CSCI567 Machine Learning (Fall 2014)

Drs. Sha & Liu

{feisha,yanliu.cs}@usc.edu

October 7, 2014
Outline

1. Administration
2. Review of last lecture
3. Kernel methods
4. Support vector machines
**Proposed Time for Quiz #1: 6pm on Oct 21 (Tue)**  
Reply to my email latest Wed midnight if the time slot does NOT work for you.
Outline

1. Administration
2. Review of last lecture
   - Bias/variance tradeoff
3. Kernel methods
4. Support vector machines
Bias/variance tradeoff

**Error decomposes into 3 terms**

\[ E_D R[h_D(x)] = \text{VARIANCE} + \text{BIAS}^2 + \text{NOISE} \]

where the first and the second term are inherently in conflict in terms of choosing what kind of \( h(x) \) we should use (unless we have an infinite amount of data).
Bias/variance tradeoff

Error decomposes into 3 terms

\[ \mathbb{E}_D R[h_D(x)] = \text{VARIANCE} + \text{BIAS}^2 + \text{NOISE} \]

where the first and the second term are inherently in conflict in terms of choosing what kind of \( h(x) \) we should use (unless we have an infinite amount of data).

If we can compute all terms analytically, they will look like this
Example: why regularized linear regression could be helpful?

**Model**

\[ h(x) = w^T x \]

**Consider the best possible** \( h^*(x) \)

\[ w^* = \arg \min_w \int_x \left[ \mathbb{E}_y[y] - w^T x \right]^2 p(x) dx \]

Note that this linear model assumes the knowledge of joint distribution, thus, not achievable. Intuitively, it is the *best* linear model that can predict the data most accurately.
More refined decomposition of the bias

\[
\int_x [\mathbb{E}_D h_D(x) - \mathbb{E}_y [y]]^2 p(x) dx = \int_x [h^*(x) - \mathbb{E}_y [y]]^2 p(x) dx \\
+ \int_x [\mathbb{E}_D h_D(x) - h^*(x)]^2 p(x) dx
\]

- Model bias: the price we pay for choosing linear functions to model data
- Estimation bias: the difference between the optimal model and the estimated model

Normally, the estimation bias is zero if we do not regularize.
Bias/variance tradeoff for regularized linear regression

We can only adjust estimation bias

\[ \int_{\mathbf{x}} \left[ \mathbb{E}_{D} h_{D}(\mathbf{x}; \lambda) - h^{*}(\mathbf{x}) \right]^{2} p(\mathbf{x}) d\mathbf{x} \]

where \( h(\mathbf{x}; \lambda) \) is the estimated model with regularized linear regression (parameterized with \( \lambda \)).

This term will not be zero anymore!

Thus, bias goes up

But, as long as this is balanced with a decrease in variance, we are willing to do so.
Visualizing the tradeoff
3 Kernel methods
   - Motivation
   - Kernel matrix and kernel functions
   - Kernelized machine learning methods

4 Support vector machines
Motivation

How to choose nonlinear basis function for regression?

$$\mathbf{w}^\top \phi(\mathbf{x})$$

where $\phi(\cdot)$ maps the original feature vector $\mathbf{x}$ to a $M$-dimensional new feature vector. In the following, we will show that we can sidestep the issue of choosing which $\phi(\cdot)$ to use — instead, we will choose equivalently a kernel function.

Regularized least square

$$J(\mathbf{w}) = \frac{1}{2} \sum_{n} (y_n - \mathbf{w}^\top \phi(\mathbf{x}_n))^2 + \frac{\lambda}{2} ||\mathbf{w}||^2$$
Motivation

How to choose nonlinear basis function for regression?

$$w^T \phi(x)$$

where $\phi(\cdot)$ maps the original feature vector $x$ to a $M$-dimensional new feature vector. In the following, we will show that we can sidestep the issue of choosing which $\phi(\cdot)$ to use — instead, we will choose equivalently a kernel function.

