San José State University

Math 251: Statistical and Machine Learning Classification

Support Vector Machine (SVM)

Dr. Guangliang Chen

Outline of the presentation:

- **Binary SVM**
  - Linearly separable, no outliers
  - Linearly separable, with outliers
  - Nonlinearly separable (Kernel SVM)
- **Multiclass SVM**
  - One-versus-one
  - One-versus-rest
- **Practical issues**

Main references

- **Olga Veksler's lecture**

  http://www.csd.uwo.ca/~olga/Courses/CS434a_541a/Lecture11.pdf

- **Jeff Howbert's lecture**

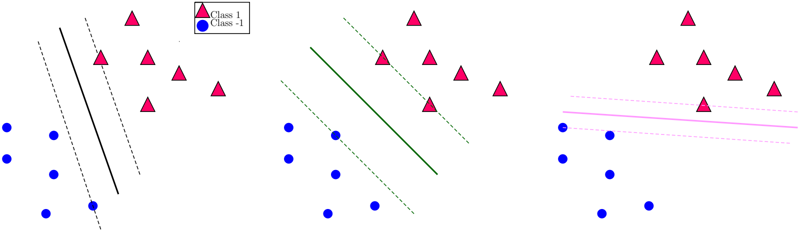
  http://courses.washington.edu/css581/lecture_slides/16_support_vector_machines.pdf

- **Chris Burges' tutorial**

  http://research.microsoft.com/pubs/67119/svmtutorial.pdf

What are support vector machines (SVM)?

Like LDA and Logistic regression, an SVM is also a linear classifier but seeks to find a maximum-margin boundary directly in the feature space.

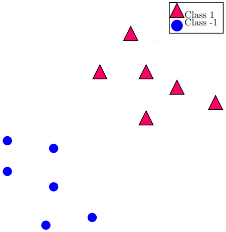


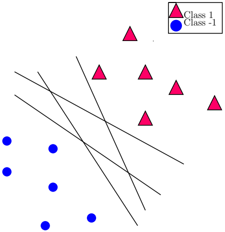
It was invented by Vapnik (during the end of last century) and considered one of the major developments in pattern recognition.

$\boxed{\text{Binary SVM: Linearly separable (no outliers)}}$

Support Vector Machine (SVM)

To introduce the idea of SVM, we consider binary classification first.

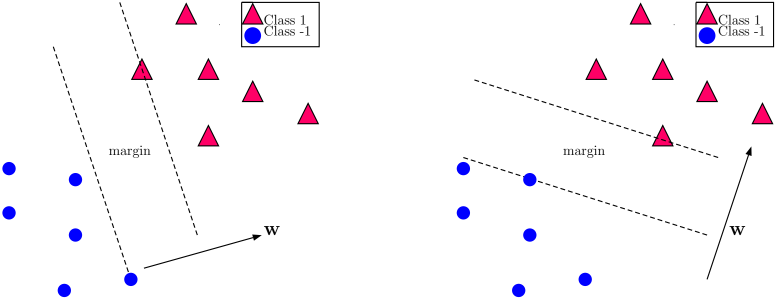




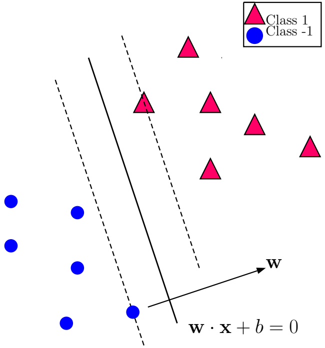
SVM effectively compares (under some criterion) all hyperplanes $\mathbf{w} \cdot \mathbf{x} + b = 0$, where $\mathbf{w}$ is a normal vector while b determines location.

The following specifies how to logically think about the problem:

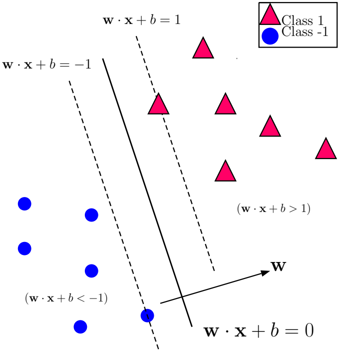
- Any fixed direction $\mathbf{w}$ determines a unique margin.



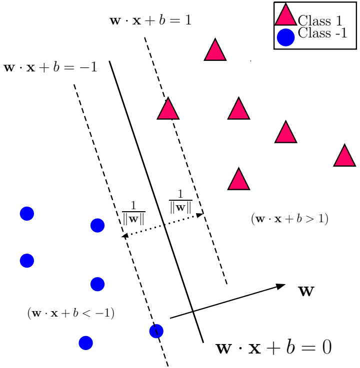
- We select b such that the center hyperplane is given by $\mathbf{w} \cdot \mathbf{x} + b = 0$. This is the **optimal** boundary *orthogonal to the given direction* $\mathbf{w}$, as it is equally far from the two classes.



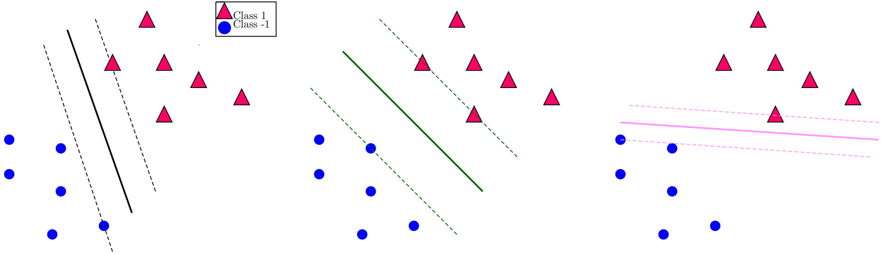
- Any scalar multiple of $\mathbf{w}$ and b denotes the same hyperplane. To uniquely fix the two parameters, we require the margin boundaries to have equations $\mathbf{w} \cdot \mathbf{x} + b = \pm 1$.



- Under such requirements, we can show that the margin between the two classes is exactly $\frac{2}{\|\mathbf{w}\|_2}$.



The larger the margin, the better the classifier



The binary SVM problem

Problem. Given training data $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ with labels $y_i = \pm 1$, SVM finds the optimal separating hyperplane by maximizing the class margin.

