Support Vector Machines and Ensemble Learning

Instructor: Steven C.H. Hoi

School of Information Systems

Singapore Management University

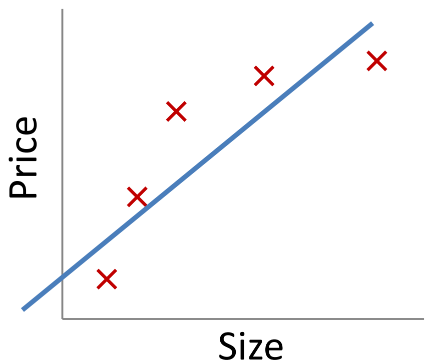
Email: chhoi@smu.edu.sg

Outline

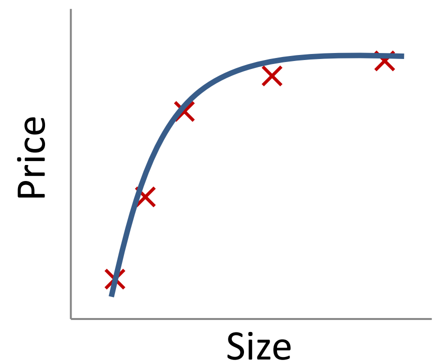
- Overfitting and Regularization
- Support Vector Machines
  - Linear SVM
  - Nonlinear SVM with Kernel
- Ensemble Learning
  - Bagging
  - Boosting

Overfitting

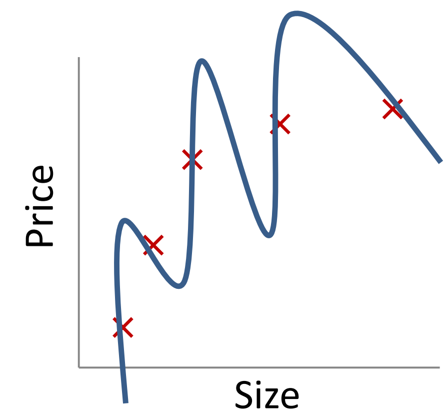
Example: Linear regression (housing prices)



$$w_0 + w_1 x$$



$$w_0 + w_1 x + w_2 x^2$$

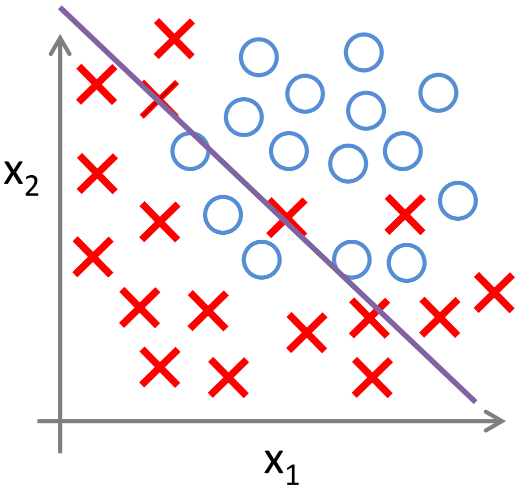


$$w_0 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4$$

Overfitting: If we have too many features (complicated predictor), the learned hypothesis may fit the training set very well, but fail to generalize to new examples (predict prices on new examples).

Overfitting vs. Underfitting

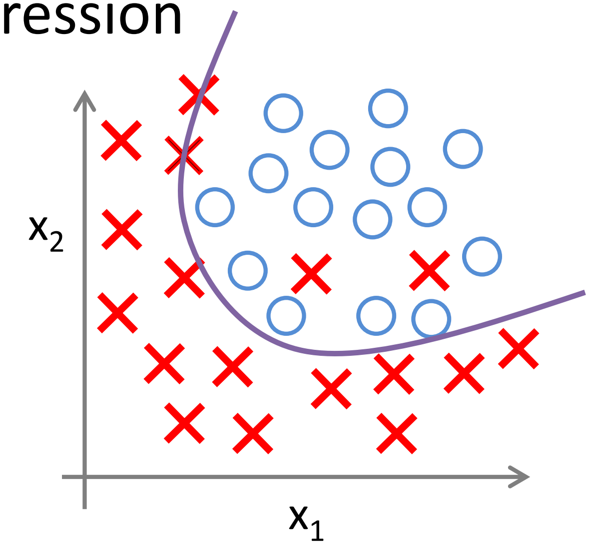
Example: Logistic regression



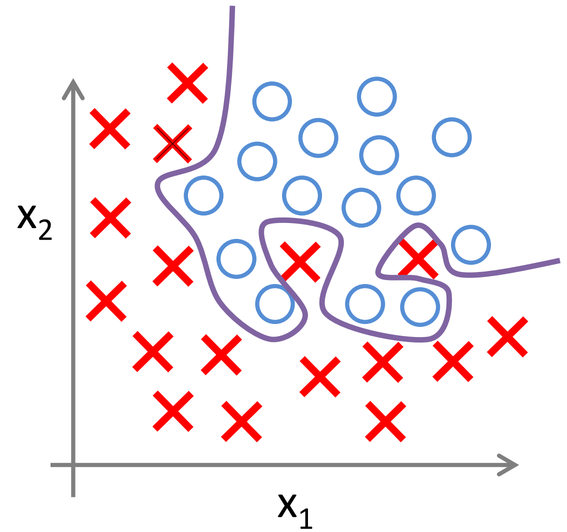
$$h(x) = g(w_0 + w_1 x_1 + w_2 x_2)$$

(g = sigmoid function)

"Underfitting"



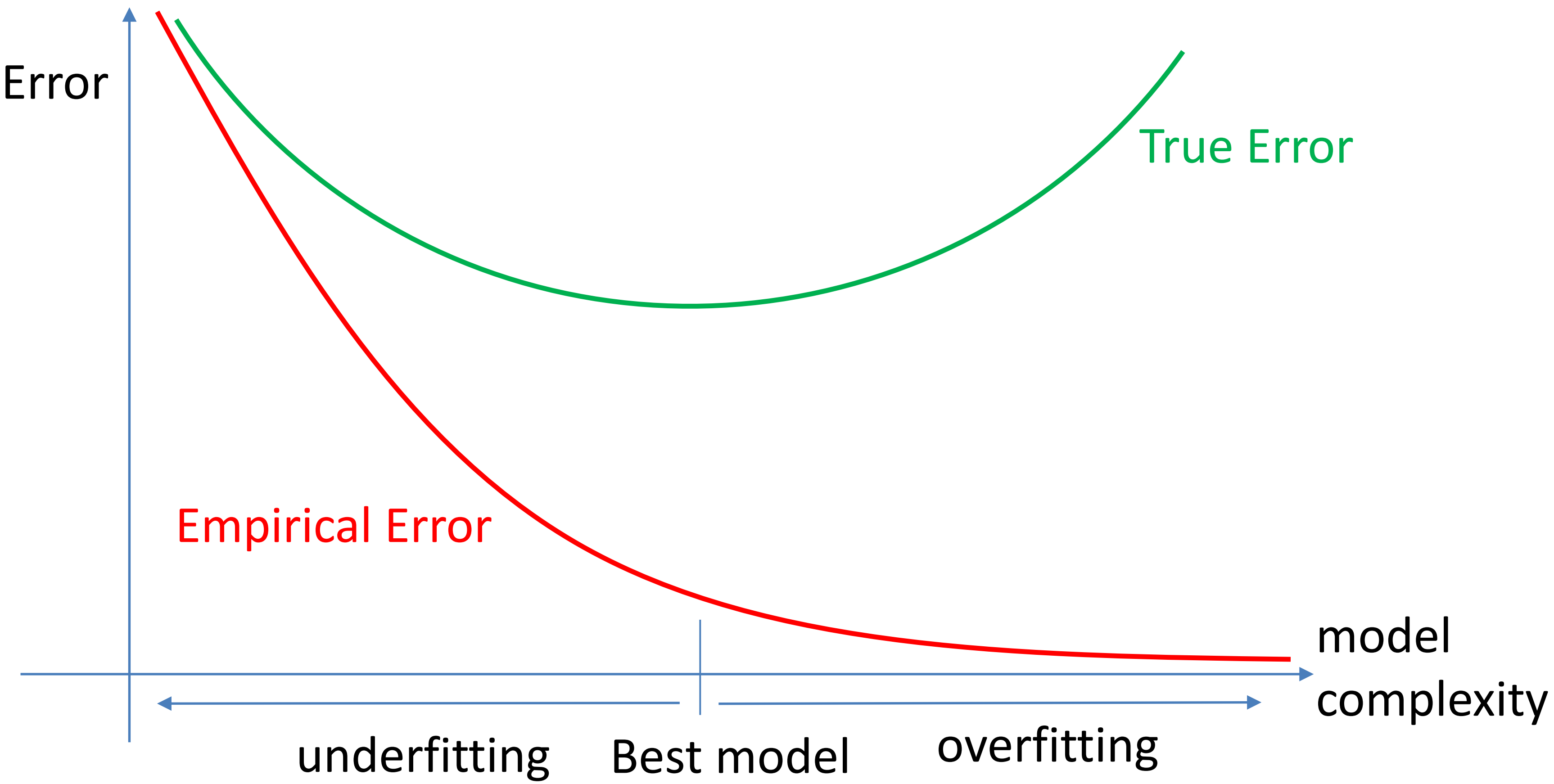
$$g(w_0 + w_1 x_1 + w_2 x_2 \\ + w_3 x_1^2 + w_4 x_2^2 \\ + w_5 x1 x_2)$$



$$g(w_0 + w_1 x_1 + w_2 x_1^2 \\ + w_3 x_1^2 x_2 + w_4 x_1^2 x_2^2 \\ + w_5 x1^2 x_2^3 + w_6 x1^3 x_2 + \ldots)$$

"Overfitting"

Model Complexity



Empirical error (training error) is no longer a good indicator of true error

Examples of Model Complexity

- Examples of Model Spaces with increasing complexity:
  - Regression with polynomials of order k=0,1,2,…
    Higher degree → higher complexity
  - Decision Trees with depth k or with k leaves

    Higher depth/ More # leaves => Higher complexity
  - Nearest-Neighbor classifiers with varying neighbourhood sizes k =1,2,3…
    Small neighborhood => Higher complexity

Occam's Razor

- William of Ockham (1285--1349)

- Principle of Parsimony:

  "One should not increase, beyond what is necessary, the number of entities required to explain anything."

- Alternatively, seek the simplest explanation.

How to address overfitting

- Reduce number of features
  - Manually select which features to keep
  - Model selection algorithms
- Regularization
  - Incorporate model complexity for optimization, penalize complex models using prior knowledge
  - Keep all the features, but reduce magnitude/values of model parameters
  - Works well when we have a lot of features, each of which contributes a bit to the prediction

Regularization

- Regularized learning framework

$$\widehat{f}_n = \arg\min_{f \in \mathcal{F}} \left\{ \widehat{R}_n(f) + C(f) \right\}$$

Cost of model /
model complexity

- Penalize complex models using prior knowledge.
- Examples
  - Regularized Linear Regression (rigid regression)
  - Regularized Logistic Regression
  - Support Vector Machines

SMU
SINGAPORE MANAGEMENT UNIVERSITY

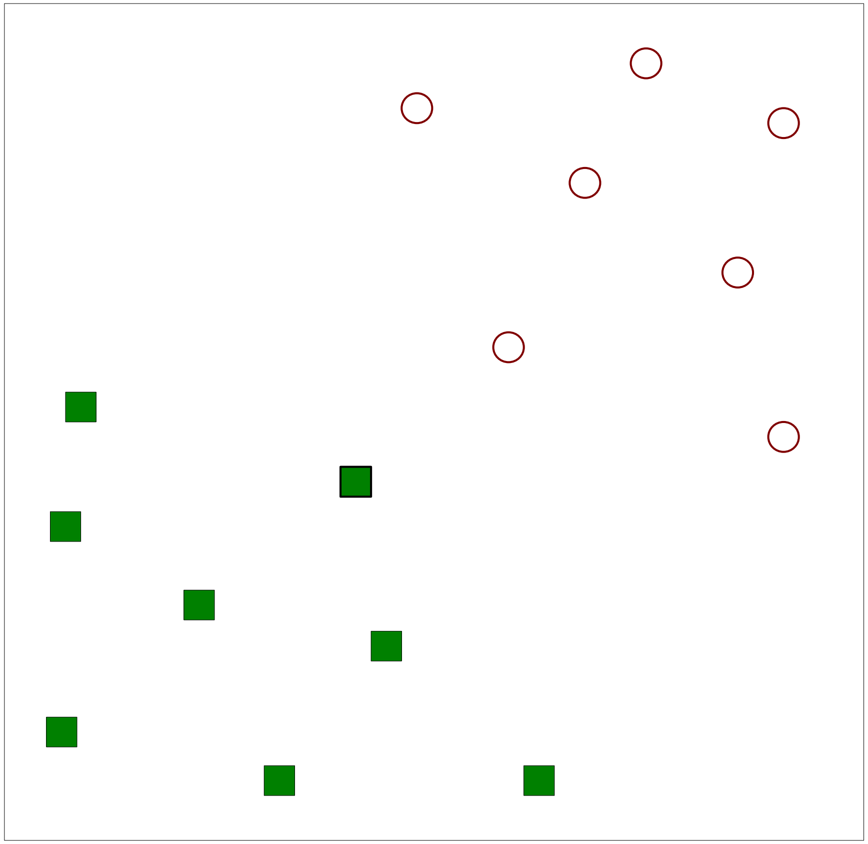
Support Vector Machines

History

- SVMs introduced in COLT-92 by Boser, Guyon & Vapnik. Became rather popular since.
- Theoretically well motivated algorithm: developed from Statistical Learning Theory (Vapnik & Chervonenkis) since the 60s
- Empirically good performance: successful applications in many fields (bioinformatics, text, image recognition, . . .)
- Centralized website: www.kernel-machines.org

