CSCI567 Machine Learning (Fall 2014)

Drs. Sha & Liu

{feisha, yanliu.cs}@usc.edu

October 9, 2014
Outline

1. Administration
2. Review of last lecture
3. Support vector machines
4. Geometric Understanding of SVM
Quiz #1

Tuesday Oct 21 6-8pm TTH301

Some exceptions are handled case by case.
Outline

1. Administration

2. Review of last lecture
   - Kernel methods
   - Kernelized machine learning methods

3. Support vector machines

4. Geometric Understanding of SVM
How to do nonlinear prediction without specifying nonlinear basis functions?

**Definition of kernel function**: a (positive semidefinite) kernel function $k(\cdot, \cdot)$ is a bivariate function that satisfies the following properties. For any $x_m$ and $x_n$,

$$k(x_m, x_n) = k(x_n, x_m) \text{ and } k(x_m, x_n) = \phi(x_m)^T \phi(x_n)$$

for some function $\phi(\cdot)$. 
How to do nonlinear prediction without specifying nonlinear basis functions?

Definition of kernel function: a (positive semidefinite) kernel function $k(\cdot, \cdot)$ is a bivariate function that satisfies the following properties. For any $x_m$ and $x_n$,

$$k(x_m, x_n) = k(x_n, x_m) \text{ and } k(x_m, x_n) = \phi(x_m)^T \phi(x_n)$$

for some function $\phi(\cdot)$.

Examples we have seen

$$k(x_m, x_n) = (x_m^T x_n)^2$$

$$k(x_m, x_n) = 2 - \frac{\sin(2\pi(x_{m1} - x_{n1}))}{x_{m1} - x_{n1}} - \frac{\sin(2\pi(x_{m2} - x_{n2}))}{x_{m2} - x_{n2}}$$
Conditions for being a positive semidefinite kernel function

**Mercer theorem** (loosely), a bivariate function $k(\cdot, \cdot)$ is a positive semidefinite kernel function, if and only if, for *any* $N$ and *any* $x_1, x_2, \ldots, x_N$, the matrix

$$K = \begin{pmatrix}
    k(x_1, x_1) & k(x_1, x_2) & \cdots & k(x_1, x_N) \\
    k(x_2, x_1) & k(x_2, x_2) & \cdots & k(x_2, x_N) \\
    \vdots & \vdots & \ddots & \vdots \\
    k(x_N, x_1) & k(x_N, x_2) & \cdots & k(x_N, x_N)
\end{pmatrix}$$

is positive semidefinite. We also refer $k(\cdot, \cdot)$ as a positive semidefinite kernel.
Flashback: why using kernel functions?

without specifying $\phi(\cdot)$, the kernel matrix

$$K = \begin{pmatrix}
  k(x_1, x_1) & k(x_1, x_2) & \cdots & k(x_1, x_N) \\
  k(x_2, x_1) & k(x_2, x_2) & \cdots & k(x_2, x_N) \\
  \vdots & \vdots & \ddots & \vdots \\
  k(x_N, x_1) & k(x_N, x_2) & \cdots & k(x_N, x_N)
\end{pmatrix}$$

is exactly the same as

$$K = \Phi \Phi^T$$

$$= \begin{pmatrix}
  \phi(x_1)^T \phi(x_1) & \phi(x_1)^T \phi(x_2) & \cdots & \phi(x_1)^T \phi(x_N) \\
  \phi(x_2)^T \phi(x_1) & \phi(x_2)^T \phi(x_2) & \cdots & \phi(x_2)^T \phi(x_N) \\
  \vdots & \vdots & \ddots & \vdots \\
  \phi(x_N)^T \phi(x_1) & \phi(x_N)^T \phi(x_2) & \cdots & \phi(x_N)^T \phi(x_N)
\end{pmatrix}$$
Kernel functions

**Definition:** a (positive semidefinite) kernel function $k(\cdot, \cdot)$ is a bivariate function that satisfies the following properties. For any $x_m$ and $x_n$,

$$k(x_m, x_n) = k(x_n, x_m)$$

and

$$k(x_m, x_n) = \phi(x_m)^T \phi(x_n)$$

for some function $\phi(\cdot)$. 

Examples we have seen:

- $k(x_m, x_n) = (x_m^T x_n)^2$
- $k(x_m, x_n) = 2 - \sin(2\pi (x_m_1 - x_n_1)) (x_m_2 - x_n_2)$

Examples that are not kernels:

- $k(x_m, x_n) = \|x_m - x_n\|_2^2$ are not our desired kernel function as it cannot be written as inner products between two vectors.

Drs. Sha & Liu (feisha,yanliu.cs@usc.edu) CSCI567 Machine Learning (Fall 2014)
Kernel functions

**Definition:** A (positive semidefinite) kernel function $k(\cdot, \cdot)$ is a bivariate function that satisfies the following properties. For any $x_m$ and $x_n$,

$$k(x_m, x_n) = k(x_n, x_m) \text{ and } k(x_m, x_n) = \phi(x_m)^T \phi(x_n)$$

for some function $\phi(\cdot)$.

**Examples we have seen**

$$k(x_m, x_n) = (x_m^T x_n)^2$$

$$k(x_m, x_n) = 2 - \frac{\sin(2\pi(x_{m1} - x_{n1}))}{x_{m1} - x_{n1}} - \frac{\sin(2\pi(x_{m2} - x_{n2}))}{x_{m2} - x_{n2}}$$
Kernel functions

**Definition**: a (positive semidefinite) kernel function $k(\cdot, \cdot)$ is a bivariate function that satisfies the following properties. For any $x_m$ and $x_n$,

$$k(x_m, x_n) = k(x_n, x_m) \text{ and } k(x_m, x_n) = \phi(x_m)^T \phi(x_n)$$

for some function $\phi(\cdot)$.

