Machine Learning

New Requirements and New Tools

Basics of Support Vector Machines

Vojislav Kecman, The University of Auckland, Auckland, NZ

THE UNIVERSITY OF AUCKLAND
FACULTY OF ENGINEERING

What is Learning from Data, a.k.a. Machine Learning, a.k.a. **Data Mining**?

- The Math in the last 2,000 years was playing with such models:

- $A = \pi r^2 = w_1 r^2$, $v = \text{sqrt}(2gh) = w_1 \text{sqrt}(h)$,
- $y = 3x - 2 = w_1x + w_2$, $z = -x + y - 3 = w_1x + w_2y + w_3$

Parameters w_i of the relations **are known**, and given the independent variable(s) the task is to find the dependent one(s)!

- TODAY; we want to learn from the pairs of the measured data, and to learn the **UNKNOWN** parameter values.
- This is an **INVERSE PROBLEM** stated as:

- **having the training pairs (x_i, y_i) find the parameters w_i, of the model, or LEARN the dependency between the x_i and y_i!**

Contents

The talk today will be on how one learns from experimental data.

WHY?

Because we live in an information age? - Possibly!
Because we live in a knowledge society? - Possibly YES!

Because we live surrounded by an **OCEAN OF 'DATA'?**
YES, FOR SURE!

And, I mean **ALL** possible 'data' because, we and our devices are surrounded by all imaginable measurements, images, sounds, smells, etc.

We want - to produce data, to transfer it, to compress it, to use it, to process it, to reuse it, to filter it, etc .

But primarily, we want to LEARN FROM DATA, a.k.a., examples, samples, measurements, records, observations, patterns

CLASSIC applications:

- increase in sleep depending on the drug,
- pulmonary function modeling by measuring oxygen consumption,
- head length and breadths of brothers,
- classification of the Brahmin, Artisan and Korwa caste based on physical measurements,
- biting flies (genus: *Leptoconops*) data for classification of the two species of flies,
- battery-failure data dependency and regression,
- various financial and market analysis (bankruptcy, stock market prediction, bonds, goods transportation cost data, production cost data, etc.),
- study of love and marriage regarding the relationships and feelings of couples,
- air pollution data classification, college test score classification and prediction, crude oil consumption modeling, closeness between 11 different languages, and so on.

(all of the above were **linear** models, taken from 20 years old statistics books)

TODAYS (primarily **NON-linear**) applications:

Note the following strong fact -> there is no field of human activities today, left untouched by learning from data!!!

Statistical learning is very, very hot nowdays - find patterns, identify, control, make prediction, make decisions, develop models, search, filter, compress, …, and some <u>today's applications are:</u>

- **computer graphics, animations,**
- **image analysis & compression, face detection, face recognition,**
- **text categorization, media news classification, multimedia (sound video) analysis**
- **bioinformatics - gene analysis, disease's study**
- **time series identification - financial, meteorological, hydro,**
- **biomedicine signals, all possible engineering signal processing**
- **predictions - sales, TV audience share, investments needed, ..etc.**

Now, some basics of a

Bias – Variance - Dilemma!

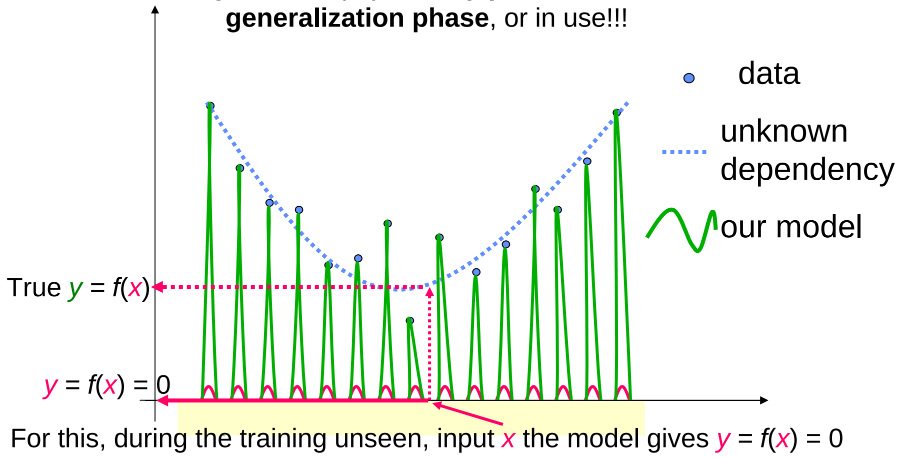
It is the must piece of the knowledge in order to get an idea of the relationship between the data, models and errors!

It will be intuitive, without math or any equation, but it will serve for warming up!

Training and Generalization

Today, having powerful computers and good math software it is easy to be **'great** and **perfect'** on the training data set!

However, such a 'greatness' pays **heavy price at unseen data**, i.e., in a **generalization phase**, or in use!!!

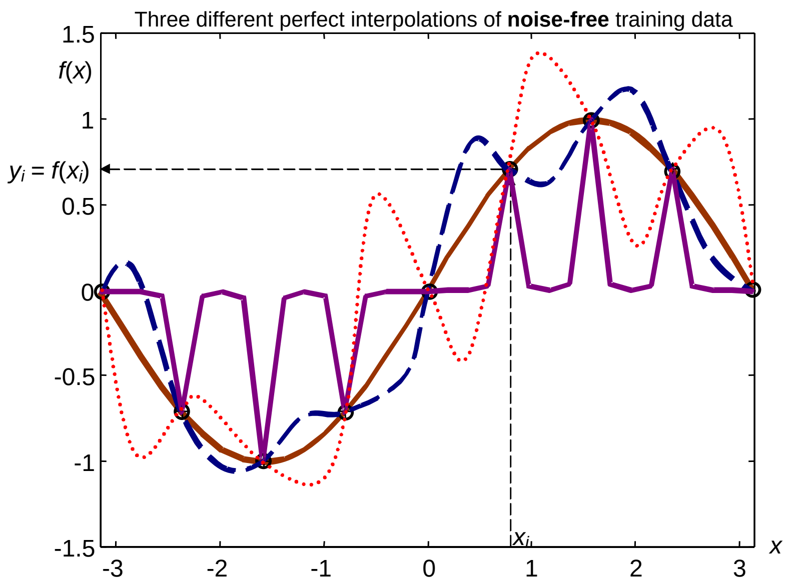


For this, during the training unseen, input x the model gives $y = f(x) = 0$

This is (deliberately chosen) extremely bad modeling, **but real!**
The same or similar phenomena will be present in the high dimensional cases, too!

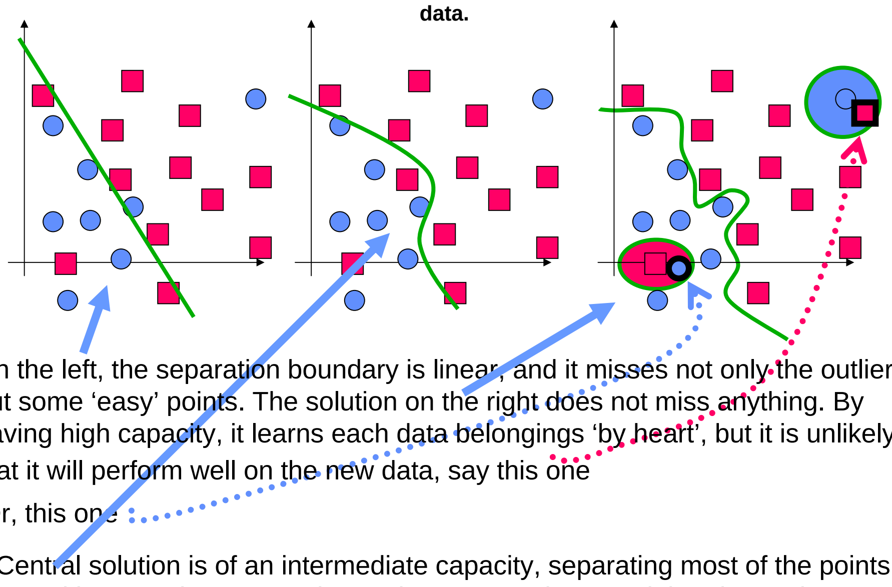
One more example showing **the perfect training results,** but **very bad generalization ones.**

Note that all three models have the training error equal zero! Bias = 0! Perfect interpolants!



And, still one more example, but now from the PATTERN RECOGNITION (CLASSIFICATION) task, showing various models and their performances.

Note that the last model (learning machine) learns perfectly, i.e., **separates all the training data.**



On the left, the separation boundary is linear, and it misses not only the outliers, but some 'easy' points. The solution on the right does not miss anything. By having high capacity, it learns each data belongings 'by heart', but it is unlikely that it will perform well on the new data, say this one

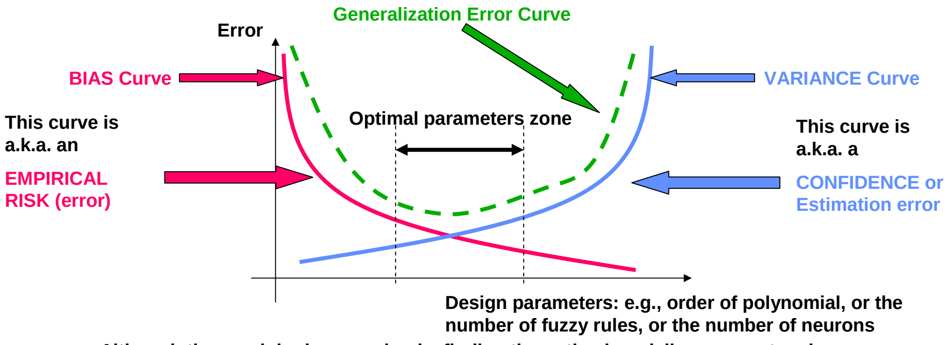
Or, this one

Central solution is of an intermediate capacity, separating most of the points, without putting too much trust into any particular training data point!!!

Obviously, we need much <u>more</u> than being good (or even excellent) on the training data set!

This 'more' means, we want that our models perform well on all future, previously unseen data, generated by the same data generator (i.e., plant, system, process, probability distribution).

The whole statistical learning fights (optimizes) the following two curves!

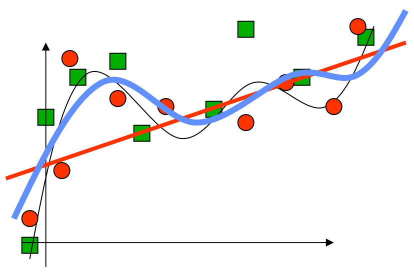

Generalization Error Curve
Error
BIAS Curve
VARIANCE Curve
This curve is a.k.a. an
EMPIRICAL RISK (error)
Optimal parameters zone
This curve is a.k.a. a
CONFIDENCE or Estimation error
Design parameters: e.g., order of polynomial, or the number of fuzzy rules, or the number of neurons

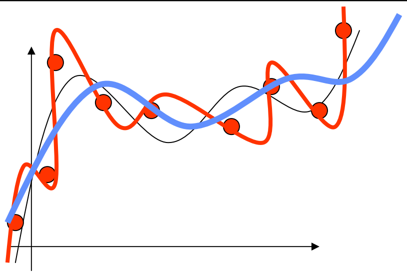
Although the graph looks very simple, finding the optimal modeling parameters is an

EXTREMALY DIFFICULT task!!!

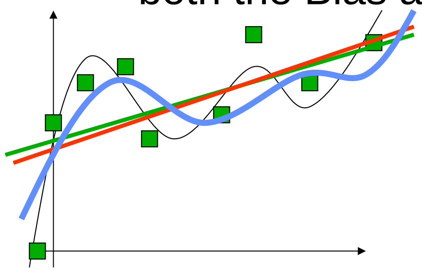
and, this is due to the following facts:

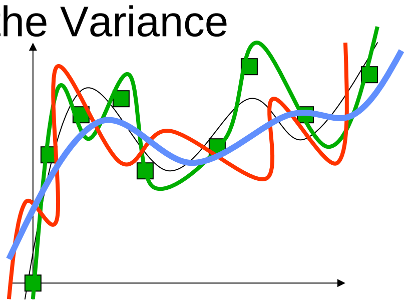
- **we never (or, rarely only) know the underlying probability distribution, meaning the data generation, function,**
- **we never know the space of (<u>target</u>) functions, or to which class of functions our f. belongs**
- **we always have scarce (insufficient, not enough) data,**
- **our data are always high- (or/and extremely high-) dimensional,**
- **there is always the noise, or data are corrupted**





Good model designer would try to get medium both the Bias and the Variance





High Bias - Low Variance

Low Bias - High Variance

The mathematics of the Bias - Variance decomposition is to be found in many sources, inclusive my, The MIT Press published, book (Kecman, 2001).

And now, back to NNs & SVMs

In the rest of presentation we tightly follow The MIT Press published book (Kecman, 2001), as well as our the most recent results.

Check the book's site **http://www.support-vector.ws** for the newest paper's and software's downloads.

Some connections between NN *i.e./or/and* SVM and classic techniques such as Fourier series and Polynomial approximations

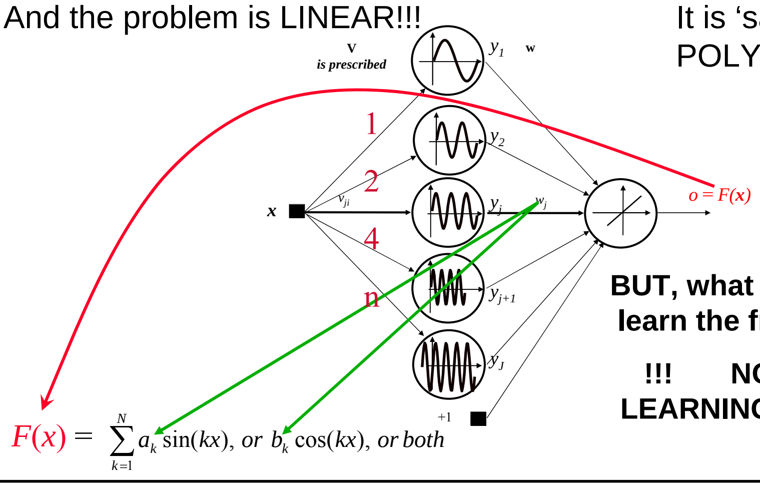
Classic approximation techniques in NN graphical appearance

FOURIER SERIES

AMPLITUDES and PHASES of sine (cosine) waves are unknown, but frequencies are known because <u>Mr Joseph Fourier has selected frequencies for us</u> -> they are INTEGER multiples of some pre-selected base frequency.

