LEC 7: Support Vector Machine (SVM)

Dr. Guangliang Chen

March 22, 2016

Outline

- 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 is SVM?

Recall that both LDA and Logistic regression are obtained from probabilistic models:

- Mixture of Gaussians $\mapsto$ LDA
- Bernoulli/multinomial $\mapsto$ Logistic regression

We also know that their decision boundaries (in the input space) are hyperplanes (thus, they are linear classifiers).

SVM is another linear classifier but works *directly* in the input space to find an '*optimal*' boundary.

Binary SVM: Linearly separable (no outliers)

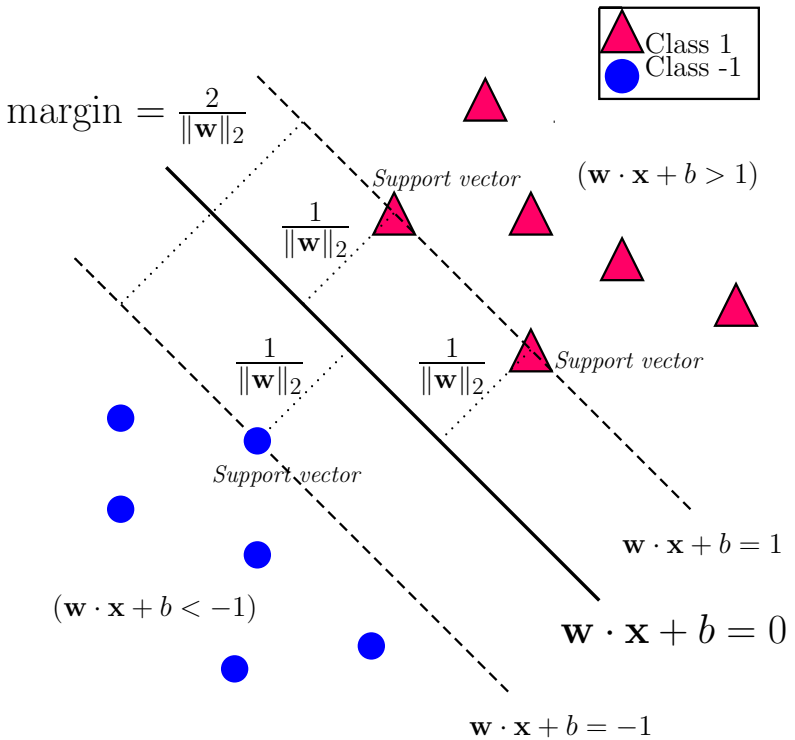
The binary SVM problem

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.

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

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

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



A more convenient formulation

The previous problem is equivalent to

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

Remarks:

- This is an optimization problem with linear, inequality constraints.
- The constraints determine a convex region enclosed by hyperplanes.
- The objective function is quadratic (also convex).
- This problem thus has a unique global solution.
- The classification rule for new points $\mathbf{x}$ is $y = \text{sgn}(\mathbf{w}\cdot\mathbf{x} + b)$.

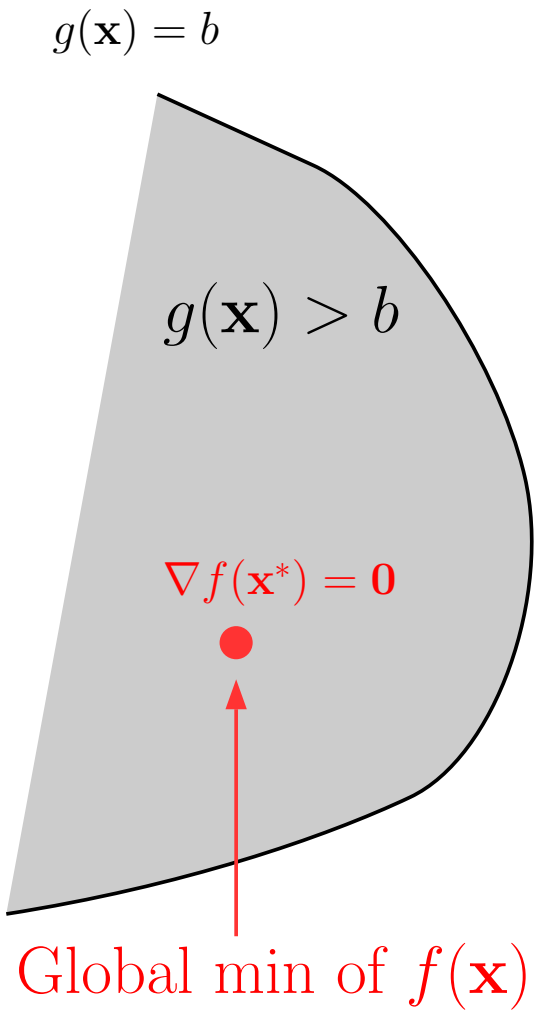
Review of multivariable calculus

Consider the following **constrained optimization** problem

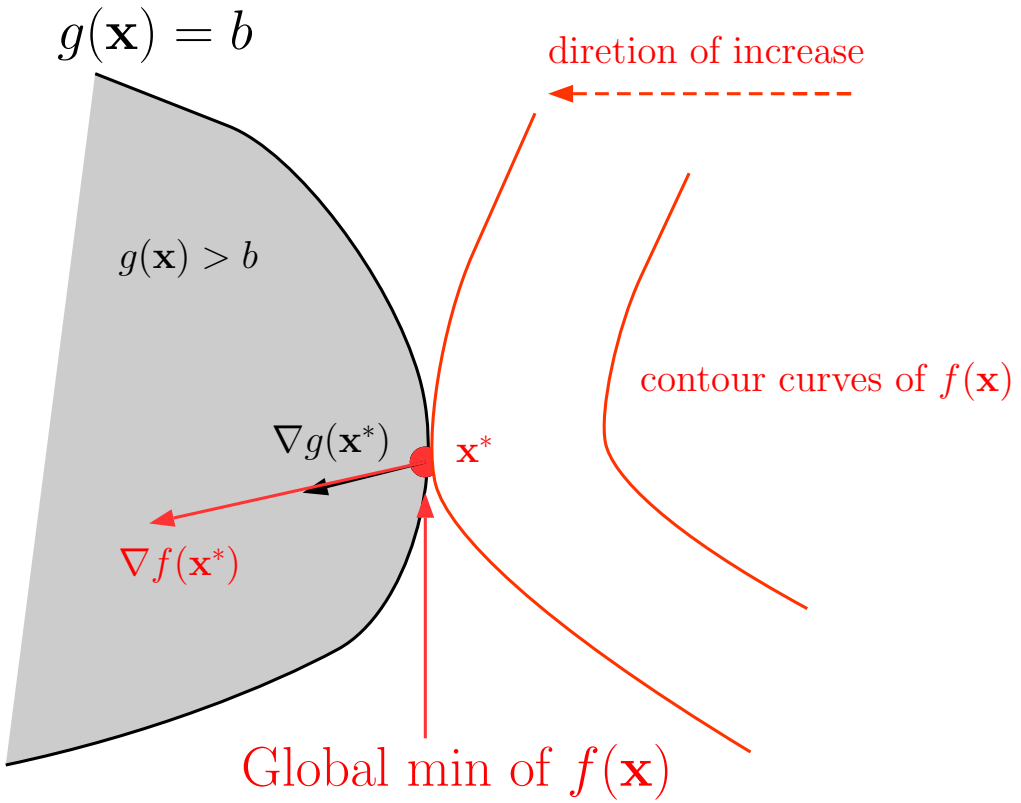
$$\min f(\mathbf{x}) \qquad \text{subject to} \quad g(\mathbf{x}) \geq b$$