**Examples we have seen**

$$k(x_m, x_n) = (x_m^T x_n)^2$$

$$k(x_m, x_n) = 2 - \frac{\sin(2\pi(x_{m1} - x_{n1}))}{x_{m1} - x_{n1}} - \frac{\sin(2\pi(x_{m2} - x_{n2}))}{x_{m2} - x_{n2}}$$

**Examples that are not kernels**

$$k(x_m, x_n) = \|x_m - x_n\|^2_2$$

are not our desired kernel function as it cannot be written as inner products between two vectors.
Mercer theorem (loosely), a bivariate function $k(\cdot, \cdot)$ is a positive semidefinite kernel function, if and only if, for any $N$ and any $x_1, x_2, \ldots, x_N$, the matrix

$$K = \begin{pmatrix}
  k(x_1, x_1) & k(x_1, x_2) & \cdots & k(x_1, x_N) \\
  k(x_2, x_1) & k(x_2, x_2) & \cdots & k(x_2, x_N) \\
  \vdots & \vdots & \ddots & \vdots \\
  k(x_N, x_1) & k(x_N, x_2) & \cdots & k(x_N, x_N)
\end{pmatrix}$$

is positive semidefinite. We also refer $k(\cdot, \cdot)$ as a positive semidefinite kernel.
Flashback: why using kernel functions?

without specifying $\phi(\cdot)$, the kernel matrix

$$K = \begin{pmatrix}
    k(x_1, x_1) & k(x_1, x_2) & \cdots & k(x_1, x_N) \\
    k(x_2, x_1) & k(x_2, x_2) & \cdots & k(x_2, x_N) \\
    \vdots & \vdots & \ddots & \vdots \\
    k(x_N, x_1) & k(x_N, x_2) & \cdots & k(x_N, x_N)
\end{pmatrix}$$

is exactly the same as

$$K = \Phi \Phi^T$$

$$= \begin{pmatrix}
    \phi(x_1)^T \phi(x_1) & \phi(x_1)^T \phi(x_2) & \cdots & \phi(x_1)^T \phi(x_N) \\
    \phi(x_2)^T \phi(x_1) & \phi(x_2)^T \phi(x_2) & \cdots & \phi(x_2)^T \phi(x_N) \\
    \vdots & \vdots & \ddots & \vdots \\
    \phi(x_N)^T \phi(x_1) & \phi(x_N)^T \phi(x_2) & \cdots & \phi(x_N)^T \phi(x_N)
\end{pmatrix}$$
Examples of kernel functions

**Polynomial kernel function with degree of** $d$

\[ k(x_m, x_n) = (x_m^T x_n + c)^d \]

for $c \geq 0$ and $d$ is a positive integer.
Examples of kernel functions

**Polynomial kernel function with degree of** $d$

$$k(x_m, x_n) = (x_m^T x_n + c)^d$$

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

**Gaussian kernel, RBF kernel, or Gaussian RBF kernel**

$$k(x_m, x_n) = e^{-\|x_m - x_n\|^2/2\sigma^2}$$
Examples of kernel functions

**Polynomial kernel function with degree of $d$**

$$k(x_m, x_n) = (x_m^T x_n + c)^d$$

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

**Gaussian kernel, RBF kernel, or Gaussian RBF kernel**

$$k(x_m, x_n) = e^{-\|x_m - x_n\|^2_2/2\sigma^2}$$

Most of those kernels have parameters to be tuned: $d$, $c$, $\sigma^2$, etc. They are hyper parameters and are often tuned on holdout data or with cross-validation.
Why $\| \mathbf{x}_m - \mathbf{x}_n \|_2^2$ is not a positive semidefinite kernel?

**Use the definition** of positive semidefinite kernel function. We choose $N = 2$, and compute the matrix

$$K = \begin{pmatrix}
0 & \| \mathbf{x}_1 - \mathbf{x}_2 \|_2^2 \\
\| \mathbf{x}_1 - \mathbf{x}_2 \|_2^2 & 0
\end{pmatrix}$$

This matrix cannot be positive semidefinite as it has both *negative* and positive eigenvalues (the sum of the diagonal elements is called the trace of a matrix, which equals to the sum of the matrix’s eigenvalues. In our case, the trace is zero.)
There are infinite numbers of kernels to use!

**Rules of composing kernels** (this is just a partial list)

- if $k(x_m, x_n)$ is a kernel, then $ck(x_m, x_n)$ is also if $c > 0$.
- if both $k_1(x_m, x_n)$ and $k_2(x_m, x_n)$ are kernels, then $\alpha k_1(x_m, x_n) + \beta k_2(x_m, x_n)$ are also if $\alpha, \beta \geq 0$
- if both $k_1(x_m, x_n)$ and $k_2(x_m, x_n)$ are kernels, then $k_1(x_m, x_n)k_2(x_m, x_n)$ are also.
- if $k(x_m, x_n)$ is a kernel, then $e^{k(x_m, x_n)}$ is also.
- \ldots

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.
Kernelization trick

Many learning methods depend on computing inner products between features — we have seen the example of regularized least squares. 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.

We will present one more to illustrate this “trick” by kernelizing nearest neighbor classifier.