And the problem is LINEAR!!!

It is 'same' with POLYNOMIALS

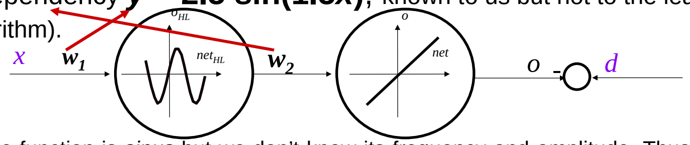


$$F(x) = \sum_{k=1}^{N} a_k \sin(kx),\ or\ b_k \cos(kx),\ or\ both$$

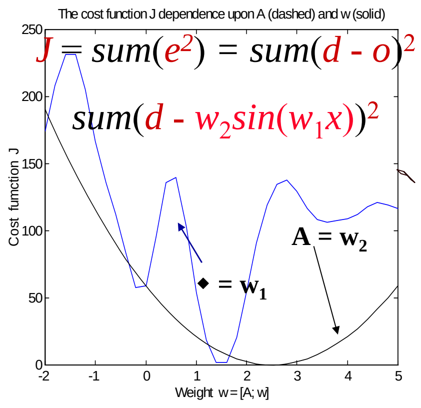
BUT, what if we want to learn the frequencies?

!!! NONLINEAR LEARNING PROBLEM !!!

Now, we want to find Fourier 'series' model $\boldsymbol{o = y = w_2 \sin(w_1 x)}$ of the underlying dependency $\boldsymbol{y = 2.5 \sin(1.5x)}$, known to us but not to the learning machine (algorithm).



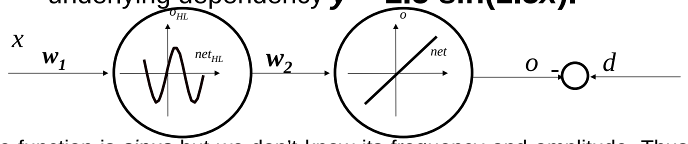
We know that the function is *sinus* but we don't know its frequency and amplitude. Thus, by using the training data set {**x**, *d*}, we want to model this system with the NN model consisting of a single neuron in HL (having *sinus* as an activation function) as given above.



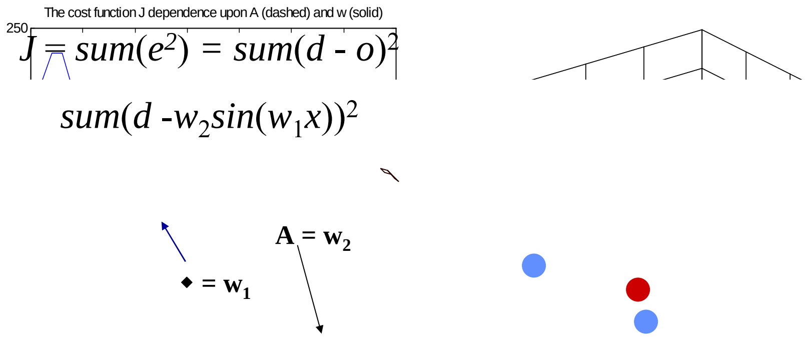
$$J = sum(e^2) = sum(d - o)^2$$

$$sum(d - w_2 sin(w_1 x))^2$$

Now, we want to find Fourier 'series' model $\boldsymbol{o = y = w_2 \sin(w_1 x)}$ of the underlying dependency $\boldsymbol{y = 2.5 \sin(1.5x)}$.



We know that the function is *sinus* but we don't know its frequency and amplitude. Thus, by using the training data set {**x, d**}, we want to model this system with the NN model consisting of a single neuron in HL (having *sinus* as an activation function) as given above.



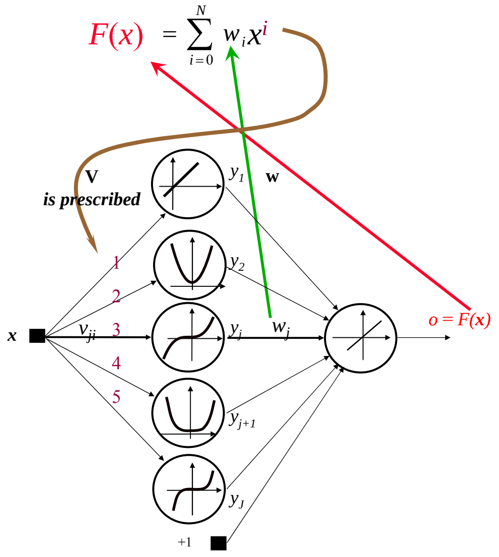
$$J = sum(e^2) = sum(d - o)^2$$

$$sum(d - w_2 sin(w_1 x))^2$$

Another classic approximation scheme is a

POLYNOMIAL SERIES

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

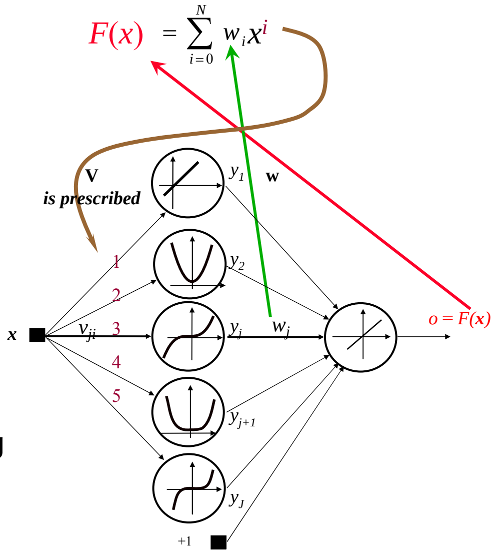


Another classic approximation scheme is a

POLYNOMIAL SERIES

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

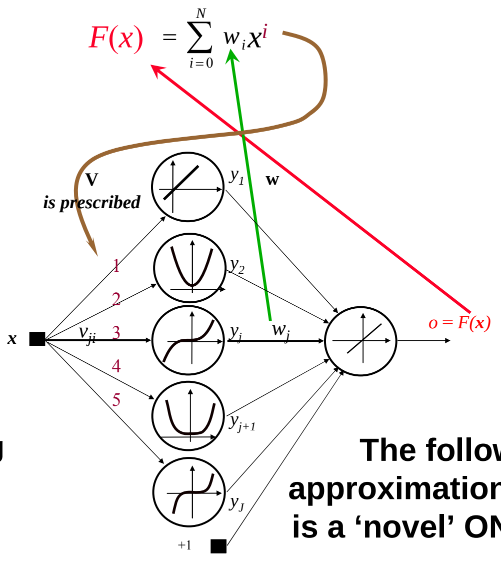
With a prescribed (integer) exponents this is again LINEAR APPROXIMATION SCHEME. Linear in terms of parameters to learn and not in terms of the resulting approximation function. This one is NL function for *i* > 1.



Another classic approximation scheme is a

POLYNOMIAL SERIES

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



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

The following approximation scheme is a ‘novel’ ONE called

RBF network here.

Approximation of some

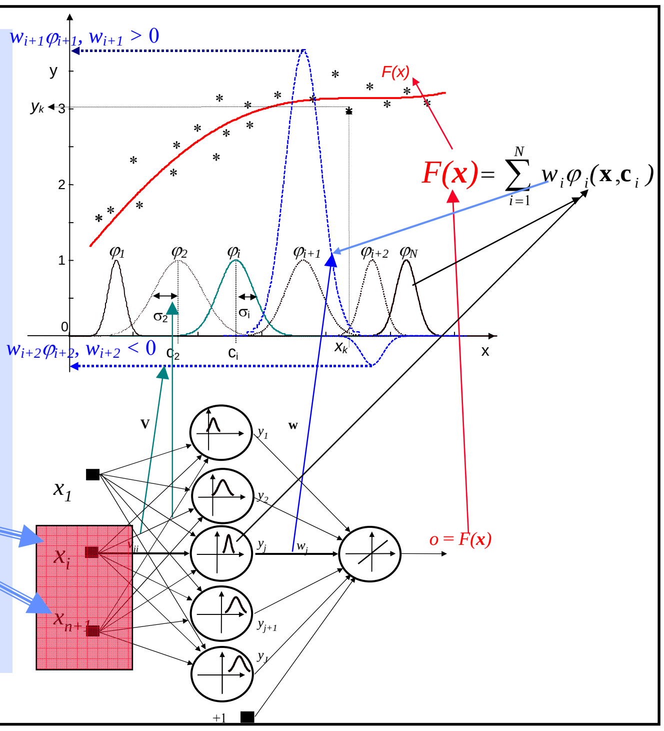
NL 1D function by

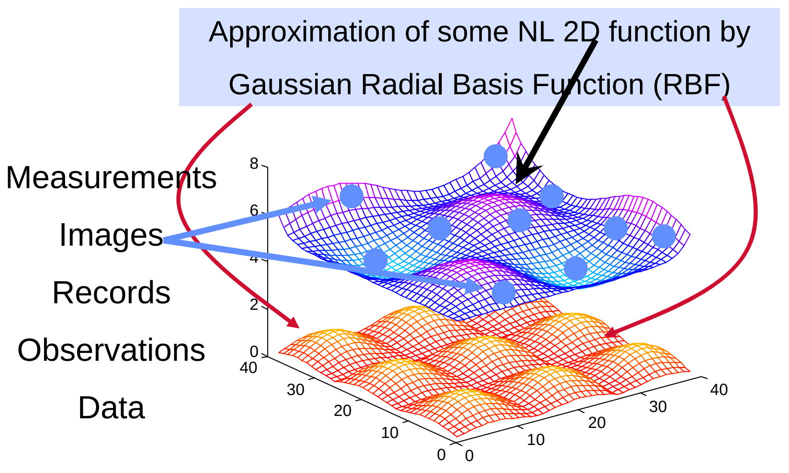
Gaussian

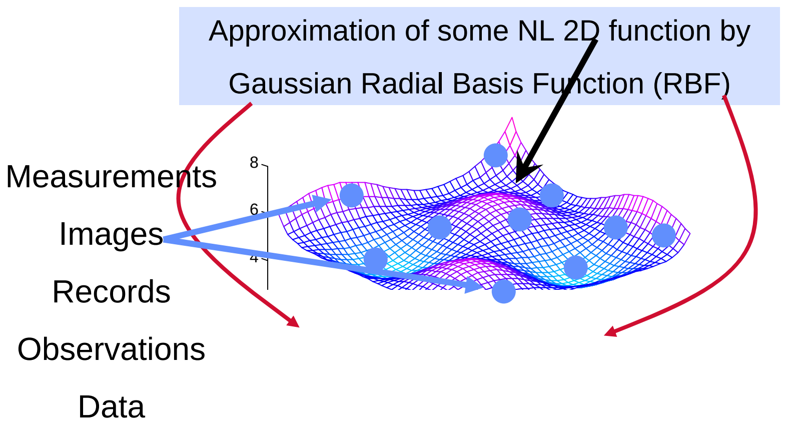
Radial Basis Function

(RBF)

In 1-D case forget these two inputs. They are here just to denote that the basic structure of the NN is the same for ANY-DIMENSIONAL INPUT




Approximation of some NL 2D function by
Gaussian Radial Basis Function (RBF)
Measurements
Images
Records
Observations
Data
8
6
4
2
0
40
30
20
10
0
0
10
20
30
40


Approximation of some NL 2D function by
Gaussian Radial Basis Function (RBF)
Measurements
Images
Records
Observations
Data
8
6
4

For FIXED Gaussian RBFs a LEARNING FROM DATA is a LINEAR PROBLEM. If the Centers and Covariance matrices are the subjects of learning, the problem becomes NONLINEAR.

Read it - **extremely difficult!**

The learning machine that uses data to find the APPROXIMATING FUNCTION (in regression problems) or the SEPARATION BOUNDARY (in classification, pattern recognition problems), is the same in high-dimensional situations.

Here, it will be either the so-called **SVM** or the **NN**

(however remember, there are other models too).

The learning machine that uses data to find the APPROXIMATING FUNCTION (in regression problems) or the SEPARATION BOUNDARY (in classification, pattern recognition problems), is the same in high-dimensional situations.

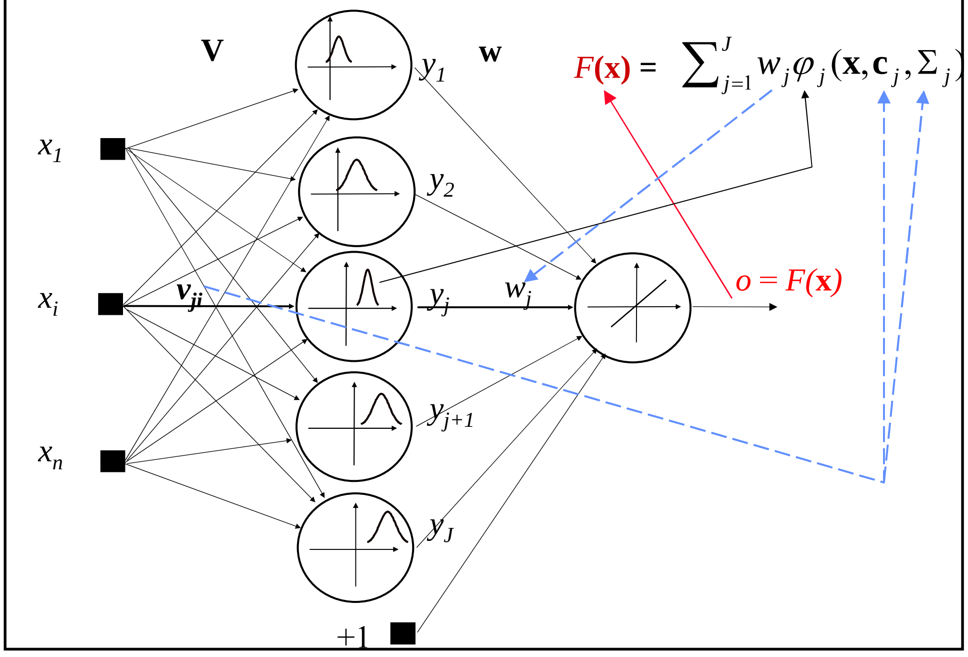
Here, it will be either the so-called **SVM** or the **NN**

(however remember, there are other models too).

