REGRESSION

MACHINE LEARNING

Task

Performance

Experience

Algorithms that improve their performance
at some task with experience
– Tom Mitchell (1998)

REGRESSION

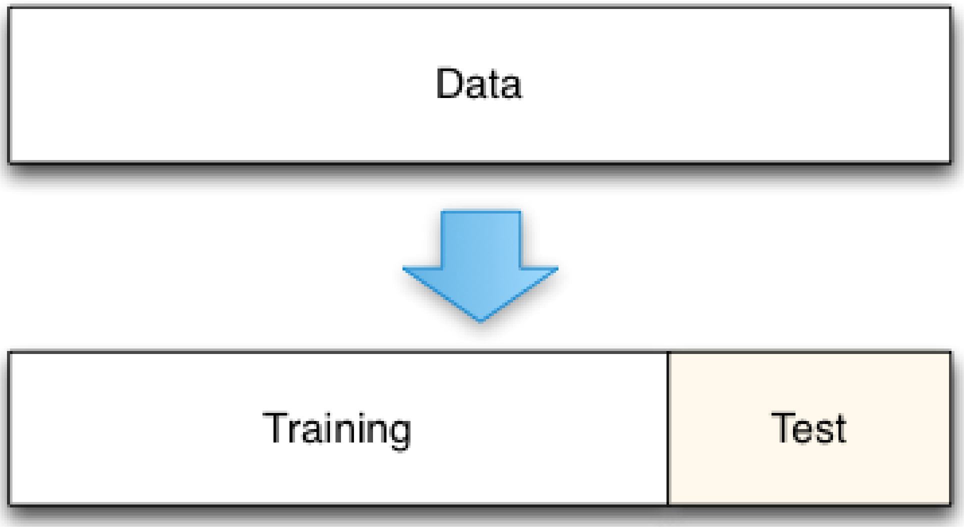
Machine Learning
> Supervised Learning
>> Regression

- **Task.** Find function $f: \mathbb{R}^d \to \mathbb{R}$ such that $y \approx f(x;\theta)$
- **Experience.** Training data $\left(x^{(1)}, y^{(1)}\right), \ldots, \left(x^{(n)}, y^{(n)}\right)$
- **Performance.** Prediction error $y - f(x;\theta)$ on test data

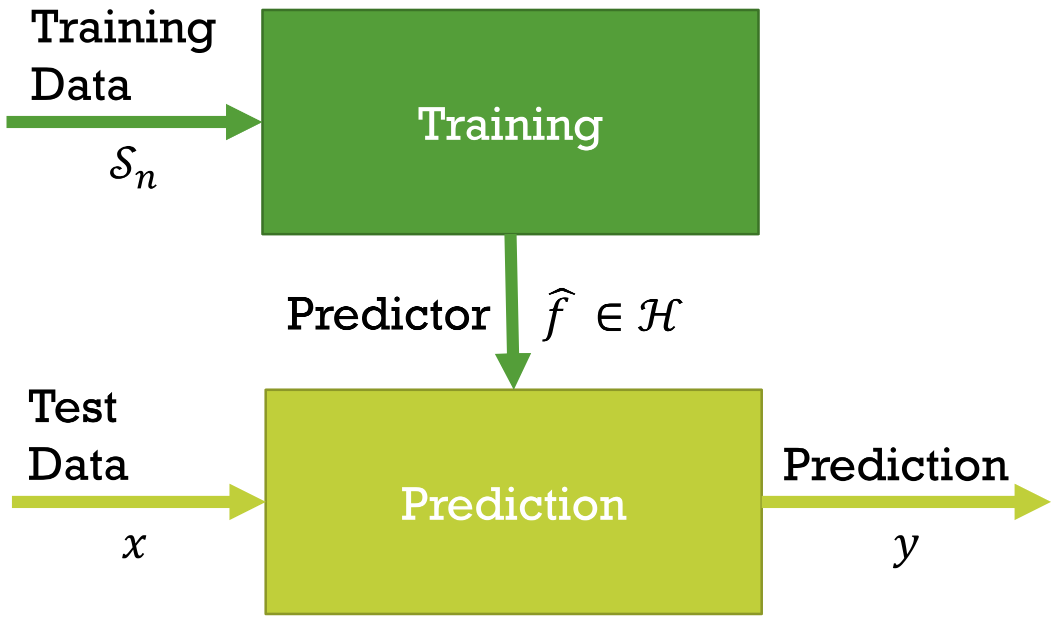
TRAINING DATA VS TEST DATA

Partition data into:

- Training data set $\mathcal{S}_n$
- Test data set $\mathcal{S}_*$



LEARNING AND PREDICTION



Assumption. Test data and training data are identically distributed.

GENERALIZATION

The goal of machine learning is to find a predictor $\hat{f} \in \mathcal{H}$ that generalizes well, i.e. that predicts well on test data $\mathcal{S}_*$.

MULTIVARIATE LINEAR REGRESSION

Given a training set $\{x^{(i)}\}_{i=1}^{m}$, where $x^{(i)} \in \mathbb{R}^n = \left(x_1^{(i)}, \dots, x_n^{(i)}\right)$ we define the design matrix as

$$X = \begin{bmatrix} x_1^{(1)} & \cdots & x_n^{(1)} \\ \vdots & \ddots & \vdots \\ x_1^{(m)} & \cdots & x_n^{(m)} \end{bmatrix},$$

and denote the parameters $\theta \in \mathbb{R}^n$ and target values $y \in \mathbb{R}^m$ as

$$\theta = \begin{bmatrix} \theta^{(1)} \\ \vdots \\ \theta^{(n)} \end{bmatrix}, \qquad y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix}.$$

Then the least squares error

$$\mathcal{J}(\theta) = \frac{1}{2}\sum_{i=1}^{m}\left(\left\langle \theta,\, x^{(i)}\right\rangle - y^{(i)}\right)^2$$

can be written as $\frac{1}{2}\left\|X\theta - y\right\|^2 = \frac{1}{2}\left\langle X\theta - y, X\theta - y\right\rangle$, or alternatively as

$$\frac{1}{2}\left(X\theta - y\right)^T\left(X\theta - y\right).$$

Exact formula

- Recall from Systems World that finding the value of θ that minimizes the distance between $X\theta$ and y is the same as finding the projection of y onto the column space of X.
- This is the same as finding θ such that $X\theta - y$ is perpendicular to every column of X, i.e.

$$X^T(X\theta - y) = 0 \implies X^T X\theta = X^T y,$$

which gives $\theta = (X^T X)^{-1} X^T y$.

