UVA CS 4501: Machine Learning

Lecture 17: Support Vector Machine (nonlinear) Kernel Trick and in Practice

Dr. Yanjun Qi

University of Virginia

Department of
Computer Science

Where are we ? ➔ Five major sections of this course

- ❑ ~~Regression (supervised)~~
- ❑ Classification (supervised)
- ❑ Unsupervised models
- ❑ Learning theory
- ❑ Graphical models

Today

- ❑ Support Vector Machine (SVM)
  - ✓ History of SVM
  - ✓ Large Margin Linear Classifier
  - ✓ Define Margin (M) in terms of model parameter
  - ✓ Optimization to learn model parameters (w, b)
  - ✓ Non linearly separable case
  - ✓ Optimization with dual form
  - ✓ Nonlinear decision boundary
  - ✓ Practical Guide

Dual SVM for linearly separable case – Training / Testing

Our dual target function: $\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{\mathrm{i,j}} \alpha_i \alpha_j \mathrm{y}_\mathrm{i} y_j \mathbf{x}_\mathbf{i}^T \mathbf{x}_\mathbf{j}$

$$\sum_i \alpha_i \mathrm{y}_\mathrm{i} = 0$$

$$\alpha_i \geq 0 \qquad \forall i$$

Dot product for all training samples

Dot product with ("all" ??) training samples

To evaluate a new sample x_{ts} we need to compute:

$$\mathrm{w}^\mathrm{T} x_{ts} + b = \sum_\mathrm{i} \alpha_i \mathrm{y}_\mathrm{i} \mathbf{x}_\mathbf{i}^T \mathbf{x}_{ts} + b$$

$$\widehat{y_{ts}} = \mathrm{sign}\left(\sum_{i \in SupportVectors} \alpha_i y_i \left(\mathbf{x}_i^T \mathbf{x}_{ts} \right) + b \right)$$

$$\max_\alpha \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x_i}^T \mathbf{x_j}$$

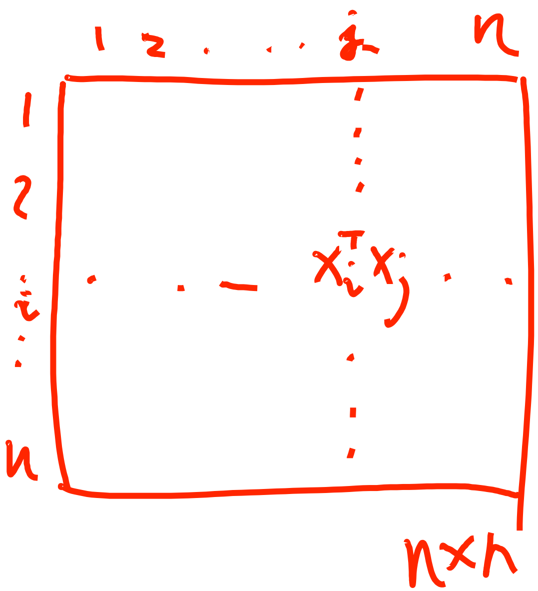
$$\sum_i \alpha_i y_i = 0$$

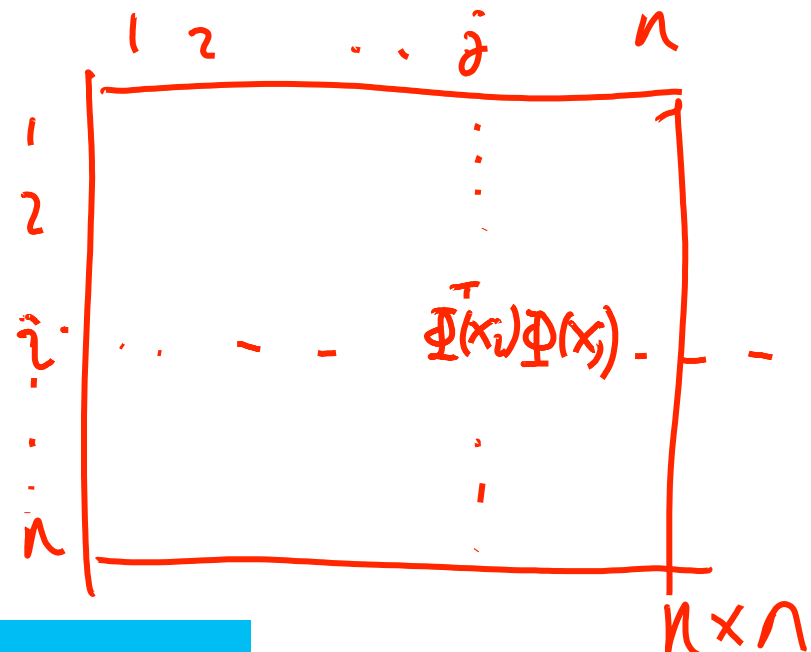
$$C > \alpha_i \geq 0, \forall i$$

$$\max_\alpha \sum_i \alpha_i - \sum_{i,j} \alpha_i \alpha_j y_i y_j \Phi(\mathbf{x_i})^T \Phi(\mathbf{x_j})$$

$$\sum_i \alpha_i y_i = 0$$

$$C > \alpha_i \geq 0, \forall i$$





Training

$$\mathrm{w}^T x_{ts} + b = \sum_{\mathrm{i}} \alpha_i \mathrm{y}_\mathrm{i} \mathbf{x}_\mathbf{i}^T \mathbf{x}_{ts} + b$$

$$\widehat{y_{ts}} = \mathrm{sign}\left(\sum_{i \in SupportVectors} \alpha_i y_i \left(\mathbf{x}_i^T \mathbf{x}_{ts} \right) + b \right)$$

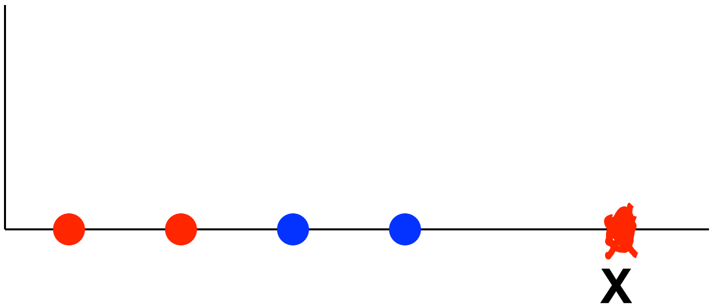
$$\Rightarrow \sum_{SV} \alpha_i y_i \underbrace{\Phi(x_i)\Phi(x_{ts})} + b$$

$$\begin{array}{c|c|} & x_{ts} \\ \hline 1 & \\ 2 & \\ 3 & \\ \vdots & \\ i & x_i^T x_{ts} \\ \vdots & \\ n & \\ \hline \end{array} \quad n \times 1$$

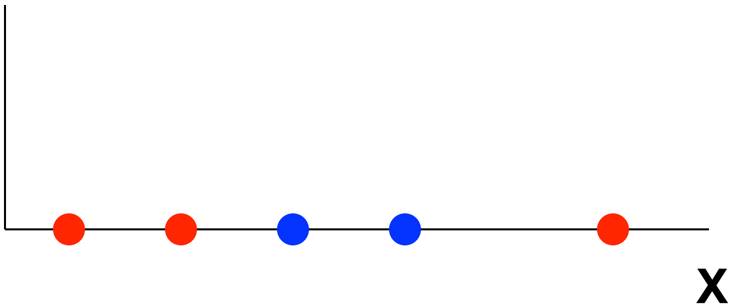
Testing

Classifying in 1-d

Can an SVM correctly classify this data?



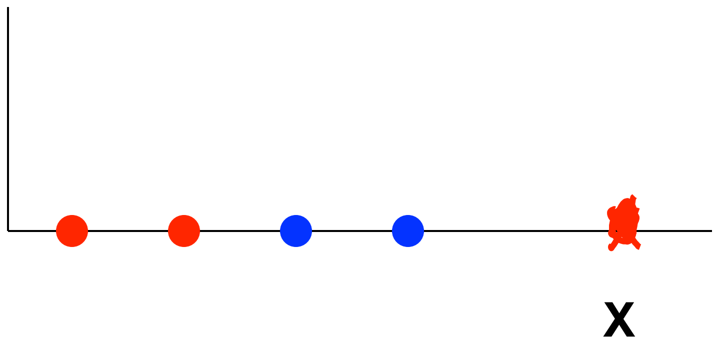
What about this?