WHAT are DIFFERENCES and SIMILARITIES?

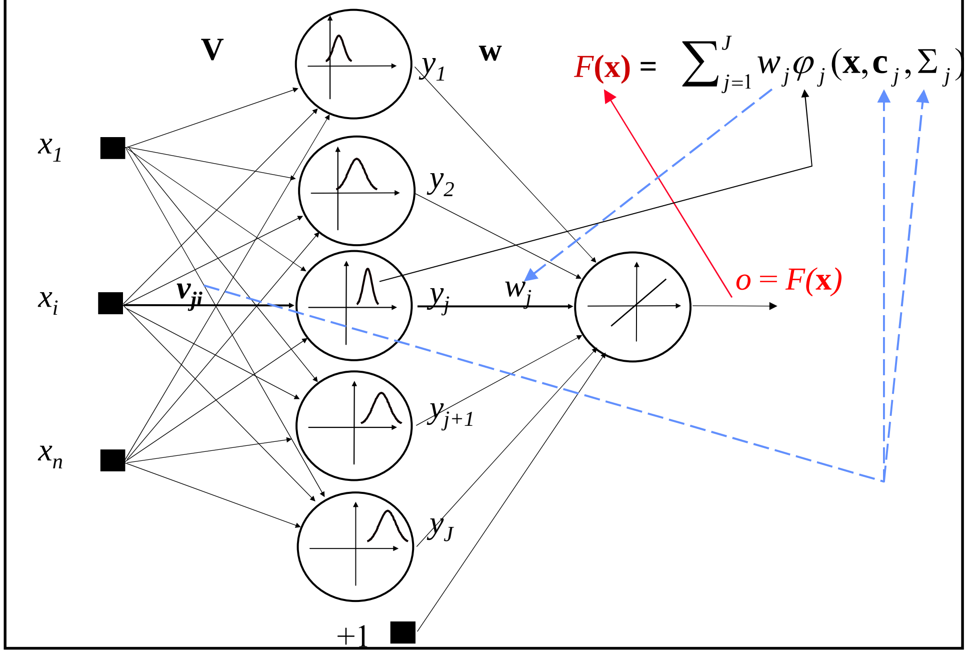
WHATCH CAREFULLY NOW !!!

This is a Neural Network,



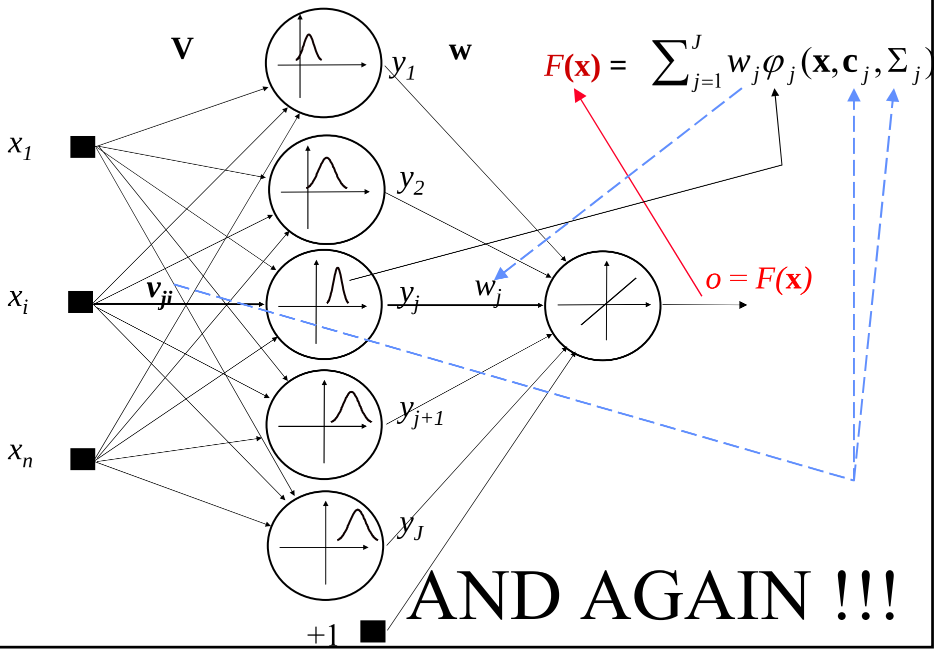
V
w
x_1
x_i
x_n
v_{ji}
y_1
y_2
y_j
y_{j+1}
y_J
w_j
+1
$F(\mathbf{x}) = \sum_{j=1}^{J} w_j \varphi_j(\mathbf{x}, \mathbf{c}_j, \Sigma_j)$
$o = F(\mathbf{x})$

and, this is a Support Vector Machine.



V
w
x_1
x_i
x_n
v_{ji}
y_1
y_2
y_j
y_{j+1}
y_J
w_j
+1
$F(\mathbf{x}) = \sum_{j=1}^{J} w_j \varphi_j(\mathbf{x}, \mathbf{c}_j, \Sigma_j)$
$o = F(\mathbf{x})$

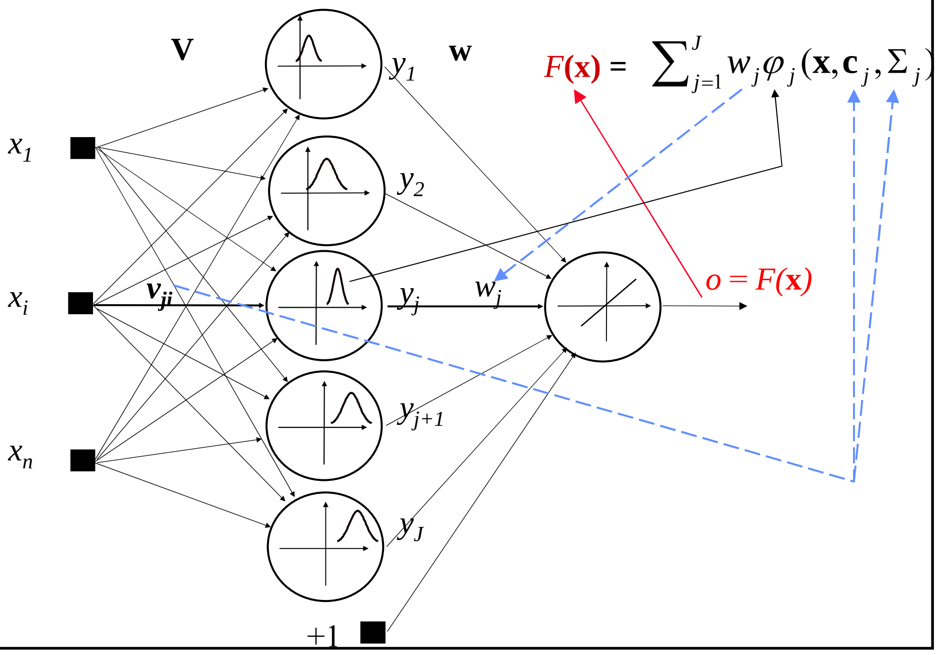
and, this is a Support Vector Machine.



V
W
x_1
x_i
x_n
v_{ji}
y_1
y_2
y_j
y_{j+1}
y_J
w_j
$F(\mathbf{x}) = \sum_{j=1}^{J} w_j \varphi_j(\mathbf{x}, \mathbf{c}_j, \Sigma_j)$
$o = F(\mathbf{x})$
+1

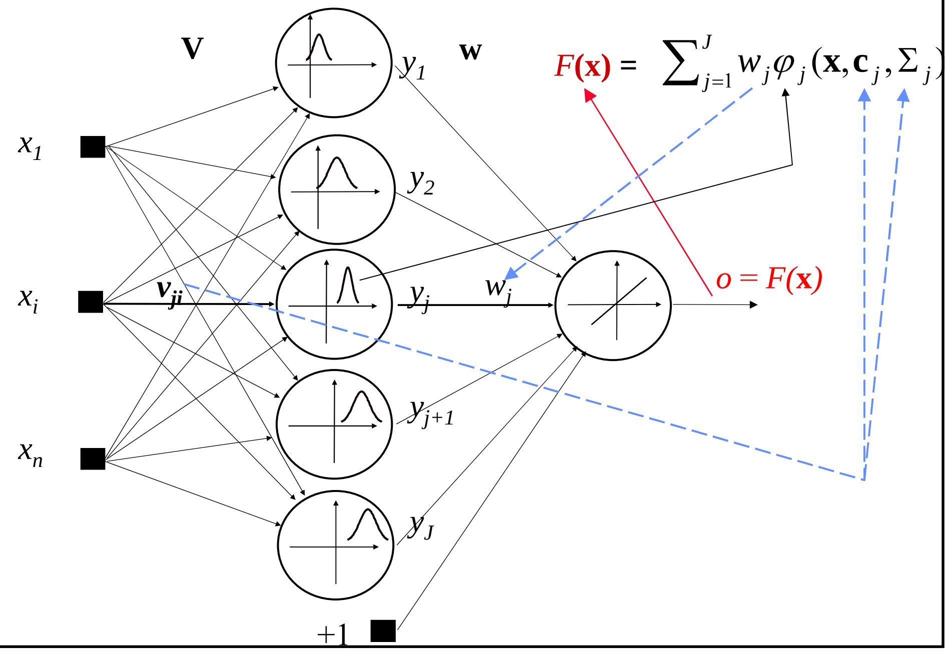
AND AGAIN !!!

This is a Neural Network,

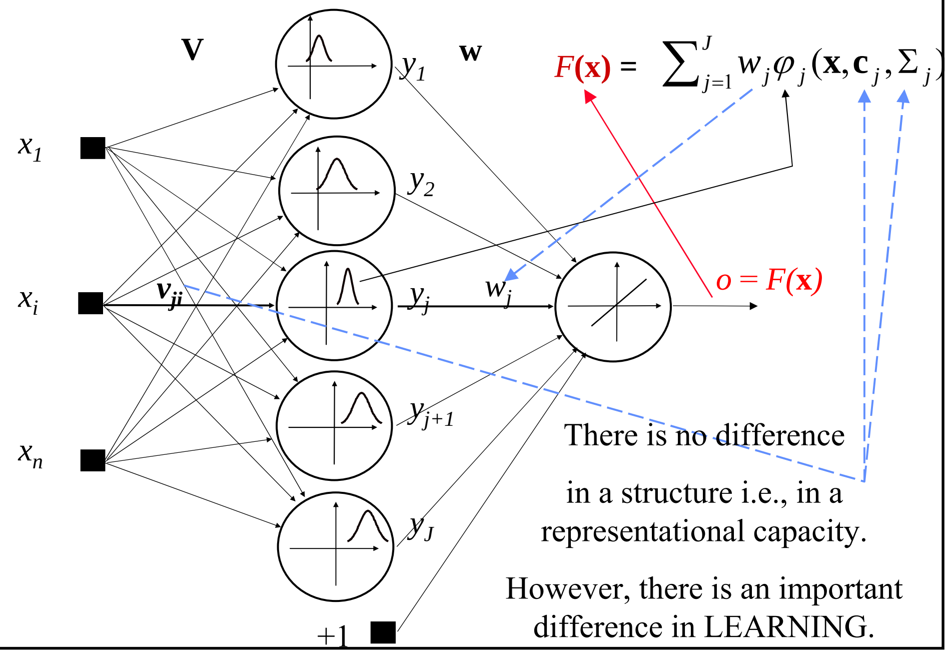


V
W
x_1
x_i
x_n
v_{ji}
y_1
y_2
y_j
y_{j+1}
y_J
w_j
$F(\mathbf{x}) = \sum_{j=1}^{J} w_j \varphi_j(\mathbf{x}, \mathbf{c}_j, \Sigma_j)$
$o = F(\mathbf{x})$
+1

and, this is a Support Vector Machine.



and, this is a Support Vector Machine.



There is no difference

in a structure i.e., in a representational capacity.

However, there is an important difference in LEARNING.

Therefore,

let’s say a little more about basics of

the learning from data first.

Note that you may find different names for the L from D:

identification, estimation, regression,classification, pattern recognition, function approximation, curve or surface fitting etc.

All these tasks used to be solved previously.

Thus, THERE IS THE QUESTION:

Is there anything new in respect to the classic statistical inference?

The classic regression and (Bayesian) classification statistical techniques are based on the very strict assumption that probability distribution models or probability-density functions are known.

Classic statistical inference is based on the following three fundamental assumptions:

The classic regression and (Bayesian) classification statistical techniques are based on the very strict assumption that probability distribution models or probability-density functions are known.

Classic statistical inference is based on the following three fundamental assumptions:

***Data can be modeled by a set of linear in parameter functions; this is a foundation of a parametric paradigm in learning from experimental data.**

The classic regression and (Bayesian) classification statistical techniques are based on the very strict assumption that probability distribution models or probability-density functions are known.

Classic statistical inference is based on the following three fundamental assumptions:

*Data can be modeled by a set of linear in parameter functions; this is a foundation of a parametric paradigm in learning from experimental data.
*In the most of real-life problems, a stochastic component of data is the normal probability distribution law, i.e., the underlying joint probability distribution is Gaussian.

The classic regression and (Bayesian) classification statistical techniques are based on the very strict assumption that probability distribution models or probability-density functions are known.

Classic statistical inference is based on the following three fundamental assumptions:

*Data can be modeled by a set of linear in parameter functions; this is a foundation of a parametric paradigm in learning from experimental data.
*In the most of real-life problems, a stochastic component of data is the normal probability distribution law, i.e., the underlying joint probability distribution is Gaussian.
*Due to the second assumption, the induction paradigm for parameter estimation is the maximum likelihood method that is reduced to the minimization of the sum-of-errors-squares cost function in most engineering applications.