Regularized least square

$$J(w) = \frac{1}{2} \sum_n (y_n - w^T \phi(x_n))^2 + \frac{\lambda}{2} \|w\|^2_2$$

Its solution $w^{MAP}$ is given by

$$\frac{\partial J(w)}{\partial w} = \sum_n (y_n - w^T \phi(x_n))(-\phi(x_n)) + \lambda w = 0$$
MAP Solution

**The optimal parameter vector is a linear combination of features**

\[ \mathbf{w}^{\text{MAP}} = \sum_n \frac{1}{\lambda} (y_n - \mathbf{w}^T \phi(x_n)) \phi(x_n) = \sum_n \alpha_n \phi(x_n) = \mathbf{\Phi}^T \alpha \]

where we have designated \( \frac{1}{\lambda} (y_n - \mathbf{w}^T \phi(x_n)) \) as \( \alpha_n \). And the matrix \( \mathbf{\Phi} \) is the *design matrix* made of *transformed* features. Its transpose is made of column vectors and is given by

\[ \mathbf{\Phi}^T = (\phi(x_1) \phi(x_2) \cdots \phi(x_N)) \in \mathbb{R}^{M \times N} \]

where \( M \) is the dimensionality of \( \phi(x) \).

Of course, we do not know what \( \alpha \) (the vector of all \( \alpha_n \)) corresponds to \( \mathbf{w}^{\text{MAP}} \)!
Dual formulation

We substitute $w^{\text{MAP}} = \Phi^T \alpha$ into $J(w)$, and obtain the following function of $\alpha$

$$ J(\alpha) = \frac{1}{2} \alpha^T \Phi \Phi^T \Phi \Phi^T \alpha - (\Phi \Phi^T y)^T \alpha + \frac{\lambda}{2} \alpha^T \Phi \Phi^T \alpha $$

Before we show how $J(\alpha)$ is derived, we make an important observation. We see repeated structures $\Phi \Phi^T$, to which we refer as Gram matrix or kernel matrix

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

$$ = \begin{pmatrix}
\phi(x_1)^T \phi(x_1) & \phi(x_1)^T \phi(x_2) & \cdots & \phi(x_1)^T \phi(x_N) \\
\phi(x_2)^T \phi(x_1) & \phi(x_2)^T \phi(x_2) & \cdots & \phi(x_2)^T \phi(x_N) \\
\vdots & \vdots & \ddots & \vdots \\
\phi(x_N)^T \phi(x_1) & \phi(x_N)^T \phi(x_2) & \cdots & \phi(x_N)^T \phi(x_N)
\end{pmatrix} \in \mathbb{R}^{N \times N} $$
Properties of the matrix $K$

- Symmetric

\[ K_{mn} = \phi(x_m)^T \phi(x_n) = \phi(x_n)^T \phi(x_m) = K_{nm} \]

- Positive semidefinite: for any vector $a$

\[ a^T K a = (\Phi^T a)^T (\Phi^T a) \geq 0 \]

- Not the same as the second-moment (covariance) matrix $C = \Phi^T \Phi$
  
  1. $C$ has a size of $D \times D$ while $K$ is $N \times N$.
  2. When $N \leq D$, using $K$ is more computationally advantageous.
The derivation of $J(\alpha)$

\[
J(w) = \frac{1}{2} \sum_n (y - w^T \phi(x_n))^2 + \frac{\lambda}{2} \|w\|^2_2
\]

\[
= \frac{1}{2} \|y - \Phi w\|^2_2 + \frac{\lambda}{2} \|w\|^2_2
\]

\[
= \frac{1}{2} \|y - \Phi \Phi^T \alpha\|^2_2 + \frac{\lambda}{2} \|\Phi^T \alpha\|^2_2
\]

\[
= \frac{1}{2} \|y - K \alpha\|^2_2 + \frac{\lambda}{2} \alpha^T \Phi \Phi^T \alpha
\]

\[
= \frac{1}{2} \alpha^T K^TK \alpha - y^T K \alpha + \frac{\lambda}{2} \alpha^T K \alpha
\]