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

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

– Find all critical points by solving

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

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

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

$$\lambda_i (y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1) = 0, \ \forall i$$

$$\lambda_i \geq 0, \ \forall i$$

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

Discussions

- The first condition implies that the optimal $\mathbf{w}$ is a linear combination of the training vectors:

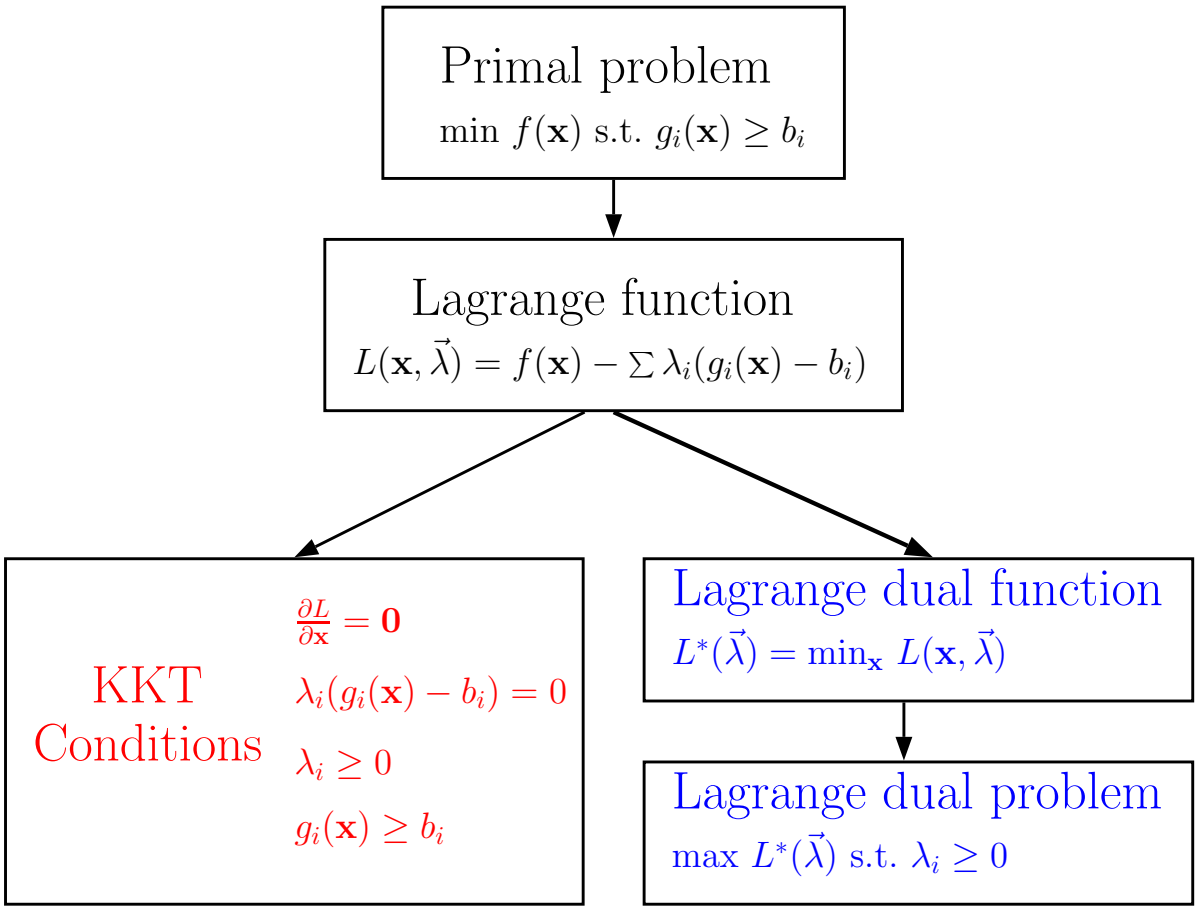
$$\mathbf{w} = \sum \lambda_i y_i \mathbf{x}_i$$

- 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 $\lambda_i > 0$):

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

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)

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

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

Remarks:

- The primal and dual problems are equivalent (at least in our case).
- 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.