Gradient descent

- If the matrix X is large, inverting $X^T X$ will be computationally costly.
- Instead we can find the least squares error using gradient descent. First we compute the gradient of the loss function:

$$\begin{aligned}\frac{\partial \mathcal{J}(\theta)}{\partial \theta_j} &= \sum_{i=1}^{m} \left(\left\langle \theta, x^{(i)} \right\rangle - y^{(i)} \right) \frac{\partial \left\langle \theta, x^{(i)} \right\rangle}{\partial \theta_j} \\ &= \sum_{i=1}^{m} \left(\left\langle \theta, x^{(i)} \right\rangle - y^{(i)} \right) \frac{\partial}{\partial \theta_j} \sum_{k=1}^{n} x_k^{(i)} \theta_k \\ &= \sum_{i=1}^{m} \left(\left\langle \theta, x^{(i)} \right\rangle - y^{(i)} \right) x_j^{(i)}\end{aligned}$$

Gradient descent cont.

- We then use it to perform the update rule

$$\theta_j(t+1) = \theta_j(t) - \alpha \sum_{i=1}^{m} \left(\left\langle \theta(t), x^{(i)} \right\rangle - y^{(i)} \right) x_j^{(i)}, \quad j = 1, \ldots, n$$

  until the change in the loss function is below a certain preset tolerance.

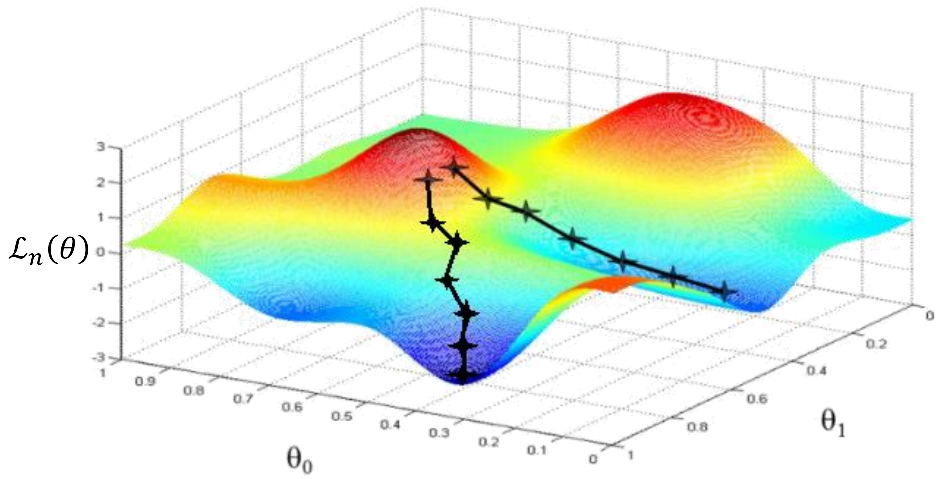
- In matrix notation the update rule can be written as

$$\theta(t+1) = \theta(t) - \alpha \left(X^T X \theta(t) - X^T y \right)$$

- This is also known as batch gradient descent, where all m training points are used at every step.
- For linear regression, the loss function is convex and has only one global minimum, so convergence is guaranteed unless the learning rate α is too large.

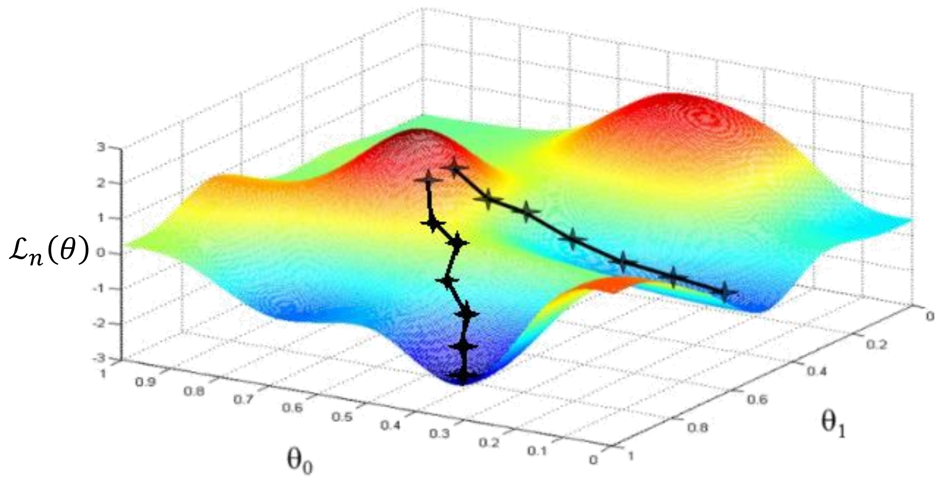
GRADIENT DESCENT

1. Initialize θ randomly.
2. Update $\theta \longleftarrow \theta - \eta_k \nabla \mathcal{L}_n(\theta)$, η_k learning rate, k iteration number.
3. Repeat (2) until convergence.
   (e.g. when improvement in $\mathcal{L}_n(\theta)$ is small enough)



LOCAL MINIMA

- Gradient descent leads us to a local minimum, which is not necessarily the global minimum. Different starting points may lead to different local minima.
- Typically, we perform gradient descent from several starting points, and run through all the local minima to find the parameter that has the smallest training loss.



