



# ISDA 2011

11th International Conference on Intelligent  
Systems Design and Applications (ISDA)  
November 22-24, 2011 - Córdoba, Spain

# Mining Ultra-Large Datasets by Kernel Machines

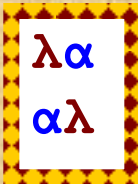
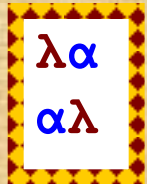
## GPU Implementations and Novel Geometric Algorithms

V. Kecman, Q. Li, R. Strack

**Presenter: Vojislav Kecman**

Learning Algorithms and

Applications Laboratory (**LAAL**)



COMPUTER  
SCIENCE



# Motivations for this talk

- There is no part of human activities left untouched by both the need for and the desire to collect **data** today
- The consequences are - we are **surrounded by**, in fact, we are **immersed in an ocean of all kinds of data** (a.k.a. measurements, images, patterns, sounds, samples, web pages, tunes, x-rays or ct images, etc.)
- **Humans can't handle ultra-large data sets but,**
- **we must develop algorithms able to learn from such datasets and to mine them efficiently**

**1<sup>st</sup>**, let's clarify the talk's title

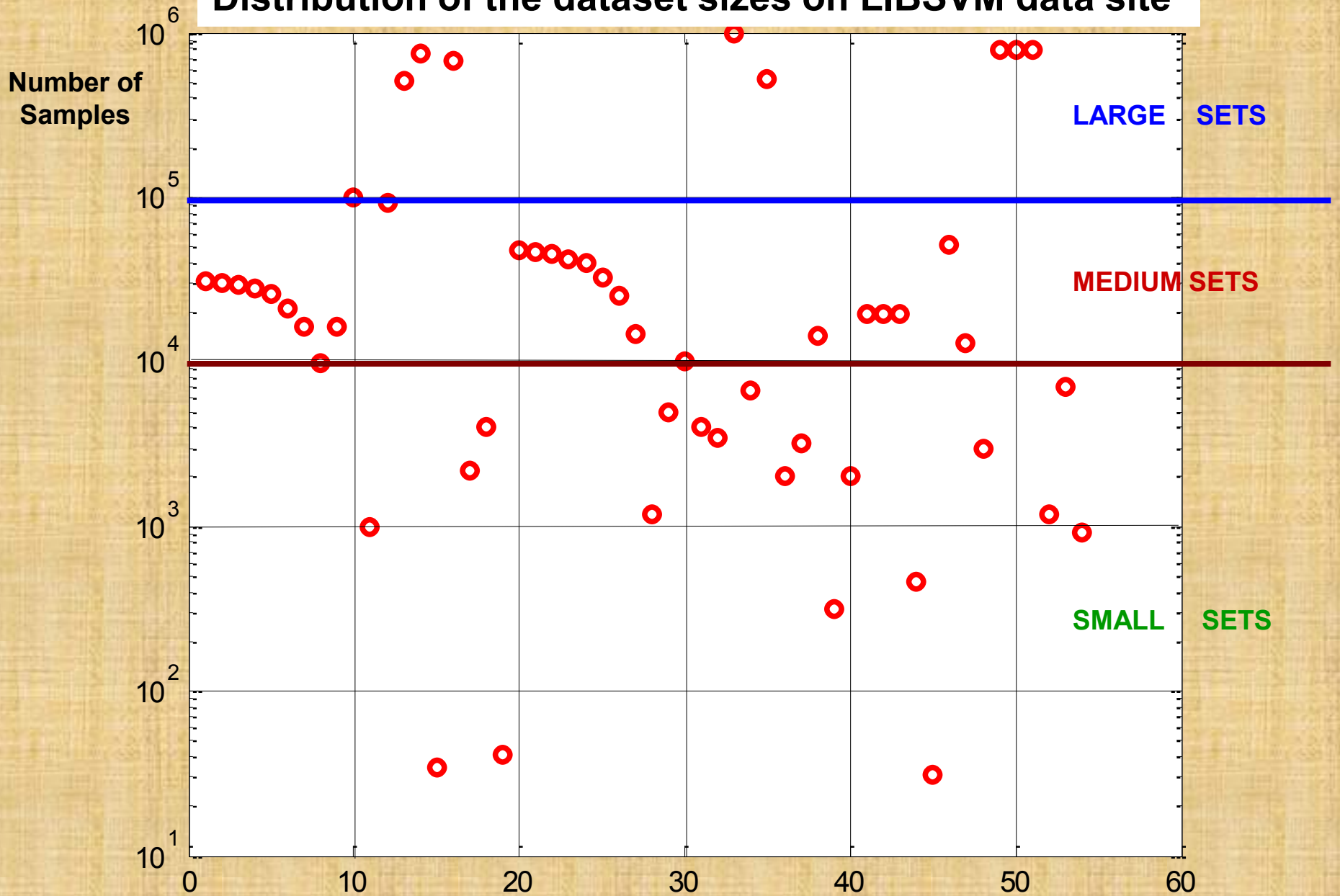
## What is an **ultra-large** dataset?

- A concept of size is continuously changing with both data producing capacities and advances in hardware, but let's define it (**for today only**):
- SMALL < ~ 10,000 samples
- MEDIUM < ~ 100,000 samples
- LARGE up to ~ 1million samples
- ULTRA-LARGE = 'LARGER' than LARGE

**However, remind that the training doesn't depend upon the size of data only. We'll see that some medium datasets are much tougher nuts to crack than some ultralarge samples collections!!!**

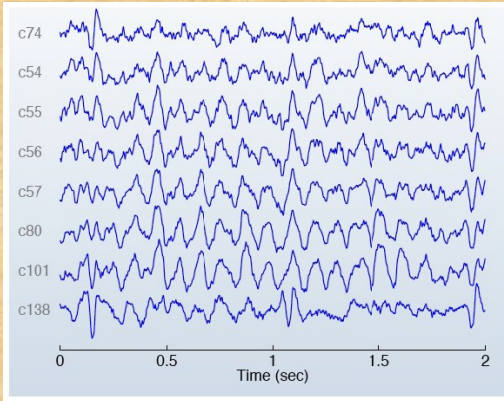
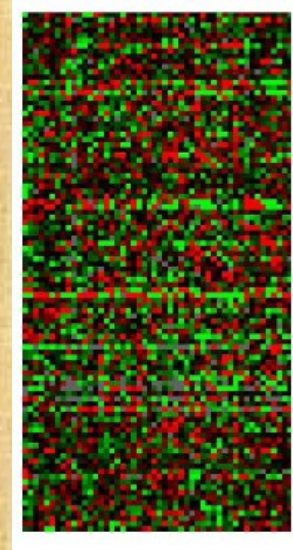


# Distribution of the dataset sizes on LIBSVM data site



# Data sets variety

	Legs	Wings	Fur	Feathers
cat	4	no	yes	no
crow	2	yes	no	yes
frog	4	no	no	no
bat	4	yes	yes	no
barstool	3	no	no	no



## Play golf dataset

Independent variables				Dep. var
OUTLOOK	TEMPERATURE	HUMIDITY	WINDY	PLAY
sunny	85	85	FALSE	Don't Play
sunny	80	90	TRUE	Don't Play
overcast	83	78	FALSE	Play
rain	70	96	FALSE	Play
rain	68	80	FALSE	Play
rain	65	70	TRUE	Don't Play
overcast	64	65	TRUE	Play
sunny	72	95	FALSE	Don't Play
sunny	69	70	FALSE	Play
rain	75	80	FALSE	Play
sunny	75	70	TRUE	Play
overcast	72	90	TRUE	Play
overcast	81	75	FALSE	Play
rain	71	80	TRUE	Don't Play

1	-0.66242	-0.03596
2	101.07	100.04
2	100.28	99.692
2	102.26	99.244
2	100.27	99.729
1	-1.2924	0.31328
1	1.4643	0.63647
2	100.24	99.294

United States Contact Us

Shop Support Community

Best-selling systems Shipped in 24 Hours Laptops & Netbooks All-in-One & Desktops Electronics & Software Printers & Ink HDTVs & Home Theater Mobile Phones & Tablets Dell Outlet View All

Sign In Cart

Search

Windows®. Life without Walls™. Dell recommends Windows 7.

Adjust Performance Personalize Security & Services Accessories Review & Checkout

SWITCH TO LIST VIEW

**Intel® Core™ i5 Processor**  
Smart performance with a speed boost

Intel® Turbo Boost Technology<sup>1</sup>  
Solid performance for everyday applications, plus the ability to increase speed as needed for demanding tasks.

Intel® Hyper-Threading Technology<sup>2</sup>  
4-way multithread processing lets each core work on two tasks at once.

Don't limit yourself  
Step up to the new 2010 3rd Gen Intel® Core™ i7 processor, the top-of-the-line processor that delivers the ultimate in smart performance.

Intel® Turbo Boost Technology performance will vary depending on the specific hardware configuration used.

Continue Personalizing

New! Instant Help Chat Now

**Alienware M11x**

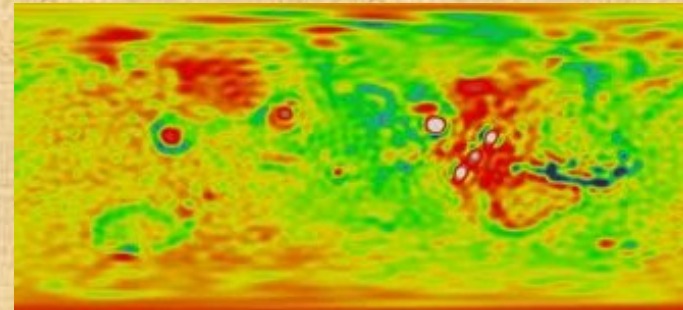
Market Value<sup>1</sup> \$1,299.00  
Total Savings \$300.00  
Dell Price \$999.00  
As low as \$39.00/month<sup>2</sup>  
Apply / Learn More

Discount Details  
Free Priority Ship Date: 1/26/2011  
View Summary

**Software & Services**  
Genuine Windows 7 Home Premium (64-bit)  
Intel® Core™ i5 620LM (3M Cache, 1.066 GHz with 1.866 GHz Max Turbo Frequency) - Overclockable  
Intel® Core™ i5 640LM (4M Cache, 1.3 GHz with 2.266 GHz Max Turbo Frequency) - Overclockable (Add \$165.00 or \$5.00/month<sup>3</sup>)  
Alienware Recommended

Intel® Core™ i5 Processor  
Intel® Core™ i5 620LM (3M Cache, 1.066 GHz with 1.866 GHz Max Turbo Frequency) - Overclockable (Included in Price)  
Intel® Core™ i7 Processor  
Intel® Core™ i7 640LM (4M Cache, 1.3 GHz with 2.266 GHz Max Turbo Frequency) - Overclockable (Add \$165.00 or \$5.00/month<sup>3</sup>)  
Alienware Recommended

Software & Services  
Genuine Windows 7 Home Premium (64-bit)  
Intel® Core™ i5 620LM (3M Cache, 1.066 GHz with 1.866 GHz Max Turbo Frequency) - Overclockable  
3GB Dual Channel DDR3 at 800MHz  
32GB SATA II 2.0/09PM  
No Internal WLAN Antenna Installed



Mars magnetic field

# Hence, our data is given as:

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & \dots \\ x_{21} & x_{21} & \dots & \dots \\ \vdots & \vdots & \dots & \vdots \\ x_{l1} & x_{l2} & \dots & \dots \end{bmatrix} \quad \mathbf{Y}_c = \begin{bmatrix} +1 \\ -1 \\ \vdots \\ -1 \end{bmatrix} \quad \text{or} \quad \mathbf{Y}_r = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_l \end{bmatrix}$$

1<sup>st</sup> data pair

l<sup>st</sup> data pair

# Single Algorithms only, not Ensemblings today

- We are discussing **single algorithms** (i.e., approaches, methods) and
- **not ensemble methods**, such as bagging, boosting, committee of trees, random forest and/or nonlinear ensemble approaches,
- which all have been proposed to improve performance of single models (NNs, SVMs, linear models, trees, etc ...) to get a **STRONG** classifier
- by **combining** multiple of **weak (base)** classifiers



Living in an **ocean of data** produced on daily basis what can, must, should humans do, right now?

a) **stop** collecting them

b) **keep** collecting the data and save them for future use

c) **collect** them and **analyze** whatever you can right now & avoid in this way drowning in data, while starving for knowledge

## **SOME TOPICS today**

- Basic Model of Computational Intelligence (i.e., Machine Learning, i.e., Data Mining) is:

### **The Sum of Weighted Basis Functions**

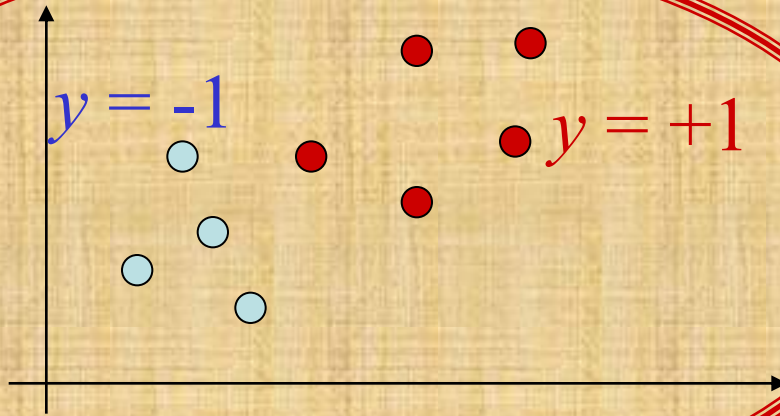
- One model == Many (almost all the) models

-- **LARGE DATA SETS & Some Contemporary Tools:**

**New Hardware (GP GPUs) & New (Geometric) Approaches**

# Let's first set the stage

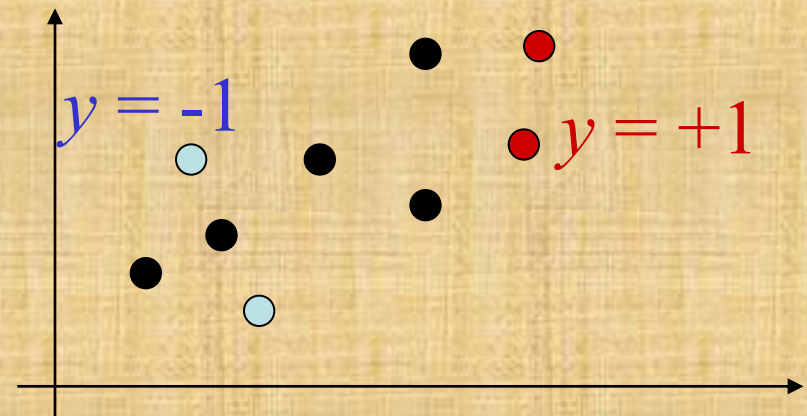
there are three (3) machine learning (ML) settings



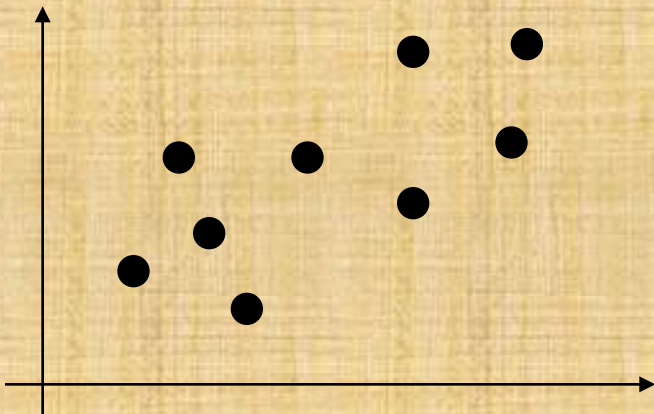
This talk is all about the supervised learning

**Supervised**

**Semi-supervised**



**Unsupervised**



**Supervised Learning** is concerned by solving two (out of three) classic statistics problems:

**Classification (Pattern Recognition)**

**Regression (Curve, Surface, Fitting, i.e., Function Approximation)**

one more statistics' problem, we will not be discussing here, is the **Density Estimation Problem**

