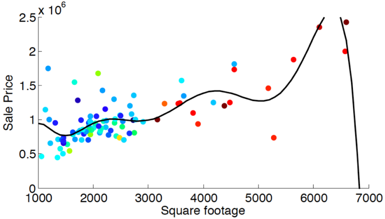
Statistical Machine Learning

Pier Francesco Palamara
Department of Statistics
University of Oxford

Slide credits and other course material can be found at:
http://www.stats.ox.ac.uk/~palamara/SML20_BDI.html

Last time: Overfitting, model selection

Fitting the housing price data with high order polynomials

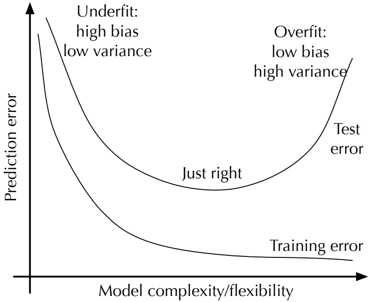


Note that the price would go to zero (or negative) if you buy bigger ones! **This is called poor generalization/overfitting.**

$$R(f) = R_N^{\text{emp}}(f) + \text{overfit penalty}.$$

- Cross-validation can be used to estimate $R(f)$ and select the adequate model complexity.
- Another possible strategy is to try to estimate the overfit penalty (e.g. via **regularization**).

Building models to trade bias with variance



- Building a machine learning model involves trading between its bias and variance. We will see many examples in the next lectures:
  - Bias reduction at the expense of a variance increase: building more complex models, e.g. adding nonlinear features and additional parameters, increasing the number of hidden units in neural nets, using decision trees with larger depth, decreasing the **regularization** parameter.
  - Variance reduction at the expense of a bias increase: early stopping, using k-nearest neighbours with larger k, increasing the **regularization** parameter.

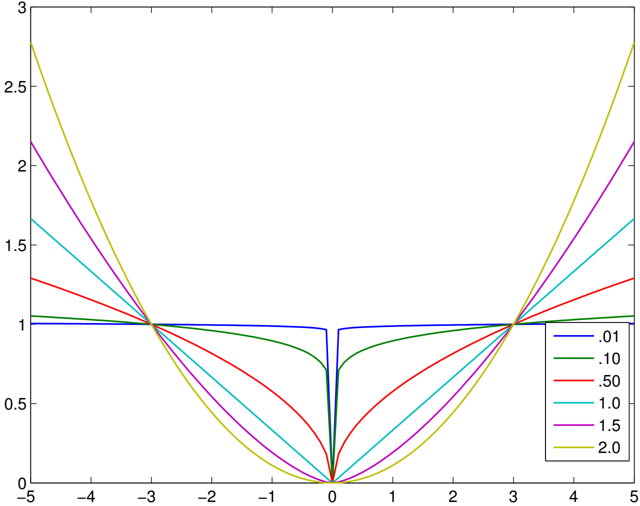
Regularization

- Flexible models for high-dimensional problems require many parameters.
- With many parameters, learners can easily overfit.
- **Regularization**: Limit flexibility of model to prevent overfitting.
- Add term **penalizing large values of parameters** θ.

$$\min_\theta R_N(f_\theta) + \lambda\|\theta\|_\rho^\rho = \min_\theta \frac{1}{N}\sum_{i=1}^{N} L(y_i, f_\theta(x_i)) + \lambda\|\theta\|_\rho^\rho$$

  where $\rho \geq 1$, and $\|\theta\|_\rho = (\sum_{j=1}^{p} |\theta_j|^\rho)^{1/\rho}$ is the L_ρ norm of θ (also of interest when $\rho \in [0, 1)$, but is no longer a norm).
- Also known as **shrinkage** methods—parameters are shrunk towards 0.
- λ is a **tuning parameter** (or **hyperparameter**) and controls the amount of regularization, and resulting complexity of the model.

Regularization

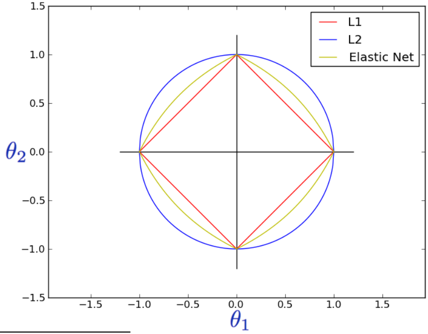


L_ρ regularization profile for different values of ρ.