STOCHASTIC GRADIENT DESCENT

training gradient = average of point gradients

$$\nabla\mathcal{L}_n(\theta;\mathcal{S}_n) = \frac{1}{n}\sum_{(x,y)\in\mathcal{S}_n}\nabla\mathcal{L}_1(\theta;x,y)$$

This average can take a long time to compute for large data sets.

Trick

Estimate the gradient by averaging over a smaller *minibatch* (subset of the training data).

$$\begin{aligned}\nabla\mathcal{L}_n(\theta;\mathcal{S}_n) &\approx \nabla\mathcal{L}_m(\theta;\mathcal{B}_m)\\ &= \frac{1}{m}\sum_{(x,y)\in\mathcal{B}_m}\nabla\mathcal{L}_1(\theta;x,y)\end{aligned}$$

Probabilistic interpretation

Assume for each i that

$$y^{(i)} = \left\langle \theta, x^{(i)} \right\rangle + \epsilon^{(i)},$$

where $\epsilon^{(i)} \sim \mathcal{N}(0, \sigma^2)$ and are independent from one another. This implies that $y^{(i)} \Big| x^{(i)} \sim \mathcal{N}\left(\langle \theta, x^{(i)} \rangle, \sigma^2\right)$, i.e.

$$p_\theta\left(y^{(i)} \middle| x^{(i)}\right) = \frac{1}{\sigma\sqrt{2\pi}} \exp \frac{-\left(y^{(i)} - \langle \theta, x^{(i)} \rangle\right)^2}{2\sigma^2}.$$

Given this setup, we want to find the optimal value of θ which maximizes the likelihood function

$$\mathcal{L}(\theta) = \prod_{i=1}^{m} p_\theta \left(y^{(i)} \middle| x^{(i)} \right),$$

or equivalently the log-likelihood function (since log is a monotonic function)

$$\begin{aligned} \ell(\theta) &= \log \prod_{i=1}^{m} p_\theta \left(y^{(i)} \middle| x^{(i)} \right) \\ &= \sum_{i=1}^{m} \log p_\theta \left(y^{(i)} \middle| x^{(i)} \right) \\ &= m \log \frac{1}{\sigma\sqrt{2\pi}} - \frac{1}{2\sigma^2} \sum_{i=1}^{m} \left(y^{(i)} - \left\langle \theta, x^{(i)} \right\rangle \right)^2 . \end{aligned}$$

Since m, π and σ are constants, we find that maximizing the likelihood is the same as minimizing the least squares error

$$\sum_{i=1}^{m} \left(y^{(i)} - \left\langle \theta, x^{(i)} \right\rangle \right)^2,$$

and that the choice of σ was inconsequential.

Bias-Variance tradeoff

- In general, assume that the targets are given by

$$y^{(i)} = f(x^{(i)}) + \epsilon^{(i)},$$

  for some unknown function f of the inputs which represents the true model.
- We assume that the $\epsilon^{(i)}$'s have mean 0 and variance σ^2, and are independent from one another.

Given a fixed dataset $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$, and denoting $\hat{f}_{\mathcal{D}}$ as the learnt hypothesis function (which depends on $\mathcal{D}$), the mean-squared error is given by

$$\frac{1}{n} \sum_{x^{(i)} \in \mathcal{D}} \int_{\mathbb{R}} \left(\hat{f}_{\mathcal{D}}(x^{(i)}) - y^{(i)} \right)^2 p(\epsilon^{(i)}) \, \mathrm{d}\epsilon^{(i)}.$$

This expression can be simplified by noting that

$$\begin{aligned}
&\int_{\mathbb{R}} \left(\hat{f}_{\mathcal{D}}(x^{(i)}) - y^{(i)}\right)^2 p(\epsilon^{(i)})\,\mathrm{d}\epsilon^{(i)} \\
&\quad = \int_{\mathbb{R}} \left(\hat{f}_{\mathcal{D}}(x^{(i)}) - f(x^{(i)}) - \epsilon^{(i)}\right)^2 p(\epsilon^{(i)})\,\mathrm{d}\epsilon^{(i)} \\
&\quad = \left(\hat{f}_{\mathcal{D}}(x^{(i)}) - f(x^{(i)})\right)^2 + \left(\hat{f}_{\mathcal{D}}(x^{(i)}) - f(x^{(i)})\right) \int_{\mathbb{R}} \epsilon_i p(\epsilon^{(i)})\,\mathrm{d}\epsilon^{(i)} \\
&\qquad\quad + \int_{\mathbb{R}} \epsilon_i^2 p(\epsilon^{(i)})\,\mathrm{d}\epsilon^{(i)} \\
&\quad = \left(\hat{f}_{\mathcal{D}}(x^{(i)}) - f(x^{(i)})\right)^2 + \sigma^2,
\end{aligned}$$

and hence

$$\frac{1}{n} \sum_{x^{(i)} \in \mathcal{D}} \int_{\mathbb{R}} \left(\hat{f}_{\mathcal{D}}(x^{(i)}) - y^{(i)}\right)^2 p(\epsilon^{(i)})\,\mathrm{d}\epsilon^{(i)} = \sigma^2 + \frac{1}{n} \sum_{x^{(i)} \in \mathcal{D}} \left(\hat{f}_{\mathcal{D}}(x^{(i)}) - f(x^{(i)})\right)^2$$

We would like to average this over all possible datasets to measure the performance of our learning algorithm. To model this, we can treat $\mathcal{D}$ as a random variable drawn from some distribution $p(\mathcal{D})$, and denote the expected error as

$$\int \sigma^2 + \frac{1}{n} \sum_{x^{(i)} \in \mathcal{D}} \left(\hat{f}_{\mathcal{D}}(x^{(i)}) - f(x^{(i)}) \right)^2 p(\mathcal{D})\, \mathrm{d}\mathcal{D}$$

$$= \sigma^2 + \frac{1}{n} \sum_{x^{(i)} \in \mathcal{D}} \mathbb{E}_{\mathcal{D}} \left[\left(\hat{f}_{\mathcal{D}}(x^{(i)}) - f(x^{(i)}) \right)^2 \right].$$

