Man-Wai MAK

Dept. of Electronic and Information Engineering,
The Hong Kong Polytechnic University

enmwmak@polyu.edu.hk
http://www.eie.polyu.edu.hk/~mwmak

References:
Lagrange multipliers and constrained optimization, www.khancademy.org

October 9, 2018
Overview

1 Motivations: Why Study Constrained Optimization and SVM
   - Why Study Constrained Optimization
   - Why Study SVM

2 Constrained Optimization
   - Hard-Constrained Optimization
   - Lagrangian Function
   - Inequality Constraint
   - Multiple Constraints
   - Software Tools

3 Support Vector Machines
   - Linear SVM: Separable Case
   - Linear SVM: Fuzzy Separation (Optional)
   - Nonlinear SVM
   - SVM for Pattern Classification
   - Software Tools
Why Study Constrained Optimization?

- Constrained optimization is used in almost every discipline:
  - **Wireless Communication**: “Energy-constrained modulation optimization,” *IEEE Transactions on Wireless Communications*, vol. 4, no. 5, pp. 2349-2360, Sept. 2005
Why Study SVM?

- SVM is a typical application of constraint optimization.
- SVMs are used everywhere:
Constrained optimization is the process of optimizing an objective function with respect to some variables in the presence of constraints on those variables.

The objective function is either

- a cost function or energy function which is to be minimized, or
- a reward function or utility function, which is to be maximized.

Constraints can be either

- hard constraints which set conditions for the variables that are required to be satisfied, or
- soft constraints which have some variable values that are penalized in the objective function if the conditions on the variables are not satisfied.
A general constrained minimization problem:

\[
\begin{align*}
\min & \quad f(x) \\
\text{subject to} & \quad g_i(x) = c_i \quad \text{for } i = 1, \ldots, n \quad (\text{Equality constraints}) \\
& \quad h_j(x) \geq d_j \quad \text{for } j = 1, \ldots, m \quad (\text{Inequality constraints})
\end{align*}
\]

where \( g_i(x) = c_i \) and \( h_j(x) \geq d_j \) are called hard constraints.

If the constrained problem has only equality constraints, the method of Lagrange multipliers can be used to convert it into an unconstrained problem whose number of variables is the original number of variables plus the original number of equality constraints.
**Example:** Maximization of a function of two variables with equality constraints:

\[
\begin{align*}
\max & \quad f(x, y) \\
\text{subject to} & \quad g(x, y) = 0
\end{align*}
\]

At the optimal point \((x^*, y^*)\), the gradient of \(f(x, y)\) and \(g(x, y)\) are anti-parallel, i.e., \(\nabla f(x^*, y^*) = -\lambda \nabla g(x^*, y^*)\), where \(\lambda\) is called the Lagrange multiplier. (See Tutorial for explanation.)
Example:

\[
\begin{align*}
& \text{max} \quad f(x, y) = x^2y \\
& \text{subject to} \quad x^2 + y^2 = 1
\end{align*}
\]

Note that the red curve \((x^2 + y^2 = 1)\) is of 2-dimension.
Extension to function of $D$ variables:

$$\begin{align*}
\max & \quad f(x) \\
\text{subject to} & \quad g(x) = 0
\end{align*}$$

(3)

where $x \in \mathbb{R}^D$. Optimal occurs when

$$\nabla f(x) + \lambda \nabla g(x) = 0.$$  

(4)

Note that the red curve is of dimension $D - 1$. 

- Constrained Optimization
Constrained Optimization

- **Left:** Gradients of the objective function \( f(x, y) = x^2 y \)
- **Right:** Gradients of \( g(x, y) = x^2 + y^2 \).

Note that \( \lambda < 0 \) in this example, which means that the gradients of \( f(x, y) \) and \( g(x, y) \) are parallel at the optimal point.
Define the Lagrangian function as

\[ L(x, \lambda) \equiv f(x) + \lambda g(x) \]  

where \( \lambda \neq 0 \) is the Lagrange multiplier.

The optimal condition (Eq. 4) will be satisfied when \( \nabla_x L = 0 \).

Note that \( \partial L/\partial \lambda = 0 \) leads to the constrained equation \( g(x) = 0 \).