When we talk about support vector machines next lecture, we will see the trick one more time.
Kernelized nearest neighbor classifier

In nearest neighbor classifier, the most important quantity to compute is the (squared) distance between two data points $x_m$ and $x_n$

$$d(x_m, x_n) = \|x_m - x_n\|^2 = x_m^T x_m + x_n^T x_n - 2x_m^T x_n$$
Kernelized nearest neighbor classifier

In nearest neighbor classifier, the most important quantity to compute is the (squared) distance between two data points \( x_m \) and \( x_n \)

\[
d(x_m, x_n) = \|x_m - x_n\|^2_2 = x_m^T x_m + x_n^T x_n - 2x_m^T x_n
\]

We replace all the inner products in the distance with a kernel function \( k(\cdot, \cdot) \), arriving at the kernelized distance

\[
d_{\text{KERNEL}}(x_m, x_n) = k(x_m, x_m) + k(x_n, x_n) - 2k(x_m, x_n)
\]
Kernelized nearest neighbor classifier

In nearest neighbor classifier, the most important quantity to compute is the (squared) distance between two data points $x_m$ and $x_n$

$$d(x_m, x_n) = \|x_m - x_n\|_2^2 = x_m^T x_m + x_n^T x_n - 2x_m^T x_n$$

We replace all the inner products in the distance with a kernel function $k(\cdot, \cdot)$, arriving at the kerneled distance

$$d^{\text{KERNEL}}(x_m, x_n) = k(x_m, x_m) + k(x_n, x_n) - 2k(x_m, x_n)$$

The distance is equivalent to compute the distance between $\phi(x_m)$ and $\phi(x_n)$

$$d^{\text{KERNEL}}(x_m, x_n) = d(\phi(x_m), \phi(x_n))$$

where the $\phi(\cdot)$ is the nonlinear mapping function implied by the kernel function.
Kernelized nearest neighbor classifier

In nearest neighbor classifier, the most important quantity to compute is the (squared) distance between two data points $x_m$ and $x_n$

$$d(x_m, x_n) = \| x_m - x_n \|^2_2 = x_m^T x_m + x_n^T x_n - 2 x_m^T x_n$$

We replace all the inner products in the distance with a kernel function $k(\cdot, \cdot)$, arriving at the kernalized distance

$$d^{\text{KERNEL}}(x_m, x_n) = k(x_m, x_m) + k(x_n, x_n) - 2k(x_m, x_n)$$

The distance is equivalent to compute the distance between $\phi(x_m)$ and $\phi(x_n)$

$$d^{\text{KERNEL}}(x_m, x_n) = d(\phi(x_m), \phi(x_n))$$

where the $\phi(\cdot)$ is the nonlinear mapping function implied by the kernel function. The nearest neighbor of a point $x$ is thus found with

$$\arg \min_n d^{\text{KERNEL}}(x, x_n)$$
Take-home exercise

You have seen examples of kernelizing

- linear regression
- nearest neighbor

But can you kernelize the following?

- Decision tree
- Logistic (or multinomial logistic) regression
Examples of kernel functions

Polynomial kernel function with degree of $d$

$$k(x_m, x_n) = (x_m^T x_n + c)^d$$

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

Gaussian kernel, RBF kernel, or Gaussian RBF kernel

$$k(x_m, x_n) = e^{-\|x_m - x_n\|_2^2 / 2\sigma^2}$$
Outline

1. Administration

2. Review of last lecture

3. Support vector machines
   - Hinge loss
   - Primal formulation of SVM
   - Basic Lagrange duality theory
   - Dual formulation of SVM
   - A very simple example

4. Geometric Understanding of SVM
Support vector machines

- One of the most commonly used machine learning algorithms.
- Convex optimization for classification and regression.
- It incorporates kernel tricks to define nonlinear decision boundaries or regression functions.
- It provides theoretical guarantees on generalization errors.
Hinge loss

**Definition** Assuming the label $y \in \{-1, 1\}$ and the decision rule is $h(x) = \text{SIGN}(f(x))$ with $f(x) = \mathbf{w}^T \phi(x) + b$, 

$$
\ell_{\text{Hinge}}(f(x), y) = \begin{cases} 
0 & \text{if } yf(x) \geq 1 \\
1 - yf(x) & \text{otherwise}
\end{cases}
$$

**Intuition**: penalize more if incorrectly classified (the left branch to the kink point)
Hinge loss

**Definition**  Assuming the label $y \in \{-1, 1\}$ and the decision rule is $h(x) = \text{SIGN}(f(x))$ with $f(x) = w^T \phi(x) + b$,

$$\ell_{\text{HINGE}}(f(x), y) = \begin{cases} 0 & \text{if } yf(x) \geq 1 \\ 1 - yf(x) & \text{otherwise} \end{cases}$$

**Intuition**: penalize more if incorrectly classified (the left branch to the kink point)

**Convenient shorthand**

$$\ell_{\text{HINGE}}(f(x), y) = \max(0, 1 - yf(x))$$
Properties

- Upper-bound (above) the 0/1 loss function (black line); optimizing it leads to reduced classification errors — namely, we use the hinge loss function as a *surrogate* to the true error function we care about.

- This function is not differentiable at the kink point!
Primal formulation of support vector machines (SVM)

Minimizing the total hinge loss on all the training data

\[
\min_{w,b} \sum_n \max(0, 1 - y_n [w^T \phi(x_n) + b]) + \frac{\lambda}{2} \|w\|^2
\]

which is analogous to regularized least square, which balances two terms (the loss and the regularizer).
Minimizing the total hinge loss on all the training data

$$\min_{w,b} \sum_n \max(0, 1 - y_n [w^T \phi(x_n) + b]) + \frac{\lambda}{2} \|w\|^2_2$$

which is analogous to regularized least square, which balances two terms (the loss and the regularizer). Conventionally, we rewrite the objective function as

$$\min_{w,b,C} \sum_n \max(0, 1 - y_n [w^T \phi(x_n) + b]) + \frac{1}{2} \|w\|^2_2$$

where $C$ is identified as $1/\lambda$. 