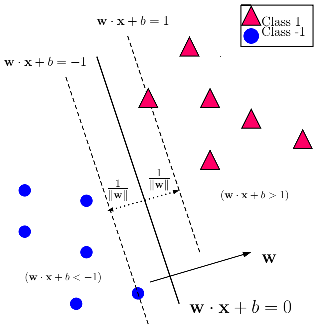
Specifically, it tries to solve

$$\max_{\mathbf{w},b} \frac{2}{\|\mathbf{w}\|_2} \quad \text{subject to}$$

$$\mathbf{w} \cdot \mathbf{x}_i + b \geq 1, \quad \text{if } y_i = +1;$$

$$\mathbf{w} \cdot \mathbf{x}_i + b \leq -1, \quad \text{if } y_i = -1$$

Remark. The classification rule for new data $\mathbf{x}$ is $y = \mathrm{sgn}(\mathbf{w} \cdot \mathbf{x} + b)$.



A more convenient formulation

The previous problem is equivalent to

$$\min_{\mathbf{w},b} \ \frac{1}{2}\|\mathbf{w}\|_2^2 \qquad \text{subject to} \quad y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 \text{ for all } 1 \leq i \leq n.$$

This is an optimization problem with linear, inequality constraints.

Remarks:

- The constraints determine a convex region enclosed by hyperplanes.
- The objective function is quadratic (also convex).
- This problem thus has a unique global solution.

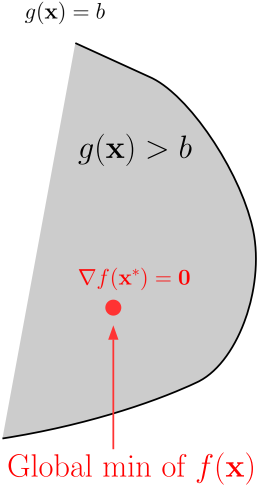
Review of multivariable calculus

Consider the following **constrained optimization** problem

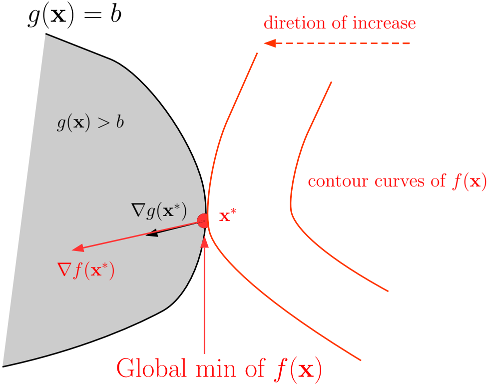
$$\min f(\mathbf{x}) \qquad \text{subject to} \quad g(\mathbf{x}) \geq b$$

There are two cases regarding where the global minimum of $f(\mathbf{x})$ is attained:

(1) At an *interior* point $\mathbf{x}^*$ (i.e., $g(\mathbf{x}^*) > b$). In this case $\mathbf{x}^*$ is just a critical point of $f(\mathbf{x})$.


$g(\mathbf{x}) = b$
$g(\mathbf{x}) > b$
$\nabla f(\mathbf{x}^*) = \mathbf{0}$
Global min of $f(\mathbf{x})$

(2) At a *boundary* point $\mathbf{x}^*$ (i.e., $g(\mathbf{x}^*) = b$). In this case, there exists a constant $\lambda > 0$ such that $\nabla f(\mathbf{x}^*) = \lambda \cdot \nabla g(\mathbf{x}^*)$.



The above two cases are unified by the **method of Lagrange multipliers**:

- Form the Lagrange function

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) - \lambda(g(\mathbf{x}) - b)$$

- Find all critical points by solving

$$\begin{aligned} \nabla_{\mathbf{x}} L = \mathbf{0} : \quad \nabla f(\mathbf{x}) &= \lambda \nabla g(\mathbf{x}) \\ \lambda(g(\mathbf{x}) - b) &= 0 \\ \lambda &\geq 0 \\ g(\mathbf{x}) &\geq b \end{aligned}$$

Remark. The solutions give all candidate points for the global minimizer (one needs to compare them and pick the best one).

Remarks:

- The above equations are called Karush-Kuhn-Tucker (KKT) conditions.

- When there are multiple inequality constraints

$$\min f(\mathbf{x}) \qquad \text{subject to} \quad g_1(\mathbf{x}) \geq b_1, \ldots, g_k(\mathbf{x}) \geq b_k$$

  the method works very similarly:

– Form the Lagrange function

$$L(\mathbf{x}, \lambda_1, \dots, \lambda_k) = f(\mathbf{x}) - \lambda_1(g_1(\mathbf{x}) - b_1) - \cdots - \lambda_k(g_k(\mathbf{x}) - b_k)$$

– Find all critical points by solving

$$\begin{aligned} \nabla_{\mathbf{x}} L = \mathbf{0}: \quad \frac{\partial L}{\partial x_1} = 0, \dots, \frac{\partial L}{\partial x_n} = 0 \\ \lambda_1(g_1(\mathbf{x}) - b_1) = 0, \dots, \lambda_k(g_k(\mathbf{x}) - b_k) = 0 \\ \lambda_1 \geq 0, \dots, \lambda_k \geq 0 \\ g_1(\mathbf{x}) \geq b_1, \dots, g_k(\mathbf{x}) \geq b_k \end{aligned}$$

and compare them to pick the best one.

Lagrange method applied to binary SVM

- The Lagrange function is

$$L(\mathbf{w}, b, \lambda_1, \ldots, \lambda_n) = \frac{1}{2}\|\mathbf{w}\|_2^2 - \sum_{i=1}^{n} \lambda_i (y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1)$$

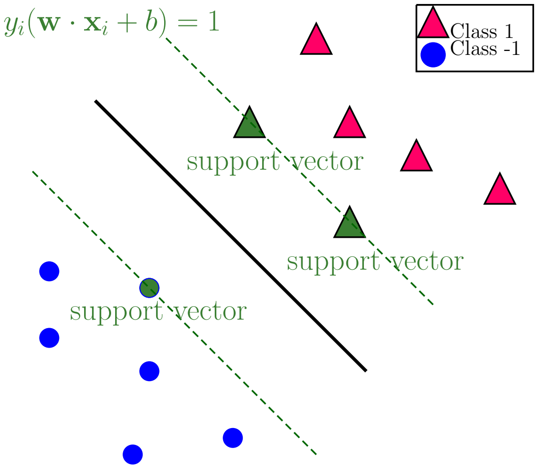
- The KKT conditions are

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum \lambda_i y_i \mathbf{x}_i = 0, \quad \frac{\partial L}{\partial b} = \sum \lambda_i y_i = 0 \\ \lambda_i (y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1) = 0,\ \forall i \\ \lambda_i \ge 0,\ \forall i \\ y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \ge 1,\ \forall i \end{aligned}$$

