The Three Pillars of Machine Learning

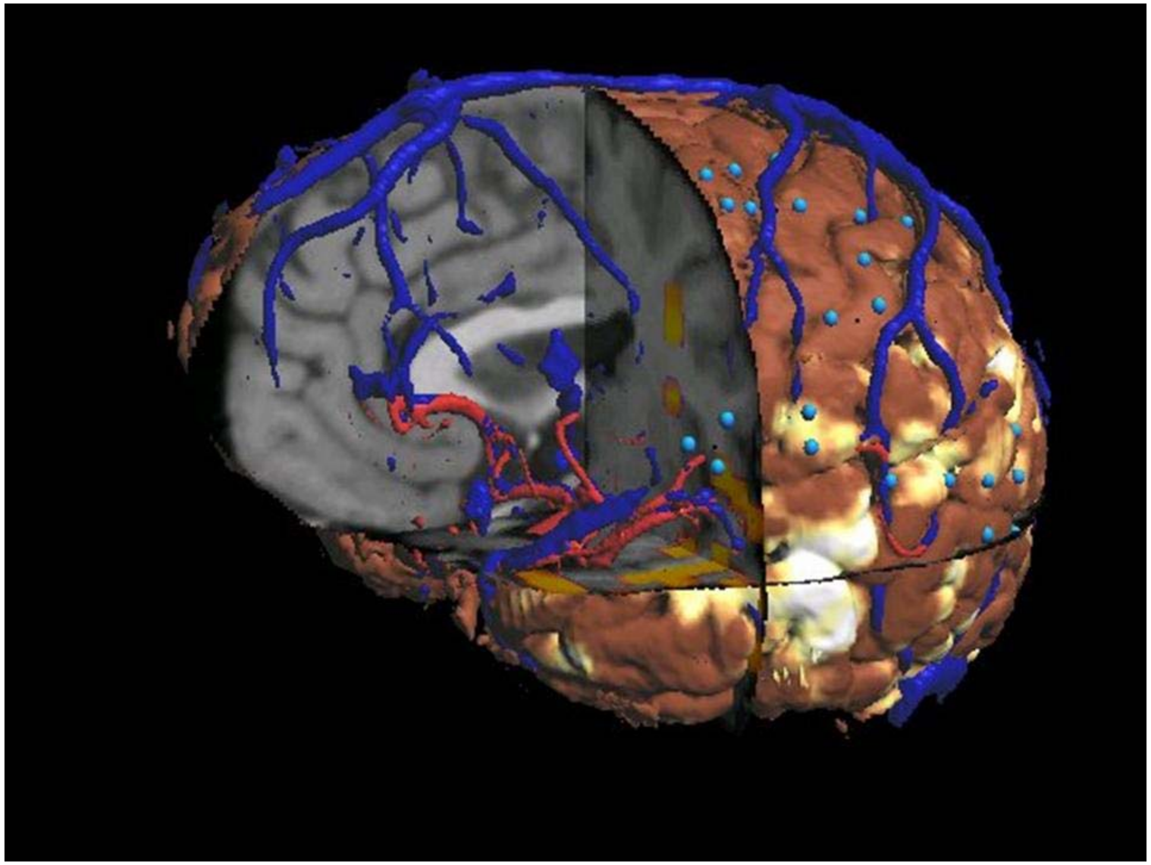
1. Learning Theory

The key to neural network and machine learning: Learning theory

The role of learning theory has grown a great deal in:

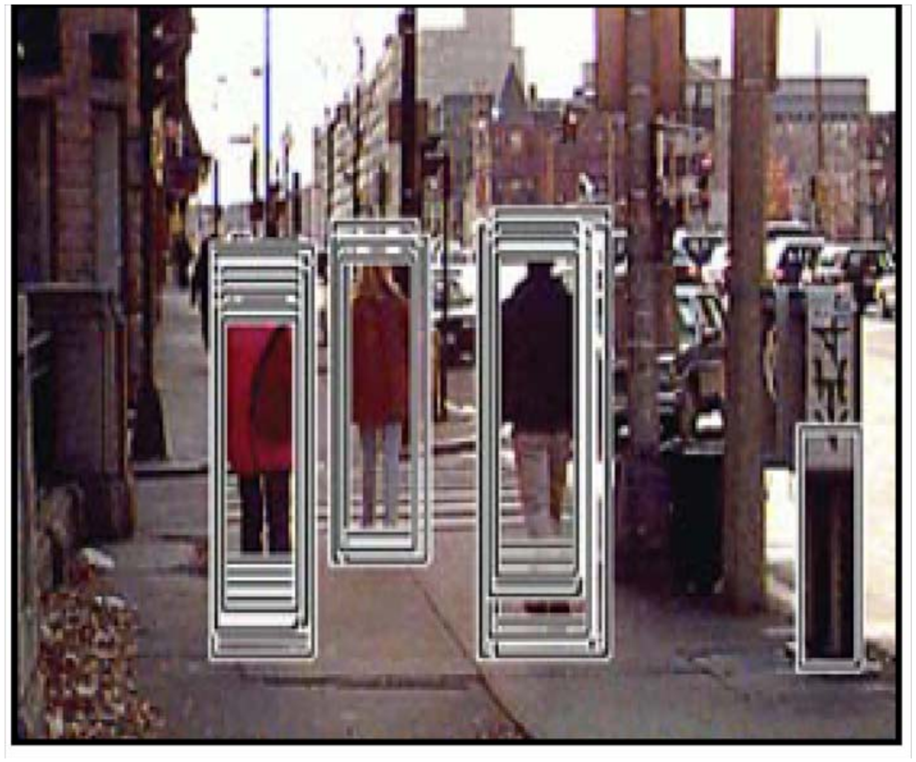
- Mathematics
- Statistics

- Finance
- Computational Biology
- Neurosciences, e.g., theory of plasticity, workings of visual cortex



University of Washington

- Computer science, e.g., vision theory, graphics, speech synthesis

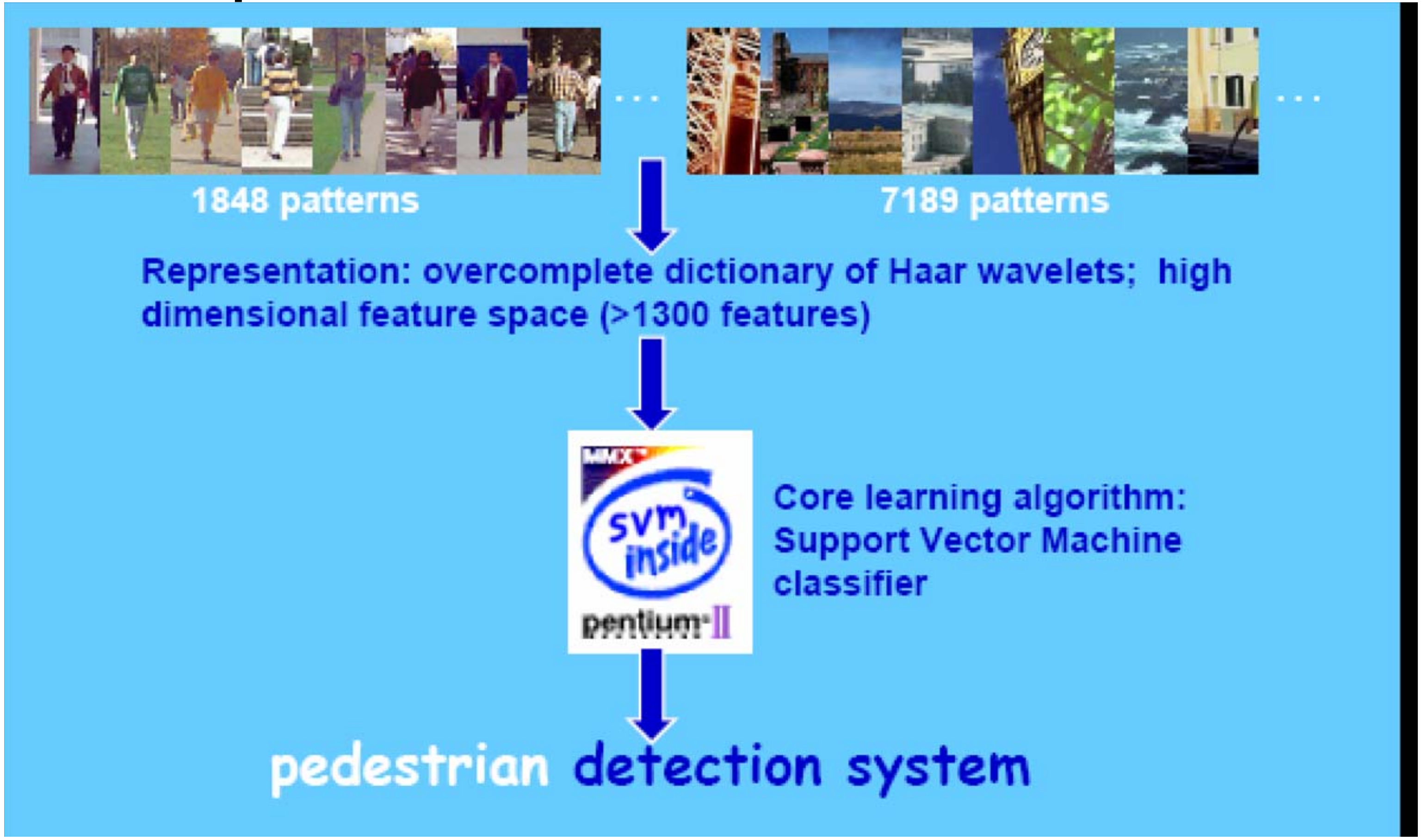


T. Poggio/MIT

Face identification:

MIT

People classification or detection:



...
1848 patterns
...
7189 patterns
Representation: overcomplete dictionary of Haar wavelets; high dimensional feature space (>1300 features)
MMX
svm inside
pentium II
Core learning algorithm: Support Vector Machine classifier
pedestrian detection system

Poggio/MIT

The Problem

What is the theory behind such learning algorithms?

