Data Mining & Machine Learning

CS37300
Purdue University

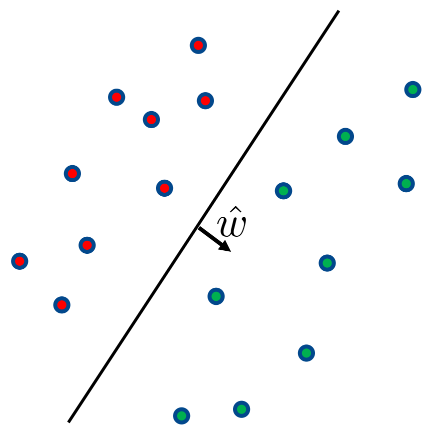
Sep 25, 2023

Today's topics

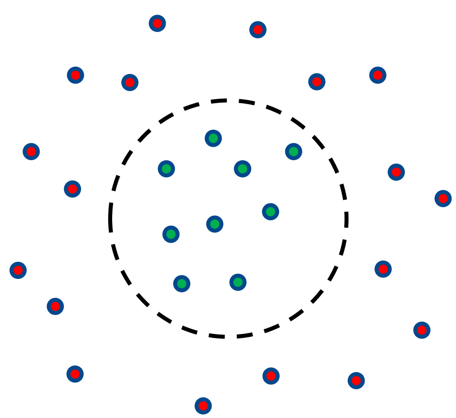
- Kernels and Kernel-SVM

Nonlinear Decision Boundaries

- Linear separators are simple



- But there are many scenarios that are separable, only non-linearly

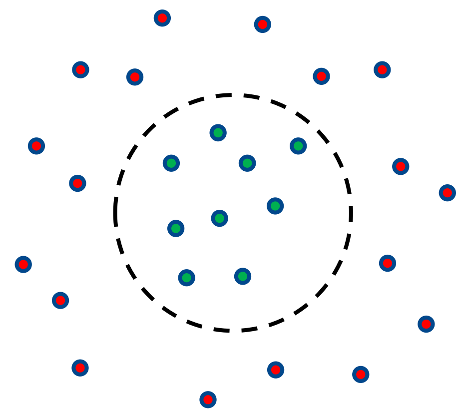


- We want to learn these with optimization based ML (SVM)

Learning Nonlinear Decision Boundaries

- We've already seen some nonlinear separators:
  - Decision Trees, KNN
- Question: How can we learn this example using SVM?

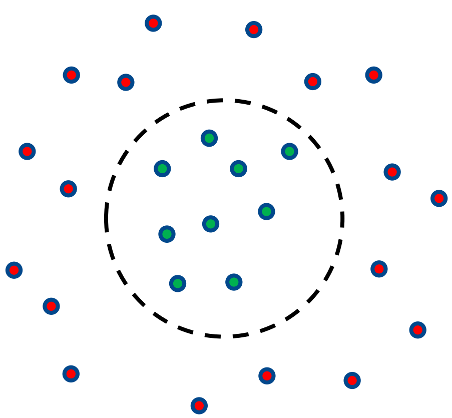
Example:



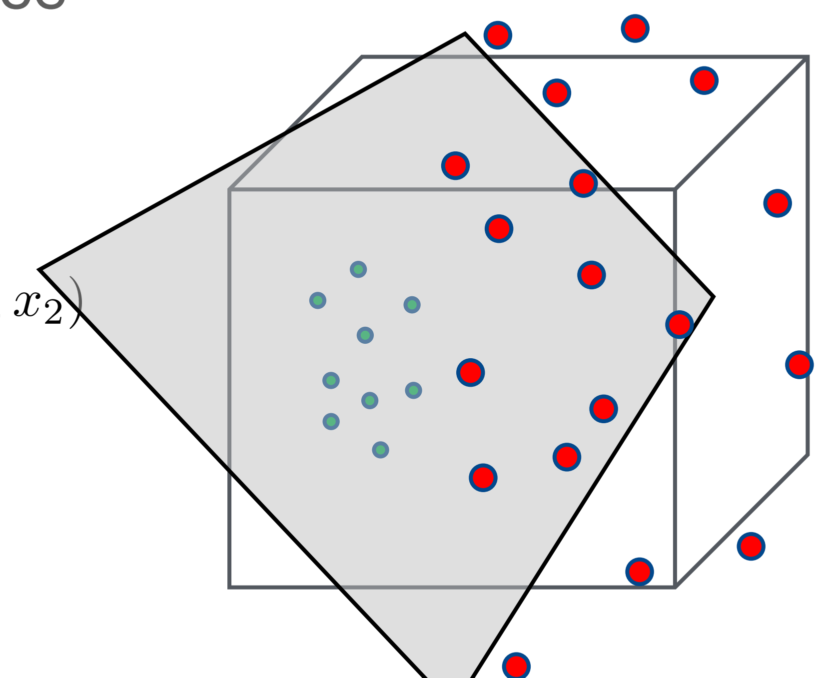
Learning Nonlinear Decision Boundaries

- We've already seen some nonlinear separators:
    - Decision Trees, KNN
- Question: How can we learn this example using SVM?
    - Map the examples into a higher-dimensional space and learn a linear separator in that space