Types of Regularization

- **Ridge regression / Tikhonov regularization**: $\rho = 2$ (Euclidean norm)
- **LASSO**: $\rho = 1$ (Manhattan norm)
- **Sparsity-inducing** regularization: $\rho \leq 1$ (nonconvex for $\rho < 1$)
- **Elastic net**[1] regularization: mixed L_1/L_2 penalty:

$$\min_{\theta} \frac{1}{N} \sum^{N} L(y_i, f_\theta(x_i)) + \lambda \left[(1-\alpha)\|\theta\|_2^2 + \alpha\|\theta\|_1\right]$$



[1] Figure source: http://scikit-learn.sourceforge.net

Regularized linear regression

A new loss or error function to minimize

$$R_N(\boldsymbol{\theta}, \theta_0) = \sum_n (y_n - \boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x}_n - \theta_0)^2 + \lambda \|\boldsymbol{\theta}\|_2^2$$

where $\lambda > 0$ controls the model complexity, “shrinking” weights towards 0.

- If $\lambda \to +\infty$, then
$$\widehat{\boldsymbol{\theta}} \to \mathbf{0}$$
- If $\lambda \to 0$, back to normal OLS (Ordinary Least Squares).

For regularized linear regression: the solution changes very little (in form) from the OLS solution

$$\operatorname{argmin} \sum_n (y_n - \boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x}_n - \theta_0)^2 + \lambda \|\boldsymbol{\theta}\|_2^2 \Rightarrow \widehat{\boldsymbol{\theta}} = \left(\boldsymbol{X}^{\mathrm{T}} \boldsymbol{X} + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{X}^{\mathrm{T}} \boldsymbol{y}$$

and reduces to the OLS solution when $\lambda = 0$, as expected.

As long as $\lambda \geq 0$, the optimization problem remains convex.

Example: overfitting with polynomials

Our regression model

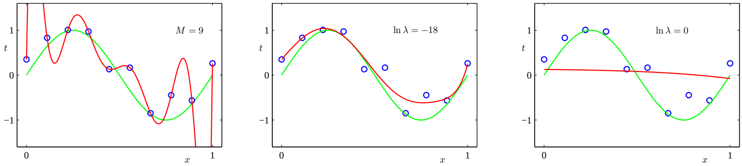
$$y = \sum_{m=1}^{M} \theta_m x^m$$

Regularization would discourage large parameter values as we saw with the OLS solution, thus potentially preventing overfitting.

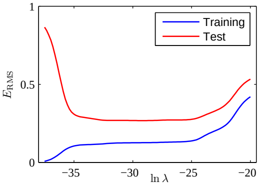
	$M=0$	$M=1$	$M=3$	$M=9$
θ_0	0.19	0.82	0.31	0.35
θ_1		-1.27	7.99	232.37
θ_2			-25.43	-5321.83
θ_3			17.37	48568.31
θ_4				-231639.30
θ_5				640042.26
θ_6				-1061800.52
θ_7				1042400.18
θ_8				-557682.99
θ_9				125201.43

Overfitting in terms of λ

Overfitting is reduced from complex model to simpler one with the help of increasing regularizers



λ vs. residual error shows the difference of the model performance on training and testing dataset



The effect of λ

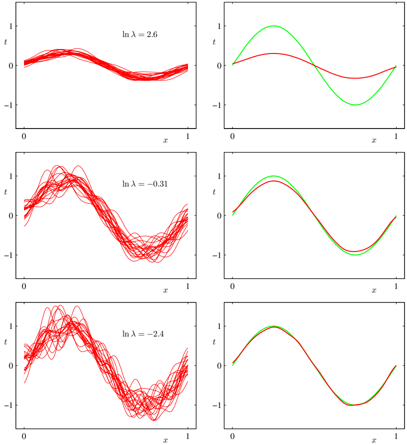
Large λ attenuates parameters towards 0

	$\ln\lambda = -\infty$	$\ln\lambda = -18$	$\ln\lambda = 0$
θ_0	0.35	0.35	0.13
θ_1	232.37	4.74	-0.05
θ_2	-5321.83	-0.77	-0.06
θ_3	48568.31	-31.97	-0.06
θ_4	-231639.30	-3.89	-0.03
θ_5	640042.26	55.28	-0.02
θ_6	-1061800.52	41.32	-0.01
θ_7	1042400.18	-45.95	-0.00
θ_8	-557682.99	-91.53	0.00
θ_9	125201.43	72.68	0.01