\[
= \frac{1}{2} \alpha^T K^2 \alpha - (Ky)^T \alpha + \frac{\lambda}{2} \alpha^T K \alpha = J(\alpha)
\]

where we have used the property that $K$ is symmetric.
Optimal $\alpha$

\[
\frac{\partial J(\alpha)}{\partial \alpha} = K^2 \alpha - Ky + \lambda K \alpha = 0
\]

which leads to (assuming that $K$ is invertible)

\[
\alpha = (K + \lambda I)^{-1} y
\]
Optimal $\alpha$

$$
\frac{\partial J(\alpha)}{\partial \alpha} = K^2\alpha - Ky + \lambda K \alpha = 0
$$

which leads to (assuming that $K$ is invertible)

$$
\alpha = (K + \lambda I)^{-1} y
$$

From this, we can compute the parameter vector

$$
w^{MAP} = \Phi^T(K + \lambda I)^{-1} y
$$

Note that we only need to know $K$ in order to compute $\alpha$ — the exact form of $\phi(\cdot)$ is not essential — as long as we know how to get inner products $\phi(x_m)^T \phi(x_n)$. That observation will give rise to the use of kernel function.
Computing prediction needs only inner products too!

Suppose we need to make a prediction on a new data point $x$, we thus compute

$$w^T \phi(x) = y^T(K + \lambda I)^{-1}\Phi x$$
Computing prediction needs only inner products too!

Suppose we need to make a prediction on a new data point $x$, we thus compute

$$w^T \phi(x) = y^T (K + \lambda I)^{-1} \Phi x$$

$$= y^T (K + \lambda I)^{-1} \begin{pmatrix} \phi(x_1)^T \phi(x) \\ \phi(x_2)^T \phi(x) \\ \vdots \\ \phi(x_N)^T \phi(x) \end{pmatrix} = y^T (K + \lambda I)^{-1} k_x$$

where we have used the property that $(K + \lambda I)^{-1}$ is symmetric (as $K$ is) and use $k_x$ as a shorthand notation for the column vector.

Note that, to make a prediction, once again, we only need to know how to get $\phi(x_n)^T \phi(x)$.
Inner products between features

Due to their central roles, let us examine more closely the inner products $\phi(x_m)^T \phi(x_n)$ for a pair of data points $x_m$ and $x_n$.

**Polynomial-based nonlinear basis functions** consider the following $\phi(x)$:

$$\phi : x \rightarrow \phi(x) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix}$$
Due to their central roles, let us examine more closely the inner products $\phi(x_m)^T \phi(x_n)$ for a pair of data points $x_m$ and $x_n$.

**Polynomial-based nonlinear basis functions** consider the following $\phi(x)$:

$$\phi : x \rightarrow \phi(x) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix}$$

This gives rise to an inner product in a special form,

$$\phi(x_m)^T \phi(x_n) = x_m^2 x_n^2 + 2x_m x_m x_n x_n + x_m^2 x_n^2$$

$$= (x_m x_n + x_m x_n)^2 = (x_m^T x_n)^2$$

Namely, the inner product can be computed by a function $(x_m^T x_n)^2$ defined in terms of the original features, *without knowing* $\phi(\cdot)$. 
A more challenging example

Consider the following mapping, which is parameterized by a parameter \( \psi_\theta(x) \):

\[
\psi_\theta(x) = \begin{pmatrix}
\cos(\theta x_1) \\
\sin(\theta x_1) \\
\cos(\theta x_2) \\
\sin(\theta x_2)
\end{pmatrix}
\]

The inner product for transformed \( x_m \) and \( x_n \) is thus,

\[
\psi_\theta(x_m)^T \psi_\theta(x_n) = \cos(\theta(x_{m1} - x_{n1})) + \cos(\theta(x_{m2} - x_{n2}))
\]

Note that, once again, the inner product can be computed alternatively with the function in the right-hand-side, which depends on the original features only. We will make it further more interesting.
Concatenating into a long feature vector