The constrained maximization can be written as:

\[
\max_{\lambda} \quad L(x, \lambda) = f(x) + \lambda g(x)
\]

subject to \( \lambda \neq 0, g(x) = 0 \)
Find the stationary point of the function \( f(x_1, x_2): \)

\[
\max \quad f(x_1, x_2) = 1 - x_1^2 - x_2^2 \\
\text{subject to} \quad g(x_1, x_2) = x_1 + x_2 - 1 = 0
\]  

Lagrangian function:

\[
L(x, \lambda) = 1 - x_1^2 - x_2^2 + \lambda(x_1 + x_2 - 1)
\]
Differenting $L(x, \lambda)$ w.r.t. $x_1$, $x_2$, and $\lambda$ and set the results to 0, we obtain

\[-2x_1 + \lambda = 0\]
\[-2x_2 + \lambda = 0\]
\[x_1 + x_2 - 1 = 0\]

The solution is $(x_1^*, x_2^*) = (\frac{1}{2}, \frac{1}{2})$, and the corresponding $\lambda = 1$.

As $\lambda > 0$, the gradients of $f(x_1, x_2)$ and $g(x_1, x_2)$ are anti-parallel at $(x_1^*, x_2^*)$. 
Maximization with inequality constraint

\[
\begin{align*}
\max & \quad f(x) \\
\text{subject to} & \quad g(x) \geq 0
\end{align*}
\] (8)

Two possible solutions for the max of \( L(x, \mu) = f(x) + \mu g(x) \):

**Inactive Constraint:** \( g(x) > 0, \quad \mu = 0, \quad \nabla f(x) = 0 \)

**Active Constraint:** \( g(x) = 0, \quad \mu > 0, \quad \nabla f(x) = -\mu \nabla g(x) \) (9)

Therefore, the maximization can be rewritten as

\[
\begin{align*}
\max & \quad L(x, \mu) = f(x) + \mu g(x) \\
\text{subject to} & \quad g(x) \geq 0, \quad \mu \geq 0, \quad \mu g(x) = 0
\end{align*}
\] (10)

which is known as the Karush-Kuhn-Tucker (KKT) condition.
Inequality Constraint

- For minimization,

\[
\min f(x) \\
\text{subject to } g(x) \geq 0
\]  \hspace{1cm} (11)

- We can also express the minimization as

\[
\min L(x, \mu) = f(x) - \mu g(x) \\
\text{subject to } g(x) \geq 0, \mu \geq 0, \mu g(x) = 0
\]  \hspace{1cm} (12)
Multiple Constraints

- Maximization with multiple equality and inequality constraints:

\[
\begin{align*}
\max & \quad f(x) \\
\text{subject to} & \quad g_j(x) = 0 \text{ for } j = 1, \ldots, J \\
& \quad h_k(x) \geq 0 \text{ for } k = 1, \ldots, K.
\end{align*}
\]  

(13)

- This maximization can be written as

\[
\begin{align*}
\max & \quad L(x, \{\lambda_j\}, \{\mu_k\}) = f(x) + \sum_{j=1}^{J} \lambda_j g_j(x) + \sum_{k=1}^{K} \mu_k h_k(x) \\
\text{subject to} & \quad \lambda_j \neq 0, g_j(x) = 0 \text{ for } j = 1, \ldots, J \text{ and} \\
& \quad \mu_k \geq 0, h_k(x) \geq 0, \mu_k h_k(x) = 0 \text{ for } k = 1, \ldots, K.
\end{align*}
\]  

(14)
Matlab Optimization Toolbox: `fmincon` can find the minimum of a function subject to nonlinear multivariable constraints.

Python: `scipy.optimize.minimize` provides a common interface to unconstrained and constrained minimization algorithms for multivariate scalar functions.
Consider a training set \( \{ \mathbf{x}_i, y_i; \ i = 1, \ldots, N \} \in \mathcal{X} \times \{+1, -1\} \) shown below, where \( \mathcal{X} \) is the set of input data in \( \mathbb{R}^D \) and \( y_i \) are the labels.

\[
\begin{align*}
\mathbf{w} \cdot \mathbf{x} + b &= +1 \\
\mathbf{w} \cdot \mathbf{x} + b &= 0 \\
\mathbf{w} \cdot \mathbf{x} + b &= -1
\end{align*}
\]

**Figure:** Linear SVM on 2-D space

- \( \square: y_i = +1; \circ: y_i = -1. \)
A linear support vector machine (SVM) aims to find a decision plane (a line for the case of 2D)

\[ \mathbf{x} \cdot \mathbf{w} + b = 0 \]

that maximizes the margin of separation (see Fig. 1).