The effect of λ

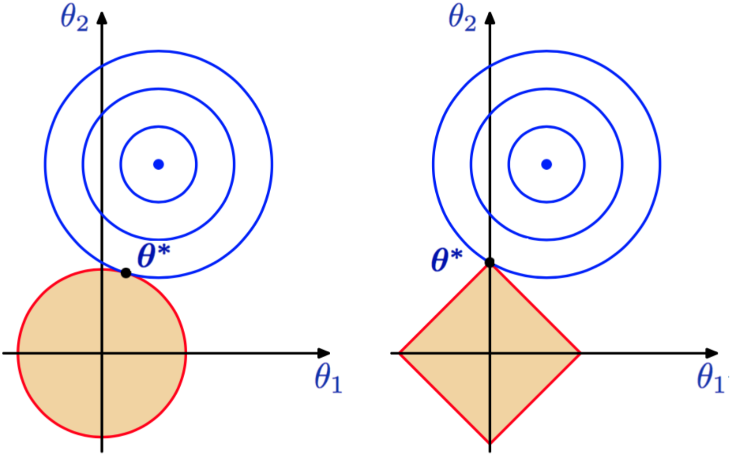
Increasing λ reduces variance (left) and increases bias (right)[2].

Variance | Bias



[2]Bishop PRML Figure 3.5

L_1 promotes sparsity



L_1 regularization often leads to optimal solutions with many zeros, i.e., the regression function depends only on the (small) number of features with non-zero parameters.

figure 3.4 of PRML.

Regularization in R demo

`http://www.stats.ox.ac.uk/~palamara/teaching/SML19/regularization.html`

What if $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ is not invertible?

Can you think of any reasons why that could happen?

Answer 1: $\mathsf{N} < \mathsf{D}$. Intuitively, not enough data to estimate all the parameters.

Answer 2: $\boldsymbol{X}$ columns are not linearly independent. Intuitively, there are two features that are perfectly correlated. In this case, solution is not unique.

Ridge regression

Intuition: what does a non-invertible $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ mean? Consider the SVD of this matrix:

$$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} = \boldsymbol{V}\begin{bmatrix} \lambda_1 & 0 & 0 & \cdots & 0 \\ 0 & \lambda_2 & 0 & \cdots & 0 \\ 0 & \cdots & \cdots & \cdots & 0 \\ 0 & \cdots & \cdots & \lambda_r & 0 \\ 0 & \cdots & \cdots & 0 & 0 \end{bmatrix}\boldsymbol{V}^{\top}$$

where $\lambda_1 \geq \lambda_2 \geq \cdots \lambda_r > 0$ and $r <$ D.

Regularization can fix this problem by ensuring all singular values are non-zero

$$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} + \lambda\boldsymbol{I} = \boldsymbol{V}\mathsf{diag}(\lambda_1 + \lambda, \lambda_2 + \lambda, \cdots, \lambda)\boldsymbol{V}^{\top}$$

where $\lambda > 0$ and $\boldsymbol{I}$ is the identity matrix

Computational complexity

Bottleneck of computing the solution? The OLS problem has a simple, closed-form solution. But computing it involves a number of matrix operations:

$$\boldsymbol{\theta} = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}$$

Matrix multiply of $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} \in \mathbb{R}^{(\mathsf{D}+1)\times(\mathsf{D}+1)}$
Inverting the matrix $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$

How many operations do we need?

- $O(\mathsf{ND}^2)$ for matrix multiplication
- $O(\mathsf{D}^3)$ (e.g., using Gauss-Jordan elimination) or $O(\mathsf{D}^{2.373})$ (recent theoretical advances) for matrix inversion
- Impractical for very large D or N
- As an alternative, we could use numerical methods. This type of approach is widely used in several other machine learning algorithms. These methods are often the only available option, since sometimes we don't have a closed form solution available.

Alternative method: an example of using numerical optimization

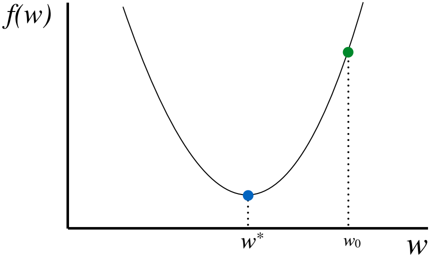
(Batch) Gradient descent

- Initialize $\boldsymbol{\theta}$ to $\boldsymbol{\theta}^{(0)}$ (e.g., randomly); set $t = 0$; choose $\eta > 0$
- Loop **until convergence**
  1. Compute the gradient $\nabla R_N(\boldsymbol{\theta}) = \boldsymbol{X}^{\mathrm{T}} \left(\boldsymbol{X}\boldsymbol{\theta}^{(t)} - \boldsymbol{y} \right)$
  2. Update the parameters $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \nabla R_N(\boldsymbol{\theta})$
  3. $t \leftarrow t + 1$

What is the complexity of each iteration?

