Max Margin-Classifier

Oliver Schulte - CMPT 726

Bishop PRML Ch. 7

Outline

Kernels and Non-linear Mappings

- Where does the maximization problem come from?
- The intuition comes from the primal version, which is based on a feature mapping ϕ.
- Theorem: Every valid kernel $k(\boldsymbol{x}, \boldsymbol{y})$ is the dot-product $\phi(\boldsymbol{x})[\phi(\boldsymbol{x})]^T$ for some set of basis functions (feature mapping) ϕ.
- The feature space $\phi(\boldsymbol{x})$ could be high-dimensional, even infinite.
- This is good because if data aren't separable in original input space ($\boldsymbol{x}$), they may be in feature space $\phi(\boldsymbol{x})$
- We can think about how to find a good linear separator using the dot product in high dimensions, then transfer this back to kernels in the original input space.

Kernels and Non-linear Mappings

- Where does the maximization problem come from?
- The intuition comes from the primal version, which is based on a feature mapping ϕ.
- Theorem: Every valid kernel $k(\boldsymbol{x}, \boldsymbol{y})$ is the dot-product $\phi(\boldsymbol{x})[\phi(\boldsymbol{x})]^T$ for some set of basis functions (feature mapping) ϕ.
- The feature space $\phi(\boldsymbol{x})$ could be high-dimensional, even infinite.
- This is good because if data aren't separable in original input space ($\boldsymbol{x}$), they may be in feature space $\phi(\boldsymbol{x})$
- We can think about how to find a good linear separator using the dot product in high dimensions, then transfer this back to kernels in the original input space.

Kernels and Non-linear Mappings

- Where does the maximization problem come from?
- The intuition comes from the primal version, which is based on a feature mapping ϕ.
- Theorem: Every valid kernel $k(\boldsymbol{x}, \boldsymbol{y})$ is the dot-product $\phi(\boldsymbol{x})[\phi(\boldsymbol{x})]^T$ for some set of basis functions (feature mapping) ϕ.
- The feature space $\phi(\boldsymbol{x})$ could be high-dimensional, even infinite.
- This is good because if data aren't separable in original input space ($\boldsymbol{x}$), they may be in feature space $\phi(\boldsymbol{x})$
- We can think about how to find a good linear separator using the dot product in high dimensions, then transfer this back to kernels in the original input space.

Kernels and Non-linear Mappings

- Where does the maximization problem come from?
- The intuition comes from the primal version, which is based on a feature mapping ϕ.
- Theorem: Every valid kernel $k(\boldsymbol{x}, \boldsymbol{y})$ is the dot-product $\phi(\boldsymbol{x})[\phi(\boldsymbol{x})]^T$ for some set of basis functions (feature mapping) ϕ.
- The feature space $\phi(\boldsymbol{x})$ could be high-dimensional, even infinite.
- This is good because if data aren't separable in original input space ($\boldsymbol{x}$), they may be in feature space $\phi(\boldsymbol{x})$
- We can think about how to find a good linear separator using the dot product in high dimensions, then transfer this back to kernels in the original input space.

Why Kernels?

- If we can use dot products with features, why bother with kernels?
  - Often easier to specify how similar two things are (dot product) than to construct explicit feature space ϕ.
  - e.g. graphs, sets, strings (NIPS 2009 best student paper award).
  - There are high-dimensional (even infinite) spaces that have efficient-to-compute kernels

Kernel Trick

- In previous lectures on linear models, we would explicitly compute $\phi(\boldsymbol{x}_i)$ for each datapoint
  - Run algorithm in feature space
- For some feature spaces, can compute dot product $\phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$ efficiently
- Efficient method is computation of a kernel function $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$
- The kernel trick is to rewrite an algorithm to only have $\boldsymbol{x}$ enter in the form of dot products
- The menu:
  - Kernel trick examples
  - Kernel functions

Kernel Trick

- In previous lectures on linear models, we would explicitly compute $\phi(\boldsymbol{x}_i)$ for each datapoint
  - Run algorithm in feature space
- For some feature spaces, can compute dot product $\phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$ efficiently
- Efficient method is computation of a kernel function $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$
- The kernel trick is to rewrite an algorithm to only have x enter in the form of dot products
- The menu:
  - Kernel trick examples
  - Kernel functions

Kernel Trick

- In previous lectures on linear models, we would explicitly compute $\phi(\boldsymbol{x}_i)$ for each datapoint
  - Run algorithm in feature space
- For some feature spaces, can compute dot product $\phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$ efficiently
- Efficient method is computation of a kernel function $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$
- The kernel trick is to rewrite an algorithm to only have $\boldsymbol{x}$ enter in the form of dot products
- The menu:
  - Kernel trick examples
  - Kernel functions

Kernel Trick

- In previous lectures on linear models, we would explicitly compute $\phi(\boldsymbol{x}_i)$ for each datapoint
  - Run algorithm in feature space
- For some feature spaces, can compute dot product $\phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$ efficiently
- Efficient method is computation of a kernel function $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$
- The kernel trick is to rewrite an algorithm to only have $\boldsymbol{x}$ enter in the form of dot products
- The menu:
  - Kernel trick examples
  - Kernel functions

A Kernel Trick

- Let's look at the nearest-neighbour classification algorithm
- For input point x_i, find point x_j with smallest distance:

$$\begin{aligned} ||\boldsymbol{x}_i - \boldsymbol{x}_j||^2 &= (\boldsymbol{x}_i - \boldsymbol{x}_j)^T(\boldsymbol{x}_i - \boldsymbol{x}_j) \\ &= \boldsymbol{x}_i{}^T\boldsymbol{x}_i - 2\boldsymbol{x}_i{}^T\boldsymbol{x}_j + \boldsymbol{x}_j{}^T\boldsymbol{x}_j \end{aligned}$$

- If we used a non-linear feature space $\phi(\cdot)$:

$$\begin{aligned} ||\phi(\boldsymbol{x}_i) - \phi(\boldsymbol{x}_j)||^2 &= \phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_i) - 2\phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j) + \phi(\boldsymbol{x}_j)^T\phi(\boldsymbol{x}_j) \\ &= k(\boldsymbol{x}_i, \boldsymbol{x}_i) - 2k(\boldsymbol{x}_i, \boldsymbol{x}_j) + k(\boldsymbol{x}_j, \boldsymbol{x}_j) \end{aligned}$$

- So nearest-neighbour can be done in a high-dimensional feature space without actually moving to it