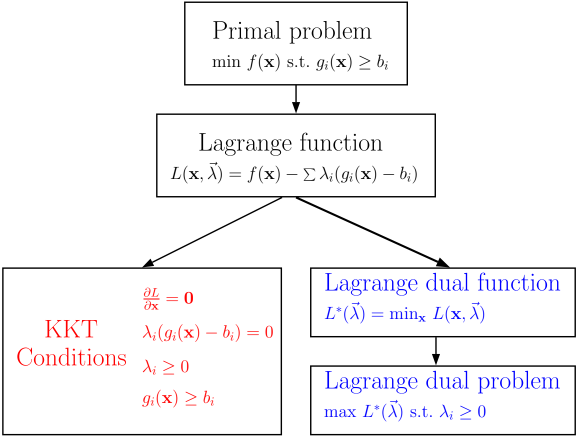
Comments:

- The first condition implies that the optimal $\mathbf{w}$ is a linear combination of the training vectors: $\mathbf{w} = \sum \lambda_i y_i \mathbf{x}_i$.
- The second line implies that whenever $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) > 1$ (i.e., $\mathbf{x}_i$ is an interior point), we have $\lambda_i = 0$. Therefore, the optimal $\mathbf{w}$ is only a linear combination of the support vectors (i.e., those satisfying $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) = 1$).
- The optimal b can be found from any support vector $\mathbf{x}_i$ (with $\lambda_i > 0$):

$$b = \frac{1}{y_i} - \mathbf{w} \cdot \mathbf{x}_i = y_i - \mathbf{w} \cdot \mathbf{x}_i$$


$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) = 1$
Class 1
Class -1
support vector
support vector
support vector

The Lagrange dual problem



For binary SVM, the **primal** problem is

$$\min_{\mathbf{w},b} \ \frac{1}{2}\|\mathbf{w}\|_2^2 \qquad \text{subject to} \quad y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 \text{ for all } i.$$

The associated Lagrange function is

$$L(\mathbf{w}, b, \lambda_1, \ldots, \lambda_n) = \frac{1}{2}\|\mathbf{w}\|_2^2 - \sum_{i=1}^n \lambda_i (y_i(\mathbf{w}\cdot\mathbf{x}_i + b) - 1)$$

By definition, the **Lagrange dual function** is

$$L^*(\lambda_1, \ldots, \lambda_n) = \min_{\mathbf{w},b} L(\mathbf{w}, b, \lambda_1, \ldots, \lambda_n), \quad \lambda_1 \geq 0, \ldots, \lambda_n \geq 0$$

To find the minimum of L over $\mathbf{w}, b$ (while fixing all λ_i), we set the gradient vector to zero to obtain

$$\mathbf{w} = \sum \lambda_i y_i \mathbf{x}_i, \quad \sum \lambda_i y_i = 0$$

Plugging the formula for $\mathbf{w}$ into L gives that

$$\begin{aligned} L^*(\lambda_1, \ldots, \lambda_n) &= \frac{1}{2} \left\| \sum_i \lambda_i y_i \mathbf{x}_i \right\|_2^2 - \sum_i \lambda_i \left(y_i \left(\left(\sum_j \lambda_j y_j \mathbf{x}_j \right) \cdot \mathbf{x}_i + b \right) - 1 \right) \\ &= \sum_i \lambda_i - \frac{1}{2} \sum_i \sum_j \lambda_i \lambda_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \end{aligned}$$

with the constraints

$$\lambda_i \geq 0, \quad \sum \lambda_i y_i = 0$$

We have obtained the Lagrange dual problem for binary SVM (without outliers)

$$\max_{\lambda_1,\ldots,\lambda_n} \sum \lambda_i - \frac{1}{2} \sum_i \sum_j \lambda_i \lambda_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$$
$$\text{subject to} \quad \lambda_i \geq 0 \text{ and } \sum \lambda_i y_i = 0$$

Remark:

- The primal and dual problems are equivalent.
- The dual problem only depends on the number of samples (one λ per $\mathbf{x}_i$), not on their dimension.
- The dual problem can also be solved by quadratic programming.
- Samples appear only through their dot products $\mathbf{x}_i \cdot \mathbf{x}_j$, an observation to be exploited for designing nonlinear SVM classifiers.

Quadratic programming in Matlab

'**quadprog**' - Quadratic programming function (requires Optimization toolbox).

x = quadprog(H,f,A,b) attempts to solve the quadratic programming problem:

$$\min_{\mathbf{x}} \frac{1}{2} \cdot \mathbf{x}^T \cdot \mathbf{H} \cdot \mathbf{x} + \mathbf{f}^T \cdot \mathbf{x} \qquad \text{subject to :} \quad \mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}$$

x = quadprog(H,f,A,b,Aeq,beq) solves the problem above while additionally satisfying the equality constraints **Aeq** $\cdot$ **x = beq**.

Binary SVM via quadratic programming

In order to use the Matlab *quadprog* function, we first need to transfrom the previous formulation to the standard form

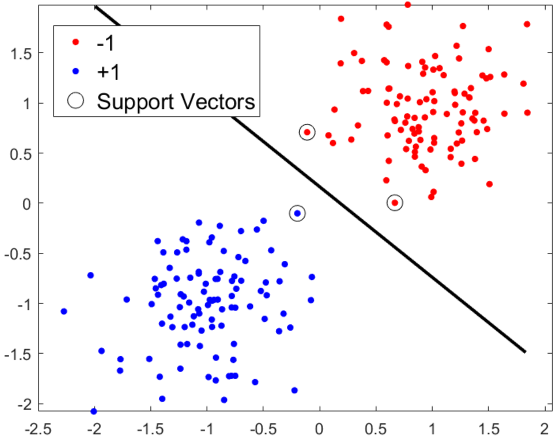
$$\min_{\lambda_1,\ldots,\lambda_n} \ \frac{1}{2}\sum_{i,j}\lambda_i\lambda_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j - \sum \lambda_i$$

$$\text{subject to} \quad -\lambda_i \le 0 \text{ and } \sum \lambda_i y_i = 0$$

and then matrice/vectorize it:

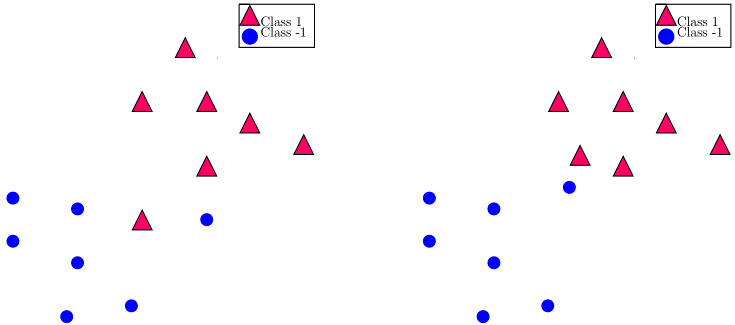
$$\min_{\vec{\lambda}} \ \frac{1}{2}\vec{\lambda}^T \mathbf{H}\vec{\lambda} + \mathbf{f}^T\vec{\lambda}$$

$$\text{subject to} \quad \mathbf{A}\vec{\lambda} \le \mathbf{b} \text{ and } \mathbf{A}_{\text{eq}}\vec{\lambda} = \mathbf{b}_{\text{eq}}$$


-1
+1
Support Vectors
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

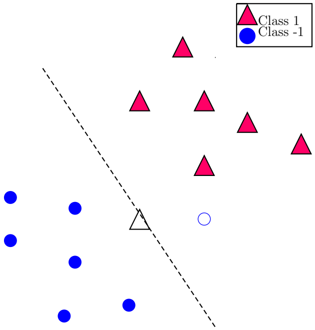
$\boxed{\text{Binary SVM: Linearly separable with outliers}}$

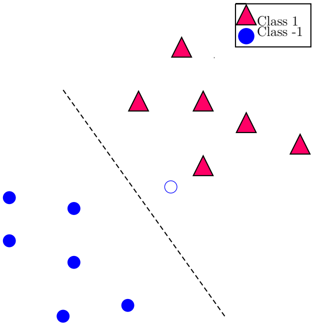
What is the optimal separating line?



(Left: not linearly separable; right: linearly separable but quite weakly)

What is the optimal separating line?





(Both data sets are more linearly separated if several points are ignored).

Introducing slack variables

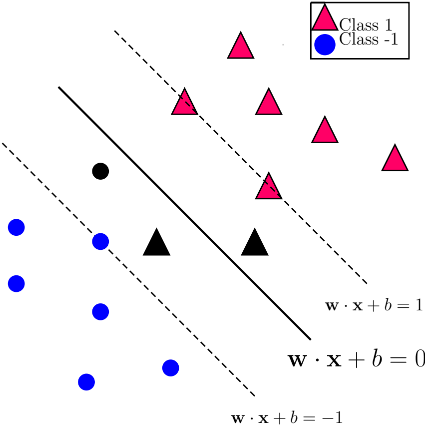
To find a linear boundary with a large margin, we must allow violations of the constraint $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$.

That is, we allow a few points to fall within the margin. They will satisfy

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) < 1$$

There are two cases:

- $y_i = +1$: $\mathbf{w} \cdot \mathbf{x}_i + b < 1$;
- $y_i = -1$: $\mathbf{w} \cdot \mathbf{x}_i + b > -1$.



Formally, we introduce slack variables $\xi_1, \ldots, \xi_n \geq 0$ (one for each sample) to allow for *exceptions*:

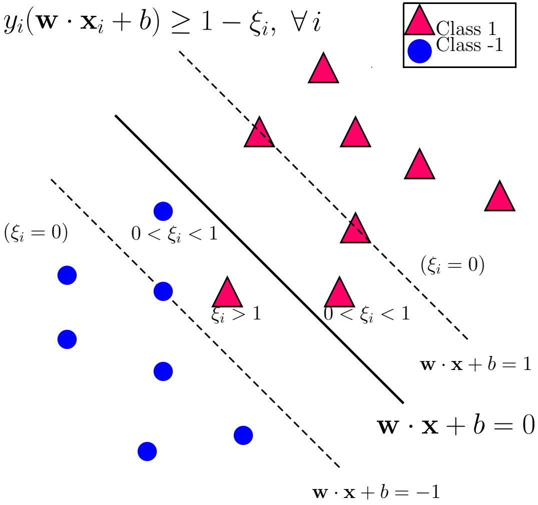
$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i, \quad \forall\ i$$

where $\xi_i = 0$ for the points in ideal locations, and $\xi_i > 0$ for the violations (chosen precisely so that the equality will hold true):

- $0 < \xi_i < 1$: Still on correct side of hyperplane but within the margin
- $\xi_i > 1$: Already on wrong side of hyperplane

We say that such an SVM has a **soft margin** to distinguish from the previous *hard margin*.

Support Vector Machine (SVM)



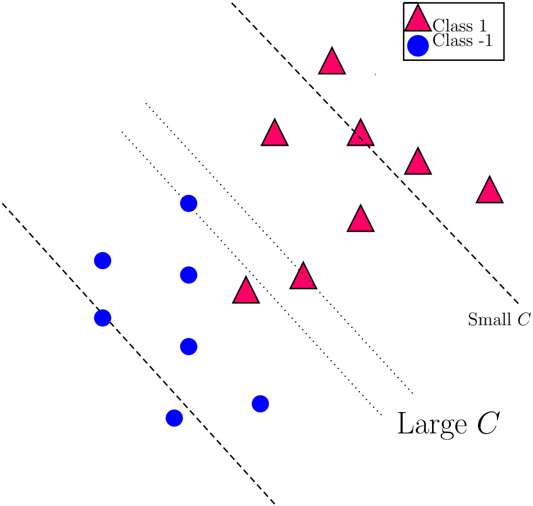
Because we want most of the points to be in ideal locations, we incorporate the slack variables into the objective function as follows

$$\min_{\mathbf{w},b,\vec{\xi}} \frac{1}{2}\|\mathbf{w}\|_2^2 + C \cdot \underbrace{\sum_i 1_{\xi_i > 0}}_{\#\text{ exceptions}}$$

where $C > 0$ is a regularization constant:

- Larger C leads to fewer exceptions (smaller margin, possible overfitting).
- Smaller C tolerates more exceptions (larger margin, possible underfitting).

Clearly, there must be a tradeoff between margin and #exceptions when selecting the optimal C (often based on cross validation).