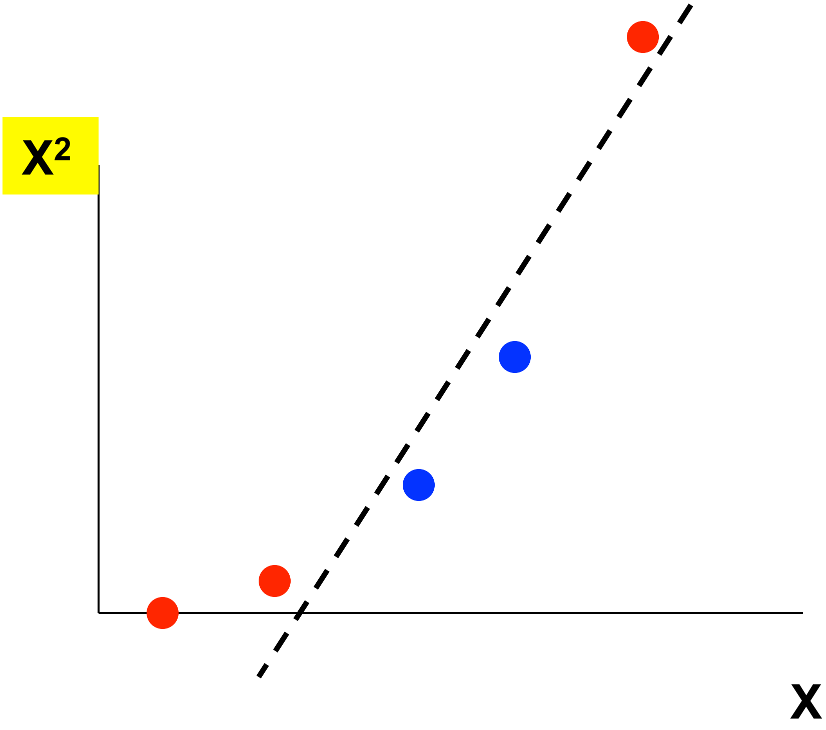
Classifying in 1-d

→ separable
→ nonlinear

Can an SVM correctly classify this data?

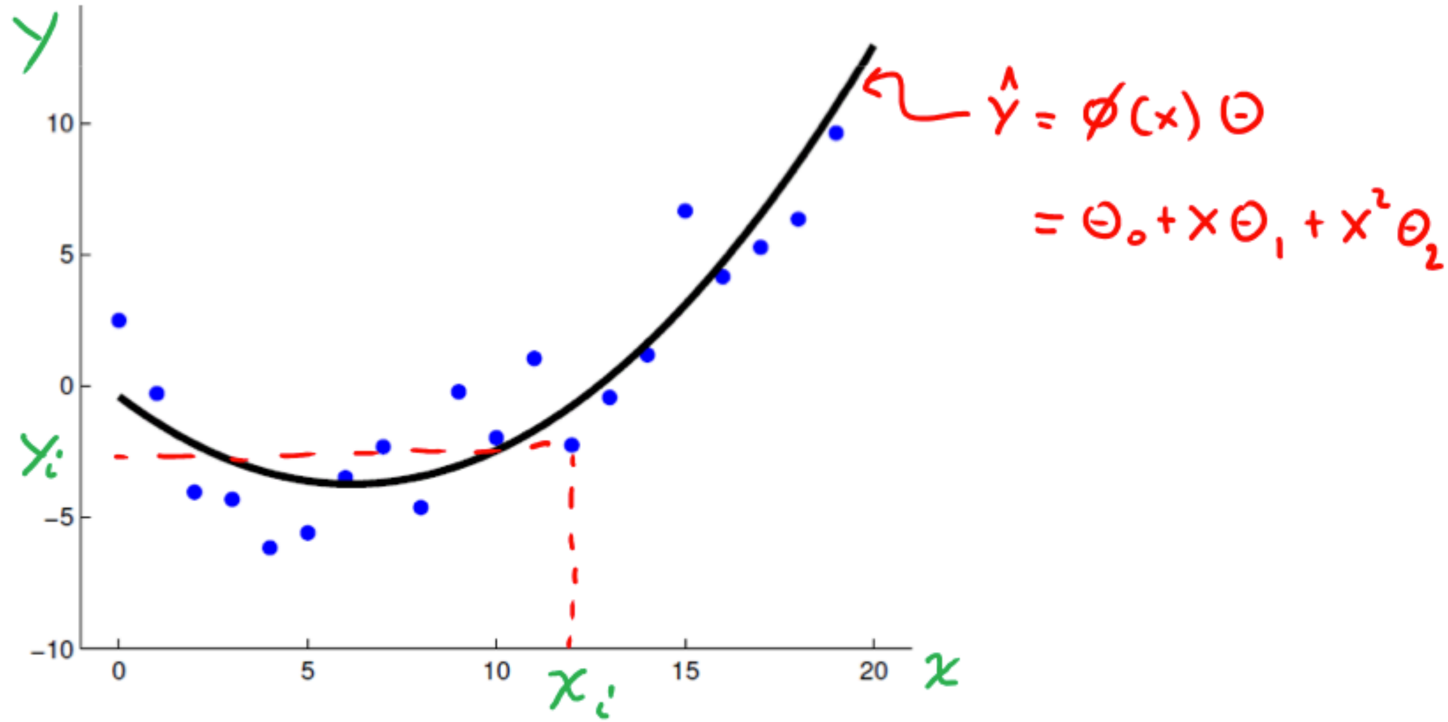


And now? (extend with polynomial basis)



RECAP: **Polynomial regression**

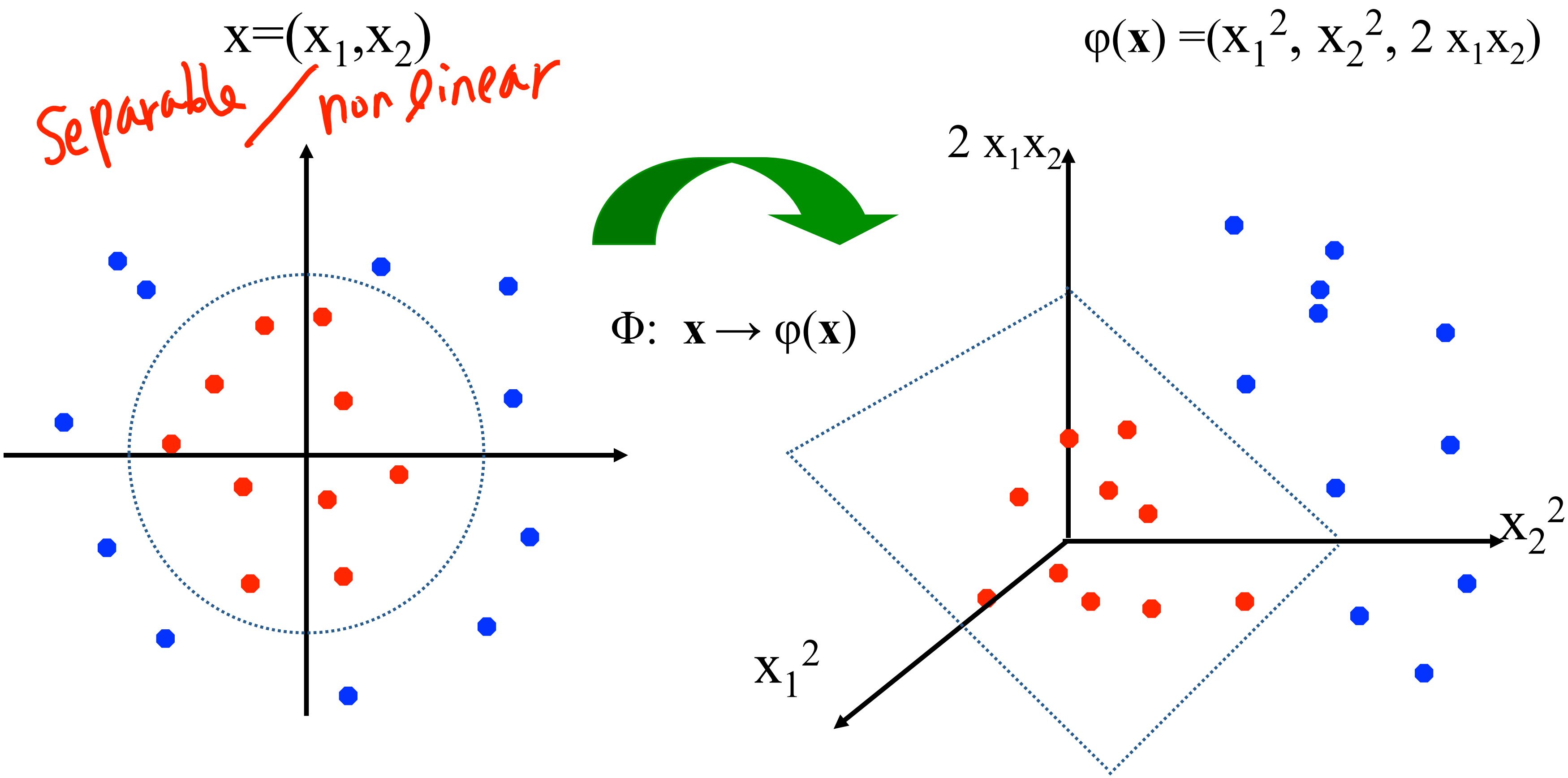
For example, $\phi(x) = [1, x, x^2]$



Dr. Nando de Freitas's tutorial slide

Non-linear SVMs: 2D

- The original input space (x) can be mapped to some higher-dimensional feature space ($\phi(\mathbf{x})$)where the training set is separable:

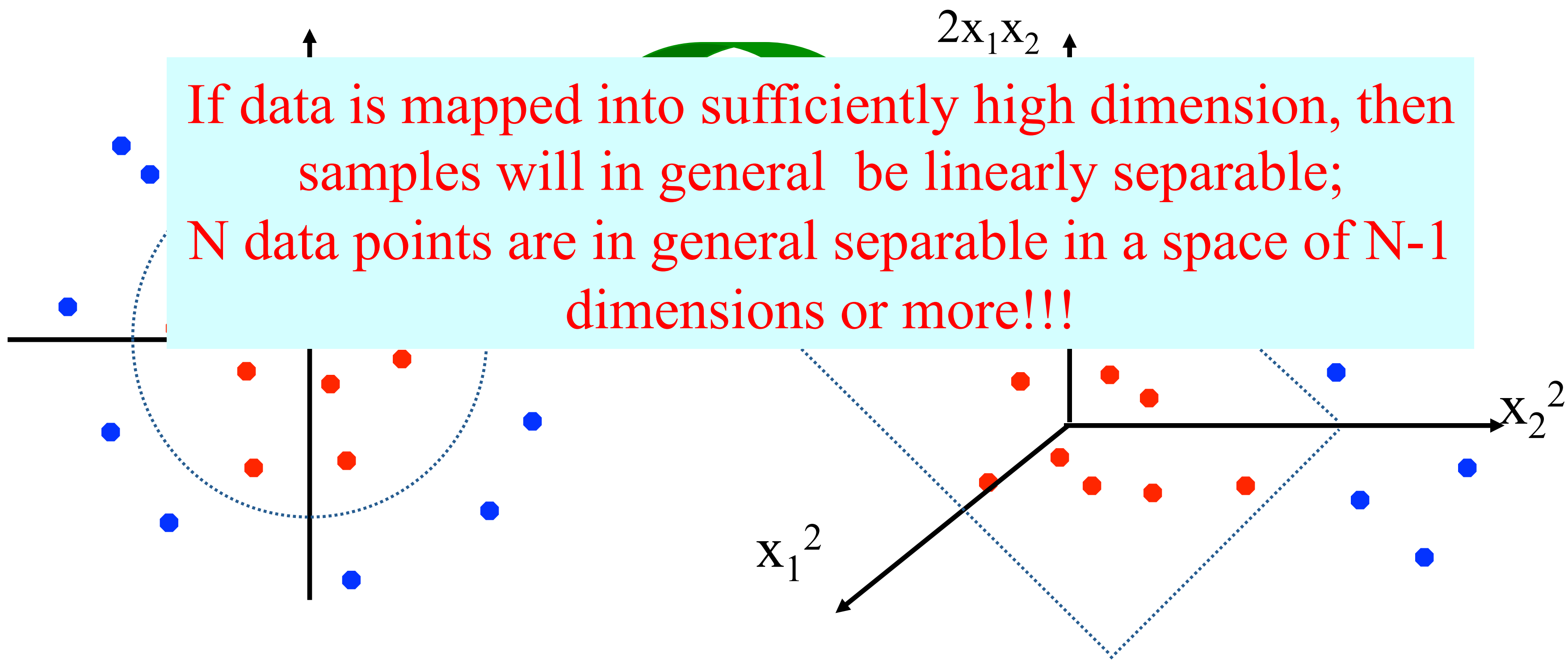


This slide is courtesy of *www.iro.umontreal.ca/~pift6080/documents/papers/**svm**_tutorial.**ppt***

Non-linear SVMs: 2D

- The original input space (x) can be mapped to some higher-dimensional feature space ($\phi(\mathbf{x})$)where the training set is separable:

$x=(x_1,x_2)$ $\qquad\qquad$ $\varphi(\mathbf{x}) =(x_1^2,x_2^2,\ 2x_1x_2)$



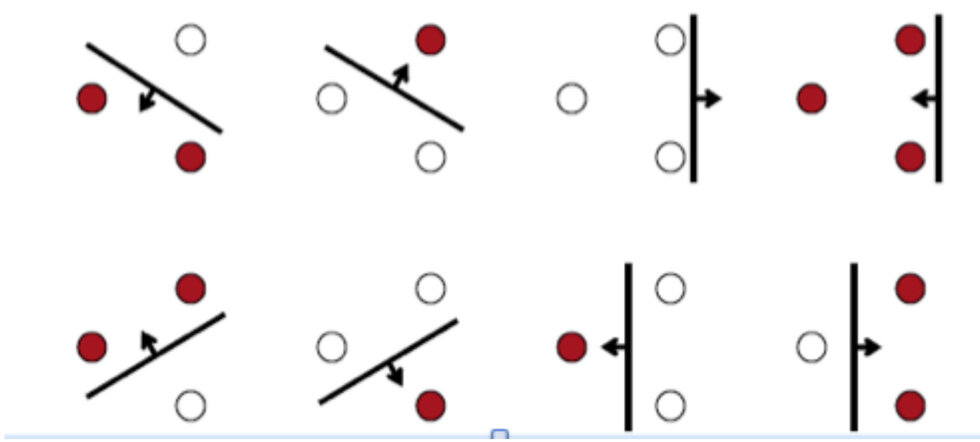
If data is mapped into sufficiently high dimension, then samples will in general be linearly separable;
N data points are in general separable in a space of N-1 dimensions or more!!!

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

A little bit theory: Vapnik-Chervonenkis (VC) dimension

If data is mapped into sufficiently high dimension, then samples will in general be linearly separable;
N data points are in general separable in a space of N-1 dimensions or more!!!

- **VC dimension of the set of oriented lines in R^2 is 3**
  - It can be shown that the VC dimension of the family of oriented separating hyperplanes in R^N is at least N+1

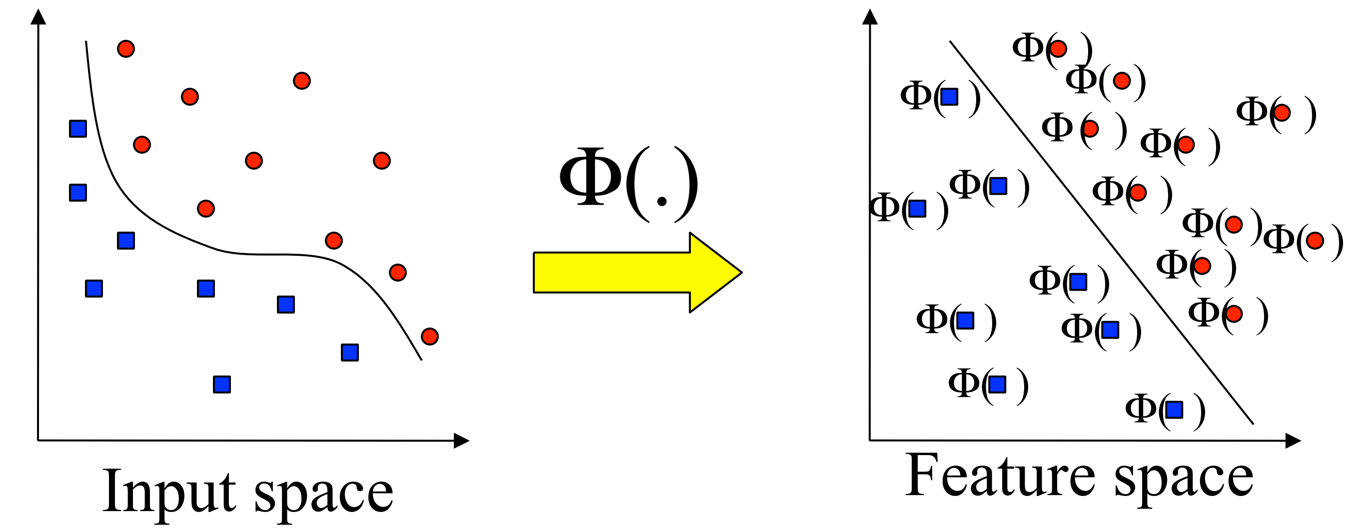


If data is mapped into sufficiently high dimension, then samples will in general be linearly separable;

N data points are in general separable in a space of N-1 dimensions or more!!!