Now, consider \((L + 1)\) \(\theta_s\), drawn from \([0 \ 2\pi]\) evenly, and make another mapping

\[
\phi_L(x) = \begin{pmatrix}
\psi_0(x) \\
\psi_{\frac{2\pi}{L}}(x) \\
\psi_{\frac{2\pi}{L}}(x) \\
\vdots \\
\psi_{\frac{L\ 2\pi}{L}}(x)
\end{pmatrix}
\]
Concatenating into a long feature vector

Now, consider \((L + 1)\) \(\theta_s\), drawn from \([0 \ 2\pi]\) evenly, and make another mapping

\[
\phi_L(x) = \left( \begin{array}{c}
\psi_0(x) \\
\psi_{\frac{2\pi}{L}}(x) \\
\psi_{2\frac{2\pi}{L}}(x) \\
\vdots \\
\psi_{L\frac{2\pi}{L}}(x) \\
\end{array} \right)
\]

What is the inner product?

\[
\phi_L(x_m)^T \phi_L(x_n) = \sum_{l=0}^{L} \psi_{l\frac{2\pi}{L}}(x_m)^T \psi_{l\frac{2\pi}{L}}(x_n)
\]

\[
= \sum_{l=0}^{L} \cos(l\frac{2\pi}{L}(x_{m1} - x_{n1})) + \cos(l\frac{2\pi}{L}(x_{m2} - x_{n2}))
\]
What if $L = +\infty$?

Instead of summing up $(L + 1)$ terms, we will be integrating

$$
\phi_{\infty}(x_m)^T \phi_{\infty}(x_n) = \lim_{L \to +\infty} \phi_L(x_m)^T \phi_L(x_n)
$$

$$
= \int_0^{2\pi} \cos(\theta(x_{m1} - x_{n1})) + \cos(\theta(x_{m2} - x_{n2})) \, d\theta
$$

$$
= 1 - \frac{\sin(2\pi(x_{m1} - x_{n1}))}{x_{m1} - x_{n1}} + 1 - \frac{\sin(2\pi(x_{m2} - x_{n2}))}{x_{m2} - x_{n2}}
$$

While as before, the right-hand-side depends on only the original features. It actually computes the inner product of two infinite-dimensional feature vectors! (Since $L \to +\infty$, the number of $\psi_i^{2\pi \frac{L}{L}}(x)$ is infinite, hence the dimensionality of $\phi_{\infty}(x)$.)

In other words, while we cannot write down every dimension of $\phi_{\infty}(x)$, we can compute its inner product easily using a function defined on the original finite feature space.
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

$$k(x_m, x_n) = (x_m^T x_n)^2$$
$$k(x_m, x_n) = 2 - \sin(2\pi(x_m - x_n))^2$$

Examples that are not kernels

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

are not our desired kernel function as it cannot be written as inner products between two vectors.
Kernel functions

**Definition**: a (positive semidefinite) kernel function $k(\cdot, \cdot)$ is a bivariate function that satisfies the following properties. For any $\mathbf{x}_m$ and $\mathbf{x}_n$,

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

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

**Examples we have seen**

$$k(\mathbf{x}_m, \mathbf{x}_n) = (\mathbf{x}_m^T \mathbf{x}_n)^2$$

$$k(\mathbf{x}_m, \mathbf{x}_n) = 2 - \frac{\sin(2\pi(x_{m1} - x_{n1}))}{x_{m1} - x_{n1}} - \frac{\sin(2\pi(x_{m2} - x_{n2}))}{x_{m2} - x_{n2}}$$
**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.
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* \( x_1, x_2, \ldots, \) and \( x_N \), 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

$$
K = \Phi \Phi^T = \\
\begin{pmatrix}
\phi(x_1)^T \phi(x_1) & \phi(x_1)^T \phi(x_2) & \cdots & \phi(x_1)^T \phi(x_N) \\
\phi(x_2)^T \phi(x_1) & \phi(x_2)^T \phi(x_2) & \cdots & \phi(x_2)^T \phi(x_N) \\
\cdots & \cdots & \ddots & \cdots \\
\phi(x_N)^T \phi(x_1) & \phi(x_N)^T \phi(x_2) & \cdots & \phi(x_N)^T \phi(x_N)
\end{pmatrix}
$$
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**

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

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 $\|x_m - x_n\|_2^2$ is not a positive semidefinite kernel?