Problem Setting

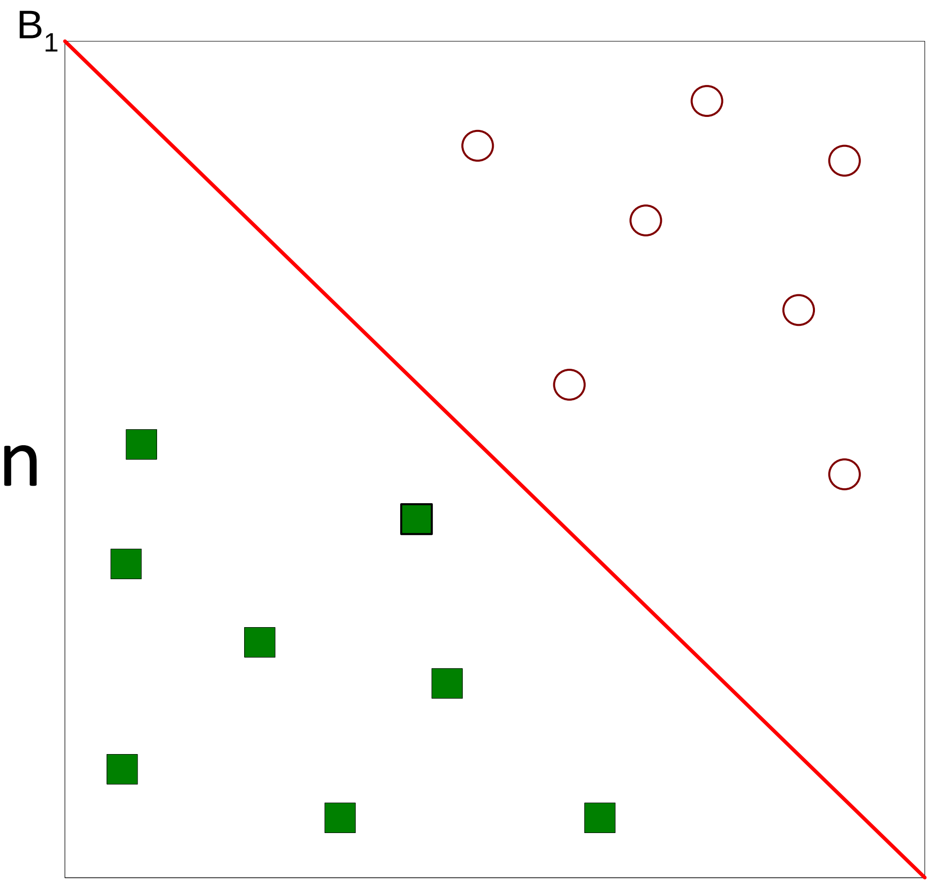
- **Problem Setting**
  - Training data
    $$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots$$
  - For two-class (binary) classification
    $$y_i \in \{+1, -1\}$$
- **Goal**
  - To find an optimal linear hyperplane (decision boundary) that separates all the data



$$f(\mathbf{x}) = \mathrm{sgn}(\mathbf{w}^\top \mathbf{x} + b)$$

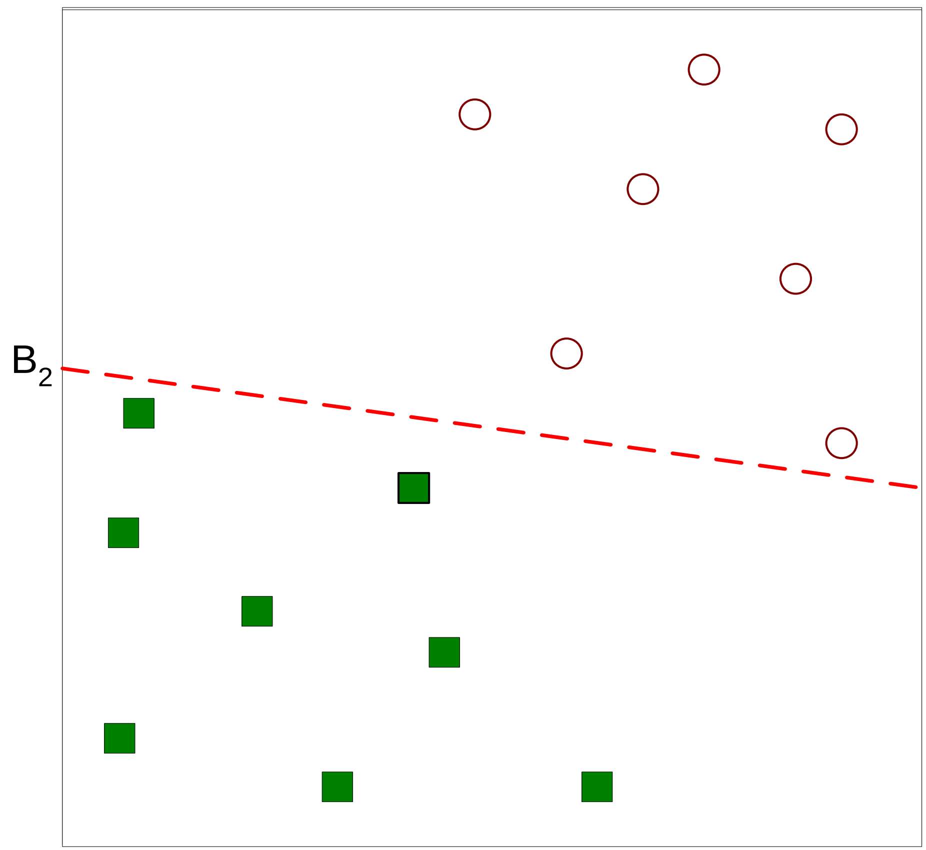
Intuition

- One possible solution



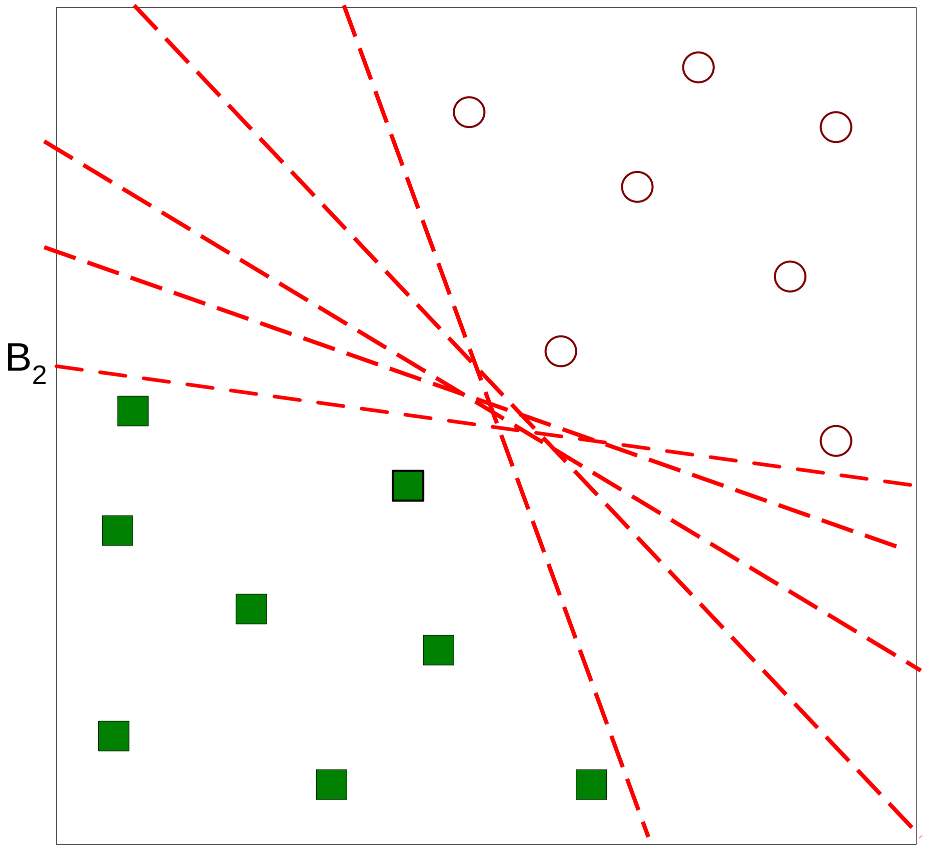
Intuition

- Another possible solution



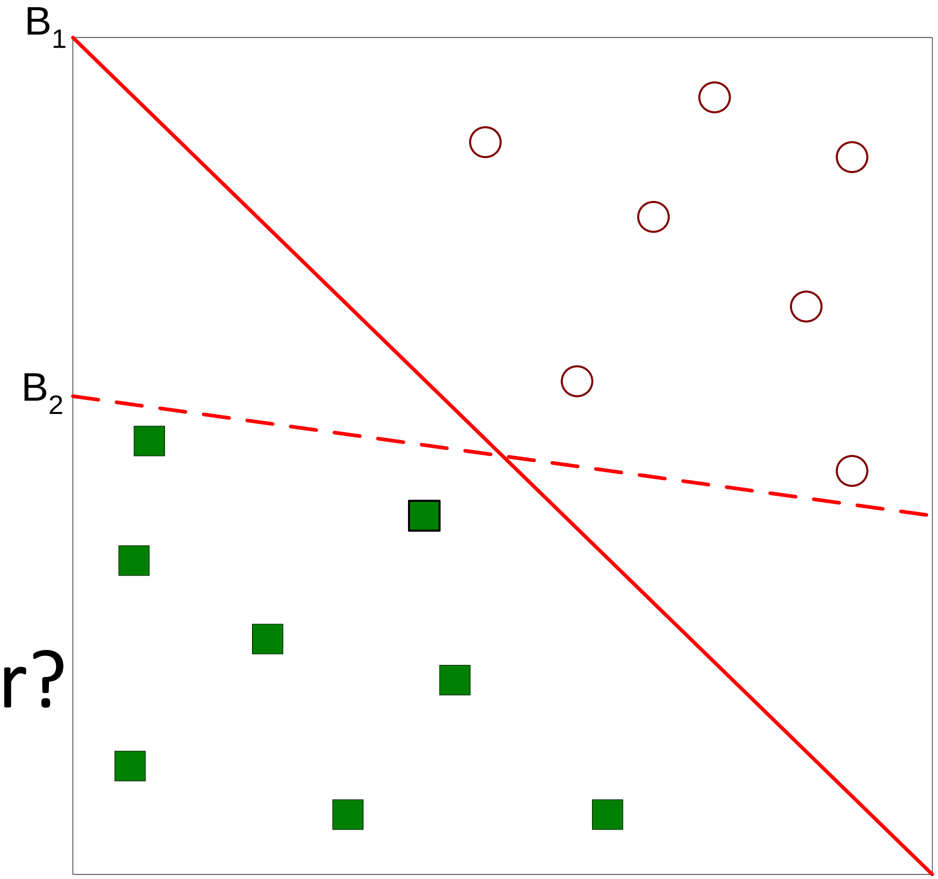
Intuition

- Too many other possible solutions



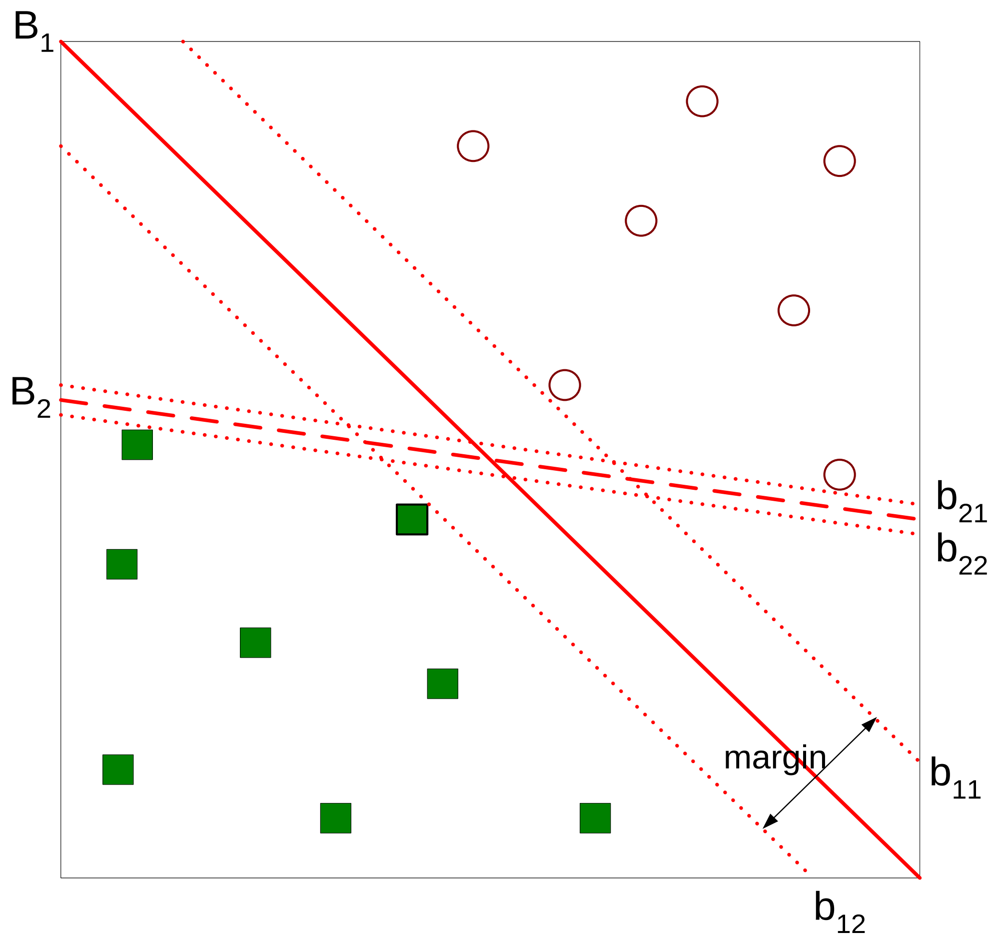
Intuition

- Which one is better than the other?
- How to define better?