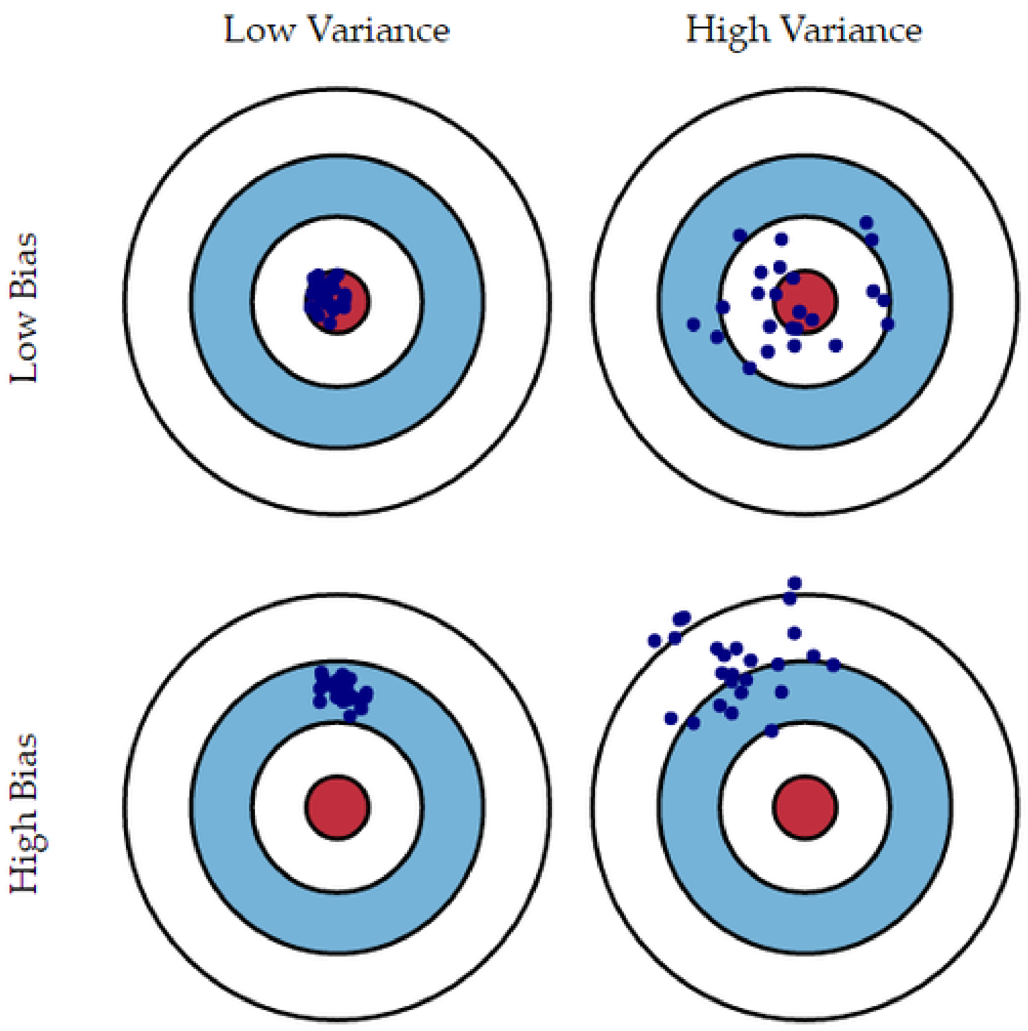
(we can bring the summation outside and sum over all $x^{(i)}$ by extending $\hat{f}_{\mathcal{D}}$ and f to be 0 if $x^{(i)}$ is not in $\mathcal{D}$)

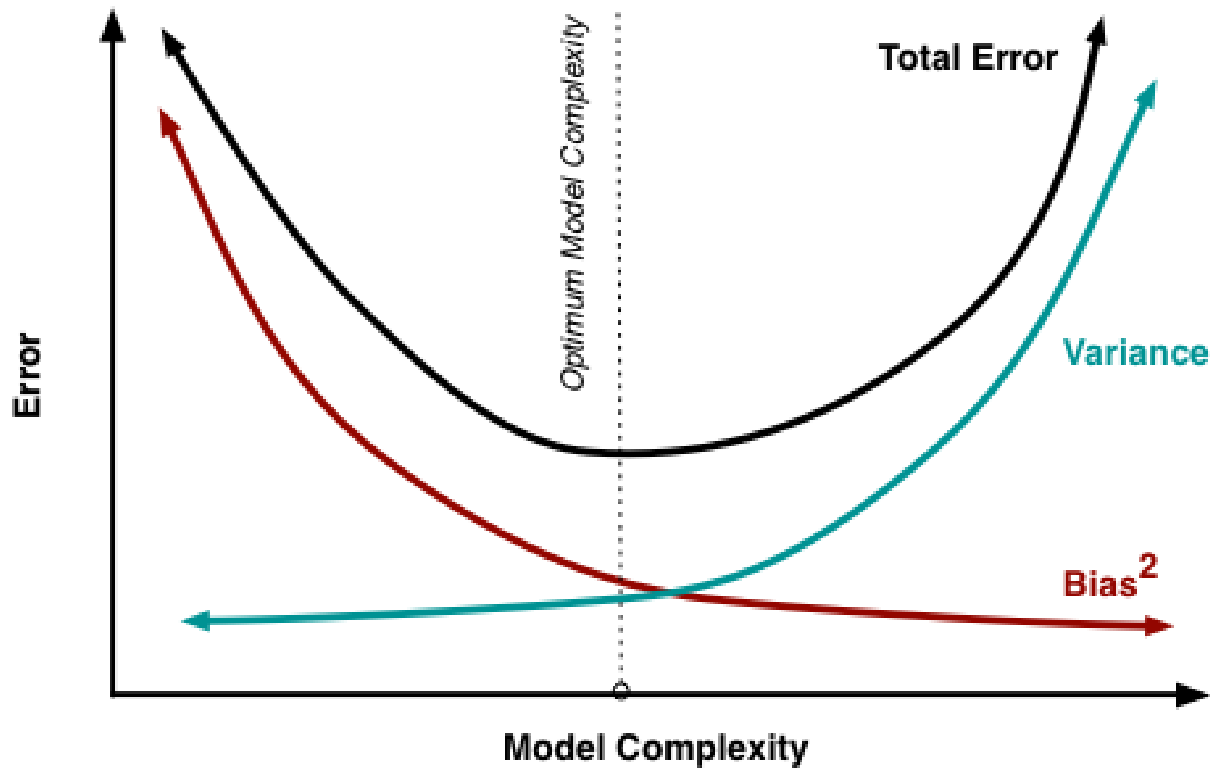
The expectation in the preceding line can be further decomposed as