A Kernel Trick

- Let's look at the nearest-neighbour classification algorithm
- For input point x_i, find point x_j with smallest distance:

$$\begin{aligned} ||\boldsymbol{x}_i - \boldsymbol{x}_j||^2 &= (\boldsymbol{x}_i - \boldsymbol{x}_j)^T(\boldsymbol{x}_i - \boldsymbol{x}_j) \\ &= \boldsymbol{x}_i{}^T\boldsymbol{x}_i - 2\boldsymbol{x}_i{}^T\boldsymbol{x}_j + \boldsymbol{x}_j{}^T\boldsymbol{x}_j \end{aligned}$$

- If we used a non-linear feature space $\phi(\cdot)$:

$$\begin{aligned} ||\phi(\boldsymbol{x}_i) - \phi(\boldsymbol{x}_j)||^2 &= \phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_i) - 2\phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j) + \phi(\boldsymbol{x}_j)^T\phi(\boldsymbol{x}_j) \\ &= k(\boldsymbol{x}_i, \boldsymbol{x}_i) - 2k(\boldsymbol{x}_i, \boldsymbol{x}_j) + k(\boldsymbol{x}_j, \boldsymbol{x}_j) \end{aligned}$$

- So nearest-neighbour can be done in a high-dimensional feature space without actually moving to it

A Kernel Trick

- Let's look at the nearest-neighbour classification algorithm
- For input point x_i, find point x_j with smallest distance:

$$\begin{aligned} ||\boldsymbol{x}_i - \boldsymbol{x}_j||^2 &= (\boldsymbol{x}_i - \boldsymbol{x}_j)^T(\boldsymbol{x}_i - \boldsymbol{x}_j) \\ &= \boldsymbol{x}_i{}^T\boldsymbol{x}_i - 2\boldsymbol{x}_i{}^T\boldsymbol{x}_j + \boldsymbol{x}_j{}^T\boldsymbol{x}_j \end{aligned}$$

- If we used a non-linear feature space $\phi(\cdot)$:

$$\begin{aligned} ||\phi(\boldsymbol{x}_i) - \phi(\boldsymbol{x}_j)||^2 &= \phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_i) - 2\phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j) + \phi(\boldsymbol{x}_j)^T\phi(\boldsymbol{x}_j) \\ &= k(\boldsymbol{x}_i, \boldsymbol{x}_i) - 2k(\boldsymbol{x}_i, \boldsymbol{x}_j) + k(\boldsymbol{x}_j, \boldsymbol{x}_j) \end{aligned}$$

- So nearest-neighbour can be done in a high-dimensional feature space without actually moving to it

Example: The Quadratic Kernel Function

- Consider again the kernel function $k(\boldsymbol{x}, \boldsymbol{z}) = (1 + \boldsymbol{x}^T \boldsymbol{z})^2$
- With $\boldsymbol{x}, \boldsymbol{z} \in \mathbb{R}^2$,

$$
\begin{aligned}
k(\boldsymbol{x}, \boldsymbol{z}) &= (1 + x_1 z_1 + x_2 z_2)^2 \\
&= 1 + 2x_1 z_1 + 2x_2 z_2 + x_1^2 z_1^2 + 2x_1 z_1 x_2 z_2 + x_2^2 z_2^2 \\
&= (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, \sqrt{2}x_1 x_2, x_2^2)(1, \sqrt{2}z_1, \sqrt{2}z_2, z_1^2, \sqrt{2}z_1 z_2, z_2^2)^T \\
&= \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{z})
\end{aligned}
$$

- So this particular kernel function does correspond to a dot product in a feature space (is valid)
- Computing $k(\boldsymbol{x}, \boldsymbol{z})$ is faster than explicitly computing $\boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{z})$
  - In higher dimensions, larger exponent, much faster

Example: The Quadratic Kernel Function

- Consider again the kernel function $k(\boldsymbol{x}, \boldsymbol{z}) = (1 + \boldsymbol{x}^T \boldsymbol{z})^2$
- With $\boldsymbol{x}, \boldsymbol{z} \in \mathbb{R}^2$,

$$\begin{aligned} k(\boldsymbol{x}, \boldsymbol{z}) &= (1 + x_1 z_1 + x_2 z_2)^2 \\ &= 1 + 2x_1 z_1 + 2x_2 z_2 + x_1^2 z_1^2 + 2x_1 z_1 x_2 z_2 + x_2^2 z_2^2 \\ &= (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, \sqrt{2}x_1 x_2, x_2^2)(1, \sqrt{2}z_1, \sqrt{2}z_2, z_1^2, \sqrt{2}z_1 z_2, z_2^2)^T \\ &= \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{z}) \end{aligned}$$

- So this particular kernel function does correspond to a dot product in a feature space (is valid)
- Computing $k(\boldsymbol{x}, \boldsymbol{z})$ is faster than explicitly computing $\boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{z})$
  - In higher dimensions, larger exponent, much faster

Example: The Quadratic Kernel Function

- Consider again the kernel function $k(\boldsymbol{x}, \boldsymbol{z}) = (1 + \boldsymbol{x}^T \boldsymbol{z})^2$
- With $\boldsymbol{x}, \boldsymbol{z} \in \mathbb{R}^2$,

$$\begin{aligned} k(\boldsymbol{x}, \boldsymbol{z}) &= (1 + x_1 z_1 + x_2 z_2)^2 \\ &= 1 + 2x_1 z_1 + 2x_2 z_2 + x_1^2 z_1^2 + 2x_1 z_1 x_2 z_2 + x_2^2 z_2^2 \\ &= (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, \sqrt{2}x_1 x_2, x_2^2)(1, \sqrt{2}z_1, \sqrt{2}z_2, z_1^2, \sqrt{2}z_1 z_2, z_2^2)^T \\ &= \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{z}) \end{aligned}$$

- So this particular kernel function does correspond to a dot product in a feature space (is valid)
- Computing $k(\boldsymbol{x}, \boldsymbol{z})$ is faster than explicitly computing $\boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{z})$
  - In higher dimensions, larger exponent, much faster

Regression Kernelized

- Many classifiers can be written as using only dot products.
- Kernelization = replace dot products by kernel.
- E.g., the kernel solution for regularized least squares regression is

$$y(\boldsymbol{x}) = \boldsymbol{k}(\boldsymbol{x})^T(\boldsymbol{K} + \lambda \boldsymbol{I}_N)^{-1}\boldsymbol{t} \quad \text{vs.} \quad \boldsymbol{\phi}(\boldsymbol{x})(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda \boldsymbol{I}_M)^{-1}\boldsymbol{\Phi}^T\boldsymbol{t}$$

  for original version
- N is number of datapoints (size of Gram matrix $\boldsymbol{K}$)
- M is number of basis functions (size of matrix $\boldsymbol{\Phi}^T\boldsymbol{\Phi}$)
- Bad if $N > M$, but good otherwise
- $\boldsymbol{k}(\boldsymbol{x}) = (k(\boldsymbol{x}, \boldsymbol{x}_1, \ldots, k(\boldsymbol{x}, \boldsymbol{x}_n))$ is the vector of kernel values over data points x_n.