2. The problem: Learning theory

Given an unknown function $f(\mathbf{x})$ whose graph is unknown, learn the function from examples.

Example 1: $\mathbf{x} = (x_1, x_2, \ldots, x_k)$
is retinal activation pattern
(i.e., $x_1 =$ activation level of retinal neuron 1, etc.), and
$y = f(\mathbf{x}) > 0$ if the retinal pattern is a chair;
$y = f(\mathbf{x}) < 0$ otherwise.

The Problem

[Thus: $f(\mathbf{x})$ encodes concept of a chair]

The Problem

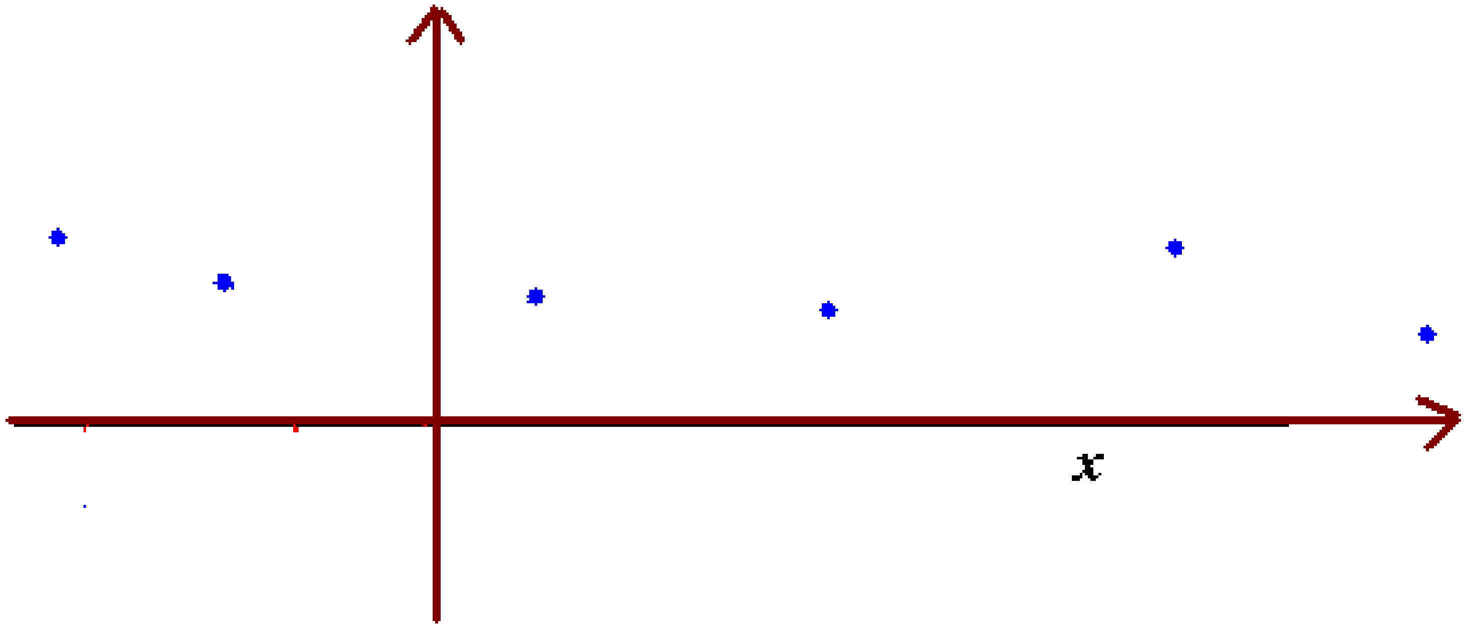
Given: examples of chairs (and non-chairs): $\mathbf{x}_1, \mathbf{x}_2$ etc., together with proper outputs $y_1, y_2,$ etc. This is the *training information*.

The Problem

Goal: Give best possible estimate of the unknown function f, i.e., try to learn the concept f from the above examples.

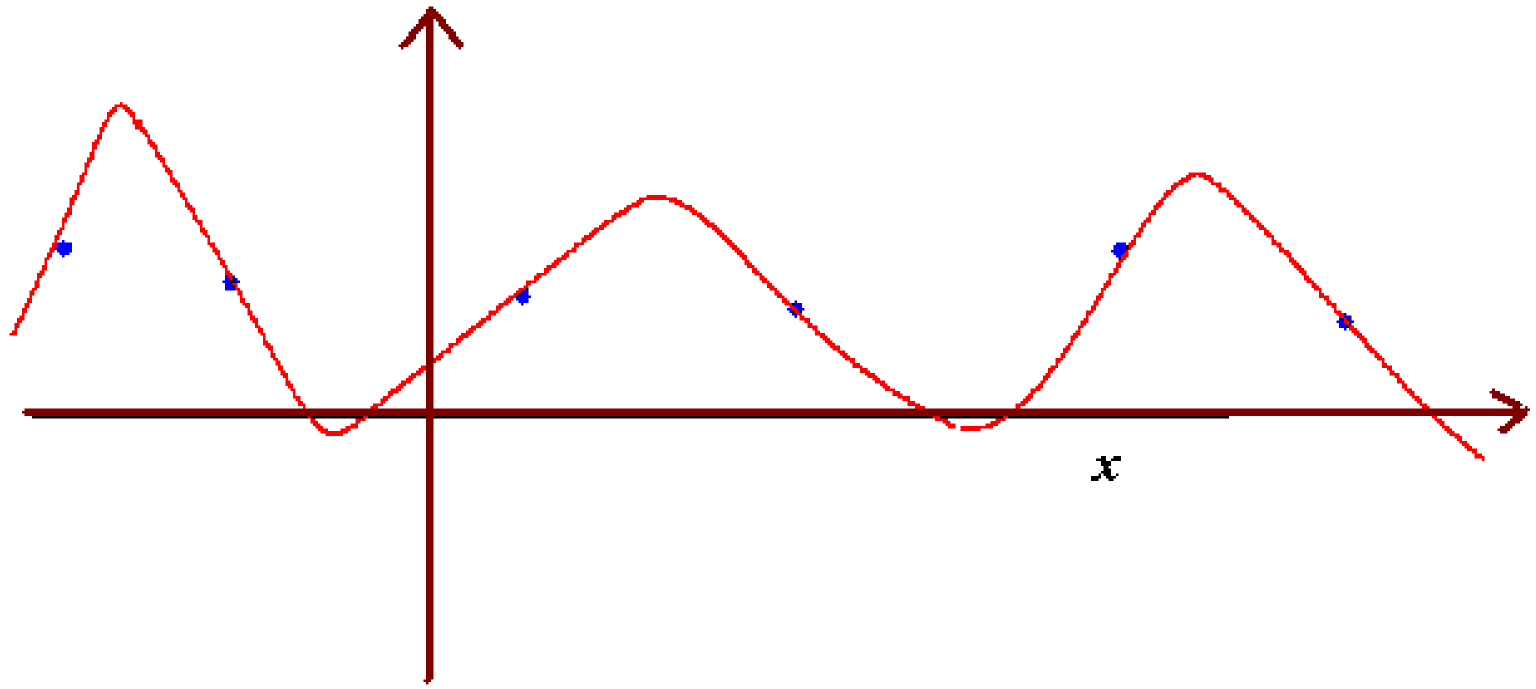
But: given a few pieces of information about the graph of f not sufficient: which is the "right" $f(\mathbf{x})$ given the data points $(\mathbf{x}, y)$ below?