Drs. Sha & Liu  
{feisha,yanliu.cs}@usc.edu  
CSCI567 Machine Learning (Fall 2014)  
October 9, 2014 22 / 49
Primal formulation of support vector machines (SVM)

Minimizing the total hinge loss on all the training data

\[
\min_{\mathbf{w}, b} \sum_{n} \max(0, 1 - y_n [\mathbf{w}^T \phi(x_n) + b]) + \frac{\lambda}{2} \|\mathbf{w}\|^2
\]

which is analogous to regularized least square, which balances two terms (the loss and the regularizer). Conventionally, we rewrite the objective function as

\[
\min_{\mathbf{w}, b} C \sum_{n} \max(0, 1 - y_n [\mathbf{w}^T \phi(x_n) + b]) + \frac{1}{2} \|\mathbf{w}\|^2
\]

where \(C\) is identified as \(1/\lambda\). We further rewrite into another equivalent form

\[
\min_{\mathbf{w}, b, \{\xi_n\}} C \sum_{n} \xi_n + \frac{1}{2} \|\mathbf{w}\|^2
\]

s.t. \(\max(0, 1 - y_n [\mathbf{w}^T \phi(x_n) + b]) = \xi_n, \forall n\)
Primal formulation of SVM

**Primal formulation**

\[
\min_{w,b,\{\xi_n\}} \quad C \sum_n \xi_n + \frac{1}{2} \|w\|_2^2 \\
\text{s.t.} \quad 1 - y_n [w^T \phi(x_n) + b] \leq \xi_n, \quad \forall \ n \\
\xi_n \geq 0, \quad \forall \ n
\]

where all \( \xi_n \) are called \textit{slack variables}. 

Remarks

This is a convex quadratic programming: the objective function is quadratic in \( w \) and the constraints are linear (inequality) constraints in \( w \) and \( \xi_n \).

Given \( \phi(\cdot) \), we can solve the optimization problem efficiently as it is convex, for example, using Matlab's \texttt{quadprog()} function.

However, there are efficient algorithms for solving this problem, taking advantage of the special structures of the objective function and the constraints. (We will not discuss them. Most existing SVM implementation/packages implement such efficient algorithms.)
Primal formulation of SVM

Primal formulation

\[
\min_{w,b,\{\xi_n\}} \quad C \sum_n \xi_n + \frac{1}{2} \|w\|_2^2
\]

s.t. \quad 1 - y_n [w^T \phi(x_n) + b] \leq \xi_n, \quad \forall \ n

\xi_n \geq 0, \quad \forall \ n

where all \( \xi_n \) are called \textit{slack variables}.

Remarks

- This is a convex quadratic programming: the objective function is quadratic in \( w \) and the constraints are linear (inequality) constraints in \( w \) and \( \xi_n \).
Primal formulation of SVM

Primal formulation

$$\min_{w, b, \{\xi_n\}} \quad C \sum_n \xi_n + \frac{1}{2} \|w\|_2^2$$

subject to

$$1 - y_n [w^T \phi(x_n) + b] \leq \xi_n, \quad \forall \ n$$

$$\xi_n \geq 0, \quad \forall \ n$$

where all $\xi_n$ are called slack variables.

Remarks

- This is a convex quadratic programming: the objective function is quadratic in $w$ and the constraints are linear (inequality) constraints in $w$ and $\xi_n$.
- Given $\phi(\cdot)$, we can solve the optimization problem efficiently as it is convex, for example, using Matlab's `quadprog()` function.
Primal formulation of SVM

Primal formulation

\[
\min_{w, b, \{\xi_n\}} \quad C \sum_n \xi_n + \frac{1}{2} \|w\|_2^2 \\
\text{s.t.} \quad 1 - y_n[w^T \phi(x_n) + b] \leq \xi_n, \quad \forall \ n \\
\xi_n \geq 0, \quad \forall \ n
\]

where all \( \xi_n \) are called \textit{slack variables}.

Remarks

- This is a convex quadratic programming: the objective function is quadratic in \( w \) and the constraints are linear (inequality) constraints in \( w \) and \( \xi_n \).
- Given \( \phi(\cdot) \), we can solve the optimization problem efficiently as it is convex, for example, using Matlab's \texttt{quadprog()} function.
- However, there are efficient algorithms for solving this problem, taking advantage of the special structures of the objective function and the constraints. (We will not discuss them. Most existing SVM implementation/packages implement such efficient algorithms.)
Key concepts you should know

- What do “primal” and “dual” mean?
- How SVM exploits dual formulation, thus results in using kernel functions for nonlinear classification
- What do support vectors mean?

Our roadmap

- We will tell you what dual looks like
- We will show you how it is derived
Dual formulation

Dual is also a convex quadratic programming

$$\begin{align*}
\max_{\alpha} & \quad \sum_n \alpha_n - \frac{1}{2} \sum_{m,n} y_m y_n \alpha_m \alpha_n \phi(x_m)^T \phi(x_n) \\
\text{s.t.} & \quad 0 \leq \alpha_n \leq C, \quad \forall \ n \\
& \quad \sum_n \alpha_n y_n = 0
\end{align*}$$
Dual formulation

Dual is also a convex quadratic programming

\[
\begin{align*}
\max_{\alpha} & \quad \sum_n \alpha_n - \frac{1}{2} \sum_{m,n} y_m y_n \alpha_m \alpha_n \phi(x_m)^T \phi(x_n) \\
\text{s.t.} & \quad 0 \leq \alpha_n \leq C, \quad \forall \ n \\
& \quad \sum_n \alpha_n y_n = 0
\end{align*}
\]

Remarks

- The optimization is convex as the objective function is concave.
  (Take-home exercise: please verify)
Support vector machines