Outline

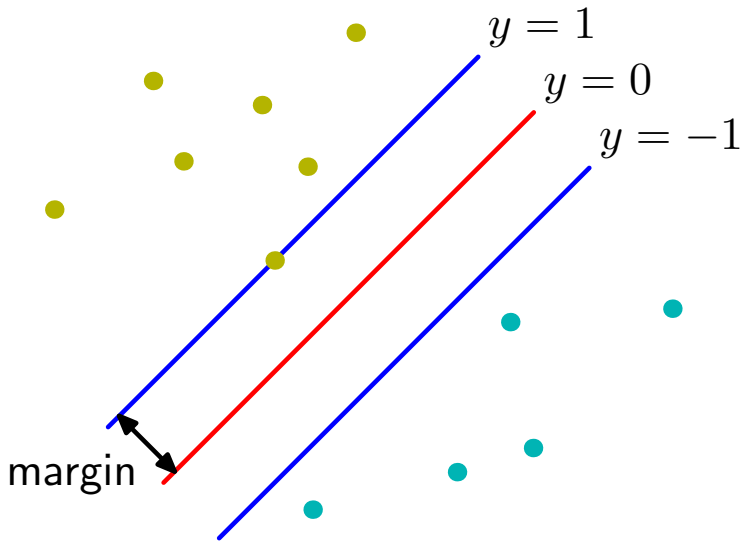
Linear Classification

- Consider a two class classification problem
- Use a linear model

$$y(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}) + b$$

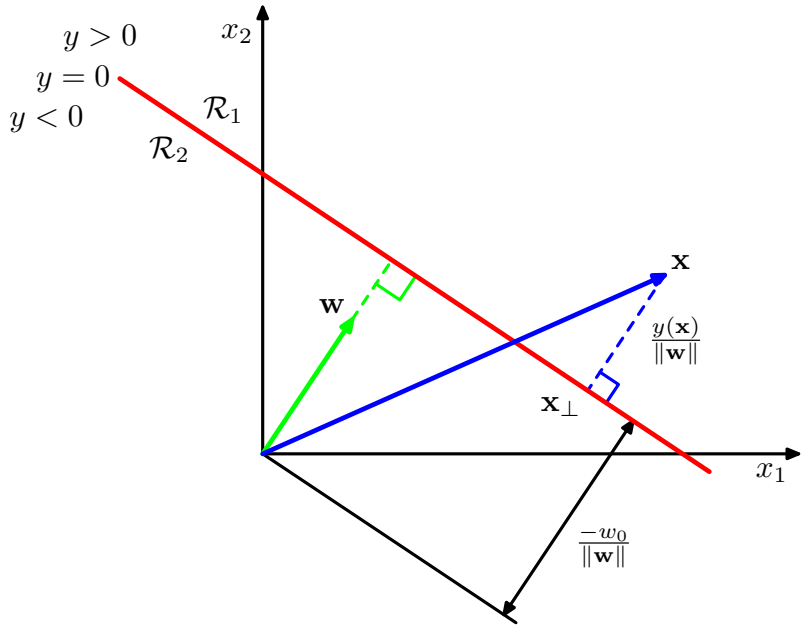
  followed by a threshold function
- For now, let's assume training data are linearly separable
  - Recall that the perceptron would converge to a perfect classifier for such data
  - But there are many such perfect classifiers

Max Margin



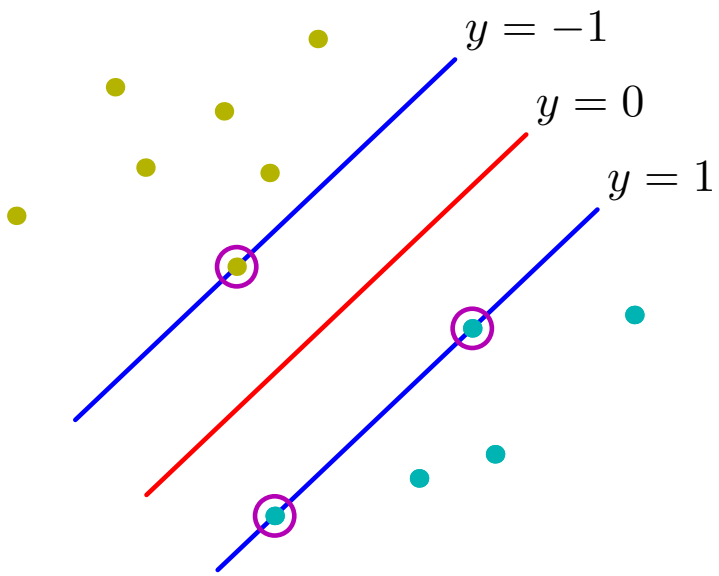
- We can define the margin of a classifier as the minimum distance to any example
- In support vector machines the decision boundary which maximizes the margin is chosen.
- Intuitively, this is the line “right in the middle” between the two classes.

Marginal Geometry



- Recall from Ch. 4 $y(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b$
  - (Using Ch.4 notation $\boldsymbol{x}$ for input rather than $\phi(\boldsymbol{x})$.)
- $\frac{\boldsymbol{w}^T\boldsymbol{x}}{||\boldsymbol{w}||} - \frac{-b}{||\boldsymbol{w}||} = \frac{y(\boldsymbol{x})}{||\boldsymbol{w}||}$ is signed distance to decision boundary.