$$
\begin{aligned}
&\mathbb{E}_{\mathcal{D}}\left[\left(\hat{f}_{\mathcal{D}}(x^{(i)}) - f(x^{(i)})\right)^2\right] \\
&\quad = \mathbb{E}_{\mathcal{D}}\left[\left(\hat{f}_{\mathcal{D}}(x^{(i)}) - \mathbb{E}_{\mathcal{D}}\left[\hat{f}_{\mathcal{D}}(x^{(i)})\right] + \mathbb{E}_{\mathcal{D}}\left[\hat{f}(x^{(i)})\right] - f(x^{(i)})\right)^2\right] \\
&\quad = \mathbb{E}_{\mathcal{D}}\left[\left(\hat{f}_{\mathcal{D}}(x^{(i)}) - \mathbb{E}_{\mathcal{D}}\left[\hat{f}_{\mathcal{D}}(x^{(i)})\right]\right)^2\right] \\
&\qquad\quad + 2\left(\mathbb{E}_{\mathcal{D}}\left[\hat{f}_{\mathcal{D}}(x^{(i)})\right] - f(x^{(i)})\right)\mathbb{E}_{\mathcal{D}}\left[\hat{f}_{\mathcal{D}}(x^{(i)}) - \mathbb{E}_{\mathcal{D}}\left[\hat{f}_{\mathcal{D}}(x^{(i)})\right]\right] \\
&\qquad\quad + \mathbb{E}_{\mathcal{D}}\left[\left(\mathbb{E}_{\mathcal{D}}\left[\hat{f}_{\mathcal{D}}(x^{(i)})\right] - f(x^{(i)})\right)^2\right] \\
&\quad = \mathbb{E}_{\mathcal{D}}\left[\left(\hat{f}_{\mathcal{D}}(x^{(i)}) - \mathbb{E}_{\mathcal{D}}\left[\hat{f}_{\mathcal{D}}(x^{(i)})\right]\right)^2\right] + \left(\mathbb{E}_{\mathcal{D}}\left[\hat{f}_{\mathcal{D}}(x^{(i)})\right] - f(x^{(i)})\right)^2.
\end{aligned}
$$

In conclusion, after summing over the $x^{(i)}$'s, the expected error can be written as

$$\underbrace{\left(\mathbb{E}\left[\hat{f}\right] - f\right)^2}_{\text{squared bias}} + \underbrace{\operatorname{Var}(\hat{f})}_{\text{variance}} + \underbrace{\sigma^2}_{\text{irreducible error}}.$$


Low Variance
High Variance
Low Bias
High Bias


Total Error
Variance
Bias^2
Optimum Model Complexity
Error
Model Complexity

5 min break

MODEL SELECTION

Overfitting. If model $\mathcal{H}$ is too big, then $\hat{f} \in \mathcal{H}$ performs

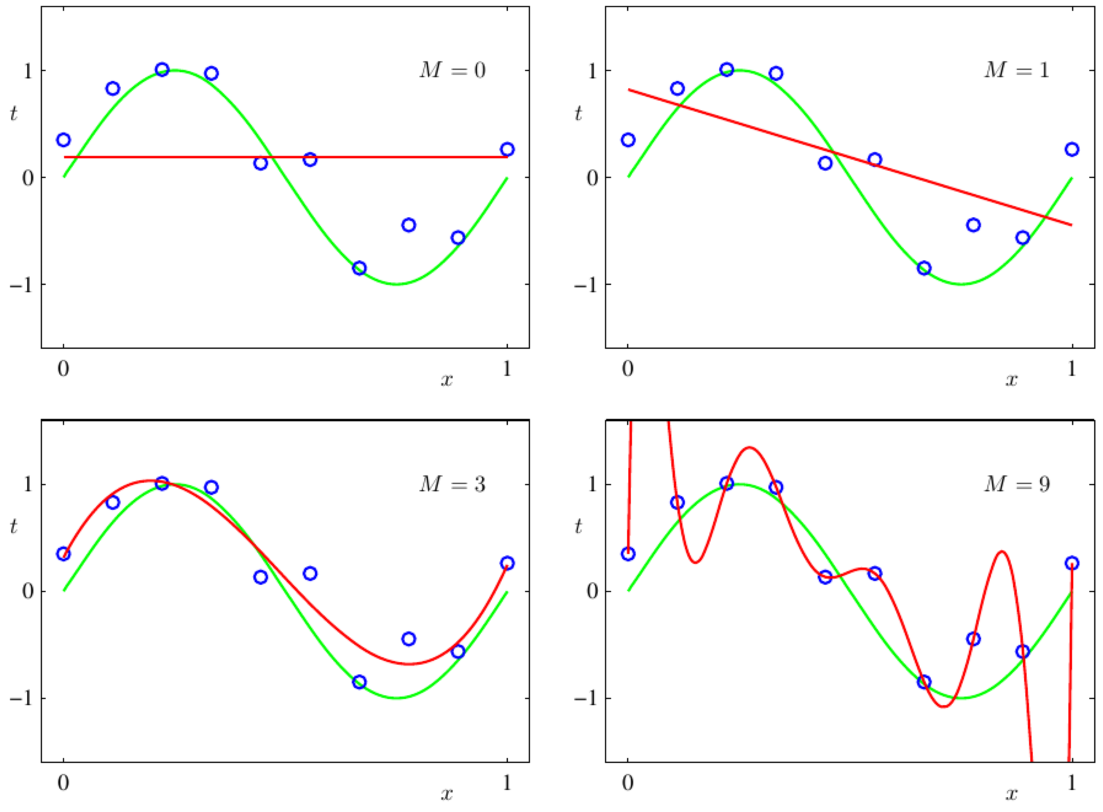
- well on training data, but poorly on test data.