Gradient Descent

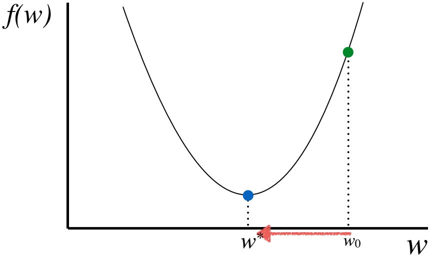
Start at a random point



Gradient Descent

Start at a random point

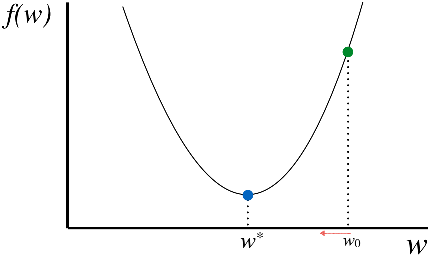
Determine a descent direction



Gradient Descent

Start at a random point

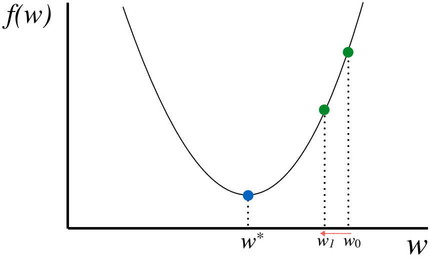
Determine a descent direction
Choose a step size



Gradient Descent

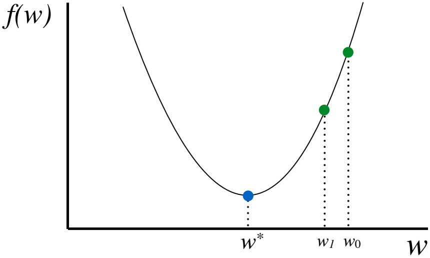
Start at a random point

Determine a descent direction
Choose a step size
Update



Gradient Descent

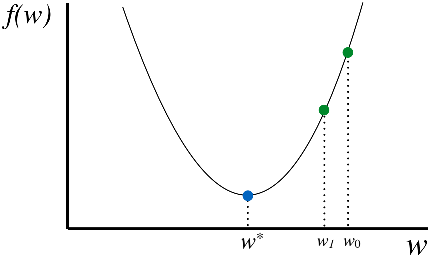
Start at a random point
Repeat
    Determine a descent direction
    Choose a step size
    Update
Until stopping criterion is satisfied



Gradient Descent

Start at a random point
Repeat
- Determine a descent direction
- Choose a step size
- Update

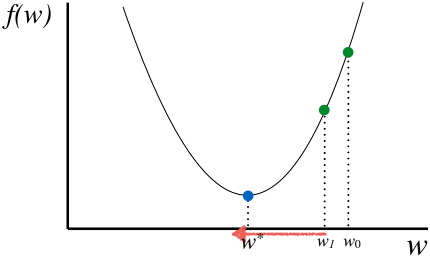
Until stopping criterion is satisfied



Gradient Descent

Start at a random point
Repeat
- Determine a descent direction
- Choose a step size
- Update

Until stopping criterion is satisfied



Gradient Descent

Start at a random point

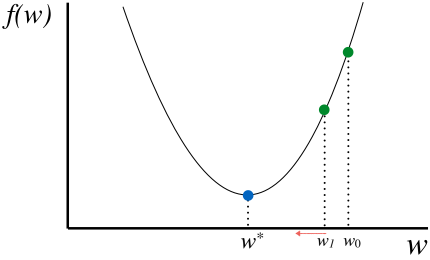
Repeat

Determine a descent direction

Choose a step size

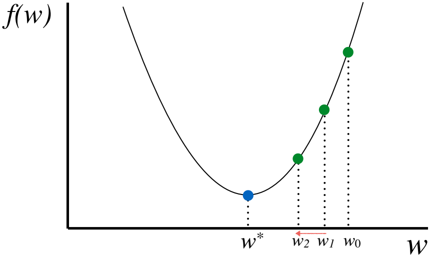
Update

Until stopping criterion is satisfied



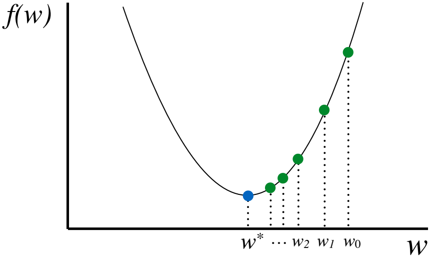
Gradient Descent

Start at a random point
Repeat
    Determine a descent direction
    Choose a step size
    Update
Until stopping criterion is satisfied