Example:



Re-represent each $\underline{x} = (x_1, x_2)$ as 3-dimensional vector $\phi(\underline{x}) = (x_1^2, x_2^2, \sqrt{2}x_1 x_2)$



Representing Nonlinear Decision Boundaries

- General strategy: consider a function: $\phi(x) \in \mathbb{R}^N$
  - N may be large, or even infinite
- Then learn a linear separator in the ϕ space:

$$\hat{h}(x) = \mathrm{sign}\big(w^\top \phi(x)\big)\,, \quad \text{where } w \in \mathbb{R}^N$$

Representing Nonlinear Decision Boundaries

- General strategy: consider a function: $\phi(x) \in \mathbb{R}^N$
  - N may be large, or even infinite
- Then learn a linear separator in the ϕ space:

$$\hat{h}(x) = \operatorname{sign}\left(w^\top \phi(x)\right), \quad \text{where } w \in \mathbb{R}^N$$

$$\hat{w} = \underset{w \in \mathbb{R}^N : \|w\|=1}{\operatorname{argmax}} \quad \min_{1 \le i \le n} y_i \left(w^\top \phi(x_i)\right)$$

$$\boxed{\begin{array}{ll} \text{Minimize} & \|w\|^2 \\ \text{subject to} & y_i \left(w^\top \phi(x_i)\right) \ge 1, \ \forall i : \ 1 \le i \le n \end{array}}$$

Representing Nonlinear Decision Boundaries

- General strategy: consider a function: $\phi(x) \in \mathbb{R}^N$
  - N may be large, or even infinite
- Then learn a linear separator in the ϕ space:

$$\hat{h}(x) = \operatorname{sign}\big(w^\top \phi(x)\big)\,, \quad \text{where } w \in \mathbb{R}^N$$

- Two concerns:

  1. Computation and memory required to represent $\phi(x)$, w
  2. Dimension of linear separators on $\mathbb{R}^N$ is N.
     $\rightarrow$ Concerns about overfitting

  Today's lecture is mostly about the first concern.
  Solution: Kernels

The Dual Form of the SVM Optimization Problem

$$\hat{w} = \operatorname*{argmax}_{w:\|w\|=1} \min_{1 \le i \le n} y_i \left(w^\top x_i \right)$$

- Recall: The SVM **classifier** is the unique solution to a **quadratic program**:

$$\boxed{\begin{array}{ll} \text{Minimize} & \|w\|^2 \\ \text{subject to} & y_i \left(w^\top x_i \right) \ge 1, \ \forall i : \ 1 \le i \le n \end{array}}$$

- We can re-express this in Lagrangian dual form:

$$\hat{w} = \textstyle\sum_{i=1}^{n} \alpha_i y_i x_i$$

where $\alpha_1, \ldots, \alpha_n$ are solutions to:

$$\boxed{\begin{array}{ll} \text{Maximize} & \displaystyle\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j x_i^\top x_j \\ \text{subject to} & \alpha_i \ge 0, \ \forall i : \ 1 \le i \le n \\ & \displaystyle\sum_{i=1}^{n} \alpha_i y_i = 0 \end{array}}$$

SVM with high-dim mapping

- We could use a mapping ϕ to get a non-linear separator:

$$\hat{w} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{\phi(x_i)}$$

where $\alpha_1, \ldots, \alpha_n$ are solutions to:

$$\boxed{\begin{aligned} &\text{Maximize} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \boldsymbol{\phi(x_i)}^\top \boldsymbol{\phi(x_j)} \\ &\text{subject to} \quad \alpha_i \geq 0, \ \forall i: \ 1 \leq i \leq n \\ &\qquad\qquad\quad \sum_{i=1}^{n} \alpha_i y_i = 0 \end{aligned}}$$

- Notice training only uses ϕ when computing inner product $\phi(x_i)^\top \phi(x_j)$
- Instead of starting by defining ϕ, we could start by defining a function $K(x_i, x_j)$ that computes an inner product, without computing ϕ: $K(x_i, x_j) = \phi(x_i)^\top \phi(x_j)$
- **Kernel**: way to compute inner products without computing $\phi(x)$!