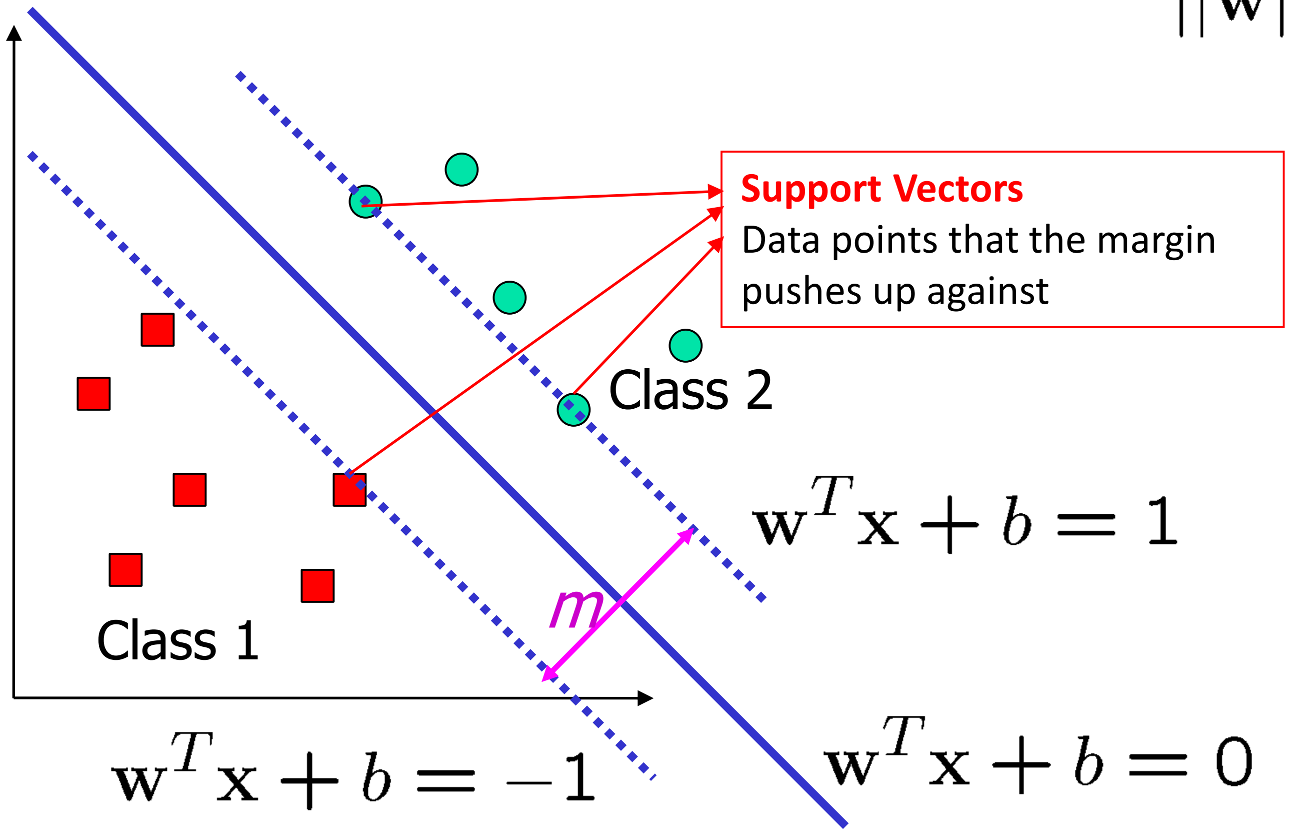
Intuition: Maximum Margin

- Intuition of "Margin"
  - The margin of a linear classifier as the width that the boundary could be increased by before hitting a data point.
- Idea of SVM
  - Find the separating hyperplane maximizing the margin



SVM: Maximum Margin Classifiers

- The decision boundary should be as far away from the data of both classes as possible
  - We should maximize the margin: $m = \frac{2}{||\mathbf{w}||}$



The Optimization Problem

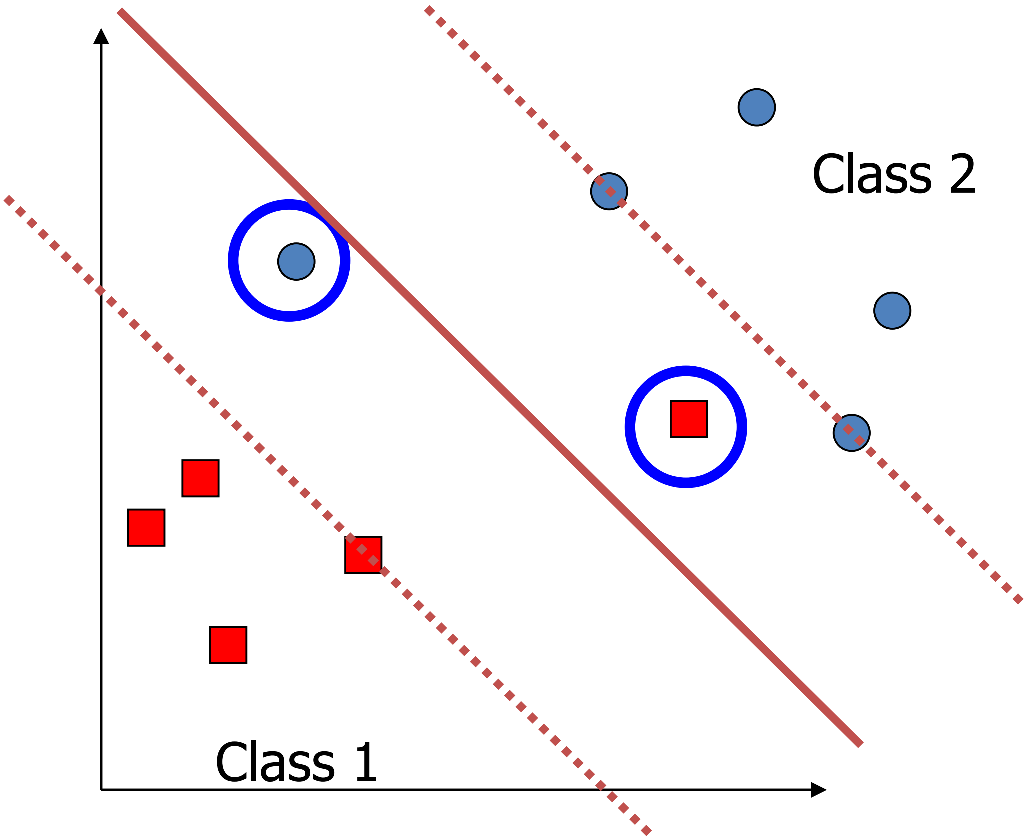
- Let $\{x_1, ..., x_n\}$ be our data set and let $y_i \in \{1,-1\}$ be the class label of x_i
- Maximize margin: $m = \frac{2}{||\mathbf{w}||}$ $||w||^2 = w^T w$
- The decision boundary should classify all points correctly $\Rightarrow$ $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \quad \forall i$
- A constrained optimization problem

$$\text{Minimize } \frac{1}{2}||\mathbf{w}||^2$$

$$\text{subject to } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \qquad \forall i$$

Linearly non-separable case

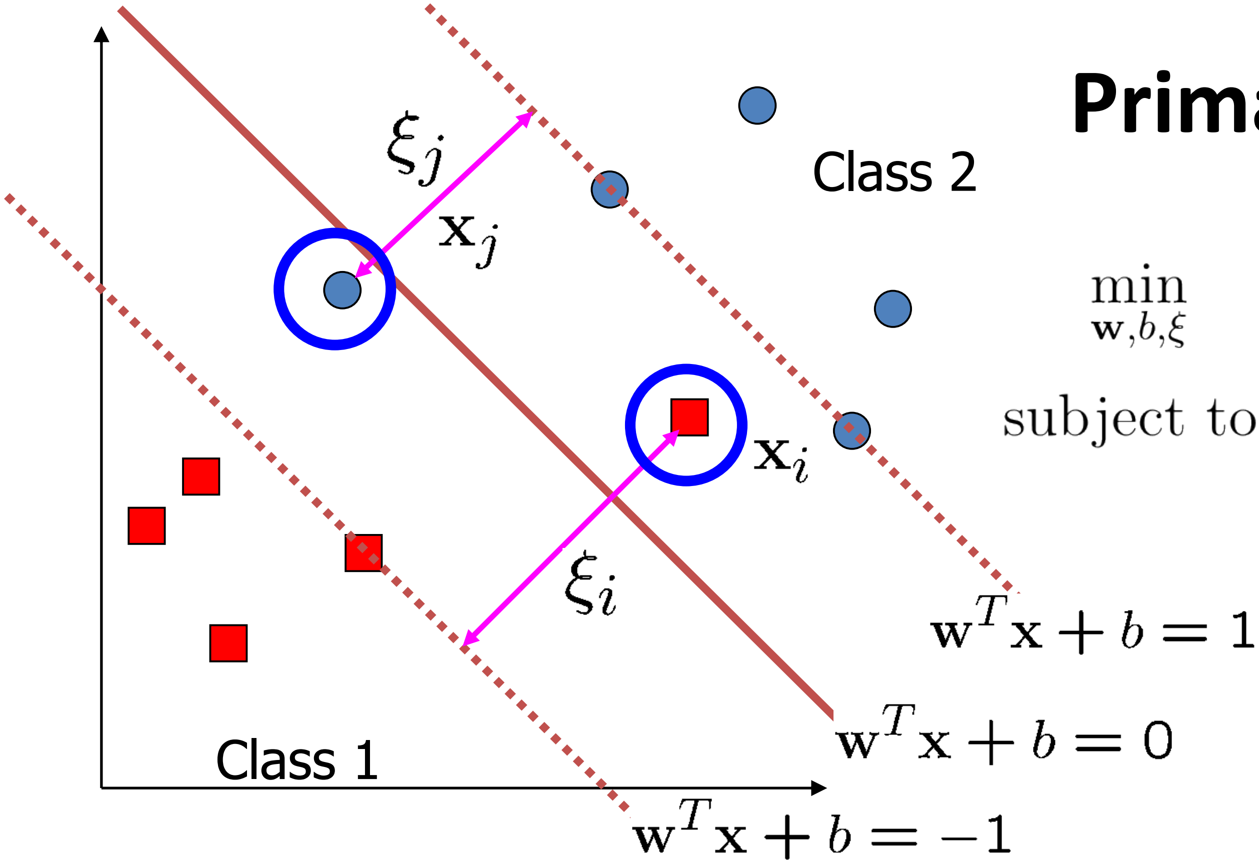
- What if the data cannot be linearly separable?



For such case, a **“hard margin”** linear SVM cannot be applied directly!!

Soft Margin SVM

- Introduce slack variables
- Relax the constraints
- Penalize the relaxation



Primal Optimization

$$\min_{\mathbf{w},b,\xi} \quad \frac{1}{2}\mathbf{w}^\top\mathbf{w} + C\sum_{i=1}^{N} \xi_i$$

$$\text{subject to} \quad y_i(\mathbf{w}^\top\mathbf{x}_i + b) \geq 1 - \xi_i,$$
$$\xi_i \geq 0, i = 1, \ldots, N$$

C is a penalty cost parameter (regularization)

Optimization: Primal vs Dual

- **Primal**:

$$\min_{\mathbf{w},b,\xi} \quad \frac{1}{2}\mathbf{w}^\top\mathbf{w} + C\sum_{i=1}^{n}\xi_i$$

$$\text{subject to} \quad y_i(\mathbf{w}^\top\mathbf{x}_i + b) \geq 1 - \xi_i,$$
$$\xi_i \geq 0, i = 1, \ldots, n$$

- **Dual**:

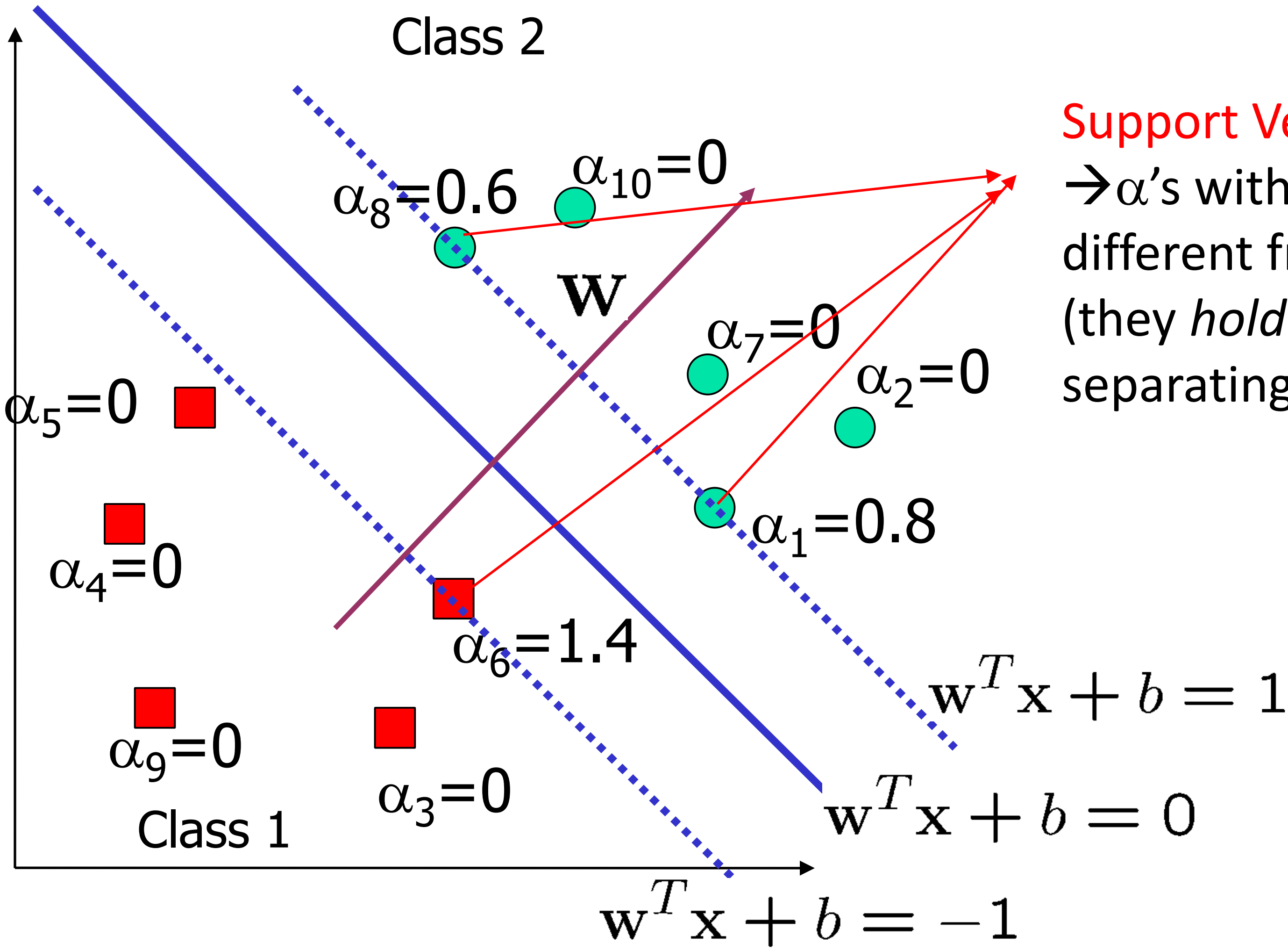
$$\text{max.} \quad W(\boldsymbol{\alpha}) = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1,j=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j$$

$$\text{subject to } C \geq \alpha_i \geq 0, \sum_{i=1}^{n}\alpha_i y_i = 0$$

- Model: $\mathbf{w} = \sum_{j=1}^{s}\alpha_{t_j}y_{t_j}\mathbf{x}_{t_j}$

SVM can be solved by Convex Optimization: Quadratic Programming (QP)

A Geometrical Interpretation

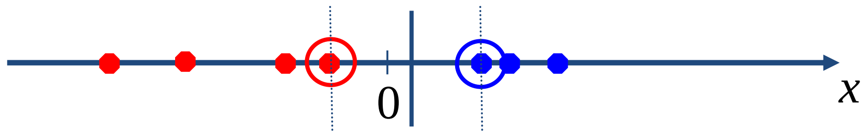


Support Vectors

→ α's with values different from zero (they *hold up* the separating plane)!