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

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

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

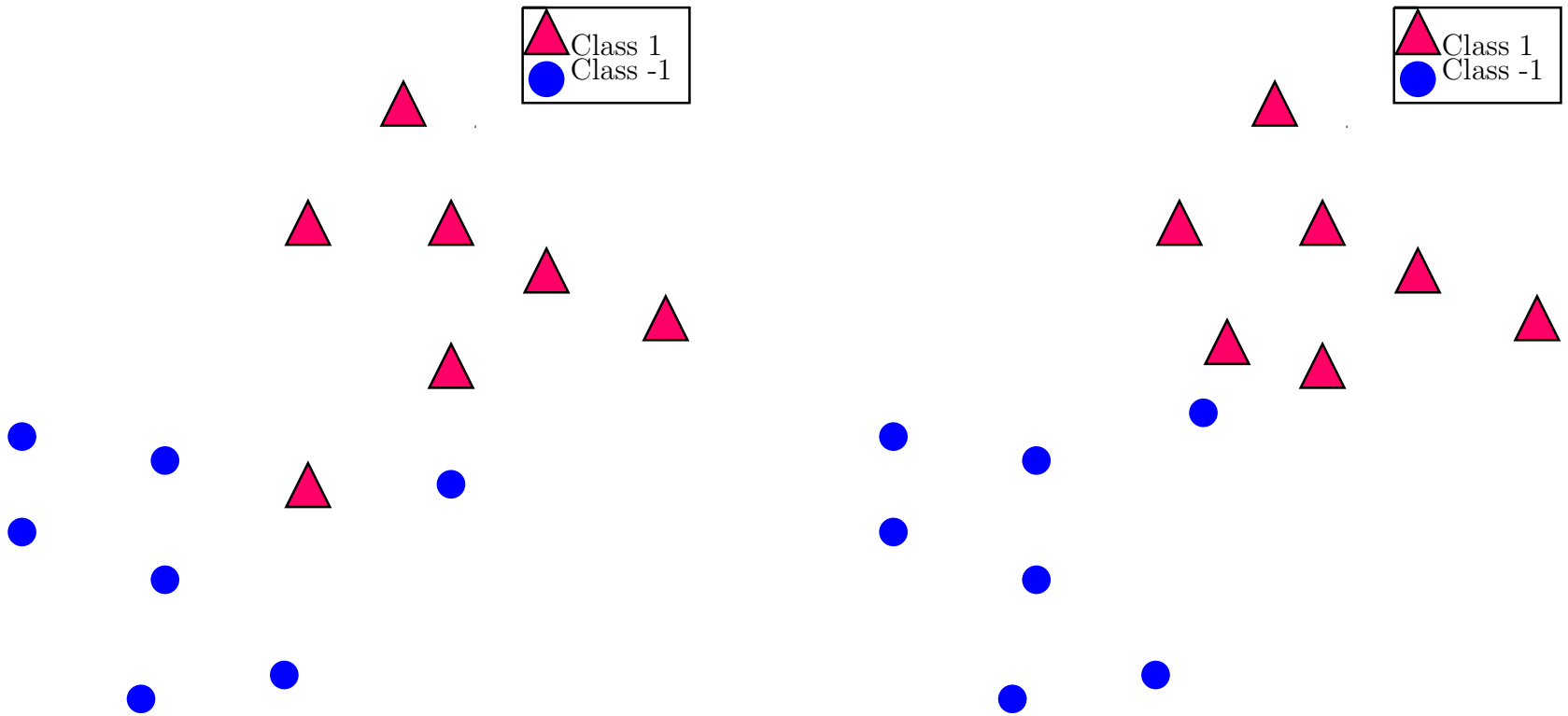
and then matrice/vectorize it:

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

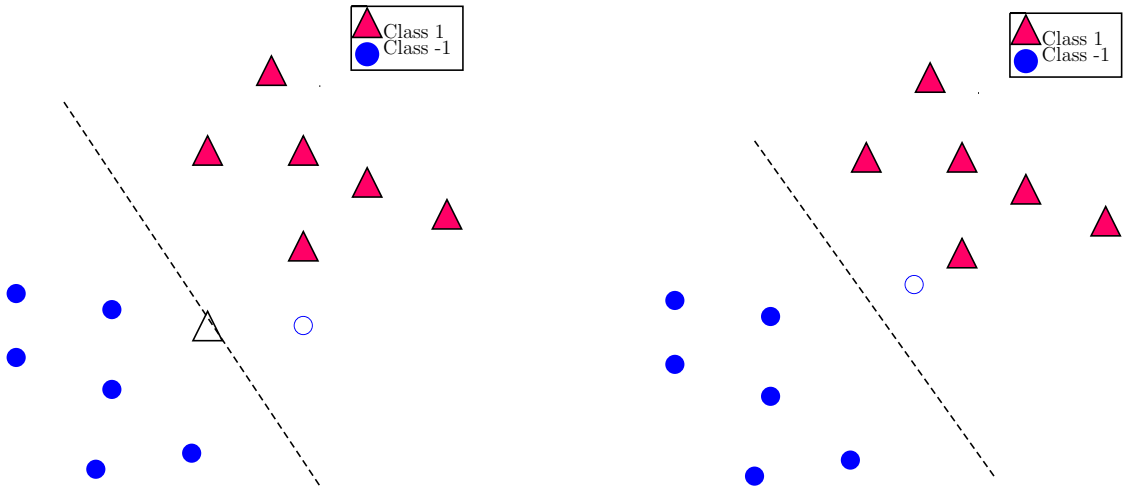
$$\text{subject to} \quad \mathbf{A}\vec{\lambda} \le \mathbf{b} \text{ and } \mathbf{A}_{\text{eq}}\vec{\lambda} = \mathbf{b}_{\text{eq}}$$

Binary SVM: Linearly separable with outliers

What is the optimal separating line?



What is the optimal separating line?



Observations:

(1) Left is linearly nonseparable and right is linearly separable (but very weakly)

(2) Both data sets contain outliers, without which the two classes would be well separated by a linear boundary.

Introducing slack variables

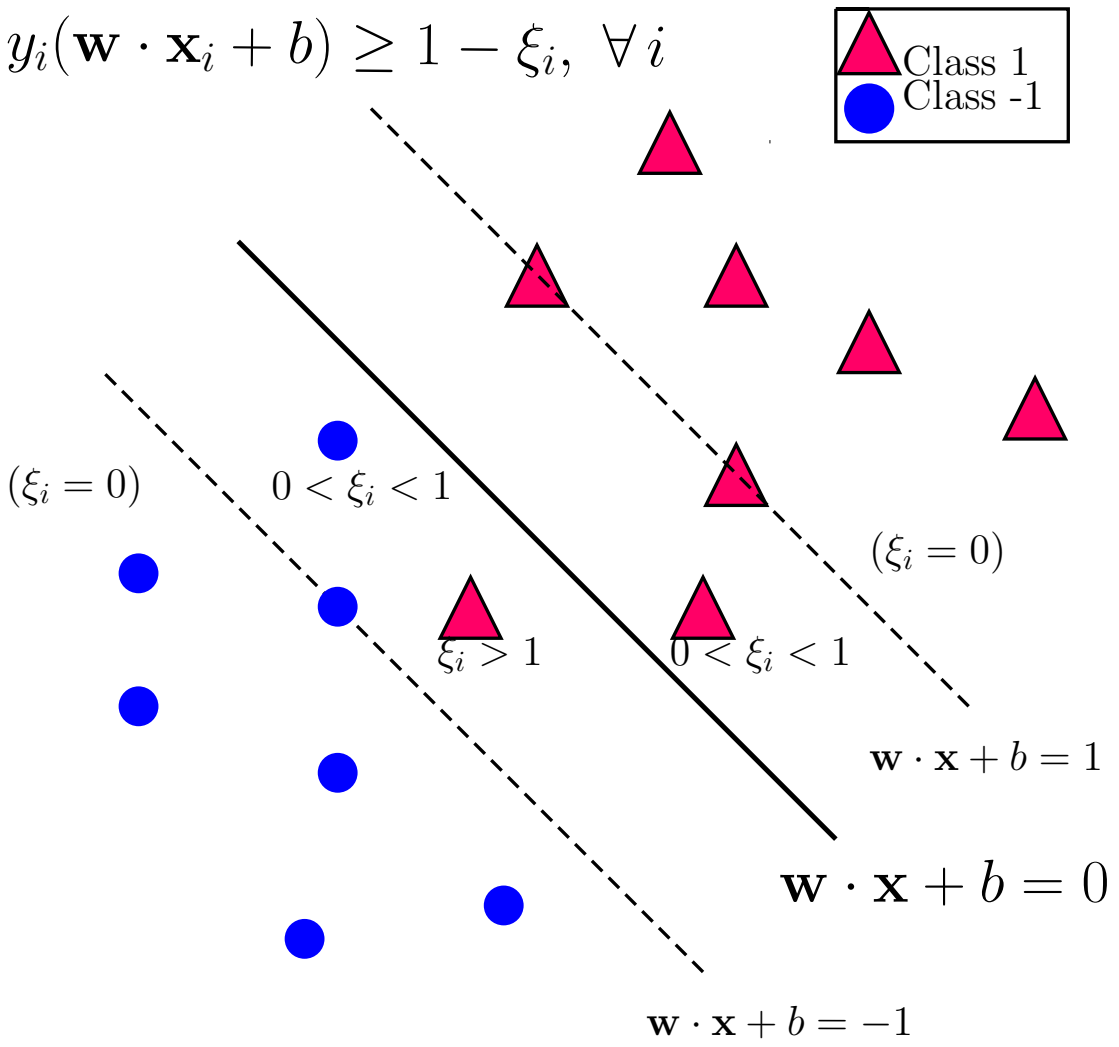
When the data contains outliers, slack variables $\xi_1, \ldots, \xi_n \geq 0$ (one for each sample) may be introduced to allow for *exceptions*:

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

where $\xi_i = 0$ for the good points (including support vectors), and $\xi_i > 0$ for the outliers (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*.


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

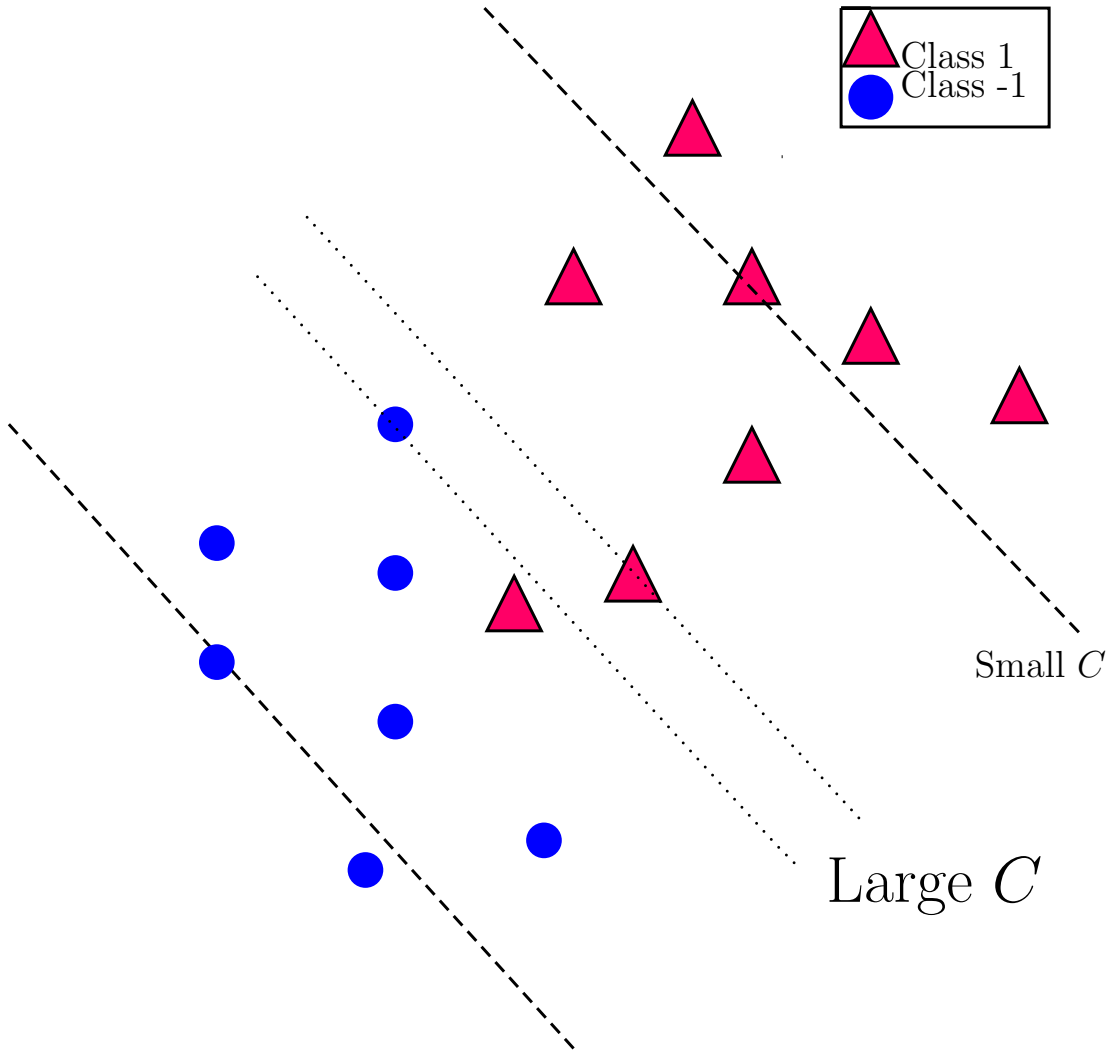
Because we want most of the points to be in ideal locations, we incorporate the slack variables into the objective function as follows

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

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

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

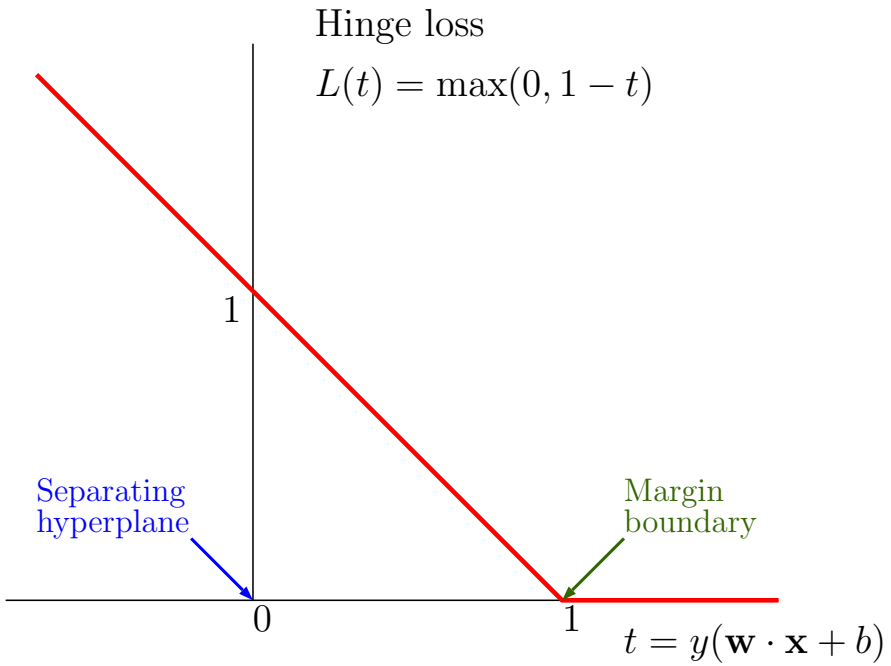
$$\text{subject to} \quad y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i \text{ and } \xi_i \geq 0 \text{ for all } i.$$

Remarks:

(1) Also a quadratic program with linear ineq. constraints (just more variables).