Underfitting. If model $\mathcal{H}$ is too small, then $\hat{f} \in \mathcal{H}$ performs

- poorly on training data, and poorly on test data.

Finding a model with the right size is called model selection.

Model selection



Overfitting vs underfitting

Overfitting:

- Model fits the training data too well as it captures noise in addition to the underlying structure
- Low bias, high variance
- Training error low, testing error high

Underfitting:

- Model is too simple to capture the underlying structure of the data
- High bias, low variance
- Training error high, testing error high

Both lead to poor prediction performance on new input.

How to deal with underfitting or overfitting

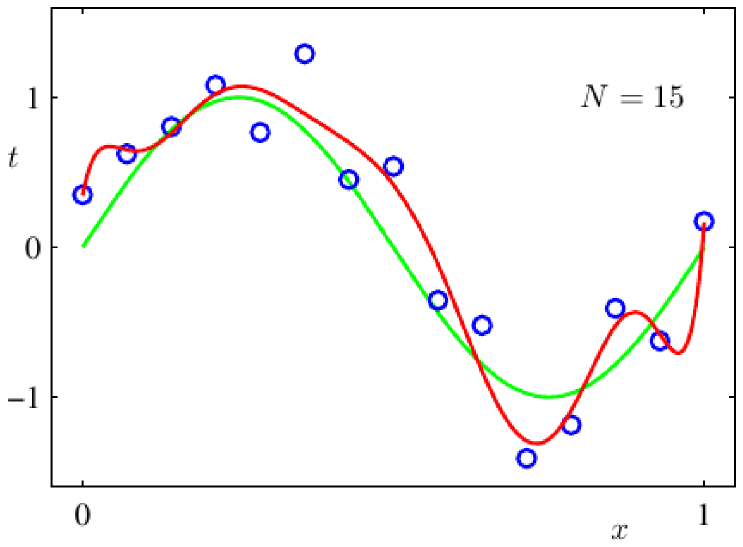
Underfitting:

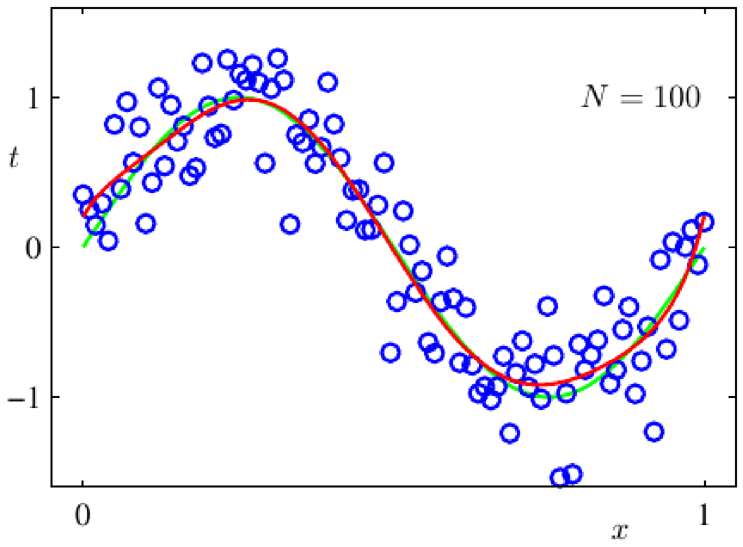
- Increase model complexity
    - Adjust regularization

Overfitting:

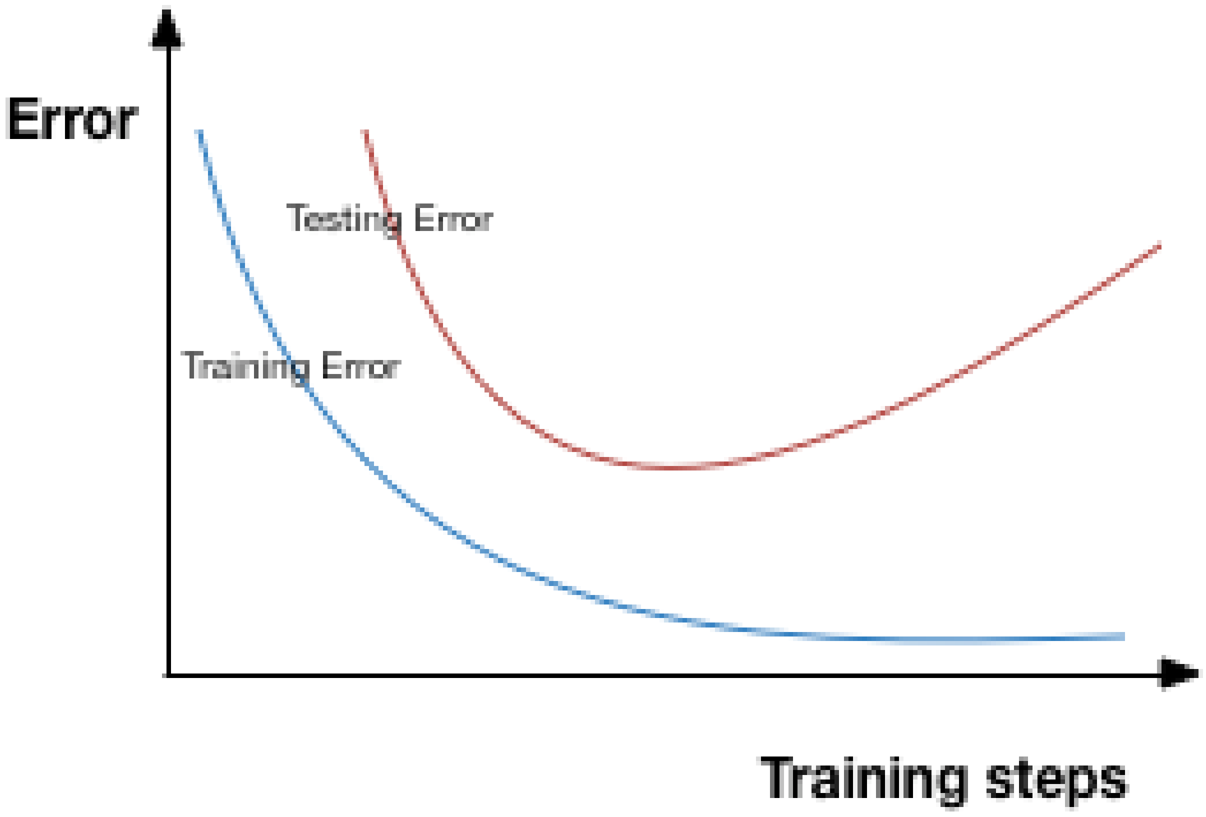
- Increase data size
- Early stopping
- Decrease model complexity
    - Adjust regularization