Support Vectors



- Assuming data are separated by the hyperplane, distance to decision boundary is $\frac{t_n y(\boldsymbol{x}_n)}{||\boldsymbol{w}||}$
- The maximum margin criterion chooses $\boldsymbol{w}, b$ by:

$$\arg\max_{\boldsymbol{w},b} \left\{ \frac{1}{||\boldsymbol{w}||} \min_n [t_n(\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n) + b)] \right\}$$

- Points with this min value are known as support vectors

Canonical Representation

- This optimization problem is complex:

$$\arg\max_{\boldsymbol{w},b}\left\{\frac{1}{||\boldsymbol{w}||}\min_{n}[t_n(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n)+b)]\right\}$$

- Note that rescaling $\boldsymbol{w}\rightarrow\kappa\boldsymbol{w}$ and $b\rightarrow\kappa b$ does not change distance $\frac{t_n y(\boldsymbol{x}_n)}{||\boldsymbol{w}||}$ (many equiv. answers)
- So for $\boldsymbol{x}_*$ closest to surface, can use $\boldsymbol{w}\rightarrow\boldsymbol{w}/y(\boldsymbol{x}_*)$ and $b\rightarrow b/y(\boldsymbol{x}_*)$ so that:

$$t_*(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_*)+b)=1$$

- All other points are at least this far away:

$$\forall n\ ,\ t_n(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n)+b)\geq 1$$

- Under these constraints, the optimization becomes:

$$\arg\max_{\boldsymbol{w},b}\frac{1}{||\boldsymbol{w}||}=\arg\min_{\boldsymbol{w},b}\frac{1}{2}||\boldsymbol{w}||^2$$

Canonical Representation

- This optimization problem is complex:

$$\arg\max_{\boldsymbol{w},b}\left\{\frac{1}{||\boldsymbol{w}||}\min_{n}[t_n(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n)+b)]\right\}$$

- Note that rescaling $\boldsymbol{w}\rightarrow\kappa\boldsymbol{w}$ and $b\rightarrow\kappa b$ does not change distance $\frac{t_n y(\boldsymbol{x}_n)}{||\boldsymbol{w}||}$ (many equiv. answers)
- So for $\boldsymbol{x}_*$ closest to surface, can use $\boldsymbol{w}\rightarrow\boldsymbol{w}/y(\boldsymbol{x}_*)$ and $b\rightarrow b/y(\boldsymbol{x}_*)$ so that:

$$t_*(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_*)+b)=1$$

- All other points are at least this far away:

$$\forall n\ ,\ t_n(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n)+b)\geq 1$$

- Under these constraints, the optimization becomes:

$$\arg\max_{\boldsymbol{w},b}\frac{1}{||\boldsymbol{w}||}=\arg\min_{\boldsymbol{w},b}\frac{1}{2}||\boldsymbol{w}||^2$$

Canonical Representation

- This optimization problem is complex:

$$\arg\max_{\boldsymbol{w},b}\left\{\frac{1}{||\boldsymbol{w}||}\min_{n}[t_n(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n)+b)]\right\}$$

- Note that rescaling $\boldsymbol{w}\rightarrow\kappa\boldsymbol{w}$ and $b\rightarrow\kappa b$ does not change distance $\frac{t_n y(\boldsymbol{x}_n)}{||\boldsymbol{w}||}$ (many equiv. answers)
- So for $\boldsymbol{x}_*$ closest to surface, can use $\boldsymbol{w}\rightarrow\boldsymbol{w}/y(\boldsymbol{x}_*)$ and $b\rightarrow b/y(\boldsymbol{x}_*)$ so that:

$$t_*(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_*)+b)=1$$

- All other points are at least this far away:

$$\forall n\ ,\ t_n(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n)+b)\geq 1$$

- Under these constraints, the optimization becomes:

$$\arg\max_{\boldsymbol{w},b}\frac{1}{||\boldsymbol{w}||}=\arg\min_{\boldsymbol{w},b}\frac{1}{2}||\boldsymbol{w}||^2$$

Canonical Representation

- This optimization problem is complex:

$$\arg\max_{\boldsymbol{w},b}\left\{\frac{1}{||\boldsymbol{w}||}\min_{n}[t_n(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n)+b)]\right\}$$

- Note that rescaling $\boldsymbol{w}\rightarrow\kappa\boldsymbol{w}$ and $b\rightarrow\kappa b$ does not change distance $\frac{t_n y(\boldsymbol{x}_n)}{||\boldsymbol{w}||}$ (many equiv. answers)
- So for $\boldsymbol{x}_*$ closest to surface, can use $\boldsymbol{w}\rightarrow\boldsymbol{w}/y(\boldsymbol{x}_*)$ and $b\rightarrow b/y(\boldsymbol{x}_*)$ so that:

$$t_*(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_*)+b)=1$$

- All other points are at least this far away:

$$\forall n\ ,\ t_n(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n)+b)\geq 1$$

- Under these constraints, the optimization becomes:

$$\arg\max_{\boldsymbol{w},b}\frac{1}{||\boldsymbol{w}||}=\arg\min_{\boldsymbol{w},b}\frac{1}{2}||\boldsymbol{w}||^2$$

Canonical Representation

- So the optimization problem is now a constrained optimization problem:

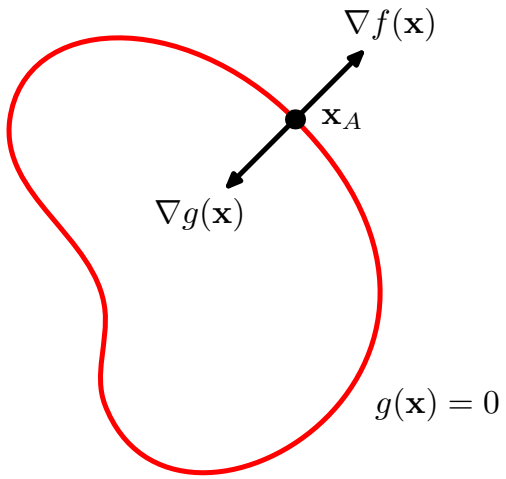
$$\arg\min_{\boldsymbol{w},b} \frac{1}{2}||\boldsymbol{w}||^2$$

$$s.t. \qquad \forall n \ , \ t_n(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n) + b) \geq 1$$

- To solve this, we need to take a detour into Lagrange multipliers

Outline

Lagrange Multipliers



Consider the problem:

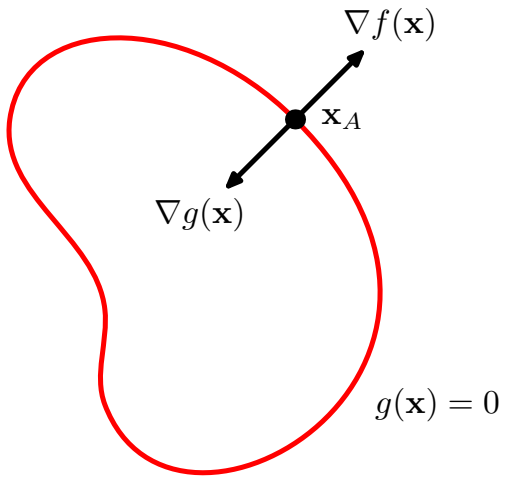
$$\max_{\boldsymbol{x}} f(\boldsymbol{x})$$
$$s.t. \quad g(\boldsymbol{x}) = 0$$