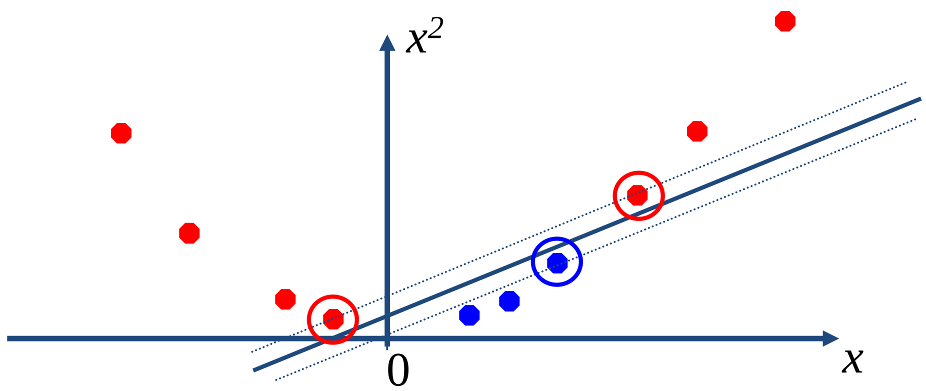
SVM: Non-linear separable case

- Datasets that are linearly separable with noise work out great:



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

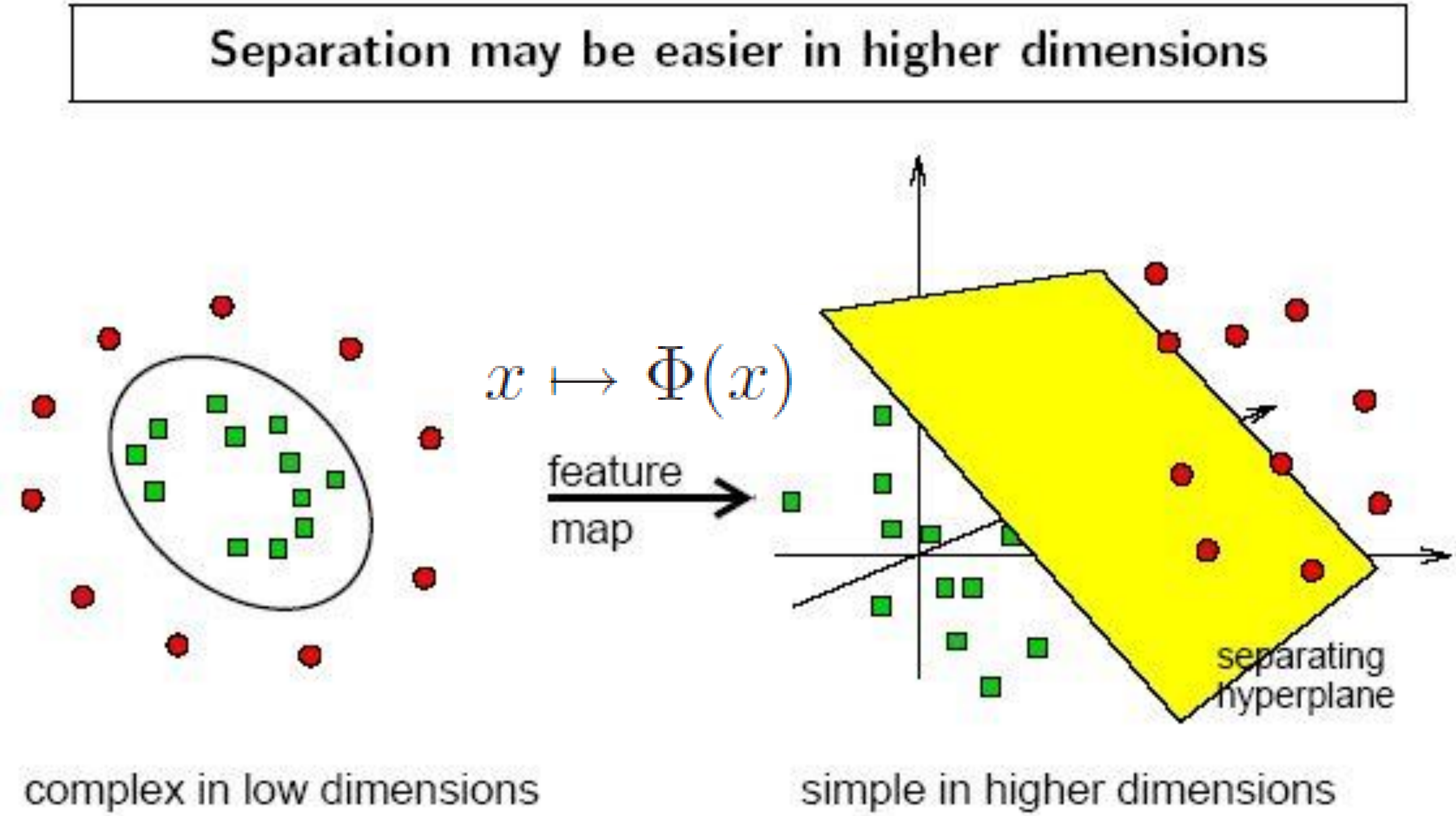
- How about… mapping data to a higher-dimensional space:



This slide is courtesy of *www.iro.umontreal.ca/~pift6080/documents/papers/**svm**_tutorial.ppt*

Nonlinear SVM

- Basic Idea of Nonlinear SVM



How to choose the feature mapping?

$$x \mapsto \Phi(x)$$

- Example (Polynomial mapping):

$$\mathbf{x} \in R^3, \phi(\mathbf{x}) \in R^{10}$$

$$\phi(\mathbf{x}) \;=\; (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_3, x_1^2,\; x_2^2, x_3^2, \sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \sqrt{2}x_2x_3)$$

- Problem of explicit feature mapping:
  - The dimensionality of feature mapping can be very large, making it hard to represent explicitly in memory, and hard to solve QP

Kernel Tricks

- **Idea:** Replacing dot product with a kernel $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i)^\top \Phi(\mathbf{x}_j)$$

- **Kernel Functions:**
  – Linear Kernel $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle = \mathbf{x}_i^\top \mathbf{x}_j$
  – Polynomial Kernel (degree d) $K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y} + 1)^d$

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^\top \mathbf{x}_j / a + b)^d$$

  – Gaussian / RBF Kernel

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(- \frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2} \right) \quad \kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(- \gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2 \right)$$

Example Transformation

- Consider the following transformation

$$\phi(\left[\begin{smallmatrix} x_1 \\ x_2 \end{smallmatrix}\right]) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

$$\phi(\left[\begin{smallmatrix} y_1 \\ y_2 \end{smallmatrix}\right]) = (1, \sqrt{2}y_1, \sqrt{2}y_2, y_1^2, y_2^2, \sqrt{2}y_1y_2)$$

- Define the kernel function *K* (**x**,**y**) as

$$\begin{aligned} \langle \phi(\left[\begin{smallmatrix} x_1 \\ x_2 \end{smallmatrix}\right]), \phi(\left[\begin{smallmatrix} y_1 \\ y_2 \end{smallmatrix}\right]) \rangle &= (1 + x_1y_1 + x_2y_2)^2 \\ &= K(\mathbf{x}, \mathbf{y}) \end{aligned}$$

$$K(\mathbf{x}, \mathbf{y}) = (1 + x_1y_1 + x_2y_2)^2$$

- The inner product $\phi(.)\phi(.)$ can be computed by *K* without going through the map $\phi(.)$ explicitly!!!

Modification Due to Kernel Function

- Change all inner products to kernel functions
- For training:

Original

$$\text{max.}\ \ W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j \boxed{\mathbf{x}_i^T \mathbf{x}_j}$$

$$\text{subject to } C \geq \alpha_i \geq 0, \sum_{i=1}^{n} \alpha_i y_i = 0$$

With kernel function

$$\text{max.}\ \ W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j \boxed{K(\mathbf{x}_i, \mathbf{x}_j)}$$

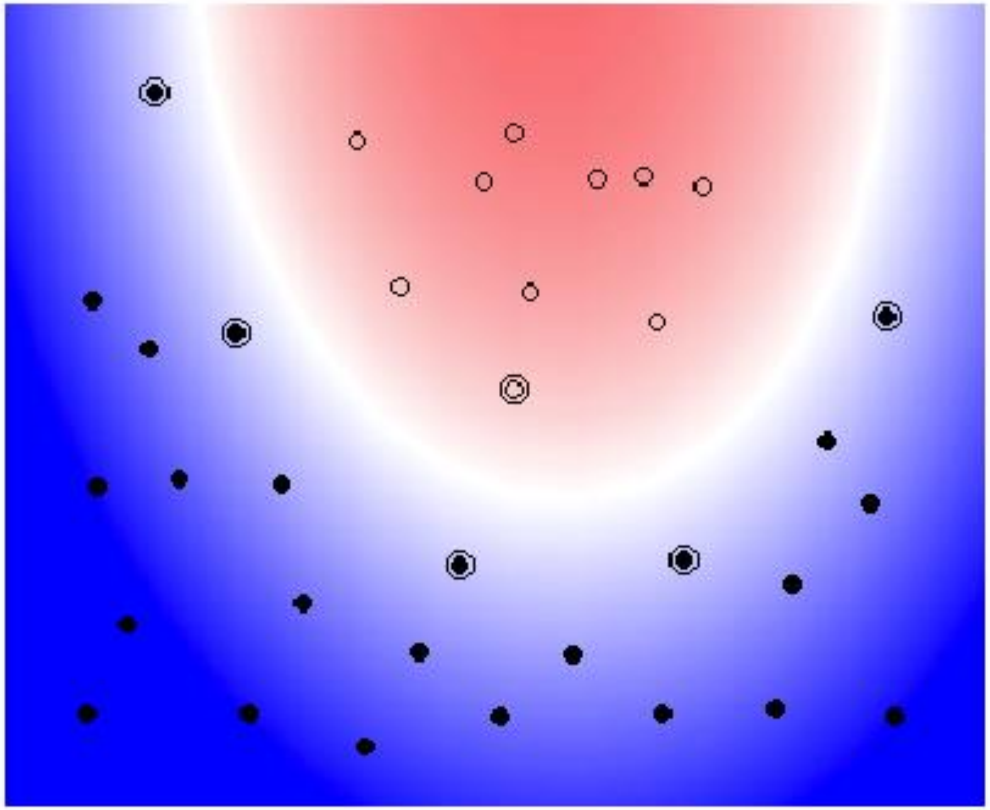
$$\text{subject to } C \geq \alpha_i \geq 0, \sum_{i=1}^{n} \alpha_i y_i = 0$$

Nonlinear SVM with Kernel (I)

Example: SVM with Polynomial of Degree 2

Kernel: $K(\vec{x}_i, \vec{x}_j) = [\vec{x}_i \cdot \vec{x}_j + 1]^2$

plot by Bell SVM applet

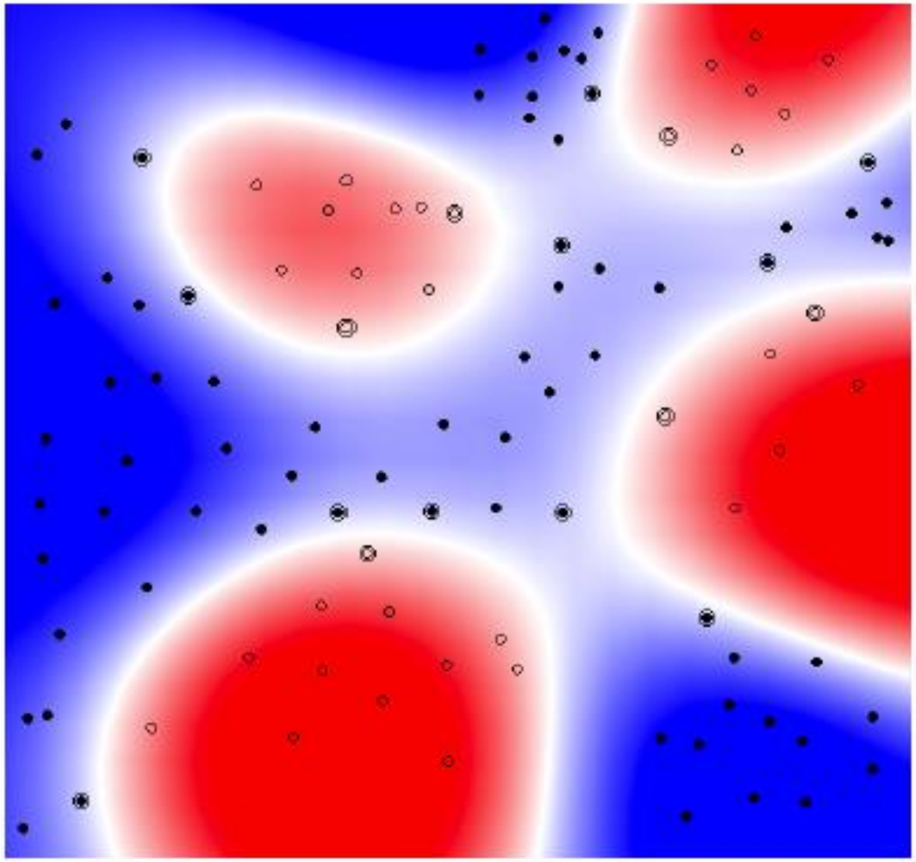


Nonlinear SVM with Kernel (II)