Dual formulation

Dual is also a convex quadratic programming

\[
\max_{\alpha} \sum_n \alpha_n - \frac{1}{2} \sum_{m,n} y_m y_n \alpha_m \alpha_n \phi(x_m)^T \phi(x_n)
\]

s.t. \(0 \leq \alpha_n \leq C, \ \forall \ n\)

\[
\sum_n \alpha_n y_n = 0
\]

Remarks

- The optimization is convex as the objective function is concave.
  \((Take\text{-}home\ exercise:\ please\ verify)\)
- **There are** \(N\) **dual variable** \(\alpha_n\), **one for each constraint**

  \(1 - y_n [\mathbf{w}^T \phi(x_n) + b] \leq \xi_n\) **in the primal formulation.**
Kernelized SVM

We replace the inner products $\phi(x_m)^T \phi(x_n)$ with a kernel function

$$\max_{\alpha} \sum_n \alpha_n - \frac{1}{2} \sum_{m,n} y_m y_n \alpha_m \alpha_n k(x_m, x_n)$$

s.t. $0 \leq \alpha_n \leq C, \forall n$

$$\sum_n \alpha_n y_n = 0$$

as in kernelized linear regression and kernelized nearest neighbor. We only need to define a kernel function and we will automatically get (nonlinearly) mapped features and the support vector machine constructed with those features.
Recovering solution to the primal formulation

Weights

\[ w = \sum_{n} y_n \alpha_n \phi(x_n) \leftarrow \text{Linear combination of the input features!} \]
Recovering solution to the primal formulation

**Weights**

\[ w = \sum_{n} y_n \alpha_n \phi(x_n) \quad \text{← Linear combination of the input features!} \]

**b**

\[ b = [y_n - w^T \phi(x_n)] = [y_n - \sum_{m} y_m \alpha_m k(x_m, x_n)], \quad \text{for any } C > \alpha_n > 0 \]
Recovering solution to the primal formulation

Weights

\[ w = \sum_{n} y_n \alpha_n \phi(x_n) \quad \text{← Linear combination of the input features!} \]

b

\[ b = [y_n - w^T \phi(x_n)] = [y_n - \sum_{m} y_m \alpha_m k(x_m, x_n)], \quad \text{for any } C > \alpha_n > 0 \]

Making prediction on a test point \( x \)

\[ h(x) = \text{SIGN}(w^T \phi(x) + b) = \text{SIGN}\left(\sum_{n} y_n \alpha_n k(x_n, x) + b\right) \]

Again, to make prediction, it suffices to know the kernel function.
Derivation of the dual

We will derive the dual formulation as the process will reveal some interesting and important properties of SVM. Particularly, why is it called “support vector”?

Recipe

- Formulate a Lagrangian function that incorporates the constraints, thru introducing dual variables
- Minimize the Lagrangian function to solve the primal variables
- Put the primal variables into the Lagrangian and express in terms of dual variables
- Maximize the Lagrangian with respect to dual variables
- Recover the solution (for the primal variables) from the dual variables
A simple example

Consider the example of convex quadratic programming

\[
\begin{align*}
\text{min} & \quad \frac{1}{2}x^2 \\
\text{s.t.} & \quad -x \leq 0 \\
& \quad 2x - 3 \leq 0
\end{align*}
\]

The Lagrangian is (note that we do not have equality constraints)

\[
L(x, \mu) = \frac{1}{2}x^2 + \mu_1 \times (-x) + \mu_2 \times (2x - 3) = \frac{1}{2}x^2 + (2\mu_2 - \mu_1)x - 3\mu_2
\]

under the constraint that \( \mu_1 \geq 0 \) and \( \mu_2 \geq 0 \).
A simple example

Consider the example of convex quadratic programming

\[
\begin{align*}
\min & \quad \frac{1}{2} x^2 \\
\text{s.t.} & \quad -x \leq 0 \\
& \quad 2x - 3 \leq 0
\end{align*}
\]

The Lagrangian is (note that we do not have equality constraints)

\[
L(x, \mu) = \frac{1}{2} x^2 + \mu_1 \times (-x) + \mu_2 \times (2x - 3) = \frac{1}{2} x^2 + (2\mu_2 - \mu_1)x - 3\mu_2
\]

under the constraint that \( \mu_1 \geq 0 \) and \( \mu_2 \geq 0 \). Its dual problem is

\[
\max_{\mu_1 \geq 0, \mu_2 \geq 0} \min_x L(x, \mu) = \max_{\mu_1 \geq 0, \mu_2 \geq 0} \min_x \frac{1}{2} x^2 + (2\mu_2 - \mu_1)x - 3\mu_2
\]
Example (cont’d)

We solve the \( \min_x L(x, \mu) \) first now it is unconstrained. The optimal \( x \) is attained by

\[
\frac{\partial}{\partial x} \left( \frac{1}{2} x^2 + (2\mu_2 - \mu_1)x - 3\mu_2 \right) = 0 \rightarrow x = -(2\mu_2 - \mu_1)
\]
We solve the $\min_x L(x, \mu)$ first now it is unconstrained. The optimal $x$ is attained by

$$\frac{\partial (\frac{1}{2} x^2 + (2\mu_2 - \mu_1)x - 3\mu_2)}{\partial x} = 0 \rightarrow x = -(2\mu_2 - \mu_1)$$

This gives us the dual objective function, by substituting the solution into the objective function,

$$g(\mu) = \min_x \frac{1}{2} x^2 + (2\mu_2 - \mu_1)x - 3\mu_2 = -\frac{1}{2} (2\mu_2 - \mu_1)^2 - 3\mu_2$$
Example (cont’d)