All three assumptions on which the classic statistical paradigm relied, turned out to be inappropriate for many contemporary real-life problems (Vapnik, 1998) due to the facts that:

All three assumptions on which the classic statistical paradigm relied, turned out to be inappropriate for many contemporary real-life problems (Vapnik, 1998) due to the facts that:

***modern problems are high-dimensional, and if the underlying mapping is not very smooth the linear paradigm needs an exponentially increasing number of terms with an increasing dimensionality of the input space *X*, i.e., with an increase in the number of independent variables. This is known as ‘the curse of dimensionality’,**

All three assumptions on which the classic statistical paradigm relied, turned out to be inappropriate for many contemporary real-life problems (Vapnik, 1998) due to the facts that:

*modern problems are high-dimensional, and if the underlying mapping is not very smooth the linear paradigm needs an exponentially increasing number of terms with an increasing dimensionality of the input space *X*, i.e., with an increase in the number of independent variables. This is known as ‘the curse of dimensionality’,

*the underlying real-life data generation laws may typically be very far from the normal distribution and a model-builder must consider this difference in order to construct an effective learning algorithm,

All three assumptions on which the classic statistical paradigm relied, turned out to be inappropriate for many contemporary real-life problems (Vapnik, 1998) due to the facts that:

*modern problems are high-dimensional, and if the underlying mapping is not very smooth the linear paradigm needs an exponentially increasing number of terms with an increasing dimensionality of the input space *X*, i.e., with an increase in the number of independent variables. This is known as ‘the curse of dimensionality’,

*the underlying real-life data generation laws may typically be very far from the normal distribution and a model-builder must consider this difference in order to construct an effective learning algorithm,

*from the first two objections it follows that the maximum likelihood estimator (and consequently the sum-of-error-squares cost function) should be replaced by a new induction paradigm that is uniformly better, in order to model non-Gaussian distributions.

There is a real life fact

the probability-density functions are TOTALLY unknown,

and there is the question

HOW TO PERFORM a distribution-free

REGRESSION or *CLASSIFICATION* ?

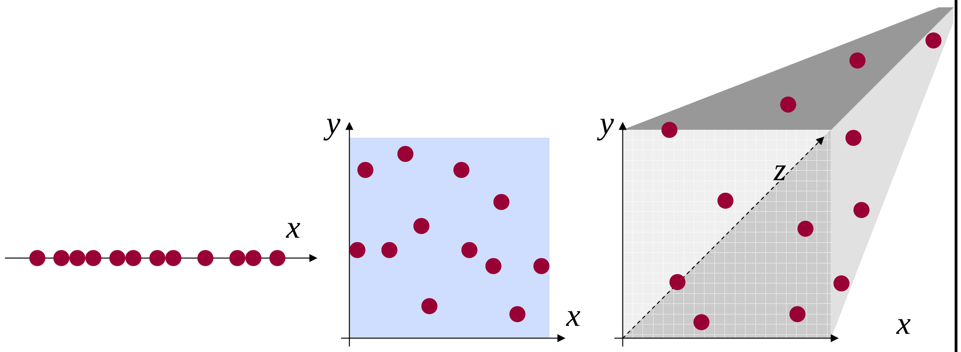
Mostly, all we have are recorded **EXPERIMENTAL DATA** (training patterns, samples, observations, records, examples):
Data is high-dimensional and **scarce (always too little data)!!!**

High-dimensional spaces seem to be **terrifyingly empty** and our learning algorithms (i.e., machines) should be able to operate in such spaces and to **learn from such a sparse data**.
There is an old saying that ***redundancy provides knowledge***.
Stated simpler
the more data pairs we have the better results will be.

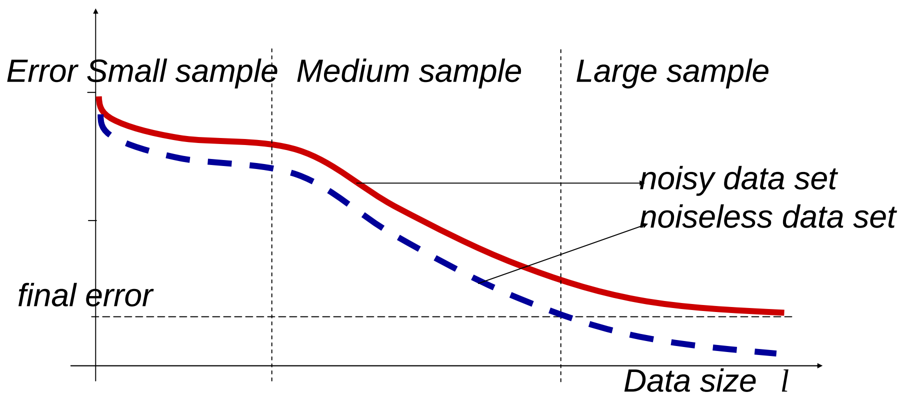
Terrifying emptiness and/or data sparseness

Just a first simple example

Imagine sampling some 1D $y = f(x)$, 2D $z = f(x, y)$, and 3D $u = f(x, y, z)$, functions and taking 10 samples on the domain (0, 1)!



Data points in 1D, 2D and 3D domains are less and less dense, and the average distance between the points increases with the dimensionality!!!



Dependency of the modeling error on the training data set size

Thus, the main characteristics of all MODERN problems is the mapping between the high-dimensional spaces.

Let’s exemplify this by the following extremely simple pattern recognition (classification) example!

Gender recognition problem: Are these two faces female or male?

F or
M?

M or
F?

Gender recognition problem: Are these two faces female or male?

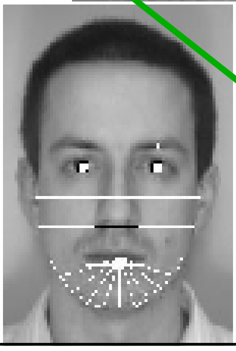
F or
M?

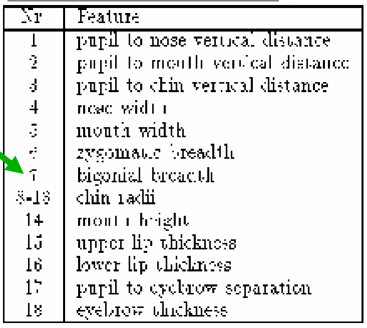
There must be something in the geometry of our faces. Here, 18 input variables, features, were chosen!

M or
F?

Problem from

Brunelli & Poggio, 1993.

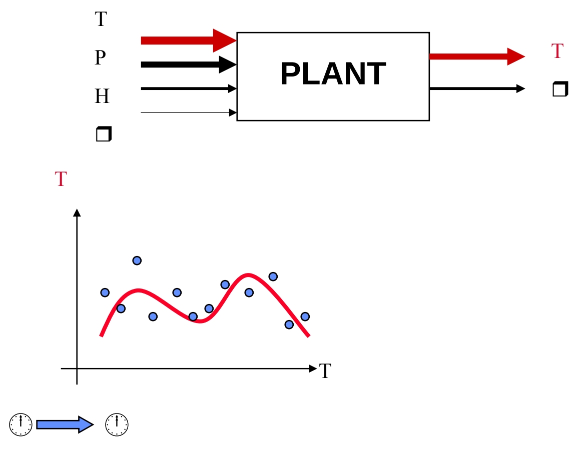




Nr	Feature
1	pupil to nose vertical distance
2	pupil to mouth vertical distance
3	pupil to chin vertical distance
4	nose width
5	mouth width
6	zygomatic breadth
7	bigonial breadth
8-13	chin radii
14	mouth height
15	upper lip thickness
16	lower lip thickness
17	pupil to eyebrow separation
18	eyebrow thickness

Approximation and classification are ‘same’ for any dimensionality of the input space. **Nothing (!) but size changes.** But the **change is DRASTIC.** High dimensionality means both an **EXPLOSION in a number OF PARAMETERS** to learn and a **SPARSE training data set.**

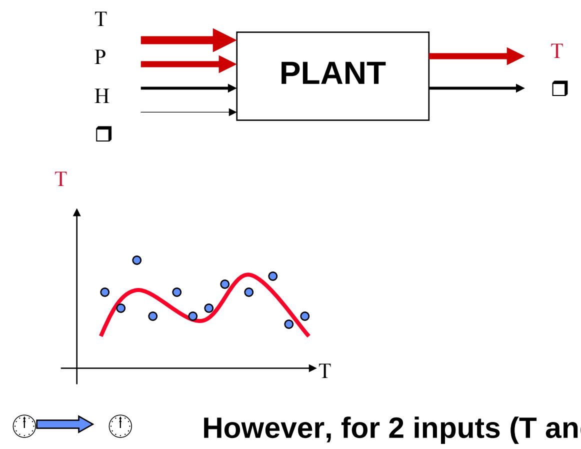
High dimensional spaces seem to be terrifyingly empty.



N data

Approximation and classification are ‘same’ for any dimensionality of the input space. **Nothing (!) but size changes.** But the **change is DRASTIC.** High dimensionality means both an **EXPLOSION in a number OF PARAMETERS** to learn and a **SPARSE training data set.**

High dimensional spaces seem to be terrifyingly empty.

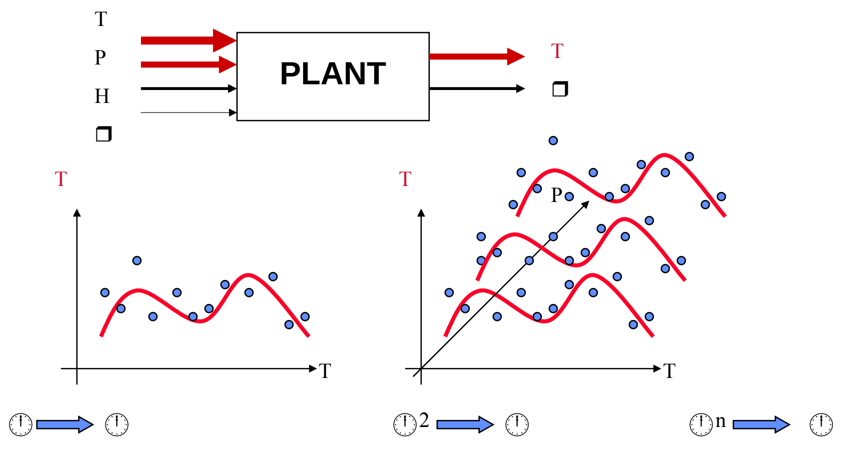


However, for 2 inputs (T and P)

N data **do we need 2N or N^2 data?**

Approximation and classification are ‘same’ for any dimensionality of the input space. **Nothing (!) but size changes.** But the **change is DRASTIC.** High dimensionality means both an **EXPLOSION in a number OF PARAMETERS** to learn and a **SPARSE training data set.**

High dimensional spaces seem to be terrifyingly empty.



N data **N^2 data** **N^n data**

CURSE of DIMENSIONALITY and SPARSITY OF DATA.

The newest promising tool FOR WORKING UNDER THESE CONSTRAINTS are the SUPPORT VECTOR MACHINES based on the STATISTICAL LEARNING THEORY (VLADIMIR VAPNIK and ALEKSEI CHERVONENKIS).

WHAT IS THE contemporary BASIC LEARNING PROBLEM???

LEARN THE DEPENDENCY (FUNCTION, MAPPING) from SPARSE DATA, under NOISE, in HIGH DIMENSIONAL SPACE!

Recall - the redundancy provides the knowledge!

A lot of data - ‘easy’ problem.

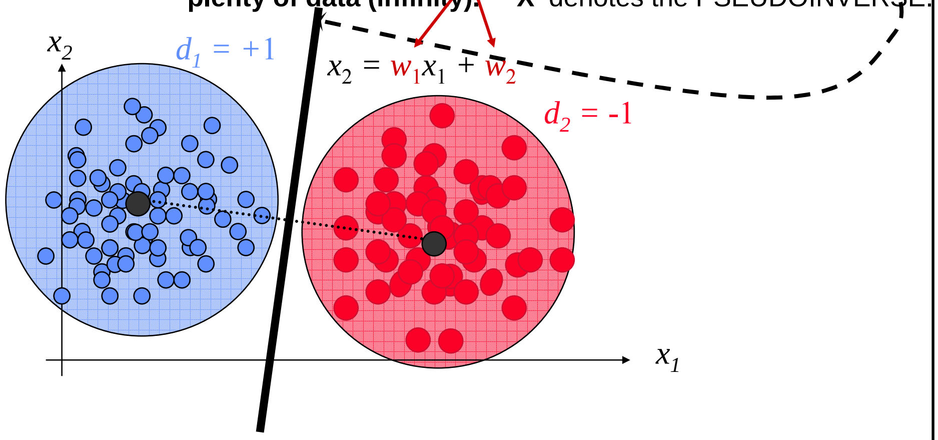
LET’S EXEMPLIFY

THE INFLUENCE OF A DATA SET SIZE ON THE SIMPLEST RECOGNITION PROBLEM

BINARY CLASSIFICATION, i.e., DICHOTOMIZATION.

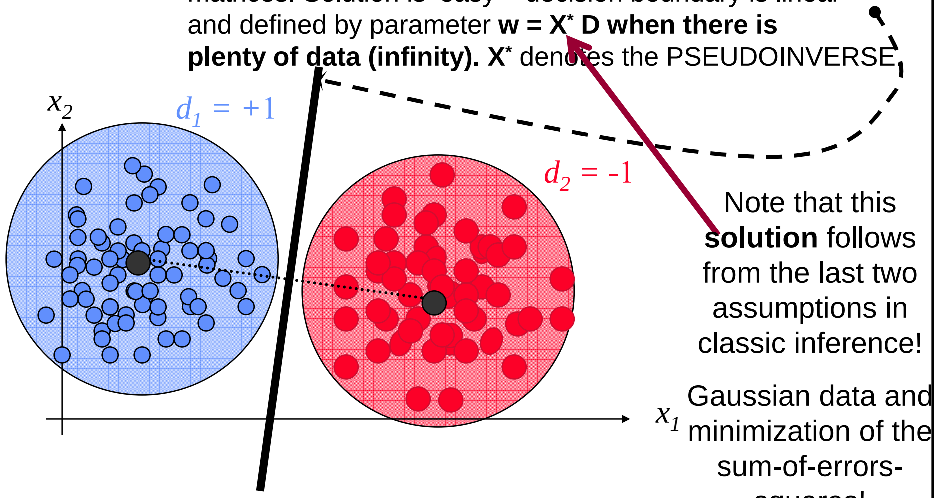
CLASSIFICATION or PATTERN RECOGNITITON EXAMPLE

Assume - Normally distributed classes, same covariance matrices. Solution is ‘easy’ - decision boundary is linear and defined by parameter **w = X^* D when there is plenty of data (infinity).** X^* denotes the PSEUDOINVERSE.