Example: SVM with RBF-Kernel

Kernel: $K(\vec{x}_i, \vec{x}_j) = \exp(-|\vec{x}_i - \vec{x}_j|^2 / \sigma^2)$

plot by Bell SVM applet



Recap of Steps in SVM Practice

- Prepare training data matrix $\{(x_i, y_i)\}$
- Select a Kernel function K
- Select the error parameter *C*
- “Train” the model (to find all α_i)
- New test data can be classified using α_i and Support Vectors

Linear Classifier vs Kernel Methods

- D: number of features, N: number of training examples
- If D is large (relative to N)
  - Use linear classifiers (SVM or Logistic Regression)
- If D is small, N is intermediate
  - Use SVM with Gaussian kernel
- If D is small, N is large
  - Create/Add more features, use linear classifiers (SVM or logistic regression) without kernels
  - Try kernel approximation, if the above are not good enough

References

- An excellent tutorial on VC-dimension and SVM:
    - C.J.C. Burges. A tutorial on support vector machines for pattern recognition. Data Mining and Knowledge Discovery, 2(2):955-974, 1998. http://citeseer.nj.nec.com/burges98tutorial.html
- Further references, for example, [Cristianini and Shawe-Taylor, 2000, Scholkopf and Smola, 2002]
- The VC/SRM/SVM Bible: (Not for beginners)
    - "Statistical Learning Theory" by Vladimir Vapnik, Wiley-Interscience; 1998
- Software for SVM:
    - LIBSVM: http://www.csie.ntu.edu.tw/~cjlin/libsvm/
    - SVM-light http://svmlight.joachims.org/
    - LIBLinear: http://www.csie.ntu.edu.tw/~cjlin/liblinear/

SMU
SINGAPORE MANAGEMENT UNIVERSITY

Ensemble Learning

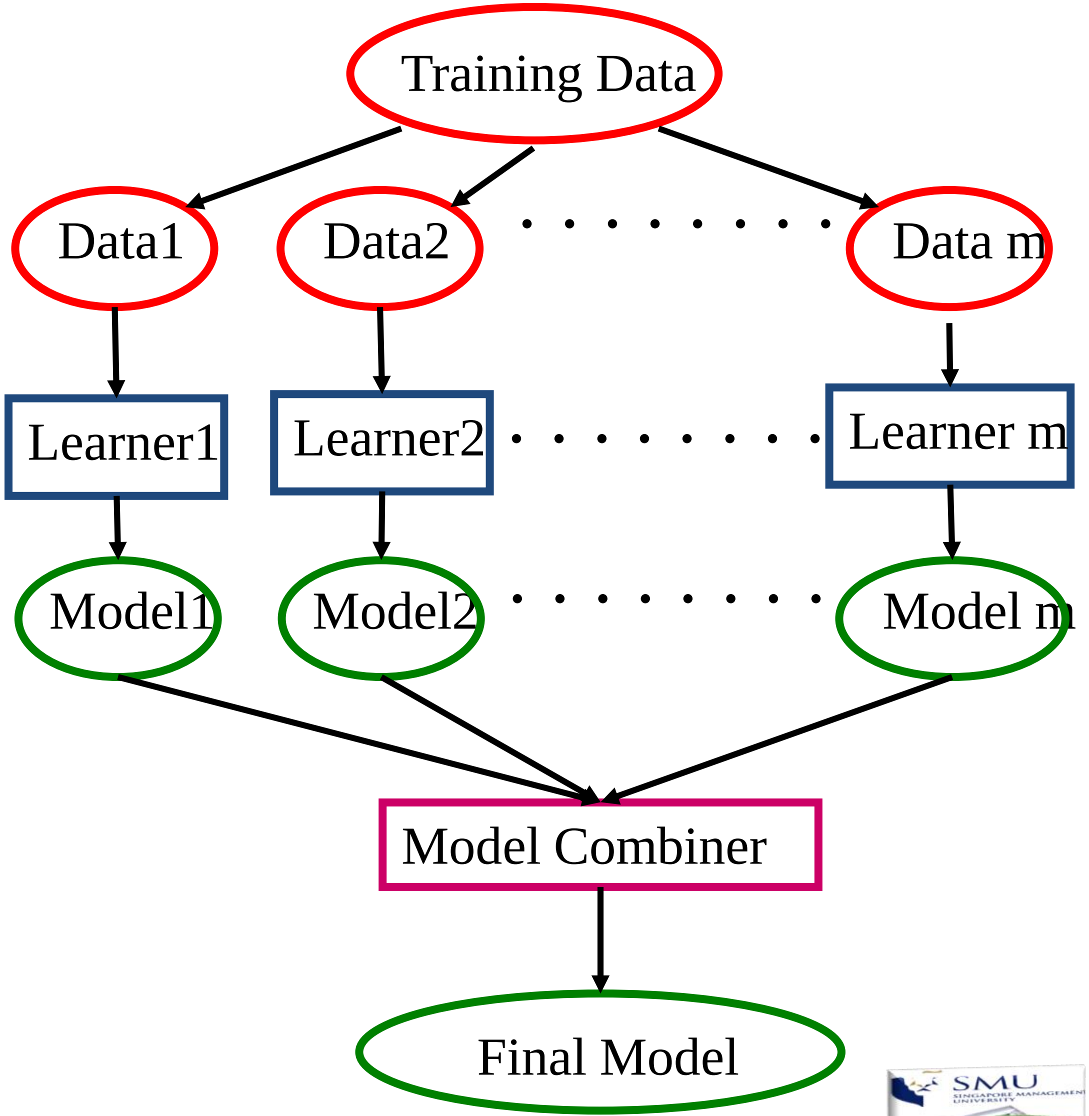
Ensemble Learning

- Bagging
- Boosting
- Stacking (appendix)

Ensemble Learning

- Basic Idea:
  Instead of learning one model, learning several and combine them

- Typically improves the accuracy, often by a lot



Why does it work?

- Suppose there are 25 base classifiers
  - Each classifier has error rate, $\varepsilon = 0.35$
  - Assume classifiers are independent
  - Probability that the ensemble classifier makes a wrong prediction (i.e.,*13 out of the 25 classifiers misclassified*):

$$\sum_{i=13}^{25} \binom{25}{i} \varepsilon^{i} (1-\varepsilon)^{25-i} = 0.06$$

Bagging

- Bagging = Boostrap aggregating
- Boostrap Sampling:

Given set *D* containing *m* training examples

- Create D^i by drawing *m* examples at random *with replacement* from *D*
- D^i expects to leave out about $(1 - 1/m)^m \approx 0.37$ of examples from D

Original Data	1	2	3	4	5	6	7	8	9	10
Bagging (Round 1)	7	8	10	8	2	5	10	10	5	9
Bagging (Round 2)	1	4	9	1	2	3	2	7	3	2
Bagging (Round 3)	1	8	5	10	5	5	9	6	3	7

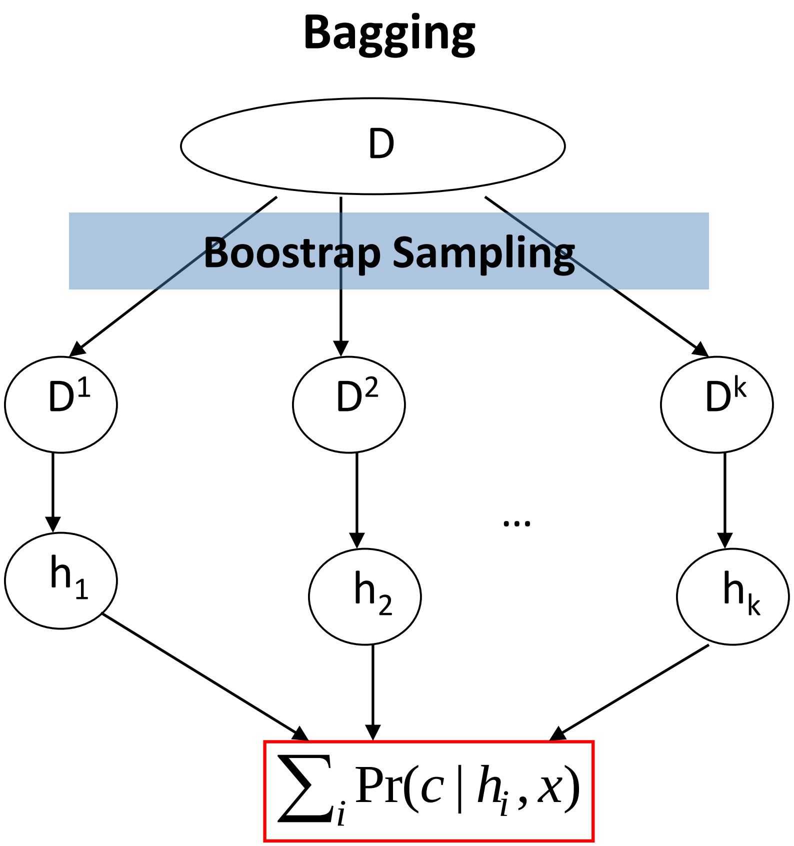
Bagging Algorithm

- Create k boostrap samples $D^1, D^2, \ldots, D^k$
- Train distinct classifier h_i on each D^i
- Classify a new instance $\mathbf{x}$ by classifier vote with equal weights

$$c^*(\mathbf{x}) = \arg\max_c \sum_{i=1}^{k} p(c|h_i, \mathbf{x})$$

Limitations with Bagging

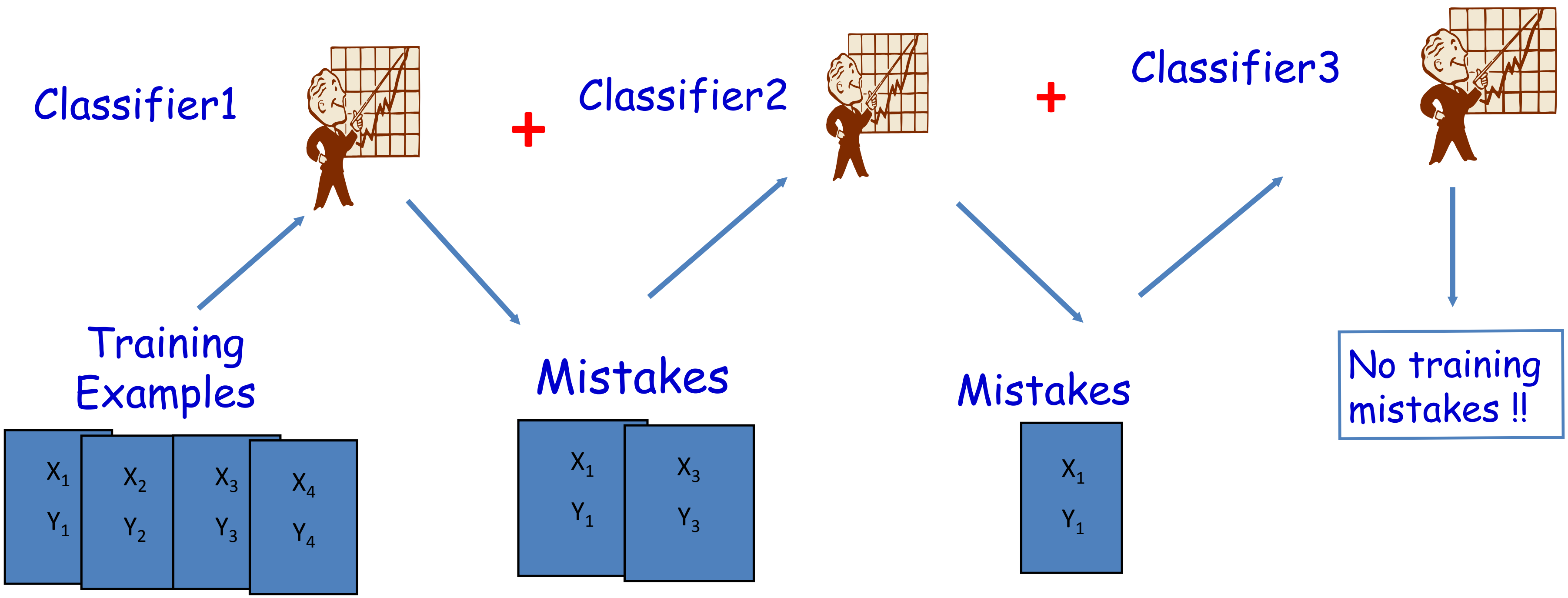
- **Inefficient boostrap sampling:**
  - Every example has equal chance to be sampled
  - No distinction between “easy” examples and “difficult” examples
- **Inefficient model combination:**
  - A constant weight for each classifier
  - No distinction between accurate classifiers and inaccurate classifiers



Improving the Efficiency of Bagging

- Better sampling strategy
  - Focus on the examples that are difficult to classify
- Better combination strategy
  - Accurate model should be assigned larger weights

Boosting: Intuition



Boosting: Example

- Instances that are wrongly classified will have their weights increased
- Instances that are correctly classified will have their weights decreased

Original Data	1	2	3	4	5	6	7	8	9	10
Boosting (Round 1)	7	3	2	8	7	9	4	10	6	3
Boosting (Round 2)	5	4	9	4	2	5	1	7	4	2
Boosting (Round 3)	4	4	8	10	4	5	4	6	3	4

- Example 4 is hard to classify
- Its weight is increased, therefore it is more likely to be chosen again in subsequent rounds