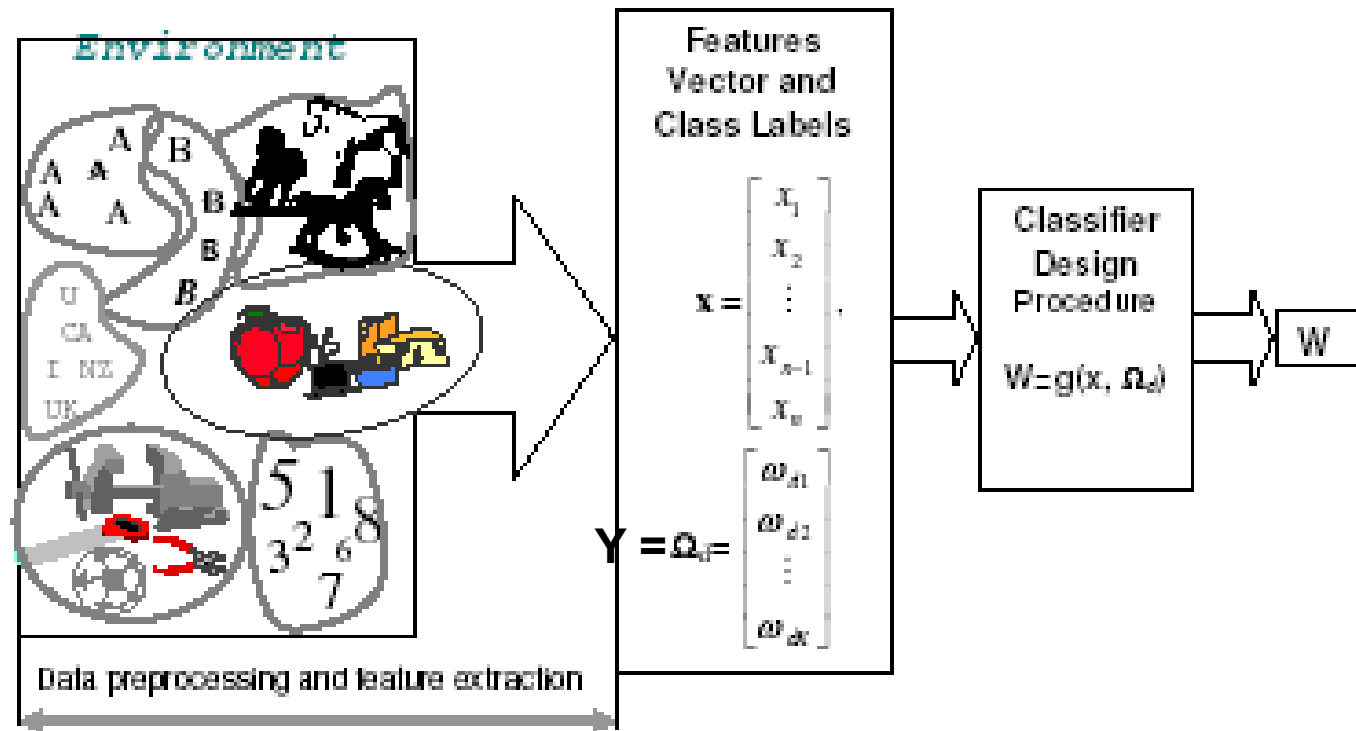
# Today, we'll discuss classification i.e., pattern recognition, only

## Training Phase:

$$\mathbf{w} = g(\mathbf{x}, \mathbf{y})$$

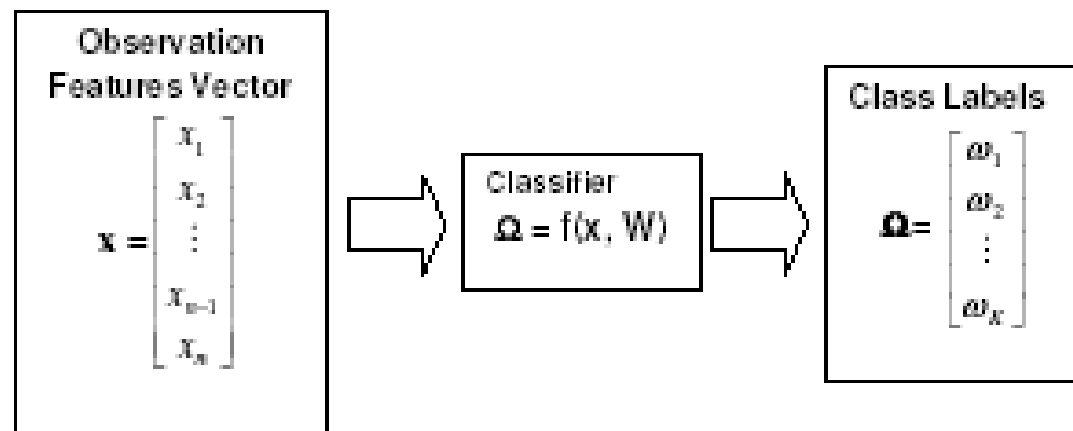
class label  $\omega_i = y_i$

This stage is a CPU time expensive one

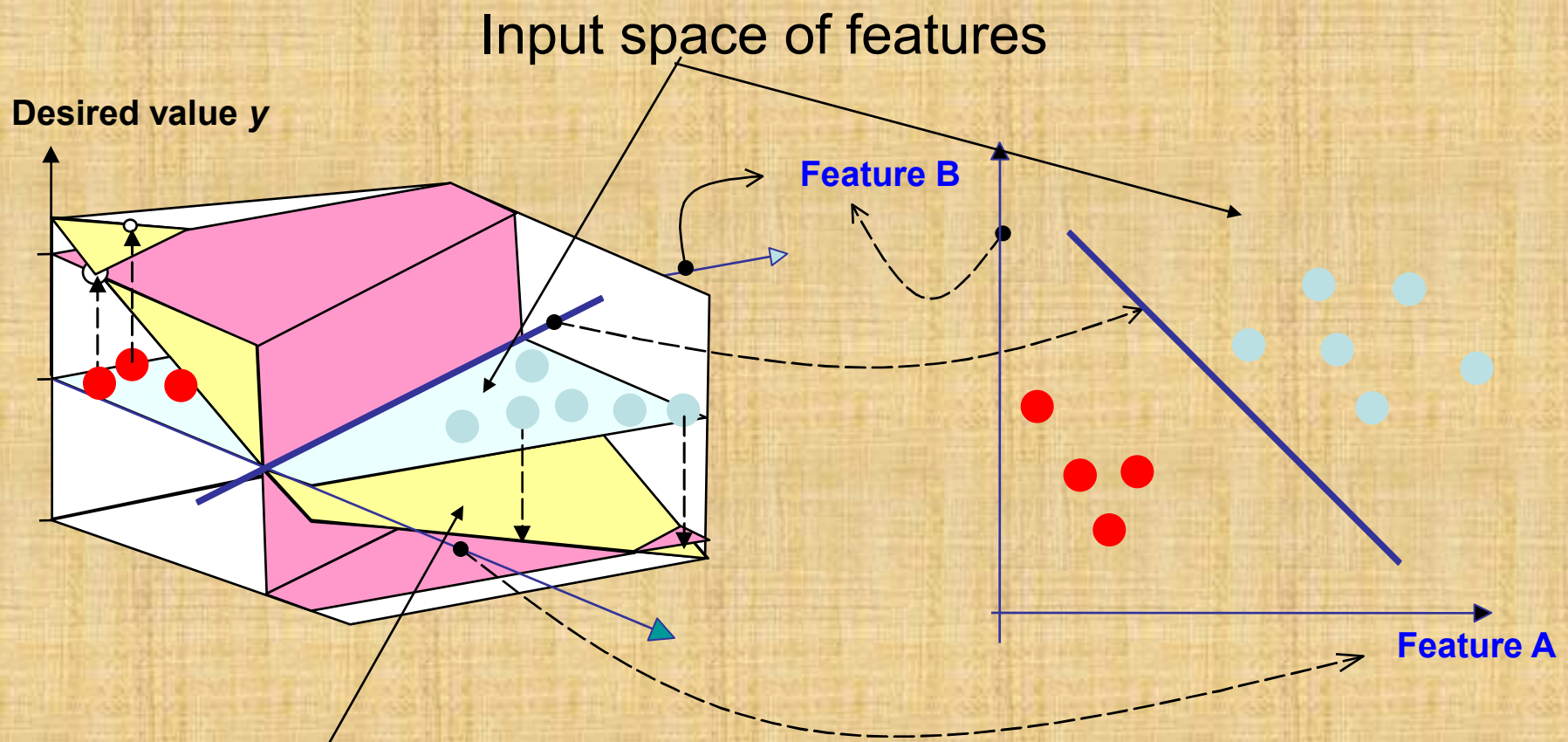


## Test, i.e. Application, Phase:

$$y_i = \omega_i = f(\mathbf{x}_i, \mathbf{w})$$



Classifying in 2 features space, we see the **decision** function.  
When # of features > 2, we deal with HYPER-surfaces, that can't be seen.  
However, the algorithms 'see' in high-dim spaces and **they will be the same.**



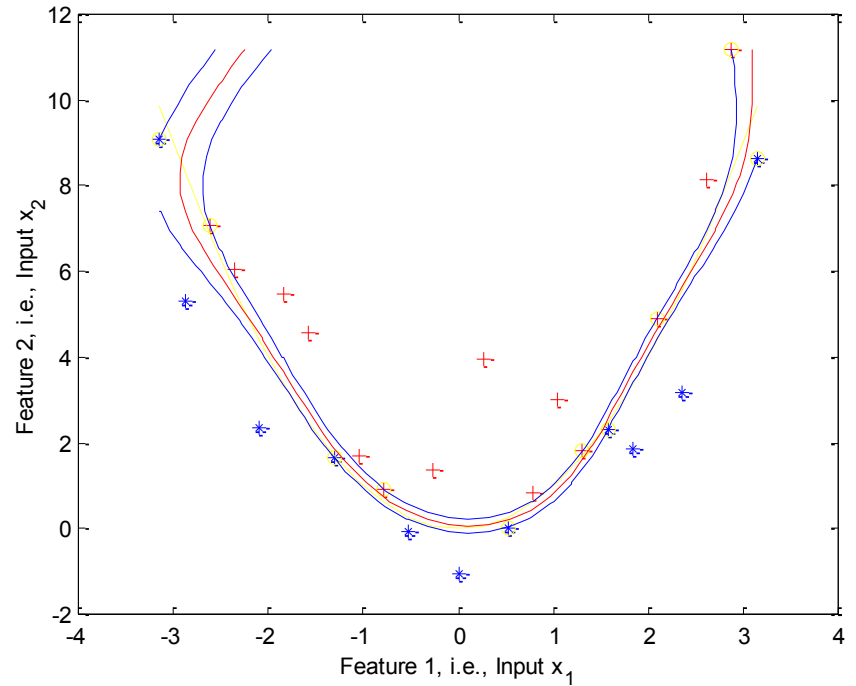
Linear **decision** function

A linear model is same for any-dim case

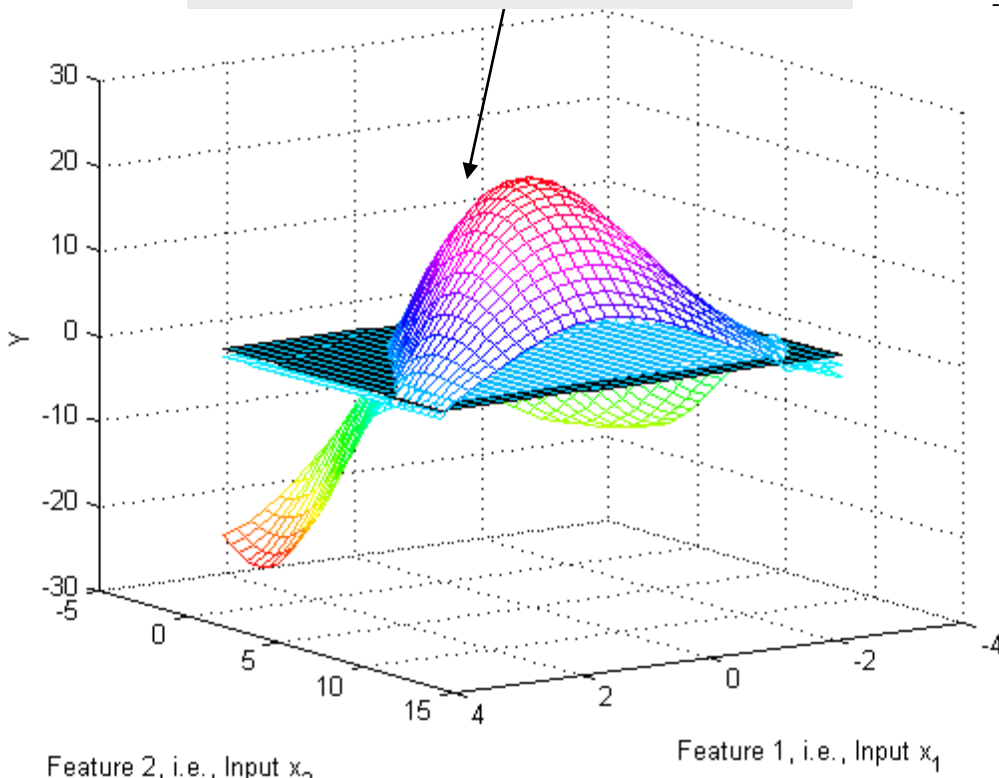
$$y = Xw$$

VERY OFTEN the decision function and separating boundary are **NONLINEAR**

Separation curve (SC) obtained by Gaussian RBF - red<sub>solid</sub>, Margins - blue, Unknown SC yell<sub>dash</sub>



NL SVM decision function



**Such complex decision functions**  
can be realized by many models,  
notably polynomial approximations,  
NNs, SVMs, decision trees, etc ...

**What are then the DIFFERENCES**  
**and/or possibly SIMILARITIES**  
**between these VARIOUSLY**  
**NAMED ML TOOLS?**

Some connections of  
classic techniques such as Fourier  
series & Polynomial approximations  
with  
NNs *or/and* SVMs



Classic approximation techniques in NN graphical appearance

## FOURIER SERIES

AMPLITUDES and PHASES of sine (cosine) waves are **unknown**,  
but frequencies are known because

Mr. Joseph Fourier has selected frequencies for us -> they are

INTEGER multiples of some **pre-selected** base frequency.

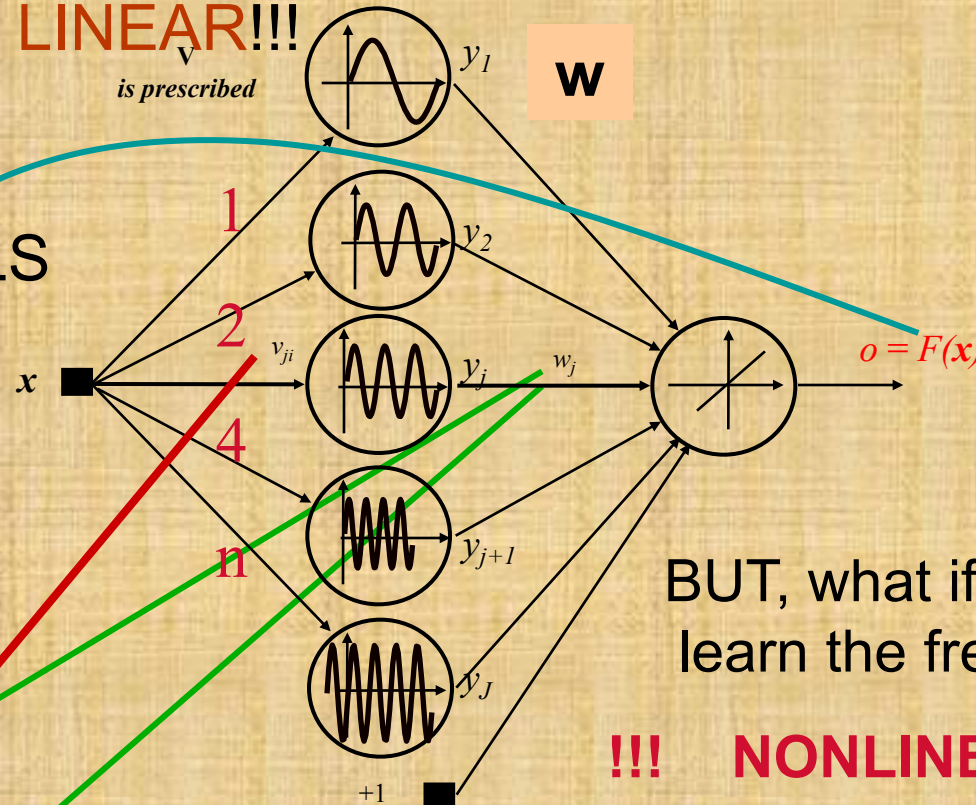
And the problem is **LINEAR!!!**

*is prescribed*

**W**

It is 'same'

with POLYNOMIALS



BUT, what if we want to learn the frequencies?