$X \longrightarrow \Phi(X)$

- Possible problems
  - High computation burden due to high-dimensionality
  - Many more parameters to estimate



SVM solves these two issues simultaneously

- “Kernel tricks” for efficient computation
- Dual formulation only assigns parameters to samples, not to features

- SVM solves these two issues simultaneously
  - "Kernel tricks" for efficient computation
  - Dual formulation only assigns parameters to samples, not features

(1). "Kernel tricks" for efficient computation

$\cancel{\Phi(X)}$

Never represent features explicitly

- ☐ Compute dot products in closed form

Very interesting theory – Reproducing Kernel Hilbert Spaces

- ☐ Not covered in detail here

$$K(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$ is called the kernel function.

$$\begin{cases} x \in R^p \\ z \in R^p \end{cases}$$

- Linear kernel (we've seen it) $K(\mathbf{x}, z) = \mathbf{x}^T z$

- Polynomial kernel (we will see an example)

$$K(\mathbf{x}, z) = \left(1 + \mathbf{x}^T z\right)^d = \Phi_p(x)^T \underbrace{\Phi_p(z)}_{p_\Phi \to O(p^d)}$$

where d = 2, 3, … To get the feature vectors we concatenate all dth order polynomial terms of the components of x (weighted appropriately)

- Radial basis kernel

$$K(\mathbf{x}, z) = \exp\left(-r\left\|\mathbf{x} - z\right\|^2\right) = \underbrace{\Phi_r(x)^T}_{p_\Phi = \infty} \Phi_r(z)$$

In this case., r is hyperpara. The feature space of the RBF kernel has an infinite number of dimensions

Never represent features explicitly

☐ Compute dot products with a closed form

Very interesting theory – Reproducing Kernel Hilbert Spaces

☐ Not covered in detail here

Example: Quadratic kernels

$$K(\mathbf{x},z)=\left(1+\mathbf{x}^T z\right)^d$$

$(1+x^Tz)^2$

$$\boxed{K(\mathbf{x},z) := \Phi(\mathbf{x})^T \Phi(z)}$$

- Consider all quadratic terms for $x_1, x_2 \ldots x_p$

$$\Phi(x)=\begin{bmatrix} 1 \\ \sqrt{2}x_1 \\ \vdots \\ \sqrt{2}x_p \\ x_1^2 \\ \vdots \\ x_p^2 \\ \sqrt{2}x_1x_2 \\ \vdots \\ \sqrt{2}x_{p-1}x_p \end{bmatrix}$$

$$\max_\alpha \sum_i \alpha_i - \sum_{\mathrm{i,j}} \alpha_i\alpha_j \mathrm{y}_\mathrm{i} y_j \Phi(\mathbf{x_i})^T \Phi(\mathbf{x_j})$$

$$\sum_i \alpha_i \mathrm{y}_\mathrm{i} = 0$$

$$\alpha_i \geq 0 \qquad \forall i$$

$$K(\mathbf{x},z)=\left(1+\mathbf{x}^T z\right)^2, \quad [d=2], \quad [p=2] \quad \begin{bmatrix} x=(x_1,x_2) \\ z=(z_1,z_2) \end{bmatrix}$$

$$K(x,z)=\left(1+x_1z_1+x_2z_2\right)^2 \Rightarrow O(p)$$

$$O(p^2) \quad = \left(1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2\right)^T \left(1, \sqrt{2}z_1, \sqrt{2}z_2, z_1^2, z_2^2, \sqrt{2}z_1z_2\right)$$

$$= \Phi(x)^T \Phi(z)$$

The kernel trick

p^d*n^2 operations if using the basis function representations in building a poly-kernel matrix

So, if we define the **kernel function** as follows, there is no need to carry out basis function explicitly

$$K(\mathbf{x},z)=(x^Tz+1)^{d=2}$$

$p*n^2$ operations in building a poly-kernel matrix for training

$$\max_\alpha \sum_i \alpha_i - \frac{1}{2}\sum_{\mathrm{i,j}} \alpha_i \alpha_j \mathrm{y}_{\mathrm{i}} y_j K(\mathbf{x_i},\mathbf{x_j})$$

$$\sum_i \alpha_i \mathrm{y}_{\mathrm{i}} = 0$$

$$C > \alpha_i \geq 0, \forall i \in train$$

Summary: Modification Due to Kernel Trick

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

Original Linear

$$\max_\alpha \sum_i \alpha_i - \frac{1}{2}\sum_{\mathrm{i,j}} \alpha_i \alpha_j \mathrm{y}_\mathrm{i} y_j \mathbf{x_i}^T \mathbf{x_j}$$

$$\sum_i \alpha_i \mathrm{y}_\mathrm{i} = 0$$

$$C > \alpha_i \geq 0, \forall i \in train$$

With kernel function - nonlinear

$$\max_\alpha \sum_i \alpha_i - \frac{1}{2}\sum_{\mathrm{i,j}} \alpha_i \alpha_j \mathrm{y}_\mathrm{i} y_j K(\mathbf{x_i}, \mathbf{x_j})$$

$$\sum_i \alpha_i \mathrm{y}_\mathrm{i} = 0$$

$$C > \alpha_i \geq 0, \forall i \in train$$

Summary: Modification Due to Kernel Function

- For testing, the new data **x_ts**

Original Linear

$$\widehat{y_{ts}} = \operatorname{sign}\left(\sum_{\mathrm{i} \in \mathrm{train}} \alpha_i \mathrm{y}_{\mathrm{i}} \mathbf{x}_{\mathbf{i}}{}^T \mathbf{x}_{ts} + b \right)$$

With kernel function - nonlinear

$$\widehat{y_{ts}} = \operatorname{sign}\left(\sum_{\mathrm{i} \in \mathrm{supportVectors}} \alpha_i \mathrm{y}_{\mathrm{i}} K(\mathbf{x}_{\mathbf{i}}, \mathbf{x}_{ts}) + b \right)$$

An example: Support vector machines with polynomial kernel

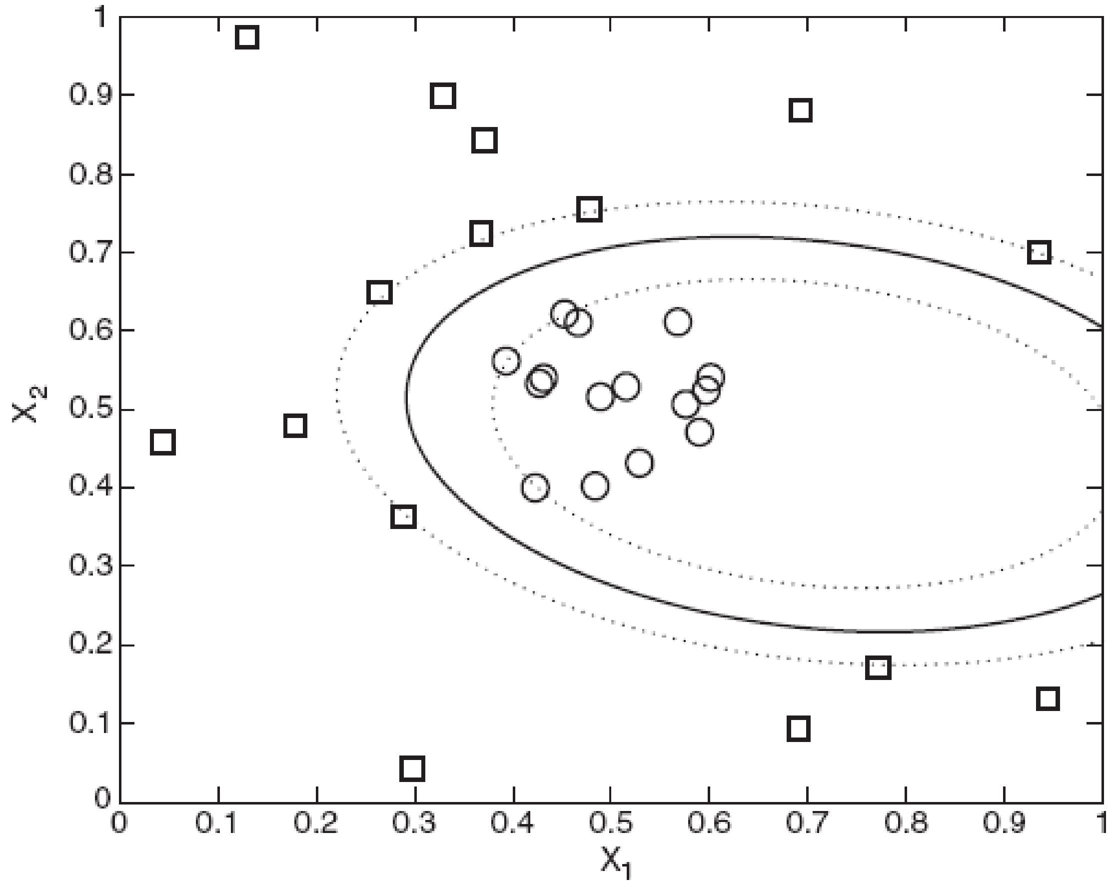


Figure 5.29. Decision boundary produced by a nonlinear SVM with polynomial kernel.