- Points on $g(\boldsymbol{x}) = 0$ must have $\nabla g(\boldsymbol{x})$ normal to surface
- A stationary point must have no change in f in the direction of the constraint surface, so $\nabla f(\boldsymbol{x})$ must also be normal to the surface.
  - So there must be some $\lambda \neq 0$ such that $\nabla f(\boldsymbol{x}) + \lambda \nabla g(\boldsymbol{x}) = 0$
- Define Lagrangian:

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

- Stationary points of $L(\boldsymbol{x}, \lambda)$ have $\nabla_{\boldsymbol{x}} L(\boldsymbol{x}, \lambda) = \nabla f(\boldsymbol{x}) + \lambda \nabla g(\boldsymbol{x}) = 0$ and $\nabla_{\lambda} L(\boldsymbol{x}, \lambda) = g(\boldsymbol{x}) = 0$

Lagrange Multipliers



Consider the problem:

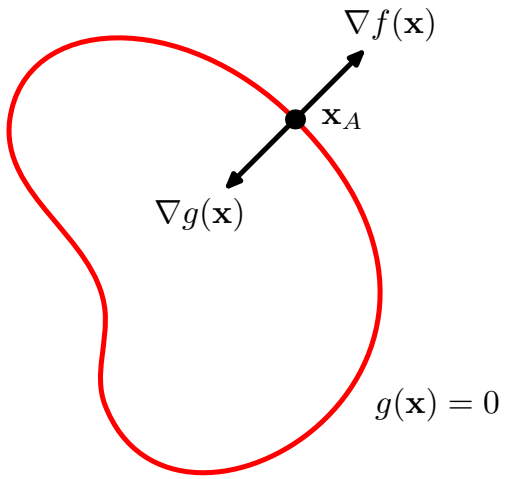
$$\max_{\boldsymbol{x}} f(\boldsymbol{x})$$
$$s.t. \quad g(\boldsymbol{x}) = 0$$

- Points on $g(\boldsymbol{x}) = 0$ must have $\nabla g(\boldsymbol{x})$ normal to surface
- A stationary point must have no change in f in the direction of the constraint surface, so $\nabla f(\boldsymbol{x})$ must also be normal to the surface.
  - So there must be some $\lambda \neq 0$ such that $\nabla f(\boldsymbol{x}) + \lambda \nabla g(\boldsymbol{x}) = 0$
- Define Lagrangian:

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

- Stationary points of $L(\boldsymbol{x}, \lambda)$ have $\nabla_{\boldsymbol{x}} L(\boldsymbol{x}, \lambda) = \nabla f(\boldsymbol{x}) + \lambda \nabla g(\boldsymbol{x}) = 0$ and $\nabla_{\lambda} L(\boldsymbol{x}, \lambda) = g(\boldsymbol{x}) = 0$

Lagrange Multipliers



Consider the problem:

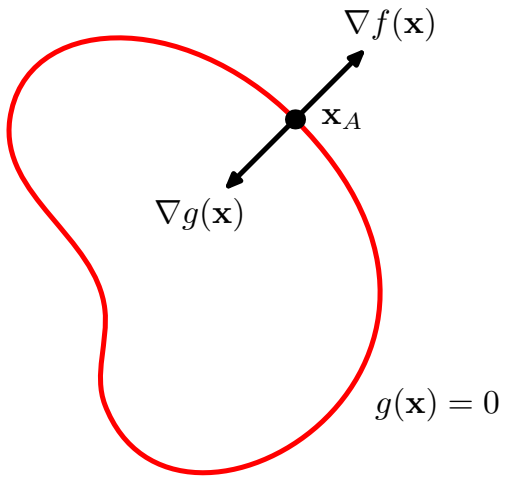
$$\max_{\boldsymbol{x}} f(\boldsymbol{x})$$
$$s.t. \quad g(\boldsymbol{x}) = 0$$

- Points on $g(\boldsymbol{x}) = 0$ must have $\nabla g(\boldsymbol{x})$ normal to surface
- A stationary point must have no change in f in the direction of the constraint surface, so $\nabla f(\boldsymbol{x})$ must also be normal to the surface.
  - So there must be some $\lambda \neq 0$ such that $\nabla f(\boldsymbol{x}) + \lambda \nabla g(\boldsymbol{x}) = 0$
- Define Lagrangian:

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

- Stationary points of $L(\boldsymbol{x}, \lambda)$ have $\nabla_{\boldsymbol{x}} L(\boldsymbol{x}, \lambda) = \nabla f(\boldsymbol{x}) + \lambda \nabla g(\boldsymbol{x}) = 0$ and $\nabla_{\lambda} L(\boldsymbol{x}, \lambda) = g(\boldsymbol{x}) = 0$

Lagrange Multipliers



Consider the problem:

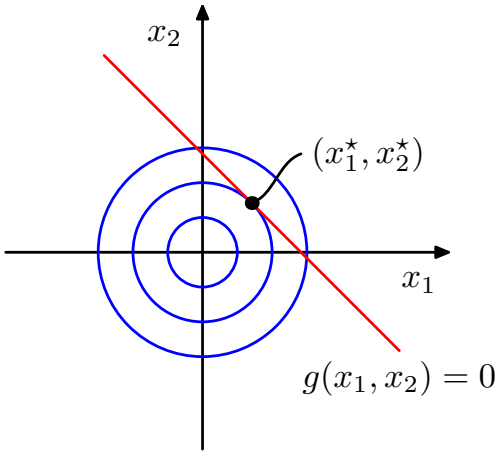
$$\max_{\boldsymbol{x}} f(\boldsymbol{x})$$
$$s.t. \quad g(\boldsymbol{x}) = 0$$

- Points on $g(\boldsymbol{x}) = 0$ must have $\nabla g(\boldsymbol{x})$ normal to surface
- A stationary point must have no change in f in the direction of the constraint surface, so $\nabla f(\boldsymbol{x})$ must also be normal to the surface.
  - So there must be some $\lambda \neq 0$ such that $\nabla f(\boldsymbol{x}) + \lambda \nabla g(\boldsymbol{x}) = 0$
- Define Lagrangian:

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

- Stationary points of $L(\boldsymbol{x}, \lambda)$ have $\nabla_{\boldsymbol{x}} L(\boldsymbol{x}, \lambda) = \nabla f(\boldsymbol{x}) + \lambda \nabla g(\boldsymbol{x}) = 0$ and $\nabla_{\lambda} L(\boldsymbol{x}, \lambda) = g(\boldsymbol{x}) = 0$