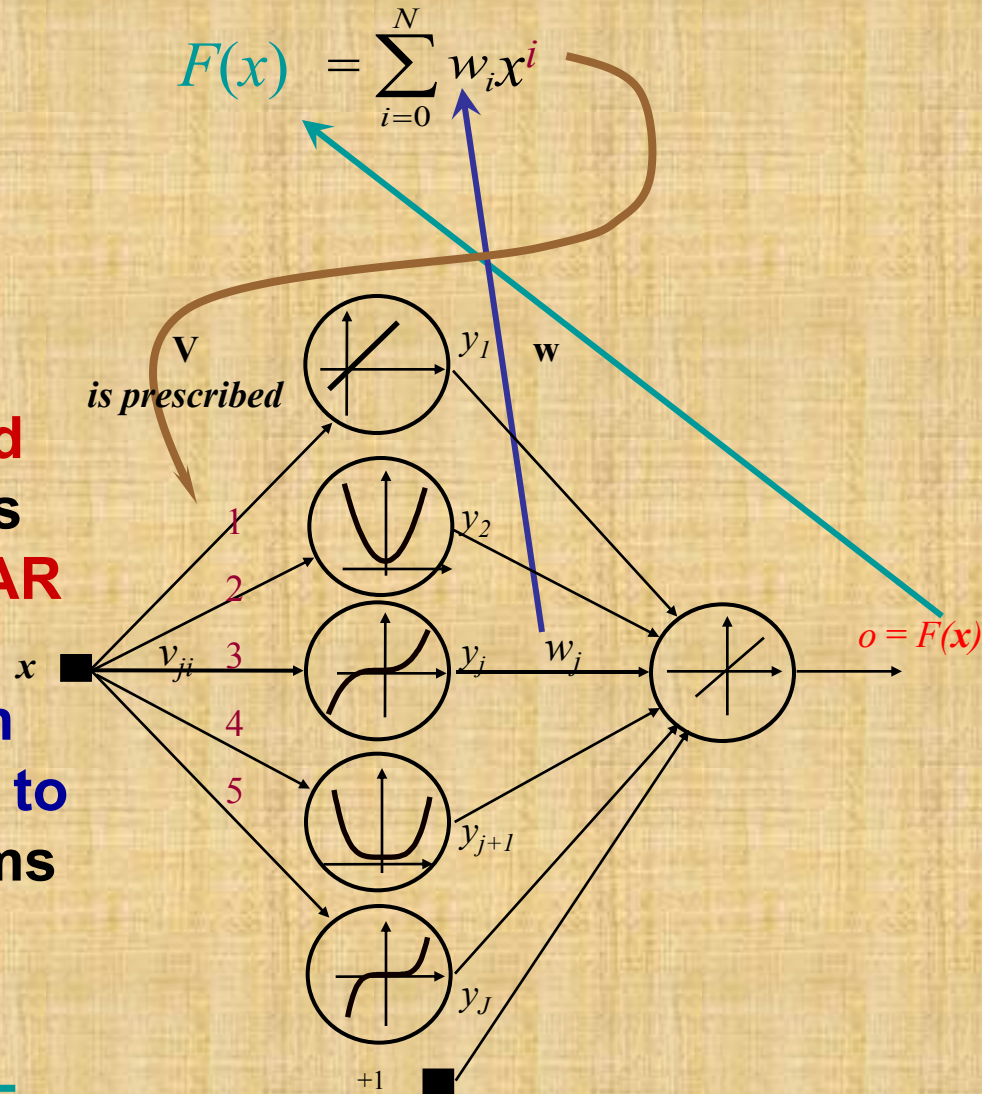
**!!! NONLINEAR LEARNING PROBLEM !!!**

$$F(x) = \sum_{k=1}^N a_k \sin(kx), \text{ or } b_k \cos(kx), \text{ or both}$$

# Another classic approximation scheme is a **POLYNOMIAL SERIES**

$$F(x) = \sum_{i=0}^N w_i x^i$$

With the **prescribed** (integer) exponents this is again a **LINEAR APPROXIMATION SCHEME**. **Linear** in terms of parameters to learn and not in terms of the resulting approximation function.  $F(x)$  is **NL function** for  $i > 1$ .



The two 'novel' learning machines in regression  
i.e., in classification (pattern recognition) are

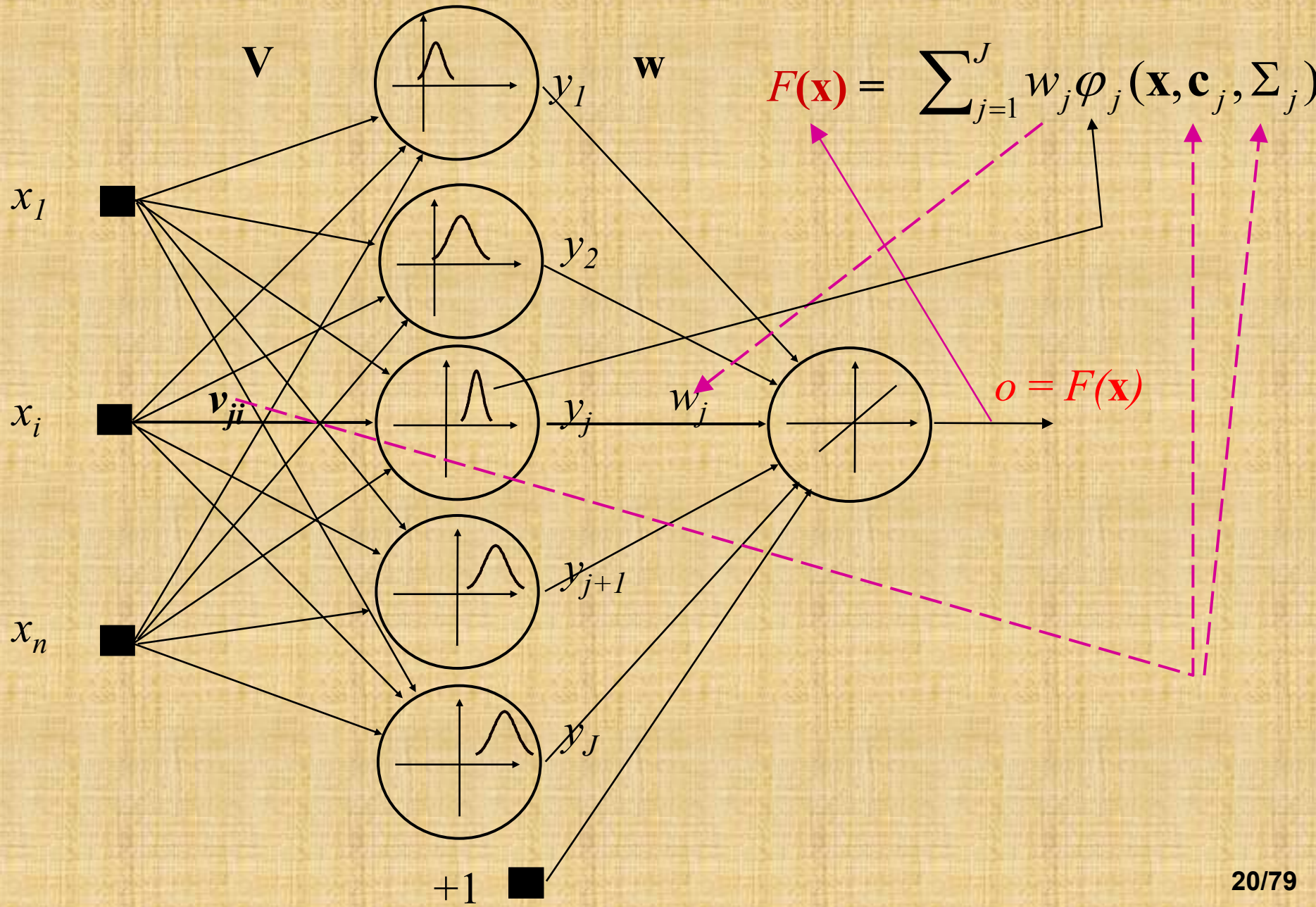
**SVMs or NNs**

(however remember, there are other models too).

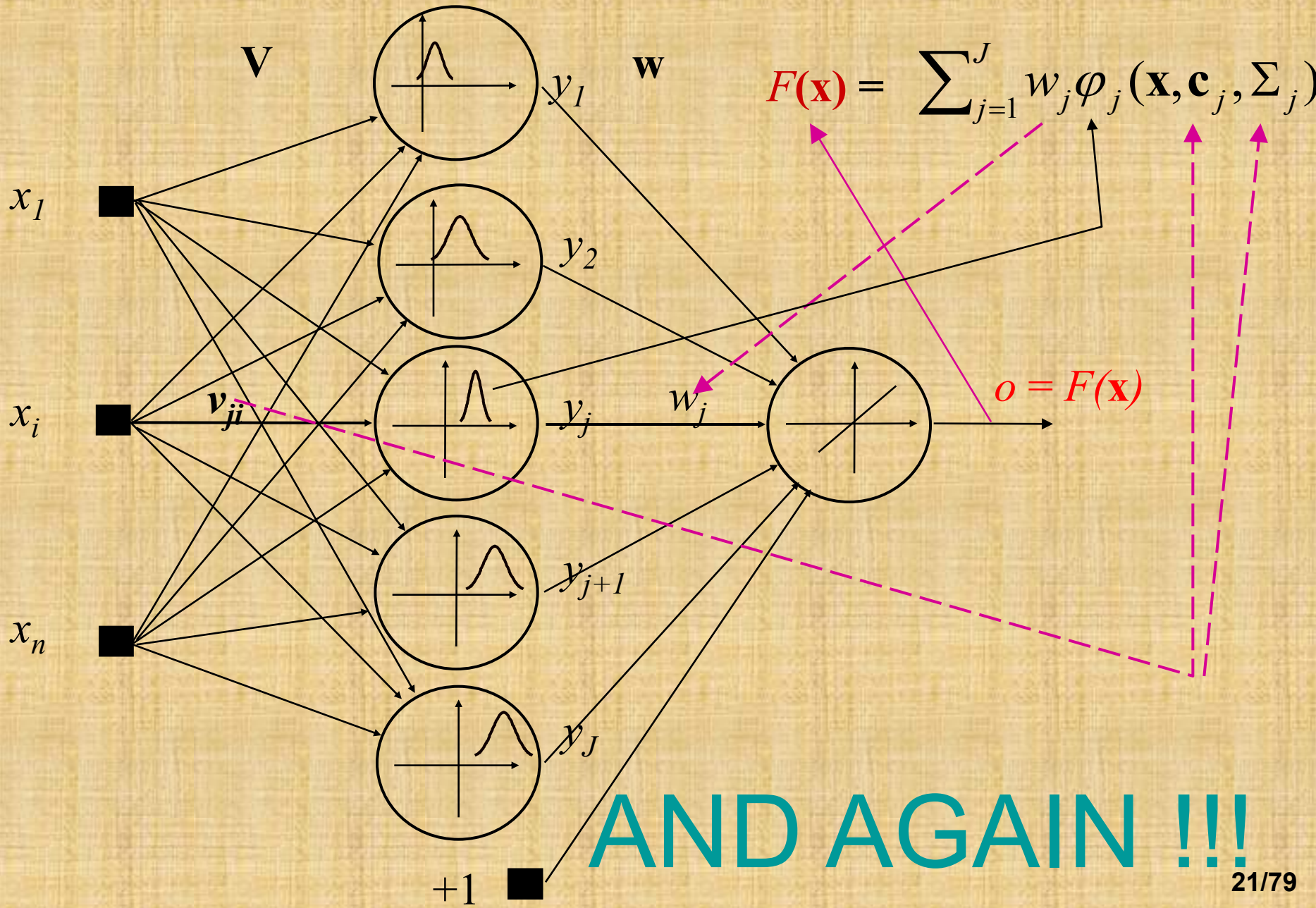
WHAT are DIFFERENCES and SIMILARITIES?

**WATCH CAREFULLY NOW !!!**

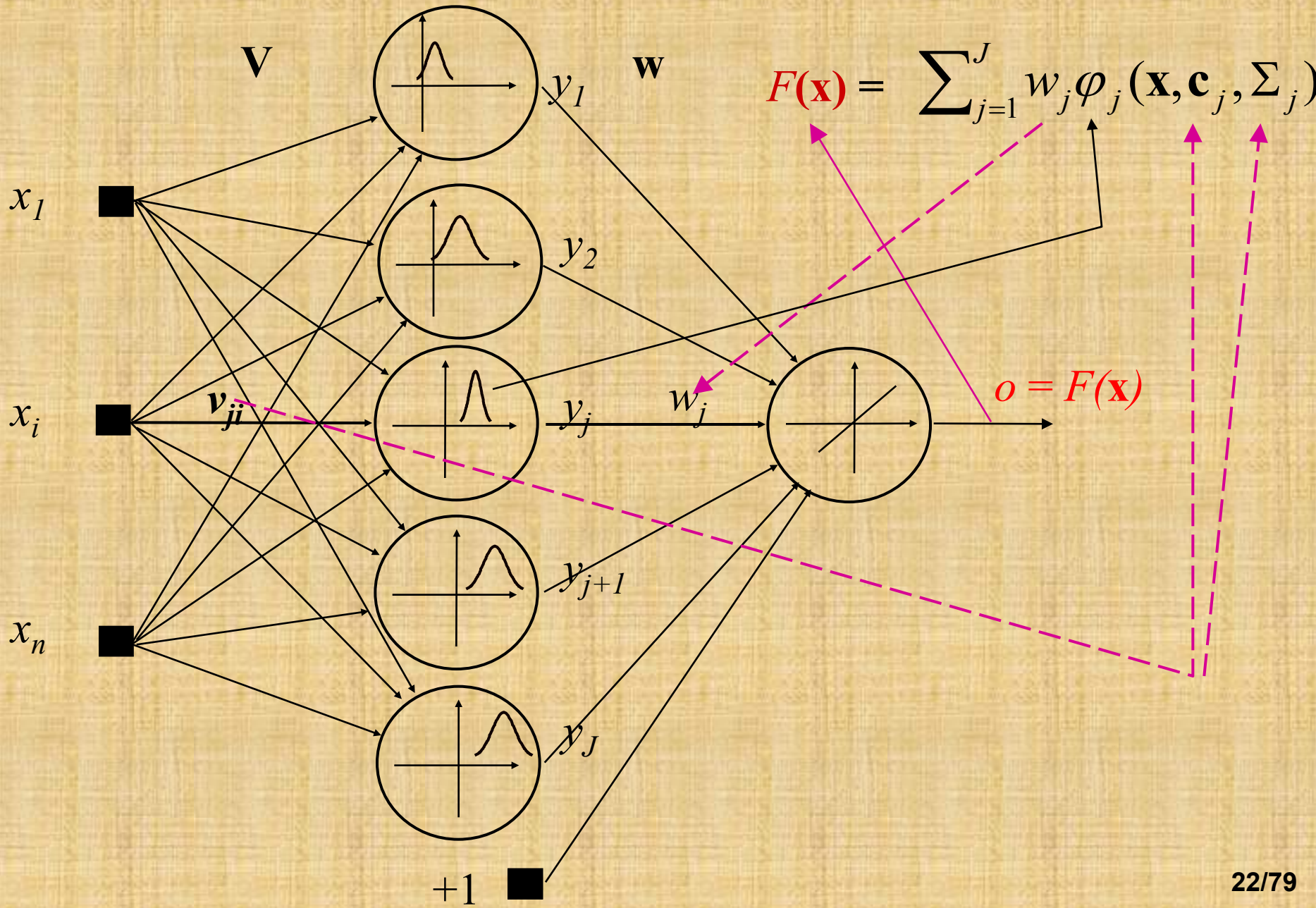
# This is a Neural Network,



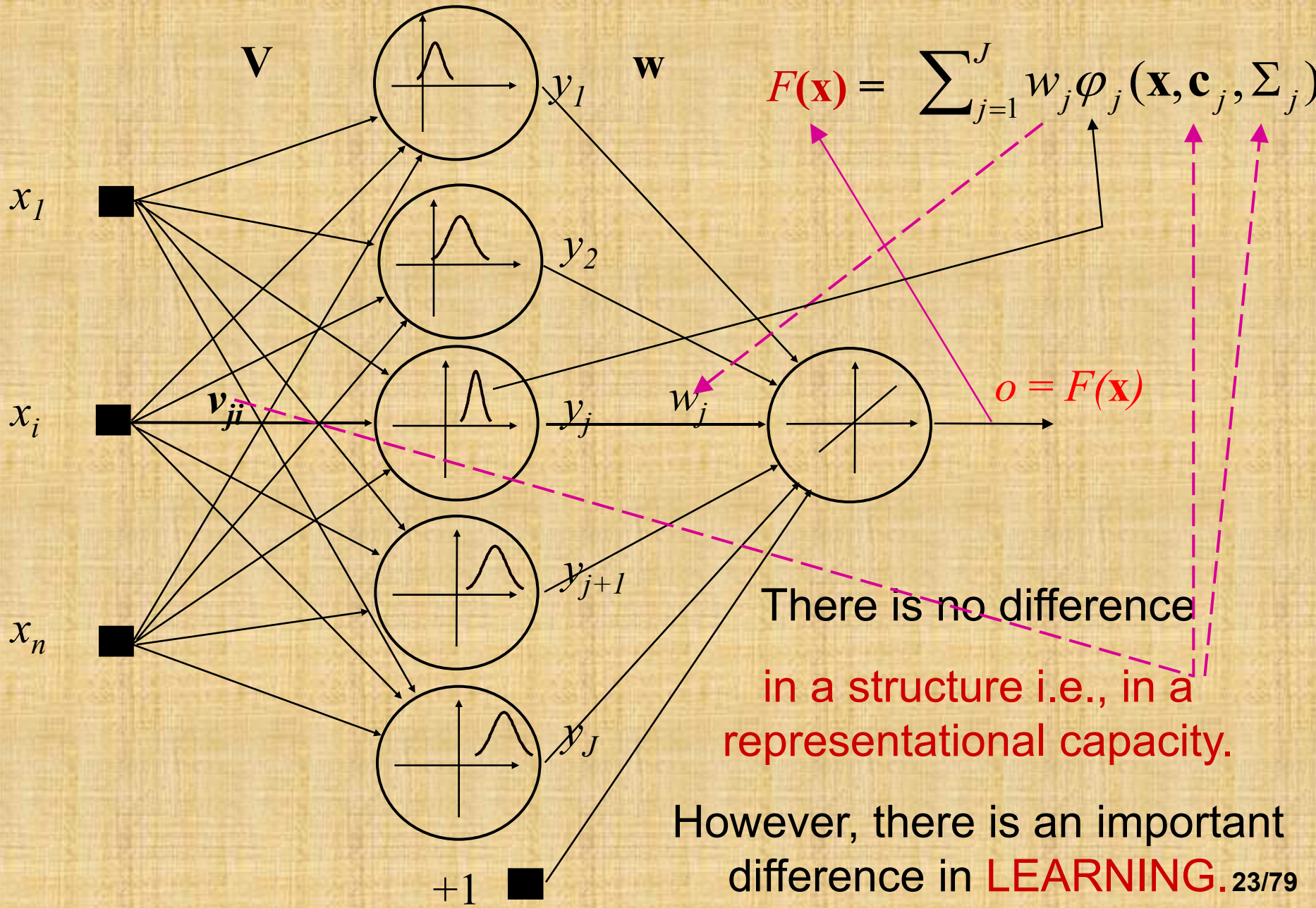
and, this is a Support Vector Machine.



