CSCE 633: Machine Learning

Lecture 13: SVMs and Kernels

Texas A&M University

9-23-19

Administration

- HW1 Due Tonight. HW2 out now.
- Exam 1 - 10/14/19.
- No Class Wednesday 9/25. There IS discussion Friday 9/27.
- My office hours are canceled this week.

Goals of this lecture

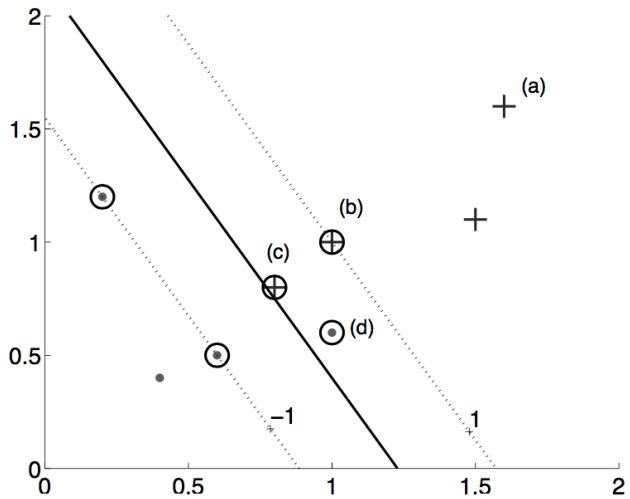
- Support Vector Machines
- Kernels
- Multi-class Classification

Support Vector Machines: Non-separable case

- If two classes are not linearly separable, we look for the hyperplane that yields the least error
- We define slack variables $\xi_n \geq 0$ which represent the deviation from the margin

$$y_n(\beta^T \mathbf{x_n} + \beta_0) \geq 1 - \xi_n$$

- *Case (a):* Far away from the margin, $\xi_n = 0$
- *Case (b):* On the right side and far from margin, $\xi_n = 0$
- *Case (c):* On the right side, but in the margin, $0 \leq \xi_n \leq 1$
- *Case (d):* On the wrong side, $\xi_n \geq 1$



We incorporate the number of misclassifications $\#\{\xi_n > 1\}$ and the number of non-separable points $\#\{\xi_n > 0\}$ as a soft error $\sum_n \xi_n$.

Support Vector Machines: Non-separable case

$$\min \tfrac{1}{2}\|\beta\|_2^2 + C\sum_{n=1}^{N}\xi_n, \;\; \text{s.t.} \;\; y_n(\beta^T\mathbf{x} + \beta_0) \geq 1 - \xi_n \;\; \text{and} \;\; \xi_n \geq 0\,, \;\; \forall n$$

1) Formulate Lagrangian function (primal problem)

$$L_p = \tfrac{1}{2}\|\beta\|_2^2 + C\sum_{n=1}^{N}\xi_n - \sum_{n=1}^{N}\alpha_n\left[y_n(\beta^T\mathbf{x}_n + \beta_0) - 1 + \xi_n\right] - \sum_{n=1}^{N}\mu_n\xi_n$$

2) Minimize Lagrangian to solve for primal variables β, β_0

$$\frac{\partial L_p}{\partial \beta} = 0 \;\; \Rightarrow \;\; \beta = \sum_{n=1}^{N}\alpha_n y_n \mathbf{x_n}$$

$$\frac{\partial L_p}{\partial \beta_0} = 0 \;\; \Rightarrow \;\; \sum_{n=1}^{N}\alpha_n y_n = 0$$

$$\frac{\partial L_p}{\partial \xi_n} = 0 \;\; \Rightarrow \;\; C - \alpha_n - \mu_n = 0$$

3) Substitute the primal variables β, β_0 into the Lagrangian and express in terms of dual variables α_n