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

$$K = \begin{pmatrix} 0 & \|x_1 - x_2\|_2^2 \\ \|x_1 - x_2\|_2^2 & 0 \end{pmatrix}$$

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.
- ... 

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., "kernerlizing" the methods, thus, introducing nonlinear features/basis.

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

When we talk about support vector machines next lecture, we will see the trick one more time.
Kernel methods

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 kernelized 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 $x_n$

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

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. The nearest neighbor of a point $x$ is thus found with

$$\text{arg min}_n d_{\text{KERNEL}}(x, x_n)$$
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
Outline

1 Administration

2 Review of last lecture

3 Kernel methods

4 Support vector machines
   - Hinge loss
   - Primal formulation of SVM
   - Basic Lagrange duality theory
   - Dual formulation 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 $f(x) = \mathbf{w}^\top \phi(x) + b$,

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

**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
- 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

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

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

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

where \(C\) is identified as \(1/\lambda\).
Primal formulation of support vector machines (SVM)

Minimizing the total hinge loss on all the training data

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

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 \(1/\lambda\). We further rewrite into another equivalent form

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

s.t. \(\max(0, 1 - y_n[w^T \phi(x_n) + b]) = \xi_n, \quad \forall \ n\)
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 slack variables.
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 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

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

where all \( \xi_n \) are called 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.
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 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.
- 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.)
Basic Lagrange duality theory

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**

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

s.t. \[ 0 \leq \alpha_n \leq C, \quad \forall \ n \]

\[
\sum_{n} \alpha_n y_n = 0
\]

**Remarks**

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

Drs. Sha & Liu  
({feisha,yanliu.cs}@usc.edu)  
CSCI567 Machine Learning (Fall 2014)  
October 7, 2014 38 / 48
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)
- 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. $0 \leq \alpha_n \leq C, \quad \forall \ n$

$$\sum_n \alpha_n y_n = 0$$

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

\[ w = \sum_n y_n \alpha_n \phi(x_n) \quad \leftarrow \text{Linear combination of the input features!} \]
Support vector machines

Dual formulation of SVM

Recovering solution to the primal formulation

Weights

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

b

\[
\begin{align*}
b &= [y_n - w^T \phi(x_n)] = [y_n - \sum_m y_m \alpha_m k(x_m, x_n)], \\
&\text{for any } C > \alpha_n > 0
\end{align*}
\]
Recovering solution to the primal formulation

Weights

\[ w = \sum_n y_n \alpha_n \phi(x_n) \leftarrow \text{Linear combination of the input features!} \]

\[ b \]

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

Making prediction on a test point \( x \)

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

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
Support vector machines
Dual formulation of SVM

Deriving the dual

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

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

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

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

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

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

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

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

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

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

Substitute the solution to the primal back into the Lagrangian

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

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

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

Dual formulation of SVM

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

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

\[
\text{s.t.} \quad \alpha_n \geq 0, \quad \forall \ n
\]
\[
\sum_n \alpha_n y_n = 0
\]
\[
C - \lambda_n - \alpha_n = 0, \quad \forall \ n
\]
\[
\lambda_n \geq 0, \quad \forall \ n
\]

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. \[0 \leq \alpha_n \leq C, \quad \forall \ n\]

\[
\sum_n \alpha_n y_n = 0
\]
Recover the solution

The primal variable $w$ is identified as

$$w = \sum_n \alpha_n y_n \phi(x_n)$$
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)

$$\lambda_n \xi_n = 0$$

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