# This is a Neural Network,



and, this is a Support Vector Machine.



Where then the  
**BASIC DIFFERENCES** between  
NNs and SVMs

*(in fact among all the other various ML models)*

are coming from?



# Well ! There are two fundamental pieces in any ML modeling

- They are the questions of:

- the **FORM**

and

- the **NORM**

# FORM

- covers – the type of the model and in particular the type of the kernel (SVM), i.e., activation (NN), i.e., basis (RBF), i.e., membership (FL) function used

# NORM

- covers – the type of the cost, i.e., merit, i.e., loss, i.e., fitness, i.e., objective, function **which is minimized over the parameters of interest** (here, we call them **weights**)

# FORM

- **'All'** our models in ML are 'same' i.e. they are the

**SUM OF THE WEIGHTED BASIS FUNCTIONS**

$$o = f(\mathbf{x}) = \sum_{j=1}^J w_j \varphi_j(\mathbf{x}, \mathbf{c}_j, \Sigma_j)$$

Hence,

**Hyperparameters** to be found during the learning (training) phase

**ONE MODEL = MANY MODELS**

Polynomial approximations, Fourier expansions, NN, SVMs, wavelets, JPEG, MPEG, Fuzzy Logic models, ..., many others ... they ALL are

# NORM

- Basically, we use primarily (only) two NORMs (cost functions) in ML which are the
- **MINIMIZATION** of the **SUM OF ERROR SQUARES** in the **OUTPUT space** (linear standard classifier, FFT, JPEG, MPEG, MLP NN and RBF NN) –  $L_2$  norm

and the

- **MAXIMIZATION** of the **MARGIN** in the **INPUT space** expressed as a **MINIMIZATION** of the **SUM OF WEIGHTS SQUARES** (SVMs)

(a variant of both may be the  $L_1$  norm or some composite norm)

# Norms (Loss Functions) of NNs and SVMs

$$E = \sum_{i=1}^P (d_i - f(\mathbf{x}_i, \mathbf{w}))^2$$

**A classic multilayer perceptron (MLP), FFT, polynomial models**

*Closeness to data*

$$E = \sum_{i=1}^P (d_i - f(\mathbf{x}_i, \mathbf{w}))^2 + \lambda \|\mathbf{P}f\|^2$$

**Regularization (RBF) NN**

*Closeness to data*

*Smoothness*

$$E = \sum_{i=1}^P L_{\varepsilon i} + \lambda \|\mathbf{P}f\|^2 = \sum_{i=1}^P L_{\varepsilon i} + \underbrace{O(h, l)}_{\text{Capacity of machine}} \quad \text{Support Vector Machines}$$

*Closeness to data*

*Capacity of machine*

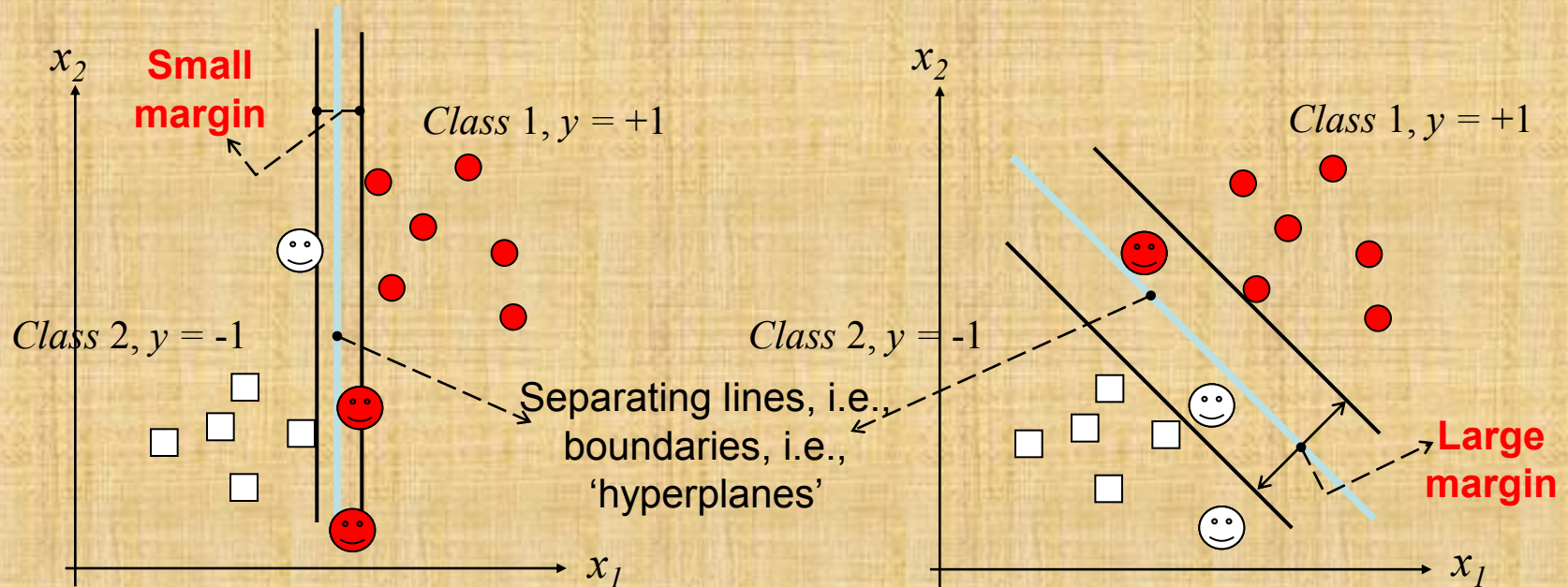
In the last expression the SRM principle uses the VC dimension  $h$  (defining model capacity) as a controlling parameter for minimizing  $E$

# SUPPORT VECTOR MACHINE

is a **MAXIMAL MARGIN CLASSIFIER** which

- creates separating hyperplane with the maximal geometric margin
- **WHY maximal margin?**

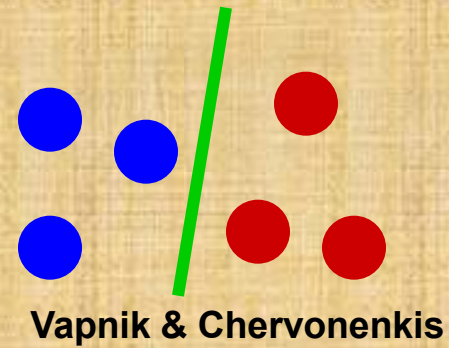
Consider two linearly **separable** classes below. Two perfect separation boundaries of two different decision functions are shown



**Thus, the larger the margin, the smaller the probability of misclassification!**

# A gentle SVMs history graph from the 'simple' linear case to the more complex ones!

- **Linear Maximal Margin Classifier for Linearly Separable Data** - no samples overlapping (late **1960-ties** and early **70-ties**).



- **Nonlinear Classifier (1992)**

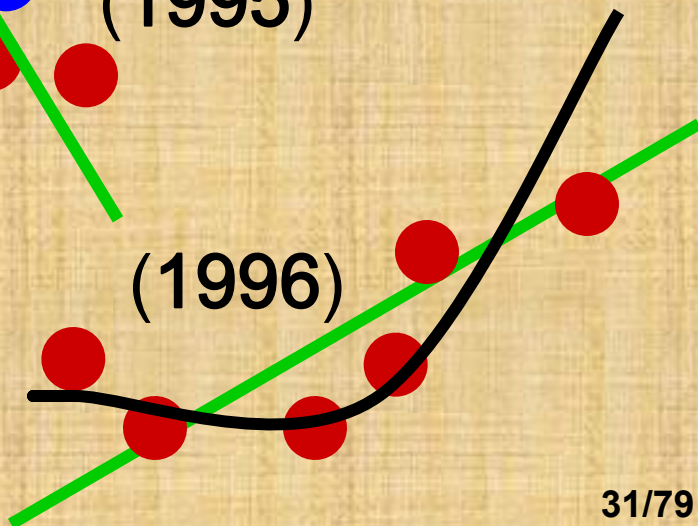


- **Linear Soft Margin Classifier for Overlapping Classes.**



- **Regression by SV Machines that can be both linear and nonlinear!**

Drucker & Burges & Kaufman, & Smola & Vapnik



The margin will be maximized by solving QP problem

minimize

Margin  
maximization!

$$J = \mathbf{w}^T \mathbf{w} = \|\mathbf{w}\|^2 = w_1^2 + w_2^2 + \dots + w_n^2$$

subject to constraints

Correct  
classification!

$$y_i[\mathbf{w}^T \mathbf{x}_i + b] \geq 1, \quad i = 1, n$$

Note that # of constraining inequalities = # of training data /

This classic QP problem with constraints ends in forming and solving a primal and dual Lagrangian

Dual Lagrangians for both (regression and classification) are given on the next slide



# SVMs Linear **Classification** Learning (Training) Setting

Dual Problem:

$$L_d = -\frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^N \alpha_i = \max_{\boldsymbol{\alpha}}$$

s.t.  $0 \leq \alpha_i \leq C$  for  $i=1, \dots, N$

$$\sum_{i=1}^N \alpha_i y_i = 0$$

# SVMs Linear **Regression** Learning (Training) Setting

Dual Problem:

$$L_d = -\frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) \mathbf{x}_i^T \mathbf{x}_j - \varepsilon \sum_{i=1}^N (\alpha_i + \alpha_i^*) + \sum_{i=1}^N (\alpha_i - \alpha_i^*) y_i = \max_{\boldsymbol{\alpha}}$$

s.t.  $0 \leq \alpha_i, \alpha_i^* \leq C$  for  $i=1, \dots, N$

$$\sum_{i=1}^N (\alpha_i - \alpha_i^*) = 0$$

(N, N) matrix

or, in a matrix notation

$$L_d(\boldsymbol{\alpha}) = -\frac{1}{2} \boldsymbol{\alpha}^T \mathbf{K} \boldsymbol{\alpha} + \mathbf{f}^T \boldsymbol{\alpha}$$

which has the

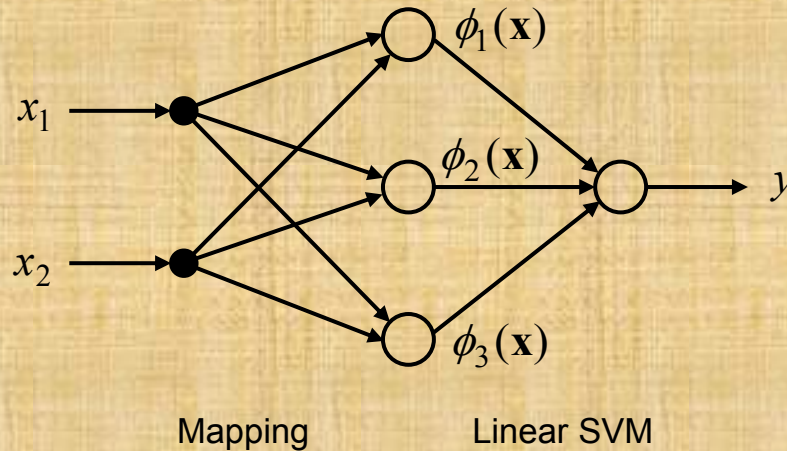
• classification

• regression

final solutions as:

$$f(\mathbf{x}) = \sum_{i=1}^N \begin{Bmatrix} \alpha_i y_i \\ \alpha_i - \alpha_i^* \end{Bmatrix} * \mathbf{x}_i^T \mathbf{x} + b$$

# Nonlinear SVMs



Kernel-Function:  $k(\mathbf{x}, \mathbf{y}) = \Phi^T(\mathbf{x}) \cdot \Phi(\mathbf{y})$

- New at NL SVM:
- Scalar product is replaced by the Kernel-Function.
  - Kernel-Function is usually **positive definite**.
  - Support Vectors Representation of an NL SVM is:

$$f(\mathbf{x}) = \sum_{i=1}^N \left\{ \begin{array}{l} \alpha_i y_i \\ \alpha_i - \alpha_i^* \end{array} \right\} k(\mathbf{x}_i, \mathbf{x}) + b$$

• classification  
• regression

# Some SVMs' constructive problems

- i) Kernel (Hessian) matrix  $\mathbf{K}$  is both DENSE & VERY badly conditioned, but
- ii) in a batch mode, **SVM training may work fine for not too large datasets.**

However, with **the number of data points increasing (say  $N > 5,000$ )** the difficulties with a standard (batch) method show up.

A training set of **50,000 examples amounts to a kernel (Hessian) matrix  $\mathbf{K}$  with  $2.5 \cdot 10^9$  (2.5 billion) elements.** Using an 8-byte floating-point representation we need **20,000 Megabytes = 20 Gigabytes of memory** while **1 million examples asks for 8 Terrabytes of memory for storing  $\mathbf{K}$ .** This cannot be fit into memory of present standard computers.

The way to go is a **DECOMPOSITION**

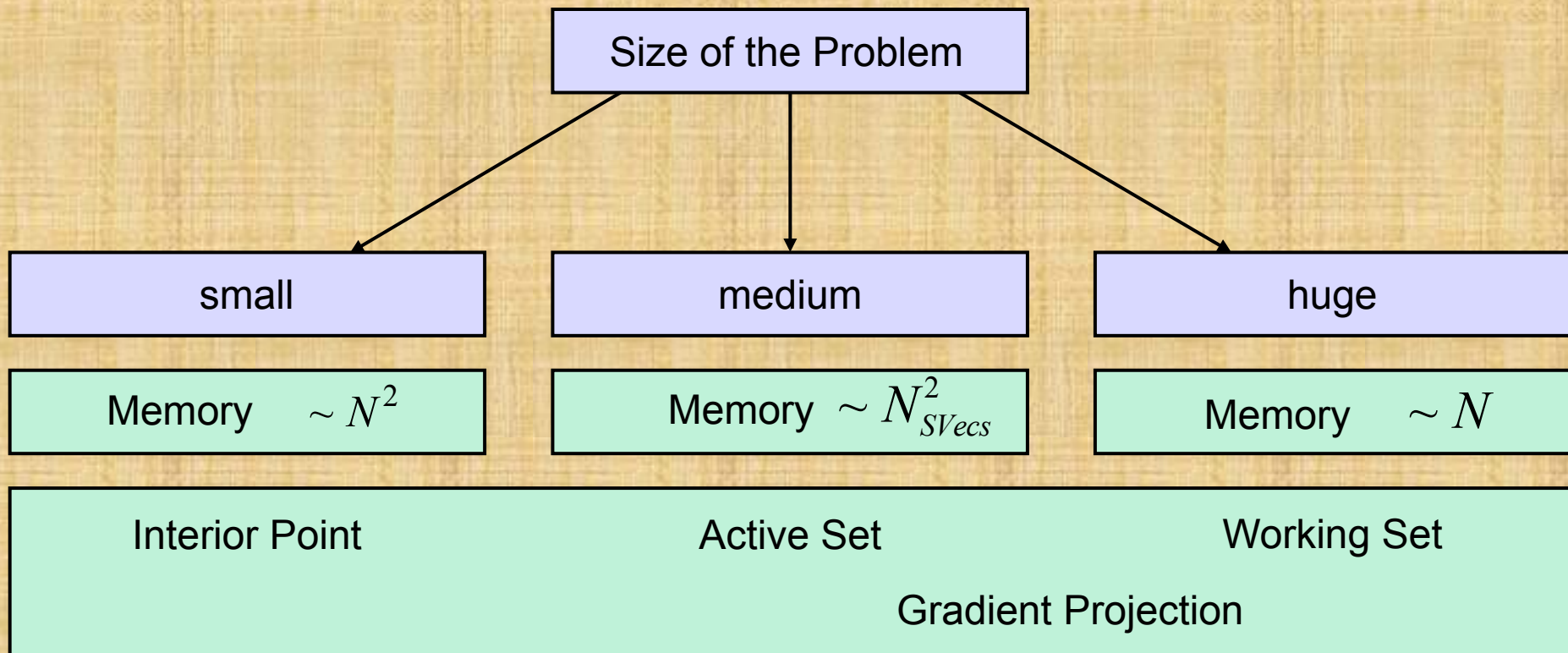
- Vapnik (1995) proposed the **chunking method**
- Osuna, Girosi (1997) present **another efficient decomposition** method.
- Platt (1997) proposed the sequential minimal optimization (**SMO**) (it works with 2 data points at the time) which became the **working horse of SVM learning.**

The newest Iterative Single Data Algorithm (**ISDA**) - Kecman, Vogt, Huang, **2003**; Huang, Kecman, **2004** - seems to be the fastest for a huge data sets at the moment – check: [Yottamine.com](http://Yottamine.com)

# Problem Size & QP Solving Algorithms\*

Original training of SVM is not scalable !?!

