Institut Ruđer Bošković

Contemporary Machine Learning i.e., Computational Intelligence,

or

Classic and Novel ML Tools

Vojislav Kecman

Learning Algorithms and

Applications Laboratory (LAAL)

λα αλ

λα αλ

VCU

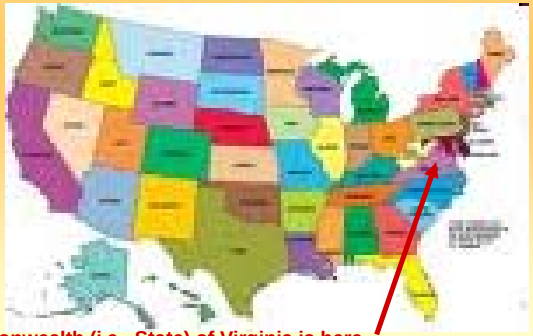
VCU School of Engineering

COMPUTER SCIENCE

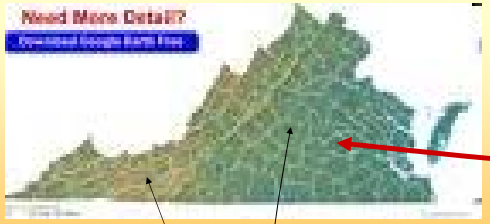
Greetings from VCU!!!

What is VCU and where is it???

- VCU stands for VIRGINIA COMMONWEALTH UNIVERSITY



and, the Commonwealth (i.e., State) of Virginia is here



Richmond, capitol of Virginia (and of Confederacy in the Civil War) is here

Virginia Tech.

UNIVERSITY OF VIRGINIA

and, the mascot of our athletic teams is Rodney the Ram

COMPUTER SCIENCE

Rodney The Ram

Today I will make a gentle walk through ML approaches and techniques as follows:

- An Introduction into the area **connecting classic tools** with **novel ones**
- Focusing on the most powerful tool in ML today SVMs
- and, if time allows
  - Advanced SVMs concepts and topics
  - Basic experimental considerations
    - Bias-Variance, Cross-Validation

DON'T WORRY. TIME WON'T ALLOW THIS TORTURE!!!

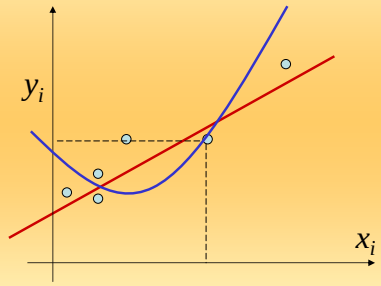
SOME TOPICS

- 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 **a drowning in data, while starving for knowledge**
- Basic Model of Computational Intelligence (i.e., machine learning) - **The Sum of Weighted Basis Functions**
- One model = Many models
- Quo wadis ML ?
- Some Contemporary Tools

What is a Learning from Data, or Data Mining, about?

- Mathematics in the last 3,000 years was playing with such models:
- $A(r) = \pi r^2 = w_1 r^2$, $v(h) = \mathrm{sqrt}(2gh) = w_1 \mathrm{sqrt}(h)$,
- $y = 3x - 2 = w_1 x + w_2$, $z = -x + y - 3 = w_1 x + w_2 y + w_3$
  Parameters $\boldsymbol{w_i}$ **of the relations are known;** given the independent variable(s) one finds the dependent one(s)!
- **TODAY; we want to learn the relation from the measured pairs (x_i, y_i) given as data sets, by infering i.e., learning the UNKNOWN parameter values w_i .**
- **This is an INVERSE PROBLEM stated as:**
  having the pairs (x_i, y_i) find the parameters w_i, of the model.
- **In other words, LEARN the dependency between the x_i and y_i!**

or, the problems to solve are a kind of this one: having the data points ◦ find weights (parameters) which define a function assumed (here linear and quadratic ones are assumed)



In an real life, examples are same in character but much larger in both DIMENSIONS and NUMBERS

Today, we live surrounded by an **OCEAN OF 'DATA'?**

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

We produce data, transfer it, compress it, use it, process it, reuse it, 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 30 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 nowadays - 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

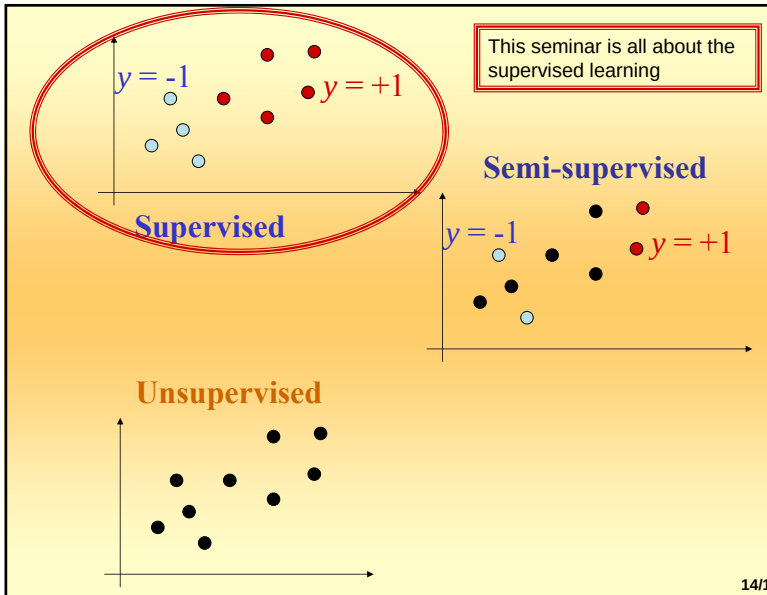
Few more examples:

- Banks: Fraud checks detection
- Google, Microsoft et al: Targeted advertising
- Supermarkets: Promotion planning
- Call centers: Speech recognition
- Scanners: Optical character recognition
- Web pages classification, Text categorization
- Post office: Zip code handwriting recognition
- Credit cards: Loan default prediction
- Stock market: Statistical arbitrage
- Drug design: Drug candidate screening
- Large Hadron Collider: Particle screening
- Airport scanner: Explosives, Drugs, Arm, Faces

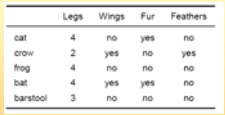
Let's first set the stage, there are three (3) machine learning (ML) settings

- **Supervised** (pairs x_i, y_i are given for **all** data pairs, where x_i are the values of the independent variables, features, inputs, attributes and y_i are class labels)
- **Semi-supervised** (pairs x_i, y_i are given for **just a fraction** of data pairs)
- **Unsupervised** (only inputs x_i are given and no single label y_i is known)

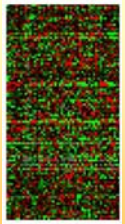
Here, we deal only with **SUPERVISED** ML problems!

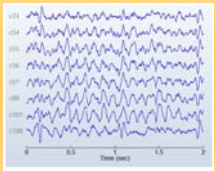


Data sets variety



	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

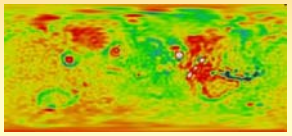




Play golf dataset

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



Mars magnetic field

Supervised Machine 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 playing with here, is the **Density Estimation Problem**

Classification (Pattern Recognition)

- Classification (Pattern Recognition) setting is as follows

You want your model, i.e., function implemented in software, i.e., NN, i.e., Decision Function, i.e., SVM
to be trained on training data sets comprised of the training pairs ($\mathbf{x}_i$, y_i), and
to be used on the new, previously unseen inputs $\mathbf{x}_i$, in order to recognize it i.e., classify it i.e., predict it.

$\mathbf{x}_i$ is called an input vector of features, or just the feature vector

y_i is called the output, i.e., desired or target value, or just label

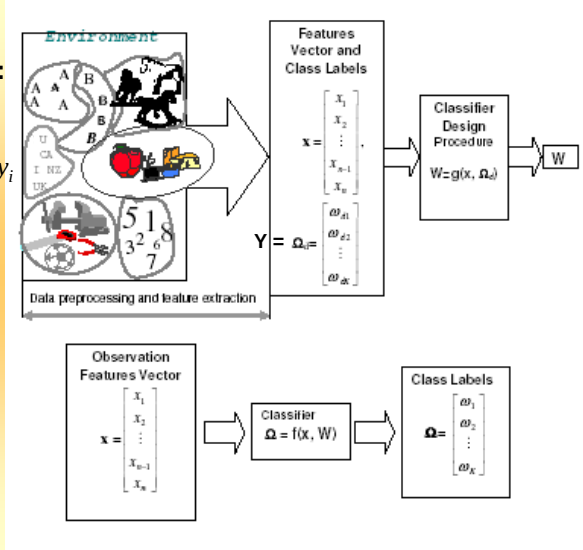
Training Phase:

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

class label $\omega_i = y_i$

Test, i.e. Application, Phase:

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



Hence, our data is given as:

1st data pair

$$\mathbf{X}=\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{21} & \cdots & x_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ x_{l1} & x_{l2} & \cdots & x_{ln} \end{bmatrix}, \quad \mathbf{Y}_{Class} = \begin{bmatrix} +1 \\ -1 \\ \vdots \\ -1 \end{bmatrix}, \text{ or } \quad \mathbf{Y}_{Regress} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_l \end{bmatrix}$$

l^{st} data pair

Check some data sets here: **http://archive.ics.uci.edu/ml/datasets.html**

Just one simple example:

- We are designing **linear** classifier by using **sum-of-error-squares** cost (merit, loss, fitness) function (norm). i.e. we work under *L_2-norm*
- A problem is 1-dimensional for visualization's purposes only
- All the mathematics is same for any-dimensional input vector **x**

Let's solve a 1-dimensional problem

Training pairs (x_i, y_i) are given:

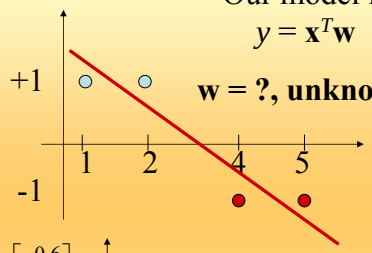
$\mathbf{x}$=[1 2 4 5]', $\mathbf{y}$=[1 1 -1 -1]'

We are after decision function. Assume linear one -> then we are after **weights** (intercept and slope)

Our model is

$y = \mathbf{x}^T\mathbf{w}$

w = ?, unknown



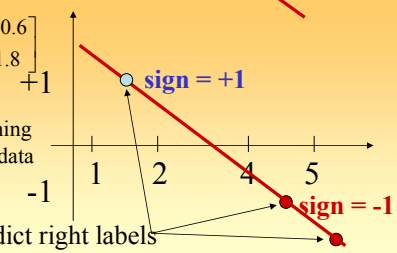
$$\mathbf{X} = \begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 4 & 1 \\ 5 & 1 \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}, \quad \mathbf{w} = \mathbf{X}^{+}\mathbf{y}, \quad \mathbf{w} = \begin{bmatrix} -0.6 \\ 1.8 \end{bmatrix}$$

After, the training we can discard the training data and given the new (test, application) data

$y_{itest} = \mathbf{x}^T_{itest}\mathbf{w}$, say for

$\mathbf{x}$=[1.5 4.5 5.5], our model will predict right labels

$\mathbf{y} = \text{sign}([0.9\ \ -0.9\ \ -1.5]^T)$



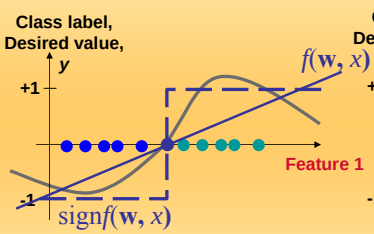
Well, let's go back to our problem of classification.

Here we show what we can see, meaning 1-dimensional or 2-dimensional (1D or 2D) problems (1D or 2D means the input vector x is either 1D or 2D)

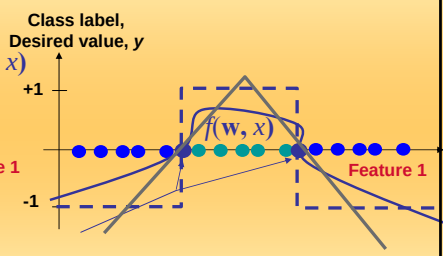
Let's analyze a very low dimensional problem of classifying two classes based on a single feature.