Lagrange Multipliers Example



- Consider the problem

$$\max_{\boldsymbol{x}} f(x_1, x_2) = 1 - x_1^2 - x_2^2$$
$$s.t. \quad g(x_1, x_2) = x_1 + x_2 - 1 = 0$$

- Lagrangian:

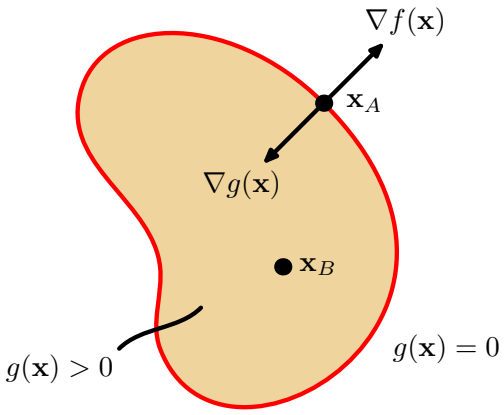
$$L(\boldsymbol{x}, \lambda) = 1 - x_1^2 - x_2^2 + \lambda(x_1 + x_2 - 1)$$

- Stationary points require:

$$\begin{aligned} \partial L/\partial x_1 &= -2x_1 + \lambda = 0 \\ \partial L/\partial x_2 &= -2x_2 + \lambda = 0 \\ \partial L/\partial \lambda &= x_1 + x_2 - 1 = 0 \end{aligned}$$

- So stationary point is $(x_1^*, x_2^*) = (\frac{1}{2}, \frac{1}{2})$, $\lambda = 1$

Lagrange Multipliers - Inequality Constraints



Consider the problem:

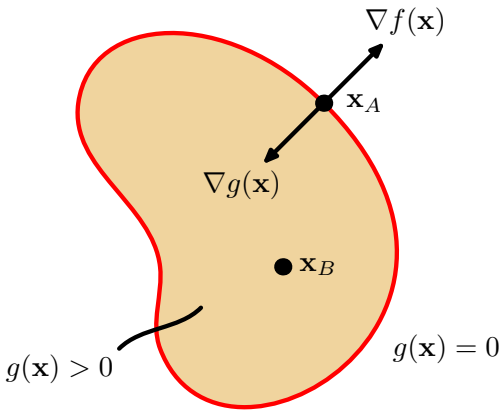
$$\max_{\boldsymbol{x}} f(\boldsymbol{x})$$
$$s.t. \quad g(\boldsymbol{x}) \geq 0$$

- Optimization over a region – solutions either at stationary points (gradients 0) in region $g(\boldsymbol{x}) > 0$ or on boundary

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

- Solutions at stationary points of the Lagrangian with either:
  - $\nabla f(\boldsymbol{x}) = 0$ and $\lambda = 0$ (in region)
  - $\nabla f(\boldsymbol{x}) = -\lambda \nabla g(\boldsymbol{x})$ and $\lambda > 0$ (boundary, $>$ for maximizing f)
  - For both, $\lambda g(\boldsymbol{x}) = 0$
- Find stationary points of L s. t. $g(\boldsymbol{x}) \geq 0, \lambda \geq 0, \lambda g(\boldsymbol{x}) = 0$

Lagrange Multipliers - Inequality Constraints



Consider the problem:

$$\max_{\boldsymbol{x}} f(\boldsymbol{x})$$
$$s.t. \quad g(\boldsymbol{x}) \geq 0$$

- Optimization over a region – solutions either at stationary points (gradients 0) in region $g(\boldsymbol{x}) > 0$ or on boundary

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

- Solutions at stationary points of the Lagrangian with either:
  - $\nabla f(\boldsymbol{x}) = 0$ and $\lambda = 0$ (in region)
  - $\nabla f(\boldsymbol{x}) = -\lambda \nabla g(\boldsymbol{x})$ and $\lambda > 0$ (boundary, $>$ for maximizing f)
  - For both, $\lambda g(\boldsymbol{x}) = 0$
- Find stationary points of L s. t. $g(\boldsymbol{x}) \geq 0, \lambda \geq 0, \lambda g(\boldsymbol{x}) = 0$

Outline

Now Where Were We

- So the optimization problem is now a constrained optimization problem:

$$\arg\min_{\boldsymbol{w},b} \frac{||\boldsymbol{w}||^2}{2}$$
$$s.t. \qquad \forall n \ , \ t_n(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n) + b) \geq 1$$

- For this problem, the Lagrangian (with N multipliers a_n) is:

$$L(\boldsymbol{w}, b, \boldsymbol{a}) = \frac{||\boldsymbol{w}||^2}{2} - \sum_{n=1}^{N} a_n \left\{ t_n(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n) + b) - 1 \right\}$$

- We can find the derivatives of L wrt $\boldsymbol{w}, b$ and set to 0:

$$\boldsymbol{w} = \sum_{n=1}^{N} a_n t_n \boldsymbol{\phi}(\boldsymbol{x}_n)$$
$$0 = \sum_{n=1}^{N} a_n t_n$$

Dual Formulation

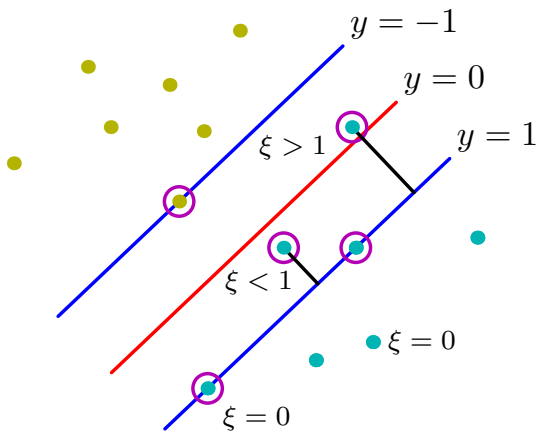
- Plugging those equations into L removes $\boldsymbol{w}$ and b results in a version of L where $\nabla_{\boldsymbol{w},b} L = 0$:

$$\tilde{L}(\boldsymbol{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m \boldsymbol{\phi}(\boldsymbol{x}_n)^T \boldsymbol{\phi}(\boldsymbol{x}_m)$$

  this new $\tilde{L}$ is the dual representation of the problem (maximize with constraints)
- Another formula for finding b, like the bias in linear regression.