Example 2: (Here **x** is just 1 dimensional)



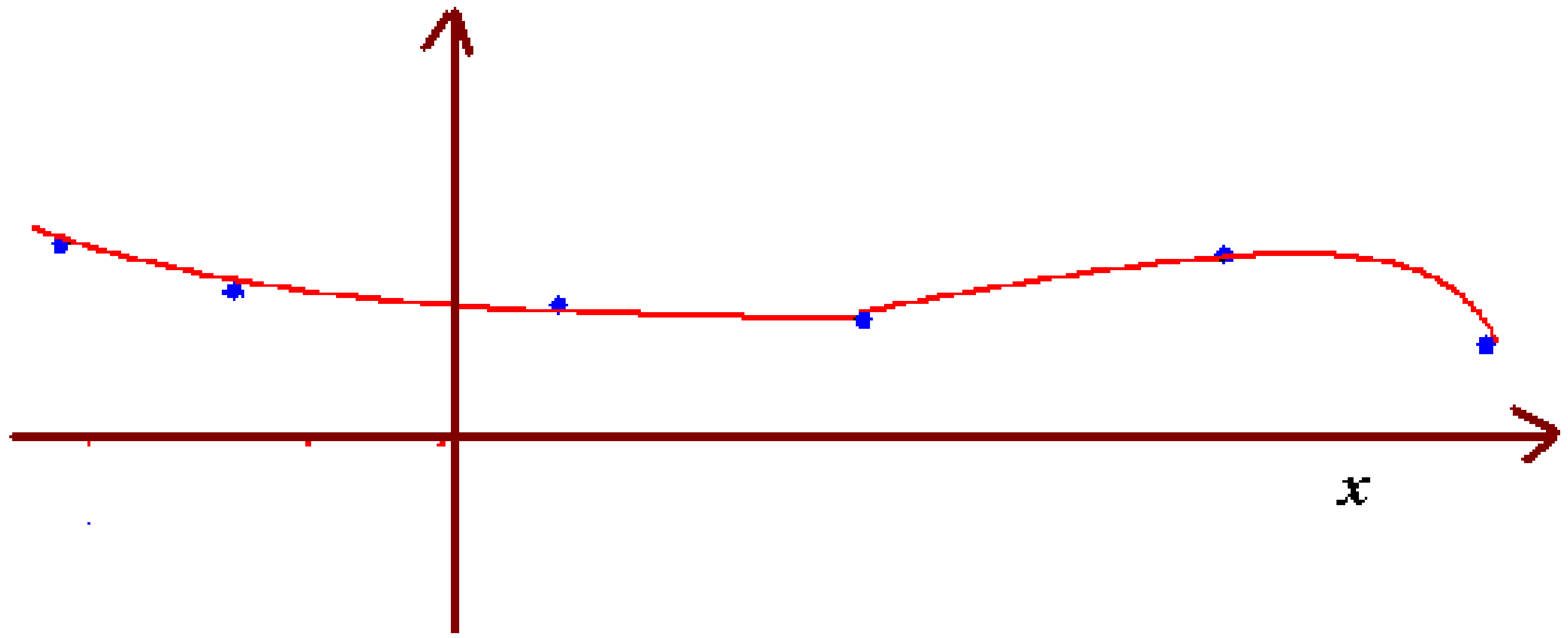
(a)

The Problem



The Problem

(b)



[How to decide?]

The Problem

Note: there is no unique solution for $f(\mathbf{x})$ - finding f is an *ill-posed problem.*

Hint: *a good machine will choose the simplest* $f(x)$ *- this is Occam's razor.*

MACHINE LEARNING: BASICS

1. Motivation: machine learning for high dimensional problems

Example in Computational Biology:

RNA-Seq Machine

Computational biology

Process: for each subject tissue sample s, obtain *feature vector*

$$\Phi(s) = \mathbf{x} = (x_1, \ldots, x_{20{,}000})$$

$=$ vector of gene expression levels

E.G., $x_1 =$ expression level of gene 1, etc.

Can we classify tissues this way?

If this is an ovarian cancer tissue sample:

Questions:

(a) What type of cancer is it?

(b) What is prognosis if untreated?

(c) What will be the reaction to standard chemotherapies?

Goals:

1. Differentiate two different but similar cancers.

2. Differentiate different cancer prognoses or potential therapies

3. Understand genetic origins and pathways of the cancer

Computational biology

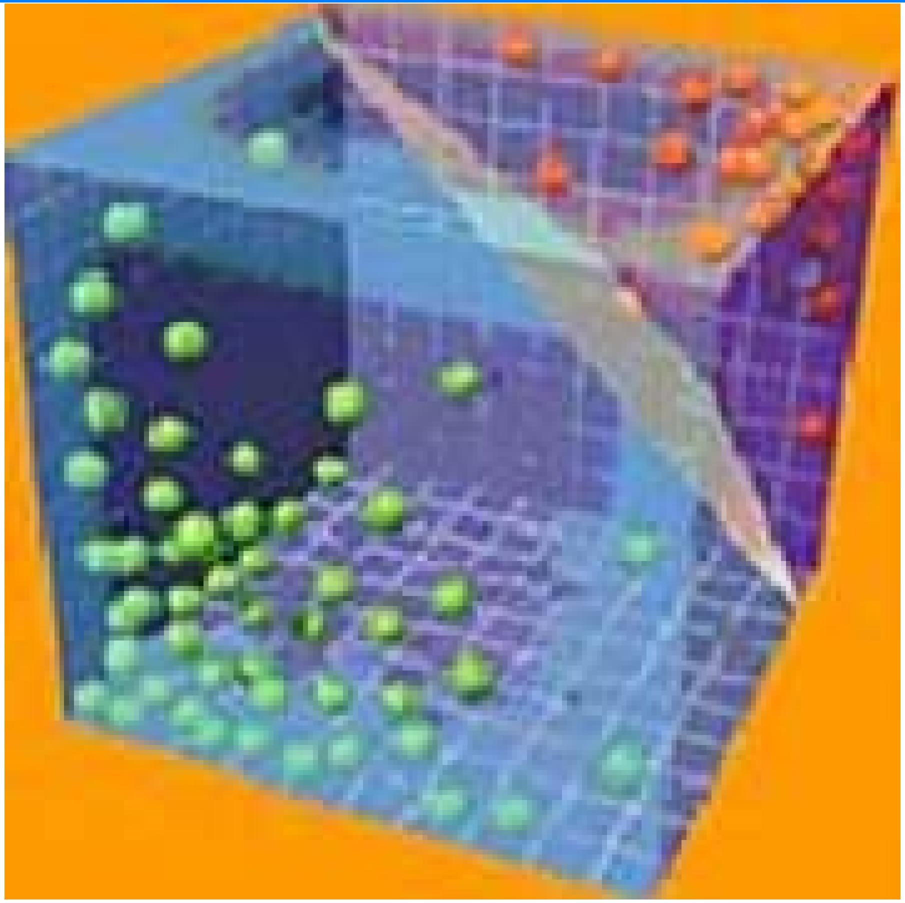
Basic difficulties: few samples to train with (e.g., 30-200); high dimension (e.g., 5,000 - 100,000).

Curse of dimensionality - too few samples and too many parameters (dimensions) to fit them.

Tool: Support vector machine (SVM)

Procedure: look at feature space F in which $\Phi(s)$ lives, and separate examples of one and the other cancer with a hyperplane:

Computational biology



e.g. Red vs. Green points represent tissues that were responsive (green) vs.

unresponsive (red) to a particular therapy T.

Computational biology

Train machine: take $n = 50$ subjects with different responses to therapy T, and locate their feature vectors in the space F, labeling them red (unresponsive) or green (responsive).

Find separating hyperplane, and use this plane to separate feature vectors of future subjects into 'responsive' and 'unresponsive'.

Computational biology

There are a number of other *machine learning methods* (often with non-linear separating boundaries) that can discriminate (classify) tissue feature vectors $\Phi(s)$ this way, with respect to prognosis, response to therapies, metastatic/non-metastatic cancer etc.

Such cancer data has often been very reliable and obtained from TCGA (the Cancer Genome Atlas) and its iterates.

2. The principle: more is more

Past beliefs: too many variables spoil the statistics; < 50 variables was typical requirement

Present: more is better

Machine learning allows massive integration of information about any object (e.g. a tissue sample):

Computational biology

On a gene level: for any gene in a tissue sample we get a series of numbers describing it, from basic measurements and databases of:

- protein-protein interactions
- co-expression (when genes activate together)
- gene ontology relationships (keywords referring to given genes in the literature)
- pathway correlations (when genes appear in the same biological pathways)

Computational biology

- epigenetic information (methylation, phosphorylation levels of genes in DNA)

Machine Learning and Suport Vector Machines (SVM)

1. Machine learning: SVM

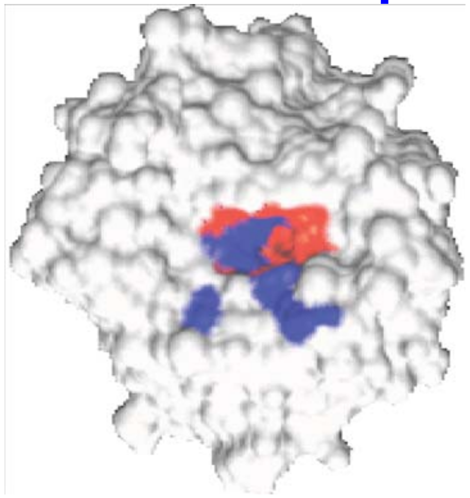
Support vector machine (SVM) is one of the most well-known machine learning tools.

Some applications of SVM in computational biology:

SVM

- **Protein binding prediction**

Will protein A bind to protein B?