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

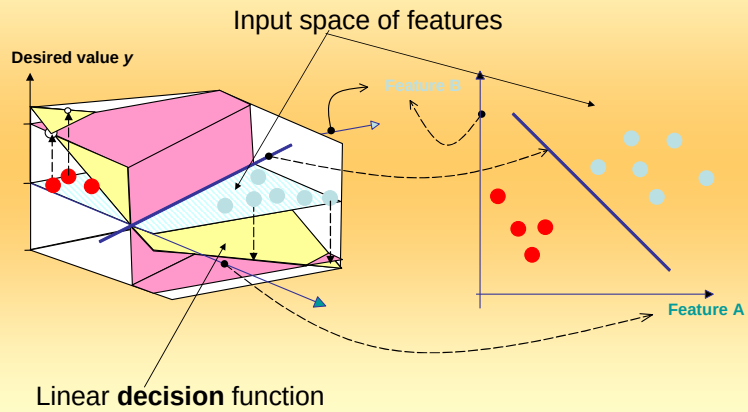


This is a very COMPLEX problem

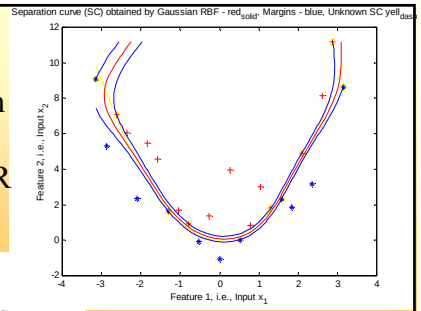
What about solving such a complex NONLINEAR problem

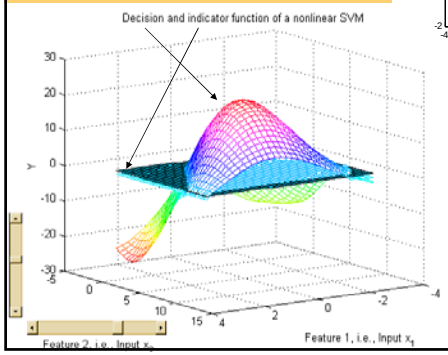
There are many possibilities, and we'll talk about them extensively!

For two dimensional problem: classifying two classes based on 2 features, we can show the **decision** function, but when number of features > 2, we deal with HYPER-surfaces, that can not be seen. However, the algorithms can 'see' in high-dimensional spaces and **they will be the same.**



VERY OFTEN the decision function and separating boundary are NON-LINEAR

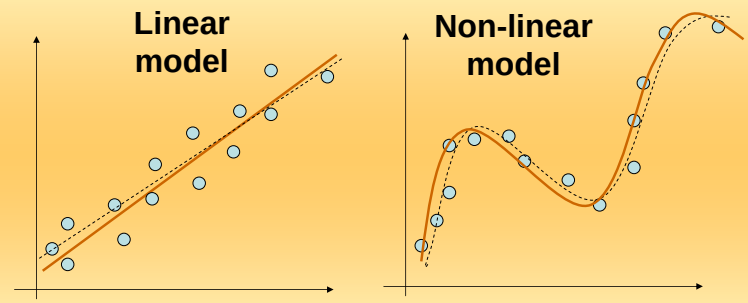




Indicator function (the term Vapnik & Chervonenkis use) is just a

sign(Decision Function)

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

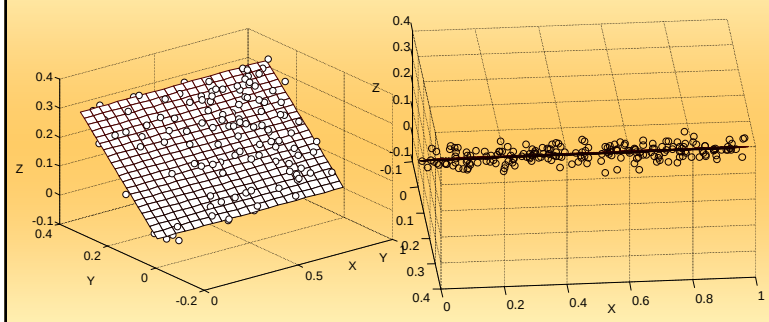


True, but unknown, function

Regression curve

Measured data points

Linear regression (function approximation, surface fitting) over two-dimensional input space



A view along the plane

And now, back to ML classic and novel tools as well as to the connections between them

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

Check my book's site **http://www.support-vector.ws**

for the newest paper's and software's downloads.

Some connections between

NNs *i.e./or/and* SVMs

and

classic techniques such as Fourier series and Polynomial approximations

Classic approximation techniques in NN graphical appearance

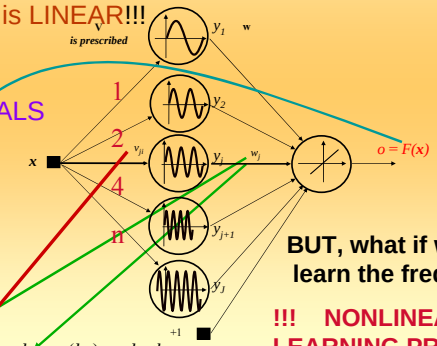
FOURIER SERIES

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

And the problem is LINEAR!!!

It is 'same'

with POLYNOMIALS

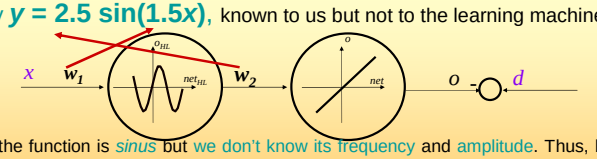


BUT, what if we want to learn the frequencies?

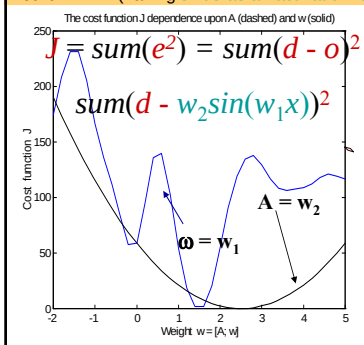
!!! NONLINEAR LEARNING PROBLEM !!!

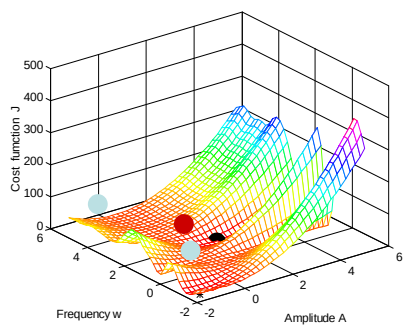
$$F(x) = \sum_{k=1}^{N} a_k \sin(kx),\ or\ b_k \cos(kx),\ or\ both$$

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

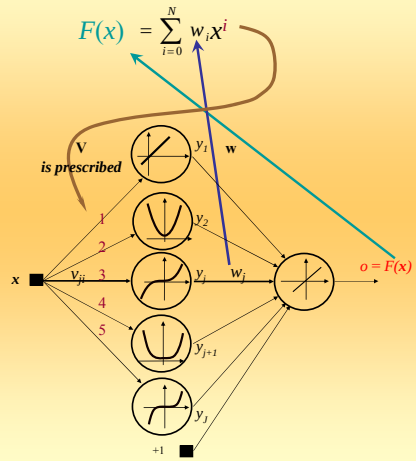




Another classic approximation scheme is a

POLYNOMIAL SERIES

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

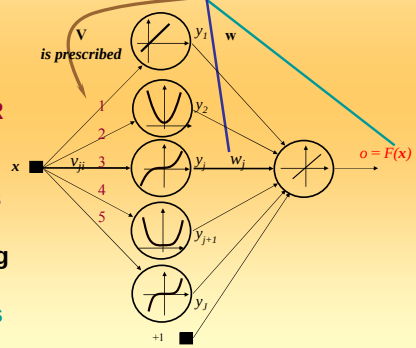


Another classic approximation scheme is a

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.

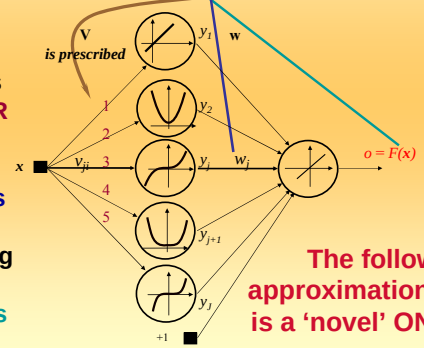


Another classic approximation scheme is a

POLYNOMIAL SERIES

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

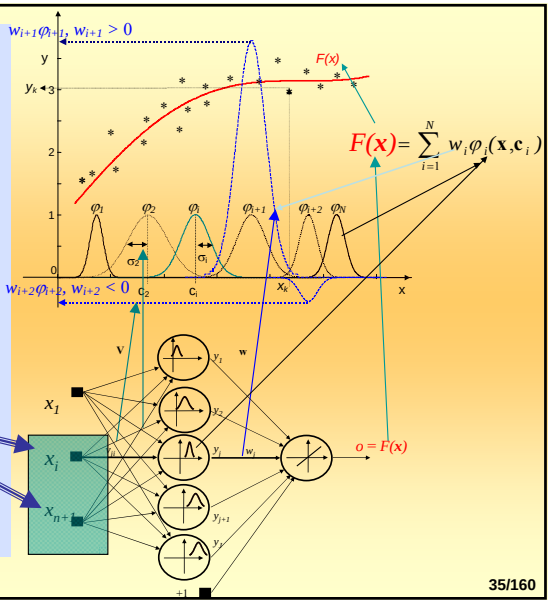
With a prescribed (integer) exponents this is again LINEAR APPROXIMATION SCHEME. Linear in terms of parameters to learn and not in terms of the resulting approximation function. This one is NL function for *i* > 1.



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)

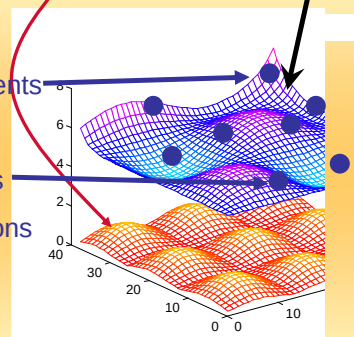
Measurements

Images

Records

Observations

Data



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.

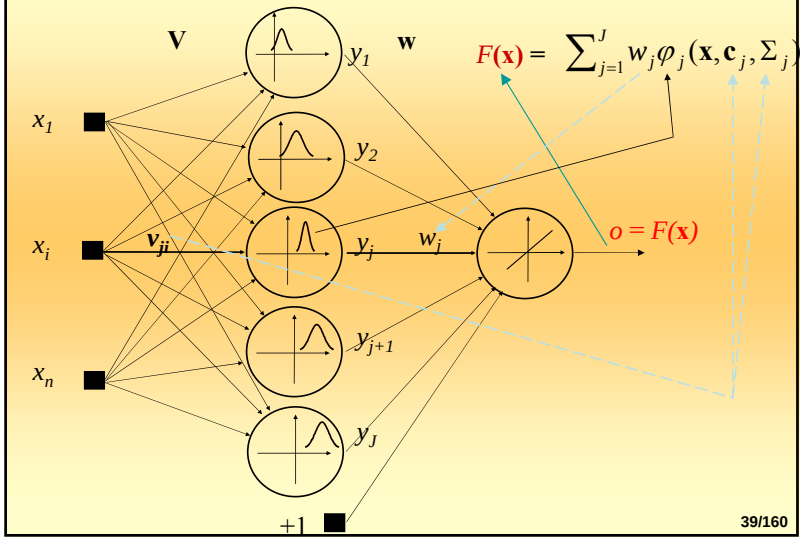
Here, it will be either the so-called **SVM** or the **NN**

(however remember, there are other models too).