SVM with high-dim mapping

- We could use a mapping ϕ to get a non-linear separator:

$$\hat{w} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{\phi(x_i)}$$

where $\alpha_1, \ldots, \alpha_n$ are solutions to:

$$\text{Maximize} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \boldsymbol{\phi(x_i)}^\top \boldsymbol{\phi(x_j)}$$

$$\text{subject to} \quad \alpha_i \geq 0, \; \forall i: \; 1 \leq i \leq n$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

- Notice training only uses ϕ when computing inner product

$$\phi(x_i)^\top \phi(x_j)$$

- And the classifier also uses an inner product (no need to compute w)

$$h_{\hat{w}}(x) = \text{sign}\left(\hat{w}^\top \phi(x)\right) = \text{sign}\left(\left(\sum_{i=1}^{n} \alpha_i y_i \phi(x_i)\right)^\top \phi(x)\right) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i \boldsymbol{\phi(x_i)}^\top \boldsymbol{\phi(x)}\right)$$

- Replace all of these with

$$K(x_i, x) = \phi(x_i)^\top \phi(x)$$

Kernel SVM

- Training time:

  Solve for $\alpha_1, \ldots, \alpha_n$:

$$\boxed{\begin{array}{ll} \text{Maximize} & \displaystyle\sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \boldsymbol{K(x_i, x_j)} \\ \text{subject to} & \alpha_i \geq 0, \ \forall i : \ 1 \leq i \leq n \\ & \displaystyle\sum_{i=1}^{n} \alpha_i y_i = 0 \end{array}}$$

- Test time:

- Classify a new point x with

$$\hat{h}(x) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i \boldsymbol{K(x_i, x)}\right)$$

Kernels

Example:

Quadratic kernel: $K(\underline{u}, \underline{v}) = \left(\underline{u}^\top \underline{v} + 1\right)^2$

For $\underline{x} \in \mathbb{R}^d$, implicitly computes an inner product in $\frac{1}{2}d(d-1) + 2d + 1$ dim

e.g., For $\underline{x} \in \mathbb{R}^2$, implicitly defined ϕ :

$$\phi(\underline{x}) = \left[1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1 x_2\right]^\top$$

Check: $\phi(\underline{u})^\top \phi(\underline{v})$

$$= \left[1, \sqrt{2}u_1, \sqrt{2}u_2, u_1^2, u_2^2, \sqrt{2}u_1 u_2\right] \left[1, \sqrt{2}v_1, \sqrt{2}v_2, v_1^2, v_2^2, \sqrt{2}v_1 v_2\right]^\top$$

$$= 1 + 2u_1 v_1 + 2u_2 v_2 + u_1^2 v_1^2 + u_2^2 v_2^2 + 2u_1 u_2 v_1 v_2$$

$$= (1 + u_1 v_1 + u_2 v_2)^2 = \left(1 + \underline{u}^\top \underline{v}\right)^2 = K(\underline{u}, \underline{v})$$

The point is that we can compute this higher-dim inner product just by evaluating K(u,v): no need to compute ϕ

Kernels

- More Examples:
  - Polynomial kernel: $K(\underline{u}, \underline{v}) = \left(\underline{u}^\top \underline{v} + 1\right)^p$
    - Implicitly computes an inner product in $\sim d^p$ dimensions
  - Gaussian kernel:

$$K(\underline{u}, \underline{v}) = e^{-\frac{\|\underline{u}-\underline{v}\|^2}{2\sigma^2}}$$

    - Implicitly computes an inner product in **infinite** dimensions
    - Remark: This is the most popular kernel in practice

What are "Legal" Kernels?

- A kernel K(·,·) is a legal definition of an inner product
- Technically, this is called a **Mercer kernel**
  - A kernel should be a symmetric function: K(u,v)=K(v,u)
  - A kernel should also be positive semi-definite: namely,
    - For any set of data points $x_1,\ldots,x_n$
    - And for any values $a_1, \ldots, a_n \in \mathbb{R}$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j K(x_i, x_j) \geq 0$$

- Mercer's Theorem: For any K satisfying the above, there exists some function ϕ such that