Gradient Descent

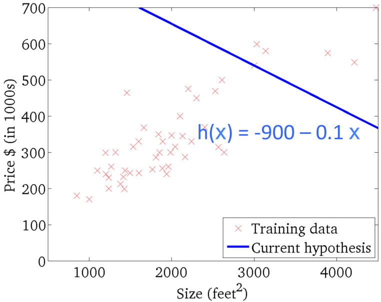
Start at a random point
Repeat
Determine a descent direction
Choose a step size
Update
Until stopping criterion is satisfied

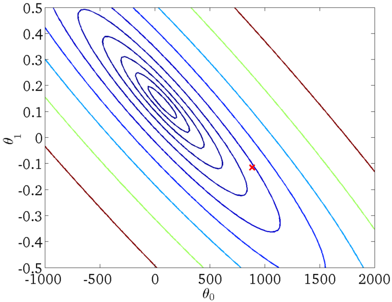


Gradient descent

$h_\theta(x)$

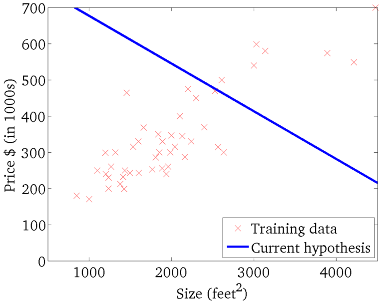
$R_N(\theta_1)$

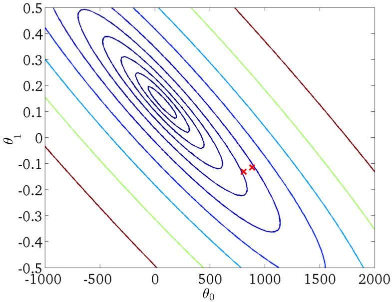




Gradient descent

$h_\theta(x)$ $\qquad$ $R_N(\theta_1)$

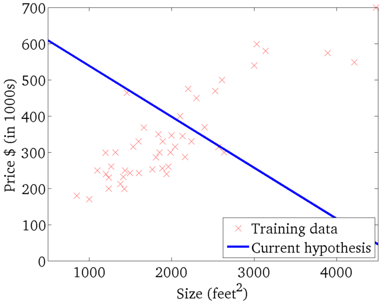


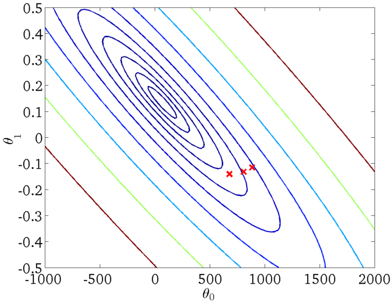


Gradient descent

$h_\theta(x)$

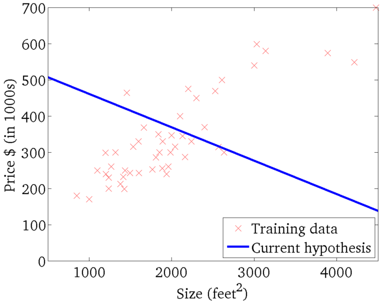
$R_N(\theta_1)$

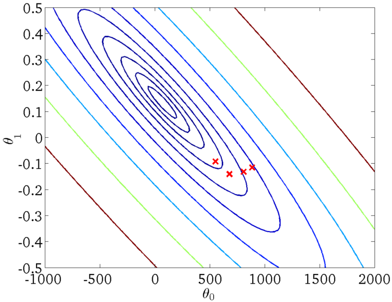




Gradient descent

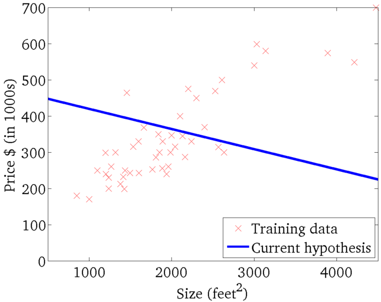
$h_\theta(x)$ $R_N(\theta_1)$

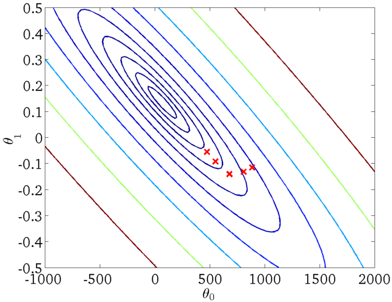