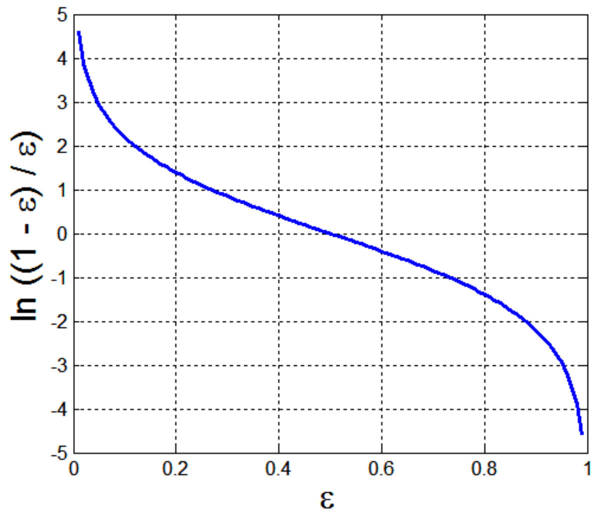
AdaBoost Algorithm

Given: $(x_1, y_1), \ldots, (x_m, y_m); x_i \in \mathcal{X}, y_i \in \{-1, +1\}$

Initialise weights $D_1(i) = 1/m$

For $t = 1, \ldots, T$:

- Find $h_t = \arg\min_{h_j \in \mathcal{H}} \epsilon_j = \sum_{i=1}^{m} D_t(i) [\![y_i \neq h_j(x_i)]\!]$
- If $\epsilon_t \geq 1/2$ then stop
- $\boxed{\text{Set } \alpha_t = \frac{1}{2} \log\left(\frac{1-\epsilon_t}{\epsilon_t}\right)}$
- Update



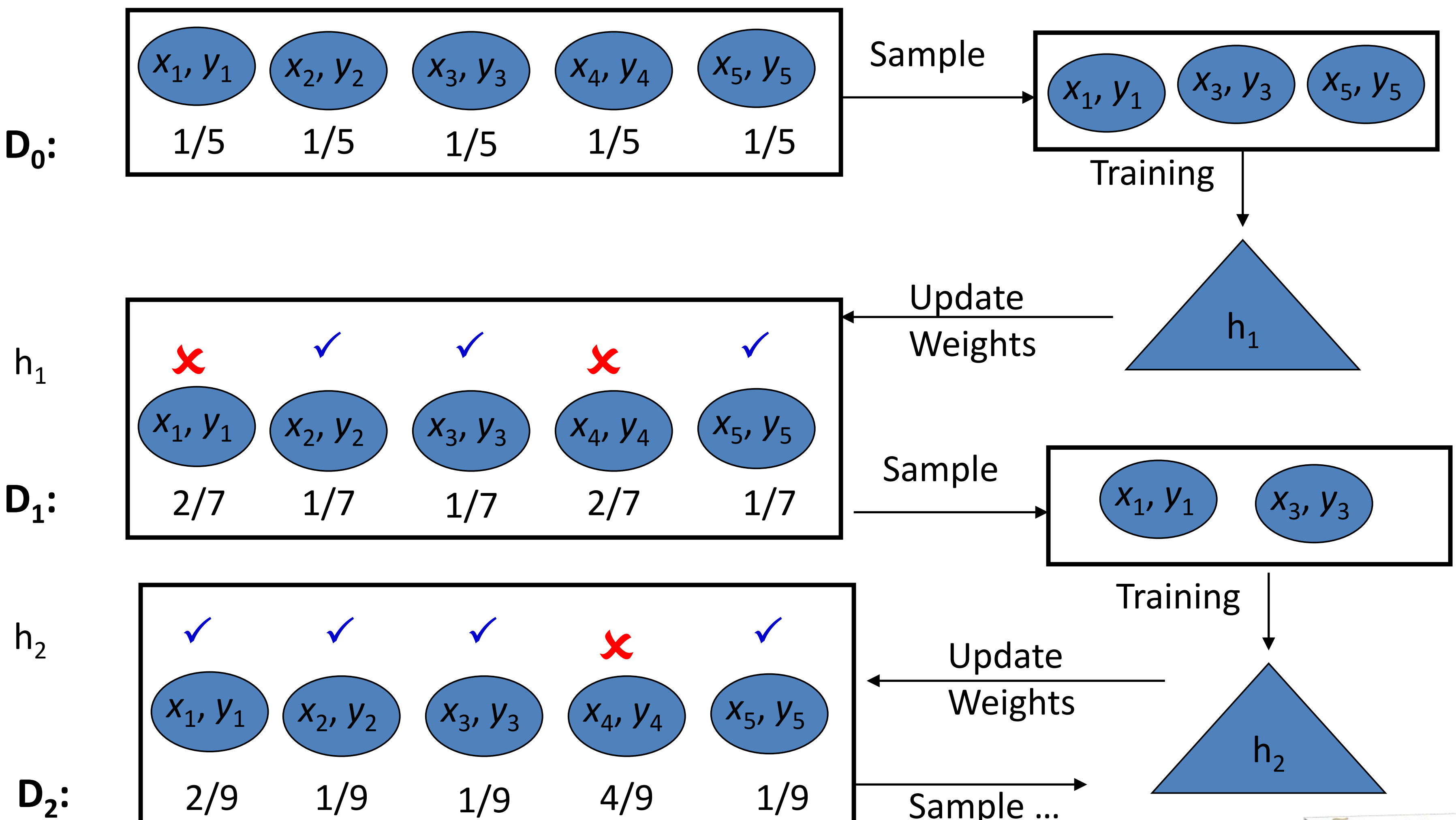
$$D_{t+1}(i) = \frac{D_t(i) exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

Where Zt is a normalization factor chosen so that Dt+1 will be a distribution.

Output the final classifier:

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

AdaBoost Example: α_t=ln2



Empirical Study of Bagging & Boosting

- **Bagging decision trees**
  - Boostrap 50 different samples from the original training data
  - Learn a decision tree over each boostrap sample
  - Predict by majority vote of 50 decision trees
- **AdaBoosting decision trees**
  - Generate 50 decision trees by AdaBoost
  - Linearly combine decision trees using the weights of AdaBoost

In general:

- AdaBoost = Bagging > C4.5
- AdaBoost usually needs less number of classifiers than Bagging

	C4.5		
Dataset	Standard	Bagging	Boosting Ada
breast-cancer-w	5.0	3.3	3.1
credit-a	14.9	12.1	12.6
credit-g	29.6	22.8	22.9
diabetes	28.3	21.9	22.3
glass	30.9	28.4	30.5
heart-cleveland	24.3	18.1	17.4
hepatitis	21.6	16.5	13.8
house-votes-84	3.5	3.6	4.4
hypo	0.5	0.4	0.4
ionosphere	8.1	6.0	6.0
iris	6.0	4.6	5.6
kr-vs-kp	0.6	0.5	0.3
labor	15.1	13.3	13.2
letter	14.0	10.6	6.7
promoters-936	12.8	9.5	6.3
ribosome-bind	11.2	9.3	9.1
satellite	13.8	10.8	10.4
segmentation	3.7	2.8	2.3
sick	1.3	1.0	0.9
sonar	29.0	21.6	19.7
soybean	8.0	8.0	7.9
splice	5.9	5.7	6.3
vehicle	29.4	26.1	24.8

Summary

- Overfitting and Regularization
- Support Vector Machines
  - Linear SVM
  - Nonlinear SVM with Kernel
- Ensemble Learning
  - Bagging
  - Boosting

Appendix

- Regularized Loss Minimization
- Quadratic Programming
- Stacking
- Multi-class SVM classification
- Deriving the Dual of SVM
- Kernel Learning
- Curse of Kernelization

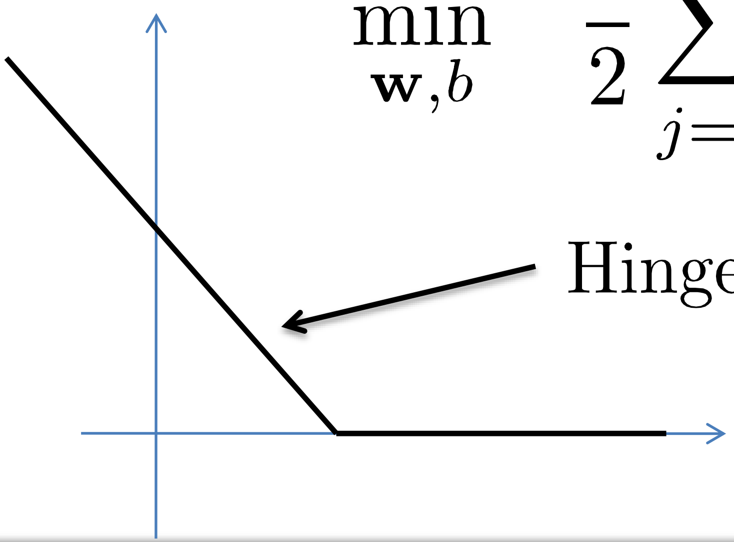
SVM as Regularized Loss Minimization

- The objective function of SVM (soft margin linear SVM) can be re-written as an unconstrained optimization:

Model Complexity

Training Error

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\sum_{j=1}^{d} w_j^2 + C\sum_{i=1}^{N}\ell(y_i[\mathbf{x}_i^\top \mathbf{w} + b])$$

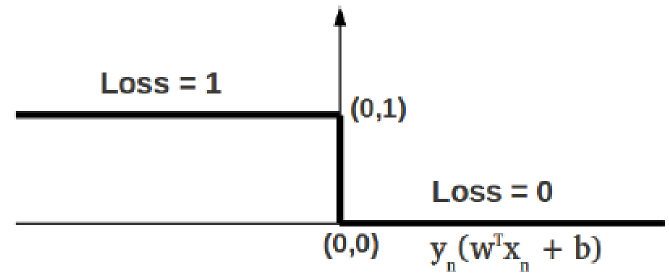


Hinge loss $\ell(z) = \max(0, 1 - z)$

Regularized Loss Minimization

$$\widehat{f}_n = \arg\min_{f \in \mathcal{F}} \left\{ \widehat{R}_n(f) + C(f) \right\}$$

$$\min_{\mathbf{w}} \sum_{i=1}^{m} \ell(h(\mathbf{x}_i), y_i) + \lambda \|\mathbf{w}\|_2^2$$



0-1 loss function

$$\min_{\mathbf{w}} \sum_{i=1}^{m} \mathbb{I}(y_i(\mathbf{w}^T \mathbf{x}_i + b) < 0) + \lambda \|\mathbf{w}\|_2^2$$

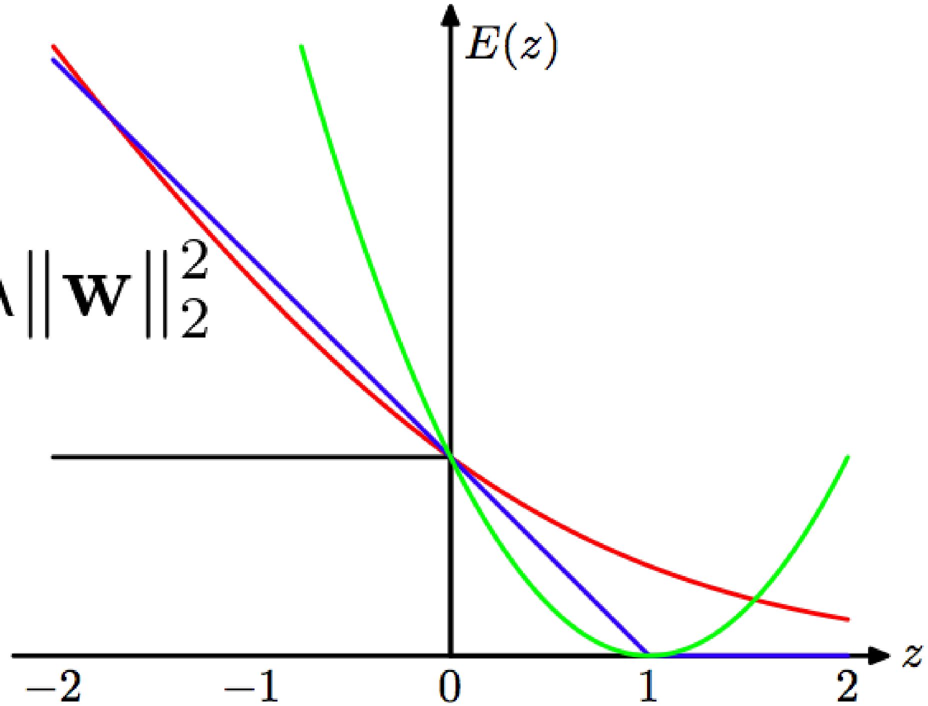
$\mathbb{I}(.)$ is the indicator function (1 if (.) is true, 0 otherwise)

The 0-1 loss is NP-hard to optimize exactly.