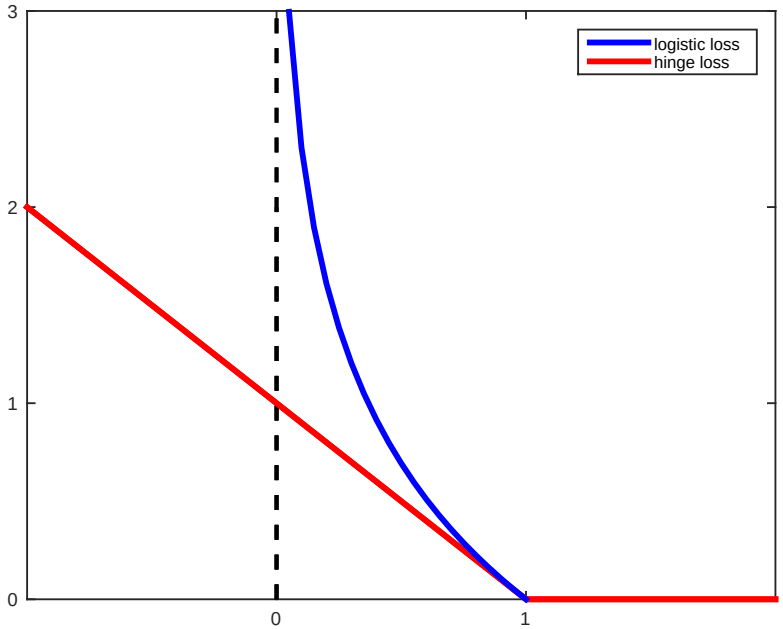
(2) The problem may be rewritten as an unconstrained optimization problem

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



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

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



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 $\mathbf{w}, b, \vec{\xi}$:

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

This yields the Lagrange dual function

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

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:

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

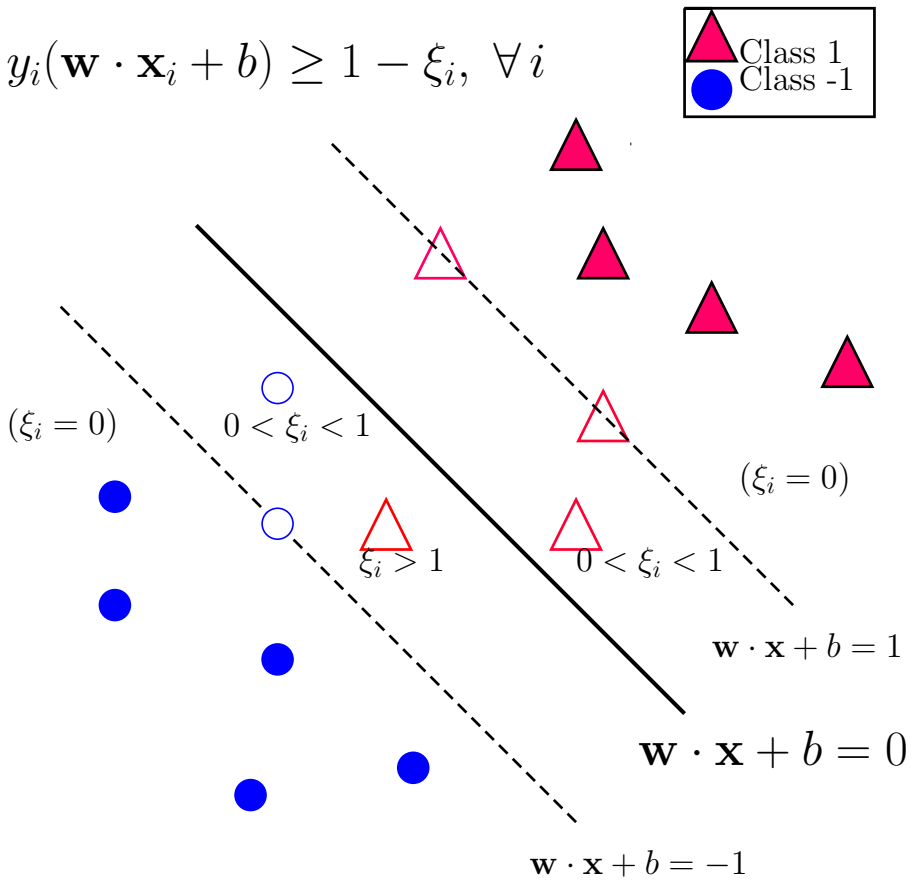
What about the KKT conditions?

The KKT conditions are the following

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

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

Binary SVM via quadratic programming

Again, we first 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).