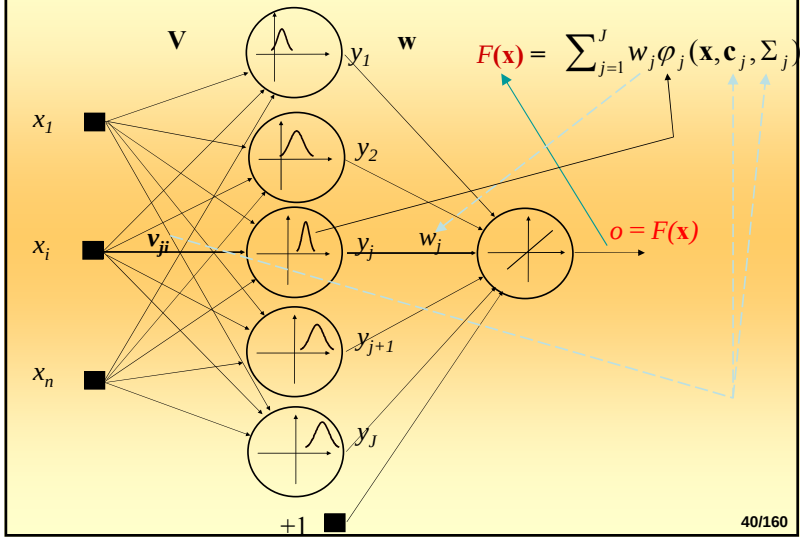
WHAT are DIFFERENCES and SIMILARITIES?

WHATCH CAREFULLY NOW !!!

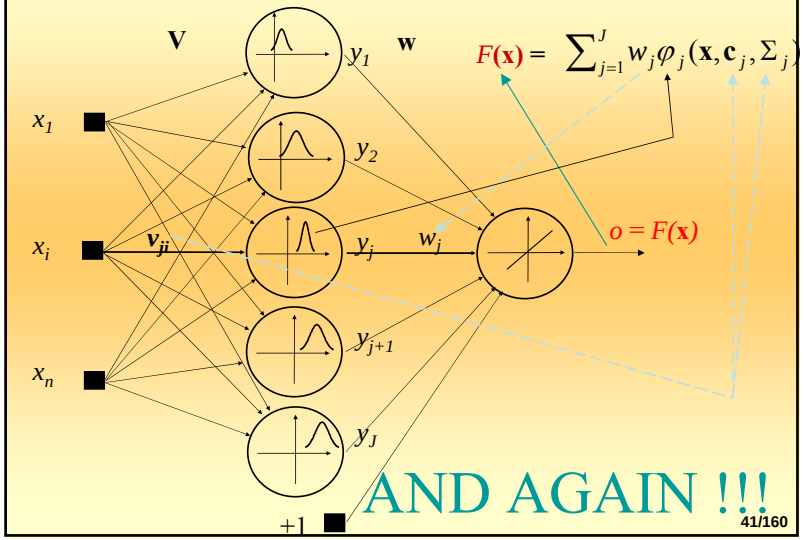
This is a Neural Network,



and, this is a Support Vector Machine.

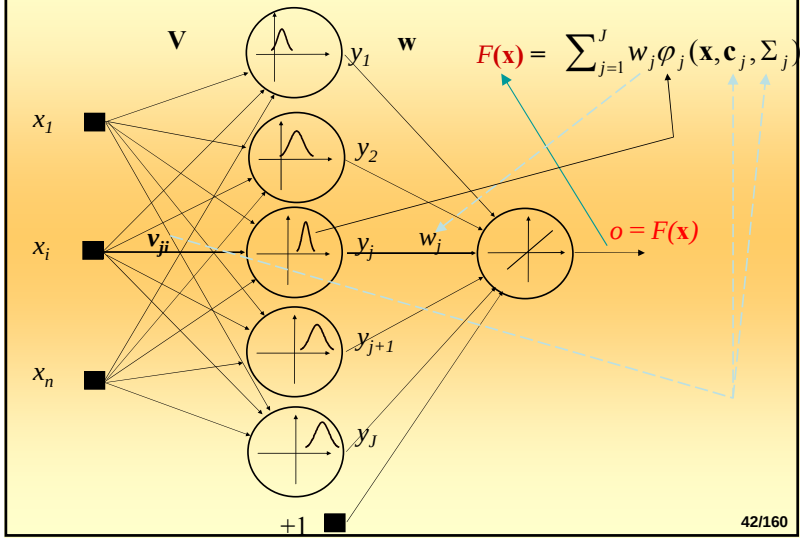


and, this is a Support Vector Machine.

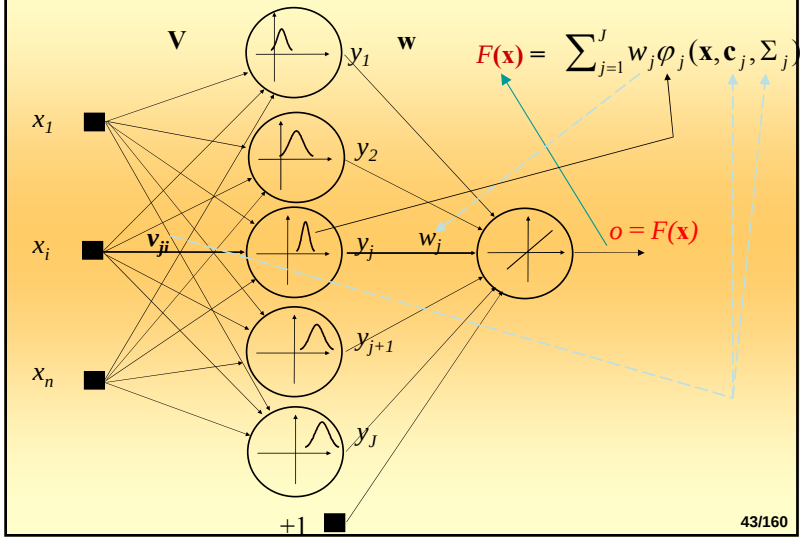


AND AGAIN !!!

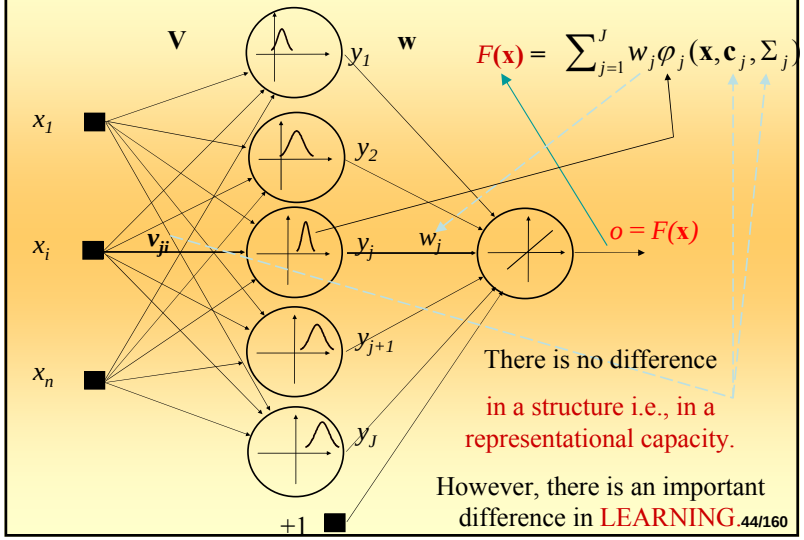
This is a Neural Network,



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.

Where is then the
BASIC DIFFERENCE
between
NNs and SVMs
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, i.e. dual variables in SVMs)

FORM

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

SUM OF THE WEIGHTED BASIS FUNCTIONS

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

Hence,

ONE MODEL = MANY MODELS

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

NORM

- We use **primarily (only) two cost functions** (**NORMS**) in ML which are a MINIMIZATION of the
- **SUM OF ERROR SQUARES in OUTPUT space** (linear standard classifier, FFT, MLP NN and RBF NN)

and the

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

Norms (Loss Functions) of NNs and SVMs

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

Closeness to data

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

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

Closeness to data *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

In the last expression the SRM principle uses the VC dimension *h* (defining model capacity) as a controlling parameter for minimizing the generalization error E (i.e., risk R).

Let's say a little more about the very basics of

the learning from data.

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 of the classic statistical paradigm turned out to be inappropriate for many contemporary real-life problems (Vapnik, Chervonenkis, 1964 - 1998) due to the facts that:

All three assumptions of the classic statistical paradigm turned out to be inappropriate for many contemporary real-life problems (Vapnik, Chervonenkis, 1964 - 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 of the classic statistical paradigm turned out to be inappropriate for many contemporary real-life problems (Vapnik, Chervonenkis, 1964 - 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 of the classic statistical paradigm turned out to be inappropriate for many contemporary real-life problems (Vapnik, Chervonenkis, 1964 - 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* ?**

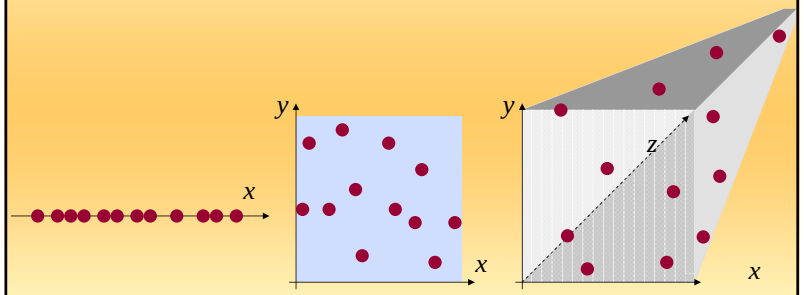
Mostly, all we have are recorded **EXPERIMENTAL DATA** (training patterns, samples, observations, records, examples):
Data is high-dimensional and **scarce (always too little data)!!!**

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.

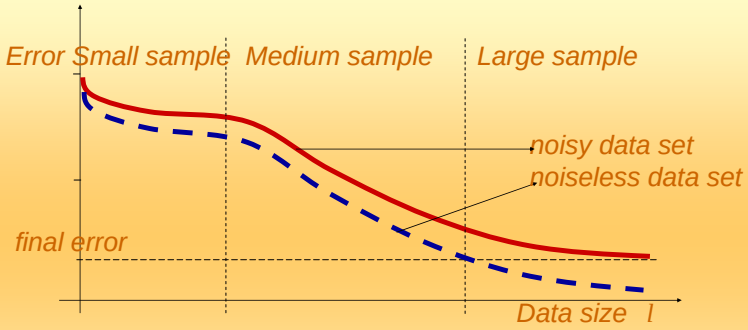
Terrifying emptiness and/or data sparseness

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, but

where are HIGH-DIMENSIONAL problems coming from?

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?

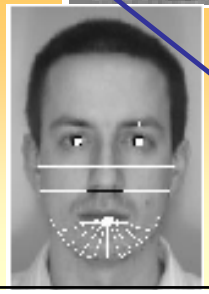
Gender recognition problem: Are these two faces female or male?

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

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

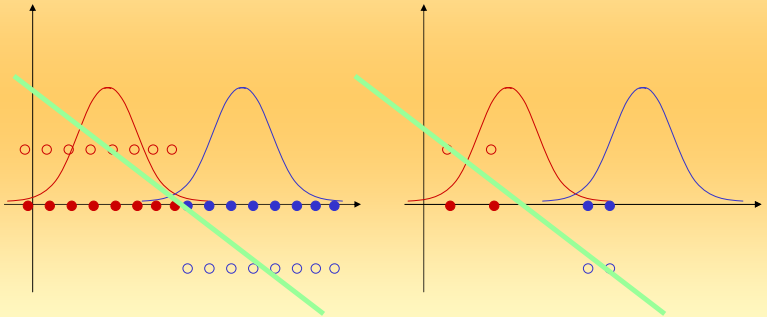
THE INFLUENCE OF A DATA SET SIZE ON THE SIMPLEST RECOGNITION PROBLEM

BINARY CLASSIFICATION, i.e., DICHOTOMIZATION.

First, the simplest case – 1-dim feature (i.e. input), generated by normal, Gaussian distribution

a) enough data

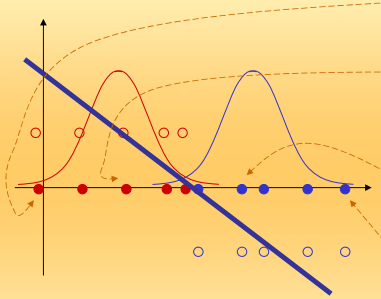
b) sparse data



Sum of error squares will work in the left hand side graph, and it will make BIG error in the right hand side one

How this algorithm works?

Note that a decision function here is given as: $y_i = w_1 x_i + w_2$



$$\mathbf{X} \quad \mathbf{w} = \mathbf{D}$$

$$\begin{bmatrix} -1 & 1.0000 \\ 3 & 1.0000 \\ 5 & 1.0000 \\ 7 & 1.0000 \\ 8 & 1.0000 \\ 9 & 1.0000 \\ 11 & 1.0000 \\ 12 & 1.0000 \\ 14 & 1.0000 \\ 16 & 1.0000 \end{bmatrix} \mathbf{w} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \end{bmatrix}$$

The solution here is given by

$$\mathbf{w} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y} = \mathbf{X}^*\mathbf{y}$$

$\mathbf{X}^*$ is known as the **pseudoinverse**

(we'll show later why is it this way)

How the **pseudoinverse** solution has actually been obtained?