Overfitting: Increase data size





Overfitting: Early stopping



REGULARIZATION

RIDGE REGRESSION

Add a penalty.

$$\mathcal{L}_{n,\lambda}(\theta) = \frac{1}{n}\sum_{\text{data }(x,y)} \frac{1}{2}(y - \theta^\top x)^2 + \frac{\lambda}{2}\|\theta\|^2$$

Pressure to fit data

Pressure to go to zero

Regularization parameter $\lambda \geq 0$

Regularizer

(Unfortunately, to include the parameter θ_0, we cannot simply apply the constant feature trick. Why?)

TRAINING ALGORITHMS

Gradient

$$\nabla \mathcal{L}_{n,\lambda}(\theta) = \lambda\theta + \frac{1}{n}(X^\top X)\theta - \frac{1}{n}X^\top Y$$

Exact Solution

$$\nabla \mathcal{L}_{n,\lambda}(\hat{\theta}) = 0 \quad \Leftrightarrow \quad \lambda\hat{\theta} + \frac{1}{n}(X^\top X)\,\hat{\theta} = \frac{1}{n}X^\top Y$$

$$\Leftrightarrow \quad \hat{\theta} = (n\lambda I + X^\top X)^{-1}X^\top Y$$

This matrix is always invertible when $\lambda > 0$.

TRAINING ALGORITHMS

Gradient

$$\nabla \mathcal{L}_{n,\lambda}(\theta) = \lambda\theta + \frac{1}{n}(X^\top X)\theta - \frac{1}{n}X^\top Y$$

Gradient Descent

$$\theta \longleftarrow (1 - \eta_k \lambda)\,\theta - \eta_k \left[\frac{1}{n}(X^\top X)\theta - \frac{1}{n}X^\top Y\right]$$

Without regularization, i.e. $\lambda = 0$, this shrinkage factor equals 1.