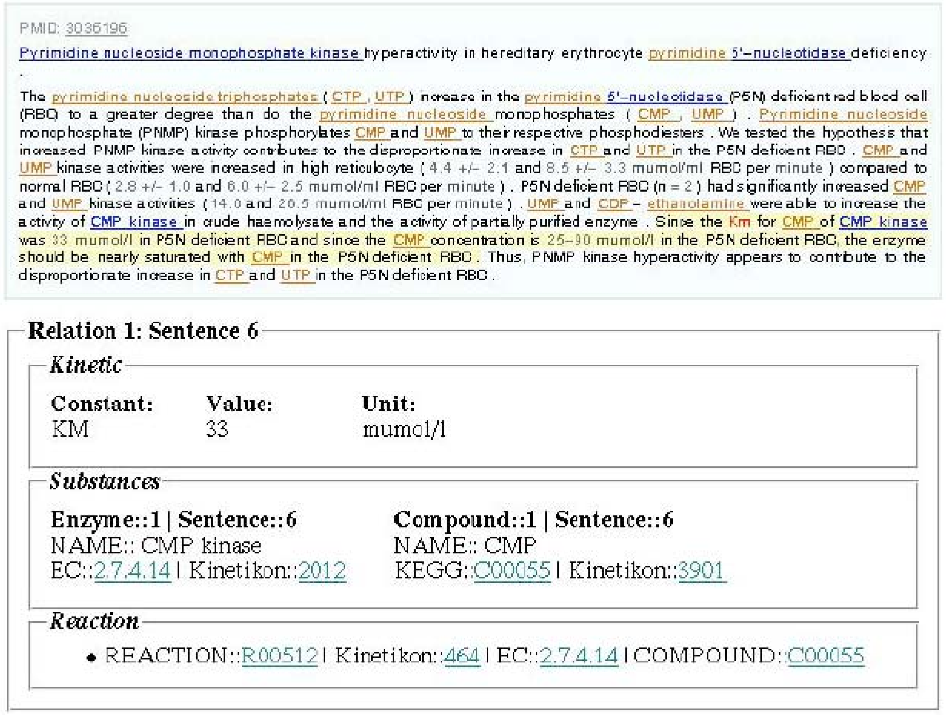
http://3dsig.weizmann.ac.il/usersfiles/3dsig/abstracts/2004/13.html

SVM

- **Text and topic mining**

Does this paper discuss transcription factor binding to DNA?

SVM



PMID: 3036196

Pyrimidine nucleoside monophosphate kinase hyperactivity in hereditary erythrocyte pyrimidine 5'-nucleotidase deficiency
.

The pyrimidine nucleoside triphosphates (CTP , UTP) increase in the pyrimidine 5'-nucleotidase (P5N) deficient red blood cell (RBC) to a greater degree than do the pyrimidine nucleoside monophosphates (CMP , UMP) . Pyrimidine nucleoside monophosphate (PNMP) kinase phosphorylates CMP and UMP to their respective phosphodiesters . We tested the hypothesis that increased PNMP kinase activity contributes to the disproportionate increase in CTP and UTP in the P5N deficient RBC . CMP and UMP kinase activities were increased in high reticulocyte (4.4 +/- 2.1 and 8.5 +/- 3.3 mumol/ml RBC per minute) compared to normal RBC (2.8 +/- 1.0 and 6.0 +/- 2.5 mumol/ml RBC per minute) . P5N deficient RBC (n = 2) had significantly increased CMP and UMP kinase activities (14.0 and 20.5 mumol/ml RBC per minute) . UMP and CDP - ethanolamine were able to increase the activity of CMP kinase in crude haemolysate and the activity of partially purified enzyme . Since the Km for CMP of CMP kinase was 33 mumol/l in P5N deficient RBC and since the CMP concentration is 25-90 mumol/l in the P5N deficient RBC, the enzyme should be nearly saturated with CMP in the P5N deficient RBC . Thus, PNMP kinase hyperactivity appears to contribute to the disproportionate increase in CTP and UTP in the P5N deficient RBC .

Relation 1: Sentence 6

Kinetic

Constant:	**Value:**	**Unit:**
KM	33	mumol/l

Substances

Enzyme::1 | Sentence::6	**Compound::1 | Sentence::6**
NAME:: CMP kinase	NAME:: CMP
EC::2.7.4.14 | Kinetikon::2012	KEGG::C00055 | Kinetikon::3901

Reaction

- REACTION::R00512 | Kinetikon::464 | EC::2.7.4.14 | COMPOUND::C00055

http://3dsig.weizmann.ac.il/3dsig/2004/abstracts/allabs.html
http://www.informatik.hu-berlin.de/forschung/gebiete/wbi/research/projects/textmining/kmeddbx.jpg

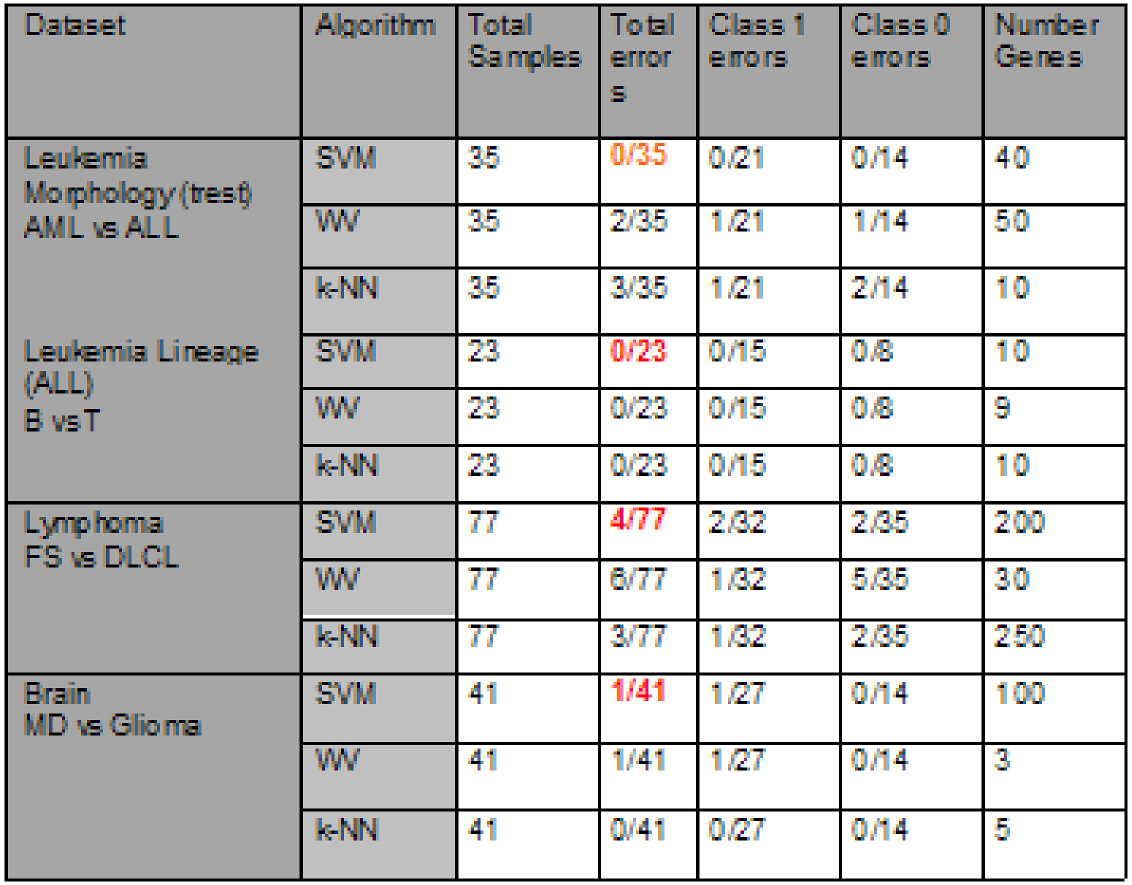
2. SVM illustration in cancer classification

Example 1: Myeloid vs. Lymphoblastic leukemias [Golub]