Assume that all data points satisfy the constraints:

\[ x_i \cdot \mathbf{w} + b \geq +1 \quad \text{for} \quad i \in \{1, \ldots, N\} \quad \text{where} \quad y_i = +1. \]  \hfill (15)

\[ x_i \cdot \mathbf{w} + b \leq -1 \quad \text{for} \quad i \in \{1, \ldots, N\} \quad \text{where} \quad y_i = -1. \]  \hfill (16)

Data points \( x_1 \) and \( x_2 \) in previous page satisfy the equality constraint:

\[ x_1 \cdot \mathbf{w} + b = +1 \]
\[ x_2 \cdot \mathbf{w} + b = -1 \]  \hfill (17)
Using Eq. 17 and Fig. 1, the distance between the two separating hyperplane (also called the margin of separation) can be computed:

\[ d(w) = (x_1 - x_2) \cdot \frac{w}{\|w\|} = \frac{2}{\|w\|} \]

Maximizing \( d(w) \) is equivalent to minimizing \( \|w\|^2 \). So, the constrained optimization problem in SVM is

\[
\min \frac{1}{2} \|w\|^2 \\
\text{subject to } y_i(x_i \cdot w + b) \geq 1 \quad \forall i = 1, \ldots, N
\]

Equivalently, minimizing a Lagrangian function:

\[
\min L(w, b, \{\alpha_i\}) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^{N} \alpha_i [y_i(x_i \cdot w + b) - 1] \\
\text{subject to } \alpha_i \geq 0, \quad y_i(x_i \cdot w + b) - 1 \geq 0, \\
\alpha_i [y_i(x_i \cdot w + b) - 1] = 0, \quad \forall i = 1, \ldots, N
\]
Setting

\[ \frac{\partial}{\partial b} L(w, b, \{\alpha_i\}) = 0 \quad \text{and} \quad \frac{\partial}{\partial w} L(w, b, \{\alpha_i\}) = 0, \quad (20) \]

subject to the constraint \( \alpha_i \geq 0 \), results in

\[ \sum_{i=1}^{N} \alpha_i y_i = 0 \quad \text{and} \quad w = \sum_{i=1}^{N} \alpha_i y_i x_i. \quad (21) \]

Substituting these results back into the Lagrangian function:

\[
L(w, b, \{\alpha_i\}) = \frac{1}{2} (w \cdot w) - \sum_{i=1}^{N} \alpha_i y_i (x_i \cdot w) - \sum_{i=1}^{N} \alpha_i y_i b + \sum_{i=1}^{N} \alpha_i
\]

\[
= \frac{1}{2} \sum_{i=1}^{N} \alpha_i y_i x_i \cdot \sum_{j=1}^{N} \alpha_j y_j x_j - \sum_{i=1}^{N} \alpha_i y_i x_i \cdot \sum_{j=1}^{N} \alpha_j y_j x_j + \sum_{i=1}^{N} \alpha_i
\]

\[
= \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j).
\]
This results in the following *Wolfe dual* formulation:

$$\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$

subject to

$$\sum_{i=1}^{N} \alpha_i y_i = 0 \quad \text{and} \quad \alpha_i \geq 0, i = 1, \ldots, N.$$  \hspace{1cm}(22)

The solution contains two kinds of Lagrange multiplier:

1. $\alpha_i = 0$: The corresponding $x_i$ are irrelevant
2. $\alpha_i > 0$: The corresponding $x_i$ are critical

$x_k$ for which $\alpha_k > 0$ are called *support vectors*. 
The SVM output is given by

\[ f(x) = \mathbf{w} \cdot \mathbf{x} + b \]

\[ = \sum_{k \in S} \alpha_k y_k \mathbf{x}_k \cdot \mathbf{x} + b \]

where \( S \) is the set of indexes for which \( \alpha_k > 0 \).