CLASSIFICATION or PATTERN RECOGNITITON EXAMPLE

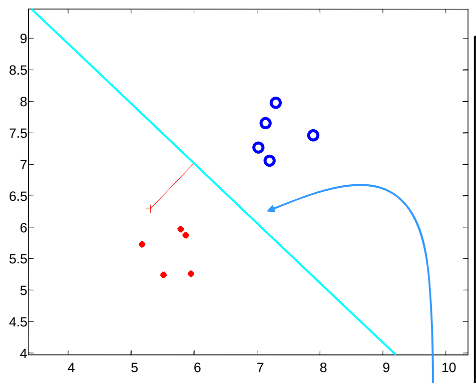
Assume - Normally distributed classes, same covariance matrices. Solution is ‘easy’ - decision boundary is linear and defined by parameter **w = X^* D when there is plenty of data (infinity).** X^* denotes the PSEUDOINVERSE.



Note that this **solution** follows from the last two assumptions in classic inference!

Gaussian data and minimization of the sum-of-errors-squares!

A simple example how this algorithm works:



$$
\begin{array}{cc}
\mathbf{X} & \mathbf{D} \\
\begin{bmatrix}
5.7948 & 5.9797 & 1.0000 \\
5.9568 & 5.2714 & 1.0000 \\
5.5226 & 5.2523 & 1.0000 \\
5.8801 & 5.8757 & 1.0000 \\
5.1730 & 5.7373 & 1.0000 \\
7.1365 & 7.6614 & 1.0000 \\
7.0118 & 7.2844 & 1.0000 \\
7.8939 & 7.4692 & 1.0000 \\
7.1991 & 7.0648 & 1.0000 \\
7.2987 & 7.9883 & 1.0000
\end{bmatrix}
&
\begin{bmatrix}
1 \\ 1 \\ 1 \\ 1 \\ 1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1
\end{bmatrix}
\end{array}
$$

$\mathbf{w} = \mathbf{X}^* \mathbf{D} \longrightarrow \mathbf{w}_{opt} = [-0.5209 \quad -0.5480 \quad 6.9731]^T$, and

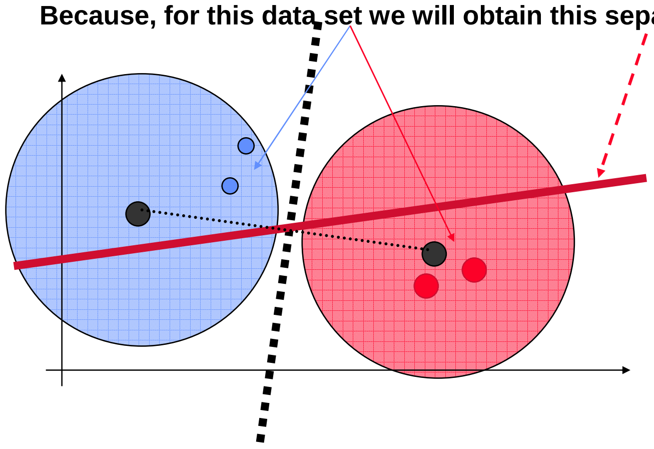
the separation boundary equals

$$x_2 = -0.951 x_1 + 12.725$$

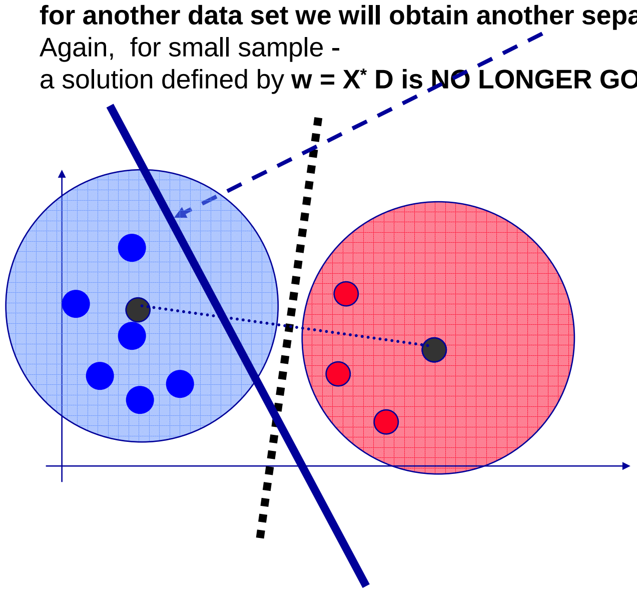
However, for a small sample -

Solution defined by **w = X* D is NO LONGER GOOD ONE !!!**

Because, for this data set we will obtain this separation line,



and,
for another data set we will obtain another separation line.
Again, for small sample -
a solution defined by **w = X* D is NO LONGER GOOD ONE !!!**

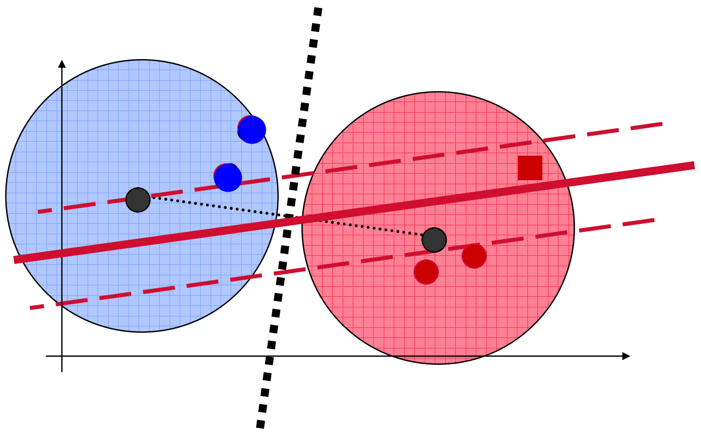


What is common for both separation lines the red and the blue one.

Both have a SMALL MARGIN.

WHAT'S WRONG WITH SMALL MARGIN? Look at the RED line!

It is very likely that the new examples (🟥 , 🟦) will be wrongly classified.

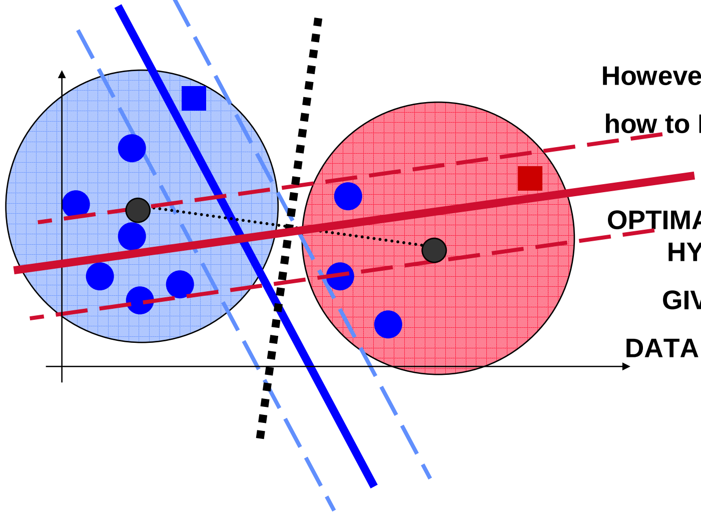


What is common for both separation lines the red and the blue one.

Both have a SMALL MARGIN.

WHAT'S WRONG WITH SMALL MARGIN? Look at the BLUE line!

It is very likely that the new examples (🟥 , 🟦) will be wrongly classified.

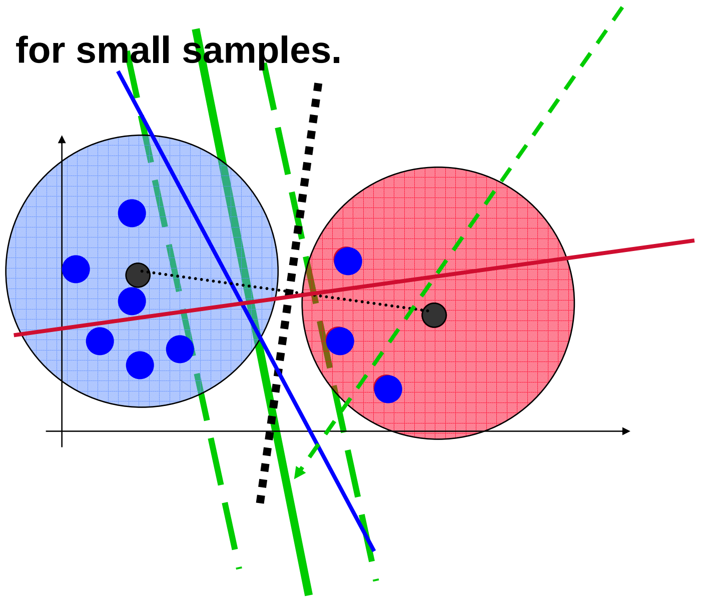


However, the question is how to DEFINE and FIND the OPTIMAL SEPARATION HYPERPLANE GIVEN (scarce) DATA SAMPLES ???

The STATISTICAL LEARNING THEORY IS DEVELOPED TO SOLVE

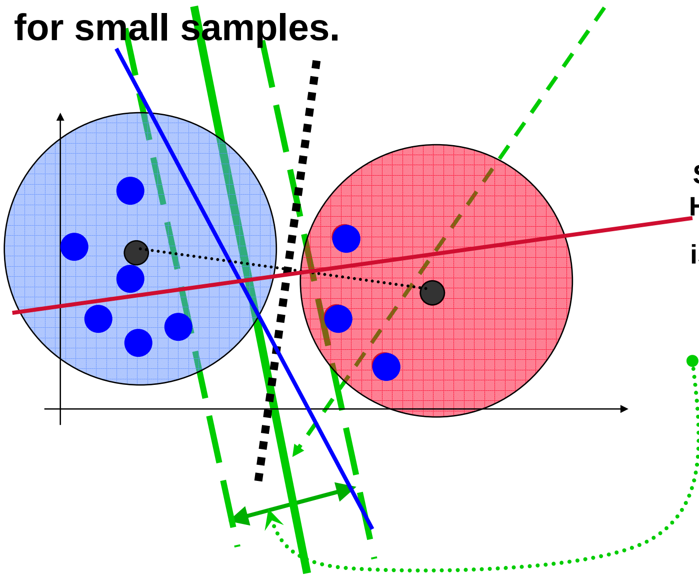
PROBLEMS of FINDING THE OPTIMAL SEPARATION HYPERPLANE

for small samples.



The STATISTICAL LEARNING THEORY IS DEVELOPED TO SOLVE PROBLEMS of FINDING THE OPTIMAL SEPARATION HYPERPLANE

for small samples.



OPTIMAL SEPARATION HYPERPLANE

is the one that has the

LARGEST MARGIN

on given

DATA SET

One more intuitive presentation why the maximal margin idea may be a good statistical approach follows on the next slide!

Note, however, that the intuition only, does not qualify for, and does not guarantee, a broad acceptance of a maximal margin approach in a statistical learning.

There are both the strong theoretical proofs about the errors, bounds and generalization properties of SVMs based on a maximal margin idea, and convincing experimental performances on various benchmark data sets..

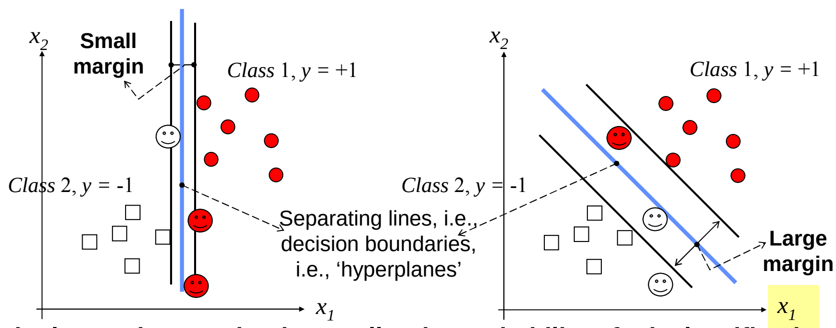
SUPPORT VECTOR MACHINE

is a MAXIMAL MARGIN CLASSIFIER

• it aims at finding the separating hyperplane with the maximal geometric margin (and not any one, that is the perceptron solution)

• WHY maximal margin?

Suppose we wan to separate two linearly separable classes, and we did it by two different decision functions.



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

Before presenting the math of SVMs, just a few more 'similarities' between NNs and SVMs follow

$$E = \underbrace{\sum_{i=1}^{P}(d_i - f(\mathbf{x}_i, \mathbf{w}))^2}_{\text{Closeness to data}}$$ **A classic multilayer perceptron**

$$E = \underbrace{\sum_{i=1}^{P}(d_i - f(\mathbf{x}_i, \mathbf{w}))^2}_{\text{Closeness to data}} + \underbrace{\lambda \|\mathbf{P}f\|^2}_{\text{Smoothness}}$$ **Regularization (RBF) NN**

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

There are two basic, constructive approaches to the minimization of the right hand side of previous equations (Vapnik, 1995 and 1998):

- choose an appropriate structure (order of polynomials, number of HL neurons, number of rules in the FL model) and, keeping the confidence interval fixed in this way, minimize the training error (i.e., empirical risk), or

- keep the value of the training error fixed (equal to zero or equal to some acceptable level) and minimize the confidence interval.

classic NNs implement the first approach (or some of its sophisticated variants) and SVMs implement the second strategy. In both cases the resulting model should resolve the trade-off between under-fitting and over-fitting the training data. The final model structure (order) should ideally *match the* ***learning machines capacity*** *with* ***training data complexity***.

O.K.

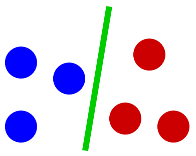
Let us do some more formal,

meaning,

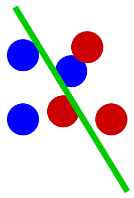
mathematical analysis of SVMs learning!

The presentation will follow an idea of a **gentle introduction, i.e., of a gradual proceeding** from the 'simple' cases to the more complex and involved ones!

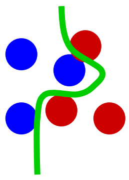
1) Linear Maximal Margin Classifier for Linearly Separable Data - no samples overlapping.

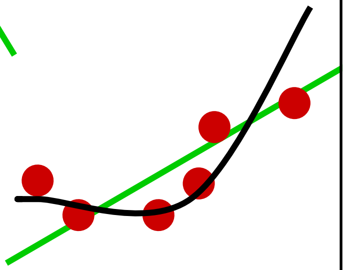


***2) Linear Soft Margin Classifier for Overlapping Classes*.**



3) Nonlinear Classifier.





4) Regression by SV Machines that can be either linear or nonlinear!