Regularized Loss Minimization

- Regularized Loss Minimization framework

$$\min_{\mathbf{w}} \sum_{i=1}^{m} \ell(h(\mathbf{x}_i), y_i) + \lambda \|\mathbf{w}\|_2^2$$



- Examples:
  - Hinge loss → SVM

    $$[1 - y_n \mathbf{w}^T \mathbf{x}_n]_+ = \max\{0, 1 - y_n \mathbf{w}^T \mathbf{x}_n\}$$

  - Logistic loss → L2-Regularized Logistic Regression

    $$\log[1 + \exp(-y_n \mathbf{w}^T \mathbf{x}_n)]$$

  - Squared loss → L2-regularized Linear Regression

Quadratic Programming

Find $$\arg\min_{\mathbf{u}} \; c + \mathbf{d}^T\mathbf{u} + \frac{\mathbf{u}^T R\mathbf{u}}{2}$$

Quadratic criterion

Subject to

$$\left.\begin{array}{c}
a_{11}u_1 + a_{12}u_2 + \ldots + a_{1m}u_m \le b_1 \\
a_{21}u_1 + a_{22}u_2 + \ldots + a_{2m}u_m \le b_2 \\
\vdots \\
a_{n1}u_1 + a_{n2}u_2 + \ldots + a_{nm}u_m \le b_n
\end{array}\right\}$$

n additional linear <u>in</u>equality constraints

And subject to

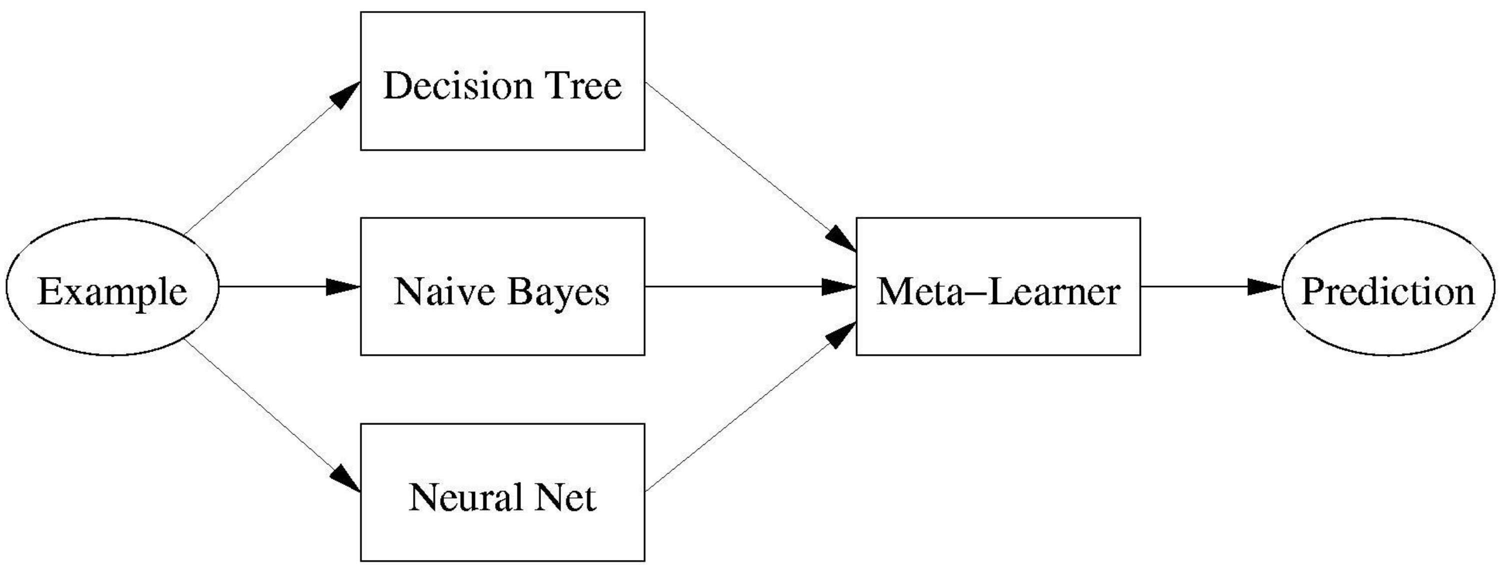
$$\left.\begin{array}{c}
a_{(n+1)1}u_1 + a_{(n+1)2}u_2 + \ldots + a_{(n+1)m}u_m = b_{(n+1)} \\
a_{(n+2)1}u_1 + a_{(n+2)2}u_2 + \ldots + a_{(n+2)m}u_m = b_{(n+2)} \\
\vdots \\
a_{(n+e)1}u_1 + a_{(n+e)2}u_2 + \ldots + a_{(n+e)m}u_m = b_{(n+e)}
\end{array}\right\}$$

e additional linear <u>e</u>quality constraints

Matlab/Octave **“quadprog”**

Stacking

- Apply multiple base learners
  (e.g.: decision trees, naive Bayes, neural nets)
- Meta-learner: Inputs = Base learner predictions
- Training by leave-one-out cross-validation:
  Meta-L. inputs = Predictions on left-out examples



Multi-class Classification

- Consider k classes
- **One-against-the rest**: Train k binary SVMs:
  - 1st class vs. (2 – k)th class
  - 2nd class vs. (1, 3 – k)th class
  - …
- k decision functions

$$(\mathbf{w}^1)^T \phi(\mathbf{x}) + b_1$$
$$\vdots$$
$$(\mathbf{w}^k)^T \phi(\mathbf{x}) + b_k$$

Multi-class Classification

- Prediction

$$\arg\max_j \ (\mathbf{w}^j)^T \phi(\mathbf{x}) + b_j$$

- Reason: If it's the 1st class, then we should have

$$
\begin{aligned}
(\mathbf{w}^1)^T \phi(\mathbf{x}) + b_1 &\geq +1 \\
(\mathbf{w}^2)^T \phi(\mathbf{x}) + b_2 &\leq -1 \\
&\vdots \\
(\mathbf{w}^k)^T \phi(\mathbf{x}) + b_k &\leq -1
\end{aligned}
$$

Multi-class Classification

- One-against-one: train k(k − 1)/2 binary SVMs

  (1,2), (1,3), . . . , (1,k), (2,3), (2,4), . . . , (k−1,k)
- Example: if 4 classes→6 binary SVMs

$y_i = 1$	$y_i = -1$	Decision functions
class 1	class 2	$f^{12}(\mathbf{x}) = (\mathbf{w}^{12})^T\mathbf{x} + b^{12}$
class 1	class 3	$f^{13}(\mathbf{x}) = (\mathbf{w}^{13})^T\mathbf{x} + b^{13}$
class 1	class 4	$f^{14}(\mathbf{x}) = (\mathbf{w}^{14})^T\mathbf{x} + b^{14}$
class 2	class 3	$f^{23}(\mathbf{x}) = (\mathbf{w}^{23})^T\mathbf{x} + b^{23}$
class 2	class 4	$f^{24}(\mathbf{x}) = (\mathbf{w}^{24})^T\mathbf{x} + b^{24}$
class 3	class 4	$f^{34}(\mathbf{x}) = (\mathbf{w}^{34})^T\mathbf{x} + b^{34}$

Multi-class Classification

- For a testing data, predict all binary SVMs

Classes		winner
1	2	1
1	3	1
1	4	1
2	3	2
2	4	4
3	4	3

- Select the one with the largest vote

class	1	2	3	4
# votes	3	1	1	1

- May use decision values as well

Multi-class Classification

- There are many other methods
- A comparison in [Hsu and Lin, 2002]
- Accuracy similar for many problems
- But 1-against-1 fastest for training
- Assume the SVM optimization with size n is $\mathcal{O}(n^d)$
- 1 vs. all
  – k problems, each has N data $k\mathcal{O}(N^d)$
- 1 vs. 1
  – k(k – 1)/2 problems, each 2N/k data on average

$$\frac{k(k-1))}{2}\mathcal{O}((\frac{2N}{k})^d)$$

Error-Correcting Output Coding

- **Motivation:**
  Applying binary classifiers to multiclass problems
- **Train:** Repeat L times:
  - Form a binary problem by randomly assigning classes to "superclasses" 0 and 1
    E.g.: A, B, D $\rightarrow$ 0; C, E $\rightarrow$ 1
  - Apply binary learner to binary problem
- Each class is represented by a binary vector
- **Test:**
  - Apply each classifier to test example, forming vector of predictions $\mathbf{P}$
  - Predict class whose vector is closest to $\mathbf{P}$ (Hamming)

Proof of SVM Dual Problem

- We first introduce "Lagrangian Methods"

Let $P(b)$ denote the optimization problem

$$\text{minimize } f(x)\,, \quad \text{subject to } h(x) = b,\ x \in X\,.$$

Let $x \in X(b) = \{x \in X : h(x) = b\}$. We say that x is **feasible** if $x \in X(b)$.

- Define the Lagrangian as:

$$L(x, \lambda) = f(x) - \lambda^\top (h(x) - b)\,.$$

λ: Lagrangian multipliers

- Because

$$\phi(b) = \inf_{x \in X(b)} f(x) \quad \text{and} \quad g(\lambda) = \inf_{x \in X} L(x, \lambda)\,. \qquad \phi(b) = \inf_{x \in X(b)} L(x, \lambda) \geq \inf_{x \in X} L(x, \lambda) = g(\lambda)\,.$$

$g(\lambda)$ is a lower bound on $\phi(b)$, i.e., a lower bound on the primal solution $P(b)$

- Dual problem

$$\text{maximize } g(\lambda)\,, \quad \text{subject to } \lambda \in Y\,,$$

$$\text{where } Y = \{\lambda : g(\lambda) > -\infty\}$$

More details at:
http://www.statslab.cam.ac.uk/~rrw1/mor/s.pdf

Proof of SVM Dual Problem

- Lagrange Function

$$L(w, b, \alpha) = \frac{1}{2}\|w\|^2 + \sum_{i=1}^{m} \alpha_i(1 - y_i(\langle w, x_i \rangle + b))$$

- Saddle point condition

$$\partial_w L(w, b, \alpha) = w - \sum_{i=1}^{m} \alpha_i y_i x_i = 0 \iff w = \sum_{i=1}^{m} \alpha_i y_i x_i$$

$$\partial_b L(w, b, \alpha) = -\sum_{i=1}^{m} \alpha_i y_i x_i = 0 \iff \sum_{i=1}^{m} \alpha_i y_i = 0$$

To obtain the dual optimization problem we have to substitute the values of w and b into L. Note that the dual variables α_i have the constraint $\alpha_i \geq 0$.

Proof of SVM Dual Problem (cont')

- **Dual Problem**
  - After substituting in terms for b,w, the Lagrange function becomes

$$\text{maximize} \quad -\frac{1}{2}\sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle + \sum_{i=1}^{m} \alpha_i$$

$$\text{subject to} \quad \sum_{i=1}^{m} \alpha_i y_i = 0 \text{ and } \alpha_i \geq 0 \text{ for all } 1 \leq i \leq m$$

  - which can be simplified as (note: we use N=m below)

$$\max_{\alpha_i \in [0,C]} \left\{ \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i^\top \mathbf{x}_j) : \sum_{i=1}^{N} \alpha_i y_i = 0 \right\}$$

Karush–Kuhn–Tucker (KKT) conditions

- At optimal solution
  Constraint · Lagrange Multiplier = 0

- In our context this means

$$\alpha_i(1 - y_i(\langle w, x_i \rangle + b)) = 0.$$

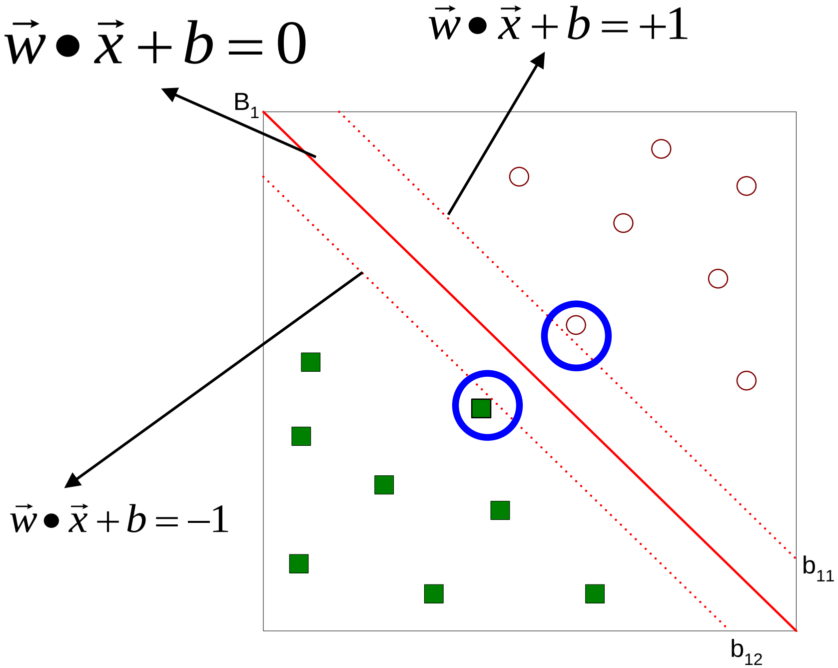
- Equivalently we have

$$\alpha_i \neq 0 \Longrightarrow y_i\left(\langle w, x_i \rangle + b\right) = 1$$

- Only points at the decision boundary can contribute to the solution.

Dual Form of SVM

$$\max_{\alpha_i \in [0,C]} \left\{ \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i^\top \mathbf{x}_j) : \sum_{i=1}^{N} \alpha_i y_i = 0 \right\}$$



$$\alpha_i(1 - y_i(\langle w, x_i \rangle + b)) = 0.$$

$$\text{Support vectors: } \alpha_i > 0$$

How to find b?

$$y_i(\mathbf{w} \cdot \mathbf{x_i} - b) = 1$$

$$\mathbf{w} \cdot \mathbf{x_i} - b = 1/y_i = y_i \iff b = \mathbf{w} \cdot \mathbf{x_i} - y_i$$

$$b = \frac{1}{N_{SV}} \sum_{i=1}^{N_{SV}} (\mathbf{w} \cdot \mathbf{x_i} - y_i)$$

Dual Form of SVM:

- Solving the Primal equals to solving the Dual

$$\max_{\alpha_i \in [0,C]} \left\{ \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i^\top \mathbf{x}_j) : \sum_{i=1}^{N} \alpha_i y_i = 0 \right\}$$

The derivation of the dual forms can be found in the appendix

- Decision function of SVM:

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i \qquad \mathbf{x}_i \text{ is a support vector if } \alpha_i > 0$$

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b = \sum_{i=1}^{N} \alpha_i y_i (\mathbf{x}_i^\top \mathbf{x}) + b$$

SVM: Nonlinear Case

- The dual problem

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} - \mathbf{e}^T \boldsymbol{\alpha} \\ \text{subject to} \quad & 0 \le \alpha_i \le C, i = 1, \ldots, l \\ & \mathbf{y}^T \boldsymbol{\alpha} = 0, \end{aligned}$$

where $Q_{ij} = y_i y_j \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ and $\mathbf{e} = [1, \ldots, 1]^T$

- The optimal solution $\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \Phi(\mathbf{x}_i)$

$$f(\mathbf{x}) = \mathbf{w}^\top \Phi(\mathbf{x}) + b = \sum_{i=1}^{N} \alpha_i y_i \Phi(\mathbf{x}_i) \Phi(\mathbf{x}) + b$$

Gaussian/RBF Kernel

- The kernel can be inner product in the infinite dimensional space. Assume x∈R. $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$