Well, it is an old good math technique for solving both over- and under-determined systems. However, be extremely cautious – the very solutions for the two different cases have entirely different meanings!

Classical (non)Linear Regression (i.e., Classification)

n = number of data, *m* = number of features, attributes, inputs

$$X_{nm}\vec{w}_{m1} = \vec{y}_{n1} \qquad /*_{left}\ X^T_{mn}$$

$$X^T_{mn}X_{nm}\vec{w}_{m1} = X^T_{mn}\vec{y}_{n1}$$

NORMAL SYSTEM ⇨ $$\left(X^T_{mn}X_{nm}\right)\vec{w}_{m1} = X^T_{mn}\vec{y}_{n1}$$

$$\left(X^T_{mn}X_{nm}\right)^{-1}\left(X^T_{mn}X_{nm}\right)\vec{w}_{m1} = \left(X^T_{mn}X_{nm}\right)^{-1}X^T_{mn}\vec{y}_{n1}$$

$$\vec{w}_{m1} = \boxed{\left(X^T_{mn}X_{nm}\right)^{-1}X^T_{mn}}\vec{y}_{n1}$$

$$\hat{\vec{y}}_{test,1} = X_{test,m}\vec{w}_{m1}$$

Pseudoinverse

Usually, *n* > *m* -> overdetermined system

But not always!!!

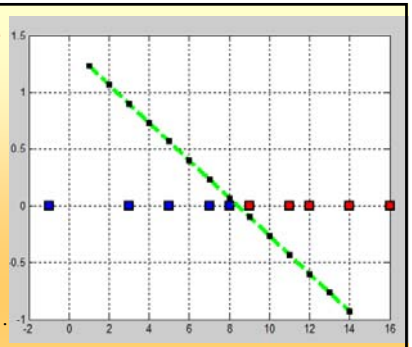
Check the differences in a meaning of the solution **w**

In the case of an over-determined system **w** results in a solution providing the **minimal sum of errors squares**,

and you should look up into the meaning of the solution **w** in the case of an under-determined system.

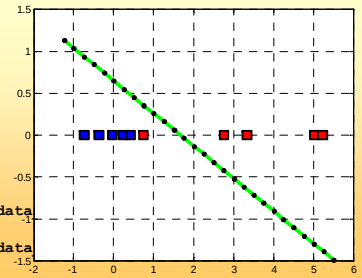
Hint: There is an infinity of solutions: which one is extracted by the pseudoinverse?

Matlab code for previous example



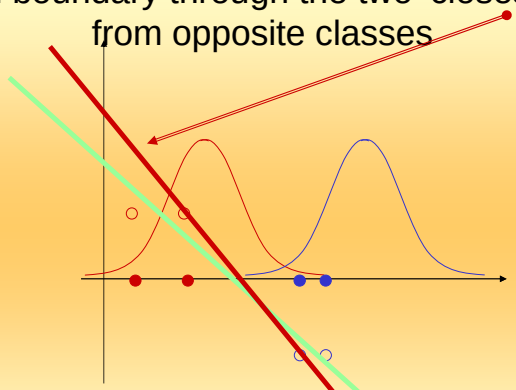
```
x1=[-1;3;5;7;8;9;11;12;14;16];
i1=ones(10,1);
x=[x1 i1]
d=[1 1 1 1 1 -1 -1 -1 -1 -1]
d=d'

w=pinv(x)*d
plot(x1(1:5),zeros(size(x1)/2,1),'bs',...
                'LineWidth',2,...
                 'MarkerEdgeColor','k',...
                 'MarkerFaceColor','b',...
                 'MarkerSize',10)
hold on
plot(x1(6:10),zeros(size(x1)/2,1),'rs',...
                'LineWidth',2,...
                 'MarkerEdgeColor','k',...
                 'MarkerFaceColor','r',...
                 'MarkerSize',10)


i2=1:1:14;
plot((w(1)*i2+w(2)),'--gs','LineWidth',4,...
                 'MarkerEdgeColor','k',...
                 'MarkerFaceColor','y',...
                 'MarkerSize',2)

grid on
```


More general matlab code



```
% linclass_lect_RS
close all, clear all
num_data_per_class = 5;   shift=3;
% the smaller the shift of the means,
% the bigger overlapping of classes
x1=[randn(num_data_per_class,1);
randn(num_data_per_class,1)+shift];
i1=ones(2*num_data_per_class,1);
x=[x1 i1];
d=[ones(num_data_per_class,1);-ones(num_data
w=pinv(x)*d
plot(x1(1:num_data_per_class),zeros(num_data
                'LineWidth',2,...
                 'MarkerEdgeColor','k',...
                 'MarkerFaceColor','b',...
                 'MarkerSize',10)
hold on
plot(x1(num_data_per_class+1:2*num_data_per_class),zeros(num_data_per_class,1),'r
s',...
                'LineWidth',2,...
                 'MarkerEdgeColor','k',...
                 'MarkerFaceColor','r',...
                 'MarkerSize',10)
x2=[min(x1)-0.5:0.25:max(x1)+0.5]';i2=ones(length(x2),1);
plot(x2,[x2 i2]*w,'--gs','LineWidth',4,...
                 'MarkerEdgeColor','k',...
                 'MarkerFaceColor','y',...
                 'MarkerSize',2)
grid on
```


However, what about this idea – draw the decision boundary through the two 'closest' points from opposite classes

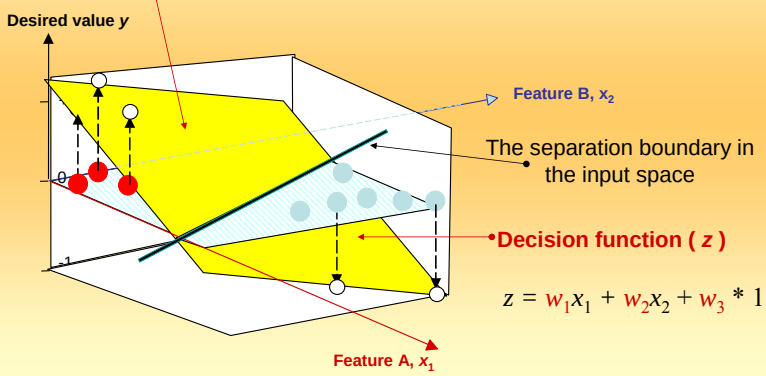


Actually, similar results will be obtained for SVMs, where we **don't bother with the sum of error squares** in the output space

Let's check now the **2-dimensional input case**, and this is the last example where we can represent the decision function graphically.

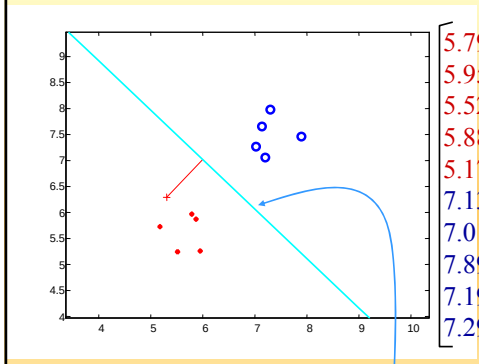
Nevertheless, the algorithms will work for **any-dimensional input**, but following the results visually will not be possible!!!

Decision function in 2-dim case is shown below



$$z = w_1 x_1 + w_2 x_2 + w_3 * 1$$

A simple example how a **classic linear classifier** algorithm works:



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

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

the separation boundary equals

$$x_2 = -0.951 x_1 + 12.725$$

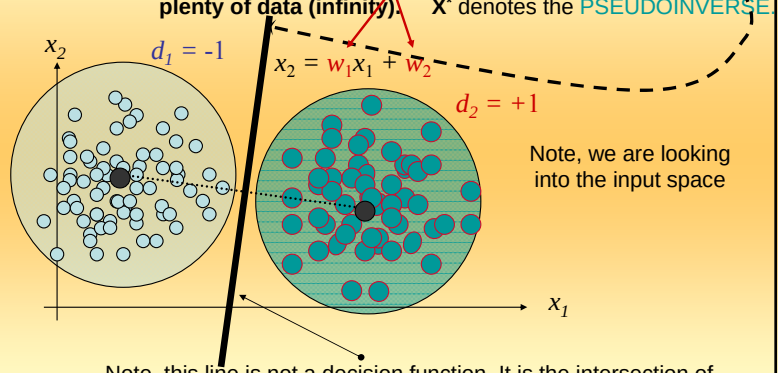
Now, a motivation for a **maximal margin idea**,

or

what to do when having only sparse training data set (not too many data)

CLASSIFICATION or PATTERN RECOGNITION 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.

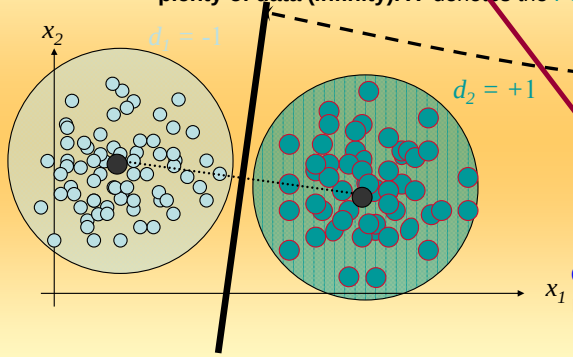


Note, we are looking into the input space

Note, this line is not a decision function. It is the intersection of the decision function and input space called separation boundary.

CLASSIFICATION or PATTERN RECOGNITION 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.



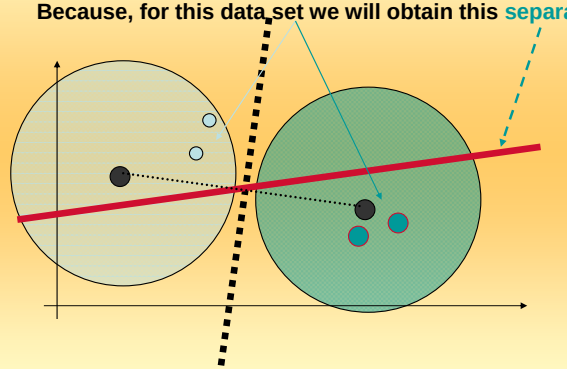
Note that this **solution** follows from the last two assumptions in classic inference!

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

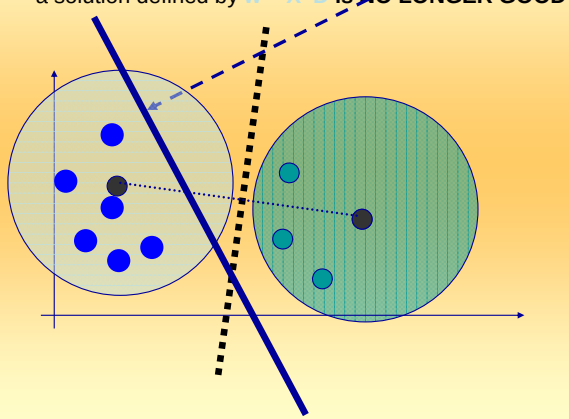
However, for a small sample -

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 !!!**

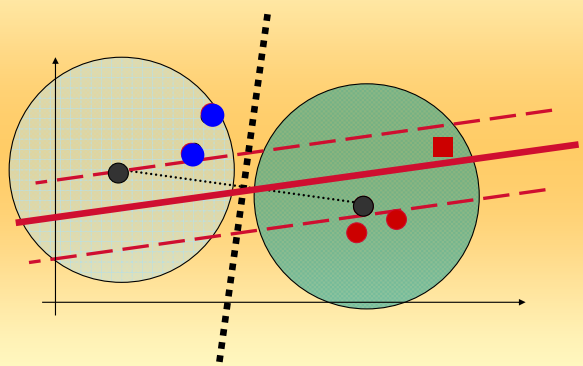


What is common for both separation lines the red and the blue one.

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.