Class 1
Class -1
Small C
Large C

ℓ_1 relaxation of the penalty term

The discrete nature of the penalty term on previous slide, $\sum_i 1_{\xi_i>0} = \|\vec{\xi}\|_0$, makes the problem intractable.

A common strategy is to replace the ℓ_0 penalty with a ℓ_1 penalty: $\sum_i \xi_i = \|\vec{\xi}\|_1$, resulting in the following full problem

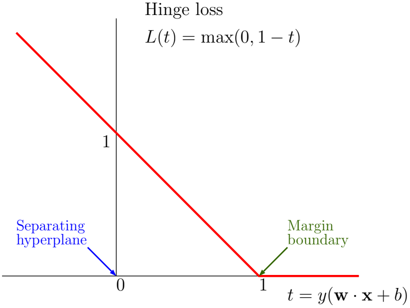
$$\min_{\mathbf{w},b,\vec{\xi}} \frac{1}{2}\|\mathbf{w}\|_2^2 + C \cdot \sum_i \xi_i$$

$$\text{subject to} \quad y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 - \xi_i \text{ and } \xi_i \geq 0 \text{ for all } i.$$

This is also a quadratic program with linear inequality constraints (just more variables): $y_i(\mathbf{w}\cdot\mathbf{x}_i + b) + \xi_i \geq 1$.

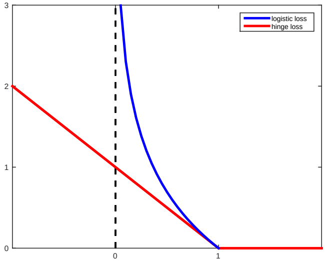
Remark. The problem may be rewritten as an unconstrained problem

$$\min_{\mathbf{w},b} \quad \underbrace{\frac{1}{2}\|\mathbf{w}\|_2^2}_{\text{regularization}} + C \cdot \underbrace{\sum_{i=1}^n \max(0, 1 - y_i(\mathbf{w}\cdot\mathbf{x}_i + b))}_{\text{hinge loss}}$$



Remark. There is a close connection to ℓ_2-regularized logistic regression:

$$\min_{\vec{\theta}} \quad \underbrace{C\|\vec{\theta}\|_2^2}_{\text{regularization}} \quad \underbrace{-\sum_{i=1}^{n} y_i \log p(\mathbf{x}_i;\vec{\theta}) + (1-y_i)\log(1-p(\mathbf{x}_i;\vec{\theta}))}_{\text{logistic loss}}$$



The Lagrange dual problem

The associated Lagrange function is

$$L(\mathbf{w}, b, \vec{\xi}, \vec{\lambda}, \vec{\mu}) = \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\lambda_i(y_i(\mathbf{w}\cdot\mathbf{x}_i + b) - 1 + \xi_i) - \sum_{i=1}^{n}\mu_i\xi_i$$

To find the dual problem we need to fix $\vec{\lambda}, \vec{\mu}$ and maximize over $\mathbf{w}, b, \vec{\xi}$:

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum \lambda_i y_i \mathbf{x}_i = 0$$
$$\frac{\partial L}{\partial b} = \sum \lambda_i y_i = 0$$
$$\frac{\partial L}{\partial \xi_i} = C - \lambda_i - \mu_i = 0, \quad \forall\, i$$

This yields the Lagrange dual function

$$L^*(\vec{\lambda}, \vec{\mu}) = \sum \lambda_i - \frac{1}{2} \sum \lambda_i \lambda_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j, \qquad \text{where}$$
$$\lambda_i \ge 0,\ \mu_i \ge 0,\ \lambda_i + \mu_i = C,\ \text{and } \sum \lambda_i y_i = 0.$$

The dual problem would be to maximize L^* over $\vec{\lambda}, \vec{\mu}$ subject to the constraints.

Since L^* is constant with respect to the μ_i, we can eliminate them to obtain a reduced dual problem:

$$\max_{\lambda_1,\ldots,\lambda_n} \sum \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$$
$$\text{subject to} \quad \underbrace{0 \le \lambda_i \le C}_{\text{box constraints}} \text{ and } \sum \lambda_i y_i = 0.$$

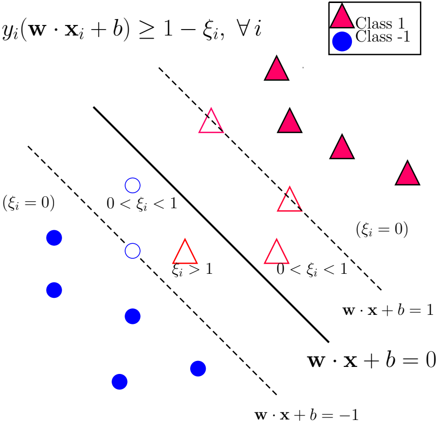
What about the KKT conditions?

The KKT conditions are the following

$$\begin{aligned}
\mathbf{w} = \sum \lambda_i y_i \mathbf{x}_i, \quad \sum \lambda_i y_i = 0, \quad \lambda_i + \mu_i = C \\
\lambda_i(y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 + \xi_i) = 0, \quad \mu_i \xi_i = 0 \\
\lambda_i \geq 0, \quad \mu_i \geq 0 \\
y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0
\end{aligned}$$

We see that

- The optimal $\mathbf{w}$ has the same formula: $\mathbf{w} = \sum \lambda_i y_i \mathbf{x}_i$.
- Any point with $\lambda_i > 0$ and correspondingly $y_i(\mathbf{w} \cdot \mathbf{x} + b) = 1 - \xi_i$ is a support vector (not just those on the margin boundary $\mathbf{w} \cdot \mathbf{x} + b = \pm 1$).
- To find b, choose any support vector $\mathbf{x}_i$ with $0 < \lambda_i < C$ (which implies that $\mu_i > 0$ and $\xi_i = 0$), and use the formula $b = \frac{1}{y_i} - \mathbf{w} \cdot \mathbf{x}_i$.


