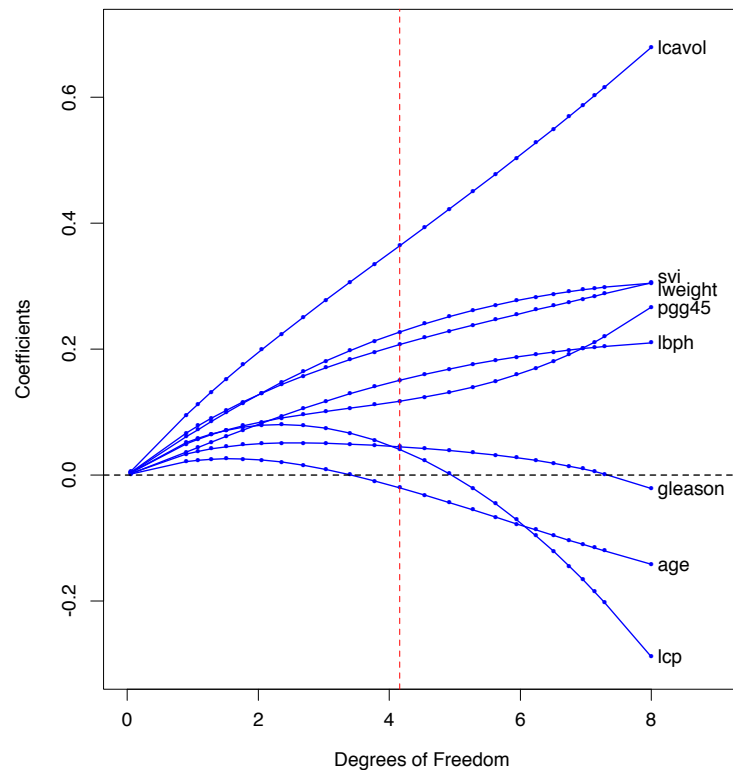


- If  $\lambda$  is big enough, the circle is very likely to intersect the diamond at one of the corners
- This makes  $L_1$  regularization much more likely to make some weights *exactly* 0

## Pros and cons of $L_1$ regularization

- If there are irrelevant input features, Lasso is likely to make their weights 0, while  $L_2$  is likely to just make all weights small
- Lasso is biased towards providing *sparse solutions* in general
- Lasso optimization is computationally more expensive than  $L_2$
- More efficient solution methods have to be used for large numbers of inputs (e.g. least-angle regression, 2003).
- $L_1$  methods of various types are very popular
- One can combine  $L_1$  and  $L_2$  regularization (elastic-net)

## Example of L1 vs L2 effect



- Note the sparsity in the coefficients induced by  $L_1$
- Lasso is an efficient way of performing the  $L_1$  optimization

## The anatomy of the error of an estimator

- Suppose we have a dataset  $\{(\mathbf{x}_i, y_i)\}_{i=1}^m$  where  $y = f(\mathbf{x}) + \epsilon$  and  $\epsilon$  is Gaussian noise with zero mean and standard deviation  $\sigma^2$
- We fit a linear hypothesis  $h(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$ , such as to minimize sum-squared error over the training data:

$$\sum_{i=1}^m (y_i - h(\mathbf{x}_i))^2$$

- Because of the hypothesis class that we chose (hypotheses linear in the parameters) for some target functions  $f$  we will have a *systematic prediction error*
- Even if  $f$  were truly from the hypothesis class we picked, depending on the data set we have, the parameters  $\mathbf{w}$  that we find may be different; this *variability* due to the specific data set on hand is a different source of error

## Bias-variance analysis

- Given a new data point  $\mathbf{x}$ , what is the *expected prediction error*?
- Assume that the data points are drawn *independently and identically distributed (i.i.d.)* from a unique underlying probability distribution  $P(\mathbf{x}, y) = P(\mathbf{x})P(y|\mathbf{x})$
- The goal of the analysis is to compute, for an arbitrary given point  $\mathbf{x}$ ,

$$E_P [(y - h(\mathbf{x}))^2 | \mathbf{x}]$$

where  $y$  is the value of  $\mathbf{x}$  in a data set, and the expectation is over all training sets of a given size, drawn according to  $P$

- For a given hypothesis class, we can also compute the *true error*, which is the expected error over the input distribution:

$$\sum_{\mathbf{x}} E_P [(y - h(\mathbf{x}))^2 | \mathbf{x}] P(\mathbf{x})$$

(if  $\mathbf{x}$  continuous, sum becomes integral with appropriate conditions).