- QP solving needs  $O(n^2)$  time and  $O(n^2)$  memory



\* Graph by M. Vogt

# Solving the SVM QP-Problems

**Matrix formulation:** maximize  $L_d(\boldsymbol{\alpha}) = -\frac{1}{2} \boldsymbol{\alpha}^T \mathbf{K} \boldsymbol{\alpha} + \mathbf{f}^T \boldsymbol{\alpha}$   
subject to 1)  $0 \leq \alpha_i, \alpha_i^* \leq C, i = 1, \dots$   
and 2) 1 equality constraint if working with bias  $b$

## Various Solution Methods Possible:

- **Interior-Point:** precise, batch, not suitable for huge data sets.
- **Active-Set:** robust, precise, maybe slow (?), memory prop. to the # of SVecs.
- **Working-Set** for huge data sets, iterative (chunking), SMO or **ISDA** -> now implemented on the **Yottamine.com** site)

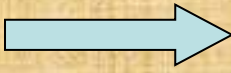
## Available Software:

- **Interior-Point:** universal-routines LOQO, CPLEX, MOSEK, MATLAB's QP solver, ...
- **Working-Set:** implemented in **SVM<sup>light</sup>**, mySVM, SVMTorch, (Hero-SVMs?) ...

**SMO, 2 datapoints only**, implemented in **LibSVM** software

**ISDA, 1 datapoint only** (our algorithm implemented on **Yottamine.com cloud**)

# Recapitulations till now:

- ‘All’ ML models are of the same form i.e., they are  Sum-of-Weighted-Functions
- The most used ones minimize either sum-of-error-squares, or maximize the margin between classes
- The later ones are the most suitable for LARGE data sets (we’ll comment this soon) and their learning amounts to Solving QP Problem with Constraints

**Finally, we have arrived at the  
LARGE DATA SETS!**

**How to handle them?**

**What algorithm is suitable?**

**What hardware i.e. software  
solution fits them the best?**

# As of today, **SVMs only** can successfully deal with **(ULTRA)LARGE Datasets. SVMs only!**

Sorry for such a bold claim, but the explanations below may help to understand it!

- How comes? What about the other ML models? Why is it this way?
- Well, it follows from the SVMs' learning algorithm which is solving the QP problem with  **$N$  inequality constraints** and 1 equality constraint, where **the former**
- **IMPOSE the SPARSENESS ONTO THE SOLUTION!**
- This then in turn, makes the training phase feasible and **expresses the model in terms of a small number of the so-called Support Vectors!**



# There are few possibilities to learn from ultra-large data sets by SVMs

- \* **parallelize** the existing QP solvers
- \*\* **implement 'novel' parallel QP solvers**
- \*\*\* **use GPUs** i.e., manycore machines
- \*\*\*\* **change** the SVMs algorithm through a 'novel' **geometry** based **insights** = hulls and spheres (balls) approaches

# Classic Parallelization

- There was a series of various attempts to parallelize SVMs algorithms on super-computers, clusters and grid machines starting from ~ 2003 and lasting till today.
- Table of examples is on the next 2 slides
  - the NEC Labs' patented **cascade SVM parallelization approach** (Graf et al & Vapnik, 2006) is not forgotten in the next table - check it at NIPS 2004. It belongs to the item **\*parallelize the existing QP solvers**, from previous slide)

<b>Author</b>	<b>Processor</b>	<b>Algorithm</b>	<b>Training Speed up</b>	<b>Testing Speed up</b>
2003, Zanni	MPI (Cray T3E, 32 processor)	VPDT variable projection decomposition technique	1.8 -6.1 (2 - 16 processor)	N/A
2005, Serafini and Zanni	Cluster	PGPDT A Parallel Gradient Projection-based Decomposition Technique for Support Vector Machines	5.2 (8 processor / single processor)	N/A
2006, Cao et al.	MPI (Cluster of multiple CPUs)	PSMO Parallel SMO	93 (over SVM and LIBSVM) (32 Processor)	N/A
2006, Serafini and Zanni	Cluster	PGPDT A Parallel Gradient Projection-based Decomposition Technique for Support Vector Machines	7.3 MNIST 12.8 Cover test (16 processor / single processor) 2 - 25 KDDCUP (24 - 32 processor / single processor)	N/A
2007, Chu et al.	Cluster (Map-Reduce)	SMO	1.6 - 1.96 (2 core/1 core)	N/A
2007, Dominik Burgger	Kepler Cluster (Every node has two cores) (MPI)	$\pi$ SVM Extension of LIBSVM	3.8 - 16 (LIBSVM)	N/A

Author	Processor	Algorithm	Training Speed up	Testing Speed up
2008 , Thanh-Nghi Do et al.	Nvidia GeForce 8800 GTX	LS-SVM Extended Least Squares SVM	47 - 100 (over LIBSVM on CPU)	N/A
2008, Catanzaro et al.	Nvidia GeForce 8800 GTX GPU, single precision	SMO	9-35 (GPU Adaptive over LIBSVM) 81-135 (GPU over LIBSVM) 524 (GPU over CPU)	N/A
2009, Carpenter	NVIDIA GTX 260 GPU	SMO (cuSVM) mixed precision algorithm	17-32 (over LIBSVM)	<u>22-172 (normal CPU)</u>
2009, Harvey	2 GPU	GPU SVM	89 - 263 (LIBSVM)	N/A
2009, Meligy	Grid Based (C and MPI)	DSVM (Distributed SVM) PSVM (parallel of Support Vector Sector Machine)	not implemented	N/A
2009, Woodsend	Hybrid MPI/OpenMP Cluster (quad-core)	OOPS (Object-Oriented Parallel Solver)	2.2 - 2066 (Milde) 43 - 125 (PSVM) 94 - 206 (PGPDT)	N/A
2010, Lopez et al.	NVIDIA Tesla C1060 GeForce 8800 GT	P2SMO Parallel-Parallel SMO	3 - 57 (Training) 3 - 112 (Classification)	N/A

Thus, it seems that GPUs are the tool of the day.  
Hence, we have developed SVMs algorithms for GPUs.

# SVMs code on GPUs

## developed at VCU



Tesla card S1060 (first series)

8 Tesla GPUs in 4U server



# GPUSVM Experimental Results for Benchmark Datasets

- Performance comparisons **between LIBSVM** and **GPUSVM** on both **accuracy** and **speed** will be shown on next **8** slides.
- **Accuracy** comparison:
  - Small datasets: Accuracies are shown for training sets.
  - Medium datasets: Accuracies are shown for both training and testing sets.
  - Large datasets: Accuracies are shown for testing sets.
- **Speed** comparison:
  - Small datasets: The training time is too trivial to be shown.
  - Medium/Large datasets: The training /testing time are shown for standard **LIBSVM ( using Xeon 1-core), OpenMP enabled**

# GPUSVM Benchmark Datasets for Hyperparameters $C$ and $\gamma$



Scale	Dataset	# of training data	# of testing data	# of features	# of classes	$C$	$\gamma$
small	iris	150	N/A	4	3	16	0.5
	heart	270	N/A	13	2	0.5	0.0625
	breast-cancer	683	N/A	10	2	0.25	0.125
medium	usps	7,291	2,007	256	10	128	0.015625
	shuttle	43,500	14,500	9	7	1	1
	mnist	60,000	10,000	780	10	16	0.003096
large	covtype	500,000	81,012	54	7	1	1

# GPUSVM & LIBSVM Accuracy Performance Comparisons

## Small datasets

Dataset	SVM	Training accuracy	# of SVs
	GPUSVM	98.1308%	144
	GPUSVM	98%	27
	GPUSVM	99.4383%	75
	GPUSVM	85.1852%	146
	GPUSVM	100%	150
	GPUSVM	97.2182%	91

## Medium datasets

Dataset	SVM	Training accuracy	Testing accuracy	# of SVs
	GPUSVM	85.7928%	85.0193%	11587
	GPUSVM	99.9863%	95.715%	1923
	GPUSVM	99.8467%	97.38%	11936
	GPUSVM	99.4736%	99.5655%	3667
	GPUSVM	99.4553%	99.4515%	35220
	GPUSVM	99.4617%	98.27%	12919



# GPUSVM & LIBSVM **Accuracy** Performance Comparisons

## Large datasets

Dataset	SVM	Testing accuracy	# of SVs
---------	-----	------------------	----------

	GPUSVM	99.2332%	38598
--	--------	----------	-------

	GPUSVM	80.3362%	267373
--	--------	----------	--------

	GPUSVM	98.037%	34992
--	--------	---------	-------

# GPUSVM & LIBSVM Speed Performance Comparisons

Medium datasets

Dataset	SVM	Processor	Training time	Speedup	Testing time	Speedup
Usps 7,291	GPUSVM	Xeon 12-core	8.998s	6.7386x	2.216s	9.1485x
		Tesla C2070		7.9405x		31.2373x
	LIBSVM	Xeon 1-core	4.901s	1x	2.113s	1x
		Tesla C2070	2.158s	2.2711x	0.081s	26.0864x
Shuttle 43.500	GPUSVM	Xeon 12-core	11.902s	3.1712x	1.88s	2.4819x
		Tesla C2070				
	LIBSVM	Xeon 1-core	9.379s	1x	2.402s	1x
		Tesla C2070	2.238s	4.1908x	0.526s	4.5665x
Mnist 60,000	GPUSVM	Xeon 12-core	199.784s	7.2625x	6.819s	8.6931x
		Tesla C2070		20.3523x		48.7083x
	LIBSVM	Xeon 1-core	256.579s	1x	86.559s	1x
		Tesla C2070	39.552s	6.4871x	1.124s	77.0098x

# GPUSVM & LIBSVM Speed Performance Comparisons

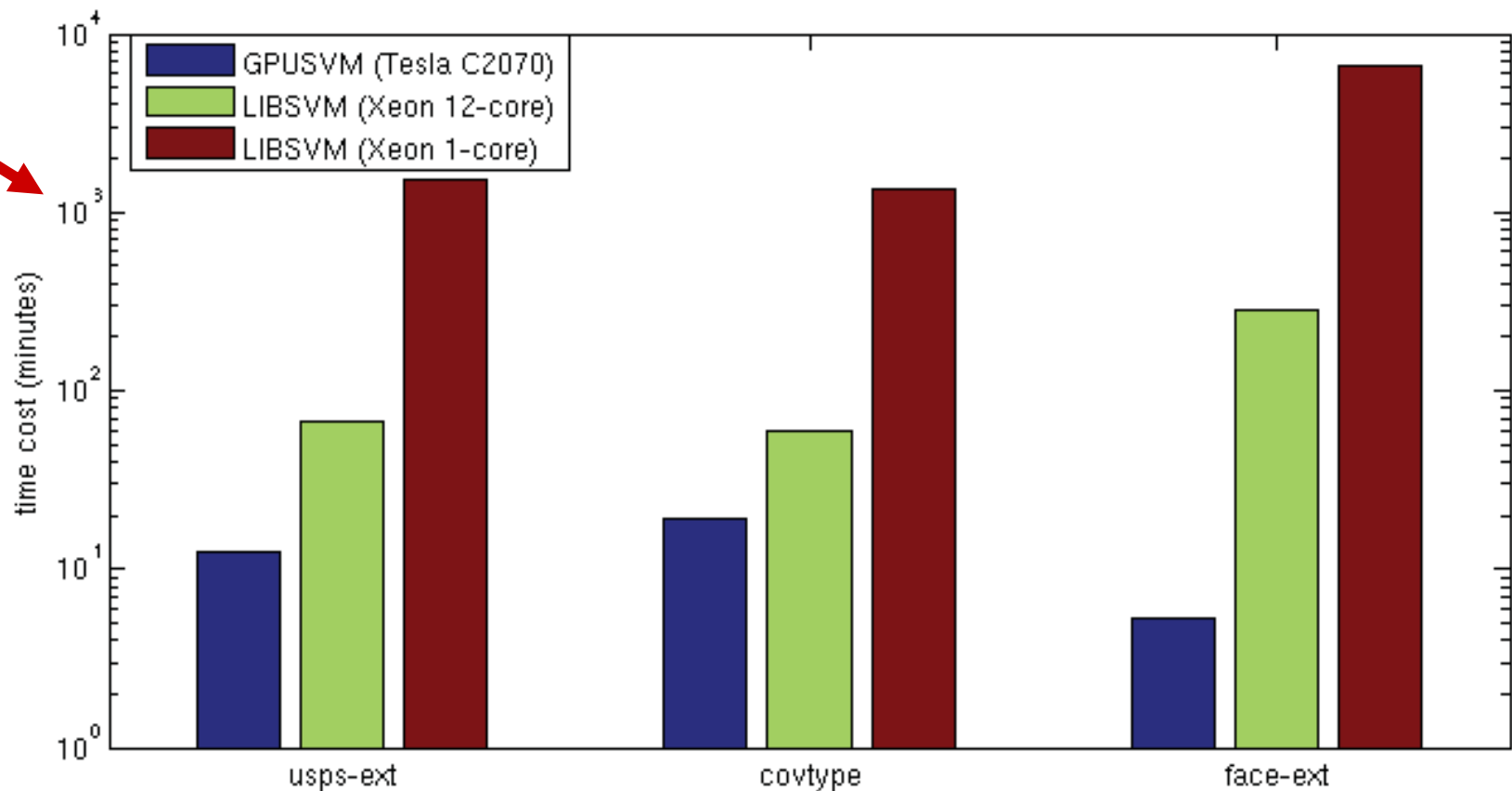
## Large datasets

Dataset	SVM	Processor	Training time	Speedup	Testing time	Speedup
Covtype 500,000	GPUSVM	Xeon 12-core	66.4m	22.8x	8.4m	22.7x
				<b>180x</b>		<b>381.4x</b>
	LIBSVM	Xeon 1-core	1347.7m	1x	198m	1x
	GPUSVM	TeslaC2070	19.4m	<b>69.5x</b>	0.7m	<b>282.9x</b>
	GPUSVM	Xeon 12-core	286.5m	22.77x	8.5m	22.9x
	GPUSVM			<b>1230.7x</b>		<b>650x</b>

# Graph for training time comparisons between GPUSVM and LIBSVM

## Large datasets

Note the logarithmic scale here. Thus, we are talking about the ORDER OF MAGNITUDES SPEED UP.



# GPUSVM & LIBSVM Performance Comparison **Summary**

- **Accuracy** performance comparisons:
  - **GPUSVM is as accurate as LIBSVM.** Both use same working set technique (SMO) for solving QP problems.
  - **GPUSVM** uses **single** precision floating point and **LIBSVM** uses **double** precision floating point. (This causes the slight difference between the total number of support vectors acquired through the learning phase and their corresponding alpha values. No effects on the accuracy whatsoever!)
  - **GPUSVM** uses **OvA** for multiclass problems while **LIBSVM** uses **OvO**. This also causes a tiny accuracy performance differences.
- **Speed** performance comparisons:
  - **LIBSVM can be accelerated** by enabling the built-in **OpenMP** feature which utilizes the full power of multi-core CPU.
  - **GPUSVM** has **close** performance on **medium** datasets compared to **LIBSVM with OpenMP** in training phase. However, GPUSVM is always faster than OpenMP enabled LIBSVM in

**Now, let's move from the  
accelerations based  
primarily on hardware to the  
speeding up by a 'new',  
geometry inspired,  
algorithm(s) i.e., software**

*22 more slides to go!*



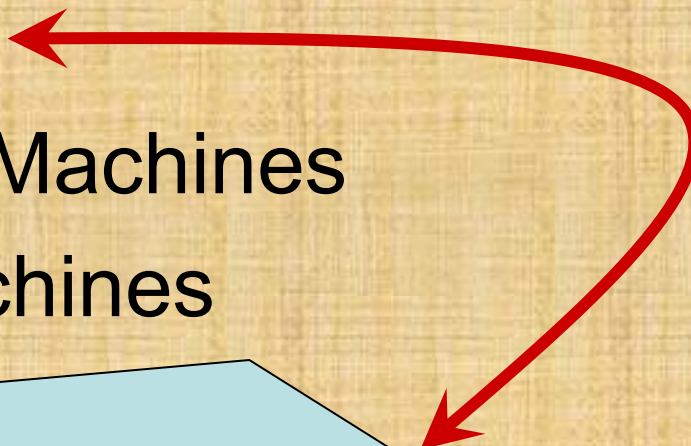
*I am enjoying it 😊 till now,  
indeed!  
What about you?*

The 'novel' approaches,  
seemingly promising for  
(ultra)large datasets, are  
based on **geometric**  
insights disguised in the  
shapes of **hulls** and  
**spheres (balls)**