Gradient descent

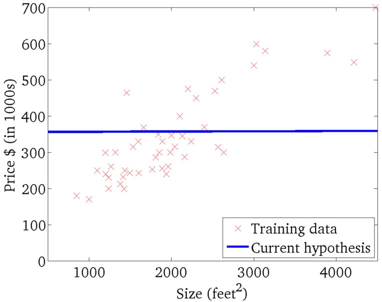
$h_\theta(x)$ $R_N(\theta_1)$

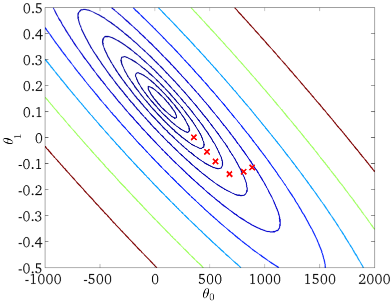




Gradient descent

$h_\theta(x)$ $R_N(\theta_1)$

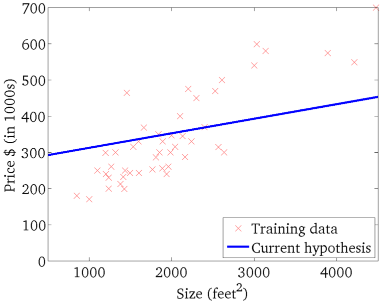


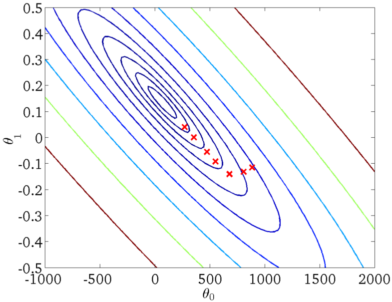


Gradient descent

$h_\theta(x)$

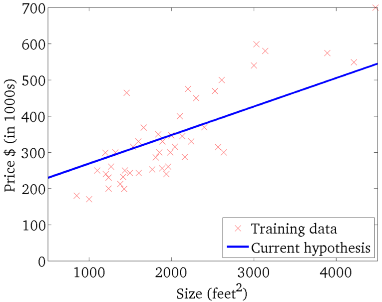
$R_N(\theta_1)$

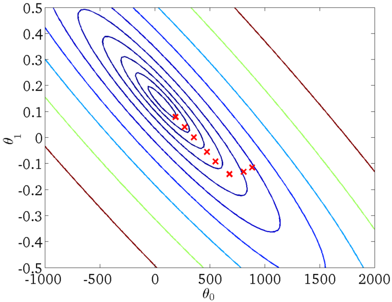




Gradient descent

$h_\theta(x)$ $R_N(\theta_1)$

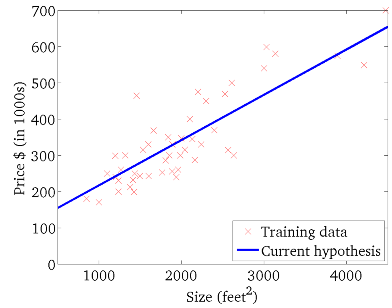


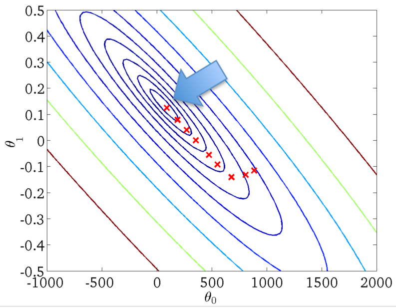


Gradient descent

$h_\theta(x)$

$R_N(\theta_1)$

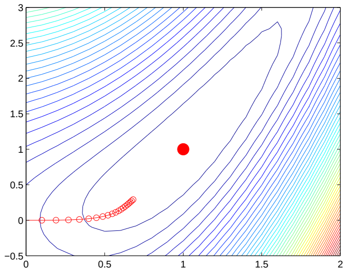




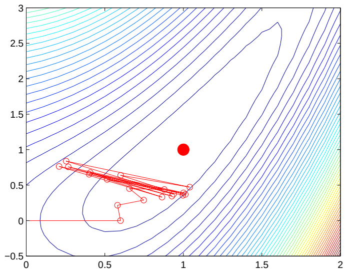
Seeing in action

Choosing the right η is important

small η is too slow?



large η is too unstable?