\( b \) can be computed by using the KTT condition, i.e., for any \( k \) such that \( y_k = 1 \) and \( \alpha_k > 0 \), we have

\[ \alpha_k [y_k (\mathbf{x}_k \cdot \mathbf{w} + b) - 1] = 0 \]

\[ \implies b = 1 - \mathbf{x}_k \cdot \mathbf{w}. \]
If the data patterns are not separable by a linear hyperplane, a set of slack variables \( \{\xi = \xi_1, \ldots, \xi_N\} \) is introduced with \( \xi_i \geq 0 \) such that the inequality constraints in SVM become

\[
y_i(x_i \cdot w + b) \geq 1 - \xi_i \quad \forall i = 1, \ldots, N.
\] (23)

The slack variables \( \{\xi_i\}_{i=1}^N \) allow some data to violate the constraints in Eq. 18.

The value of \( \xi_i \) indicates the degree of violation of the constraint.

The minimization problem becomes

\[
\min \frac{1}{2}\|w\|^2 + C \sum_i \xi_i, \quad \text{subject to} \quad y_i(x_i \cdot w + b) \geq 1 - \xi_i, \quad (24)
\]

where \( C \) is a user-defined penalty parameter to penalize any violation of the safety margin for all training data.
The new Lagrangian is

\[ L(w, b, \alpha) = \frac{1}{2} \|w\|^2 + C \sum_{i} \xi_i - \sum_{i=1}^{N} \alpha_i (y_i (x_i \cdot w + b) - 1 + \xi_i) - \sum_{i=1}^{N} \beta_i \xi_i, \]

where \( \alpha_i \geq 0 \) and \( \beta_i \geq 0 \) are, respectively, the Lagrange multipliers to ensure that \( y_i (x_i \cdot w + b) \geq 1 - \xi_i \) and that \( \xi_i \geq 0 \).

Differentiating \( L(w, b, \alpha) \) w.r.t. \( w \), \( b \), and \( \xi_i \), we obtain the Wolfe dual:

\[
\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)
\]

subject to \( 0 \leq \alpha_i \leq C, \ i = 1, \ldots, N, \sum_{i=1}^{N} \alpha_i y_i = 0. \)
Linear SVM: Fuzzy Separation (Optional)

Three types of support vectors:

1. On the margin:
   \[ C > \alpha_i > 0, \xi_i = 0 \]
   \[ y_i (w^T x_i + b) = 1 \]
   \[ \alpha_{11} = 0.44; \xi_{11} = 0 \]
   \[ \alpha_1 = 2.85; \xi_1 = 0 \]

2. Inside the margin:
   \[ \alpha_i = C; 0 < \xi_i < 2 \]
   \[ y_i (w^T x_i + b) \leq 1 \]
   \[ \alpha_{10} = 10; \xi_{10} = 0.667 \]

3. Outside the margin:
   \[ \alpha_i = C; \xi_i \geq 2 \]
   \[ y_i (w^T x_i + b) \leq 1 \]
   \[ \alpha_{20} = 10; \xi_{20} = 2.667 \]
Nonlinear SVM

- Assume that we have a nonlinear function $\phi(x)$ that maps $x$ from the input space to a much higher (possibly infinite) dimensional space called the feature space.

- While data are not linearly separable in the input space, they will become linearly separable in the feature space.

Decision boundary

Nonlinear Kernels

The decision function in Eq. 4.5.1 can be implemented by a two-layer architecture depicted in Figure 4.9, where it is shown that the original input space is mapped to a new feature space, manifested by the middle hidden-layer in the network.
Nonlinear SVM

- A 1-D problem requiring two decision boundaries (thresholds).
- 1-D linear SVMs could not solve this problem because they can only provide one decision threshold.
We may use a nonlinear function $\phi$ is to perform the mapping:

$$\phi : x \rightarrow [x \ x^2]^T.$$

The decision boundary in Fig. 28(b) is a straight line that can perfectly separate the two classes.