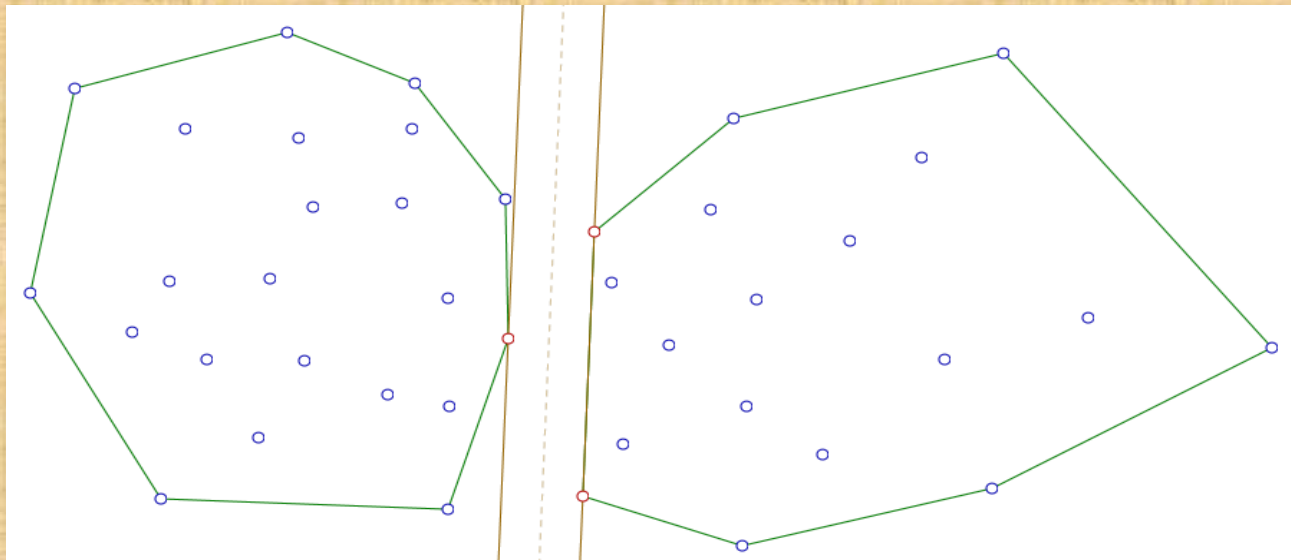
# SVM - Geometric Approaches

- Convex Hulls
- Core (**Ball**) Vector Machines
- **Sphere** Vector Machines



We've played with hulls,  
and we abandoned them for  
now, but the basic idea is

-find two closest points belonging to the two Convex Hulls



SVMs as the Reduced Convex Hulls

# Reduced Convex Hulls

- Can be solved using existing algorithms:
  - Closest Point Problem
    - Gilbert's algorithm
  - Nearest Point Problem
- Mitchell-Dem'yanov-Malozemov
  - Schlesinger-Kozinec
- Non-separable problems can be solved using Reduced Convex Hulls
- **Usually slower than SMO implementations and thus put aside for now**

# Core i.e., Ball, Vector Machines

- Solving minimal enclosing ball problem

$$\arg \min_{R, \mathbf{c}} R^2$$

$$\forall_i \|\mathbf{c} - \varphi(\mathbf{x}_i)\|^2 \leq R^2$$

- is equivalent to solving a modified  $L2$  SVM

$$\arg \min_{\mathbf{w}, b, \zeta} \frac{1}{2} \|\mathbf{w}\|^2 + \frac{b^2}{2} - \rho + \frac{C}{2} \sum_i \zeta_i^2$$

$$\begin{aligned} \forall_i y_i(\mathbf{w}x_i + b) &\geq \rho - \zeta_i \\ \forall_i \zeta_i &\geq 0 \end{aligned}$$

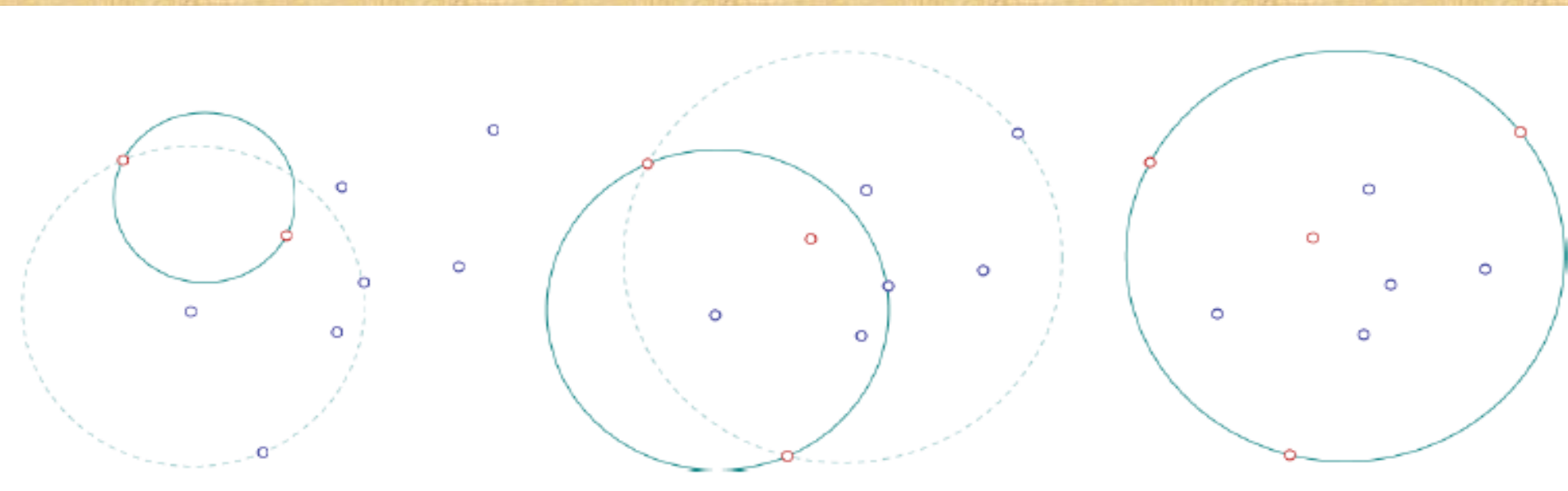
- in a feature space defined by kernel

$$\tilde{k}_{ij} = y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) + \frac{\delta_{ij}}{C} + y_i y_j$$

# Core Vector Machines

At each iteration:

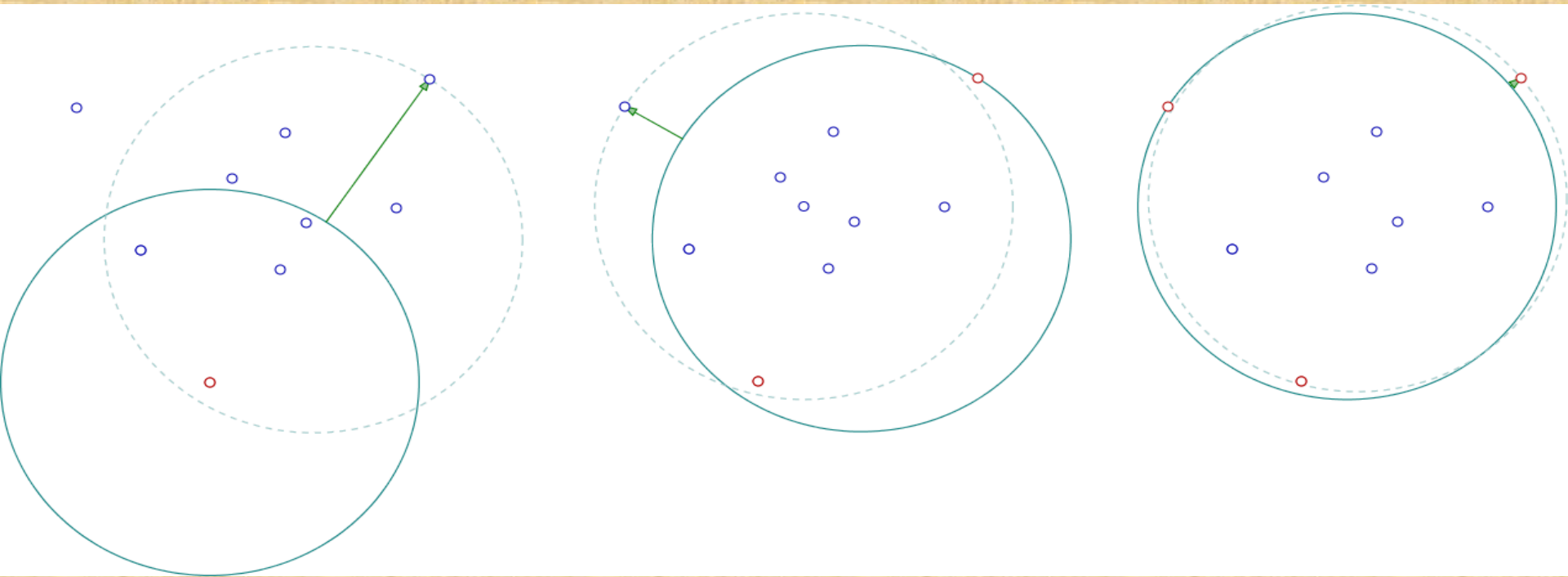
- one violating point is added to the core-set
- Minimum Enclosing Ball problem is solved for all points belonging to the core-set (**using SMO algorithm**)



# Ball Vector Machines

At each iteration:

- instead of solving entire QP problem just one update is performed - ball is shifted towards the **max violating point**



# Enclosing Sphere Machines (ESM)

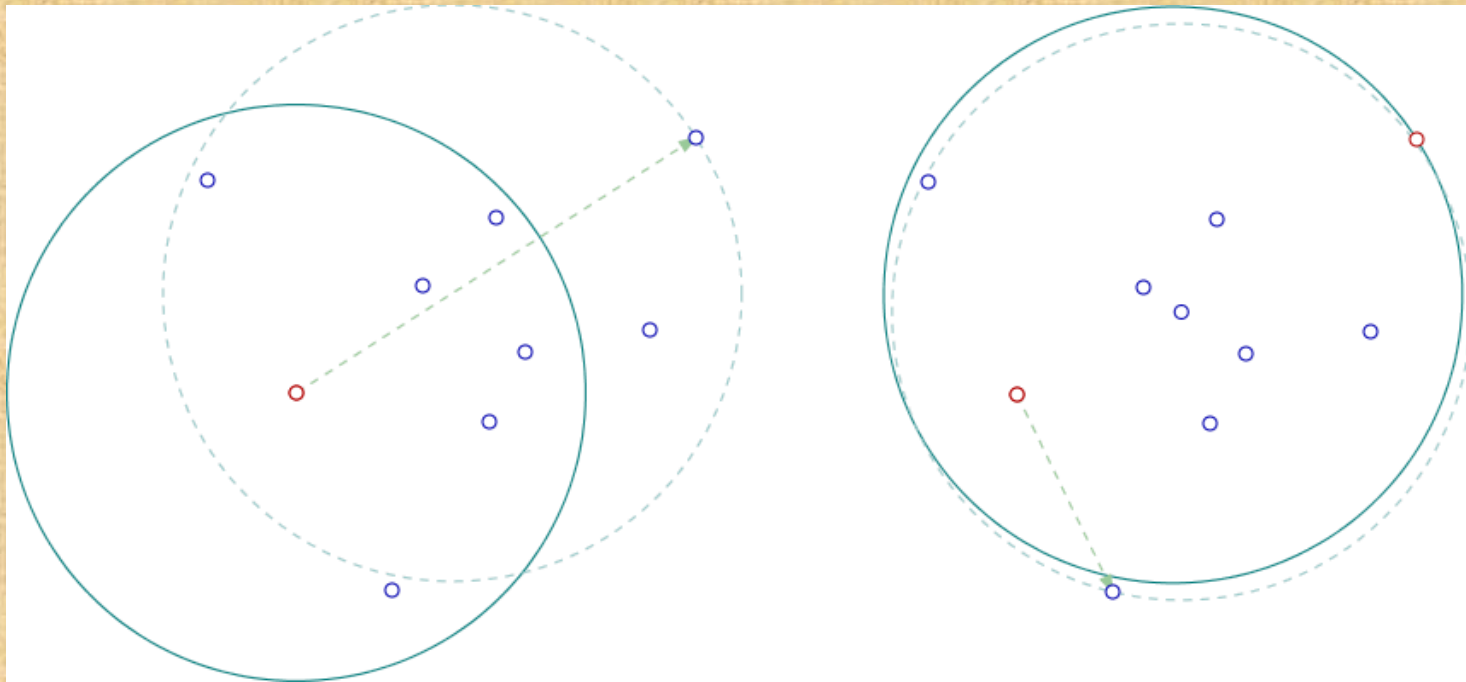
**Our approach**

At each iteration two vectors are found:

- one that violates “ball enclosing” conditions
- one that violates KKT conditions:

$$\forall i : \alpha_i \left( R^2 - \|\mathbf{c} - \varphi(\mathbf{x}_i)\|^2 \right) = 0$$

and ball is shifted along the line joining these two vectors



Now only, we present results of ***extremely extensive comparisons*** of one of the most powerful & possibly the most used off-shelf SVM software **LIBSVM** (both  $L_1$  &  $L_2$  models) vs. the last two **geometric approaches (balls and spheres)** for training SVMs, in a very strict

**DOUBLE (NESTED) *k*-fold CROSS VALIDATION** i.e. **RESAMPLING**

**experiment**



Remind!

***k*-fold CROSS VALIDATION** is for  
**MODEL** (i.e., its **HYPERPARAMETERS**)  
**SELECTION**

while a

**DOUBLE (NESTED) *k*-fold CROSS  
VALIDATION** i.e., **RESAMPLING** is for  
**MODELS COMPARISONS**

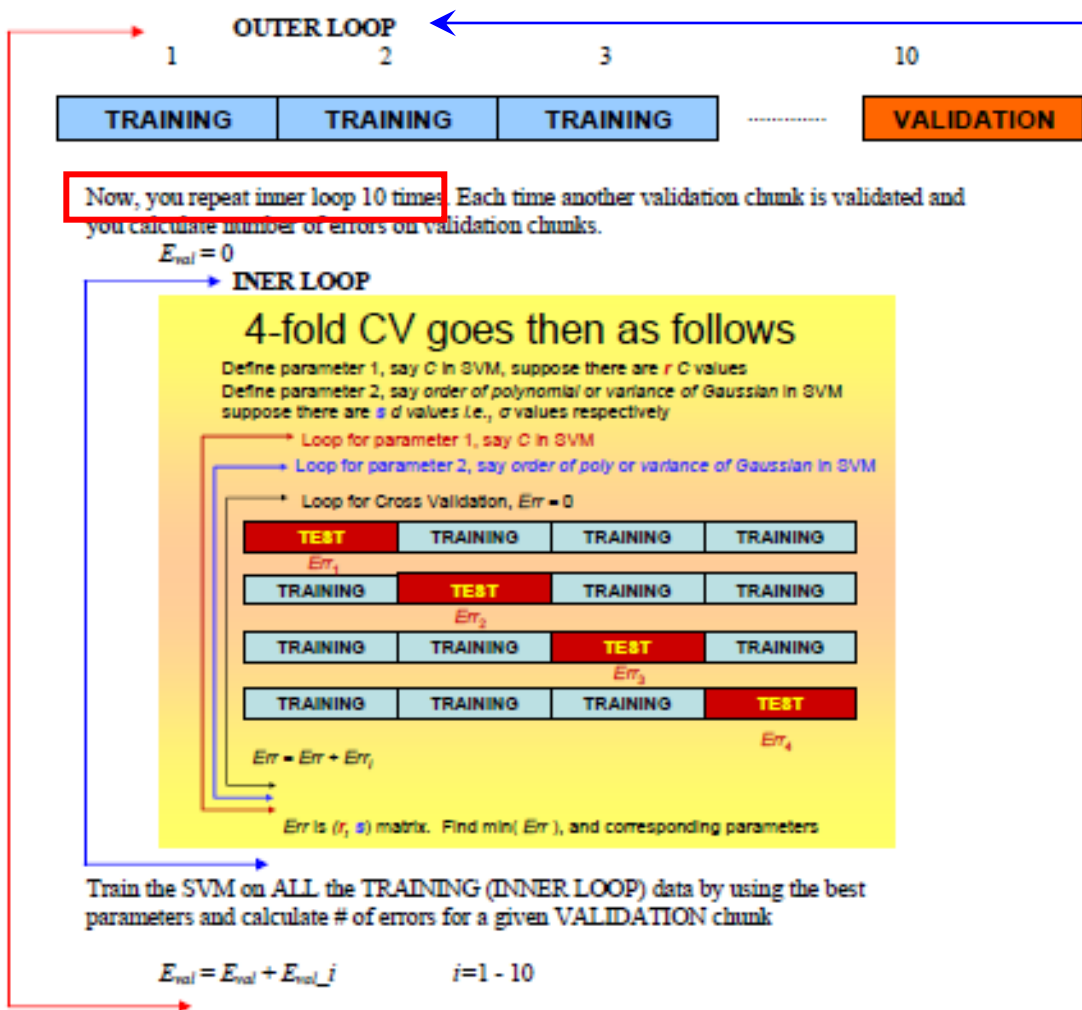
DOUBLE i.e., NESTED CV is used for **MODEL COMPARISONS** and it goes as follows:

There are two loops. The Outer loop and the Inner one.

In each one you do a  $k$ -fold CV.  $k_o$  is not necessarily equal to  $k_i$ .