We solve the $\min_x L(x, \mu)$ first now it is unconstrained. The optimal $x$ is attained by

$$\frac{\partial(\frac{1}{2}x^2 + (2\mu_2 - \mu_1)x - 3\mu_2)}{\partial x} = 0 \rightarrow x = -(2\mu_2 - \mu_1)$$

This gives us the dual objective function, by substituting the solution into the objective function,

$$g(\mu) = \min_x \frac{1}{2}x^2 + (2\mu_2 - \mu_1)x - 3\mu_2 = -\frac{1}{2}(2\mu_2 - \mu_1)^2 - 3\mu_2$$

We get our dual problem as

$$\max_{\mu_1 \geq 0, \mu_2 \geq 0} -\frac{1}{2}(2\mu_2 - \mu_1)^2 - 3\mu_2$$

We will solve the dual next.
Solving the dual

Note that,

\[ g(\mu) = -\frac{1}{2}(2\mu_2 - \mu_1)^2 - 3\mu_2 \leq 0 \]

for all \( \mu_1 \geq 0, \mu_2 \geq 0 \). Thus, to maximize the function, the optimal solution is

\[ \mu_1^* = 0, \quad \mu_2^* = 0 \]

This brings us back the optimal solution of \( x \)

\[ x^* = -(2\mu_2^* - \mu_1^*) = 0 \]

Namely, we have arrived at the same solution as the one we guessed from the primal formulation
Deriving the dual for SVM

**Lagrangian**

\[
L(w, \{\xi_n\}, \{\alpha_n\}, \{\lambda_n\}) = C \sum_n \xi_n + \frac{1}{2} \|w\|^2 - \sum_n \lambda_n \xi_n
\]
\[
+ \sum_n \alpha_n \{1 - y_n [w^T \phi(x_n) + b] - \xi_n\}
\]

under the constraint that \(\alpha_n \geq 0\) and \(\lambda_n \geq 0\).
Minimizing the Lagrangian

**Taking derivatives with respect to the primal variables**

\[
\frac{\partial L}{\partial w} = w - \sum_n y_n \alpha_n \phi(x_n) = 0
\]
Minimizing the Lagrangian

Taking derivatives with respect to the primal variables

\[ \frac{\partial L}{\partial w} = w - \sum_n y_n \alpha_n \phi(x_n) = 0 \]

\[ \frac{\partial L}{\partial b} = \sum_n \alpha_n y_n = 0 \]
Minimizing the Lagrangian

**Taking derivatives with respect to the primal variables**

\[
\frac{\partial L}{\partial w} = w - \sum_n y_n \alpha_n \phi(x_n) = 0
\]

\[
\frac{\partial L}{\partial b} = \sum_n \alpha_n y_n = 0
\]

\[
\frac{\partial L}{\partial \xi_n} = C - \lambda_n - \alpha_n = 0
\]
Minimizing the Lagrangian

**Taking derivatives with respect to the primal variables**

\[
\frac{\partial L}{\partial w} = w - \sum_n y_n \alpha_n \phi(x_n) = 0
\]

\[
\frac{\partial L}{\partial b} = \sum_n \alpha_n y_n = 0
\]

\[
\frac{\partial L}{\partial \xi_n} = C - \lambda_n - \alpha_n = 0
\]

This gives rise to equations linking the primal variables and the dual variables as well as new constraints on the dual variables:

\[
w = \sum_n y_n \alpha_n \phi(x_n)
\]

\[
\sum_n \alpha_n y_n = 0
\]

\[
C - \lambda_n - \alpha_n = 0
\]
Rewrite the Lagrange in terms of dual variables

Substitute the solution to the primal back into the Lagrangian

\[ g(\{\alpha_n\},\{\lambda_n\}) = L(w, \{\xi_n\}, \{\alpha_n\}, \{\lambda_n\}) \]
Rewrite the Lagrange in terms of dual variables

Substitute the solution to the primal back into the Lagrangian

\[ g(\{\alpha_n\}, \{\lambda_n\}) = L(w, \{\xi_n\}, \{\alpha_n\}, \{\lambda_n\}) \]

\[ = \sum_n (C - \alpha_n - \lambda_n) \xi_n + \frac{1}{2} \sum_n y_n \alpha_n \phi(x_n) \|^2 + \sum_n \alpha_n \]

\[ + \left( \sum_n \alpha_n y_n \right) b - \sum_n \alpha_n y_n \left( \sum_m y_m \alpha_m \phi(x_m) \right)^T \phi(x_n) \]
Rewrite the Lagrange in terms of dual variables

Substitute the solution to the primal back into the Lagrangian

\[ g(\{\alpha_n\}, \{\lambda_n\}) = L(w, \{\xi_n\}, \{\alpha_n\}, \{\lambda_n\}) = \sum_n (C - \alpha_n - \lambda_n) \xi_n + \frac{1}{2} \left\| \sum_n y_n \alpha_n \phi(x_n) \right\|_2^2 + \sum_n \alpha_n \]

\[ + \left( \sum_n \alpha_n y_n \right) b - \sum_n \alpha_n y_n \left( \sum_m y_m \alpha_m \phi(x_m) \right)^T \phi(x_n) \]

\[ = \sum_n \alpha_n + \frac{1}{2} \left\| \sum_n y_n \alpha_n \phi(x_n) \right\|_2^2 - \sum_{m,n} \alpha_n \alpha_m y_m y_n \phi(x_m)^T \phi(x_n) \]

Several terms vanish because of the constraints \[ \sum_n \alpha_n y_n = 0 \] and \[ C - \lambda_n - \alpha_n = 0. \]
Rewrite the Lagrange in terms of dual variables