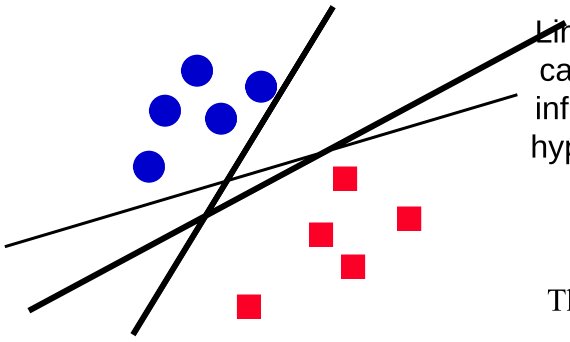
1) Linear Maximal Margin Classifier for Linearly Separable Data

Binary classification - no samples overlapping

Given some training data

$$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l), \quad y_i \in \{-1, +1\}$$

find the function $f(\mathbf{x}, \mathbf{w}_0) \in f(\mathbf{x}, \mathbf{w})$ which best approximates the unknown discriminant (separation) function $y = f(\mathbf{x})$.



Linearly separable data can be separated by in infinite number of linear hyperplanes that can be written as

$$f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T\mathbf{x} + b$$

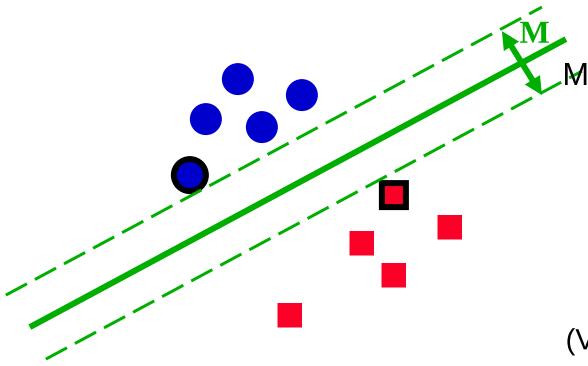
The problem is: find the optimal separating hyperplane

1) Vapnik-Chervonenkis: Optimal separating hyperplane is the one with

MAXIMAL MARGIN **!**

This hyperplane is uniquely determined by the vectors on the margin

the support vectors!



MARGIN IS DEFINED by **w** as follows:

$$M = \frac{2}{\|\mathbf{w}\|}$$

(Vapnik, Chervonenkis ‘74)

The optimal canonical separating hyperplane (OCSH), i.e., a separating hyperplane with the largest margin (defined by $M = 2 / ||\mathbf{w}||$), specifies *support vectors,* i.e., training data points closest to it, which satisfy $y_j[\mathbf{w}^T\mathbf{x}_j + b] \equiv 1$, $j = 1, N_{SV}$. At the same time, the OCSH must separate data correctly, i.e., it should satisfy inequalities

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

where l denotes a number of training data and N_{SV} stands for a number of SV.

Note that maximization of M means a minimization of $||\mathbf{w}||$.
Minimization of a norm of a hyperplane normal weight vector $||\mathbf{w}|| = \sqrt{\mathbf{w}^T\mathbf{w}} = \sqrt{w_1^2 + w_2^2 + ... + w_n^2}$ leads to a maximization of a margin M. Because *sqrt*(*f*) is a monotonic function, its minimization is equivalent to a minimization of *f* .

Consequently, a minimization of norm $||\mathbf{w}||$ equals a minimization of

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

and this leads to a maximization of a margin M.

Thus the problem to solve is:

minimize

$$J = \mathbf{w}^{\mathrm{T}} \mathbf{w} = || \mathbf{w} ||^2$$

subject to constraints

$$y_i[\mathbf{w}^{\mathrm{T}} \mathbf{x}_i + b] \geq 1$$

and this is a classic QP problem with constraints that ends in forming and solving of a primal and/or dual Lagrangian.

Thus the problem to solve is:

minimize

Margin maximization!

$$J = \mathbf{w}^{\mathrm{T}} \mathbf{w} = || \mathbf{w} ||^2$$

subject to constraints

Correct classification!

$$y_i[\mathbf{w}^{\mathrm{T}} \mathbf{x}_i + b] \geq 1$$

and this is a classic QP problem with constraints that ends in forming and solving of a primal and/or dual Lagrangian.

Now, from the one sphere of mathematics (say, an intuitive geometric one) we should jump into the another sphere,
into the sphere of a nonlinear optimization (say, into an algebraic sphere).

Basics of the General Optimization Problem

Optimize	$f(\mathrm{w})$
Subject To (s.t.)	$g(\mathrm{w}) = 0$
	$\mathbf{w} > \mathbf{0}$, or $\mathbf{w} >= \mathbf{0}$

LINEAR PROGRAMMING problem: when f**(w)** and g**(w)** are **linear** and w_i's > 0

INTEGER PROGRAMMING problem: when w_i**'s should take only integer values.**

QUADRATIC PROGRAMMING problem f**(w) quadratic,** g**(w)** is **linear,**

NONLINEAR PROGRAMMING problem**,** f**(w) and** g**(w)** are **general nonlinear functions!**

How ones solve such QP problems with constraints:

Step 1) Forming a Primal Lagrangian in terms of primal (original) variables *w*-*s*, *b and* α-s (by an augmenting of the cost function by the constraints multiplied by dual variables α-s).

Step 2) Using the Karush-Kuhn-Tucker (KKT) conditions and forming a Dual Lagrangian in terms of α-s only.

Step 3) Solving a Dual Lagrangian for α-s.

Step 4) Using the KKT conditions for calculation of primal variables *w*-s and *b*.

Step 5) Creating the decision function for a classification problem, or the regression one for the function approximation task.

Step 6) Applying the SVM's model obtained.

A QP problem $J = \mathbf{w}^T \mathbf{w} = \|\mathbf{w}\|^2$, subject to constraints $y_i[\mathbf{w}^T \mathbf{x}_i + b] \geq 1$ is solved by **the *saddle point*** of the Lagrange functional (Lagrangian).

(In forming the Lagrangian for constraints of the form $g_i > 0$, the inequality constraints equations are multiplied by nonnegative Lagrange multipliers α_i (i.e., $\alpha_i > 0$) and subtracted from the objective function).

Step 1) Thus, a *primal variables Lagrangian* $L(\mathbf{w}, b, \boldsymbol{\alpha})$ is,

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{l} \alpha_i \{y_i[\mathbf{w}^T\mathbf{x}_i + b] - 1\}$$

where the α_i are Lagrange multipliers. The search for an *optimal* **saddle point** $(\mathbf{w}_o, b_o, \alpha_0)$ is necessary because Lagrangian L must be *minimized* with respect to **w** and *b*, and has to be *maximized* with respect to nonnegative α_i (i.e., maximal $\alpha_i \geq 0$ should be found). This problem can be solved either in **a *primal space*** (which is the space of parameters **w** and *b*) or in **a *dual space*** (which is the space of Lagrange multipliers α_i).

The second approach gives insightful results and we will consider this solution in a dual space below. In order to do that, we use the Karush-Kuhn-Tucker (KKT) conditions for the optimum of a constrained function.

Step 2) Karush-Kuhn-Tucker (KKT) conditions are:

- at the saddle point ($\mathbf{w}_o$, b_o, α_o), derivatives of Lagrangian L with respect to primal variables should vanish which leads to,

$$\frac{\partial L}{\partial \mathbf{w}_o} = 0, \qquad \text{i.e.,} \qquad \mathbf{w}_o = \sum_{i=1}^{l} \alpha_i y_i \mathbf{x}_i \qquad \text{(a)}$$

$$\frac{\partial L}{\partial b_o} = 0, \qquad \text{i.e.,} \qquad \sum_{i=1}^{l} \alpha_i y_i = 0 \qquad \text{(b)}$$

- and, in addition, the complementarity conditions

$$\alpha_i\{y_i[\mathbf{w}^T\mathbf{x}_i + b] - 1\} = 0, \quad i = 1, l.$$

must be satisfied.

Substituting (a) and (b) in a *primal variables Lagrangian* $L(\mathbf{w}, b, \alpha)$ (on previous page), we change to the *dual variables Lagrangian* $L_d(\alpha)$

Step 2-3) $$L_d(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j \mathbf{x}_i^T \mathbf{x}_j$$

Step 3) Such a <u>*standard quadratic optimization problem*</u> can be expressed in a ***matrix notation*** and formulated as follows:

Maximize

$$L_d(\alpha) = -0.5\boldsymbol{\alpha}^T\mathbf{H}\,\boldsymbol{\alpha} + \mathbf{1}^T\boldsymbol{\alpha},$$

subject to

$$\mathbf{y}^T\boldsymbol{\alpha} = 0,$$
$$\boldsymbol{\alpha} \geq \mathbf{0},$$

where $(\boldsymbol{\alpha})_i = \alpha_i$, $\mathbf{H}$ denotes the Hessian matrix ($H_{ij} = y_i y_j(\mathbf{x}_i\mathbf{x}_j) = y_i y_j \mathbf{x}^T_i\mathbf{x}_j$) of this problem and $\mathbf{1}$ is a unit vector $\mathbf{1} = [1\ 1\ .\ .\ .\ 1]^T$.

Some standard optimization programs typically ***minimize*** given objective function. Obviously, we can apply such programs and the same solution would be obtained if we

minimize

$$L_d(\alpha) = 0.5\boldsymbol{\alpha}^T\mathbf{H}\boldsymbol{\alpha} - \mathbf{1}^T\boldsymbol{\alpha},$$

subject to the same constraints namely

$$\mathbf{y}^T\boldsymbol{\alpha} = 0, \qquad \boldsymbol{\alpha} \geq \mathbf{0}.$$

Step 4) Solutions α_{oi} of the dual optimization problem above determine the parameters of the optimal hyperplane $\mathbf{w}_o$ (according to (a)) and b_o (according to the complementarity conditions) as follows,

$$\mathbf{w}_o = \sum_{i=1}^{N_{SV}} \alpha_{oi} y_i \mathbf{x}_i, \qquad i = 1, N_{SV}$$

$$b_o = \frac{1}{N_{SV}} \left(\sum_{s=1}^{N_{SV}} \left(\frac{1}{y_s} - \mathbf{x}_s^T \mathbf{w}_o \right) \right), \qquad s = 1, N_{SV}.$$

For b, and for the soft margin SVMs (see in three slides) we use only unbounded SVecs for which

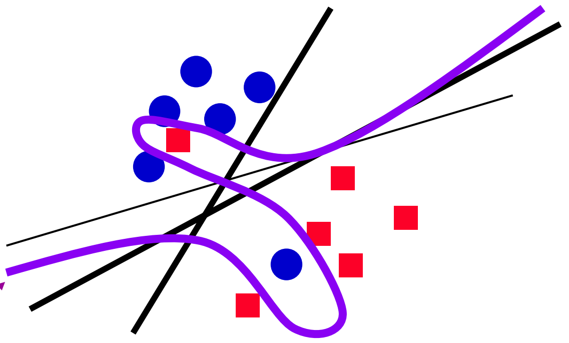
$$0 < \alpha_i < C$$

N_{SV} denotes the number of support vectors. Note that an optimal weight vector $\mathbf{w}_o$, the same as the bias term b_0, is calculated by **using support vectors only**. This is because Lagrange multipliers for all non-support vectors equal zero ($\alpha_{oi} = 0$, $i = N_{SV} + 1, l$). Finally, having calculated $\mathbf{w}_o$ and b_o we obtain a decision hyperplane $d(\mathbf{x})$ and an indicator function $i_F = o = \text{sign}(d(\mathbf{x}))$ as given below

Step 5-6)

$$d(\mathbf{x}) = \sum_{i=1}^{l} w_{oi} x_i + b_o = \sum_{i=1}^{l} y_i \alpha_i \mathbf{x}^T \mathbf{x}_i + b_o, \qquad i_F = o = \text{sign}(d(\mathbf{x})).$$

However, the previous algorithm **will not work for linearly NOT separable classes** i.e., in the case when there is data overlapping as shown below



There is no single hyperplane that can perfectly separate all data!

But, separation can now be done in two ways:

- 1) allow some misclassified data
- 2) try to find NONLINEAR separation boundary

2) Linear Soft Margin Classifier for Overlapping Classes

(allowing misclassification)

Possible idea!

Minimize $\frac{1}{2}\mathbf{w}^T\mathbf{w} + C(\# \text{ of training errors})$

where C is a penalty parameter, trading off the margin size for the number of misclassified data points. Large C leads to small number of misclassification and bigger margin and vice versa.

HOWEVER!!! There is a serious problem! **Counting errors can't be accommodated within the NICE (meaning reliable, well understood and well developed) quadratic programming approach.**
Also, **it doesn't distinguish between disastrous errors and near misses)!**

SOLUTION! Minimize

$$\frac{1}{2}\mathbf{w}^T\mathbf{w} + C(\text{distance of error points to their correct side})$$

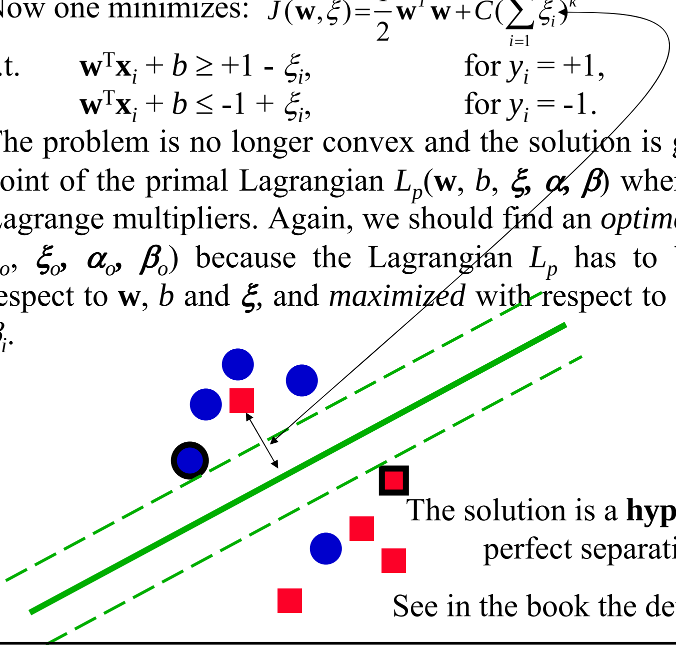
2) Linear Soft Margin Classifier for Overlapping Classes

Now one minimizes: $J(\mathbf{w},\xi) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C(\sum_{i=1}^{l}\xi_i)^k$

s.t. $\mathbf{w}^T\mathbf{x}_i + b \geq +1 - \xi_i$, for $y_i = +1$,

$\mathbf{w}^T\mathbf{x}_i + b \leq -1 + \xi_i$, for $y_i = -1$.