$$
\begin{aligned}
L_d &= \frac{1}{2}\|\beta\|_2^2 + C\sum_{n=1}^{N}\xi_n - \beta^T\sum_{n=1}^{N}\alpha_n y_n \mathbf{x_n} \\
&\quad - \beta_0\sum_{n=1}^{N}\alpha_n y_n + \sum_{n=1}^{N}\alpha_n - \sum_{n=1}^{N}\alpha_n\xi_n - \sum_{n=1}^{N}\mu_n\xi_n \\
&= \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{M}\alpha_n\alpha_m y_n y_m \mathbf{x_n}^T\mathbf{x_m} + \sum_{n=1}^{N}\xi_n\left(C - \alpha_n - \mu_n\right) - \\
&\quad - \sum_{m=1}^{N}\alpha_m y_m \mathbf{x_m}^T \cdot \sum_{n=1}^{N}\alpha_n y_n \mathbf{x_n} - \beta_0\sum_{n=1}^{N}\alpha_n y_n + \sum_{n=1}^{N}\alpha_n \\
&= \sum_{n=1}^{N}\alpha_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{M}\alpha_n\alpha_m y_n y_m \mathbf{x_n}^T\mathbf{x_m}
\end{aligned}
$$

Support Vector Machines: Non-separable case

4) Maximize the Lagrangian with respect to dual variables (dual problem)

$$\max_{\alpha_n} \left\{ \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m \mathbf{x_n}^T \mathbf{x_m} \right\}$$

s.t. $\sum_{n=1}^{N} \alpha_n y_n = 0$ and $0 \leq \alpha_n \leq C$, for $n = 1, \ldots, N$

- Solved numerically using quadratic optimization methods
- $\alpha_n = 0$: instances at the correct side of the hyperplane with sufficient margin
- $\alpha_n > 0$: support vectors
  - $0 < \alpha_n < C$: instances lying on the margin (some contribution in the error function)
  - $\alpha_n = C$: instances in the margin or misclassified (largest possible contribution in the error function)

Coordinate Ascent

Unconstrained optimization problem

$$\max_{\boldsymbol{\alpha}} \beta(\alpha_1, \alpha_2, \ldots, \alpha_m)$$

We have already seen gradient descent (or ascent, if we negate the optimization function), now we consider another optimization method called coordinate ascent.

Loop until convergence

1 For $i = 1, \ldots, m$

1a $\alpha_i = \textit{arg}\max_{\hat{\alpha}_i} \beta(\alpha_1, \ldots, \hat{\alpha}_i, \ldots, \alpha_m)$

- In the innermost loop, hold all variables constant except for some fixed α_i
- Re-optimize β with respect to just the parameter α_i
- When *arg* max of the inner loop can be performed efficiently, coordinate ascent can be a fairly efficient algorithm

Sequential Minimal Optimization (SMO)

$$\beta(\boldsymbol{\alpha}) = -\frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{M}\alpha_n\alpha_m y_n y_m \mathbf{x_n}^T\mathbf{x_m} + \sum_{n=1}^{N}\alpha_n$$

s.t. $\sum_{n=1}^{N}\alpha_n y_n = 0$ and $\alpha_n \geq 0$, for $n = 1, \ldots, N$

Loop until convergence

1 For $i = 1, \ldots, m$

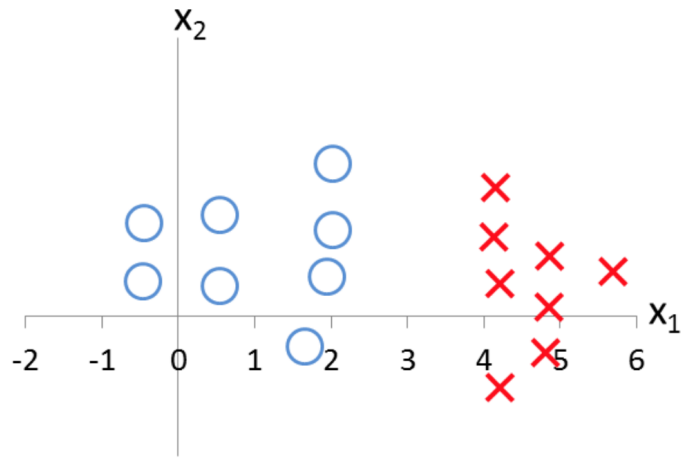
- 1a Select some α_i and α_j to update
- 1b Re-optimize β wrt α_i and α_j while holding all other α_k's fixed

Let's assume that we optimize wrt α_1, α_2, while $\alpha_3, \ldots, \alpha_N$ are constant