$$K(u,v) = \phi(u)^\top \phi(v)$$

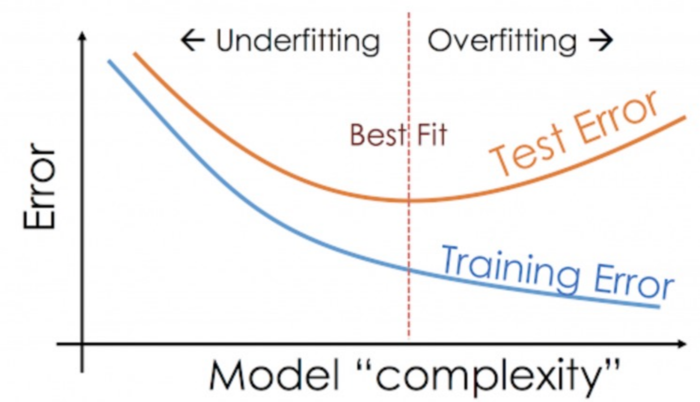
- In other words: K is a valid kernel

Example: Soft-SVM with Gaussian Kernel

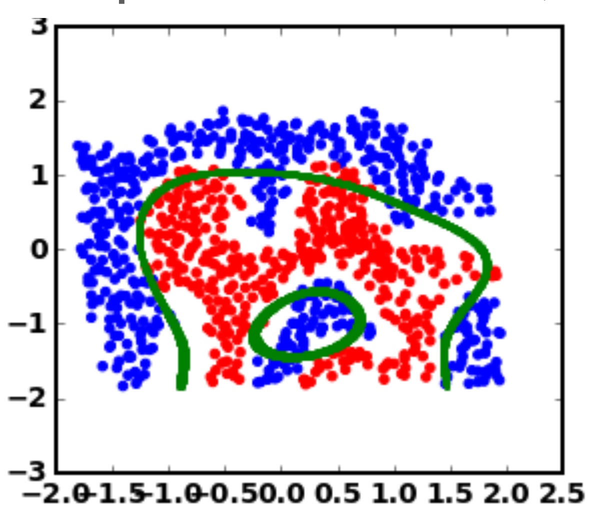
- A common choice is to use the Gaussian kernel

$$K(\underline{u}, \underline{v}) = e^{-\frac{\|\underline{u}-\underline{v}\|^2}{2\sigma^2}}$$

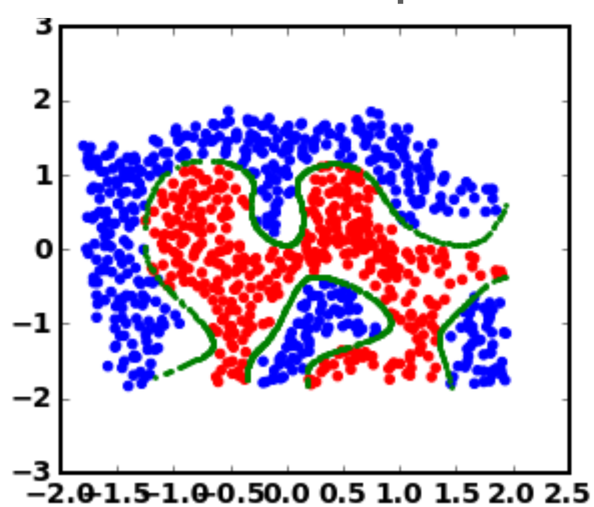
- σ called the **bandwidth**
- It controls smoothness of the boundary
- Gives a notion of model complexity:



- large σ → simple boundaries, small σ → complex boundaries



large σ



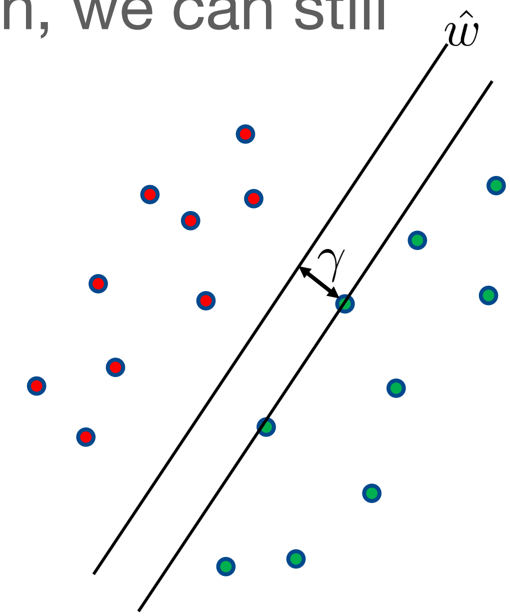
small σ

What about overfitting?

- If we're using such a high-dimensional representation, won't the dimension be large? Doesn't this lead to overfitting?

- Margin: If we can find a solution with large margin, we can still avoid overfitting.

- This is true even with kernels

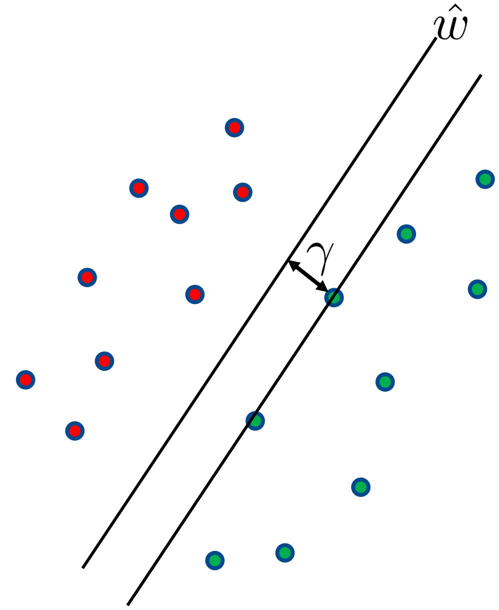


What about overfitting?

