Support Vector Machine (cont.)
Summarization So Far

- We demonstrated that we prefer to have linear classifiers with large margin.
- We formulated the problem of finding the maximum margin linear classifier as a quadratic optimization problem.
- This problem can be solved by solving its dual problem, and efficient QP algorithms are available.
- Problem solved?
Non-separable Data and Noise

• What if the data is not linearly separable?

• Even when linearly separable, we may have noise in data, and maximum margin classifier is not robust to noise!
Soft Margin

- Allow functional margins to be less than 1
  - But will charge a penalty
Soft-Margin Maximization

Hard margin

\[
\begin{align*}
\min_{w,b} & \quad \frac{1}{2} \|w\|^2 \\
\text{subject to:} & \quad y^i(w \cdot x^i + b) \geq 1, \quad i = 1, \ldots, N
\end{align*}
\]

Soft margin

\[
\begin{align*}
\min_{w,b,\xi} & \quad \frac{1}{2} \|w\|^2 + c \sum_i \xi_i \\
\text{subject to:} & \quad y^i(w \cdot x^i + b) \geq 1 - \xi_i, \quad i = 1, \ldots, N \\
& \quad \xi_i \geq 0, \quad i = 1, \ldots, N
\end{align*}
\]

- Introduce **slack variables** $\xi_i$ to allow functional margins to be smaller than 1
- Parameter $c$ controls the tradeoff between maximizing the margin and fitting the training example
Dual Formulation of Soft Margin

\[ \begin{align*}
\max & \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y^i y^j < x^i \cdot x^j > \\
\text{Subject to:} & \quad \sum_{i=1}^{N} \alpha_i y^i = 0 \\
& \quad 0 \leq \alpha_i \leq c \quad i=1, \ldots, N
\end{align*} \]

• The dual problem is almost identical to the separable case, except for that \( \alpha_i \)'s are now bounded by \( c \)

• \( c \) controls the tradeoff between maximizing margin and fitting training data

• It’s effect is to put a **box constraint** on \( \alpha \), the weights of the support vectors

• It limits the influence of outliers
Dual Formulation of Soft Margin

\[ \max \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y^i y^j < x^i \cdot x^j > \]
Subject to: \[ \sum_{i=1}^{N} \alpha_i y^i = 0 \]
\[ 0 \leq \alpha_i \leq c \quad i=1,\ldots, N \]

- We now have also have support vectors for data that have functional margin less than one (in addition to those that equal 1), but there \( \alpha_i \)'s will only equal c

**support vectors (\( \alpha_i > 0 \))**
- \( c > \alpha_i > 0 \): \( y^i (w \cdot x^i + b) = 1 \), i.e., \( \xi_i = 0 \)
- \( \alpha_i = c \): \( y^i (w \cdot x^i + b) \leq 1 \), i.e., \( \xi_i \geq 0 \)

The optimal \( w \) can then be computed:
\[ w = \sum \alpha_i y^i x^i \]
Linear SVMs: Overview

• So far our classifier is a separating hyperplane.

• Most “important” training points are support vectors; they define the hyperplane.

• Quadratic optimization algorithms can identify which training points $x^i$ are support vectors with non-zero Lagrange multipliers $\alpha_i$.

• For both training and classification, we see training data appear only inside inner products:

\[
\begin{align*}
Q(\alpha) &= \sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y^i y^j <x^i \cdot x^j> \\
\text{is maximized and} & \\
(1) & \quad \sum \alpha_i y^i = 0 \\
(2) & \quad 0 \leq \alpha_i \leq c \text{ for all } \alpha_i
\end{align*}
\]

\[f(x) = \sum \alpha_i y^i <x^i \cdot x> + b\]
Non-linear SVMs

• Datasets that are linearly separable with some noise work out great:

• But what are we going to do if the dataset is just too hard?

• How about… mapping data to a higher-dimensional space:
Non-linear SVMs: Feature Spaces

• General idea: For any data set, the original input space can always be mapped to some higher-dimensional feature space such that the data is linearly separable:

\[ \Phi: x \rightarrow \phi(x) \]
**Example: Quadratic Space**

- Assume $m$ input dimensions
  \[ \mathbf{x} = (x_1, x_2, \ldots, x_m) \]

- Number of quadratic terms:
  \[ (m+2)\text{-choose-2} = \frac{(m+2)(m+1)}{2} \]

- The number of dimensions increase rapidly - expensive to compute!

You may be wondering about the $\sqrt{2}$'s
You will find out why they are there soon!
Kernel Function

- The linear classifier relies on inner product between vectors $K(x^i, x^j) = \langle x^i \cdot x^j \rangle$

- If every data point is mapped into high-dimensional space via some transformation $\Phi$: $x \rightarrow \phi(x)$, the inner product becomes:

$$K(x_i, x_j) = \langle \phi(x^i) \cdot \phi(x^j) \rangle$$

- A **kernel function** is a function that is equivalent to an inner product in some feature space.