Binary SVM: Nonlinearly separable, with outliers

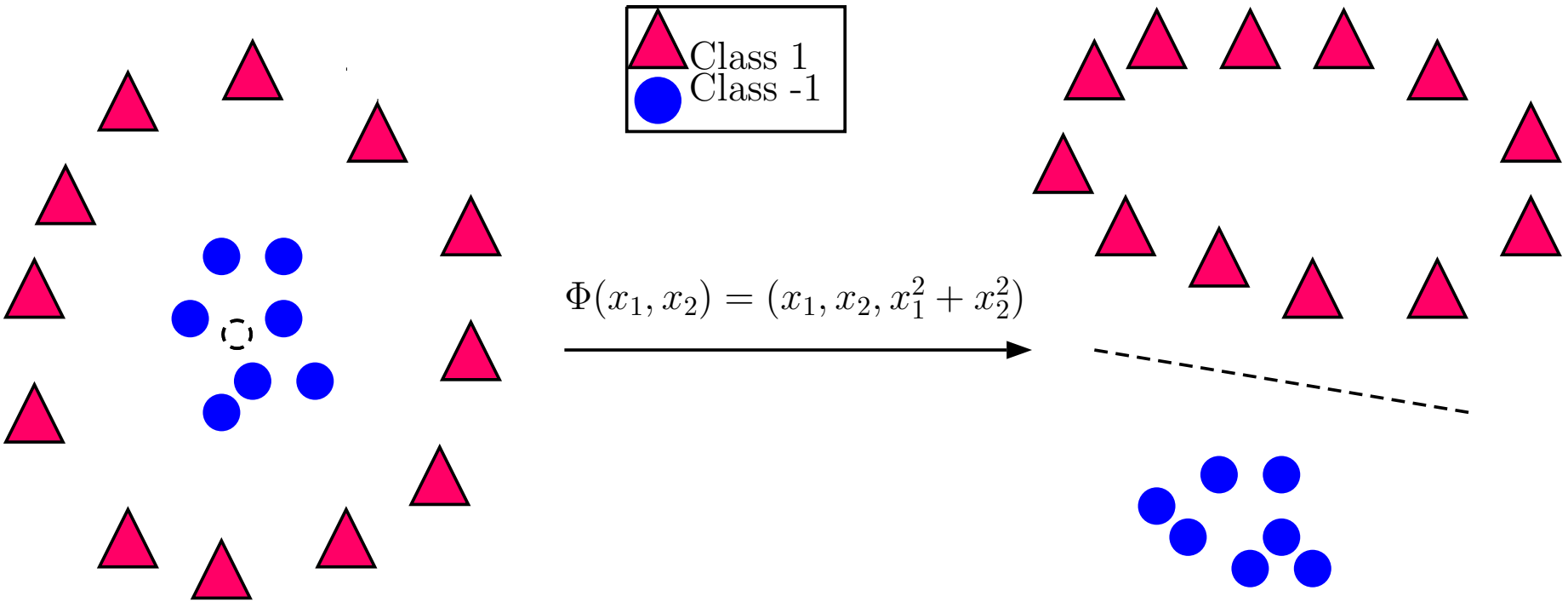
Feature map

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 $\mathbf{x}$

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

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 $0 < \lambda_{i_0} < C$):

$$b = \frac{1}{y_{i_0}} - \mathbf{w} \cdot \Phi(\mathbf{x}_{i_0})$$

Consequently,

$$y = \operatorname{sgn}\left(\sum \lambda_i y_i \kappa(\mathbf{x}_i, \mathbf{x}) + b\right), \qquad \text{where} \quad b = \frac{1}{y_{i_0}} - \sum \lambda_i y_i \kappa(\mathbf{x}_i, \mathbf{x}_{i_0})$$

Steps of kernel SVM

- Pick a kernel function κ (which corresponds 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 good kernel functions?

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

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

- **Quadratic**

$$\kappa(\mathbf{x}, \tilde{\mathbf{x}}) = (1 + \mathbf{x} \cdot \tilde{\mathbf{x}})^2$$

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

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

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

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

Matlab functions for binary svm (in full generality)

SVMSTRUCT = svmtrain(TRAINING, Y) trains a support vector machine (SVM) classifier on data taken *from two groups*. TRAINING is a numeric matrix of predictor data. Rows of TRAINING correspond to observations; columns correspond to features. Y is a column vector that contains the known class labels for TRAINING. SVMSTRUCT contains information about the trained classifier, including the support vectors.

SVMSTRUCT = svmtrain(TRAINING, Y, 'PARAM',val, ...) specifies one or more of the following name/value pairs:

- '**kernel_function**' A string or a function handle specifying the kernel function used to represent the dot product in a new space. The value can be one of the following:

  '**linear**' - Linear kernel or dot product (default). In this case, svmtrain finds the optimal separating plane in the original space.

  '**quadratic**' - Quadratic kernel

  '**polynomial**' - Polynomial kernel with default order 3. To specify another order, use the 'polyorder' argument.

  '**rbf**' - Gaussian Radial Basis Function with default scaling factor 1. To specify another scaling factor, use the 'rbf_sigma' argument.

- '**rbf_sigma**' A positive number specifying the scaling factor in the Gaussian radial basis function kernel. Default is 1.
- '**polyorder**' A positive integer specifying the order of the polynomial kernel. Default is 3.
- '**boxconstraint**' The box constraint C for the soft margin. C can be a positive numeric scalar or a vector of positive numbers with the number of elements equal to the number of rows in TRAINING.
  Default is 1.
- '**showplot**' A logical value specifying whether or not to show a plot. When the value is true, svmtrain creates a plot of the grouped data and the separating line for the classifier, when using data with 2 features (columns). Default is false.

SVMSTRUCT is a structure containing information about the trained classifier, including

- .**SupportVectors** - Matrix of data points with each row corresponding to a support vector.
- .**Alpha** - Vector of Lagrange multipliers for the support vectors. The sign is positive for support vectors belonging to the first group and negative for support vectors belonging to the second group.
- .**Bias** - Intercept of the hyperplane that separates the two groups.
- .**SupportVectorIndices** - A column vector indicating the indices of support vectors.

It will be fed directly into the classifying function **svmclassify** (see next page).

svmclassify Classify data using a support vector machine

GROUP = svmclassify(SVMSTRUCT, TEST) classifies each row in TEST using the support vector machine classifier structure SVMSTRUCT created using SVMTRAIN, and returns the predicted class level GROUP. TEST must have the same number of columns as the data used to train the classifier in SVMTRAIN. GROUP indicates the group to which each row of TEST is assigned.

GROUP = svmclassify(...,'SHOWPLOT',true) plots the test data TEST on the figure created using the SHOWPLOT option in SVMTRAIN.