- From the constraint:
  $\alpha_1 y_1 + \alpha_2 y_2 = \sum_{n=3}^{N}\alpha_n y_n = \zeta \rightarrow \alpha_1 = (\zeta - \alpha_2 y_2)/y_1$
- The objective is $\beta(\boldsymbol{\alpha}) = \beta\left(\left(\zeta - \alpha_2 y_2\right)/y_1, \alpha_2, \ldots, \alpha_N\right)$ (some quadratic function of $\alpha_2 \rightarrow$ easily solved by setting its derivative to zero)

Support Vector Machines: Linearly separable case

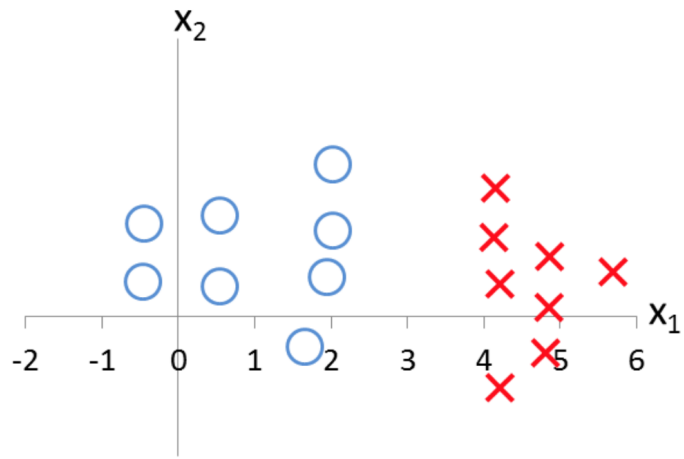
Question: Consider the following training data, where samples denoted by $\times$ belong to class 1 and samples denoted by $\bigcirc$ belong to class -1. What would be the values of the weights of an SVM with $f(x) = \beta_0 + w_1 x_1 + w_2 x_2$?



- A) $\beta_0 = 3,\ w_1 = 1,\ w_2 = 0$
- B) $\beta_0 = -3,\ w_1 = 1,\ w_2 = 0$
- C) $\beta_0 = 3,\ w_1 = 0,\ w_2 = 1$
- D) $\beta_0 = -3,\ w_1 = 0,\ w_2 = 1$

Support Vector Machines: Linearly separable case

Question: Consider the following training data, where samples denoted by $\times$ belong to class 1 and samples denoted by $\bigcirc$ belong to class -1. What would be the values of the weights of an SVM with $f(x) = \beta_0 + w_1 x_1 + w_2 x_2$?



- A) $\beta_0 = 3,\ w_1 = 1,\ w_2 = 0$
- B) $\beta_0 = -3,\ w_1 = 1,\ w_2 = 0$
- C) $\beta_0 = 3,\ w_1 = 0,\ w_2 = 1$
- D) $\beta_0 = -3,\ w_1 = 0,\ w_2 = 1$

The correct answer is B

SVM finds the largest margin line:

If $x_1 - 3 > 0$, then class 1

If $x_1 - 3 < 0$, then class -1

Support Vector Machines: Non-separable case

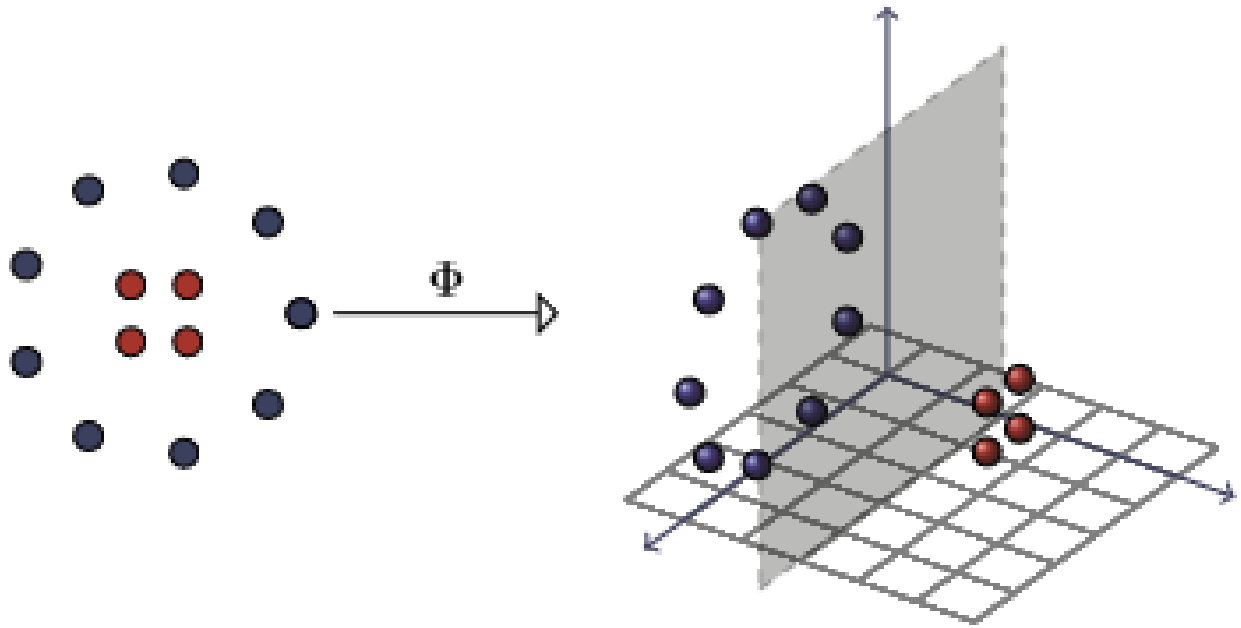
4) Maximize the Lagrangian with respect to dual variables (dual problem)