Outline

Non-Separable Data

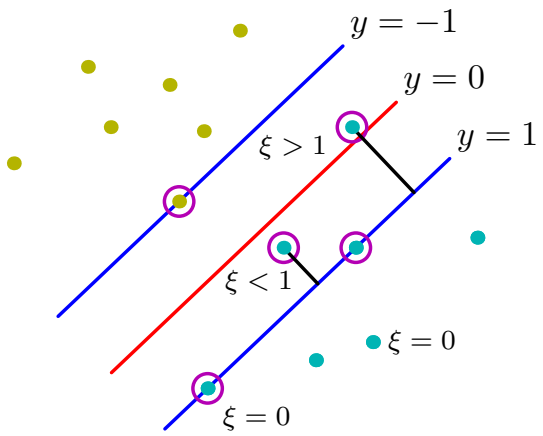


- For most problems, data will not be linearly separable (even in feature space ϕ)
- Can relax the constraints from

$$t_n y(\boldsymbol{x}_n) \geq 1 \quad \text{to} \quad t_n y(\boldsymbol{x}_n) \geq 1 - \xi_n$$

- The $\xi_n \geq 0$ are called slack variables
  - $\xi_n = 0$, satisfy original problem, so x_n is on margin or correct side of margin
  - $0 < \xi_n < 1$, inside margin, but still correctly classifed
  - $\xi_n > 1$, mis-classified

Loss Function For Non-separable Data



- Non-zero slack variables are bad, penalize while maximizing the margin:

$$\min C \sum_{n=1}^{N} \xi_n + \frac{1}{2}||\boldsymbol{w}||^2$$

- Constant $C > 0$ controls importance of large margin versus incorrect (non-zero slack)
  - Set using cross-validation
- Optimization is same quadratic, different constraints, convex

SVM Loss Function

- The SVM for the separable case solved the problem:

$$\arg\min_{\boldsymbol{w}} \frac{1}{2}||\boldsymbol{w}||^2$$
$$s.t. \qquad \forall n \ , \ t_n y_n \geq 1$$

- Can write this as:

$$\arg\min_{w} \sum_{n=1}^{N} E_\infty(t_n y_n - 1) + \lambda||\boldsymbol{w}||^2$$

where $E_\infty(z) = 0$ if $z \geq 0$, ∞ otherwise

- Non-separable case relaxes this to be:

$$\arg\min_{w} \sum_{n=1}^{N} E_{SV}(t_n y_n - 1) + \lambda||\boldsymbol{w}||^2$$

where $E_{SV}(t_n y_n - 1) = [1 - y_n t_n]_+$ hinge loss

  - $[u]_+ = u$ if $u \geq 0$, 0 otherwise

SVM Loss Function

- The SVM for the separable case solved the problem:

$$\begin{aligned} & \arg\min_{\boldsymbol{w}} \frac{1}{2}||\boldsymbol{w}||^2 \\ s.t. \quad & \forall n \ , \ t_n y_n \geq 1 \end{aligned}$$

- Can write this as:

$$\arg\min_{\boldsymbol{w}} \sum_{n=1}^{N} E_\infty(t_n y_n - 1) + \lambda||\boldsymbol{w}||^2$$

where $E_\infty(z) = 0$ if $z \geq 0$, ∞ otherwise

- Non-separable case relaxes this to be:

$$\arg\min_{\boldsymbol{w}} \sum_{n=1}^{N} E_{SV}(t_n y_n - 1) + \lambda||\boldsymbol{w}||^2$$

where $E_{SV}(t_n y_n - 1) = [1 - y_n t_n]_+$ hinge loss

  - $[u]_+ = u$ if $u \geq 0$, 0 otherwise

SVM Loss Function

- The SVM for the separable case solved the problem:

$$\begin{aligned} & \arg\min_{\boldsymbol{w}} \frac{1}{2}||\boldsymbol{w}||^2 \\ s.t. \quad & \forall n \ , \ t_n y_n \geq 1 \end{aligned}$$

- Can write this as:

$$\arg\min_{\boldsymbol{w}} \sum_{n=1}^{N} E_\infty(t_n y_n - 1) + \lambda||\boldsymbol{w}||^2$$

where $E_\infty(z) = 0$ if $z \geq 0$, ∞ otherwise

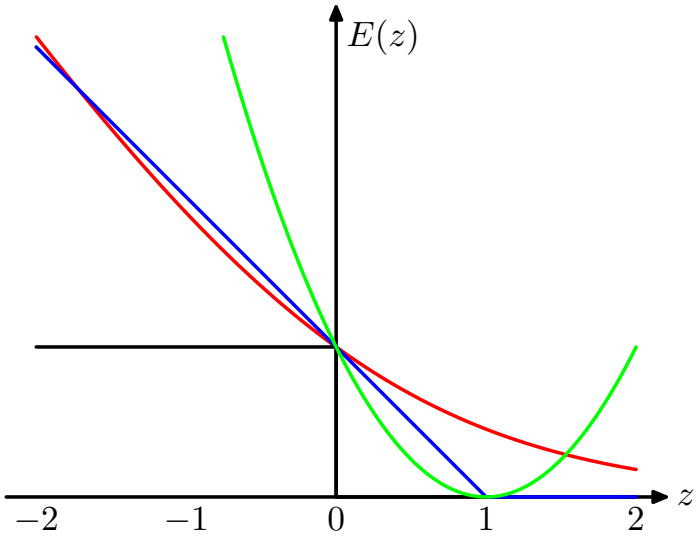
- Non-separable case relaxes this to be:

$$\arg\min_{\boldsymbol{w}} \sum_{n=1}^{N} E_{SV}(t_n y_n - 1) + \lambda||\boldsymbol{w}||^2$$

where $E_{SV}(t_n y_n - 1) = [1 - y_n t_n]_+$ hinge loss

  - $[u]_+ = u$ if $u \geq 0$, 0 otherwise

Loss Functions



- Linear classifiers, compare loss function used for learning
- $z = y_n t_n \geq 1$ iff there is an error (with $t_n \in \{+1, -1\}$).
  - Black is misclassification error
  - Transformed simple linear classifier, squared error: $(y_n - t_n)^2$
  - Transformed logistic regression, cross-entropy error: $t_n \ln y_n$
  - SVM, hinge loss: $\xi_n = [1 - y_n t_n]_+$
    - positive *only* if there is a mistake, otherwise 0.
    - encourages sparse solutions.

Two Views of Learning as Optimization

- The original SVM goal was of the form:
  - Find the simplest hypothesis that is consistent with the data, or
  - Maximize simplicity, given a consistency constraint.
- This general idea appears in much scientific model building, in image processing, and other applications.
- Bayesian methods use a criterion of the form
  - Find a trade-off between simplicity and data fit, or
  - Maximize sum of the type (data fit - λ simplicity)
  - e.g., $ln(P(D|M)) - \lambda ln(P(M))$ where the model prior M is higher for simpler models.