$y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 - \xi_i,\ \forall\, i$
Class 1
Class -1
$(\xi_i = 0)$
$0 < \xi_i < 1$
$\xi_i > 1$
$0 < \xi_i < 1$
$(\xi_i = 0)$
$\mathbf{w}\cdot\mathbf{x} + b = 1$
$\mathbf{w}\cdot\mathbf{x} + b = 0$
$\mathbf{w}\cdot\mathbf{x} + b = -1$

Binary SVM via quadratic programming

Again, we need to transform the previous formulation to the standard form

$$\min_{\lambda_1,\ldots,\lambda_n} \ \frac{1}{2}\sum_{i,j}\lambda_i\lambda_j y_i y_j \mathbf{x}_i\cdot\mathbf{x}_j - \sum \lambda_i$$

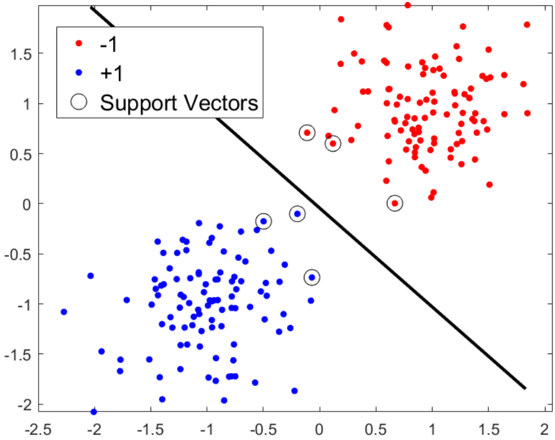
$$\text{subject to} \quad -\lambda_i \le 0,\ \lambda_i \le C,\ \text{and} \ \sum \lambda_i y_i = 0$$

and then matrice/vectorize it:

$$\min_{\vec{\lambda}} \ \frac{1}{2}\vec{\lambda}^T\mathbf{H}\vec{\lambda} + \mathbf{f}^T\vec{\lambda}$$

$$\text{subject to} \quad \mathbf{A}\vec{\lambda} \le \mathbf{b} \text{ and } \mathbf{A}_{\text{eq}}\vec{\lambda} = \mathbf{b}_{\text{eq}}$$

Note. Both $\mathbf{A}, \mathbf{b}$ are twice as tall as before (the other variables remain the same).


-1
+1
Support Vectors
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

$\boxed{\text{Binary SVM: Nonlinearly separable, with outliers}}$

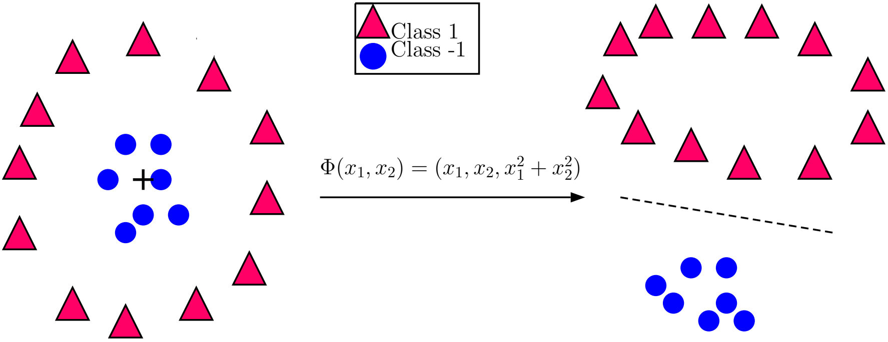
Feature map

When the classes are nonlinearly separable, a transformation of the data (both training and test) is often used (so that the training classes in the new space becomes linearly separable):

$$\Phi: \mathbf{x}_i \in \mathbb{R}^d \mapsto \Phi(\mathbf{x}_i) \in \mathbb{R}^\ell$$

where often $\ell \gg d$, and sometimes $\ell = \infty$.

- The function Φ is called a *feature map*,
- The target space $\mathbb{R}^\ell$ is called a *feature space*, and
- The images $\Phi(\mathbf{x}_i)$ are called *feature vectors*.


Class 1
Class -1
$\Phi(x_1, x_2) = (x_1, x_2, x_1^2 + x_2^2)$

The kernel trick

In principle, once we find a good feature map $\Phi : \mathbb{R}^d \to \mathbb{R}^\ell$ we just need to work in the new space to build a binary SVM model and classify test data (after being transformed in the same way):

- SVM in feature space

$$\min_{\mathbf{w},b,\vec{\xi}} \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum \xi_i \quad \text{subject to}$$

$$y_i(\mathbf{w} \cdot \Phi(\mathbf{x}_i) + b) \geq 1 - \xi_i, \text{ and } \xi_i \geq 0 \text{ for all } i.$$

- Decision rule for test data $\mathbf{x}$

$$y = \operatorname{sgn}(\mathbf{w} \cdot \Phi(\mathbf{x}) + b)$$

However, in many cases the feature space is very high dimensional, making computing intensive.

We can apply a **kernel trick** thanks to the Lagrange dual formulation of SVM:

$$\max_{\lambda_1,\ldots,\lambda_n} \sum \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \underbrace{\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)}_{:=\kappa(\mathbf{x}_i,\mathbf{x}_j)}$$

$$\text{subject to} \quad 0 \le \lambda_i \le C \text{ and } \sum \lambda_i y_i = 0$$

That is to specify only the dot product function κ of the feature space, called a **kernel function** and avoid explicitly using the feature map Φ.

In the toy example,

$$\Phi(\mathbf{x}) = (\mathbf{x}, \|\mathbf{x}\|_2^2),$$

and

$$\kappa(\mathbf{x}, \tilde{\mathbf{x}}) = \mathbf{x} \cdot \tilde{\mathbf{x}} + \|\mathbf{x}\|_2^2 \cdot \|\tilde{\mathbf{x}}\|_2^2.$$

Can the decision rule also avoid the explicit use of Φ?

$$y = \operatorname{sgn}(\mathbf{w} \cdot \Phi(\mathbf{x}) + b)$$

The answer is yes, because $\mathbf{w}$ is a linear combination of the support vectors in the feature space:

$$\mathbf{w} = \sum \lambda_i y_i \Phi(\mathbf{x}_i)$$

and so is b (for any support vector $\Phi(\mathbf{x}_{i_0})$ with $0 < \lambda_{i_0} < C$):

$$b = y_{i_0} - \mathbf{w} \cdot \Phi(\mathbf{x}_{i_0})$$

Consequently,

$$y = \operatorname{sgn}\left(\sum \lambda_i y_i \kappa(\mathbf{x}_i, \mathbf{x}) + b\right),$$

where

$$b = y_{i_0} - \sum \lambda_i y_i \kappa(\mathbf{x}_i, \mathbf{x}_{i_0})$$

Steps of kernel SVM

- Pick a kernel function κ (corresponding to some feature map Φ)
- Solve the following quadratic program

$$\max_{\lambda_1,\ldots,\lambda_n} \sum_i \lambda_i - \frac{1}{2}\sum_{i,j} \lambda_i\lambda_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j)$$

$$\text{subject to} \quad 0 \le \lambda_i \le C \text{ and } \sum \lambda_i y_i = 0$$

- Classify new data $\mathbf{x}$ based on the following decision rule:

$$y = \operatorname{sgn}\left(\sum \lambda_i y_i \kappa(\mathbf{x}_i, \mathbf{x}) + b\right)$$

where b can be determined from any support vector with $0 < \lambda_i < C$.