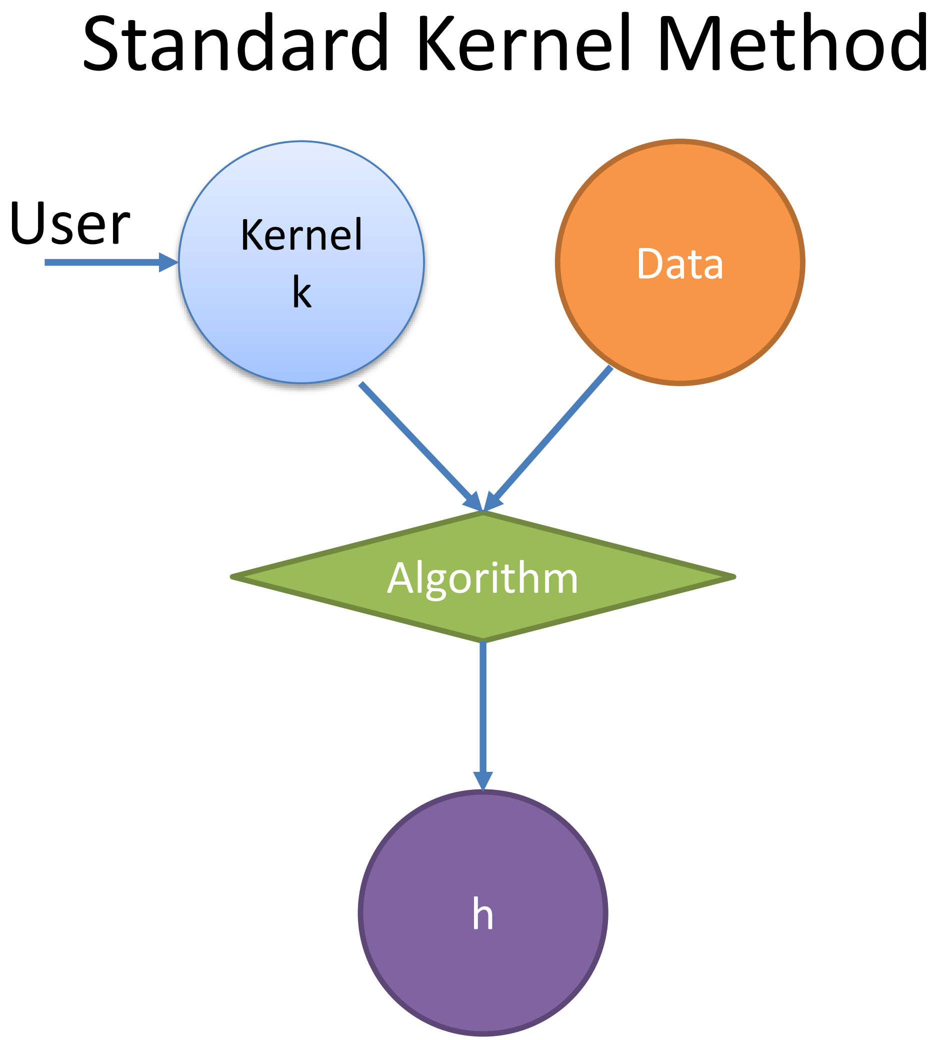
$$e^{-\gamma \|x_i - x_j\|^2} = e^{-\gamma (x_i - x_j)^2} = e^{-\gamma x_i^2 + 2\gamma x_i x_j - \gamma x_j^2}$$

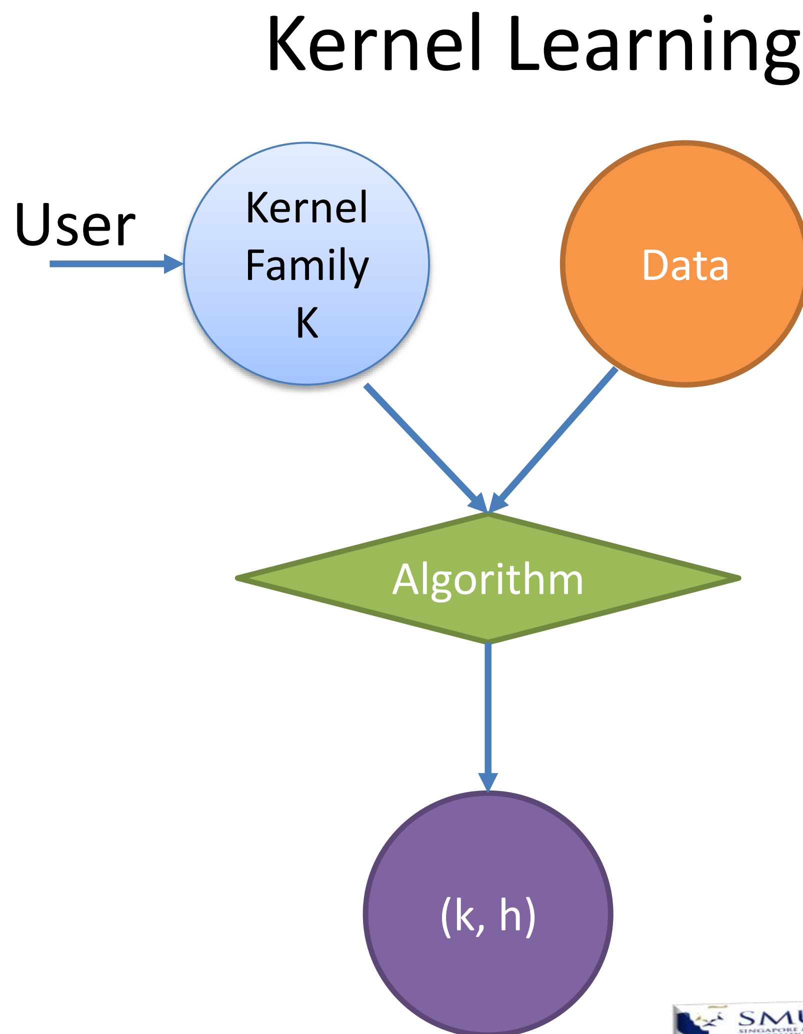
$$= e^{-\gamma x_i^2 - \gamma x_j^2} \left(1 + \frac{2\gamma x_i x_j}{1!} + \frac{(2\gamma x_i x_j)^2}{2!} + \frac{(2\gamma x_i x_j)^3}{3!} + \cdots \right)$$

$$= e^{-\gamma x_i^2 - \gamma x_j^2} \left(1 \cdot 1 + \sqrt{\frac{2\gamma}{1!}} x_i \cdot \sqrt{\frac{2\gamma}{1!}} x_j + \sqrt{\frac{(2\gamma)^2}{2!}} x_i^2 \cdot \sqrt{\frac{(2\gamma)^2}{2!}} x_j^2 \right.$$
$$\left. + \sqrt{\frac{(2\gamma)^3}{3!}} x_i^3 \cdot \sqrt{\frac{(2\gamma)^3}{3!}} x_j^3 + \cdots \right) = \phi(x_i)^T \phi(x_j),$$

$$\phi(x) = e^{-\gamma x^2} \left[1, \sqrt{\frac{2\gamma}{1!}} x, \sqrt{\frac{(2\gamma)^2}{2!}} x^2, \sqrt{\frac{(2\gamma)^3}{3!}} x^3, \cdots \right]^T.$$

Kernel Learning





Kernel Learning

- What if we have multiple kernel functions

$$\kappa_1(\mathbf{x},\mathbf{x}), \kappa_2(\mathbf{x},\mathbf{x}), \ldots, \kappa_m(\mathbf{x},\mathbf{x})$$

- Which one is the best ?
- How can we combine multiple kernels ?

$$\max_{\alpha_i \in [0,C]} \left\{ \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j) : \sum_{i=1}^{N} \alpha_i y_i = 0 \right\}$$

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b = \sum_{i=1}^{N} \alpha_i \kappa(\mathbf{x}_i, \mathbf{x}) + b$$

Kernel Learning

- **Multiple Kernel Learning (MKL)**
  - Learning an optimal linear combination of multiple kernels and the kernel classifier
- **The MKL Framework**
  - Extract features from all available data sources
  - Construct kernel functions/matrices
    - Different features
    - Different kernel types
    - Different kernel parameters
  - Find the optimal kernel combination and the kernel classifier

Kernel Learning

- **Multiple Kernel Learning (MKL)**
  - Formulation for binary classification

$$\kappa(\mathbf{x}, \mathbf{x}; \gamma) = \sum_{k=1}^{m} \gamma_k \kappa_k(\mathbf{x}, \mathbf{x})$$

$$\gamma = (\gamma_1, \ldots, \gamma_m) \in \Delta = \left\{ \gamma \in \mathbb{R}_+^m : \sum_{k=1}^{m} \gamma_k = 1 \right\}$$

$$\min_{\gamma \in \Delta} \max_{\alpha_i \in [0,C]} \left\{ \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j; \gamma) : \sum_{i=1}^{N} \alpha_i y_i = 0 \right\}$$

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b = \textstyle\sum_{i=1}^{N} \alpha_i \kappa(\mathbf{x}_i, \mathbf{x}; \gamma) + b$$

MKL Solutions

- Semi-infinite Programming (SIP) (Sonnenberg et al. 2005)
- Subgradient descend (Rakotomamonjy et al. 2007)
- Level-Method (Xu et al. 2009)
- Mirror-descend (Aflalo et al .2011)
- Alternating update method (Group Lasso) (Xu et al. 2010)
- Online MKL formulation (Hoi et al. 2013)

Curse of Kernelization

- Challenge
  - Training kernel classifiers is often much more computationally expensive
  - For kernel SVM, if one solves it by typical QP solvers, it will need O(N^3). Even for faster solvers (SMO) or others, it typically needs at least O(N^2) time cost.
  - But linear classifiers can be trained in much fasters, typically in linear time O(N)
- Question
  - How to train kernel machines for large-scale datasets?

Kernel Approximation

- Our goal
  - To construct a new representation $\mathbf{z}(\mathbf{x}) \in \mathbb{R}^D$
    so that : $\kappa(\mathbf{x}_i, \mathbf{x}_j) \approx \mathbf{z}(\mathbf{x}_i)^\top \mathbf{z}(\mathbf{x}_j)$
- Linear model
  - The hypothesis can be rewritten:

$$f(\mathbf{x}) = \sum_{i=1}^{B} \alpha_i \kappa(\mathbf{x}_i, \mathbf{x}) \approx \sum_{i=1}^{B} \alpha_i \mathbf{z}(\mathbf{x}_i)^\top \mathbf{z}(\mathbf{x}) = \mathbf{w}^\top \mathbf{z}(\mathbf{x})$$

    where $\mathbf{w}^\top = \sum_{i=1}^{B} \alpha_i \mathbf{z}(\mathbf{x}_i)$
  - Then apply linear classifiers on the new representation **z**
- Two methods
  - Kernel Functional Approximation: Fourier method
  - Kernel Matrix Approximation: Nyström method

Fourier features for Kernel Function Approximation

- Inverse Fourier transform
  - A shift-invariant kernel is the kernel that can be written as $\kappa(\mathbf{x}_1, \mathbf{x}_2) = k(\mathbf{x}_1 - \mathbf{x}_2)$, where k is some function
  - Inverse Fourier transform

$$k(\mathbf{x}_1 - \mathbf{x}_2) = \int p(\mathbf{u}) e^{i\mathbf{u}^\top(\mathbf{x}_1 - \mathbf{x}_2)} d\mathbf{u} \;\; = \mathrm{E}_{\mathbf{u}}[e^{i\mathbf{u}^\top \mathbf{x}_1} \cdot e^{-i\mathbf{u}^\top \mathbf{x}_2}]$$

$$= \mathrm{E}_{\mathbf{u}}[[\cos(\mathbf{u}^\top \mathbf{x}_1), \sin(\mathbf{u}^\top \mathbf{x}_1)] \cdot [\cos(\mathbf{u}^\top \mathbf{x}_2), \sin(\mathbf{u}^\top \mathbf{x}_2)]]$$

- Random Fourier features

$$\mathbf{z}_t(\mathbf{x}) = (\sin(\mathbf{u}_1^\top \mathbf{x}), \cos(\mathbf{u}_1^\top \mathbf{x}), ..., \sin(\mathbf{u}_D^\top \mathbf{x}), \cos(\mathbf{u}_D^\top \mathbf{x}))$$

where $\mathbf{u}_1, ... \mathbf{u}_D$ are randomly generated

Nyström Method for Kernel Matrix Approximation

- Kernel Matrix Decompostion
  - Given $\mathbf{K} \in \mathbb{R}^{T\times T}$ with rank *r*, then $\mathbf{K} = \mathbf{V}\mathbf{D}\mathbf{V}^\top$, where $\mathbf{V}$ are orthogonal $\mathbf{D} = \mathrm{diag}(\sigma_1, \ldots, \sigma_r,)$
  - For $k < r$, $\mathbf{K}_k = \sum_{i=1}^{k} \sigma_i V_i V_i^\top = \mathbf{V}_k \mathbf{D}_k \mathbf{V}_k^\top$ is the best rank-k approximation of $\mathbf{K}$
- Nyström method
  - Given $\mathbf{K}$, randomly sample $B \ll T$ columns to form a matrix $\mathbf{C} \in \mathbb{R}^{T\times B}$, and then derive a smaller kernel $\mathbf{W} \in \mathbb{R}^{B\times B}$ on the *B* instances
  - $\hat{\mathbf{K}} = \mathbf{C}\mathbf{W}_k^{+}\mathbf{C}^\top \approx \mathbf{K}$, where ‘+’ is pseudo inverse

Nyström Method for Kernel Matrix Approximation

- Use a set of B instances randomly selected from the training data set to approximate the kernel value of any other instances

$$\hat{\kappa}(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{C}_i \mathbf{V}_k \mathbf{D}_k^{-\frac{1}{2}})(\mathbf{C}_j \mathbf{V}_k \mathbf{D}_k^{-\frac{1}{2}})^\top =$$

$$([\kappa(\mathbf{x}_1, \mathbf{x}_i), \ldots, \kappa(\mathbf{x}_B, \mathbf{x}_i)]\mathbf{V}_k \mathbf{D}_k^{-\frac{1}{2}})(\kappa(\mathbf{x}_1, \mathbf{x}_j), \ldots, \kappa(\mathbf{x}_B, \mathbf{x}_j)\mathbf{V}_k \mathbf{D}_k^{-\frac{1}{2}})^\top$$

- For any instance x, the new representation can be written as

$$\mathbf{z}_t(\mathbf{x}) = [\kappa(\mathbf{x}_1, \mathbf{x}), \ldots, \kappa(\mathbf{x}_B, \mathbf{x})]\mathbf{V}_k \mathbf{D}_k^{-\frac{1}{2}})$$