Kernel Trick: Implicit Basis Representation

- For some kernels (e.g. RBF) the implicit transform basis form \phi(**x**) is infinite-dimensional!
  - But calculations with kernel are done in original space, so computational burden and curse of dimensionality aren't a problem.

$$K(\mathbf{x}, z) = \exp\left(-r\left\|\mathbf{x} - z\right\|^2\right)$$

*p*n^2* operations in building a RBF-kernel matrix for training

➔ Gaussian RBF Kernel corresponds to an infinite-dimensional vector space.

YouTube video of Caltech: Abu-Mostafa explaining this in more detail
https://www.youtube.com/watch?v=XUj5JbQihlU&t=25m53s

Kernel Functions (Extra)

- In practical use of SVM, only the kernel function (and not basis function) is specified

- Kernel function can be thought of as a similarity measure between the input objects

- Not all similarity measure can be used as kernel function, however Mercer's condition states that any positive semi-definite kernel *K*(*x*, *y*), i.e.

$$\sum_{i,j} K(x_i, x_j) c_i c_j \geq 0$$

can be expressed as a dot product in a high dimensional space.

Kernel Matrix

- Kernel function creates the kernel matrix, which summarize all the data

$$
\begin{array}{c|ccccccc}
 & x_1 & x_2 & \cdots & j & \cdots & x_n \\
\hline
x_1 & & & & \vdots & & \\
x_2 & & & & \vdots & & \\
\vdots & & & & \vdots & & \\
i & \cdots & \cdots & , & k(x_i, x_j), & \cdots & \cdots \\
\vdots & & & & \vdots & & \\
x_n & & & & \vdots & & \\
\end{array}_{\; n \times n}
\Rightarrow K_{n \times n}
$$

[kernel matrix]

positive-semi-definite

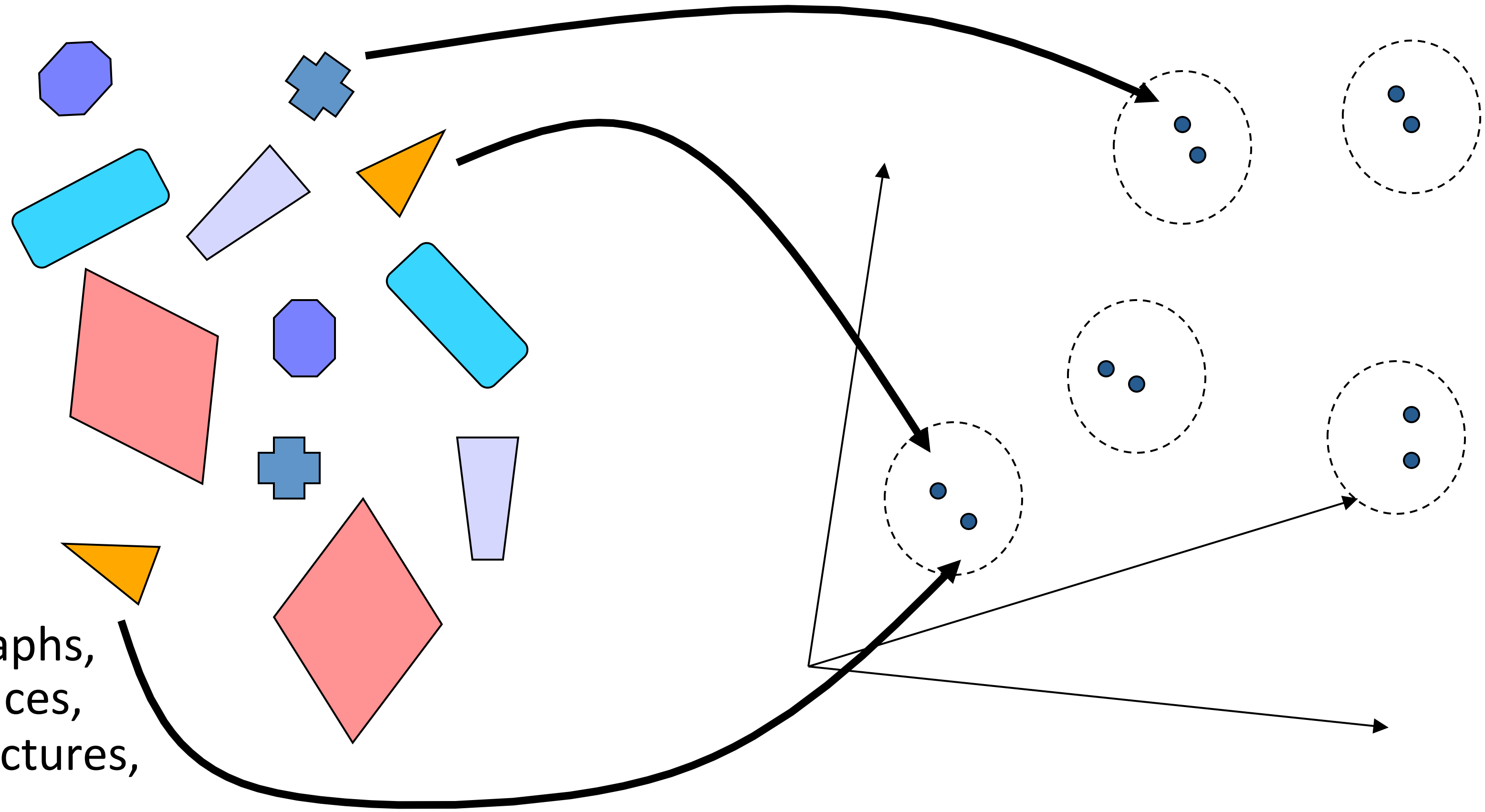
Choosing the Kernel Function

- Probably the most tricky part of using SVM.
- The kernel function is important because it creates the kernel matrix, which summarize all the data
- Many principles have been proposed (diffusion kernel, Fisher kernel, string kernel, tree kernel, graph kernel, ...)
  - Kernel trick has helped Non-traditional data like strings and trees able to be used as input to SVM, instead of feature vectors
- In practice, a low degree polynomial kernel or RBF kernel with a reasonable width is a good initial try for most applications.

Kernel trick has helped Non-traditional data like strings and trees able to be used as input to SVM, instead of feature vectors

$K(x, z)$

Vector vs. Relational data



e.g. Graphs, Sequences, 3D structures,

Original Space

Feature Space

Mercer Kernel vs. Smoothing Kernel

- The Kernels used in Support Vector Machines are different from the Kernels used in LocalWeighted /Kernel Regression.

- We can think
  - Support Vector Machines' kernels as **Mercer Kernels**
  - Local Weighted / Kernel Regression's kernels as **Smoothing Kernels**

Why do SVMs work?