ALL: acute lymphoblastic leukemia
AML: acute myeloblastic leukemia

SVM training: leave one out cross-validation

SVM

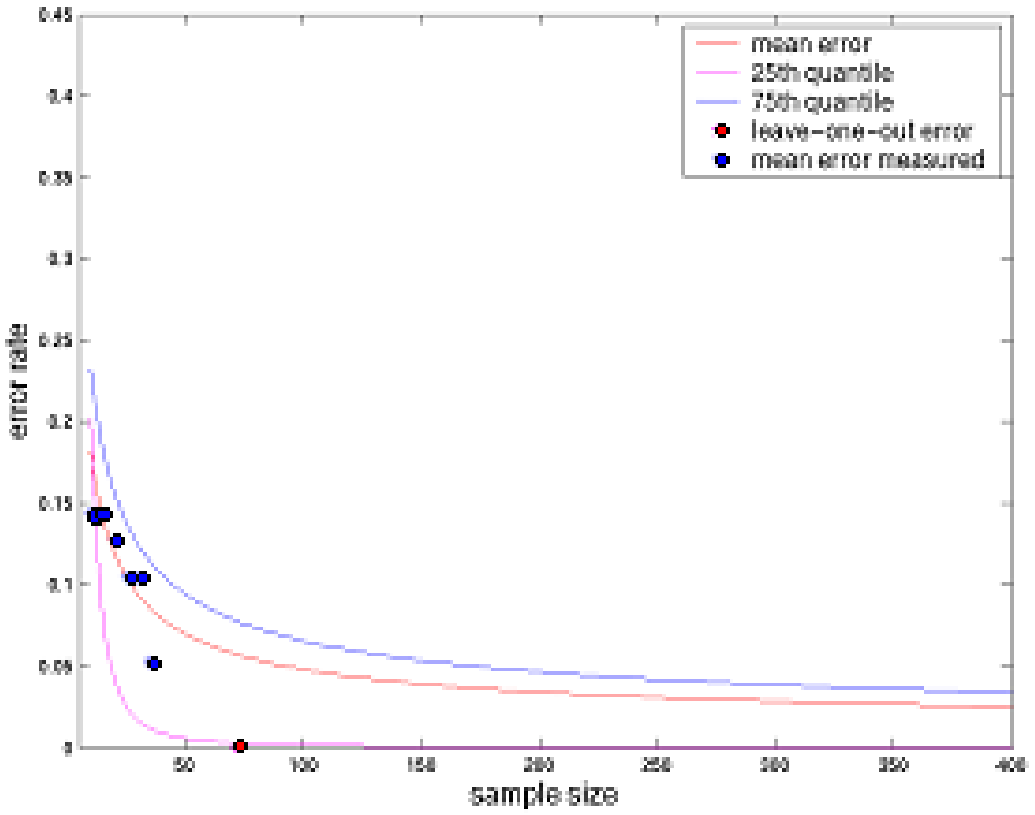


Dataset	Algorithm	Total Samples	Total error s	Class 1 errors	Class 0 errors	Number Genes
Leukemia Morphology (trest) AML vs ALL	SVM	35	0/35	0/21	0/14	40
	WV	35	2/35	1/21	1/14	50
	k-NN	35	3/35	1/21	2/14	10
Leukemia Lineage (ALL) B vs T	SVM	23	0/23	0/15	0/8	10
	WV	23	0/23	0/15	0/8	9
	k-NN	23	0/23	0/15	0/8	10
Lymphoma FS vs DLCL	SVM	77	4/77	2/32	2/35	200
	WV	77	6/77	1/32	5/35	30
	k-NN	77	3/77	1/32	2/35	250
Brain MD vs Glioma	SVM	41	1/41	1/27	0/14	100
	WV	41	1/41	1/27	0/14	3
	k-NN	41	0/41	0/27	0/14	5

S. Mukherjee

fig. 1: Myeloid and Lymphoblastic Leukemia classification by SVM, along with other discrimination tasks; k-NN is k-nearest neighbors; WV is weighted voting

SVM



S. Mukherjee

fig 2: AML vs. ALL error rates with increasing sample size; Above curves are error rates within test sets after training of machine with a training set.

3. Result: SVM on cancer (Alon, et al., PNAS)

Recall: 40 samples colon cancer tissue
22 samples of normal colon tissue (62 total).

For each sample computed

$$\mathbf{x} = (x_1, \ldots, x_p) = \text{gene expression array}$$

SVM

Let

$$D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{62}$$

be collection of samples and correct classifications:

$$y_i = \begin{cases} 1 & \text{if } \mathbf{x}_i \text{ cancerous} \\ -1 & \text{if } \mathbf{x}_i \text{ non-cancerous} \end{cases}.$$

Feature space F is 6,500 dimensional (6,500 genes)

SVM

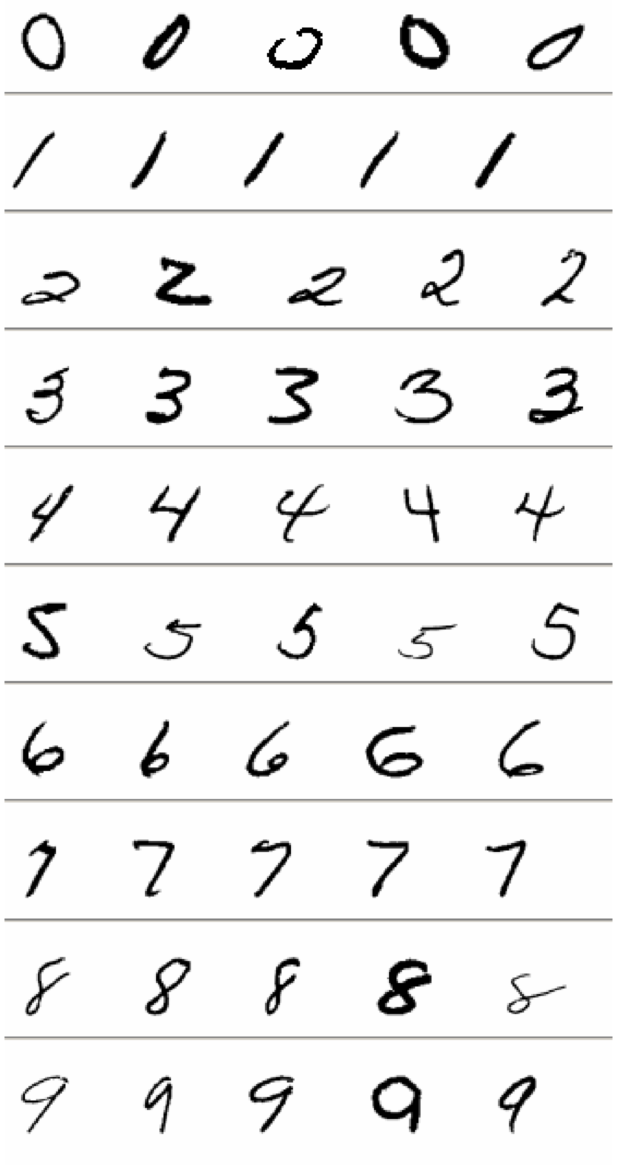
Result: using leave one out cross validation (leave one sample out and train a machine on the others) obtained:

Misclassification of 6/62 tissues.

4. Example application: handwritten digit recognition - USPS (Scholkopf, Burges, Vapnik)

Handwritten digits:

SVM



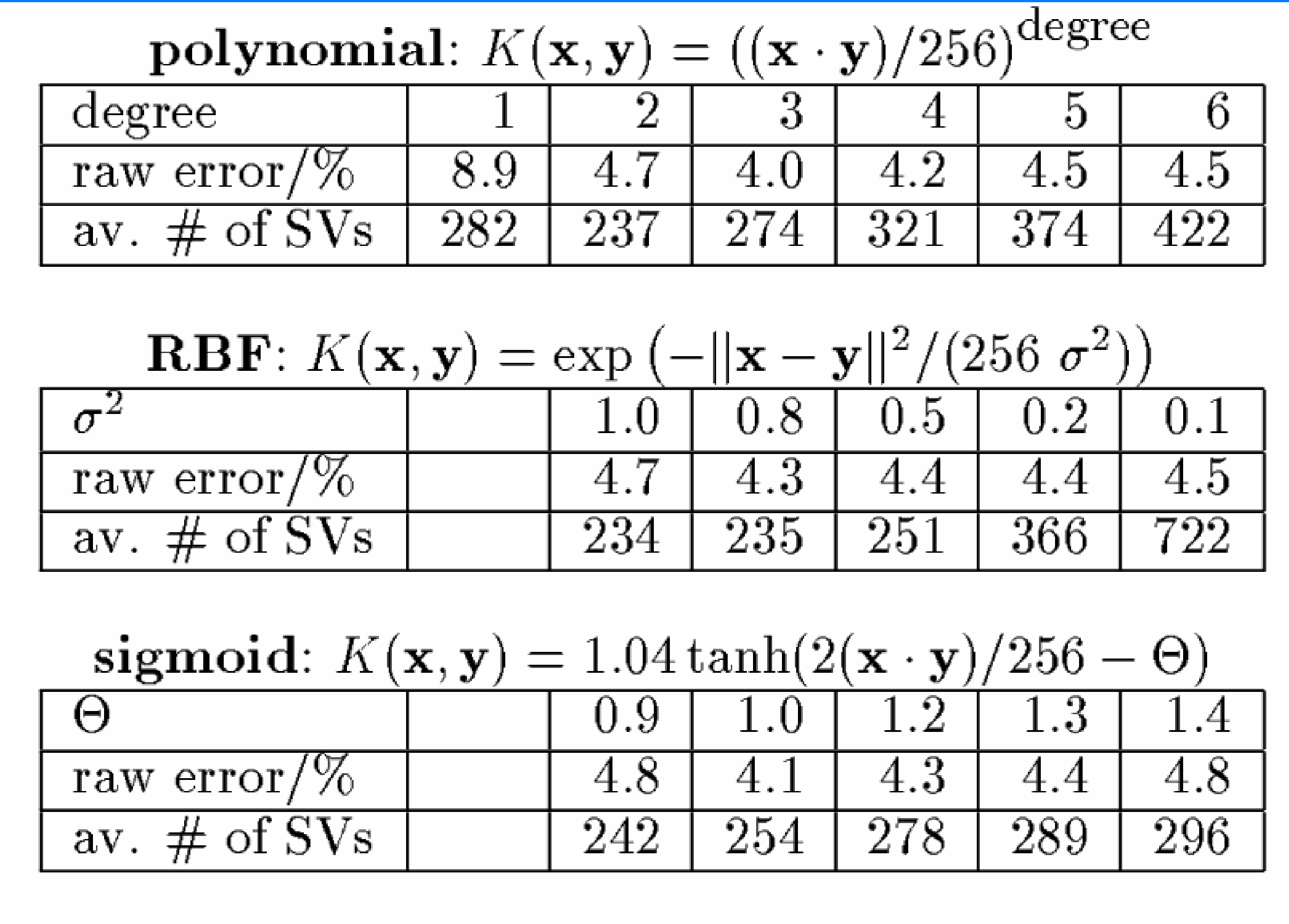
SVM

Training set (sample size): 7300; Test set: 2000

10 class classifier; each class has a separating SVM function:

Results:

SVM



polynomial: $K(\mathbf{x},\mathbf{y}) = ((\mathbf{x}\cdot\mathbf{y})/256)^{\text{degree}}$

degree	1	2	3	4	5	6
raw error/%	8.9	4.7	4.0	4.2	4.5	4.5
av. # of SVs	282	237	274	321	374	422

RBF: $K(\mathbf{x},\mathbf{y}) = \exp\left(-\|\mathbf{x}-\mathbf{y}\|^2/(256\,\sigma^2)\right)$

σ^2		1.0	0.8	0.5	0.2	0.1
raw error/%		4.7	4.3	4.4	4.4	4.5
av. # of SVs		234	235	251	366	722

sigmoid: $K(\mathbf{x},\mathbf{y}) = 1.04\tanh(2(\mathbf{x}\cdot\mathbf{y})/256 - \Theta)$

Θ		0.9	1.0	1.2	1.3	1.4
raw error/%		4.8	4.1	4.3	4.4	4.8
av. # of SVs		242	254	278	289	296

5. Example: machine learning and finance

A note on machines and neural networks:

The notion of machine learning includes neural network architectures.

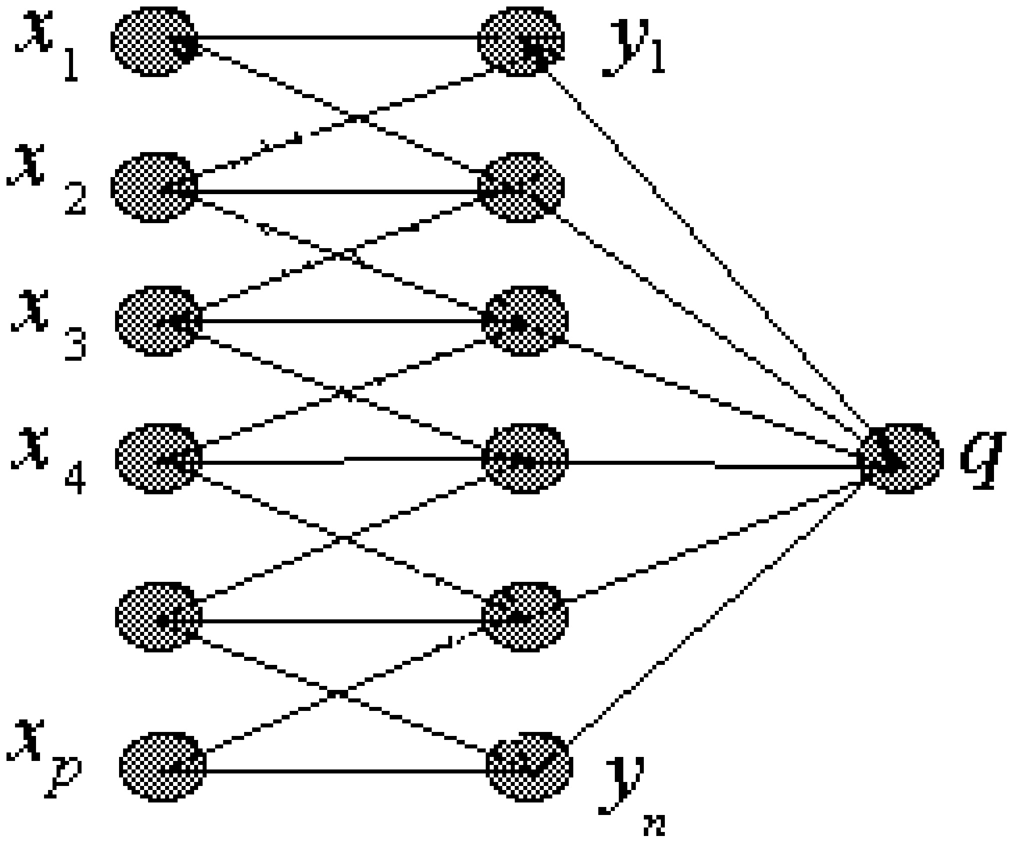
The vector of inputs to such a network is **x** $= (x_1, \ldots, x_p)$ and the output (prediction) is q.

ML and Finance

Thus we are implementing the function

$$f(\mathbf{x}) = q.$$

Neural net:



ML and Finance

Input: feature vector **x** $= (x_1, \ldots, x_p)$
Output: number q.

Training: show the network examples of known 'correct' outputs q_i based on input (training) vectors $\mathbf{x}_i$:

$$\{(\mathbf{x}_1, q_1), \ldots, (\mathbf{x}_N, q_N)\}.$$

Example 2: $\mathbf{x}_1$ is a time series of consecutive daily prices of a given stock, while q_1 is a predicted return on the stock over the day immediately following this time series.

This machine is a neural network trained with weights (parameters) that transform input vector $\mathbf{x} = (x_1, \ldots, x_p)$ entering on the left layer into a single ouptut number $q = f(\mathbf{x}) =$ predicted stock price formed on the right.

ML and Finance

Other Machine Learning methods are really generalized versions of the above input-output network.

ML and Finance

Input: feature vector

$$\mathbf{x} = (x_1, \ldots, x_p)$$

desired ouptut: $q =$ predicted stock price.

6. The simplest machine: linear regression

Example: Let's build a regression machine to predict stock price from time series.

Training data:

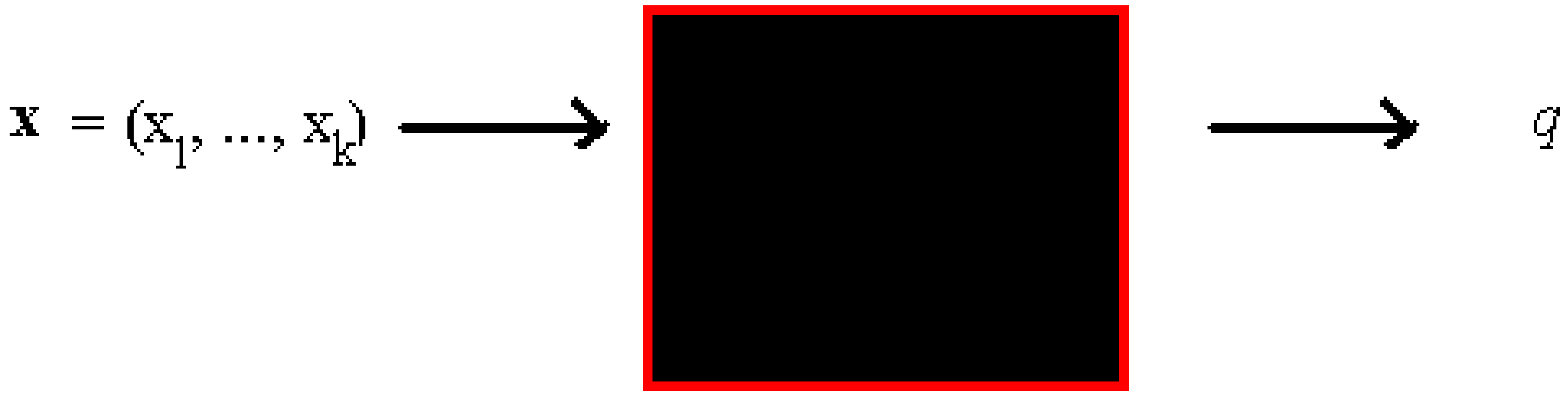
$$\{(\mathbf{x}_1, q_1), \ldots, (\mathbf{x}_N, q_N)\}.$$

ML and Finance

Machine will find a rule that takes prior prices x_i to today's predicted price q :

$$\mathbf{x} = (x_1, \ldots, x_p) \to q.$$

Regression machine:



$$q = \beta_1 x_1 + \ldots + \beta_p x_p + \beta_0 = q = \boldsymbol{\beta} \cdot \mathbf{x} + \beta_0$$