Substitute the solution to the primal back into the Lagrangian

\[
g(\alpha_n, \lambda_n) = L(w, \xi_n, \alpha_n, \lambda_n) = \sum_n (C - \alpha_n - \lambda_n)\xi_n + \frac{1}{2} \left\| \sum_n y_n \alpha_n \phi(x_n) \right\|^2 + \sum_n \alpha_n \\
+ \left( \sum_n \alpha_n y_n \right) b - \sum_n \alpha_n y_n \left( \sum_m y_m \alpha_m \phi(x_m) \right)^T \phi(x_n) \\
= \sum_n \alpha_n + \frac{1}{2} \left\| \sum_n y_n \alpha_n \phi(x_n) \right\|^2 - \sum_{m,n} \alpha_n \alpha_m y_m y_n \phi(x_m)^T \phi(x_n) \\
= \sum_n \alpha_n - \frac{1}{2} \sum_{m,n} \alpha_n \alpha_m y_m y_n \phi(x_m)^T \phi(x_n)
\]

Several terms vanish because of the constraints \( \sum_n \alpha_n y_n = 0 \) and \( C - \lambda_n - \alpha_n = 0 \).
The dual problem

Maximizing the dual under the constraints

\[
\max_\alpha \quad g(\{\alpha_n\}, \{\lambda_n\}) = \sum_n \alpha_n - \frac{1}{2} \sum_{m,n} y_m y_n \alpha_m \alpha_n k(x_m, x_n)
\]

s.t.
\[
\alpha_n \geq 0, \quad \forall \ n
\]
\[
\sum_n \alpha_n y_n = 0
\]
\[
C - \lambda_n - \alpha_n = 0, \quad \forall \ n
\]
\[
\lambda_n \geq 0, \quad \forall \ n
\]
The dual problem

Maximizing the dual under the constraints

\[
\max_\alpha g(\{\alpha_n\}, \{\lambda_n\}) = \sum_n \alpha_n - \frac{1}{2} \sum_{m,n} y_my_n\alpha_m\alpha_n k(x_m, x_n)
\]

s.t. \( \alpha_n \geq 0, \quad \forall \ n \)

\[
\sum_n \alpha_n y_n = 0
\]

\( C - \lambda_n - \alpha_n = 0, \quad \forall \ n \)

\( \lambda_n \geq 0, \quad \forall \ n \)

We can simplify as the objective function does not depend on \( \lambda_n \), thus we can convert the equality constraint involving \( \lambda_n \) with an inequality constraint on \( \alpha_n \leq C \):

\[\alpha_n \leq C \iff \lambda_n = C - \alpha_n \geq 0 \iff C - \lambda_n - \alpha_n = 0, \lambda_n \geq 0\]
Final form

$$\max_{\alpha} \sum_n \alpha_n - \frac{1}{2} \sum_{m,n} y_m y_n \alpha_m \alpha_n \phi(x_m)^T \phi(x_n)$$

s.t. \( 0 \leq \alpha_n \leq C, \ \forall \ n \)

$$\sum_n \alpha_n y_n = 0$$
Recover the solution

The primal variable $\mathbf{w}$ is identified as

$$
\mathbf{w} = \sum_{n} \alpha_n y_n \phi(x_n)
$$
Recover the solution

The primal variable $w$ is identified as

$$w = \sum_{n} \alpha_n y_n \phi(x_n)$$

To identify $b$, we need something else.
Complementary slackness and support vectors

At the optimal solution to both primal and dual, the following must be satisfied for every inequality constraint (these are called KKT conditions)

\[ \lambda_n \xi_n = 0 \]
\[ \alpha_n \{1 - \xi_n - y_n [w^T \phi(x_n) + b]\} = 0 \]
Complementary slackness and support vectors

At the optimal solution to both primal and dual, the following must be satisfied for every inequality constraint (these are called KKT conditions)

\[ \lambda_n \xi_n = 0 \]
\[ \alpha_n \{1 - \xi_n - y_n[w^T \phi(x_n) + b]\} = 0 \]

From the first condition, if \( \alpha_n < C \), then

\[ \lambda_n = C - \alpha_n > 0 \rightarrow \xi_n = 0 \]

Thus, in conjunction with the second condition, we know that, if \( C > \alpha_n > 0 \), then

\[ 1 - y_n[w^T \phi(x_n) + b] = 0 \rightarrow b = y_n - w^T \phi(x_n) \]

as \( y_n \in \{-1, 1\} \).

For those \( n \) whose \( \alpha_n > 0 \), we call such training samples as “support vectors”. (We will discuss their geometric interpretation later).
Outline

1. Administration
2. Review of last lecture
3. Support vector machines
4. Geometric Understanding of SVM
Intuition: where to put the decision boundary?

Consider the binary classification in the following figure. We have assumed, for convenience, that the training dataset is separable — there is a decision boundary that separates the two classes perfectly.

There are infinite many ways of putting the decision boundary \( \mathcal{H} : \mathbf{w}^T \phi(\mathbf{x}) + b = 0 \)! Our intuition is, however, to put the decision boundary to be in the middle of the two classes as much as possible. In other words, we want the decision boundary is to be far to every point as much as possible as long as the decision boundary classifies every point correctly.
**Distances**

The distance from a point $\phi(x)$ to the decision boundary is

$$d_\mathcal{H}(\phi(x)) = \frac{|w^T \phi(x) + b|}{\|w\|_2}$$

(We have derived the above in the recitation/quiz0. Please re-verify it as a take-home exercise). We can remove the absolute $|\cdot|$ by exploiting the fact that the decision boundary classifies every point in the training dataset correctly. Namely, $(w^T \phi(x) + b)$ and $x$'s label $y$ are of the same sign. The distance is now,

$$d_\mathcal{H}(\phi(x)) = \frac{y[w^T \phi(x) + b]}{\|w\|_2}$$
Maximizing margin