- We will decompose this expectation into three components



## Recall: Statistics 101

- Let  $X$  be a random variable with possible values  $x_i, i = 1 \dots n$  and with probability distribution  $P(X)$
- The *expected value* or *mean* of  $X$  is:

$$E[X] = \sum_{i=1}^n x_i P(x_i)$$

- If  $X$  is continuous, roughly speaking, the sum is replaced by an integral, and the distribution by a density function
- The *variance* of  $X$  is:

$$\begin{aligned} Var[X] &= E[(X - E(X))^2] \\ &= E[X^2] - (E[X])^2 \end{aligned}$$

## The variance lemma

$$\begin{aligned} \text{Var}[X] &= E[(X - E[X])^2] \\ &= \sum_{i=1}^n (x_i - E[X])^2 P(x_i) \\ &= \sum_{i=1}^n (x_i^2 - 2x_i E[X] + (E[X])^2) P(x_i) \\ &= \sum_{i=1}^n x_i^2 P(x_i) - 2E[X] \sum_{i=1}^n x_i P(x_i) + (E[X])^2 \sum_{i=1}^n P(x_i) \\ &= E[X^2] - 2E[X]E[X] + (E[X])^2 \cdot 1 \\ &= E[X^2] - (E[X])^2 \end{aligned}$$

We will use the form:

$$E[X^2] = (E[X])^2 + \text{Var}[X]$$

## Bias-variance decomposition

- Simple algebra:

$$\begin{aligned} E_P [(y - h(\mathbf{x}))^2 | \mathbf{x}] &= E_P [(h(\mathbf{x}))^2 - 2yh(\mathbf{x}) + y^2 | \mathbf{x}] \\ &= E_P [(h(\mathbf{x}))^2 | \mathbf{x}] + E_P [y^2 | \mathbf{x}] - 2E_P [y | \mathbf{x}] E_P [h(\mathbf{x}) | \mathbf{x}] \end{aligned}$$

- Let  $\bar{h}(\mathbf{x}) = E_P[h(\mathbf{x}) | \mathbf{x}]$  denote the *mean prediction* of the hypothesis at  $\mathbf{x}$ , when  $h$  is trained with data drawn from  $P$
- For the first term, using the variance lemma, we have:

$$E_P[(h(\mathbf{x}))^2 | \mathbf{x}] = E_P[(h(\mathbf{x}) - \bar{h}(\mathbf{x}))^2 | \mathbf{x}] + (\bar{h}(\mathbf{x}))^2$$

- Note that  $E_P[y | \mathbf{x}] = E_P[f(\mathbf{x}) + \epsilon | \mathbf{x}] = f(\mathbf{x})$  (because of linearity of expectation and the assumption on  $\epsilon \sim \mathcal{N}(0, \sigma)$ )
- For the second term, using the variance lemma, we have:

$$E[y^2 | \mathbf{x}] = E[(y - f(\mathbf{x}))^2 | \mathbf{x}] + (f(\mathbf{x}))^2$$

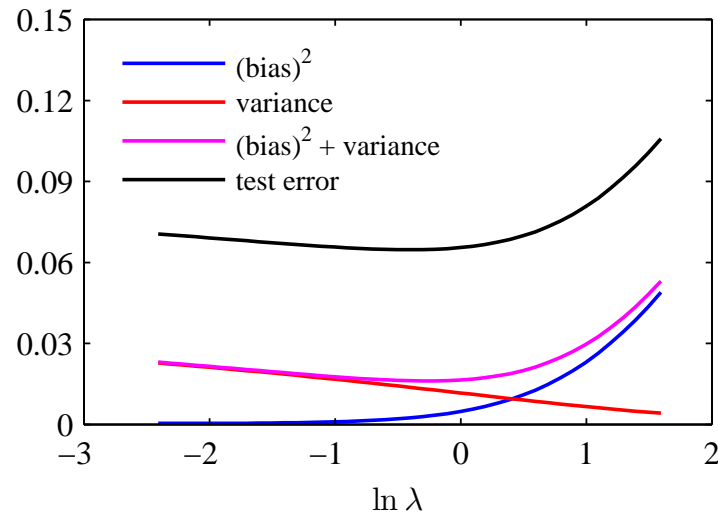
## Bias-variance decomposition (2)

- Putting everything together, we have:

$$\begin{aligned} E_P [(y - h(\mathbf{x}))^2 | \mathbf{x}] &= E_P [(h(\mathbf{x}) - \bar{h}(\mathbf{x}))^2 | \mathbf{x}] + (\bar{h}(\mathbf{x}))^2 - 2f(\mathbf{x})\bar{h}(\mathbf{x}) \\ &\quad + E_P [(y - f(\mathbf{x}))^2 | \mathbf{x}] + (f(\mathbf{x}))^2 \\ &= E_P [(h(\mathbf{x}) - \bar{h}(\mathbf{x}))^2 | \mathbf{x}] + (f(\mathbf{x}) - \bar{h}(\mathbf{x}))^2 + E[(y - f(\mathbf{x}))^2 | \mathbf{x}] \end{aligned}$$