$$\max_{\alpha_n} \left\{ \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m \mathbf{x_n}^T \mathbf{x_m} \right\}$$

s.t. $\sum_{n=1}^{N} \alpha_n y_n = 0$ and $0 \leq \alpha_n \leq C$, for $n = 1, \ldots, N$

- Solved numerically using quadratic optimization methods
- $\alpha_n = 0$: instances at the correct side of the hyperplane with sufficient margin
- $\alpha_n > 0$: support vectors
  - $0 < \alpha_n < C$: instances lying on the margin (some contribution in the error function)
  - $\alpha_n = C$: instances in the margin or misclassified (largest possible contribution in the error function)

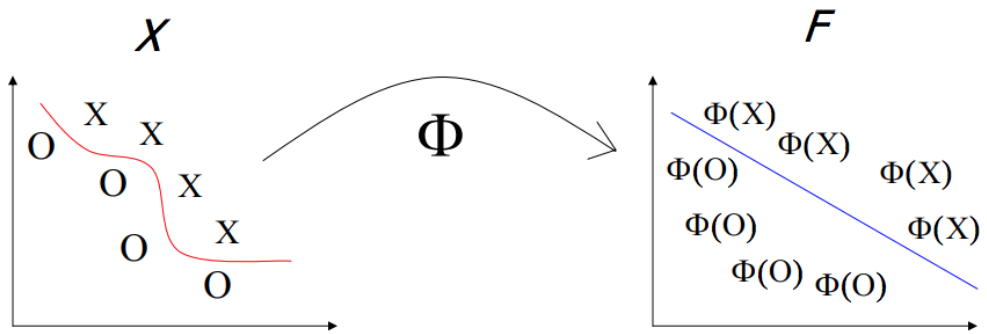
Higher Dimensions



Kernel Functions

Motivation

- Given a set of vectors, there are many tools available for one to use to detect linear relations among the data
- But what if the relations are non-linear in the original space?
- Solution: Map the data into a (possibly high dimensional) vector space where linear relations exist among the data, then apply a linear algorithm in this space



Kernel Functions

A function that takes as its inputs vectors in the original space and returns the dot product of the vectors in the feature space is called a kernel function

Definition

A (positive semi-definite) kernel function $L(\cdot, \cdot)$ is a bivariate function for which for any $\mathbf{x_m}$ and $\mathbf{x_n}$

$$K(\mathbf{x_n}, \mathbf{x_m}) = K(\mathbf{x_m}, \mathbf{x_n}) \text{ and } K(\mathbf{x_n}, \mathbf{x_m}) = \phi^T(\mathbf{x}_n)\phi(\mathbf{x}_m)$$

Examples

$$K(\mathbf{x_n}, \mathbf{x_m}) = (\mathbf{x_n}^T \mathbf{x_m})^2 \quad , \quad K(\mathbf{x_n}, \mathbf{x_m}) = \exp\left(-\frac{\|\mathbf{x_n} - \mathbf{x_m}\|_2^2}{2\sigma^2}\right)$$