- If we are using huge features spaces (e.g., with kernels), how come we are not overfitting the data?
  - ✓ Number of parameters remains the same (and most are set to 0)
  - ✓ While we have a lot of input values, at the end we only care about the support vectors and these are usually a small group of samples
  - ✓ The minimization (or the maximizing of the margin) function acts as a sort of regularization term leading to reduced overfitting

Why SVM Works? (Extra)

- Vapnik argues that the fundamental problem is not the number of parameters to be estimated. Rather, the problem is about the flexibility of a classifier

- Vapnik argues that the flexibility of a classifier should not be characterized by the number of parameters, but by the capacity of a classifier
  - This is formalized by the "VC-dimension" of a classifier

- The SVM objective can also be justified by structural risk minimization: the empirical risk (training error), plus a term related to the generalization ability of the classifier, is minimized

- Another view: the SVM loss function is analogous to ridge regression. The term $\frac{1}{2}||w||^2$ "shrinks" the parameters towards zero to avoid overfitting

Today

- Support Vector Machine (SVM)
  - ✓ History of SVM
  - ✓ Large Margin Linear Classifier
  - ✓ Define Margin (M) in terms of model parameter
  - ✓ Optimization to learn model parameters (w, b)
  - ✓ Non linearly separable case
  - ✓ Optimization with dual form
  - ✓ Nonlinear decision boundary
  - ✓ Practical Guide

Software

- A list of SVM implementation can be found at
  - http://www.kernel-machines.org/software.html

- Some implementation (such as LIBSVM) can handle multi-class classification
- SVMLight is among one of the earliest implementation of SVM
- Several Matlab toolboxes for SVM are also available

Summary: Steps for Using SVM in HW

- Prepare the feature-data matrix
- Select the kernel function to use
- Select the parameter of the kernel function and the value of *C*
  - You can use the values suggested by the SVM software, or you can set apart a validation set to determine the values of the parameter
- Execute the training algorithm and obtain the $\backslash\mathrm{alpha}_i$
- Unseen data can be classified using the $\backslash\mathrm{alpha}_i$ and the support vectors

Practical Guide to SVM

- From authors of as LIBSVM:
  - A Practical Guide to Support Vector Classification Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin, 2003-2010
  - http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf

LIBSVM

- http://www.csie.ntu.edu.tw/~cjlin/libsvm/
  - ✓Developed by Chih-Jen Lin etc.
  - ✓Tools for Support Vector classification
  - ✓Also support multi-class classification
  - ✓C++/Java/Python/Matlab/Perl wrappers
  - ✓Linux/UNIX/Windows
  - ✓SMO implementation, fast!!!

A Practical Guide to Support Vector Classification

(a) Data file formats for LIBSVM

- Training.dat

+1 1:0.708333 2:1 3:1 4:-0.320755

-1 1:0.583333 2:-1 4:-0.603774 5:1

+1 1:0.166667 2:1 3:-0.333333 4:-0.433962

-1 1:0.458333 2:1 3:1 4:-0.358491 5:0.374429

…

- Testing.dat

(b) Feature Preprocessing

- (1) Categorical Feature
  - Recommend using m numbers to represent an m-category attribute.
  - Only one of the m numbers is one, and others are zero.
  - For example, a three-category attribute such as {red, green, blue} can be represented as (0,0,1), (0,1,0), and (1,0,0)

A Practical Guide to Support Vector Classification

Feature Preprocessing

- (2) Scaling before applying SVM is very important
  - to avoid attributes in greater numeric ranges dominating those in smaller numeric ranges.
  - to avoid numerical difficulties during the calculation
  - Recommend linearly scaling each attribute to the range [1, +1] or [0, 1].

① Normalization → $\begin{cases} \text{mean } 0 \\ \text{std } 1 \end{cases}$

② Scaling → linear ⇒ $[ax+b]$

e.g. $\left[\dfrac{X - X_{min}}{\max - X_{min}} \right]$

A Practical Guide to Support Vector Classification

For i-th feature $\Rightarrow$ [Column operation on $X_{n \times p}$]

$$\begin{cases} \text{Centering}: & X_i - \bar{X}_i \Rightarrow E(X_i) = 0 \\ \text{Scaling}: & aX_i + b \Rightarrow \text{e.g.} \ \frac{X_i - \min(X_i)}{\max(X_i) - \min(X_i)} \\ \text{Normalization}: & \Rightarrow \begin{cases} E(X_i) = 0 \\ \mathrm{Var}(X_i) = 1 \end{cases} \end{cases}$$

Of course we have to use the same method to scale both training and testing data. For example, suppose that we scaled the first attribute of training data from $[-10, +10]$ to $[-1, +1]$. If the first attribute of testing data lies in the range $[-11, +8]$, we must scale the testing data to $[-1.1, +0.8]$. See Appendix B for some real examples.

If training and testing sets are separately scaled to $[0, 1]$, the resulting accuracy is lower than 70%.

```
$ ../svm-scale -l 0 svmguide4 > svmguide4.scale
$ ../svm-scale -l 0 svmguide4.t > svmguide4.t.scale
$ python easy.py svmguide4.scale svmguide4.t.scale
Accuracy = 69.2308% (216/312) (classification)
```

Using the same scaling factors for training and testing sets, we obtain much better accuracy.

```
$ ../svm-scale -l 0 -s range4 svmguide4 > svmguide4.scale
$ ../svm-scale -r range4 svmguide4.t > svmguide4.t.scale
$ python easy.py svmguide4.scale svmguide4.t.scale
Accuracy = 89.4231% (279/312) (classification)
```

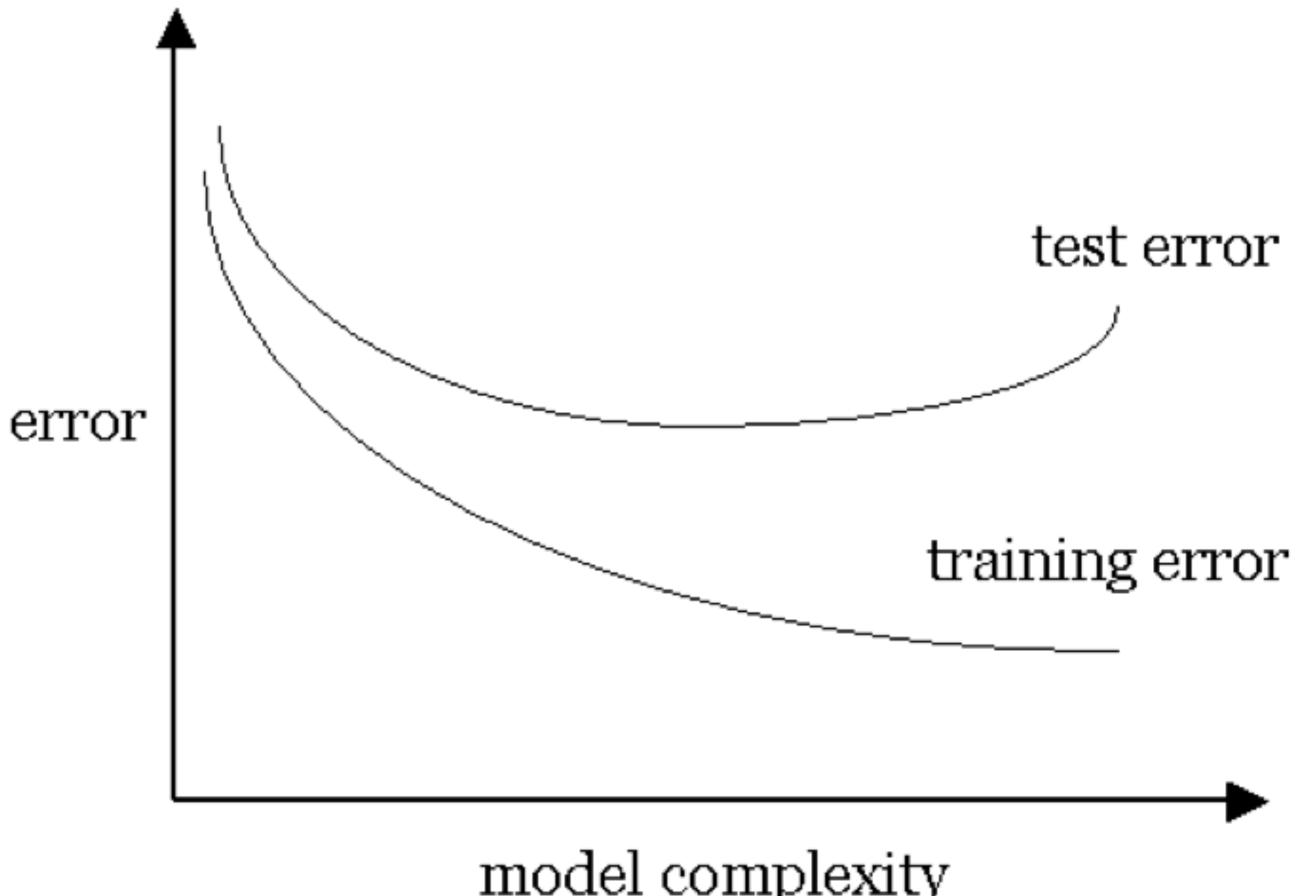
Feature Preprocessing

- (3) missing value
  - Very very tricky !
  - Easy way: to substitute the missing values by the mean value of the variable
  - A little bit harder way: imputation using nearest neighbors
  - Even more complex: e.g. EM based (beyond the scope)