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);
```

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. However, for the MNIST data the pixel values are all between 0 and 1, so it is not needed (disable this option as follows):
  SVMSTRUCT = svmtrain(TRAINING, Y, 'autoscale',false, ...)
  % default = true (in 'fitcsvm' default = no scaling)
- **High dimensional data**
- **Hyper-parameters**
  - The penalty parameter C (for general SVM)
  - Kernel parameter: $\gamma = \frac{1}{2\sigma^2}$ (Gaussian), p (polynomial)

High dimensional data

High dimensional data makes training expensive and also leads to overfitting when using flexible kernel SVMs (such as Gaussian or polynomial).

Two ways to handle this:

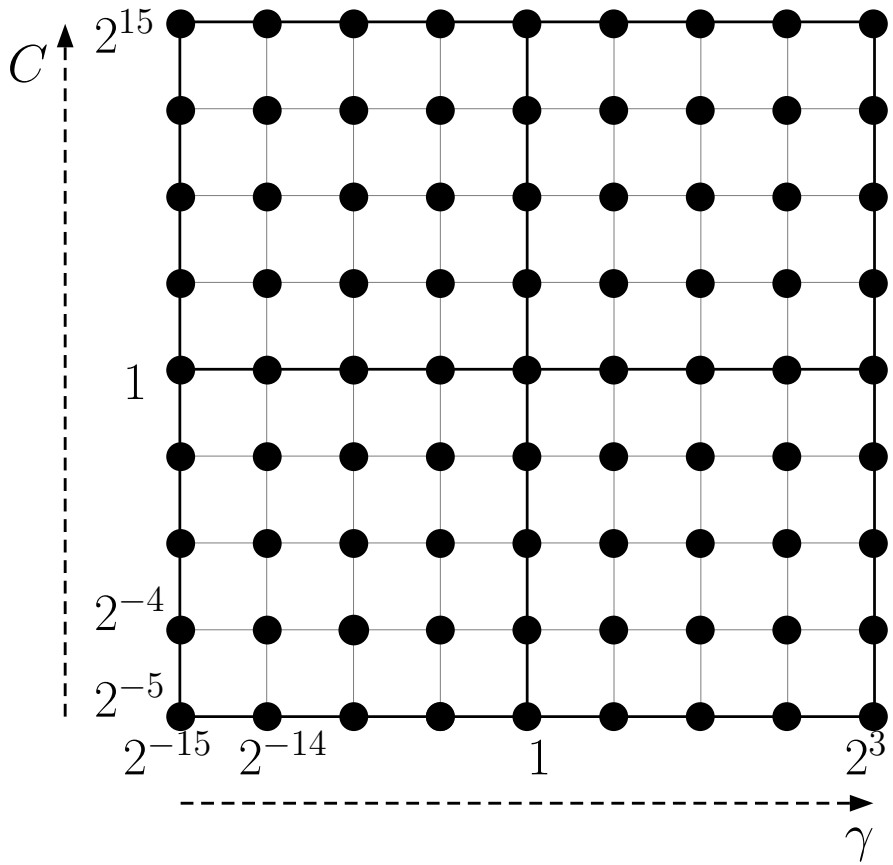
- Reduce dimensionality for kernel SVM (e.g. by PCA)
- Use linear SVM directly

Parameter estimation for Gaussian-kernel SVM

It is a powerful, general-purpose kernel, but there is a practical challenge in selecting the optimal γ (or σ) and C.

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

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



We set the parameter σ in the Gaussian kernel

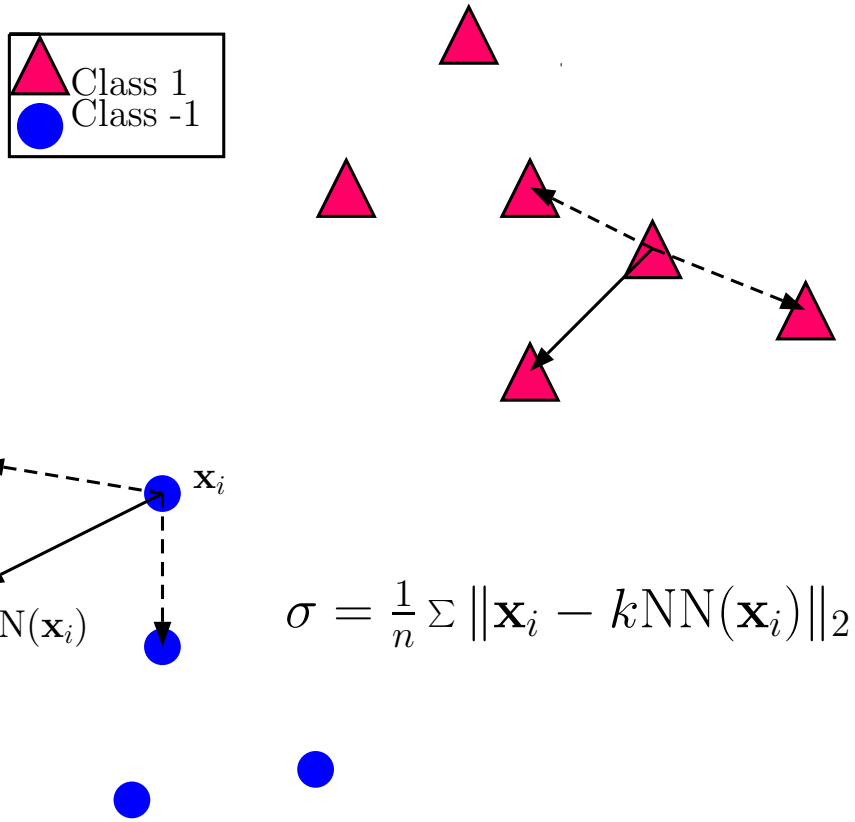
$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = e^{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2\sigma^2}}$$

directly based on training data:

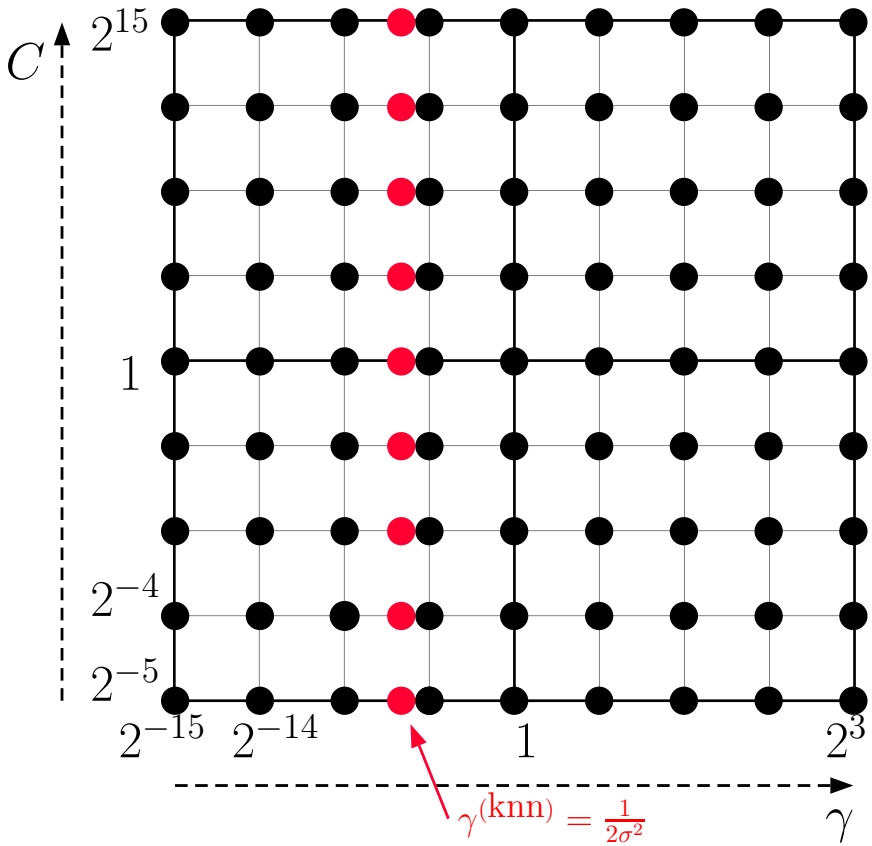
$$\sigma = \frac{1}{n} \sum_{i=1}^{n} \|\mathbf{x}_i - k\mathrm{NN}(\mathbf{x}_i)\|_2$$

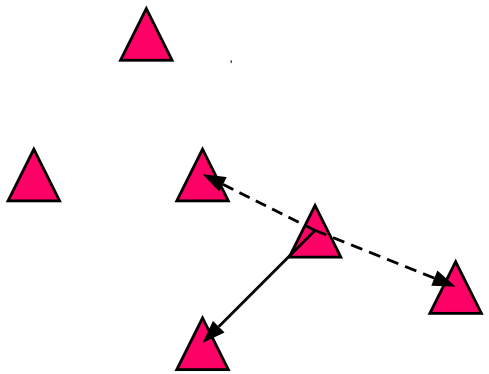
where $k\mathrm{NN}(\mathbf{x}_i)$ is the kNN of $\mathbf{x}_i$ in the training class it belongs to.

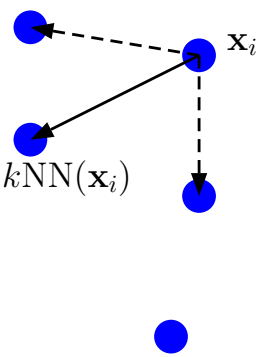
Remark. When n is large, we may use only a small, random subset of training data to estimate σ, leading to a stochastic algorithm.



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





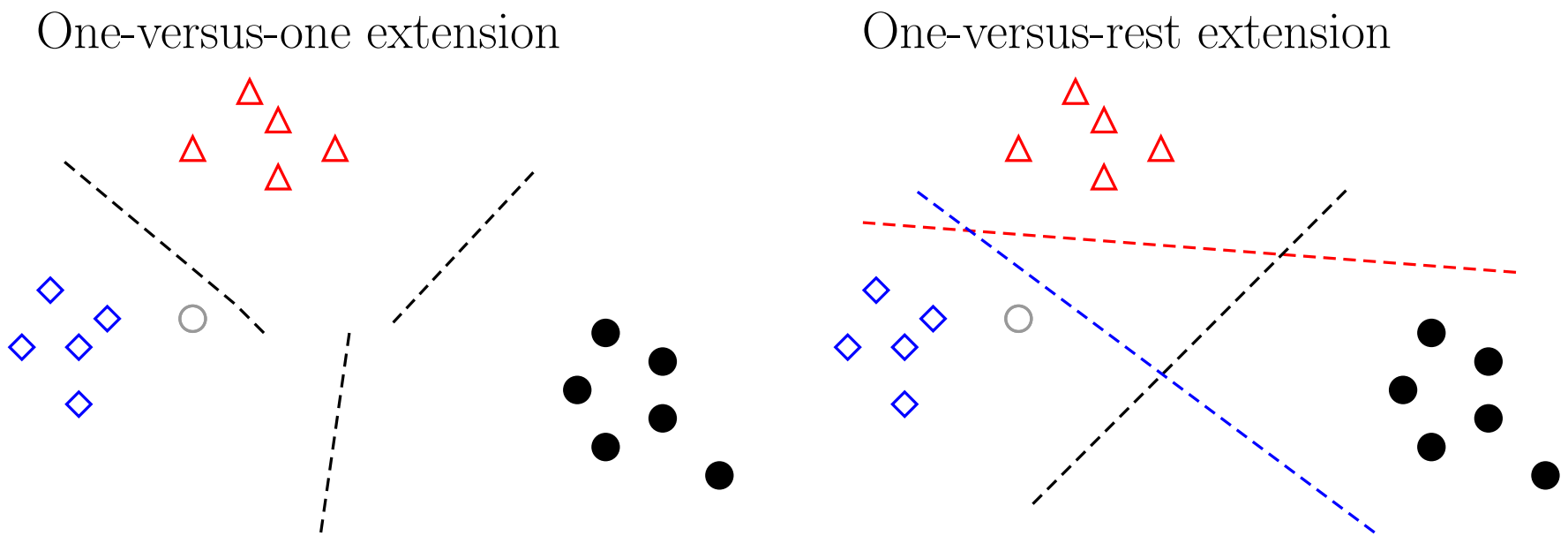


$$\sigma = \frac{1}{n} \Sigma \, \|\mathbf{x}_i - k\text{NN}(\mathbf{x}_i)\|_2$$

Multiclass extensions

Multiclass SVM

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



In either case, the "maximally winning" class is adopted as the final prediction:

- For one-versus-one multiclass SVM, the final prediction for a test point is the most frequent label predicted;
- For one-versus-rest multiclass SVM,
  - The reference classes are assigned label 1 (the rest with label -1)
  - For each binary model, record the 'score': $\mathbf{w}^{(i)} \cdot \mathbf{x} + b^{(i)}$ (not the predicted label)
  - The final prediction is the reference class with the largest (positive) score

$$y = \arg\max_i \ \mathbf{w}^{(i)} \cdot \mathbf{x} + b^{(i)}$$

Matlab implementation for Multiclass SVM

The previously mentioned functions, 'svmtrain' and 'fitcsvm', are 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 (due Friday noon, April 29)

This homework tests the SVM classifier on the MNIST digits. In Questions 2-5 below, apply PCA 50 to the entire training set first to reduce the dimensionality. In all questions below try different values of $C = 2^{-4}, 2^{-3}, \ldots, 2^{5}$ (no validation).

1. Apply the linear SVM classifier to the following pars of digits (without performing dimensionality reduction): (1) $1, 7$ (2) $3, 5$, and (3) $4, 9$. Plot the test errors against the different values of C for each pair of digits.

2. Implement the one-versus-one extension of the third-degree polynomial kernel SVM classifier and apply it to the projected MNIST handwritten digits. Plot the test errors against C.

3. Repeat Question 2 with the one-versus-rest extension instead. Which extension is better?

4. Implement the one-versus-one extension of the Gaussian kernel SVM classifier and apply it to the projected MNIST handwritten digits. To set the kernel parameter σ, use a random sample of 600 points with $k = 8$. Plot the test errors.

5. Repeat Question 4 with the one-versus-rest extension instead. Which extension is better?

Important reminder: Please start early as some part of this homework might take a long time to run.

Midterm project 5: SVM

Summarize the SVM method and results. Additionally, you can

- Add plots of validation errors versus C and compare with the test error curves.
- Try other PCA dimensions
- Try a higher order polynomial kernel SVM (e.g., degree 4).