- Using kernels, we do not need to embed the data into the space explicitly, because a number of algorithms only require the inner products between input vectors $\rightarrow$ We never need the coordinates of the data in the feature space
- Kernels can be perceived as similarity measure

Kernel Functions

Example

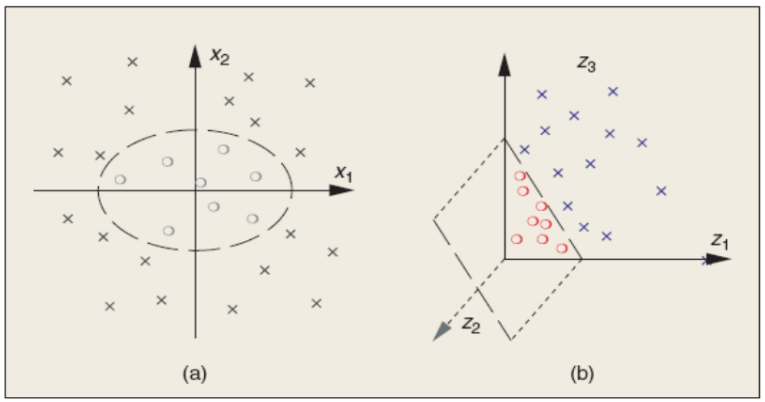
- Consider a two-dimensional input space $\mathcal{X} \subset \mathbb{R}^2$ with the feature map:

$$\phi: \quad \mathbf{x} = [x_1\ x_2]^T \quad \rightarrow \quad \phi(\mathbf{x}) = [x_1^2\ x_2^2\ \sqrt{2}x_1x_2]^T \in \mathbb{R}^3$$

- The inner product in the feature space is

$$\phi(\mathbf{x})\phi(\mathbf{z}) = \left\langle \left(x_1^2\ x_2^2\ \sqrt{2}x_1x_2\right), \left(z_1^2\ z_2^2\ \sqrt{2}z_1z_2\right) \right\rangle$$
$$= x_1^2z_1^2 + x_2^2z_2^2 + 2x_1x_2z_1z_2 = (x_1z_1 + x_2z_2)^2 = \langle \mathbf{x}, \mathbf{z} \rangle^2$$

- Then $K(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x}, \mathbf{z} \rangle^2$



▲ 1. Effect of the map $\phi(x_1, x_2) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$. (a) Input space $\mathcal{X}$ and (b) feature space $\mathcal{H}$.

Kernel Functions

Mercer's condition

A bivariate function $K(\cdot,\cdot)$ is a positive semidefinite kernel function, if and only if, for any N and any $\mathbf{x_1}, \ldots \mathbf{x_N}$ the following matrix, called Gram matrix, is positive semi-definite.

$$\mathbf{K} = \begin{bmatrix} \phi(\mathbf{x_1})\phi(\mathbf{x_1}) & \phi(\mathbf{x_1})\phi(\mathbf{x_2}) & \ldots & \phi(\mathbf{x_1})\phi(\mathbf{x_N}) \\ \phi(\mathbf{x_2})\phi(\mathbf{x_1}) & \phi(\mathbf{x_2})\phi(\mathbf{x_2}) & \ldots & \phi(\mathbf{x_2})\phi(\mathbf{x_N}) \\ \vdots & \vdots & \vdots & \vdots \\ \phi(\mathbf{x_N})\phi(\mathbf{x_1}) & \phi(\mathbf{x_N})\phi(\mathbf{x_2}) & \ldots & \phi(\mathbf{x_N})\phi(\mathbf{x_N}) \end{bmatrix} = \boldsymbol{\Phi}^T\boldsymbol{\Phi}$$

Mercer's condition tells us whether or not a prospective kernel is actually a dot product in some space

Examples of Kernel Functions

Polynomial kernel function with degree of d

$$K(\mathbf{x_n}, \mathbf{x_m}) = (\mathbf{x_n}^T \mathbf{x_m} + c)^d$$

for $c \geq 0$ and d a positive integer.

Gaussian kernel, RBF (radial basis function) kernel, or Gaussian RBF kernel

$$K(\mathbf{x_n}, \mathbf{x_m}) = \exp\left(-\frac{\|\mathbf{x_n} - \mathbf{x_m}\|_2^2}{2\sigma^2}\right)$$

Sigmoid Kernel

$$K(\mathbf{x_n}, \mathbf{x_m}) = \tanh(a \cdot (x_n, x_m) + b), a, b/geq0$$

Most of those kernels have parameters to be tuned: d, c, σ^2, etc. They are hyper parameters and are often tuned on holdout data or with cross-validation.