TRAINING LOSS VS TEST LOSS

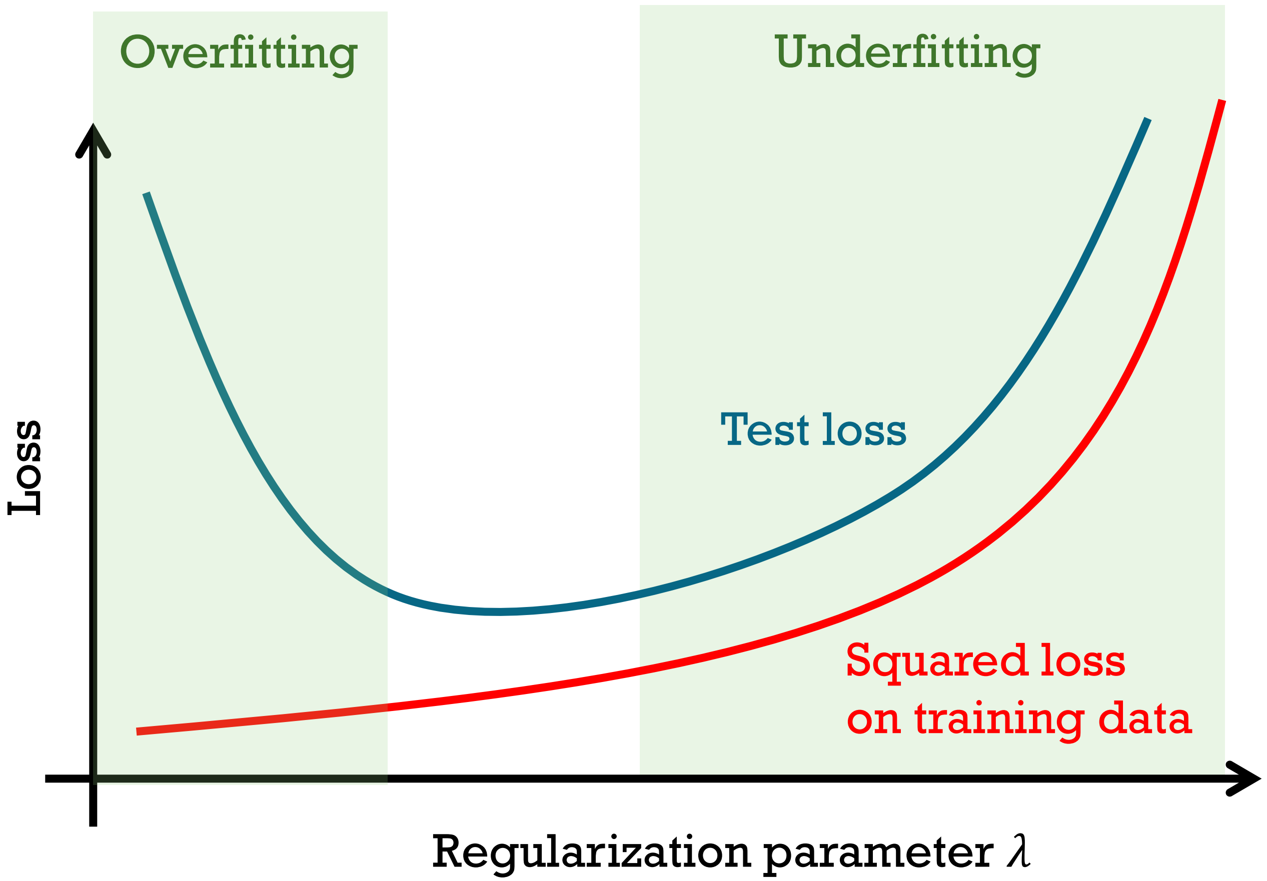
Training Loss

$$\mathcal{L}_{n,\lambda}(\theta) = \frac{1}{n}\sum_{\boxed{\text{trg data } (x,y)}} \frac{1}{2}(y - \theta^\top x)^2 + \boxed{\frac{\lambda}{2}\|\theta\|^2}$$

Test Loss

$$\mathcal{R}(\theta) = \frac{1}{n}\sum_{\boxed{\text{test data } (x,y)}} \frac{1}{2}(y - \theta^\top x)^2$$

EFFECT OF REGULARIZATION



Ridge regression vs LASSO

- LASSO (Least absolute shrinkage and selection operator) is similar to ridge regression, in that we have regression analysis performed with regularization.
- The difference is that the L_1 norm (hence the name L_1 regularization) is used to constrain the weights:

$$\arg\min_{\theta} \sum_{i=1}^{n} \left(\left\langle \theta, x^{(i)} \right\rangle - y^{(i)} \right)^2 + \lambda \sum_{j} |\theta_j| \, ;$$

  in contrast to the L_2 norm we have been seeing with ridge regression.
- With LASSO, the constraint on the weights typically sets some components of θ to 0, which can lead to sparser solutions, feature selection and better interpretability of the model.

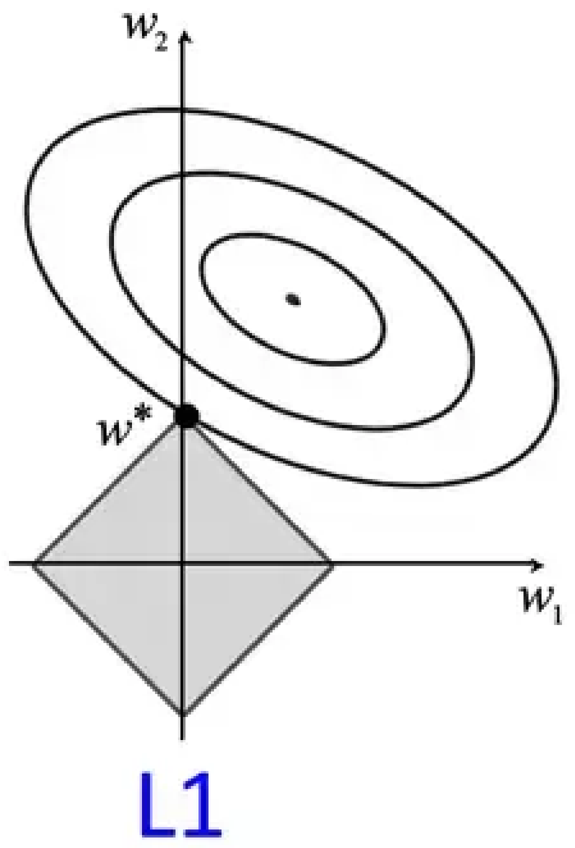
We can explain this by reformulating the problems in an alternative but equivalent form:

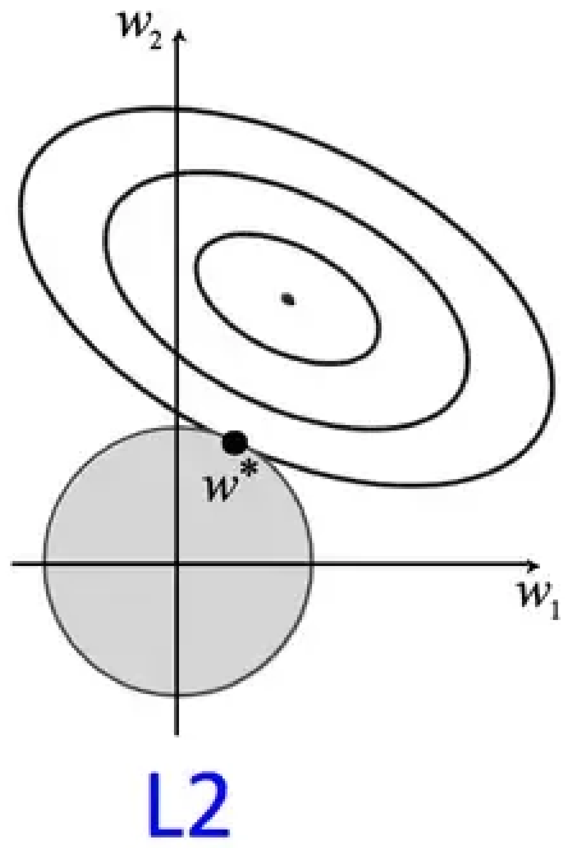
Ridge regression:

$$\begin{aligned} &\arg\min_{\theta} \sum_{i=1}^{n} \left(\left\langle \theta, x^{(i)} \right\rangle - y^{(i)} \right)^2 \\ &\text{subject to} \quad \sum_j \theta_j^2 \leq t \end{aligned}$$

LASSO:

$$\begin{aligned} &\arg\min_{\theta} \sum_{i=1}^{n} \left(\left\langle \theta, x^{(i)} \right\rangle - y^{(i)} \right)^2 \\ &\text{subject to} \quad \sum_j |\theta_j| \leq t \end{aligned}$$


w_2
w^*
w_1
L1


w_2
w^*
w_1
L2