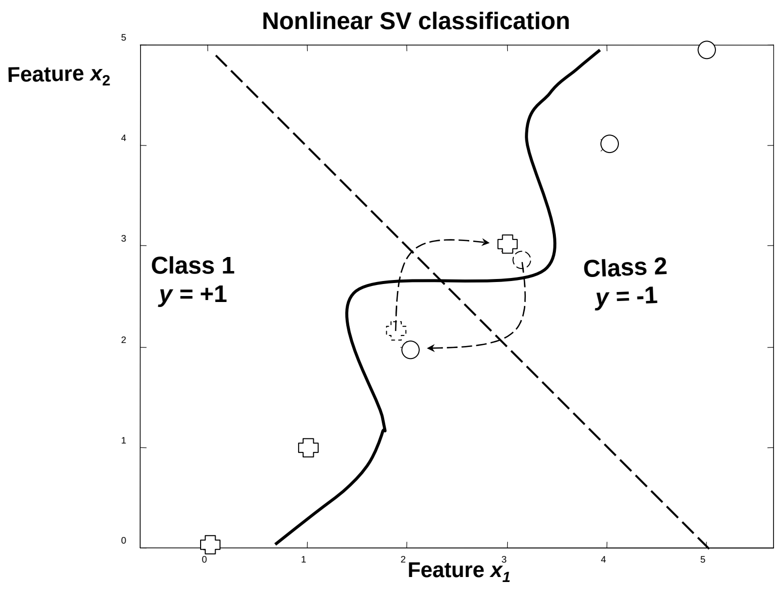
The problem is no longer convex and the solution is given by the saddle point of the primal Lagrangian $L_p(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta})$ where α_i and β_i are the Lagrange multipliers. Again, we should find an *optimal* saddle point ($\mathbf{w}_o$, b_o, $\boldsymbol{\xi}_o$, $\boldsymbol{\alpha}_o$, $\boldsymbol{\beta}_o$) because the Lagrangian L_p has to be *minimized* with respect to $\mathbf{w}$, b and $\boldsymbol{\xi}$, and *maximized* with respect to nonnegative α_i and β_i.



The solution is a **hyperplane again**. No perfect separation however!

See in the book the details of the solution!

However, the hyperplanes cannot be the solutions when the decision boundaries are nonlinear.



Now, the SVM should be constructed by

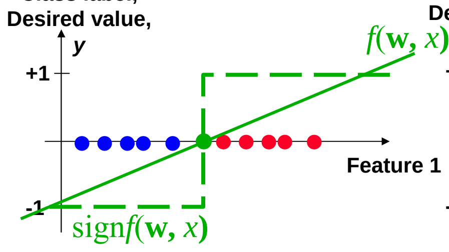
i) mapping input vectors nonlinearly into a high dimensional feature space and,

ii) by constructing the OCSH in the high dimensional feature space.

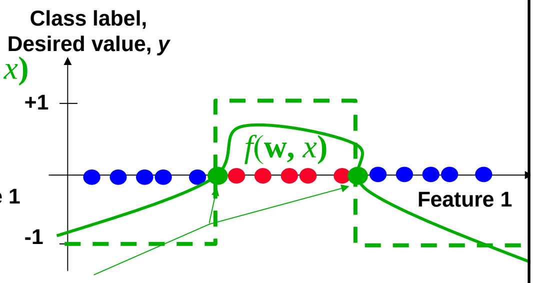
Let's analyze a very low dimensional problem of classifying two classes based on a single feature.

Thus, we believe that the Feature 1 **only** can be useful for classification!

Label classes as: y = +1 for class 1, y = -1 for class 2



This is an EASY problem



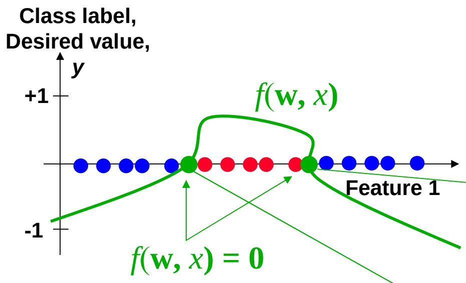
This is a very COMPLEX problem

What about solving such a complex NONLINEAR problem

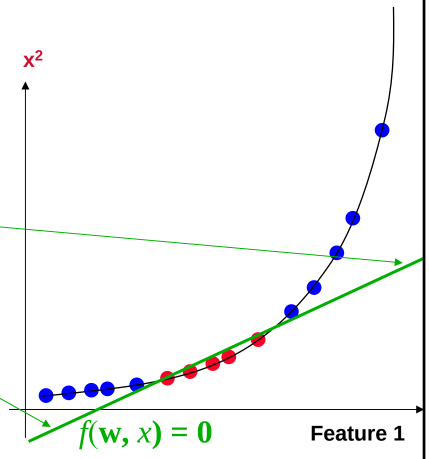
There are two possibilities:

1) Solve in original *x domain*

2) Map data into an **extended features' *domain***



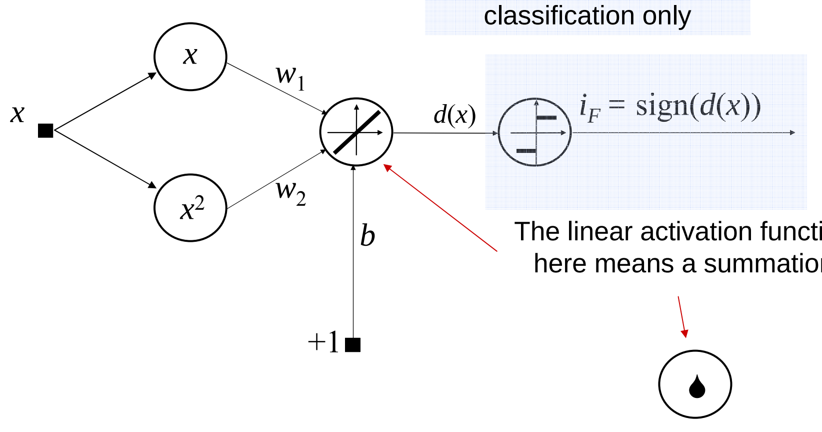
Design a NONLINEAR *f*(**w**, *x*)



Design a LINEAR decision function in a NEW features plane.

Note that we do not see Class Labels here!

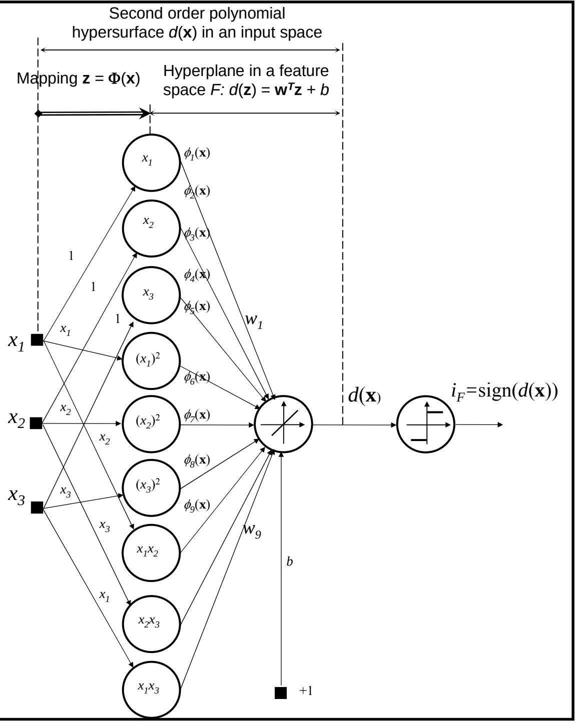
An extension (mapping) of an input space x into the feature one $[x \quad x^2]$ can be given the graphical representation in the form of a 'neural' network below



The thresholding shown here is needed for a classification only

The linear activation function here means a summation

SVMs arise from more complex mapping of an n-dimensional input vector **x** = $[x_1\ x_2\ \ldots\ x_n]^T$ into a feature vector **z** = $\boldsymbol{\Phi}$(**x**).



Now, we apply a '**kernel trick'.**

One basic idea in designing nonlinear SV machines is to map input vectors $\mathbf{x} \in \Re^n$ into vectors $\mathbf{z}$ of a higher dimensional *feature space* $F(\mathbf{z}) = \Phi(\mathbf{x})$ where Φ represents mapping: $\Re^n \rightarrow \Re^f$ and to solve a linear classification problem in this feature space

$$\mathbf{x} \in \Re^n \rightarrow \mathbf{z}(\mathbf{x}) = [a_1\phi_1(\mathbf{x}), a_2\phi_2(\mathbf{x}), \ldots, a_f\phi_f(\mathbf{x})]^T \in \Re^f$$

The solution for an indicator function $i_F(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{z}(\mathbf{x}) + b)$, which is a linear classifier in a feature space F, will create a nonlinear separating hypersurface in the original input space given by

$$i_F(\mathbf{x}) = \text{sign}\ (\sum_{i=1}^{l} \alpha_i y_i \mathbf{z}^T(\mathbf{x})\mathbf{z}(\mathbf{x}_i) + b)$$

$$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{z}_i^T\mathbf{z}_j = \Phi^T(\mathbf{x}_i)\Phi(\mathbf{x}_j).$$

Note that a *kernel function* $K(\mathbf{x}_i, \mathbf{x}_j)$ is a function in input space.

Kernel functions	**Type of classifier**
$K(\mathbf{x}, \mathbf{x}_i) = [(\mathbf{x}^T\mathbf{x}_i) + 1]^d$	Polynomial of degree d
$K(\mathbf{x}, \mathbf{x}_i) = e^{-\frac{1}{2}[(\mathbf{x}-\mathbf{x}_i)^T \Sigma^{-1} (\mathbf{x}-\mathbf{x}_i)]}$	Gaussian RBF
$K(\mathbf{x}, \mathbf{x}_i) = \tanh[(\mathbf{x}^T\mathbf{x}_i) + b]$*	Multilayer perceptron

*only for certain values of *b*

The learning procedure is the same as the construction of a 'hard' and 'soft' margin classifier in **x**-space previously.

Now, in **z**-space, the dual Lagrangian that should be maximized is

$$L_d(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j \mathbf{z}_i^T \mathbf{z}_j \qquad \text{or,}$$

$$L_d(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j)$$

and the constraints are

$$\alpha_i \geq 0, \qquad i = 1, l$$

In a more general case, because of noise or generic class' features, there will be an overlapping of training data points. Nothing but constraints change as for the soft margin classifier above. Thus, the nonlinear 'soft' margin classifier will be the solution of the quadratic optimization problem given above subject to constraints

$$C \geq \alpha_i \geq 0, \qquad i = 1, l \qquad \text{and} \qquad \sum_{i=1}^{l} \alpha_i y_i = 0$$

The decision hypersurface is given by

$$\boxed{d(\mathbf{x}) = \sum_{i=1}^{l} y_i \alpha_i K(\mathbf{x}, \mathbf{x}_i) + b}$$

We see that the final structure of the SVM is equal to the NN model.
In essence it is a weighted linear combination of some kernel (basis) functions. We'll show this (hyper)surfaces in simulations later.

Regression

by

Support Vector Machines

Comparisons of some popular regression schemes

d is a dimension of the model. For NL models it corresponds to the # of HL neurons, i.e., to the # of SVs!

Method	Functional to minimize	Solution
Linear regression	$\Sigma e^2 = \Sigma(y - f(\mathbf{x}, \mathbf{w}))^2$ $d << l$	$f(\mathbf{x}, \mathbf{w}) = \mathbf{x}^T\mathbf{w}$ $\mathbf{w} = \mathbf{X}^+\mathbf{y}$
Ridge regression	$\Sigma e^2 = \Sigma(y - f(\mathbf{x}, \mathbf{w}))^2 + \lambda\|\mathbf{w}\|^2$ $d << l$	$f(\mathbf{x}, \mathbf{w}) = \mathbf{x}^T\mathbf{w}$ $\mathbf{w} = (\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I})^{-1} \mathbf{X}^T\mathbf{y}$
RBF networks, **approximation**	$\Sigma e^2 = \Sigma(y - f(\mathbf{x}, \mathbf{w}))^2$ $d << l$	$f(\mathbf{x}, \mathbf{w}) = \Sigma_{i=1,d}\, w_i g(\mathbf{x} - \mathbf{c}_i)$ $\mathbf{w} = \mathbf{G}^+\mathbf{y}$, $\mathbf{c}_i$ is predefined
RBF networks, **interpolation**	$\Sigma e^2 = \Sigma(y - f(\mathbf{x}, \mathbf{w}))^2$ $d = l$	$f(\mathbf{x}, \mathbf{w}) = \Sigma_{i=1,l}\, w_i g(\|\mathbf{x} - \mathbf{x}_i\|)$ $\mathbf{w} = \mathbf{G}^{-1}\mathbf{y}$, $\mathbf{c}_i = \mathbf{x}_i$
Regularization Networks (RNs)	$\Sigma(y - f(\mathbf{x}, \mathbf{w}))^2 + \lambda\|f\|_{FS}^2$ $d = l$	$f(\mathbf{x}, \mathbf{w}) = \Sigma_{i=1,l}\, w_i g(\|\mathbf{x} - \mathbf{x}_i\|)$ $\mathbf{w} = (\mathbf{G} + \lambda\mathbf{I})^{-1}\mathbf{y}$, $\mathbf{c}_i = \mathbf{x}_i$
SVMs	$L_\varepsilon + \lambda\|f\|_{FS}^2$ # of SV $<< l$	$f(\mathbf{x}, \mathbf{w}) = \Sigma_{i=1,l}\, w_i g(\|\mathbf{x} - \mathbf{x}_i\|)$ $\mathbf{w}$ by **QP**, $\mathbf{c}_i = \mathbf{x}_i$, but note that many $w_i = 0$, SPARSENESS

The crucial difference between RNs and SVMs is in a loss function used! Note that an **application of Vapnik's ε-insensitivity loss function L_ε** leads to QP learning and to the **sparse solution**. Only a fraction of data points is important! They are SVs! $\longrightarrow$ Data compression!

Regression by SVMs

Initially developed for solving classification problems, SV techniques can be successfully applied in regression, i.e., for a functional approximation problems (Drucker et al, (1997), Vapnik et al, (1997)).