Examples of Kernel Functions

Document similarity

- Let x_{ij} be the #times word j occurs in document i (bag-of-words)
- Cosine similarity between documents i and i' counts the number of shared words

$$K(\mathbf{x_i}, \mathbf{x_{i'}}) = \frac{\mathbf{x_i}^T \mathbf{x_{i'}}}{\|\mathbf{x_i}^T\|_2 \|\mathbf{x_{i'}}^T\|_2}$$

Edit distance (e.g. gene alignment)

- # insertions, deletions, substitutions it takes to convert one gene to another

Rules for composing kernels

There are infinite number of kernels to use

- If $K(\mathbf{x_n}, \mathbf{x_m})$ is a kernel, then $cK(\mathbf{x_n}, \mathbf{x_m})$, $c > 0$, is a kernel
- If $K(\mathbf{x_n}, \mathbf{x_m})$ is a kernel, then $\exp^{K(\mathbf{x_n}, \mathbf{x_m})}$ is a kernel
- If $K_1(\mathbf{x_n}, \mathbf{x_m})$ and $K_2(\mathbf{x_n}, \mathbf{x_m})$ are kernels, then $\alpha K_1(\mathbf{x_n}, \mathbf{x_m}) + \beta K_2(\mathbf{x_n}, \mathbf{x_m})$, $\alpha, \beta > 0$, is a kernel
- If $K_1(\mathbf{x_n}, \mathbf{x_m})$ and $K_2(\mathbf{x_n}, \mathbf{x_m})$ are kernels, then $K_1(\mathbf{x_n}, \mathbf{x_m})K_2(\mathbf{x_n}, \mathbf{x_m})$ is a kernel

In practice, using which kernel, or which kernels to compose a new kernel, remains somewhat as "black art", though most people will start with polynomial and Gaussian RBF kernels.

Example: Audio-visual speech recognition, where notion of similarity is differently defined for speech and image

Kernelization trick

- Many learning methods depend on computing inner products between features, e.g. linear regression.
- For those methods, we can use a kernel function in the place of the inner products, i.e., "kernelizing" the methods, thus, introducing nonlinear features/basis.
- Instead of first transforming the original features into the new feature space and then computing the inner product, we can compute the inner product in the new feature space directly through the kernel function

Support Vector Machines: Kernel Trick

- Set of basis functions $\mathbf{z} = \phi(\mathbf{x})$
- Map the problem into the new space and solve as before
- For SVM, this leads to certain simplifications

Setup for two classes

- Input: $\mathbf{x} \in \mathbb{R}^D$
- Output: $y \in \{-1, 1\}$
- Training data: $\mathcal{D}^{train} = \{(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_N}, y_N)\}$
- Non-separable model:

$$f(\mathbf{x_n}) = \begin{cases} 1, & \text{if } \beta^T \phi(\mathbf{x_n}) + \beta_0 \geq 1 - \xi_n \\ -1, & \text{if } \beta^T \phi(\mathbf{x_n}) + \beta_0 \leq -(1 - \xi_n) \end{cases}$$

Support Vector Machines: Kernel Trick

$$\min \tfrac{1}{2}\|\beta\|_2^2 + C\sum_{n=1}^{N}\xi_n, \;\; \text{s.t.} \;\; y_n(\beta^T\phi(\mathbf{x}) + \beta_0) \geq 1 - \xi_n \;\; \text{and} \;\; \xi_n \geq 0\,, \;\; \forall n$$

1) Formulate Lagrangian function (primal problem)

$$L_p = \tfrac{1}{2}\|\beta\|_2^2 + C\sum_{n=1}^{N}\xi_n - \sum_{n=1}^{N}\alpha_n\left[y_n(\beta^T\phi(\mathbf{x}_n) + \beta_0) - 1 + \xi_n\right] - \sum_{n=1}^{N}\mu_n\xi_n$$