Say  $k_o = 10$ , and  $k_i = 4$

In outer loop you make 10 roughly equal splits



Environment of our experiments was as follows

SVMs with Gaussian kernel

Double **5x5 CV**,

**8x8** hyperparameters ( $C, \sigma$ )

which amounts to

**1600** runs for each dataset

Runs for each dataset have

been performed on **5**

**Xeon E5520 2.3 GHz CPUs**

Training time is then

summed up i.e., given as a

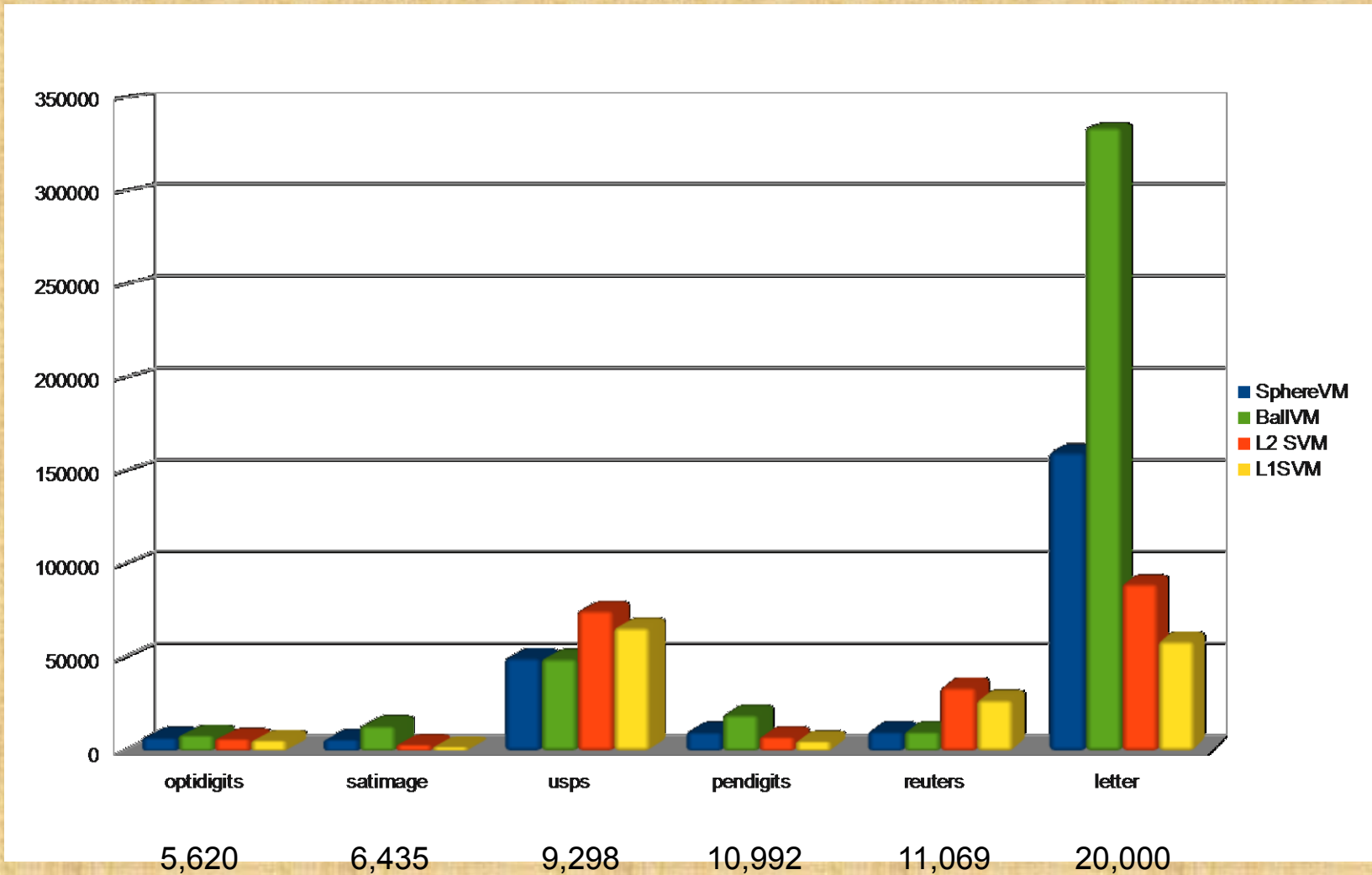
single CPU time needed.

$E_{val}$  calculated for different models (SVM, Dec. tree, ALH, k-NN) are compared and winner is declared.

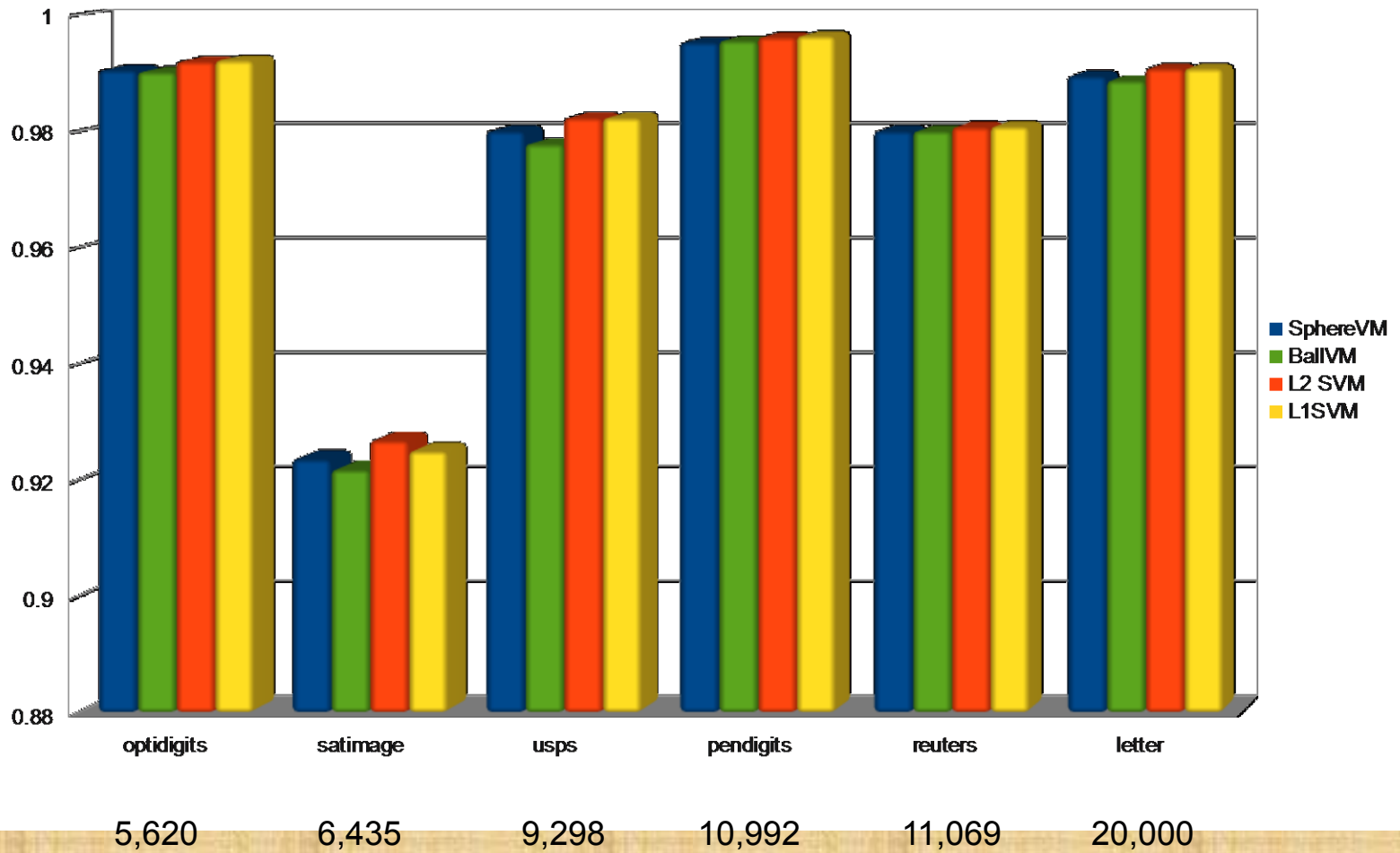
# Comparisons results for datasets below

Data set	Number of classes	Number of attributes	Number of samples	
optdigits	10	64	5,620	<b>S</b>
satimage	6	36	6,435	
usps	10	256	9,298	
pendigits	10	16	10,992	<b>M</b>
reuters	2	8,315	11,069	
letter	26	16	20,000	
adult	2	123	48,842	
w3a	2	300	49,749	
shuttle	7	7	58,000	
web	2	300	64,700	<b>L</b>
ijcnn1	2	22	141,691	
intrusion	2	127	5,209,460	<b>UL</b>