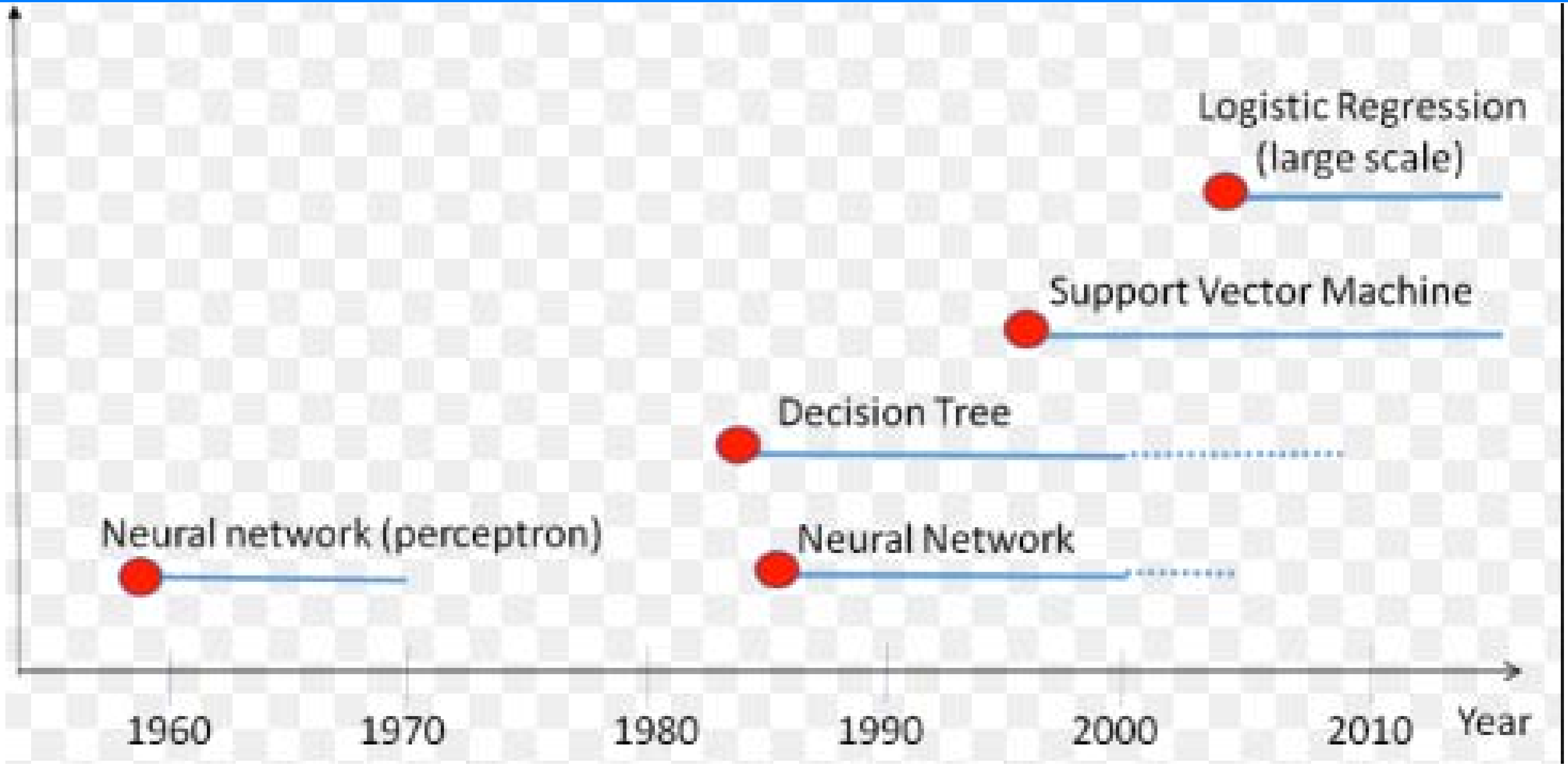
ML and Finance

$$\begin{tikzcd} \mathbf{x} = (x_1, \ldots, x_p) \arrow[r] & \boxed{q = \boldsymbol{\beta} \cdot \mathbf{x} + \beta_0} \arrow[r] & q \end{tikzcd}$$

How do we train the machine? Find coefficients $\beta_0, \ldots, \beta_p$ from ordinary regression based on the training data set

$$\{(\mathbf{x}_1, q_1), \ldots, (\mathbf{x}_N, q_N)\}.$$

ML and Finance

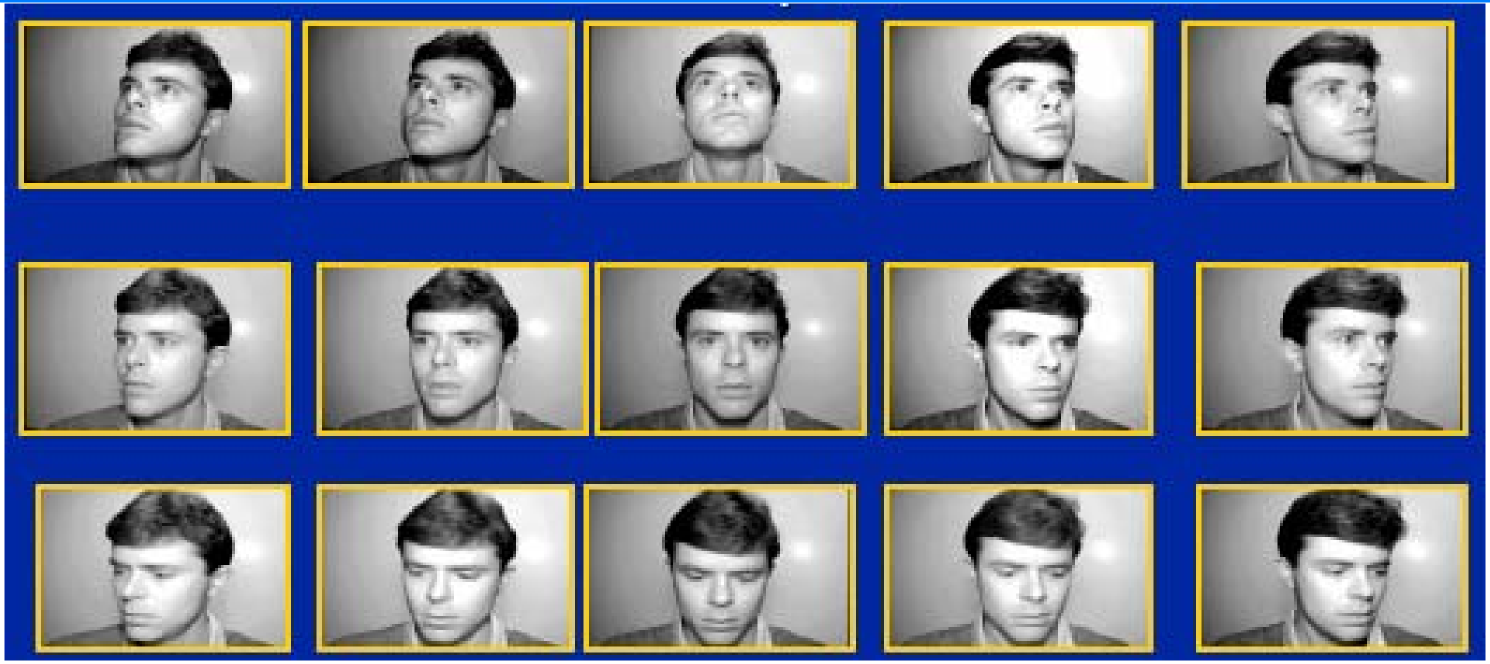


http://www.aboutdm.com

The Three Pillars of Machine Learning:

1. **The world can be injected into any computer using feature vectors** - any object can be summarized as a string of numbers. Example -

ML Pillars



This object (a face) can be converted to a feature vector (numbers) in different ways:

(a) for example a face is just 10^6 numbers (photograph pixel intensities).

ML Pillars

(b) or a face is a list of distances between the primary facial features (e.g. eye, ears, lip corners), or (better) distance ratios.

Main point: feature vectors are the language of learning machines - objects are now strings of numbers.

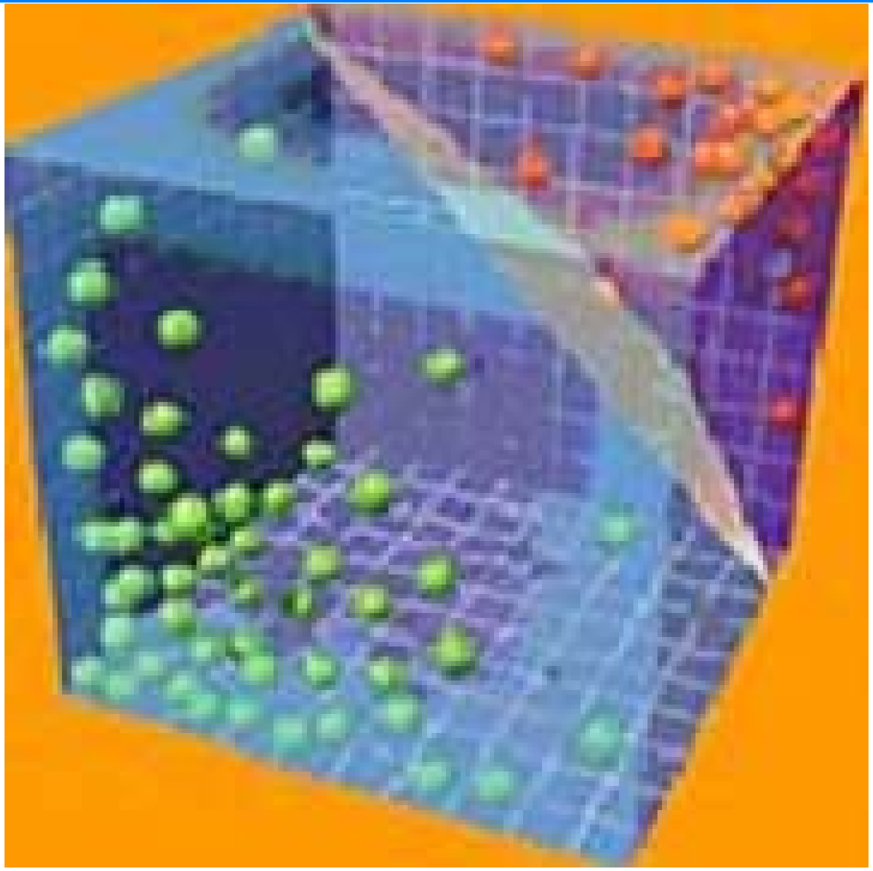
ML Pillars

(2) Geometrization of data: a n-dimensional feature vector is a string of numbers that become coordinates in an n more dimensional space (feature space).