What are popular kernel functions?

- **Linear** (= no kernel, just regular SVM)

$$\kappa(\mathbf{x}, \tilde{\mathbf{x}}) = \mathbf{x} \cdot \tilde{\mathbf{x}}$$

- **Polynomial** (of degree $p \geq 1$)

$$\kappa(\mathbf{x}, \tilde{\mathbf{x}}) = (1 + \mathbf{x} \cdot \tilde{\mathbf{x}})^p$$

- **Gaussian** (also called Radial Basis Function, or RBF)

$$\kappa(\mathbf{x}, \tilde{\mathbf{x}}) = e^{-\|\mathbf{x}-\tilde{\mathbf{x}}\|_2^2/(2\sigma^2)} = e^{-\gamma\|\mathbf{x}-\tilde{\mathbf{x}}\|_2^2}$$

- **Sigmoid** (also called Hyperbolic Tangent)

$$\kappa(\mathbf{x}, \tilde{\mathbf{x}}) = \tanh(\gamma \mathbf{x} \cdot \tilde{\mathbf{x}} + r)$$

The MATLAB 'fitcsvm' function for binary SVM

% SVM training with different kernels

SVMModel = fitcsvm(trainX, Y, 'BoxConstraint', 1, 'KernelFunction', 'linear') % both are default values

SVMModel = fitcsvm(trainX, Y, 'BoxConstraint', 1, 'KernelFunction', 'gaussian', 'KernelScale', 1) % 'KernelFunction' may be set to 'rbf'. 'KernelScale' is the sigma parameter (default = 1)

SVMModel = fitcsvm(trainX, Y, 'BoxConstraint', 1, 'KernelFunction', 'polynomial', 'PolynomialOrder', 3) % default order = 3

% SVM validation (important for parameter tuning)
CVSVMModel = crossval(SVMModel); % 10-fold by default

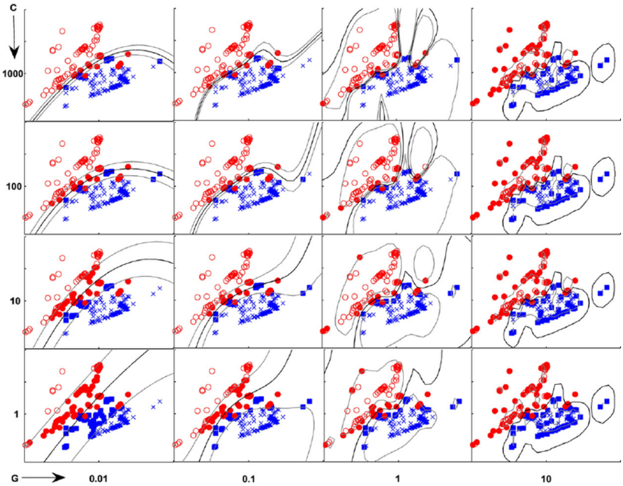
kloss = kfoldLoss(CVSVMModel);

% SVM testing

pred = predict(SVMModel, testX);

Experiments

- The polynomial kernel
  - C: margin parameter
  - p: degree of polynomial (shape parameter)
- The Gaussian kernel (see plot on next slide)
  - C: margin parameter
  - σ (or γ): smoothness parameter


C
1000
100
10
1
G
0.01
0.1
1
10

Practical issues

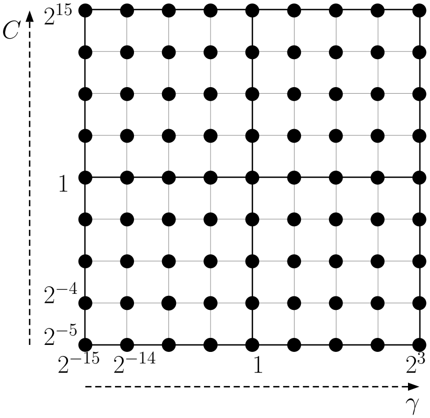
- **Scaling**: SVM often requires to rescale each dimension (pixel in our case) linearly to an interval [0,1] or [-1,1], or instead standardizes it to zero mean, unit variance.
- **High dimensional data**: Training is expensive and tends to overfit the data when using flexible kernel SVMs (such as Gaussian or polynomial). Dimensionality reduction by PCA is often needed.
- **Hyper-parameter tuning**
  - The tradeoff parameter C (for general SVM)
  - Kernel parameter: $\gamma = \frac{1}{2\sigma^2}$ (Gaussian), p (polynomial)

Parameter estimation for Gaussian-kernel SVM

GkSVM is a powerful, general-purpose kernel, but there is a practical difficulty in tuning γ and C.

Typically, it is tuned by cross validation in a grid search fashion[1]:

$$\gamma = 2^{-15}, 2^{-14}, \dots, 2^{3}, \text{ and}$$
$$C = 2^{-5}, 2^{-4}, \dots, 2^{15}$$



[1] LIBSVM: https://www.csie.ntu.edu.tw/~cjlin/libsvm/

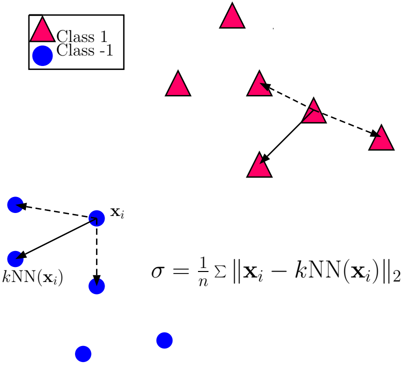
We[a] set the parameter σ in the Gaussian kernel