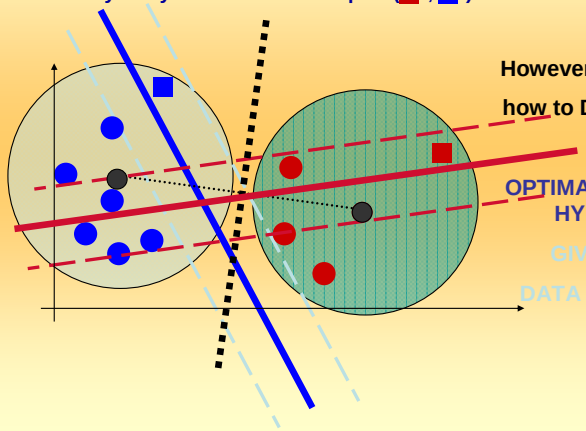


What is common for both separation lines the red and the blue one.

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

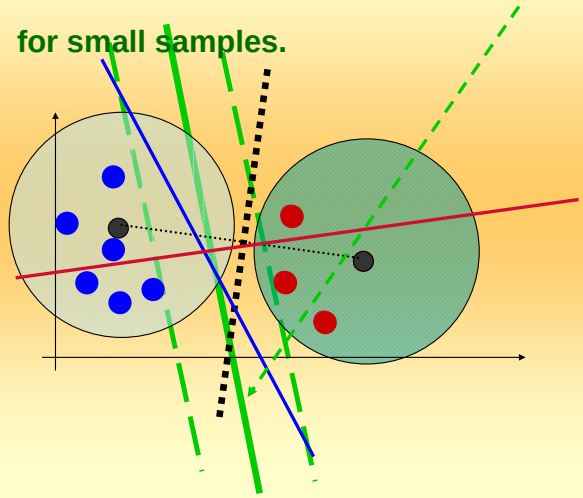
GIVEN (scarce)

DATA SAMPLES ???

The STATISTICAL LEARNING THEORY IS DEVELOPED TO SOLVE

PROBLEMS of FINDING THE OPTIMAL SEPARATION HYPERPLANE

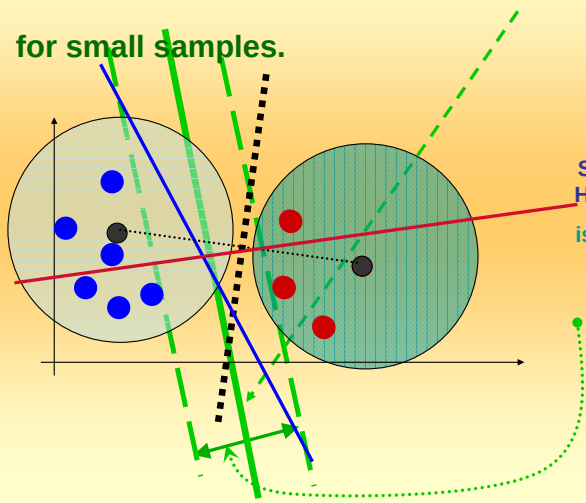
for small samples.



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..

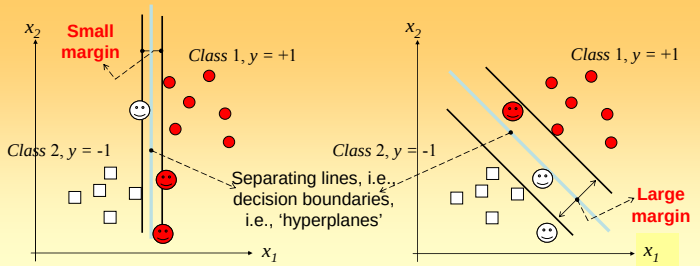
SUPPORT VECTOR MACHINE

is a MAXIMAL MARGIN CLASSIFIER

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

• WHY maximal margin?

Suppose we want 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!

There are two basic, constructive approaches to the minimization of the right hand side of previous equations (Vapnik, Chervonenkis 1964 - 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*.

SVMs

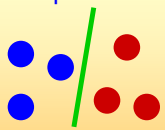
Let us do some more formal,

meaning,

mathematical analysis of SVMs learning!

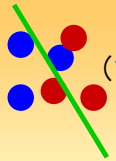
We follow an idea of a **gentle SVMs introduction, i.e., of a gradual proceeding** from the 'simple' cases to the more complex ones!

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

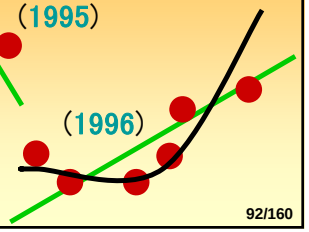


3) ***Nonlinear Classifier*** (1992)

2) ***Linear Soft Margin Classifier for Overlapping Classes.*** (1995)



4) ***Regression by SV Machines*** that can be both **linear** and **nonlinear**! (1996)



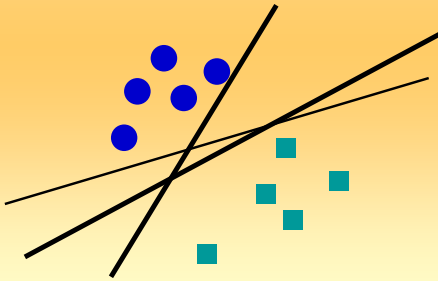
1) Linear Maximal Margin Classifier for Linearly Separable Data

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

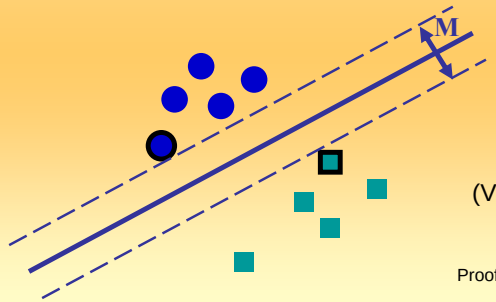
1) Vapnik-Chervonenkis: Optimal separating hyperplane is the one with

MAXIMAL MARGIN !

This hyperplane is uniquely determined by the vectors on the margin

the support vectors!

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



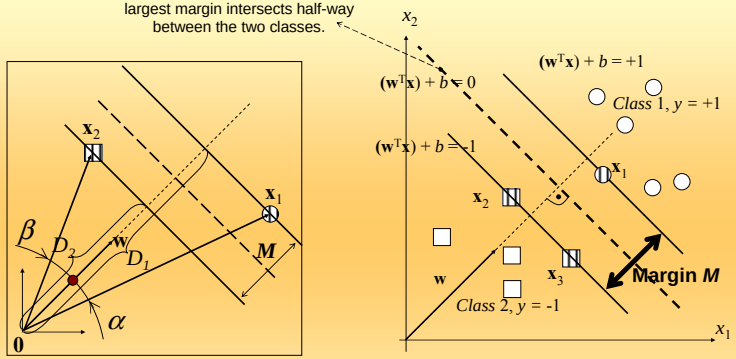
$$M = \frac{2}{\|\mathbf{w}\|}$$

(Vapnik, Chervonenkis '74)

Proof for *M* is on the next two slides

The relation between the weight vector w and the margin *M*

Optimal separating hyperplane with the largest margin intersects half-way between the two classes.



The margin *M* that is to be maximized during the training stage is a projection, onto the separating hyperplane's normal (weight) vector direction, of a distance between any two support vectors belonging to **different** classes. In the example above this margin *M* can be found as follows:

$$M = (\mathbf{x}_1 - \mathbf{x}_2)_{\mathbf{w}} = (\mathbf{x}_1 - \mathbf{x}_3)_{\mathbf{w}},$$

where the subscript $_{\mathbf{w}}$ denotes the projection onto the weight vector **w** direction. The margin *M* can now be found by using support vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ as follows

$$D_1 = \|\mathbf{x}_1\|\cos(\alpha),\ D_2 = \|\mathbf{x}_2\|\cos(\beta) \text{ and } M = D_1 - D_2,$$

where α and β are the angles between **w** and $\mathbf{x}_1$ and between **w** and $\mathbf{x}_2$ respectively as given on page 4 e.g.,

$$\cos(\alpha) = \frac{\mathbf{x}_1^T\mathbf{w}}{\|\mathbf{x}_1^T\|\|\mathbf{w}\|}$$

Substituting cosines into the expression for *M* above results in

$$M = (\mathbf{x}_1^T\mathbf{w} - \mathbf{x}_2^T\mathbf{w}) / \|\mathbf{w}\|$$

and by using the fact that $\mathbf{x}_1$ and $\mathbf{x}_2$ are support vectors satisfying

$$y_j|\mathbf{w}^T\mathbf{x}_j + b| = 1,\ j = 1, 2, \text{ that is}$$

$$\mathbf{w}^T\mathbf{x}_1 + b = 1 \text{ and } \mathbf{w}^T\mathbf{x}_2 + b = -1$$

we finally obtain $M = \dfrac{2}{\|\mathbf{w}\|}$!!!!!!

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 # of training data and N_{SV} stands for a # of SV. See the next slide about the meaning of the inequality above!

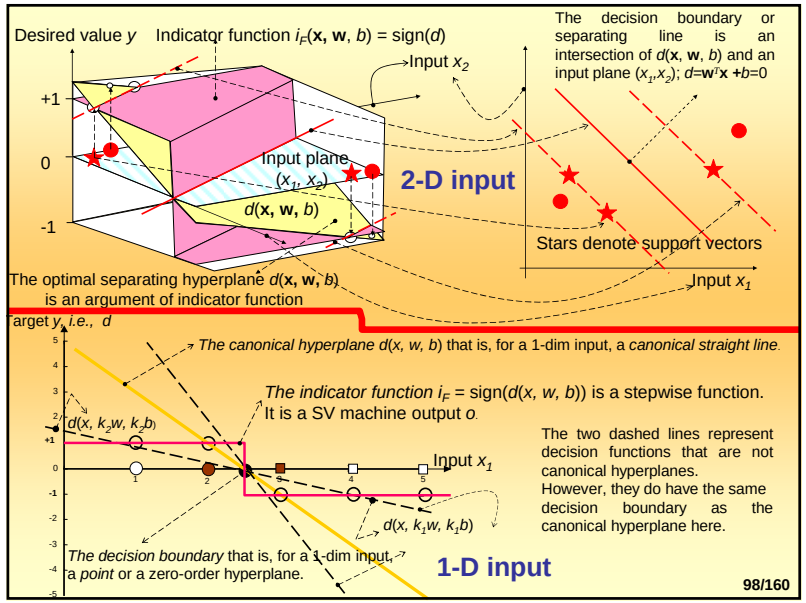
Note that maximization of M means a minimization of $\|\mathbf{w}\|$.

Minimization of a norm of a hyperplane normal weight vector $\|\mathbf{w}\| = \sqrt{\mathbf{w}^T\mathbf{w}} = \sqrt{w_1^2 + w_2^2 + \ldots + w_n^2}$ 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

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

and this leads to a maximization of a margin M.



Thus the problem to solve is:

minimize

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

subject to constraints

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

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!

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

subject to constraints

Correct classification!

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

Note that # of constraining **in**equalities = # of training data *l*

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 $f(\mathbf{w})$

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

LINEAR PROGRAMMING problem: when $f(\mathbf{w})$ and $g(\mathbf{w})$ are linear and w_i's > 0

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

QUADRATIC PROGRAMMING problem $f(\mathbf{w})$ quadratic, $g(\mathbf{w})$ is linear.

NONLINEAR PROGRAMMING problem, $f(\mathbf{w})$ **and** $g(\mathbf{w})$ 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* $L(\mathbf{w}, b, \alpha)$ 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 $\mathbf{w}$ 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 $\mathbf{w}$ 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 *standard quadratic optimization problem* 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

$$\mathbf{y}^T\boldsymbol{\alpha}=0, \quad \longrightarrow \text{Note that there are 1 equality constraint here}$$

$$\boldsymbol{\alpha}\geq \mathbf{0}, \quad \longrightarrow \text{Note that there are } l \text{ inequality constraints here}$$

where, $\mathbf{H}$ 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,

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

All Support Vectors

For b, **we use only FREE, i.e., unbounded, SVecs** for which

$$0 < \alpha_i < C$$

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

Story about C comes in few slides!!!