Unlike pattern recognition problems (where the desired outputs y_i are discrete values e.g., Boolean), here we deal with *real valued* functions.

Now, the general regression learning problem is set as follows;

the learning machine is given *l* training data from which it attempts to learn the input-output relationship (dependency, mapping or function) $f(\mathbf{x})$.

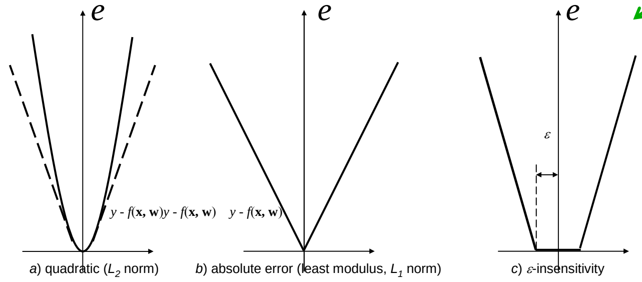
A training data set $D = \{[\mathbf{x}(i), y(i)] \in \Re^n \times \Re,\ i = 1,...,l\}$ consists of *l* pairs $(\mathbf{x}_1, y_1)$, $(\mathbf{x}_2, y_2)$, …, $(\mathbf{x}_l, y_l)$, where the inputs $\mathbf{x}$ are *n*-dimensional vectors $\mathbf{x} \in \Re^n$ and system responses $y \in \Re$, are continuous values. The SVM considers approximating functions of the form

$$f(\mathbf{x}, \mathbf{v}) = \sum_{i=1}^{N} v_i \varphi_i(\mathbf{x})$$

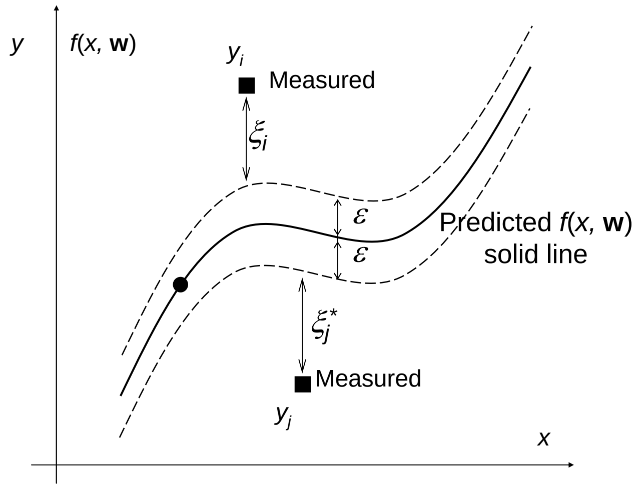
Vapnik introduced a more general error (loss) function -
the so-called **ε-insensitivity loss function**

$$|y - f(\mathbf{x}, \mathbf{w})|_\varepsilon = \begin{cases} 0 & \text{if } |y - f(\mathbf{x}, \mathbf{w})| \le \varepsilon \\ |y - f(\mathbf{x}, \mathbf{w})| - \varepsilon, & \text{otherwise.} \end{cases}$$

Thus, the loss is equal to 0 if the difference between the predicted *f*(**x, w**) and the measured value is less than ε. Vapnik's ε-insensitivity loss function **defines an ε tube** around *f*(**x, w**). If the predicted value is within the tube the loss (error, cost) is zero. For all other predicted points outside the tube, the loss equals the magnitude of the difference between the predicted value and the radius ε of the tube. **See the next figure.**



a) quadratic (L_2 norm) *b*) absolute error (least modulus, L_1 norm) *c*) ε-insensitivity



The parameters used in (1-dimensional) support vector regression.

Now, minimizing risk R equals

$$R_{\mathbf{w},\xi,\xi^*} = \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{l}\xi + C\sum_{i=1}^{l}\xi^*$$

and the constraints are,

$$\begin{array}{ll} y_i - \mathbf{w}^T\mathbf{x}_i - b \le \varepsilon + \xi, & i = 1, l, \\ \mathbf{w}^T\mathbf{x}_i + b - y_i \le \varepsilon + \xi^*, & i = 1, l, \\ \xi \ \ge\ 0 & i = 1, l, \\ \xi^* \ge\ 0 & i = 1, l, \end{array}$$

where ξ and ξ^* are slack variables shown in previous figure for measurements **‘above’** and **‘below’** an ε-tube respectively. Both slack variables are positive values. Lagrange multipliers (that will be introduced during the minimization) α_i and α_i^* corresponding to ξ and ξ^* will be nonzero values for training points ‘above’ and ‘below’ an ε-tube respectively. Because no training data can be on both sides of the tube, either α_i or α_i^* will be nonzero. For data points inside the tube, both multipliers will be equal to zero.

Similar to procedures applied to SV classifiers, we solve this constrained optimization problem by forming a *primal variables Lagrangian* $L_p(\mathbf{w}, \xi, \xi^*)$ Step 1

$$L_p(\mathbf{w}, b, \xi, \xi^*, \alpha_i, \alpha_i^*, \beta_i, \beta_i^*) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\left(\sum_{i=1}^{l}\xi + \sum_{i=1}^{l}\xi^*\right) - \sum_{i=1}^{l}\alpha_i^*\left[y_i - \mathbf{w}^T\mathbf{x}_i - b + \varepsilon + \xi_i^*\right]$$

$$-\sum_{i=1}^{l}\alpha_i\left[\mathbf{w}^T\mathbf{x}_i + b - y_i + \varepsilon + \xi_i\right] - \sum_{i=1}^{l}(\beta_i^*\xi_i^* + \beta_i\xi_i)$$

A primal variables Lagrangian $L_p(w_i, b, \xi, \xi^*, \alpha, \alpha^*, \beta, \beta^*)$ has to be *minimized* with respect to primal variables $\mathbf{w}$, b, ξ and ξ^* and *maximized* with respect to nonnegative LaGrange multipliers α, α^*, β and β^*. This problem can be solved again either in a ***primal space*** or in a ***dual*** one. Below, we consider a solution in a dual space. Applying Karush-Kuhn-Tucker (KKT) conditions for regression, we will *maximize a dual variables Lagrangian* $L_d(\alpha, \alpha^*)$ Step 3

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

subject to constraints

$$\sum_{i=1}^{l}\alpha_i^* = \sum_{i=1}^{l}\alpha_i$$

$0 \le \alpha_i^* \le C \qquad i = 1, l,$

$0 \le \alpha_i \le C \qquad i = 1, l.$

Note that **a dual variables Lagrangian $L_d(\alpha, \alpha^*)$ is expressed in terms of LaGrange multipliers α and α^* only**, and that - the size of the problem, with respect to the size of an SV classifier design task, is doubled now.

There are $2l$ unknown multipliers for linear regression and the Hessian matrix **H** of the quadratic optimization problem in the case of regression is a $(2l, 2l)$ matrix.

The ***standard quadratic optimization problem*** above can be expressed in a ***matrix notation*** and formulated as follows:

Maximize Step 3 in a matrix form

$$L_d(\alpha) = -0.5\boldsymbol{\alpha}^T\mathbf{H}\,\boldsymbol{\alpha} + \mathbf{f}^T\boldsymbol{\alpha},$$

subject to constraints above where for a **linear regression**,

$\mathbf{H} = [\mathbf{x}^T\mathbf{x} + 1]$, $\mathbf{f} = [\varepsilon - y_1\ \varepsilon - y_2, \ldots, \varepsilon - y_N, \varepsilon + y_1, \varepsilon + y_2, \ldots, \varepsilon + y_{2N}]$.

More interesting, common and challenging problem is to aim at solving the ***nonlinear regression tasks***. Here, similar as in the case of nonlinear classification, this will be achieved by considering a linear regression hyperplane in the so-called *feature space*.

Thus, we use the same basic idea in designing SV machines for creating a nonlinear regression function.

We map input vectors $\mathbf{x} \in \Re^n$ into vectors $\mathbf{z}$ of a higher dimensional *feature space* F ($\mathbf{z} = \Phi(\mathbf{x})$ where Φ represents mapping: $\Re^n \rightarrow \Re^f$) and **we solve a linear regression problem in this feature space.**

A mapping $\Phi(\mathbf{x})$ is again chosen in advance. Such an approach again leads to solving a quadratic optimization problem with inequality constraints in a **z**-space. The solution for an regression hyperplane $f = \mathbf{w}^T\mathbf{z}(\mathbf{x}) + b$ which is linear in a feature space F, will create a nonlinear regressing hypersurface in the original input space. In the case of nonlinear regression, after calculation of LaGrange multiplier vectors $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}^*$, we can find an optimal desired weight vector of the *kernels expansion* $\mathbf{v}_o$ as Step 4

$$\mathbf{v}_o = \boldsymbol{\alpha}^* - \boldsymbol{\alpha},$$

and an optimal bias b_o can be found from $b_o = \frac{1}{l}\sum_{i=1}^{l}(y_i - g_i)$.

where $\mathbf{g} = \mathbf{G}\,\mathbf{v}_o$ and the matrix **G** is a corresponding design matrix of given RBF kernels.

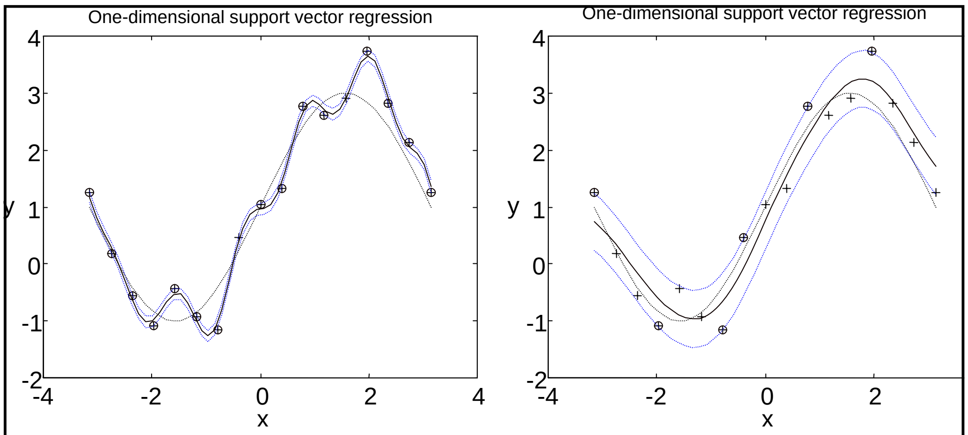
Step 5

The best nonlinear regression hyperfunction is given by

$$z = f(\mathbf{x}, \mathbf{v}) = \mathbf{G}\mathbf{v} + b.$$

There are a few learning parameters in constructing SV machines for regression. The two most relevant are **the insensitivity zone *e*** and the **penalty parameter *C*** that determines the trade-off between the training error and VC dimension of the model. **Both parameters should be chosen by the user.**

Generally, an increase in an insensitivity zone *e* has smoothing effects on modeling highly noisy polluted data. Increase in *e* means a reduction in requirements on the accuracy of approximation. It decreases the number of SVs leading to data compression too. See the next figures.



The influence of **a insensitivity zone *e* on modeling quality.** *A* nonlinear SVM creates a regression function with **Gaussian kernels** and models a highly polluted (25% noise) sinus function (dashed). 17 measured training data points (plus signs) are used.

Left: *e* = 0.1. 15 SV are chosen (encircled plus signs).

Right: *e* = 0.5. 6 chosen SV produced a much better regressing function.

Some of the constructive problems:

The SV training works almost perfectly for not too large data basis.

However, when the number of data points is large (say l > 2000) the QP problem becomes extremely difficult to solve with standard methods. For example, a training set of **50,000 examples amounts to a Hessian matrix H with 2.5*10^9 (2.5 billion) elements. Using an 8-byte floating-point representation we need 20,000 Megabytes = 20 Gigabytes of memory** (Osuna et al, 1997). This cannot be easily fit into memory of present standard computers.

There are three, now classic, approaches that resolve the QP for large data sets. Vapnik in (**Vapnik, 1995**) proposed the ***chunking method*** that is the decomposition approach. Another decomposition approach is proposed in (**Osuna, Girosi, 1997**). The sequential minimal optimization (**Platt, 1997**) algorithm is of different character (works with 2 data points at the time) and it seems to be an 'error back propagation' for a SVM learning.

The newest iterative single data (per-pattern) algorithm (Kecman, Vogt, Huang, 2003; Huang, Kecman, 2004) seems to be the fastest for a huge data sets (say, for more than a few hundred thousands data pairs) at the moment!

Let us summarize the story by presenting the basic constructive steps that lead to SV machine:

- **selection of the kernel function that determines the shape of the decision and regression function in classification and regression problems respectively,**

- **selection of the 'shape', i.e., 'smoothing' parameter in the kernel function (for example, polynomial degree and variance of the Gaussian RBF for polynomials and RBF kernels respectively),**

- **choice of the penalty factor *C* and selection of the desired accuracy by defining the insensitivity zone *e*,**

- **solution of the QP problem in *l* and 2*l* variables in the case of classification and regression problems respectively.**

Now, what to do when faced with a huge data sets???

See about the Iterative algorithm known as Iterative Single Data Algorithm **ISDA** aimed at solving huge data set problems iteratively.

There is also an interesting approach for medium sized data sets based on an Active Set method see the chapters below:

Kecman, V., T. M. Huang, M. Vogt, Chapter 'Iterative Single Data Algorithm for Training Kernel Machines from Huge Data Sets: Theory and Performance', in a Springer-Verlag book, 'Support Vector Machines: Theory and Applications', Ed. L. Wang, 2005

Vogt, M., V. Kecman, Chapter 'Active-Set Methods for Support Vector Machines', in a Springer-Verlag book, 'Support Vector Machines: Theory and Applications', Ed. L. Wang, 2005.

Both chapters are downloadable from:

http://www.support-vector.ws

Let us conclude the seminar by a comparisons between the SVMs and NNs

- **both the NNs and SVMs learn from experimental data,**

- **both the NNs and SVMs are universal approximators in the sense that they can approximate any function to any desired degree of accuracy,**

- **after the learning they are given with the same mathematical model, as the sum of weighted basis (kernel) funstions, and they can be presented graphically with the same so-called NN's graph,**

- **they differ by the learning method used. While NNs typically use either EBP (or some more sophisticated gradient descent algorithm) or some other linear algebra based approach, the SVMs learn by solving the QP or LP problem.**

That would be the end

Questions please

The end