Pros and Cons of Learning Criteria

- The Bayesian approach has a solid probabilistic foundation in Bayes' theorem.
- Seems to be especially suitable for noisy data.
- The constraint-based approach is often easy for users to understand.
- Often leads to sparser simpler models.
- Suitable for "clean" data.

Conclusion

- Readings: Ch. 7 up to and including Ch. 7.1.2
- Many algorithms can be re-written with only dot products of features
  - We've seen NN, perceptron, regression; also PCA, SVMs (later)
- Maximum margin criterion for deciding on decision boundary
  - Linearly separable data
- Relax with slack variables for non-separable case
- Global optimization is possible in both cases
  - Convex problem (no local optima)
  - Descent methods converge to global optimum

Logistic Regression Learning: Iterative Reweighted Least Squares

- Iterative reweighted least squares (IRLS) is a descent method
  - As in gradient descent, start with an initial guess, improve it
  - Gradient descent - take a step (how large?) in the gradient direction
- IRLS is a special case of a Newton-Raphson method
  - Approximate function using second-order Taylor expansion:

$$\hat{f}(w+v) = f(w) + \nabla f(w)^T (v-w) + \frac{1}{2}(v-w)^T \nabla^2 f(w)(v-w)$$

  - Closed-form solution to minimize this is straight-forward: quadratic, derivatives linear
- In IRLS this second-order Taylor expansion ends up being a weighted least-squares problem, as in the linear regression case
  - Hence the name IRLS

Logistic Regression Learning: Iterative Reweighted Least Squares

- Iterative reweighted least squares (IRLS) is a descent method
  - As in gradient descent, start with an initial guess, improve it
  - Gradient descent - take a step (how large?) in the gradient direction
- IRLS is a special case of a Newton-Raphson method
  - Approximate function using second-order Taylor expansion:

$$\hat{f}(\boldsymbol{w}+\boldsymbol{v}) = f(\boldsymbol{w}) + \nabla f(\boldsymbol{w})^T(\boldsymbol{v}-\boldsymbol{w}) + \frac{1}{2}(\boldsymbol{v}-\boldsymbol{w})^T\nabla^2 f(\boldsymbol{w})(\boldsymbol{v}-\boldsymbol{w})$$

  - Closed-form solution to minimize this is straight-forward: quadratic, derivatives linear
- In IRLS this second-order Taylor expansion ends up being a weighted least-squares problem, as in the linear regression case
  - Hence the name IRLS

Logistic Regression Learning: Iterative Reweighted Least Squares

- Iterative reweighted least squares (IRLS) is a descent method
  - As in gradient descent, start with an initial guess, improve it
  - Gradient descent - take a step (how large?) in the gradient direction
- IRLS is a special case of a Newton-Raphson method
  - Approximate function using second-order Taylor expansion:

$$\hat{f}(\boldsymbol{w}+\boldsymbol{v}) = f(\boldsymbol{w}) + \nabla f(\boldsymbol{w})^T(\boldsymbol{v}-\boldsymbol{w}) + \frac{1}{2}(\boldsymbol{v}-\boldsymbol{w})^T\nabla^2 f(\boldsymbol{w})(\boldsymbol{v}-\boldsymbol{w})$$

  - Closed-form solution to minimize this is straight-forward: quadratic, derivatives linear
- In IRLS this second-order Taylor expansion ends up being a weighted least-squares problem, as in the linear regression case
  - Hence the name IRLS

Logistic Regression Learning: Iterative Reweighted Least Squares

- Iterative reweighted least squares (IRLS) is a descent method
  - As in gradient descent, start with an initial guess, improve it
  - Gradient descent - take a step (how large?) in the gradient direction
- IRLS is a special case of a Newton-Raphson method
  - Approximate function using second-order Taylor expansion:

$$\hat{f}(\boldsymbol{w}+\boldsymbol{v}) = f(\boldsymbol{w}) + \nabla f(\boldsymbol{w})^T(\boldsymbol{v}-\boldsymbol{w}) + \frac{1}{2}(\boldsymbol{v}-\boldsymbol{w})^T\nabla^2 f(\boldsymbol{w})(\boldsymbol{v}-\boldsymbol{w})$$

  - Closed-form solution to minimize this is straight-forward: quadratic, derivatives linear
- In IRLS this second-order Taylor expansion ends up being a weighted least-squares problem, as in the linear regression case
  - Hence the name IRLS

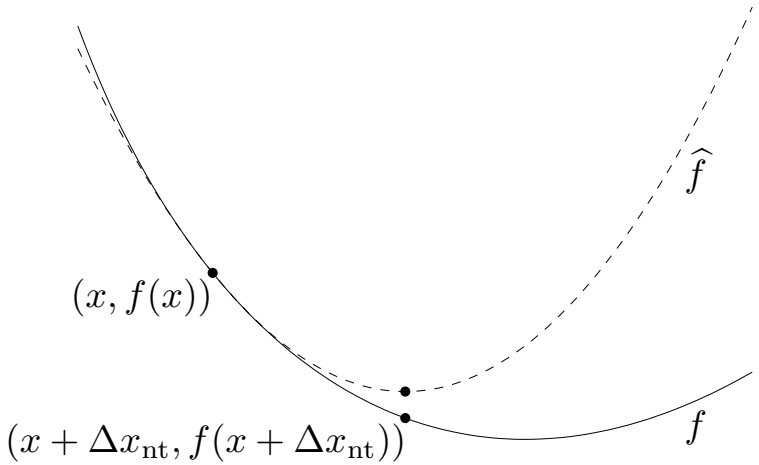
Logistic Regression Learning: Iterative Reweighted Least Squares

- Iterative reweighted least squares (IRLS) is a descent method
  - As in gradient descent, start with an initial guess, improve it
  - Gradient descent - take a step (how large?) in the gradient direction
- IRLS is a special case of a Newton-Raphson method
  - Approximate function using second-order Taylor expansion:

$$\hat{f}(\boldsymbol{w}+\boldsymbol{v}) = f(\boldsymbol{w}) + \nabla f(\boldsymbol{w})^T(\boldsymbol{v}-\boldsymbol{w}) + \frac{1}{2}(\boldsymbol{v}-\boldsymbol{w})^T\nabla^2 f(\boldsymbol{w})(\boldsymbol{v}-\boldsymbol{w})$$

  - Closed-form solution to minimize this is straight-forward: quadratic, derivatives linear
- In IRLS this second-order Taylor expansion ends up being a weighted least-squares problem, as in the linear regression case
  - Hence the name IRLS

Newton-Raphson



- Figure from Boyd and Vandenberghe, *Convex Optimization*
  - Excellent reference, free for download online
    `http://www.stanford.edu/~boyd/cvxbook/`