A Practical Guide to Support Vector Classification

(c) Model Selection

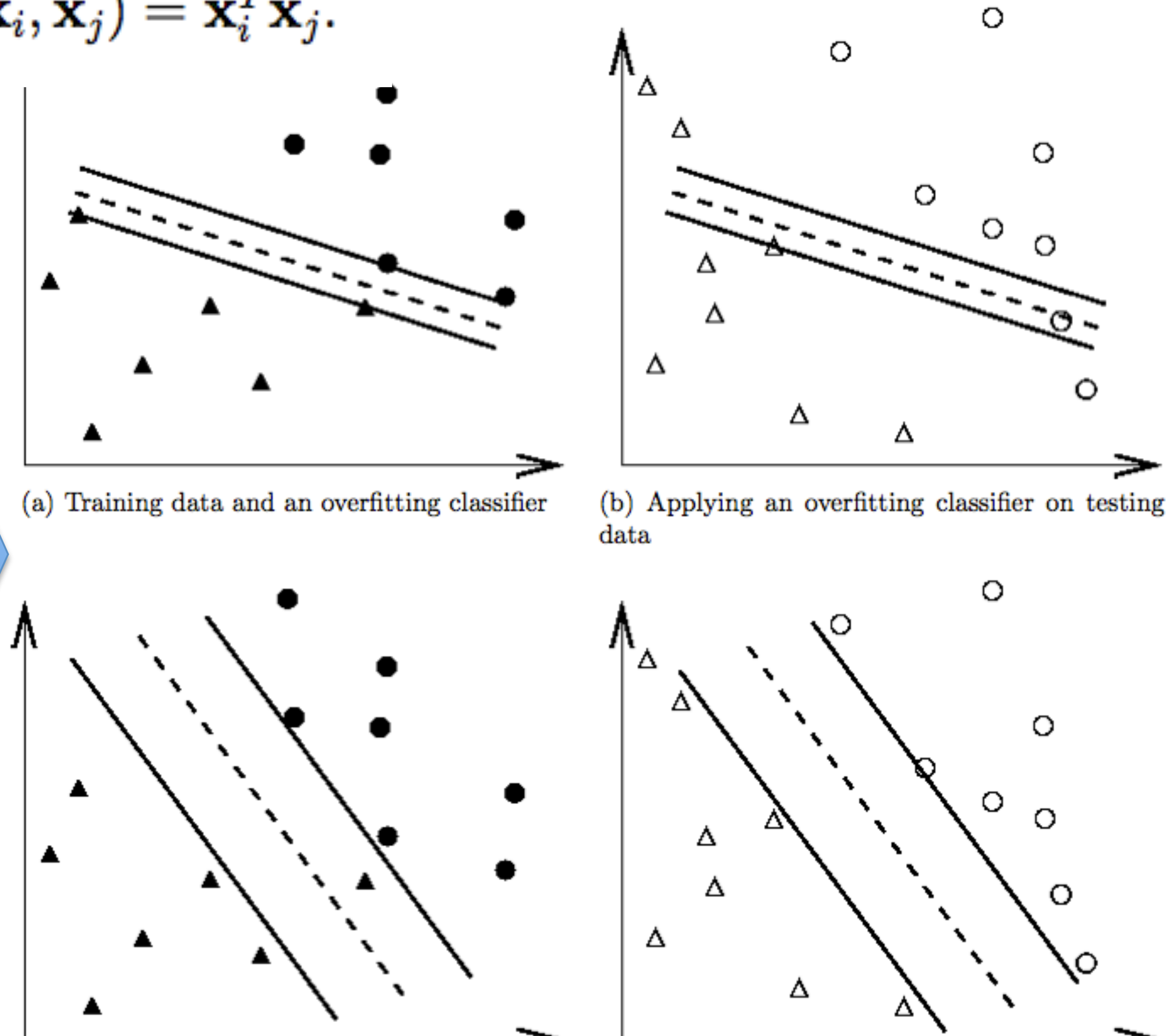
Our goal: find the model M which minimizes the test error:



(c) Model Selection (e.g. for linear kernel)

- linear: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$.

Select the right penalty parameter C



(a) Training data and an overfitting classifier

(b) Applying an overfitting classifier on testing data

(c) Training data and a better classifier

(d) Applying a better classifier on testing data

(c) Model Selection

- radial basis function (RBF): $K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2)$, $\gamma > 0$.

  two parameters for an RBF kernel: C and γ

- polynomial: $K(\mathbf{x}_i, \mathbf{x}_j) = (\gamma {\mathbf{x}_i}^T \mathbf{x}_j + r)^d$, $\gamma > 0$.

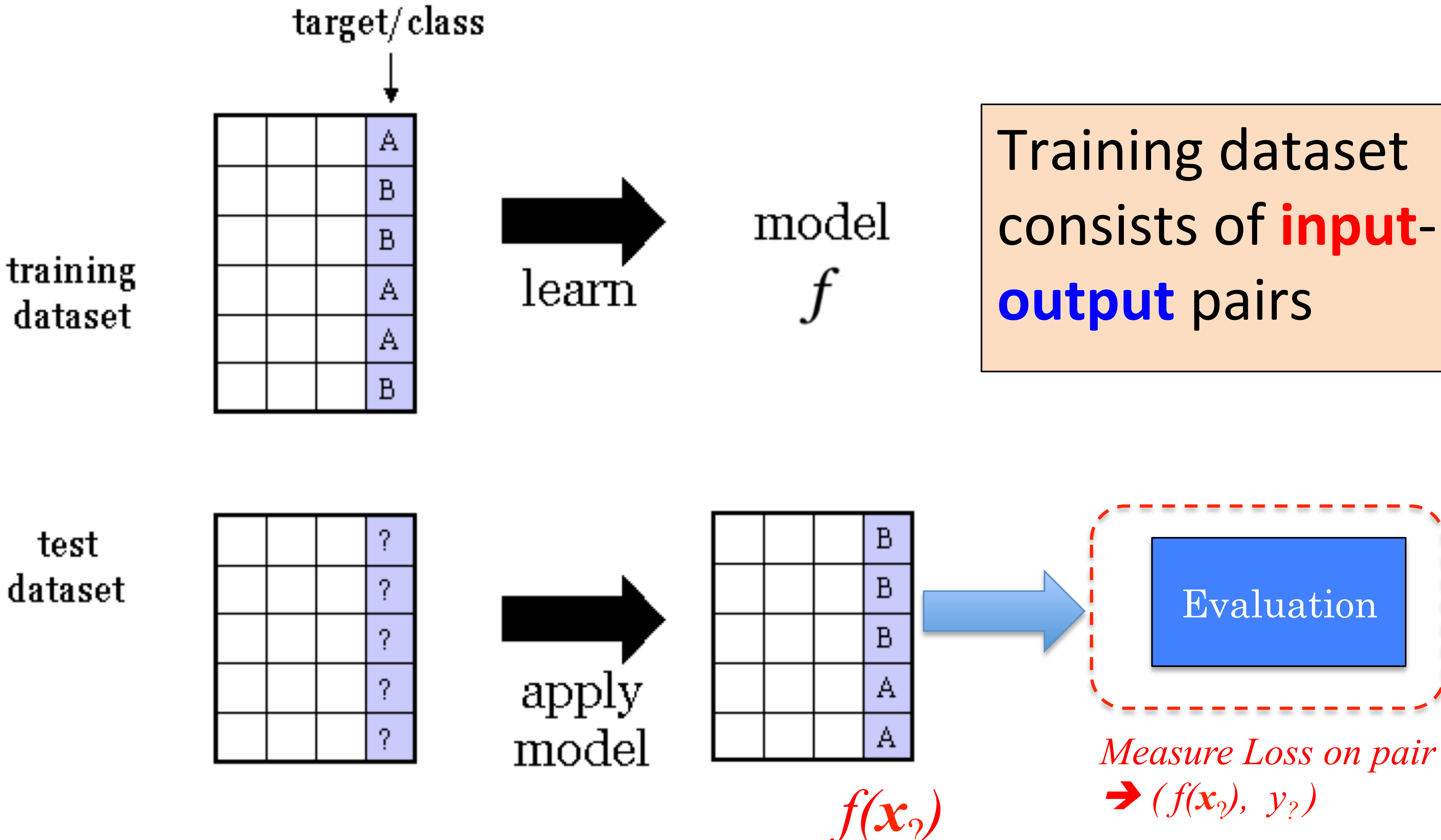
  Three parameters for a polynomial kernel

A Practical Guide to Support Vector Classification

(d) Pipeline Procedures

- (1) train / test
- (2) k-folds cross validation
- (3) k-CV on train to choose hyperparameter / then test