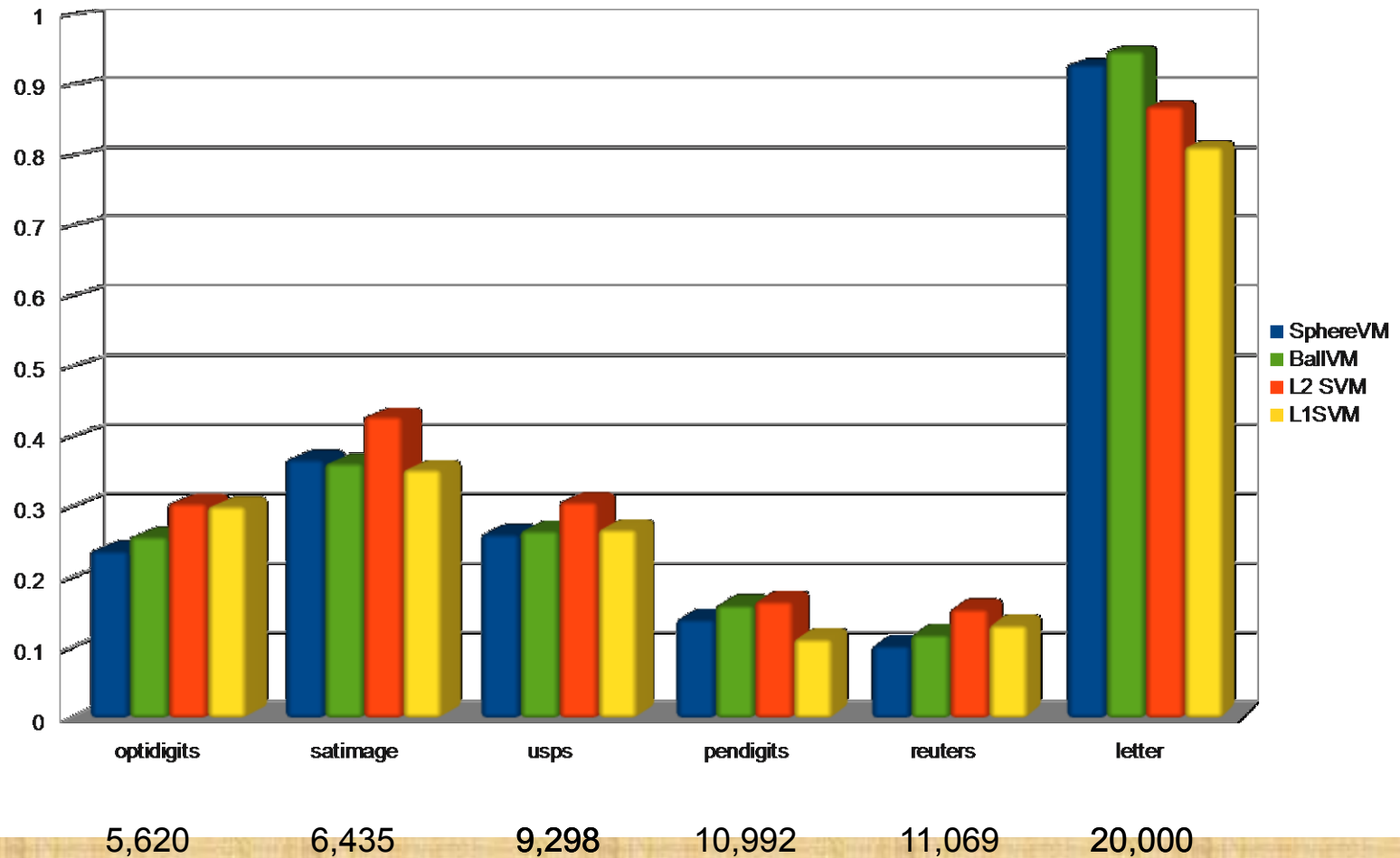
# Learning time - S & M data sets



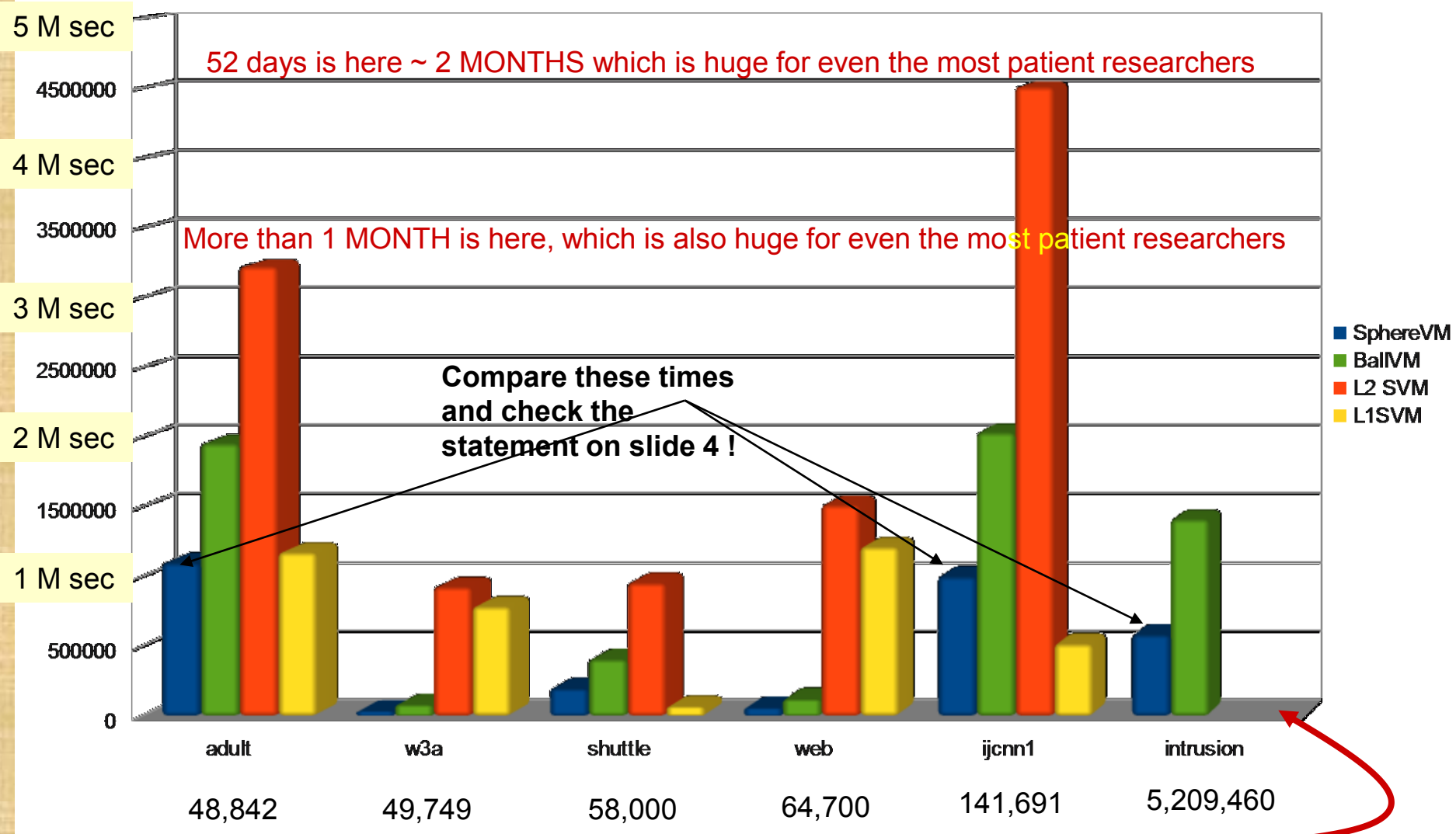
# Accuracy - S & M data sets



# Ratio of number of SVs S & M data sets

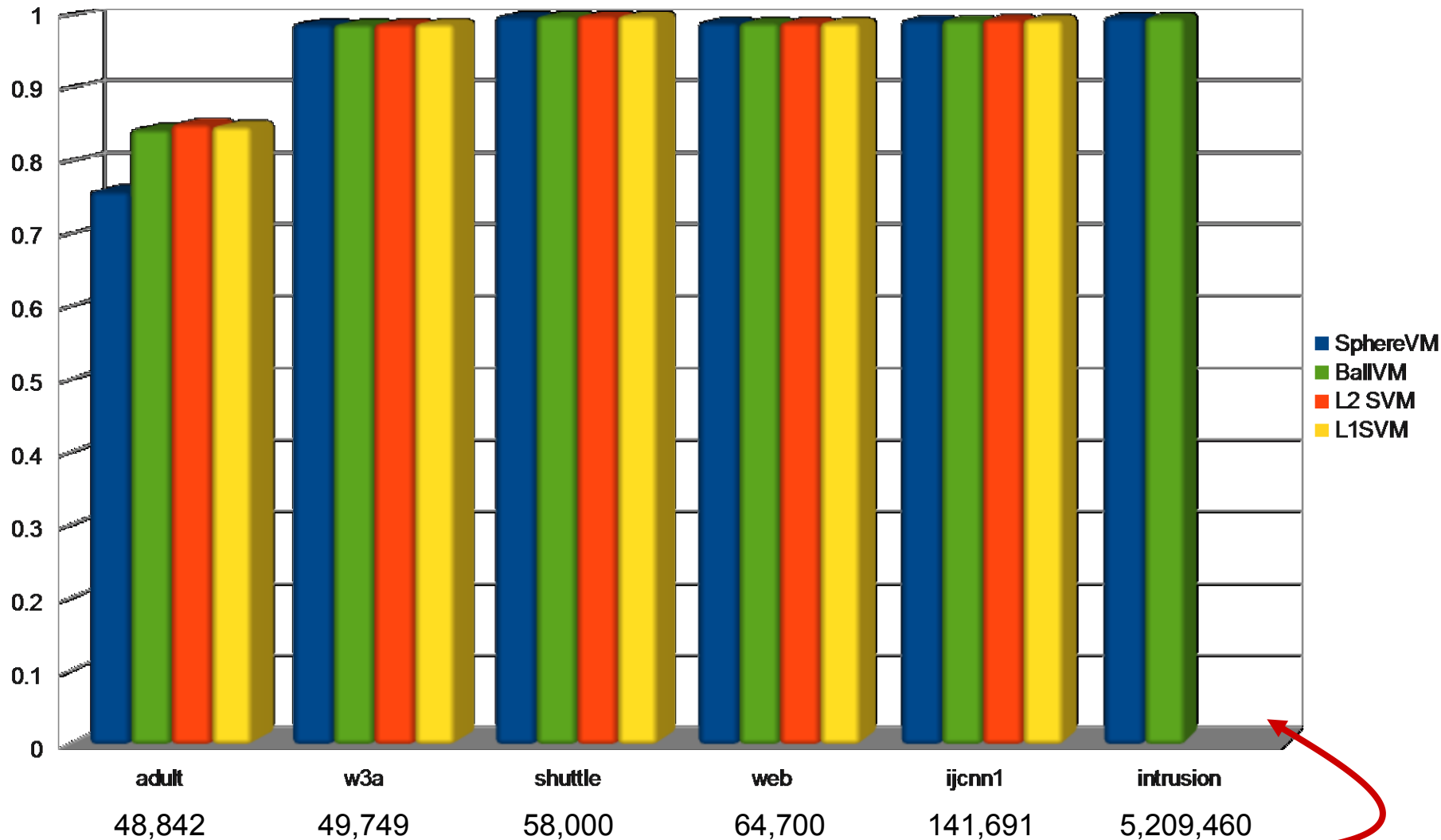


# Learning time – M & L & UL data sets



Notice that both LIBSVMs were not able to finish the learning here. L1 LIBSVM needed 60h/1 iteration only 71/79

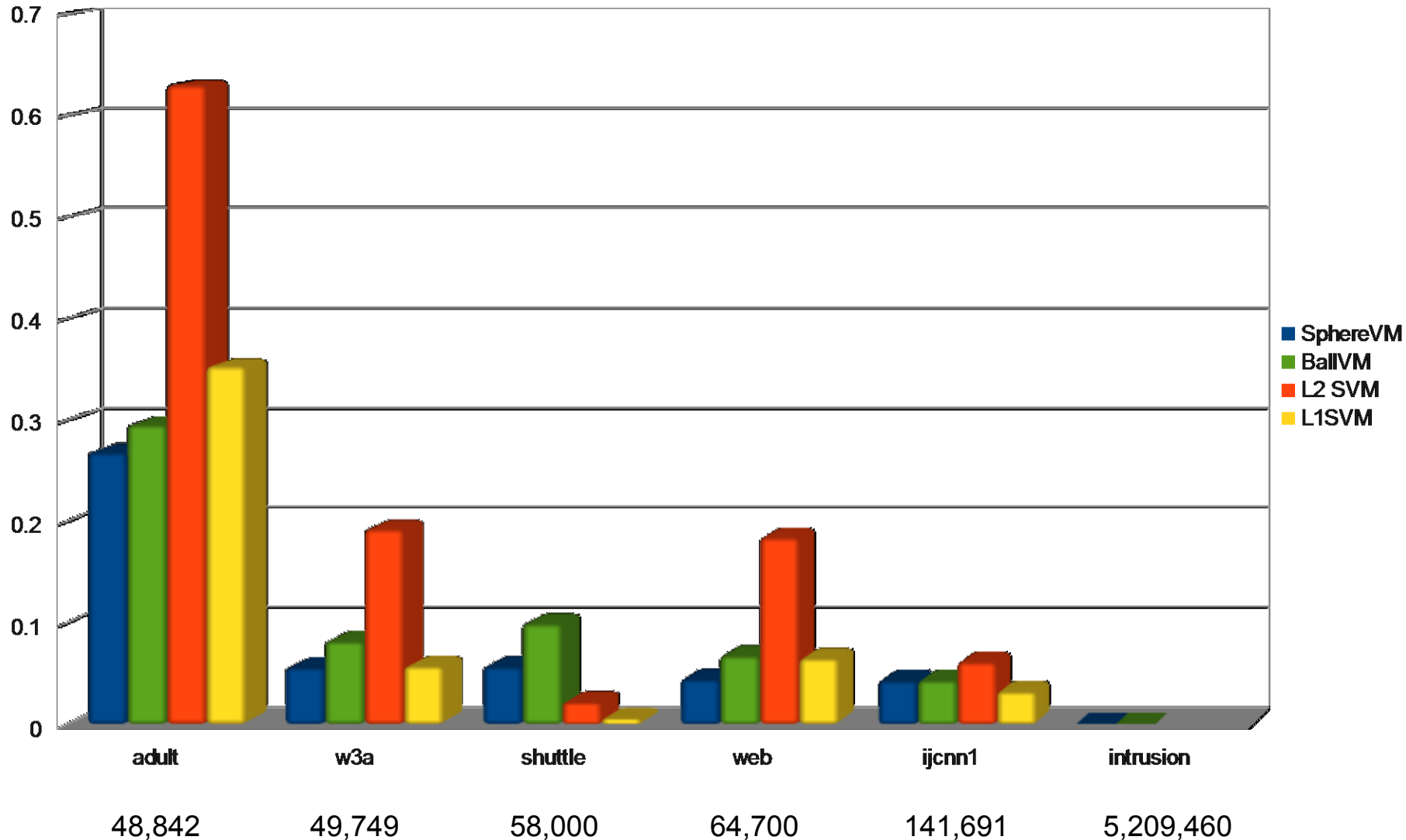
# Accuracy - M & L & UL data sets



Notice that both LIBSVMs were not able to finish the learning here. L1 LIBSVM needed 60h/1 iteration only **72/79**

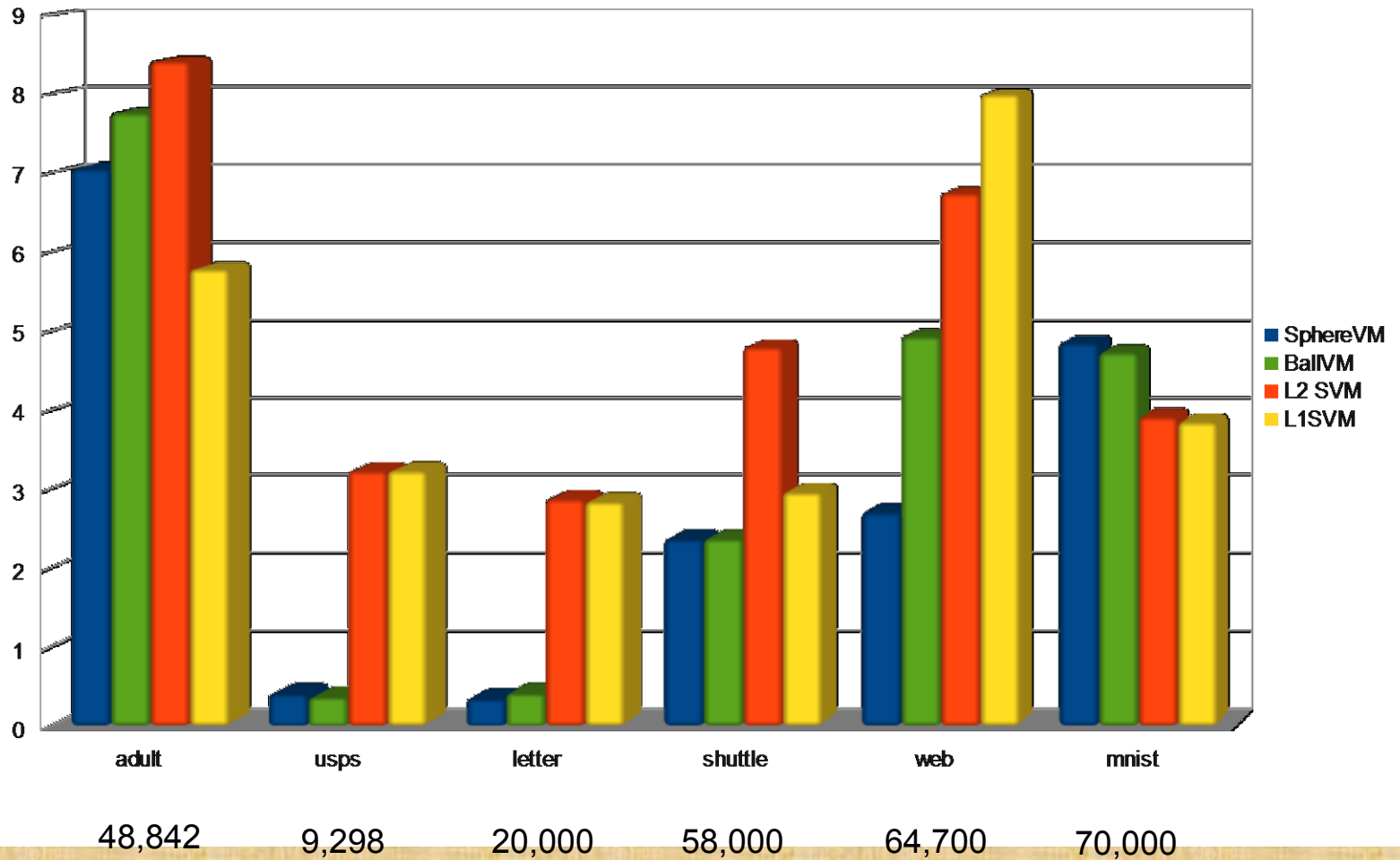


# Ratio of number of SVs M & L & UL data sets



# Multithreading by OpenMP\*

## A speedup for 12 threads



Thus, **sphere SVMs** seem to offer both a capacity to handle, and significant accelerations for, both **HARD** (not necessarily UL) and **ULTRALARGE** datasets (**over 1 mil samples**).

The very next (we believe a feasible) step may well be implementing spheres on GP GPUs speeding them even more

## Data set

- 1 optdigits
- 2 satimage
- 3 usps
- 4 pendigits
- 5 reuters
- 6 letter
- 7 adult
- 8 w3a
- 9 shuttle
- 10 web 11
- ijcnn1 12
- intrusion

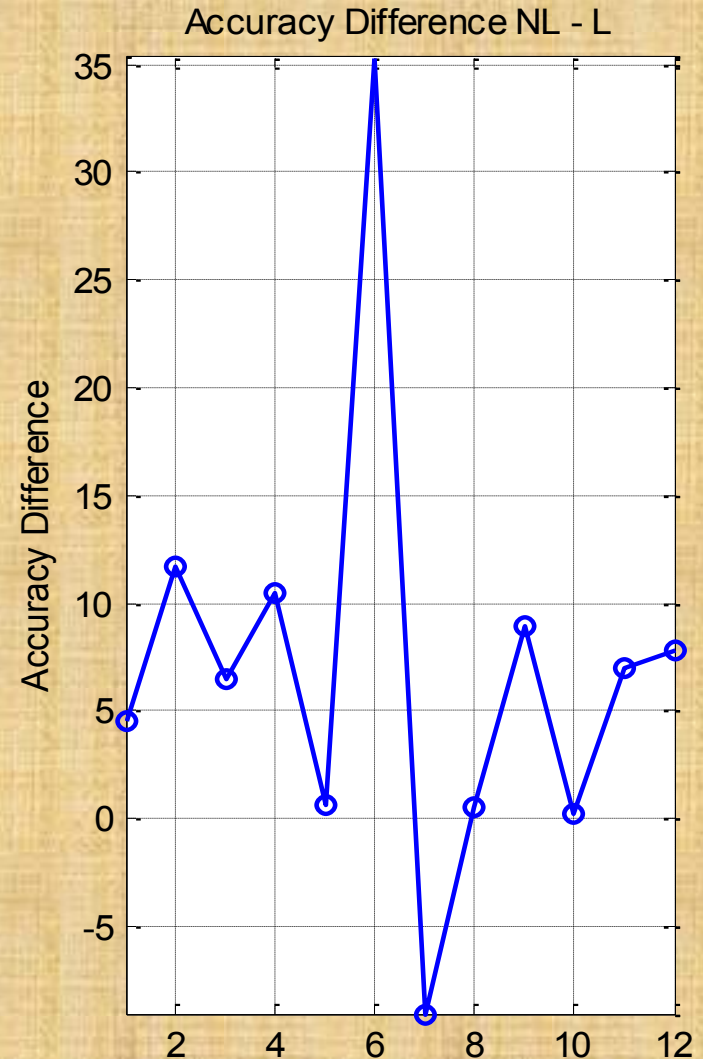
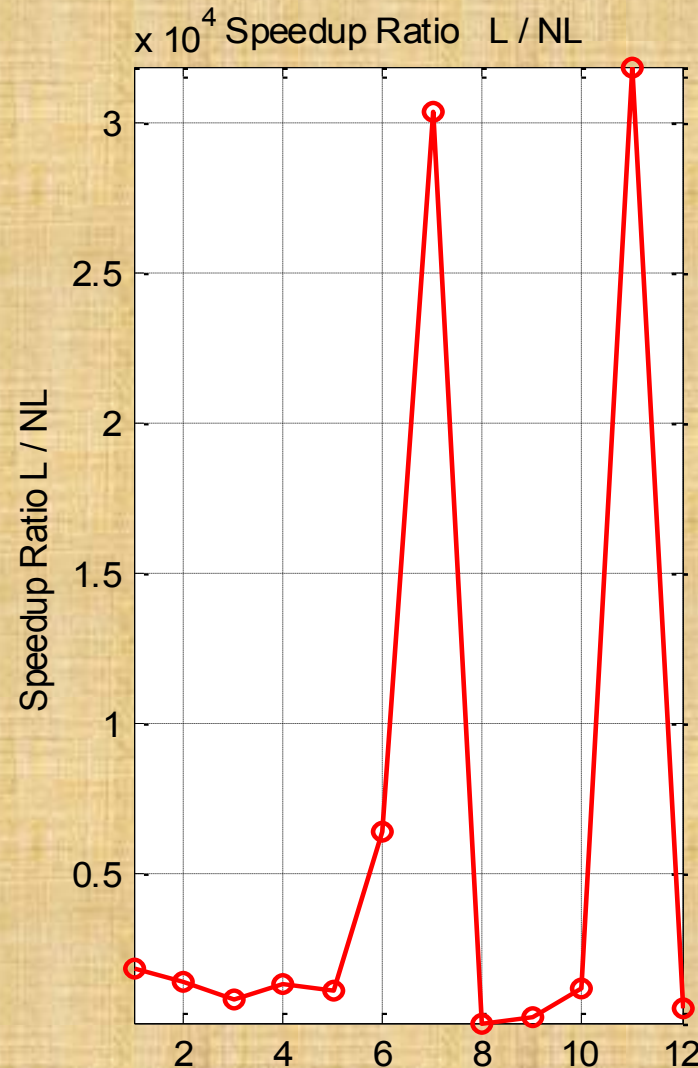
## # of data

- 1 - 3,823
- 2 - 4,435
- 3 - 7,291
- 4 - 7,494
- 5 - 7,770
- 6 - 15,000 M
- 7 - 32,561 M
- 8 - 4,912
- 9 - 43,500 M
- 10 - 49,749 M
- 11 - 49,990 M
- 12 - 4,898,431 UL

The Last But Not the Least

# Always Run Linear SVM First

Here, we run our LINEARSVM



It's **no time** (yet) for **CONCLUSIONS** on the topic of learning from HUGE datasets, except that

- An ever-increasing number of data samples requires **rethinking** about how to approach the machine learning tasks
- The very rethinking must include advances in both **HARDWARE** and **ALGORITHMS**
- **GPU manycore** processors are the first obvious choice for the hardware right now
- The next good option is to use some ideas from the **geometry**
- **Our spheres algorithm** for training SVMs have been successfully implemented and presented

# Thanks for both being patient and having stamina

• Q  
- U  
• E  
- S  
» T T T T T  
- I  
• O  
- N  
• S



*PLEASE !!!*