- Margin for kernel SVM:

geometric margin of classifier w:

$$\gamma = \min_{1 \leq i \leq n} y_i \left(\frac{w^\top}{\|w\|} \phi(x_i) \right)$$



Recall: The $\hat{w}$ solution of SVM primal problem satisfies $\|\hat{w}\| = \frac{1}{\gamma}$

For kernel SVM, this means

$$\frac{1}{\gamma^2} = \|\hat{w}\|^2 = \hat{w}^\top \hat{w} = \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

Solve for γ to compute the margin

So we can also calculate the margin without computing ϕ, using K

Summary

- Kernel methods are a convenient family of algorithms
- They allow us to specify a non-linear representation for the classifier just by providing a kernel function, and the rest is automatic
- In this sense, they are “plug-and-play”: very easy to use
- We need to be careful that the implicit high-dimensional representation doesn’t lead to overfitting
- If the solution has large margin, it can avoid overfitting
- Recall that SVM is designed to maximize the margin, so it is well-suited to this type of guarantee

Gradient descent

Logistic regression learning

$$minimize \sum_{i=1}^{N} (-y_i w^T x_i + log(1 + e^{w^T x_i}))$$

$$\frac{dlogL(w|D)}{dw_j} = \sum_{i=1}^{N} (-y_i x_{ij} + \frac{1}{1 + e^{w^T x_i}} e^{w^T x_i} x_{ij})$$

$$= \sum_{i=1}^{N} (-y_i + \frac{1}{1 + e^{w^T x_i}} e^{w^T x_i}) x_{ij}$$

$$= \sum_{i=1}^{N} (-y_i + P(y_i = 1 | w)) x_{ij}$$

Convex!

But no closed form solution!

Convex optimization problems

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & x \in C \end{array}$$

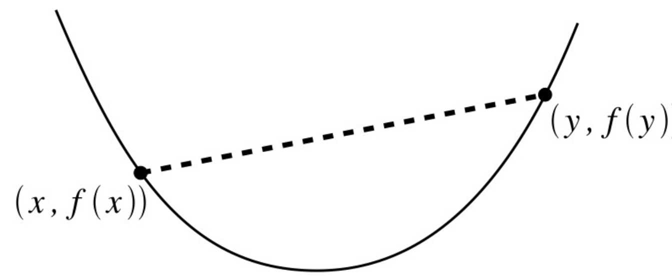
- *x* is the optimization variable (*e.g., model parameters*)
  f (*e.g., score function*) is a **convex function**
  C is a **convex set** (*e.g., constraints on model parameters*)
- For convex optimization problems, all locally optimal points are globally optimal

Convex functions

- In graph of convex function f, the line connecting two points must lie above the function:

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha) f(y) \text{ for all } 0 \leq \alpha \leq 1$$

- Practical test for convexity: a twice differentiable function f of a variable is convex on an interval if an only if for any x in the interval: $f''(x) \geq 0$
  - Strictly convex if $f''(x) > 0$

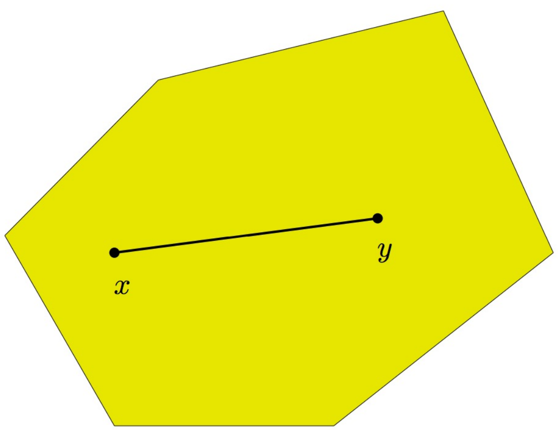


- Sum of convex functions is convex; max of convex functions is convex

Convex set

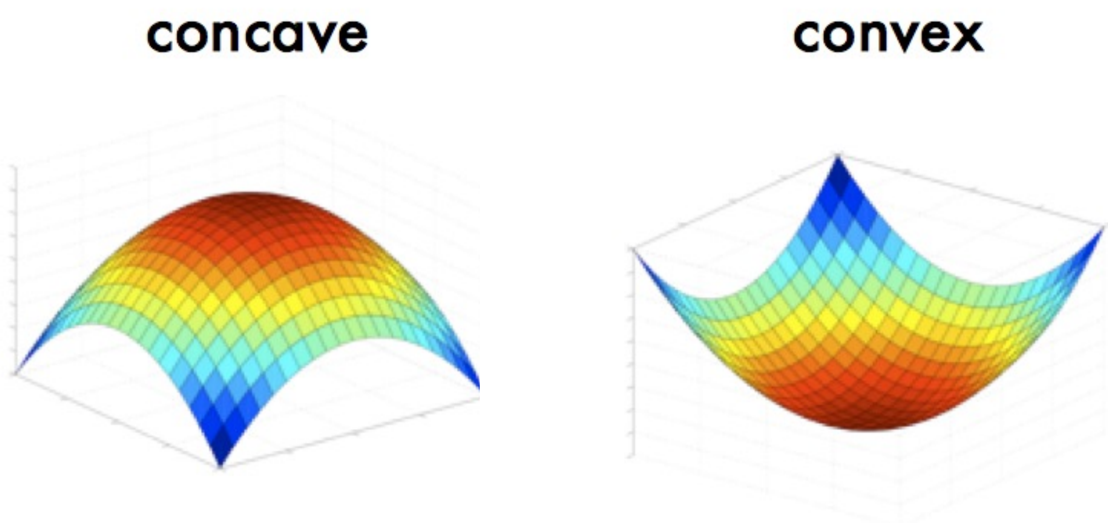
- A set C is convex if for any $x, y \in C$ and any θ with $0 \leq \theta \leq 1$ we have