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)

Remember this scalar product

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

Both the beauty and the power of working with SVMs can be seen below too

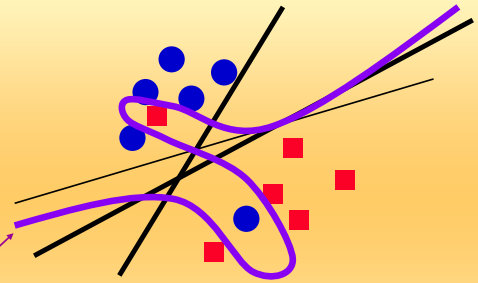
Once the support vectors have been found, we can calculate the bound on the expected probability of committing an error on a test example as follows

$$E_n\,[P(\text{error})] \leq \frac{E\,[\text{number of support vectors}]}{n}, \qquad (2.20)$$

where E_n denotes expectation over all training data sets of size n. Note how easy it is to estimate this bound that is independent of the dimensionality of the input space. Therefore, an SV machine having a small number of support vectors will have good generalization ability even in a very high-dimensional space.

My Springer book, page 30

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

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

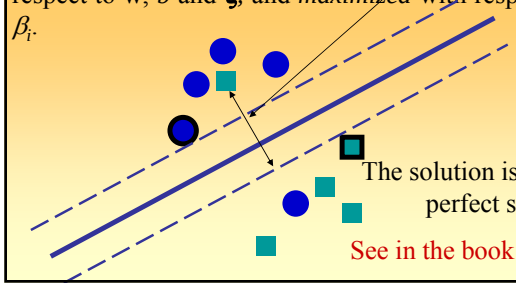
$$\frac{1}{2}\mathbf{w}^T\mathbf{w} + C(distance\ of\ \ error\ points\ to\ their\ correct\ side)$$

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!

For overlapping classes dual problem is formulated as

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

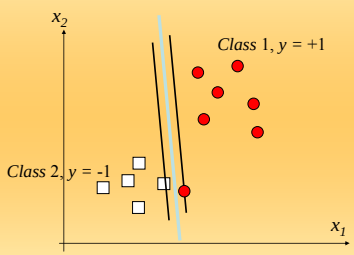
s.t. $0 \leq \alpha_i \leq C$ for $i = 1, \ldots, N$

$$\sum_{i=1}^{N}\alpha_i y_i = \alpha^T\mathbf{y} = 0$$

This C is the NOVELTY in respect to the hard margin classifier

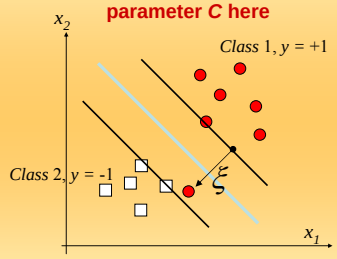
See in my Springer book the details of the solution!

Hard vs. Soft Margin SVMs
an example on robustness



Hard Margin SVM
leads to small margin

Use of penalty parameter *C* here



Soft Margin SVM
leads to large margin

QP setting of a LINEAR SVM learning problem:

HARD MARGIN:

PRIMAL: minimize $J = \mathbf{w}^T \mathbf{w} = \| \mathbf{w} \|^2$, s.t. $y_i[\mathbf{w}^T \mathbf{x}_i + b] \geq 1$!

DUAL: minimize $\sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j \mathbf{x}_i^T \mathbf{x}_j$ s.t. $\alpha_i \geq 0$, $\sum_{i=1}^{l} \alpha_i y_i = 0$

SOFT MARGIN:

DUAL: minimize $\sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j \mathbf{x}_i^T \mathbf{x}_j$ s.t. $C \geq \alpha_i \geq 0$, $\sum_{i=1}^{l} \alpha_i y_i = 0$

Learning is expressed **in terms of training data** and it depends *only* on **the scalar products of input patterns** ($\mathbf{x}_i^T \mathbf{x}_j$).

Comments: Solving primal results in the same weight vector **w** as in the dual solution, but 'primal' **w** is composed of all training data. Primal does not select relevant points - support vectors (i.e., it does not compress the information as the dual does). $\alpha_i > 0$ only for SVs, in a dual setting!!!

Just a fraction of relevant data (SVs) composes a decision hyperplane.

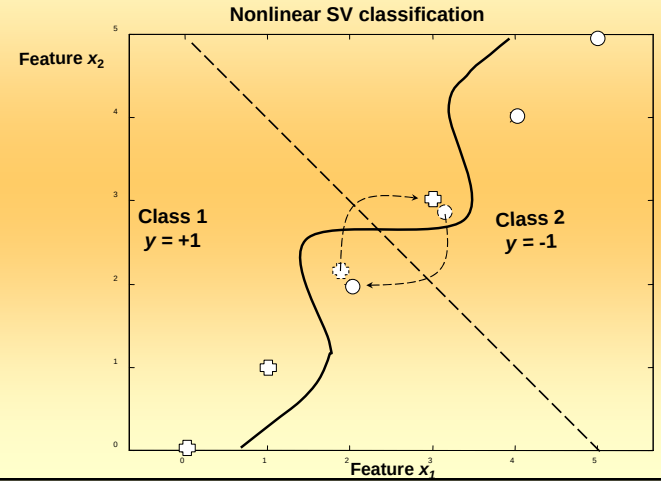
Here the LINEAR SVM models story ends!!!

What to do, and how to go about, when the true decision function (i.e., separation boundary) is NONLINEAR???

Remind, for example, that even if data are generated by normal (Gaussian) distribution but with different covariance matrices, the true decision function will be a quadratic function (see Example 1.10 on page 95, in chapter 1 of my The MIT book)

Hence, the hyperplanes cannot be the solutions when the decision boundaries are TRULY nonlinear, SAY AS IN THE CASE OF TWO GAUSSIAN CLASSES HAVING DIFFERENT COVARIANCE MATRICES or AS IN THE EXAMPLE SHOWN BELOW

Nonlinear SV classification



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.

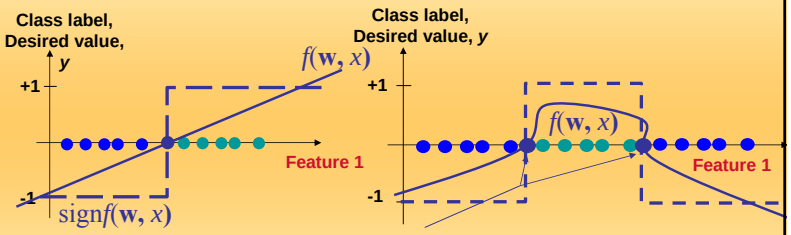
Check my Springer Verlag book

for all the derivations!!!

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:

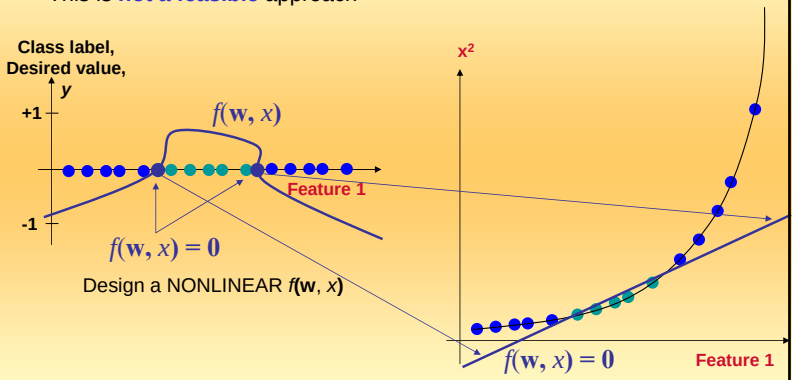
Example 1:

1) Solve in original *x domain*

This is **not a feasible** approach

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

This is **an RIGHT** approach



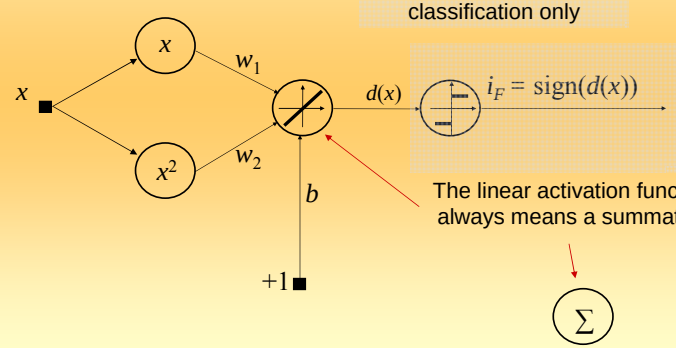
Design a NONLINEAR $f(\mathbf{w}, x)$

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

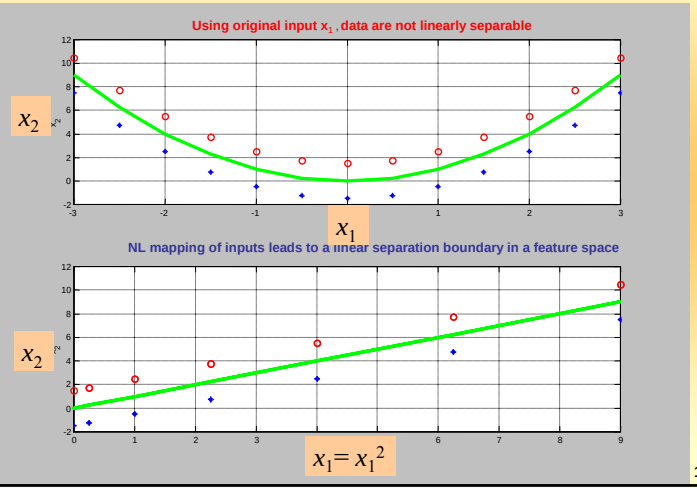
The thresholding shown here is needed for a classification only



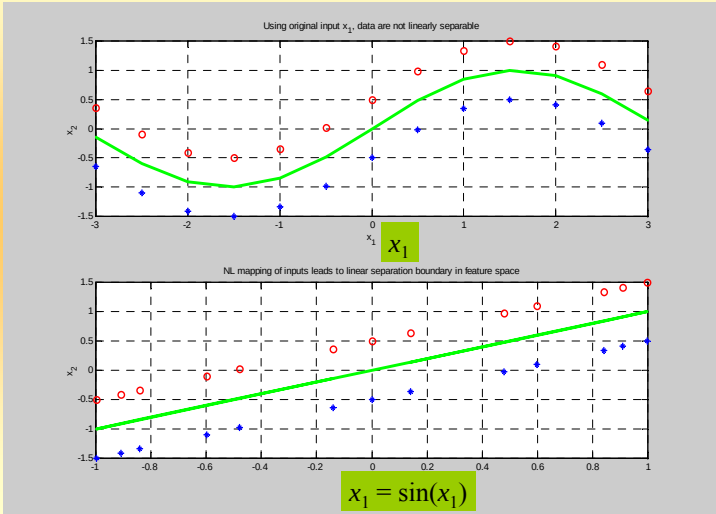
The linear activation function always means a summation

Why may the mapping of input space **X** (x_1, x_2) into feature space **F** $(f(x_1), f(x_2))$ be useful?

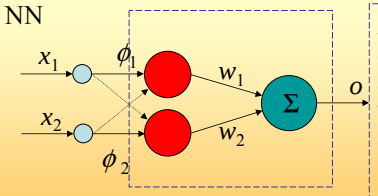
Example 2: Nonlinear (quadratic) separation boundary in **X** (x_1, x_2) is transformed into linear one in **F** (x_1^2, x_2) by (polynomial) mapping x_1 into x_1^2



Example 3: Nonlinear (sinusoidal) separation boundary in **X** (x_1, x_2) is transformed into linear one in **F** $(\sin(x_1), x_2)$ by (trigonometric) mapping x_1 into $\sin(x_1)$



In both cases the mapping performed can be represented as the following SVM i.e., NN



After the mapping **Φ** is chosen, the linear margin classifier, i.e., the SVM, is to be designed **in a feature space** with the hard, or soft learning algorithms presented so far!