**Margin** The margin is defined as the smallest distance from all the training points

\[
\text{MARGIN} = \min_n \frac{y_n[w^T \phi(x_n) + b]}{\|w\|_2}
\]
Maximizing margin

**Margin** The margin is defined as the smallest distance from all the training points

$$
\text{MARGIN} = \min_n \frac{y_n[w^T \phi(x_n) + b]}{\|w\|_2}
$$

Since we are interested in finding a $w$ to put all points as distant as possible from the decision boundary, we maximize the margin

$$
\max_w \min_n \frac{y_n[w^T \phi(x_n) + b]}{\|w\|} = \max_w \frac{1}{\|w\|_2} \min_n y_n[w^T \phi(x_n) + b]
$$

$H : w^T \phi(x) + b = 0$

$|w^T \phi(x) + b| / \|w\|_2$
Rescaled margin

Since the margin does not change if we scale \((w, b)\) by a constant factor \(c\) (as \(w^T \phi(x) + b = 0\) and \((cw)^T \phi(x) + (cb) = 0\) are the same decision boundary), we fix the scale by forcing

\[
\min_n y_n [w^T \phi(x_n) + b] = 1
\]
Rescaled margin

Since the **margin** does not change if we scale \((w, b)\) by a constant factor \(c\) (as \(w^T \phi(x) + b = 0\) and \((cw)^T \phi(x) + (cb) = 0\) are the same decision boundary), we fix the scale by forcing

\[
\min_n y_n [w^T \phi(x_n) + b] = 1
\]

In this case, our margin becomes

\[
\text{MARGIN} = \frac{1}{\|w\|_2}
\]

precisely, the closest point to the decision boundary has a distance of that.
Primal formulation

Combining everything we have, for a separable training dataset, we aim to

$$\max_w \frac{1}{\|w\|_2^2} \quad \text{such that} \quad y_n[w^T \phi(x_n) + b] \geq 1, \quad \forall \ n$$

This is equivalent to

$$\min_w \frac{1}{2}\|w\|_2^2$$

$$\text{s.t.} \quad y_n[w^T \phi(x_n) + b] \geq 1, \quad \forall \ n$$

This starts to look like our first formulation for SVMs. For this geometric intuition, SVM is called *max margin* (or large margin) classifier. The constraints are called *large margin constraints*. 
SVM for non-separable data

Suppose there are training data points that cannot be classified correctly no matter how we choose $w$. For those data points,

$$y_n[w^T \phi(x_n) + b] \leq 0$$

for any $w$. Thus, the previous constraint

$$y_n[w^T \phi(x_n) + b] \geq 1, \quad \forall \ n$$

is no longer feasible.
SVM for non-separable data

Suppose there are training data points that cannot be classified correctly no matter how we choose $w$. For those data points,

$$y_n[w^T \phi(x_n) + b] \leq 0$$

for any $w$. Thus, the previous constraint

$$y_n[w^T \phi(x_n) + b] \geq 1, \ \forall \ n$$

is no longer feasible. To deal with this issue, we introduce slack variables $\xi_n$ to help

$$y_n[w^T \phi(x_n) + b] \geq 1 - \xi_n, \ \forall \ n$$

where we also require $\xi_n \geq 0$. Note that, even for “hard” points that cannot be classified correctly, the slack variable will be able to make them satisfy the above constraint (we can keep increasing $\xi_n$ until the above inequality is met.)
We obviously do not want $\xi_n$ goes to infinity, so we balance their sizes by penalizing them toward zero as much as possible

$$\min_w \frac{1}{2} \|w\|_2^2 + C \sum_n \xi_n$$

s.t. $y_n [w^T \phi(x_n) + b] \geq 1 - \xi_n, \ \forall \ n$

$\xi_n \geq 0, \ \forall \ n$

where $C$ is our tradeoff (hyper)parameter. This is precisely the primal formulation we first got for SVM.
Meaning of “support vectors” in SVMs

**Complementary slackness** At optimum, we have to have

\[ \alpha_n \{1 - \xi_n - y_n[w^T \phi(x_n) + b]\} = 0, \quad \forall \ n \]

That means, for some \( n \), \( \alpha_n = 0 \). Additionally, our optimal solution is given by

\[ w = \sum_n \alpha_n y_n \phi(x_n) = \sum_{n: \alpha_n > 0} \alpha_n y_n \phi(x_n) \]

In words, our solution is only determined by those training samples whose corresponding \( \alpha_n \) is strictly positive. Those samples are called “support vectors”.

Non-support vectors whose \( \alpha_n = 0 \) can be removed by the training dataset — this removal will not affect the optimal solution (i.e., after the removal, if we construct another SVM classifier on the reduced dataset, the optimal solution is the same as the one on the original dataset.)
Who are support vectors?

**Case analysis** Since, we have

\[ 1 - \xi_n - y_n[w^T \phi(x_n) + b] = 0 \]

We have

- \( \xi_n = 0 \). This implies \( y_n[w^T \phi(x_n) + b] = 1 \). They are on points that are \( 1/|w|_2^2 \) away from the decision boundary.
- \( \xi_n < 1 \). These are points that can be classified correctly but do not satisfy the large margin constraint – they have smaller distances to the decision boundary.
- \( \xi_n > 1 \). These are points that are misclassified.
Visualization of how training data points are categorized

\[ \mathcal{H} : \mathbf{w}^T \phi(x) + b = 0 \]
\[ \mathbf{w}^T \phi(x) + b = 1 \]
\[ \mathbf{w}^T \phi(x) + b = -1 \]
\[ \xi_n = 0 \]
\[ \xi_n < 1 \]
\[ \xi_n > 1 \]

Support vectors are those being circled with the orange line.