2) Minimize Lagrangian to solve for primal variables β, β_0

$$\frac{\partial L_p}{\partial \beta} = 0 \;\; \Rightarrow \;\; \beta = \sum_{n=1}^{N}\alpha_n y_n \phi(\mathbf{x_n})$$

$$\frac{\partial L_p}{\partial \beta_0} = 0 \;\; \Rightarrow \;\; \sum_{n=1}^{N}\alpha_n y_n = 0$$

$$\frac{\partial L_p}{\partial \xi_n} = 0 \;\; \Rightarrow \;\; C - \alpha_n - \mu_n = 0$$

3) Substitute the primal variables β, β_0 into the Lagrangian and express in terms of dual variables α_n

$$L_d = \sum_{n=1}^{N}\alpha_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{M}\alpha_n\alpha_m y_n y_m \underbrace{\phi^T(\mathbf{x}_n)\phi(\mathbf{x}_m)}_{K(\mathbf{x_n},\mathbf{x_m})}$$

$K(\mathbf{x_n}, \mathbf{x_m})$ is called Kernel function

Support Vector Machines: Kernel Trick

- Instead of transforming $\mathbf{x_n}$, $\mathbf{x_m}$ through ϕ and computing their inner product, we directly apply the kernel function to the original space
- Lagrangian for the dual problem

$$L_d = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{M} \alpha_n \alpha_m y_n y_m K(\mathbf{x_n}, \mathbf{x_m})$$

$$K(\mathbf{x_n}, \mathbf{x_m}) = \phi^T(\mathbf{x}_n)\phi(\mathbf{x}_m)$$

- Similarly for taking a decision in the test set

$$f(\mathbf{x}) = \beta^T \phi(\mathbf{x}) = \left(\sum_{n=1}^{N} \alpha_n y_n \phi(\mathbf{x_n}) \right)^T \phi(\mathbf{x})$$

$$= \sum_{n=1}^{N} \alpha_n y_n \phi^T(\mathbf{x_n})\phi(\mathbf{x}) = \sum_{n=1}^{N} \alpha_n y_n K(\phi(\mathbf{x_n}), \phi(\mathbf{x}))$$

- The cost of computing the primal variables is $O(p^3)$, while for the dual variables is $O(n^3)$ $\rightarrow$ a kernel method can be useful in high dimensional settings

Support Vector Machines: Multiple Kernel Learning

- Weighted sum of kernels: $K(\mathbf{x_n}, \mathbf{x_m}) = \sum_{l=1}^{L} \nu_l K_l(\mathbf{x_n}, \mathbf{x_m})$, $\nu_l \geq 0$
- Objective function

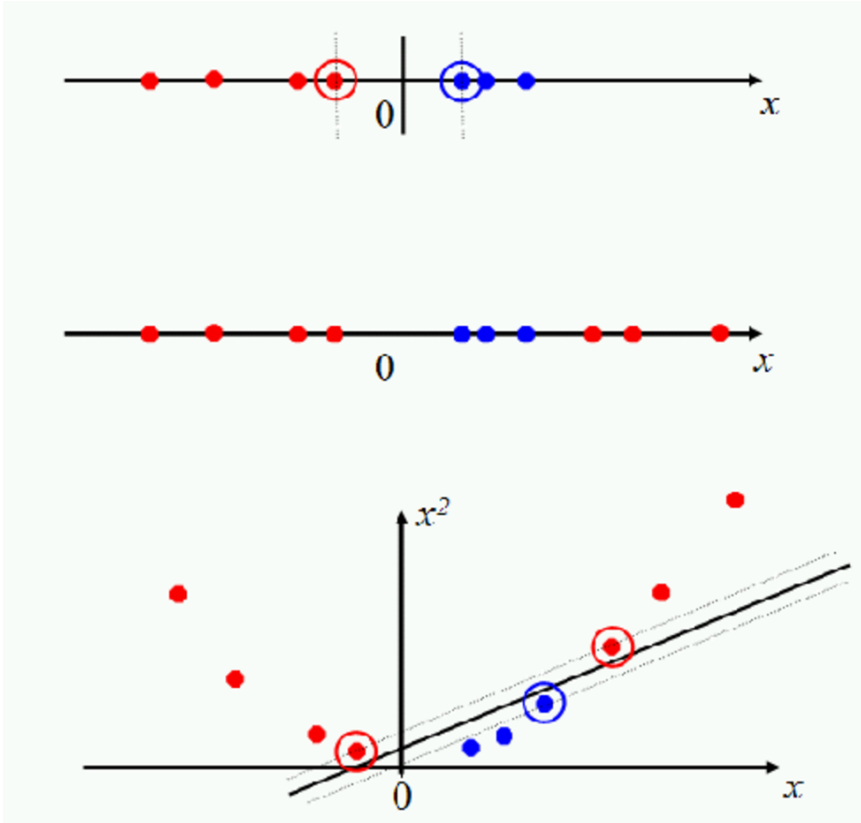
$$L_d = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{M} \alpha_n \alpha_m y_n y_m \sum_{l=1}^{L} \nu_l K_l(\mathbf{x_n}, \mathbf{x_m})$$