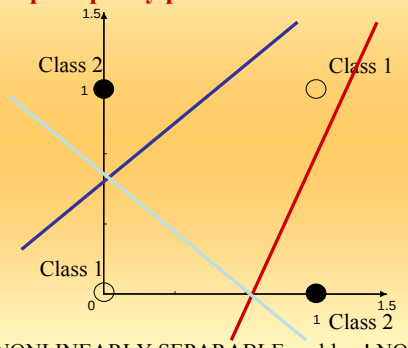
Despite the fact that NL mapping works nicely, there are two basic comments needed now:

a) We decided in advance which NL mapping to perform, **for we knew the nonlinearity.** Generally we do not know the very character of separation (hyper)surfaces and we will try to solve each problem with a few standard mappings (polynomial and RBF Gaussian ones primarily).

b) The dimension of a feature space in two previous examples is same as the one of the original input space. This is, however, not typical and we will usually map input space into much richer space (space of the much higher dimension, **possibly into space of infinite dimension(!)**

Solution of a problem, regarding both the nonlinear mapping and a dimensionality of the feature space (that is related to the number of neurons in a hidden layer) used, **is usually not unique.**

Consider the simplest parity problem - XOR one:

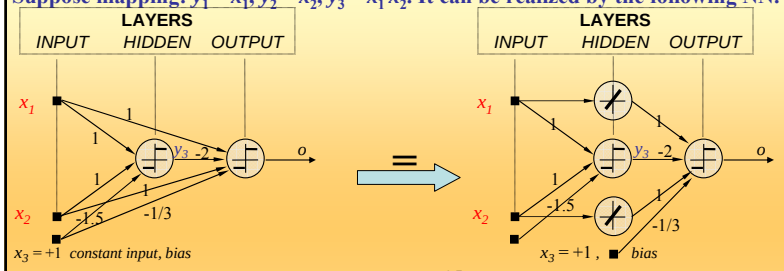


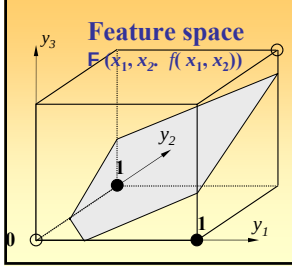
This is a classic NONLINEARLY SEPARABLE problem! NO linear separation line!

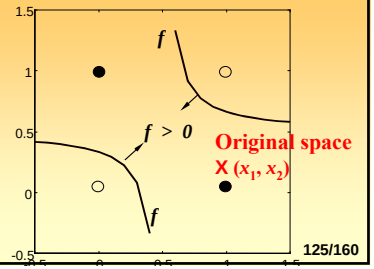
We show solutions by using both **a polynomial** and **an RBF** approach.
Other polynomial and RBF solutions, as well as other NL ones are possible, too!!!

XOR problem - polynomial solution: $f(\mathbf{x}) = x_1 + x_2 - 2\,x_1\,x_2 - 1/3$

Suppose mapping: $y_1 = x_1$, $y_2 = x_2$, $y_3 = x_1\,x_2$. It can be realized by the following NN.





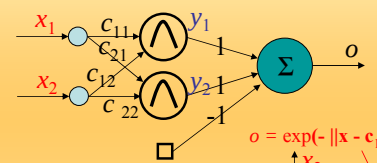


XOR problem - Gaussian RBF solution. We choose two Gaussians only, i.e., a dimension of a feature space is 2: $c_1 = [1\ 1]^T$ and $c_2 = [0\ 0]^T$.

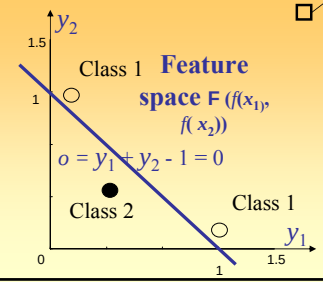
This is a following mapping: $\mathbf{y} = [\exp(-\|\mathbf{x} - \mathbf{c}_1\|^2)\quad \exp(-\|\mathbf{x} - \mathbf{c}_2\|^2)]^T$. Hence, **x -> y:**

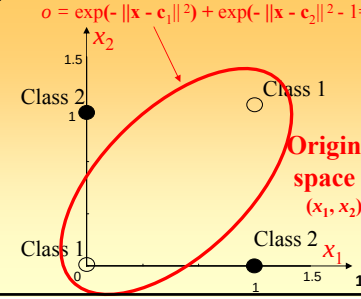
[0 0] -> [0.135 1], [1 1] -> [1 0.135], [0 1] -> [0.368 0.368], [0 1] -> [0.368 0.368].

The following NN, will produce linear separation boundary in a feature space and the NL one in the original space.



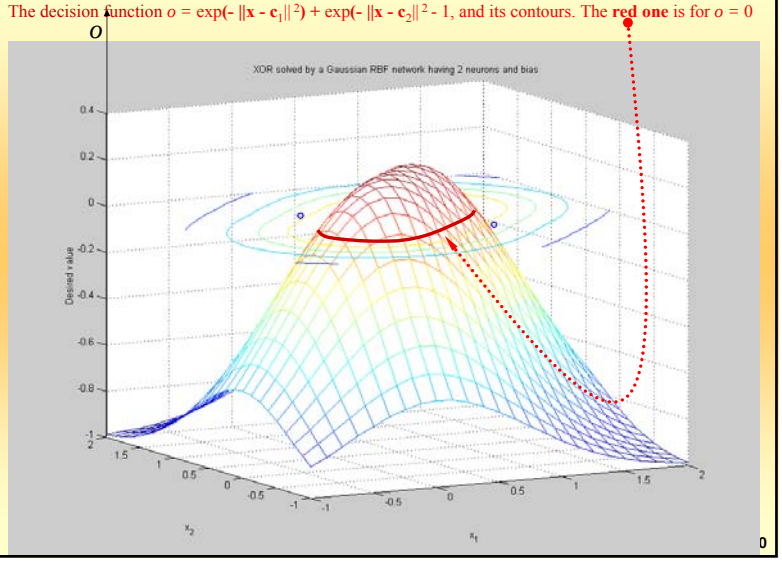
$o = \exp(-\|\mathbf{x} - \mathbf{c}_1\|^2) + \exp(-\|\mathbf{x} - \mathbf{c}_2\|^2 - 1 = 0$



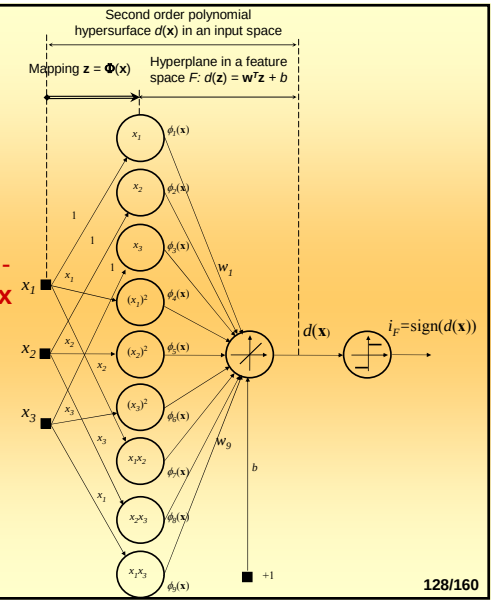


cont. XOR problem - Gaussian RBF solution;

The decision function $o = \exp(-\|\mathbf{x} - \mathbf{c}_1\|^2) + \exp(-\|\mathbf{x} - \mathbf{c}_2\|^2 - 1$, and its contours. The **red one** is for $o = 0$



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** = $\Phi(\mathbf{x})$.



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

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

solve a linear classification problem in this feature space

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

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

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

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

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

POLYNOMIAL KERNELS:
Let $x \in \Re^2$ i.e., $\mathbf{x} = [x_1 \;\; x_2]^T$, and if we choose $\boldsymbol{\Phi}(\mathbf{x}) = [x_1^2 \;\; \sqrt{2}x_1x_2 \;\; x_1^2]^T$ (i.e., there is an $\Re^2 \to \Re^3$ mapping), then the dot product

$$\begin{aligned}\boldsymbol{\Phi}^T(\mathbf{x}_i)\boldsymbol{\Phi}(\mathbf{x}_j) &= [x_{i1}^2 \;\; \sqrt{2}x_{i1}x_{i2} \;\; x_{i1}^2][x_{j1}^2 \;\; \sqrt{2}x_{j1}x_{j2} \;\; x_{j1}^2]^T \\ &= [x_{i1}^2x_{j1}^2 + 2x_{i1}x_{i2}x_{j1}x_{i2} + x_{i2}^2x_{j2}^2] = (\mathbf{x}_i^T\mathbf{x}_j)^2 = K(\mathbf{x}_i, \mathbf{x}_j), \text{or} \\ K(\mathbf{x}_i, \mathbf{x}_j) &= (\mathbf{x}_i^T\mathbf{x}_j)^2 = \boldsymbol{\Phi}^T(\mathbf{x}_i)\boldsymbol{\Phi}(\mathbf{x}_j)\end{aligned}$$

Note that in order to calculate the scalar product in a feature space $\boldsymbol{\Phi}^T(\mathbf{x}_i)\boldsymbol{\Phi}(\mathbf{x}_j)$ we do not need to perform the mapping $\boldsymbol{\Phi}(\mathbf{x}) = [x_1^2 \;\; \sqrt{2}x_1x_2 \;\; x_1^2]^T$ at all. Instead, we calculate this product directly in the input space by computing $(\mathbf{x}_i^T\mathbf{x}_j)^2$. This is very well known under the popular name of *the kernel trick*. Interestingly, note also that other mappings such as an

$\Re^2 \to \Re^3$ mapping given by $\boldsymbol{\Phi}(\mathbf{x}) = [x_1^2 - x_2^2 \;\; 2x_1x_2 \;\; x_1^2 + x_2^2]$, or an
$\Re^2 \to \Re^4$ mapping given by $\boldsymbol{\Phi}(\mathbf{x}) = [x_1^2 \;\; x_1x_2 \;\; x_1x_2 \;\; x_2^2]$

also accomplish the same task as $(\mathbf{x}_i^T\mathbf{x}_j)^2$.

Now, assume the following mapping

$$\boldsymbol{\Phi}(\mathbf{x}) = [1 \;\; \sqrt{2}x_1 \;\; \sqrt{2}x_2 \;\; \sqrt{2}x_1x_2 \;\; x_1^2 \;\; x_2^2],$$

i.e., there is an $\Re^2 \to \Re^5$ mapping plus bias term as the constant 6th dimension's value. Then the dot product in a feature space $\mathcal{S}$ is given as

$$\begin{aligned}\boldsymbol{\Phi}^T(\mathbf{x}_i)\boldsymbol{\Phi}(\mathbf{x}_j) &= 1 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2} + 2x_{i1}x_{i2}x_{j1}x_{i2} + x_{i1}^2x_{j1}^2 + x_{i2}^2x_{j2}^2 \\ &= 1 + 2(\mathbf{x}_i^T\mathbf{x}_j) + (\mathbf{x}_i^T\mathbf{x}_j)^2 = (\mathbf{x}_i^T\mathbf{x}_j + 1)^2 = K(\mathbf{x}_i, \mathbf{x}_j), \text{ or} \\ K(\mathbf{x}_i, \mathbf{x}_j) &= (\mathbf{x}_i^T\mathbf{x}_j + 1)^2 = \boldsymbol{\Phi}^T(\mathbf{x}_i)\boldsymbol{\Phi}(\mathbf{x}_j)\end{aligned}$$

Thus, the last mapping leads to the second order *complete* polynomial.

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

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

$$L(\alpha) = -0.5\boldsymbol{\alpha}'\mathbf{H}\,\boldsymbol{\alpha} + \mathbf{1}'\boldsymbol{\alpha},$$

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

$$\mathbf{H} = \mathbf{Y'Y.*K}$$

and the constraints are

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

In a more general case, because of a 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.

In the case of NL SVMs we never, or only rarely, calcu-late a weight vector **w**. Solving NL SVM is performed in the so-called feature space which is of a very high, including infinite, dimension. In fact we don't need **w!!!** Instead we use *alphas* as follows (in S. Abe's book):

$$b = \frac{1}{|U|} \sum_{j \in U} \left(y_j - \sum_{i \in S} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_j) \right)$$

$$D(\mathbf{x}) = \sum_{i \in S} \alpha_i \, y_i \, K(\mathbf{x}_i, \mathbf{x}) + b$$

where, U is a set of all *free* i.e., ***u**nbounded* SVecs, and S is a set of ***all*** SVecs

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! → 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, (1996), 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) $f(\mathbf{x})$.