We may write the decision function as

$$x^2 - c = [0 \ 1]^T \begin{bmatrix} x \\ x^2 \end{bmatrix} - c = 0$$

Or equivalently,

$$w^T \phi(x) + b = 0, \quad (27)$$

where $w = [0 \ 1]^T$, $\phi(x) = [x \ x^2]^T$, and $b = -c$. 
Nonlinear SVM

- **Left:** A 2-D example in which linear SVMs will not be able to perfectly separate the two classes.
- **Right:** By transforming $x = [x_1 \ x_2]^T$ to:

  $$\phi : x \rightarrow [x_1^2 \ \sqrt{2}x_1x_2 \ x_2^2]^T,$$

  (28)

we will be able to use a linear SVM to separate the 2 classes in three dimensional space.
The linear SVM has the form
\[ f(x) = \sum_{i \in S} \alpha_i y_i \phi(x_i)^T \phi(x) + b \]
\[ = w^T \phi(x) + b, \]
where \( S \) is the set of support vector indexes and \( w = \sum_{i \in S} \alpha_i y_i \phi(x_i) \).

In this simple problem, the dot products \( \phi(x_i)^T \phi(x_j) \) for any \( x_i \) and \( x_j \) in the input space can be easily evaluated
\[ \phi(x_i)^T \phi(x_j) = x_{i1}^2 x_{j1}^2 + 2x_{i1}x_{i2}x_{j1}x_{j2} + x_{i2}^2 x_{j2}^2 = (x_i^T x_j)^2. \] (29)
The SVM output becomes
\[ f(x) = \sum_{i=1}^{N} \alpha_i y_i \phi(x_i) \cdot \phi(x) + b \]

However, the dimension of \( \phi(x) \) is very high and could be infinite in some cases, meaning that this function may not be implementable.

Fortunately, the dot product \( \phi(x_i) \cdot \phi(x) \) can be replaced by a kernel function:
\[ \phi(x_i) \cdot \phi(x) = \phi(x)^{T} \phi(x) = K(x_i, x) \]

which can be efficiently implemented.
Common kernel functions include

**Polynomial Kernel** :  
\[ K(x, x_i) = \left(1 + \frac{x \cdot x_i}{\sigma^2}\right)^p, \quad p > 0 \]  
(30)

**RBF Kernel** :  
\[ K(x, x_i) = \exp\left\{-\frac{\|x - x_i\|^2}{2\sigma^2}\right\} \]  
(31)

**Sigmoidal Kernel** :  
\[ K(x, x_i) = \frac{1}{1 + e^{-\frac{x \cdot x_i + b}{\sigma^2}}} \]  
(32)
Comparing kernels:

- **Linear SVM**, $C=1000.0$, #SV=7, acc=95.00%, normW=0.94
- **RBF SVM**, $2\sigma=8.0$, $C=1000.0$, #SV=7, acc=100.00%
- **Polynomial SVM**, degree=2, $C=10.0$, #SV=7, acc=90.00%
Figure: Decision boundaries produced by a 2nd-order polynomial kernel (top), a 3rd-order polynomial kernel (left), and an RBF kernel (right).
SVM for Pattern Classification

- SVM is good for binary classification:
  \[ f(x) > 0 \Rightarrow x \in \text{Class 1}; \quad f(x) \leq 0 \Rightarrow x \in \text{Class 2} \]

- To classify multiple classes, we use the one-vs-rest approach to converting \( K \) binary classifications to a \( K \)-class classification:

\[
f^{(0)}(x) = \sum_{i \in SV^0} y_i^{(0)} \alpha_i^{(0)} x_i^T x + b^{(0)}
\]

\[
f^{(9)}(x) = \sum_{i \in SV^9} y_i^{(9)} \alpha_i^{(9)} x_i^T x + b^{(9)}
\]

\[ k^* = \arg\max_k f^{(k)}(x) \]

Pick Max Score

Classifying digit ‘0’ and the rest

Convert to Vector

Classifying digit ‘9’ from the rest
Matlab: fitcsvm trains an SVM for two-class classification.

Python: svm from the sklearn package provides a set of supervised learning methods used for classification, regression and outliers detection.

C/C++: LibSVM is a library for SVM. It also has Java, Perl, Python, Cuda, and Matlab interface.

Java: SVM-JAVA implements sequential minimal optimization for training SVM in Java.

Javascript: http://cs.stanford.edu/people/karpathy/svmjs/demo/