Evaluation Choice-I:
Train and Test



Training dataset consists of **input-output** pairs

Evaluation Choice-II: Cross Validation

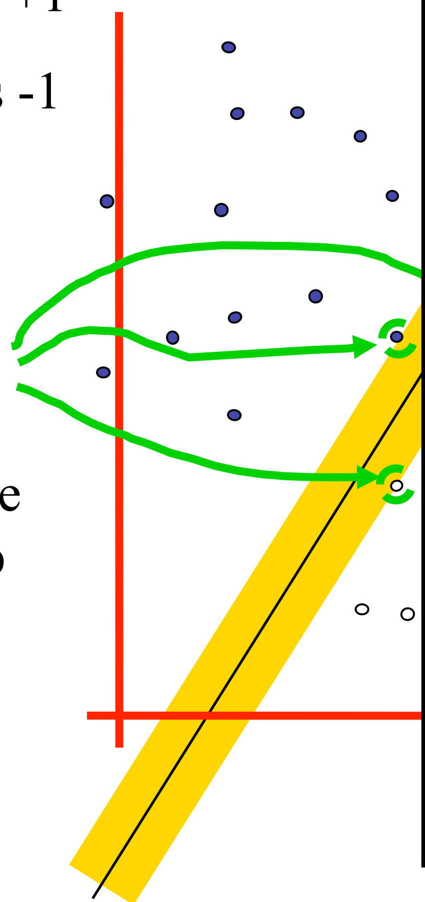
- Problem: don't have enough data to set aside a test set
- Solution: Each data point is used both as train and test
- Common types:
  - -K-fold cross-validation (e.g. K=5, K=10)
  - -2-fold cross-validation
  - -Leave-one-out cross-validation (LOOCV)

A good practice is : to random shuffle all training sample before splitting

Why Maximum Margin for SVM ?

• denotes +1

° denotes -1



Support Vectors are those datapoints that the margin pushes up against

1. Intuitively this feels safest.
2. If we've made a small error in the location of the boundary (it's been jolted in its perpendicular direction) this gives us least chance of causing a misclassification.
3. **LOOCV is easy since the model is immune to removal of any non-support-vector datapoints.**
4. There's some theory (using VC dimension) that is related to (but not the same as) the proposition that this is a good thing.
5. Empirically it works very very well.

Evaluation Choice-III:

Many beginners use the following procedure now:

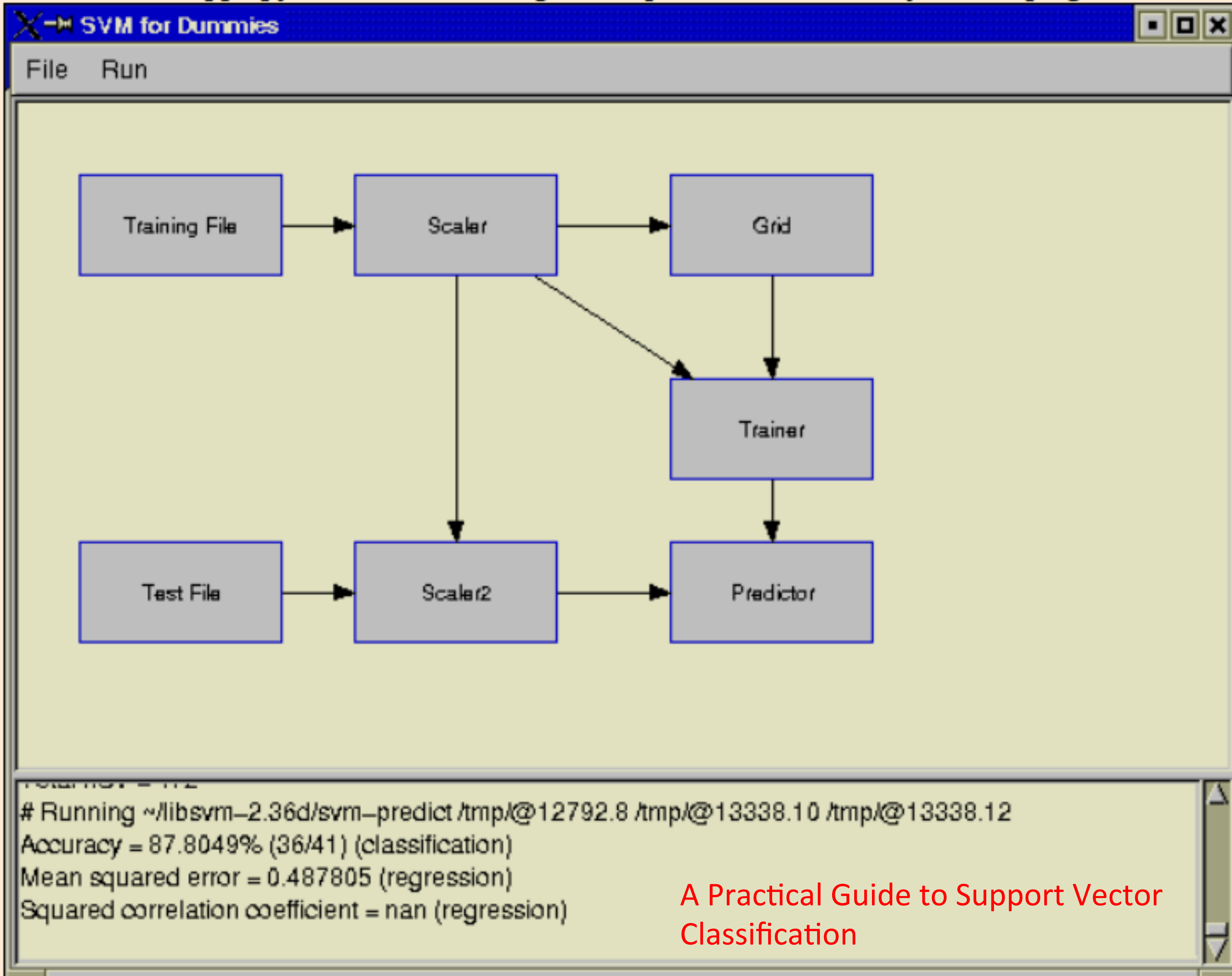
- Transform data to the format of an SVM package
- Randomly try a few kernels and parameters
- Test

We propose that beginners try the following procedure first:

- Transform data to the format of an SVM package
- Conduct simple scaling on the data
- Consider the RBF kernel $K(\mathbf{x}, \mathbf{y}) = e^{-\gamma\|\mathbf{x}-\mathbf{y}\|^2}$
- Use cross-validation to find the best parameter C and γ
- Use the best parameter C and γ to train the whole training set[5]
- Test

For HW2-Q2

A Practical Guide to Support Vector Classification



A Practical Guide to Support Vector Classification

Today: Review & Practical Guide

- ❑ Support Vector Machine (SVM)
  - ✓ History of SVM
  - ✓ Large Margin Linear Classifier
  - ✓ Define Margin (M) in terms of model parameter
  - ✓ Optimization to learn model parameters (w, b)
  - ✓ Non linearly separable case
  - ✓ Optimization with dual form
  - ✓ Nonlinear decision boundary
  - ✓ Practical Guide
    - ✓ File format / LIBSVM
    - ✓ Feature preprocsssing
    - ✓ Model selection
    - ✓ Pipeline procedure

Support Vector Machine

Task
↓
Representation
↓
Score Function
↓
Search/Optimization
↓
Models, Parameters

classification

$$K(\mathbf{x}, z) := \Phi(\mathbf{x})^T \Phi(z)$$

↓
Kernel Tric- Func K(xi, xj)
↓
Margin + Hinge Loss (optional)
↓
QP with Dual form

$$w = \sum_i \alpha_i x_i y_i$$

↓
Dual Weights

$$\underset{\mathbf{w},b}{\operatorname{argmin}} \sum_{i=1}^{p} w_i^2 + C\sum_{i=1}^{n} \varepsilon_i$$

$$\text{subject to } \forall \mathbf{x}_i \in Dtrain : y_i\left(\mathbf{x}_i \cdot \mathbf{w} + b\right) \geq 1 - \varepsilon_i$$

$$\max_\alpha \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\sum_i \alpha_i y_i = 0, \qquad \alpha_i \geq 0 \qquad \forall i$$

References

- Big thanks to Prof. Ziv Bar-Joseph and Prof. Eric Xing @ CMU for allowing me to reuse some of his slides
- Elements of Statistical Learning, by Hastie, Tibshirani and Friedman
- Prof. Andrew Moore @ CMU's slides
- Tutorial slides from Dr. Tie-Yan Liu, MSR Asia
- A Practical Guide to Support Vector Classification Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin, 2003-2010
- Tutorial slides from Stanford "Convex Optimization I — Boyd & Vandenberghe