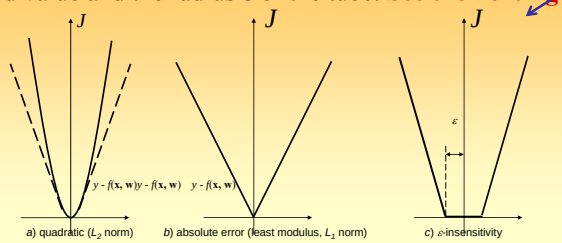
A training data set $D = \{[\mathbf{x}(i), y(i)] \in \Re^n \times \Re, i = 1,...,l\}$ consists of l pairs $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_l, y_l)$, where the inputs $\mathbf{x}$ are n-dimensional vectors $\mathbf{x} \in \Re^n$ and system responses $y \in \Re$, are continuous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

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

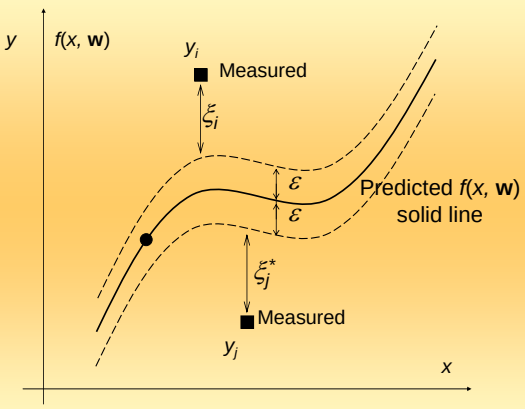
Vapnik introduced a more general error (loss) function - the so-called **ε-insensitivity loss function**

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

Thus, the loss is equal to 0 if the difference between the predicted $f(\mathbf{x}, \mathbf{w})$ and the measured value is less than ε. Vapnik's ε-insensitivity loss function **defines an ε tube** around $f(\mathbf{x}, \mathbf{w})$. 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

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

and the constraints are,

$y_i - \mathbf{w}^T\mathbf{x}_i - b \le \varepsilon + \xi_i$, $\quad i = 1, l$,
$\mathbf{w}^T\mathbf{x}_i + b - y_i \le \varepsilon + \xi_i^*$, $\quad i = 1, l$,
$\xi_i \ge 0$ $\quad i = 1, l$,
$\xi_i^* \ge 0$ $\quad i = 1, l$,

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

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

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

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 $L_d(\boldsymbol{\alpha}, \boldsymbol{\alpha}^*)$ is expressed in terms of LaGrange multipliers $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}^*$ 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**,

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

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

A mapping $\boldsymbol{\Phi}(\mathbf{x})$ 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.

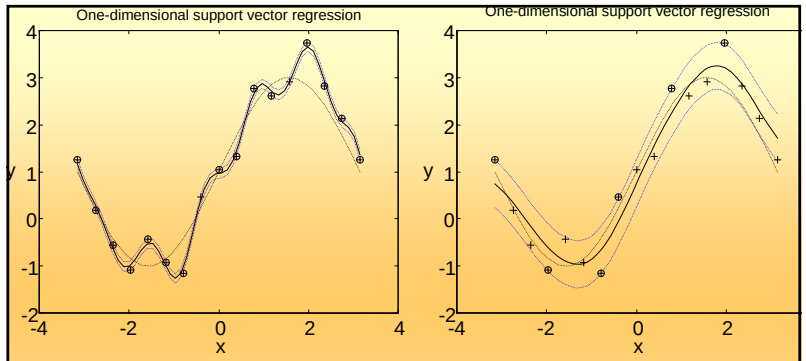
Step 5

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!

SVMs Linear Classification Learning 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. $0 \le \alpha_i \le C$ for $i = 1, \ldots, N$

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

SVMs Linear Regression Learning 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. $0 \le \alpha_i, \alpha_i^* \le C$ for $i = 1, \ldots, N$

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

Final solution is:

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

classification

regression

Let us conclude the presentation of SVMs by summarizing 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.**

Let us conclude the part on 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) functions, 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.**

Some additions

I've been talking mostly about SVMs, but what are the alternatives?

Basic, the most popular and powerful, ones would be:

- **The least squares classifiers**, (Gauss and Legendre, ~ 200 years ago, today **FFT** and **JPEG** are still using it),
- **Linear discriminant analysis**, LDA (R.A. Fisher, 1936), for multivariate normal distributions; it uses hyperplanes as decision functions. A generalization of LDA is
- **Quadratic discriminant analysis**, which allows quadratic decision functions. Both methods are still used by many practitioners often with good success.
- **k-nearest-neighbor, KNN,** introduced in 1951; see Fox and Hodges (1951, 1952). Many followers. It's still in heavy use. It was the first method for which universal consistency was established; see Stone (1977)

- **Cluster analysis** is an **UNsupervised** approach to recognize clusters in unlabeled data. Check the books by Hartigan (1975) and Kaufman and Rousseau (2005) for an introduction to cluster analysis techniques. K-means cluster analysis.
- **Parametric logistic regression** proposed by D. R. Cox to model binomial distributed outputs; see Cox and Snell (1989). This method is based on linear decision functions but **does not make specific assumptions on the distribution of the inputs**. Parametric logistic regression is **a special case of generalized linear**, see McCullagh and Nelder (1989). Hastie and Tibshirani (1990) proposed a semi-parametric generalization called **generalized additive models** where the inputs may influence the outputs in an additive but not necessarily linear manner. The **lasso** (Tibshirani, 1996) is a method for regularizing a least squares regression. It minimizes the usual sum of squared errors, with a bound on the sum of the absolute values of the coefficients.
- Other 'classic' methods for classification and regression are **trees**, Breiman et al. (1984). Trees often produce not only accurate results but are also able to uncover the predictive structure of the problem.
- **Neural networks** are **non-linear statistical data modeling tools** that can be used to model complex relationships between inputs and outputs or to find patterns in data sets. The motivation for neural networks, which were very popular in the 1990s, goes back to McCullogh and Pitts (1943) and Rosenblatt (1962). We refer also to Bishop (1996), Anthony and Bartlett (1999), and Vidyasagar (2002).

- There also exist various other **kernel-based methods**. For **wavelets**, we refer to Daubechies (1991), and for **splines** to Wahba (1990). Recent developments for kernel-based methods in the context of **SVMs** are also described by Cristianini and Shawe-Taylor (2000), Schoelkopf and Smola (2002), and Shawe-Taylor and Cristianini (2004).
- **Boosting algorithms** are based on an adaptive aggregation to construct from a set of weak learners a strong learner; see Schapire (1990), Freund (1995), and Freund and Schapire (1997). Finally, the books by Hastie et al. (2001, 2009), Duda et al. (2001), and Bishop (2006) give a broad overview of various techniques used in statistical machine learning, whereas both Devroye et al. (1996) and Gyoerfi et al. (2002) treat several classification and regression methods in a mathematically more rigorous way.

Now, some basics of a

Bias – Variance - Dilemma!

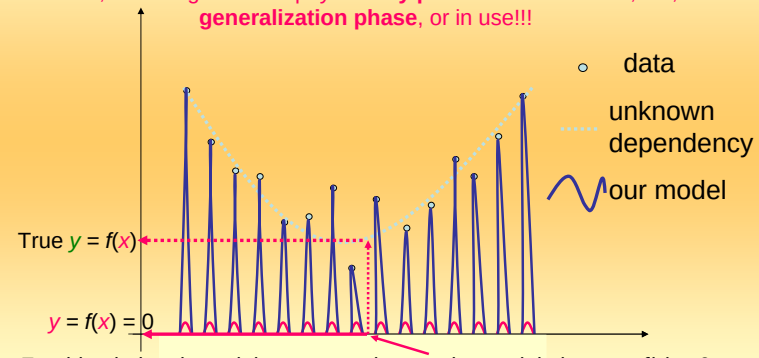
It is the must piece of the knowledge in order to get an idea of the relationship between the data, models and errors!

It will be **intuitive**, without math or any equation and it will serve for warming up! Check Kecman's book (there are many others better and more specialized too) if you like math.

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!!!



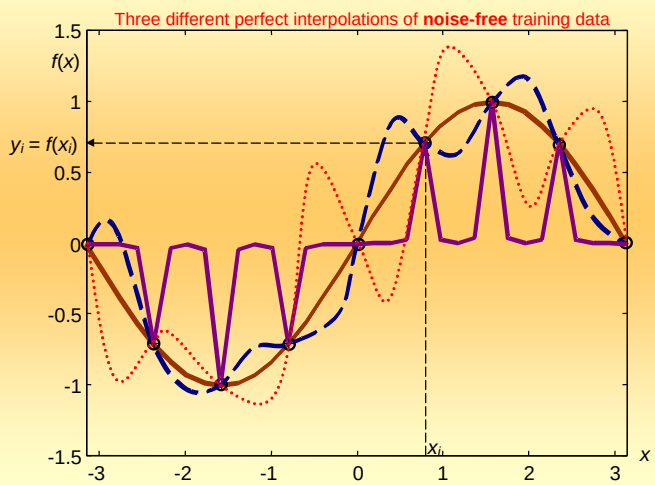
For this, during the training unseen, input x the model gives $y = f(x) = 0$

This is (deliberately chosen) extremely bad modeling, **but real!**

The same or similar phenomena will be present in the high dimensional cases, too!

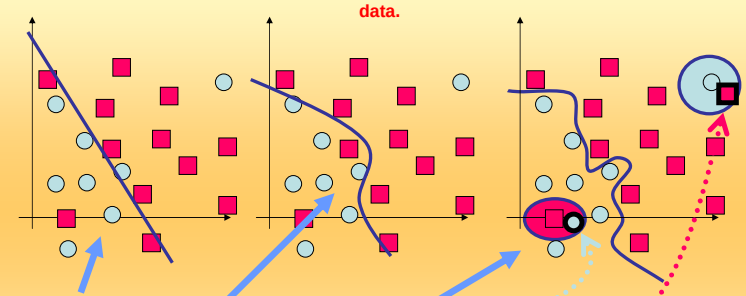
One more example showing **the perfect training results,** but **very bad generalization ones.**

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

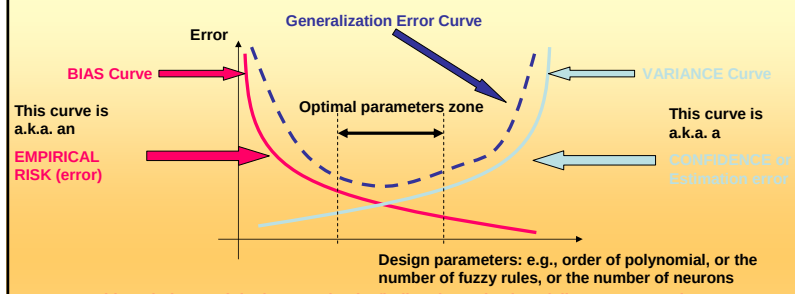
Or, this one

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!



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**

Bias & Variance

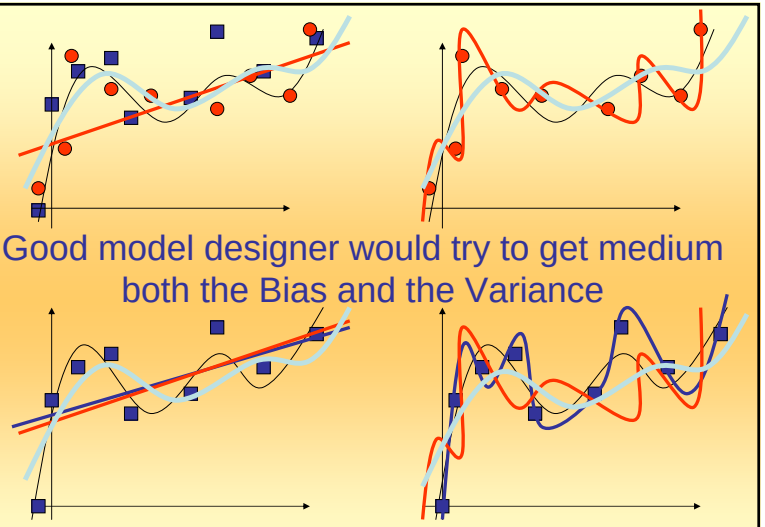
In modeling an **unknown dependency (regression or discrimination function)**, without knowledge of its mathematical form (**target space**), our **models (functions from hypothesis space**) produce **approximating functions**, which may be incapable of representing the **target function** behavior.

A difference between the model output and unknown target function is called **the bias.**

When there are not sufficient data, (or even if there appears to be sufficient representative data, **noise** contamination can still contribute that) **the sample of data** that is **available for training** *may not be representative of* ***average*** *data generated by the target function*.

Consequently, there may be a difference between a network output **for a particular data set**, and network function output **for the average of all data sets** produced by the target function.

The *square* of this difference is called **the variance.**



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

High Bias - Low Variance

Low Bias - High Variance