$$\theta x + (1 - \theta)y \in C$$



Concave vs convex

- Maximizing a concave function is equivalent to minimizing a convex function



Solve convex optimization problem

- Minimize a convex function without any constraints on the variables
  - If f'(x)=0 then x is a stationary point of f
  - If f'(x)=0 and f''(x) is not negative then x is a local minimum of f (for convex function, this is also a global minimum)
  - If f is a strictly convex function, any stationary point of ***f*** is the unique global minimum of f

Gradient descent

- For some convex functions, we may be able to take the derivative, but it may be difficult to directly solve for parameter values
- Solution:
  - Start at some value of the parameters
  - Take derivative and use it to move the parameters in the direction of the negative gradient
  - Repeat until stopping criteria is met (e.g., gradient close to 0)

<u>Gradient Descent Rule:</u>

$$\underline{\mathbf{w}}_{\text{new}} = \underline{\mathbf{w}}_{\text{old}} - \eta \, \Delta(\underline{\mathbf{w}})$$

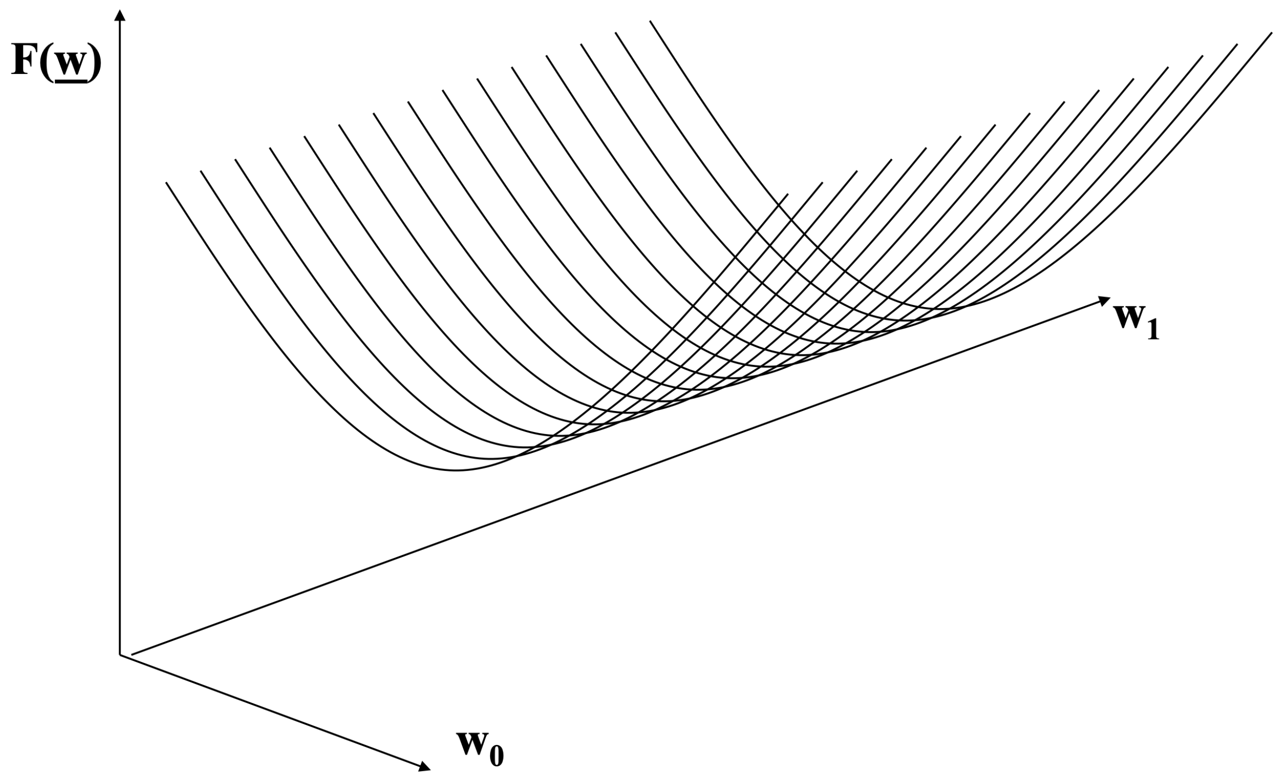
where

$\Delta(\underline{w})$ is the gradient and
η is the learning rate (small, positive)

Notes:

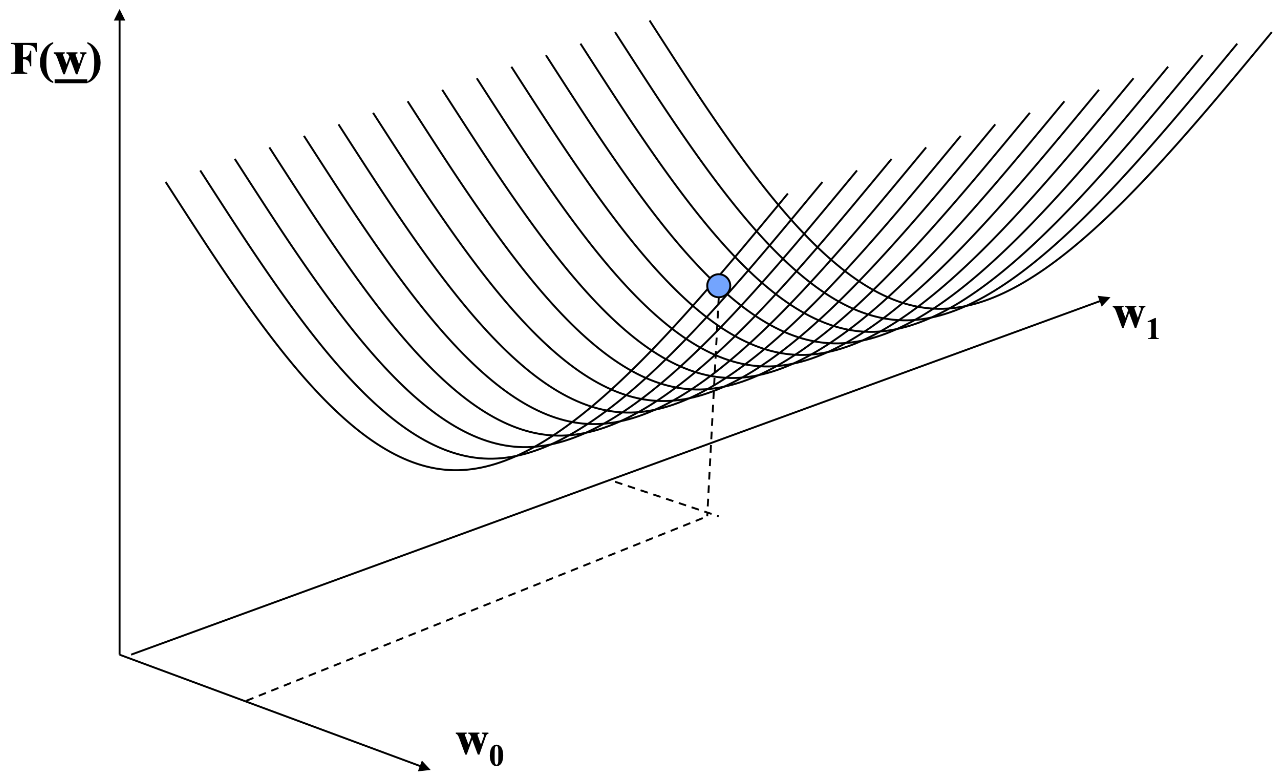
1. This moves us downhill in direction $\Delta(\underline{\mathbf{w}})$ (steepest downhill direction)
2. How far we go is determined by the value of η

Illustration of gradient descent



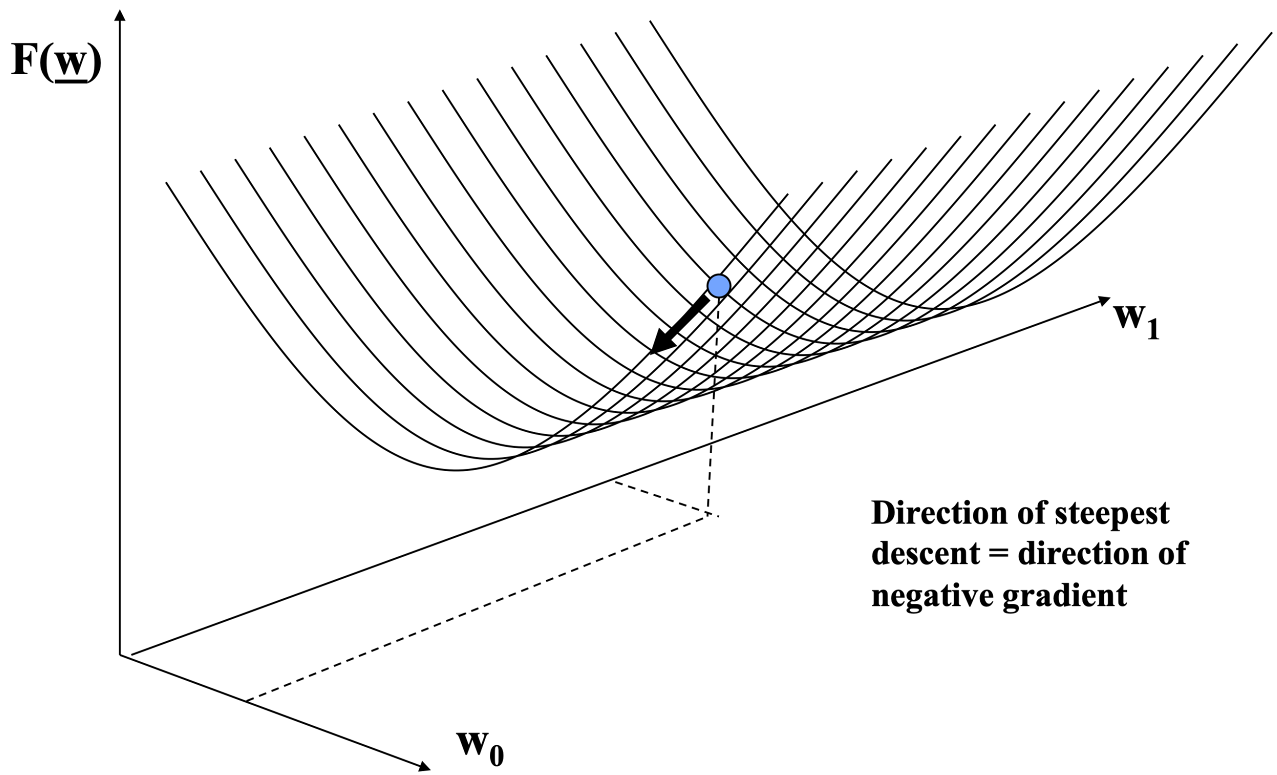
Slides adapted from CS175, UCIrvine, Padhraic Smyth