To see if gradient descent is working, print out function value at each iteration.

- The value should decrease at each iteration.
- Otherwise, adjust η.

Stochastic gradient descent

Widrow-Hoff rule: update parameters using one example at a time

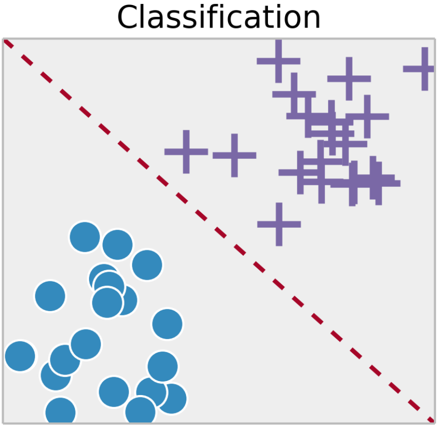
- Initialize $\boldsymbol{\theta}$ to $\boldsymbol{\theta}^{(0)}$ (anything reasonable is fine); set $t = 0$; choose $\eta > 0$
- Loop **until convergence**
  1. randomly choose training sample $\boldsymbol{x}_t$
  2. Compute its contribution to the gradient
  $$\boldsymbol{g}_t = (\boldsymbol{x}_t^{\mathrm{T}} \boldsymbol{\theta}^{(t)} - y_t)\boldsymbol{x}_t$$
  3. Update the parameters
  $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \boldsymbol{g}_t$
  4. $t \leftarrow t + 1$

How does the complexity per iteration compare with gradient descent?

Gradient descent: mini-summary

- Batch gradient descent computes the exact gradient.
- Stochastic gradient descent approximates the gradient with a single data point; Its expectation equals the true gradient.
- Mini-batch variant: trade-off between accuracy of estimating gradient and computational cost
- Similar ideas extend to other ML optimization problems.
    - For large-scale problems, stochastic gradient descent often works well.

Classification



Recall: Loss function

- Suppose we made a prediction $\hat{Y} = f(X) \in \mathcal{Y}$ based on observation of X.
- How good is the prediction? We can use a **loss function** $L : \mathcal{Y} \times \mathcal{Y} \mapsto \mathbb{R}^+$ to formalize the quality of the prediction.
- Typical loss functions:
  - **Squared loss** for regression

$$L(Y, f(X)) = (f(X) - Y)^2 .$$

  - **Absolute loss** for regression

$$L(Y, f(X)) = |f(X) - Y| .$$

  - **Misclassification loss** (or **0-1 loss**) for classification

$$L(Y, f(X)) = \begin{cases} 0 & f(X) = Y \\ 1 & f(X) \neq Y \end{cases} .$$

  Many other choices are possible, e.g., **weighted misclassification loss**.
- In classification, if estimated probabilities $\hat{p}(k)$ for each class $k \in \mathcal{Y}$ are returned, **log-likelihood loss** (or **log loss**) $L(Y, \hat{p}) = -\log \hat{p}(Y)$ is often used.

The Bayes Classifier

- What is the optimal classifier if the joint distribution (X, Y) were known?
- The density g of X can be written as a mixture of K components (corresponding to each of the classes):

$$g(x) = \sum_{k=1}^{K} \pi_k g_k(x),$$

  where, for $k = 1, \dots, K$,
  - $\mathbb{P}(Y = k) = \pi_k$ are the class probabilities,
  - $g_k(x)$ is the conditional density of X, given $Y = k$.
- The **Bayes classifier** $f_{\text{Bayes}} : x \mapsto \{1, \dots, K\}$ is the one with minimum risk:

$$\begin{aligned} R(f) =& \mathbb{E}\left[L(Y, f(X))\right] = \mathbb{E}_X\left[\mathbb{E}_{Y|X}[L(Y, f(X))|X]\right] \\ =& \int_{\mathcal{X}} \mathbb{E}\left[L(Y, f(X))|X = x\right] g(x) dx \end{aligned}$$

- The minimum risk attained by the Bayes classifier is called **Bayes risk**.
- Minimizing $\mathbb{E}[L(Y, f(X))|X = x]$ separately for each x suffices.