- Example: we can define a kernel as

$$K(x^i, x^j) = (x^i \cdot x^j + 1)^2$$

*This is equivalent to mapping to the quadratic space!*
Example: Quadratic Kernel

Consider a 2-d input space: (generalizes to n-d)

\[
K(x^i, x^j) = (x^i \cdot x^j + 1)^2 \\
= (x_1^i x_1^j + x_2^i x_2^j + 1)^2 \\
= x_1^{i^2} x_1^{j^2} + 2 x_1^i x_2^i x_1^j x_2^j + x_2^{i^2} x_2^{j^2} + 2 x_1^i x_1^j + 2 x_2^i x_2^j + 1 \\
= (x_1^{i^2}, \sqrt{2} x_1^i x_1^j, x_2^{i^2}, \sqrt{2} x_1^i, \sqrt{2} x_2^i, 1) \cdot (x_1^{j^2}, \sqrt{2} x_1^j x_2^j, x_2^{j^2}, \sqrt{2} x_1^j, \sqrt{2} x_2^j, 1) \\
= \Phi(x^i) \cdot \Phi(x^j)
\]

A kernel function *implicitly* maps data to a high-dimensional space (without the need to compute each \(\phi(x)\) explicitly).

Computing inner product of quadratic features is \(O(m^2)\) time vs. \(O(m)\) time for kernel
Non-linear SVMs

• Dual problem formulation:

\[
\text{Find } \alpha_1 \ldots \alpha_N \text{ such that } \\
\sum \alpha_i - \frac{1}{2} \sum \alpha_i \alpha_j y_i y_j \langle x^i, x^j \rangle \text{ is maximized and} \\
(1) \sum \alpha_i y_i = 0 \\
(2) \quad 0 \leq \alpha_i \leq c \text{ for all } \alpha_i
\]

\[f(x) = \sum \alpha_i y_i \langle x^i, x \rangle + b\]

• To classify a given new data point \(x\), we compute

\[f(x) = \sum \alpha_i y_i \langle x^i, x \rangle + b\]

• Optimization techniques for finding \(\alpha_i\)'s remain the same!

• This shows the utility of the dual formulation.
Kernel Functions

• In practical, the user specifies the kernel function $K$, without explicitly stating the transformation $\phi(\cdot)$

• Given a kernel function, finding its corresponding transformation can be very cumbersome
  – This is why people only specify the kernel function without worrying about the exact transformation

• Another view: a kernel function computes some kind of measure of similarity between objects

• If you have a reasonable measure of similarity for your application, can we use it as the kernel in an SVM?
What Functions are Kernels?

- Consider some finite set of \( m \) points, let matrix \( K \) be defined as follows:

\[
\begin{array}{cccccc}
K(x^1, x^1) & K(x^1, x^2) & K(x^1, x^3) & \ldots & K(x^1, x^m) \\
K(x^2, x^1) & K(x^2, x^2) & K(x^2, x^3) & & K(x^2, x^m) \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
K(x^m, x^1) & K(x^m, x^2) & K(x^m, x^3) & \ldots & K(x^m, x^m)
\end{array}
\]

- This is called the **Kernel Matrix**

- Mercer’s theorem:
  A function \( K \) is a kernel function iff for any finite sample \( \{x^1, x^2, \ldots, x^m\} \), its corresponding kernel matrix is symmetric and semi-definite.
Examples of Kernel Functions

- **Linear:** \( K(x^i, x^j) = <x^i, x^j> \)
  - Mapping \( \Phi: \mathbf{x} \rightarrow \phi(\mathbf{x}) \), where \( \phi(\mathbf{x}) \) is \( \mathbf{x} \) itself

- **Polynomial of power \( p \):** \( K(x^i, x^j) = (1 + x^i \cdot x^j)^p \)
  - Mapping \( \Phi: \mathbf{x} \rightarrow \phi(\mathbf{x}) \), where \( \phi(\mathbf{x}) \) has \( \binom{d + p}{p} \) dimensions

- **Gaussian (radial-basis function):** \( K(x^i, x^j) = e^{-\frac{||x_i - x_j||^2}{2\sigma^2}} \)
  - Mapping \( \Phi: \mathbf{x} \rightarrow \phi(\mathbf{x}) \), where \( \phi(\mathbf{x}) \) is *infinite-dimensional*: every point is mapped to a function (a Gaussian); combination of functions for support vectors is the separator.

- **Higher-dimensional space still has intrinsic dimensionality \( d \), but linear separators in it correspond to non-linear separators in original space.**
Critical Steps for Using SVM

• Select the kernel function to use (important but often trickiest part of SVM)
  – In practice, a low degree polynomial kernel or RBF kernel with a reasonable width is a good initial try and usually support by off-the-shelf software

• Select the parameter of the kernel function and the value of $c$
  – You can use the values suggested by the SVM software
    see www.kernel-machines.org/software.html for a list of available software
  – You can set apart a validation set to determine the values of the parameter
SVM Summary

• Advantages of SVMs
  – polynomial-time exact optimization rather than approximate methods
    • unlike decision trees and neural networks
  – Kernels allow very flexible hypotheses
  – Can be applied to very complex data types, e.g., graphs, sequences

• Disadvantages of SVMs
  – Must choose a good kernel and kernel parameters
  – Very large problems are computationally intractable
    • quadratic in number of examples
    • problems with more than 20k examples are very difficult to solve exactly

Recent developments in SVM and constrained convex optimization has made this much less an issue. It is now capable of dealing much larger datasets, with hundreds of thousands of examples.