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = e^{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2\sigma^2}}$$

directly based on training data:

$$\sigma = \frac{1}{n} \sum \|\mathbf{x}_i - k\mathrm{NN}(\mathbf{x}_i)\|_2$$

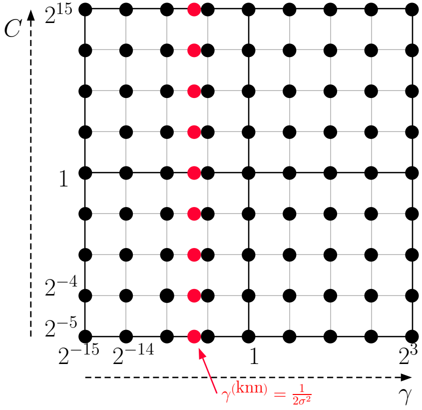
where $k\mathrm{NN}(\mathbf{x}_i)$ is the kth nearest neighbor of $\mathbf{x}_i$ within its own class.[b]

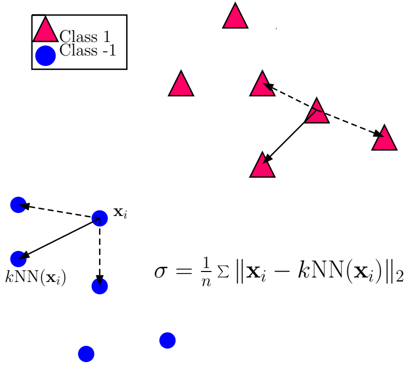


[a] G. Chen, W. Florero-Salinas, and D. Li (2017), "Simple, Fast and Accurate Hyper-parameter Tuning in Gaussian-kernel SVM", Intl. Joint Conf. on Neural Networks

[b] When n is large, we may use only a small, randomly selected subset of training data to estimate σ, leading to a stochastic algorithm.

Grid search method vs. kNN tuning (for $k = 3$)

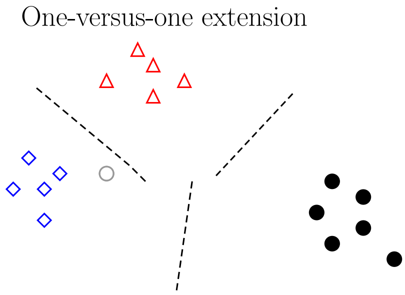


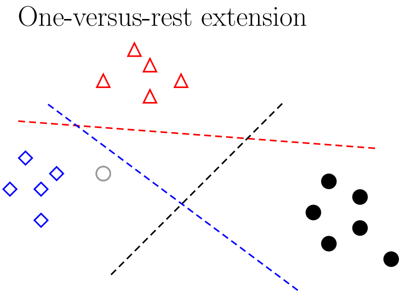


$\boxed{\text{Multiclass extensions}}$

Multiclass SVM

Like logistic regression, binary SVM can be extended to a multiclass setting in one of the following ways:





The final prediction for a test point $\mathbf{x}_0$ is determined as follows :

- **one-versus-one multiclass SVM**: the overall prediction is the most frequent label;
- **one-versus-rest multiclass SVM**:
  - For each j, fit a binary SVM model between class j (with label 1) and the rest of training data (with label -1)
  - For each binary model, record the 'score': $\mathbf{w}^{(j)} \cdot \mathbf{x}_0 + b^{(j)}$
  - The final prediction is the reference class with the highest score

$$\hat{y}_0 = \arg\max_j \ \mathbf{w}^{(j)} \cdot \mathbf{x}_0 + b^{(j)}$$

Matlab implementation for Multiclass SVM

The previously mentioned function, 'fitcsvm', is designed only for binary classification. To use multiclass SVM, you have the following options:

- Implement one-versus-one and one-versus-rest on your own (note that you have already done this for logistic regression)

- Use the Matlab function ‘fitcecoc’:

  temp = templateSVM(‘BoxConstraint’, 1, ‘KernelFunction’, ‘gaussian’, ‘KernelScale’, 1); % Gaussian kernel SVM
  temp = templateSVM(‘BoxConstraint’, 1, ‘KernelFunction’, ‘polynomial’, ‘PolynomialOrder’, 3); % polynomial kernel SVM
  Mdl = fitcecoc(trainX,Y,’Coding’,’onevsone’,‘learners’,temp);
  Mdl = fitcecoc(trainX,Y,’Coding’,’onevsall’,‘learners’,temp);

Python functions for SVM

See documentation at

- **scikit-learn** (`http://scikit-learn.org/stable/modules/svm.html`)
- **LibSVM** (`http://www.csie.ntu.edu.tw/~cjlin/libsvm/`)

Remarks:

- scikit-learn uses LibSVM to handle all computations, so the two should be the same thing.
- LibSVM contains an efficient, grid-search based Matlab implementation for SVM (including the multiclass extensions).

Summary

- Binary SVM (hard/soft margin, and kernel) and multiclass extensions
- **Advantages**:
  - Based on nice theory
  - Excellent generalization properties
  - Globally optimal solution
  - Can handle outliers and nonlinear boundary simultaneously
- **Disadvantage**: SVM might be slower than some other methods due to parameter tuning and quadratic programming

HW5

1. (a) Solve the following constrained optimization problem by hand:

$$\min_{x,y} \; y - x^2 \qquad \text{subject to} \quad 4 - x^2 - y^2 \geq 0$$

   (b) First find the Lagrange dual of the following (primal) problem

$$\min_{x} \; \frac{1}{2}x^2 \qquad \text{subject to} \quad 2x - 1 \geq 0$$

   and then verify that the two problems have the same solution.

2. Apply the one-vs-one multiclass linear SVM classifier, with different values of $C = 2^{-4}, 2^{-3}, \ldots, 2^{5}$, to the Fashion-MNIST data (after PCA 95%). Plot the test errors against the different values of C. How does it compare with the one-vs-one extension of the logistic regression classifier?

3. Implement the one-versus-one extension of the third-degree polynomial kernel SVM classifier and apply it with different values of $C = 2^{-4}, 2^{-3}, \ldots, 2^{5}$ to the Fashion-MNIST (after PCA 95%). Plot the test errors against C.

4. Implement the one-versus-one extension of the Gaussian kernel SVM classifier and apply it with different values of $C = 2^{-4}, 2^{-3}, \ldots, 2^{5}$ to the Fashion-MNIST data (after PCA 95%). To set the kernel parameter σ, use a random sample of 100 points with $k = 7$ (or a better choice). Report the value of σ you got and plot the corresponding test errors against C. How does it compare with the third-degree polynomial kernel SVM classifier?