The Bayes Classifier

- Consider the 0-1 loss.
- The risk simplifies to:

$$\begin{aligned}\mathbb{E}\Big[L(Y,f(X))\Big|X=x\Big] =& \sum_{k=1}^{K} L(k,f(x))\mathbb{P}(Y=k|X=x) \\ =& 1-\mathbb{P}(Y=f(x)|X=x)\end{aligned}$$

- The risk is minimized by choosing the class with the greatest probability given the observation:

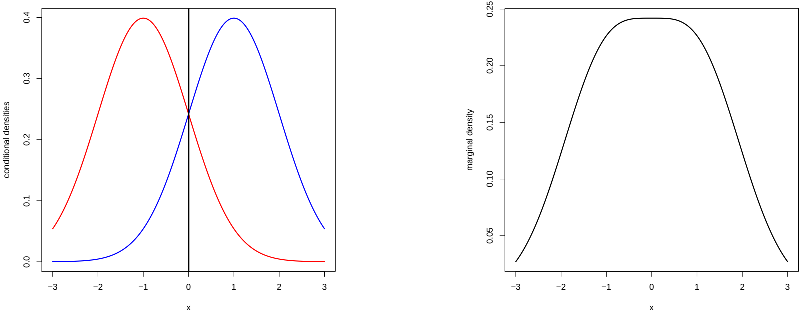
$$\begin{aligned}f_{\text{Bayes}}(x) &= \arg\max_{k=1,\ldots,K} \mathbb{P}(Y=k|X=x) \\ &= \arg\max_{k=1,\ldots,K} \frac{\pi_k g_k(x)}{\sum_{j=1}^{K} \pi_j g_j(x)} = \arg\max_{k=1,\ldots,K} \ \pi_k g_k(x).\end{aligned}$$

- The functions $x \mapsto \pi_k g_k(x)$ are called **discriminant functions**. The discriminant function with maximum value determines the predicted class of x.

The Bayes Classifier: Example

A simple two Gaussians example: Suppose $X \sim \mathcal{N}(\mu_Y, 1)$, where $\mu_1 = -1$ and $\mu_2 = 1$ and assume equal class probabilities $\pi_1 = \pi_2 = 1/2$.

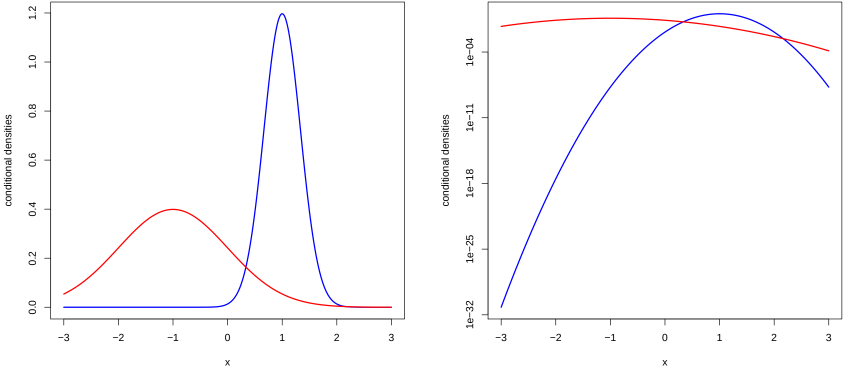
$$g_1(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(x+1)^2}{2}\right) \quad \text{and} \quad g_2(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(x-1)^2}{2}\right).$$



Optimal classification is $f_{\text{Bayes}}(x) = \underset{k=1,\ldots,K}{\arg\max}\ \pi_k g_k(x) = \begin{cases} 1 & \text{if } x < 0, \\ 2 & \text{if } x \geq 0. \end{cases}$

The Bayes Classifier: Example

How do you classify a new observation x if now the standard deviation is still 1 for class 1 but $1/3$ for class 2?



Looking at density in a log-scale, optimal classification is to select class 2 if and only if $x \in [0.34, 2.16]$.

Plug-in Classification

- The Bayes Classifier:

$$f_{\text{Bayes}}(x) = \arg\max_{k=1,\ldots,K} \pi_k g_k(x).$$

- We know neither the conditional densities g_k nor the class probabilities π_k!
- The **plug-in classifier** chooses the class

$$f(x) = \arg\max_{k=1,\ldots,K} \hat{\pi}_k \hat{g}_k(x),$$

- where we plugged in
  - estimates $\hat{\pi}_k$ of π_k and $k = 1, \ldots, K$ and
  - estimates $\hat{g}_k(x)$ of conditional densities,
- **Linear Discriminant Analysis** is an example of plug-in classification.