- The first term,  $E_P[(h(\mathbf{x}) - \bar{h}(\mathbf{x}))^2 | \mathbf{x}]$ , is the *variance* of the hypothesis  $h$  at  $\mathbf{x}$ , when trained with finite data sets sampled randomly from  $P$
- The second term,  $(f(\mathbf{x}) - \bar{h}(\mathbf{x}))^2$ , is the *squared bias* (or systematic error) which is associated with the class of hypotheses we are considering
- The last term,  $E[(y - f(\mathbf{x}))^2 | \mathbf{x}]$  is the *noise*, which is due to the problem at hand, and cannot be avoided

# Error decomposition



- The bias-variance sum approximates well the test error over a set of 1000 points
- x-axis measures the hypothesis complexity (decreasing left-to-right)
- Simple hypotheses usually have high bias (bias will be high at many points, so it will likely be high for many possible input distributions)
- Complex hypotheses have high variance: the hypothesis is very dependent on the data set on which it was trained.

# Bias-variance trade-off

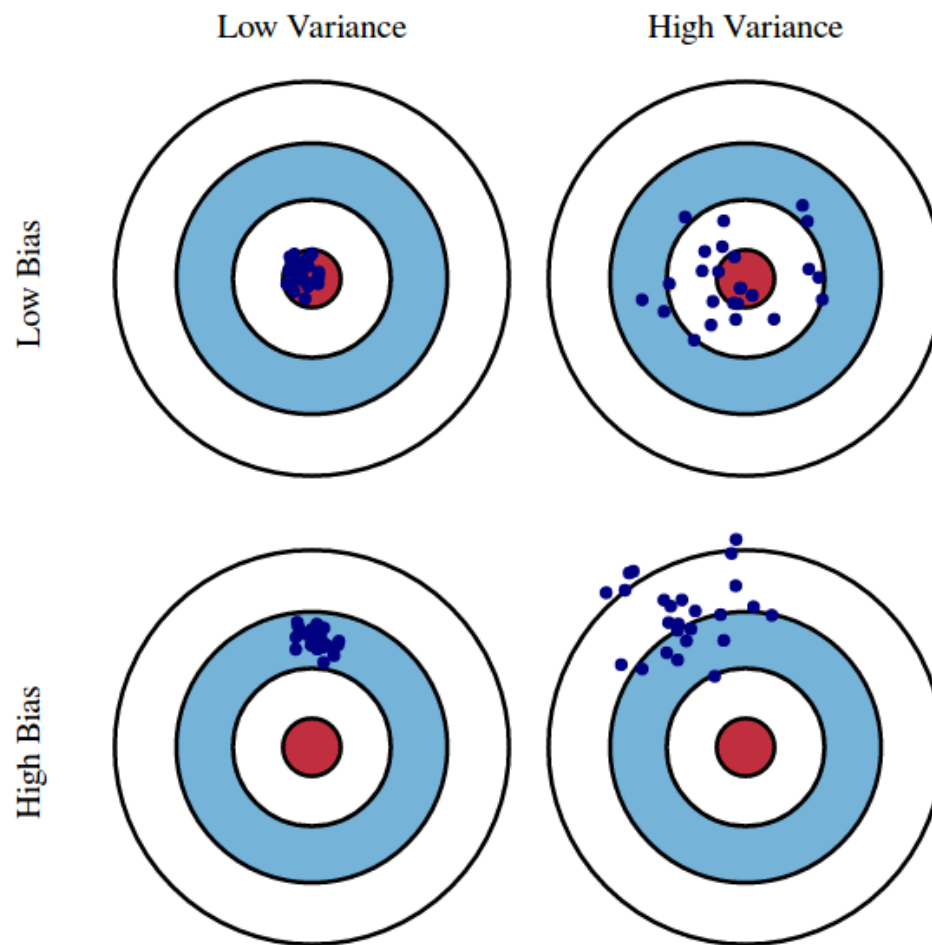


image credit: Scott Fortman-roe (<http://scott.fortmann-roe.com/docs/BiasVariance.html>)

## Bias-variance trade-off

- Typically, bias comes from not having good hypotheses in the considered class
- Variance results from the hypothesis class containing “too many” hypotheses
- Hence, we are faced with a *trade-off*: choose a more expressive class of hypotheses, which will generate higher variance, or a less expressive class, which will generate higher bias
- Making the trade-off has to depend on the amount of data available to fit the parameters (data usually mitigates the variance problem)

## More on overfitting

- Overfitting depends on the amount of data, relative to the complexity of the hypothesis
- With more data, we can explore more complex hypotheses spaces, and still find a good solution