Illustration of gradient descent



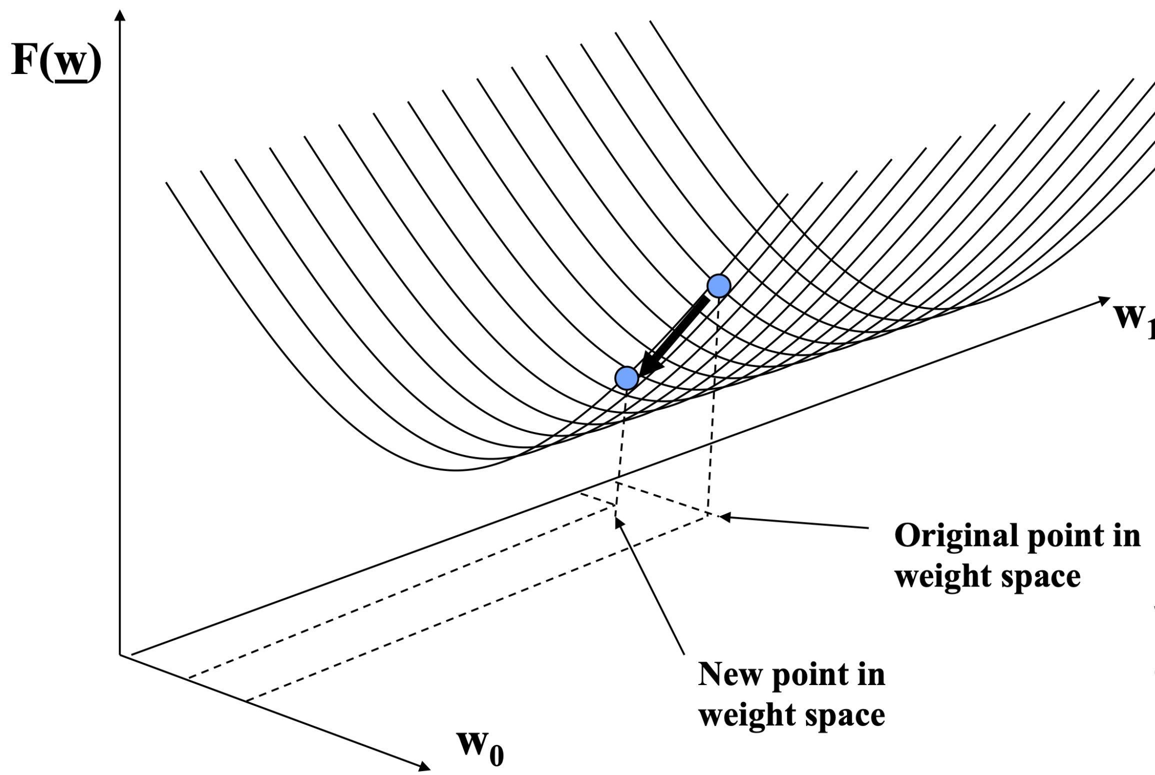
Slides adapted from CS175, UCIrvine, Padhraic Smyth

Illustration of gradient descent



Slides adapted from CS175, UCIrvine, Padhraic Smyth

Illustration of gradient descent



For convex functions, when gradient descent converges, the solution is global minimum.

Slides adapted from CS175, UCIrvine, Padhraic Smyth