  Solved for both the support vectors α_n and the weights ν_l

- For taking a decision in the test set

$$f(\mathbf{x}) = \sum_{n=1}^{N} \alpha_n y_n \sum_{l=1}^{L} K_l(\phi(\mathbf{x_n}), \phi(\mathbf{x}))$$

Support Vector Machines: Multiple Kernel Learning



Projecting data that is not linearly separable into a higher dimensional space can make it linearly separable

Multi-class kernel machines

One-vs-all approach

- Define K two-class problems, each separating one class from all others
- Learn K binary support vector machines $f_i(\mathbf{x})$, $i = 1, \ldots, K$
  - Class 1: Examples from class i
  - Class -1: Examples from all classes besides class i, i.e. $\{1, \ldots, K\} \setminus i$
- Decision during testing

$$f(\mathbf{x}) = arg \max_i f_i(\mathbf{x})$$

Multi-class kernel machines

One-vs-one approach

- Define $K(K-1)$ two-class problems, each separating class i from class j
- Learn $K(K-1)$ binary support vector machines $f_{ij}(\mathbf{x})$
  - Class 1: Examples from class i
  - Class -1: Examples from class j
  - Note that $f_{ij} = -f_{ji}$
- Decision during testing

$$f(\mathbf{x}) = arg\max_i \left(\sum_j f_i(\mathbf{x}) \right)$$

Multi-class kernel machines

Comparison of one-vs-all and one-vs-one approach

- One-vs-one
  - requires $O(K^2)$ classifiers instead of $O(K)$
  - but each classifier is on average smaller $O(2\frac{n}{K})$
- One-vs.-all approach solves $O(K)$ separate problems, each of size $O(n)$

Multi-class kernel machines

Multi-class formulation

- Define K weights for each class $\beta_1, \ldots, \beta_K$ and K bias terms $\beta_{01}, \ldots, \beta_{0K}$
- Training data $\{(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_N}, y_N)\}$, $y_n \in \{1, \ldots, K\}$
- Optimization criterion

$$\min_{\beta_1, \ldots \beta_K} \frac{1}{2} \sum_{k=1}^{K} \|\beta_k\|_2^2 + C \sum_{k=1}^{K} \sum_{n=1}^{N} \xi_{ni}$$

s.t. $\beta_{\mathbf{y_n}}^T \mathbf{x_n} + \beta_{0y_n} \geq \beta_{\mathbf{k}}^T \mathbf{x_n} + \beta_{0k} + 2 - \xi_{nk}$, $\forall k \neq y_n$

(i.e. so that the weight for each class yields a sufficient margin form the other classes)

What have we learnt so far

- Kernel functions $K(\mathbf{x_n}, \mathbf{x})$
  - $K(\mathbf{x_n}, \mathbf{x_m}) = K(\mathbf{x_m}, \mathbf{x_n})$ and $K(\mathbf{x_n}, \mathbf{x_m}) = \phi^T(\mathbf{x}_n)\phi(\mathbf{x}_m)$
  - positive definite kernel $\rightarrow$ positive-definite Gram matrix
- Kernel trick
  - Instead of first transforming the original features into the new feature space and then computing the inner product, we can compute the inner product in the new feature space directly through the kernel function
  - Kernel SVM: The cost of computing the primal variables is $O(D^3)$, while for the dual variables is $O(N^3) \rightarrow$ a kernel method can be useful in high dimensional settings
- Multi-class SVM
  - one-vs-one, one-vs-all, multiclass formulation

Takeaways and Next Time

- Support Vector Machines
- Kernels
- Multi-class SVMs
- Next Time: Discussion - SVM with Kernels
- Next Time: Lecture - Trees