Thus objects to be classified become geometric points in a space.

Identifying an object becomes identifying its location in feature space:

ML Pillars



Kernel Trick

(3) **Kernel trick**: Sometimes feature vectors are large (e.g. 10^6 numbers = 10^6 dimensional space). Keeping track of 10^6 numbers for each example - too much!

Computers will protest ... curse of dimensionality!

Trick: fix the data matrix:

Kernel Trick

Example 3: N independent company daily price histories $x_1, \ldots, x_p$ followed by a predicted price q after x_p.

	Features					Outcome
Company 1	x_1	x_2	x_3	$\ldots$	x_p	q
Company 2	x_1	x_2	x_3	$\ldots$	x_p	q
$\vdots$	$\vdots$					$\vdots$
Company N	x_1	x_2	x_3	$\ldots$	x_p	q

Kernel Trick

# features $= p \ =$ dimensions
# samples $= N$

Often p >> N (i.e., very high frequency price samples can have more time slices k than sample companies N)

$\Rightarrow$ $($Curse of dimensionality$)$

Data matrix:

$$\mathbf{X} = k \text{ rows } \left\{ \underbrace{\begin{bmatrix} x_1 & x_2 & x_3 & \dots & x_p & q \\ x_1 & x_2 & x_3 & \dots & x_p & q \\ x_1 & x_2 & x_3 & \dots & x_p & q \\ \vdots & \vdots & & & \ddots & \\ x_1 & x_2 & x_3 & \dots & x_p & q \end{bmatrix}}_{N \text{ columns}} \right.$$

Kernel Trick

$$\mathbf{X}^T = \begin{bmatrix} x_1 & x_1 & x_1 & \dots & x_1 \\ x_2 & x_2 & x_2 & \dots & x_2 \\ x_3 & x_3 & x_3 & \dots & x_3 \\ \vdots & \vdots & & \ddots & \\ x_p & x_p & x_p & \dots & x_p \\ q & q & q & \dots & q \end{bmatrix}$$

Then: usually use the final matrix to be the *covariance matrix*

Kernel Trick

$$\mathbf{X}^T \cdot \mathbf{X}$$

[note this is $p \times p$ = *huge matrix!*]

Now use the *kernel matrix*

$$\mathbf{X} \cdot \mathbf{X}^T$$

[now this is an $N \times N$ matrix - can be much smaller!]

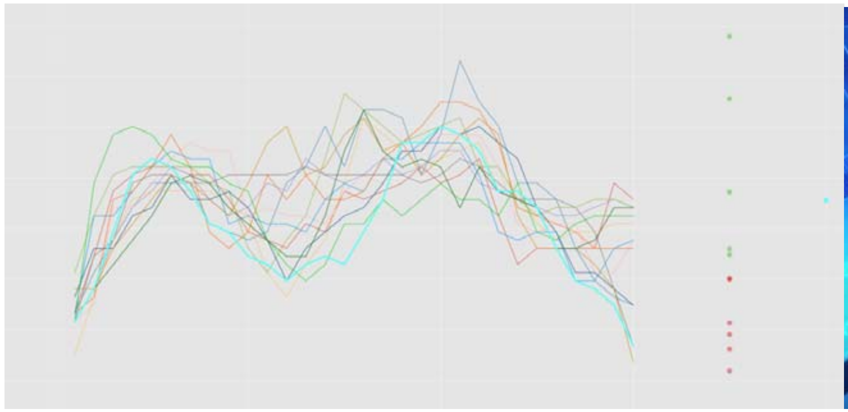
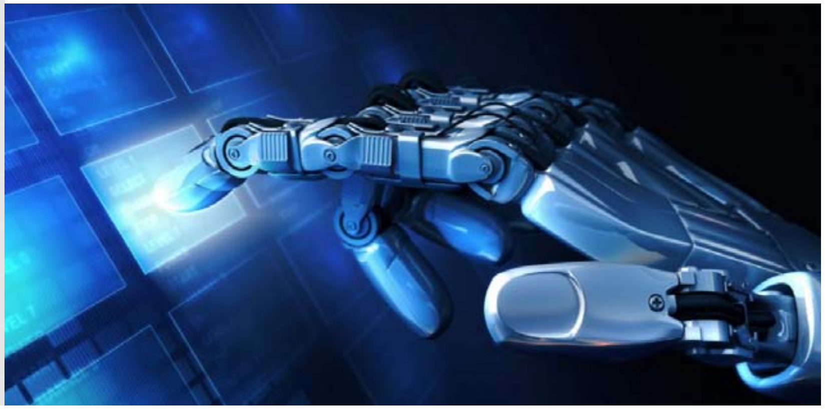
But both the large and the small matrix encode exactly the same information!

Kernel Trick

Thus high dimensional problem ($k \times k$) becomes low dimensional problem ($N \times N$)

$\longrightarrow$ Kernel trick!

7. Example (continued): Machine Learning and Stock Trading:

https://blogs.msdn.microsoft.com/meechao/2017/01/19/building-an-experimental-stock-trading-system-using-machine-learning-and-python/

HIGH FREQUENCY TRADING USING ML:

For a specific company (say IBM), train a machine to predict return y (percentage change in stock value) in the next *millisecond* based on the pattern of stock values over the last p milliseconds, $(x_1, \ldots, x_p)$.

That is, find a machine M to implement the best approximate map

Equities

$$y = f_M(x_1, \ldots, x_p).$$

High frequency methods are well developed (sometimes not publicized).

FUNAMENTALS (LOW FREQUENCY) TRADING USING ML:

Also important:

Try a feature vector

$$\mathbf{x} = (x_1, \dots, x_p).$$

where $x_1, x_2, \dots$ are:

- company fundamental numbers (say month to month or quarterly)

Equities

- US Economic indicators (leading economic indicators, interest rates, etc.)

- Sentiment indicators (possibly only low frequency updates)

Equities

Main point:

Fundamentals dominate low frequency trading;

Psychology/game theory dominate high frequency trading.

Machine learning can be used in both!

More about low frequency trading

- Focus on low frequency trading by identifying by predicting price from slowly changing fundamental performance indicators $(x_1, \ldots, x_p)$

Low Frequency Trading

- Lots (hundreds) of performance indicators x_i (e.g. taxes, operating income etc.) are available for thousands of companies.

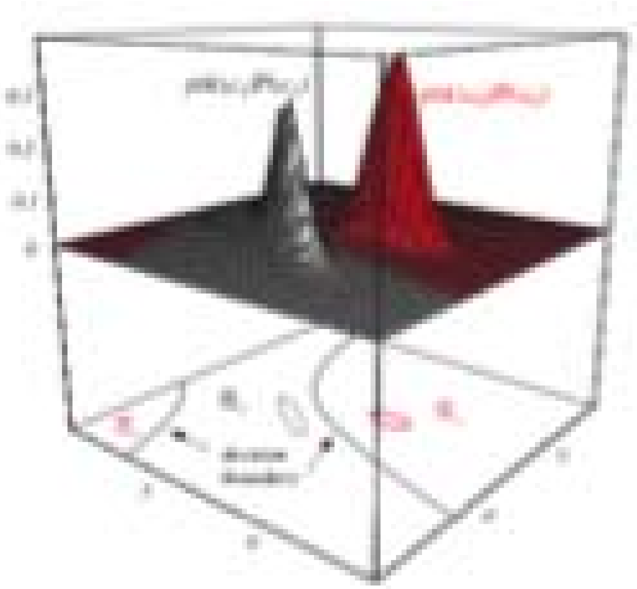
- ML techniques can explore and combine large numbers of indicators x_i to identify extreme performers (likely to have very high earnings or large losses).

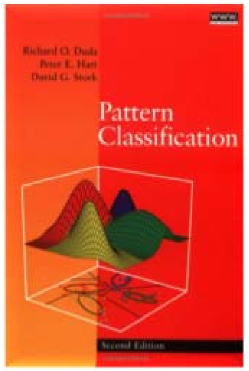
Low Frequency Trading

- Each stock at any time is now a point in a high-dimensional space (one dimension per performance indicator). Can have hundreds or thousands of indicators x_i.

- Goal is always the same: use indicators $(x_1, \ldots, x_p)$ to predict return y in the next time period
- One can use an ensemble of ML methods not just Support Vector Machines.

Low Frequency Trading




Richard O. Duda
Peter E. Hart
David G. Stork

Pattern Classification

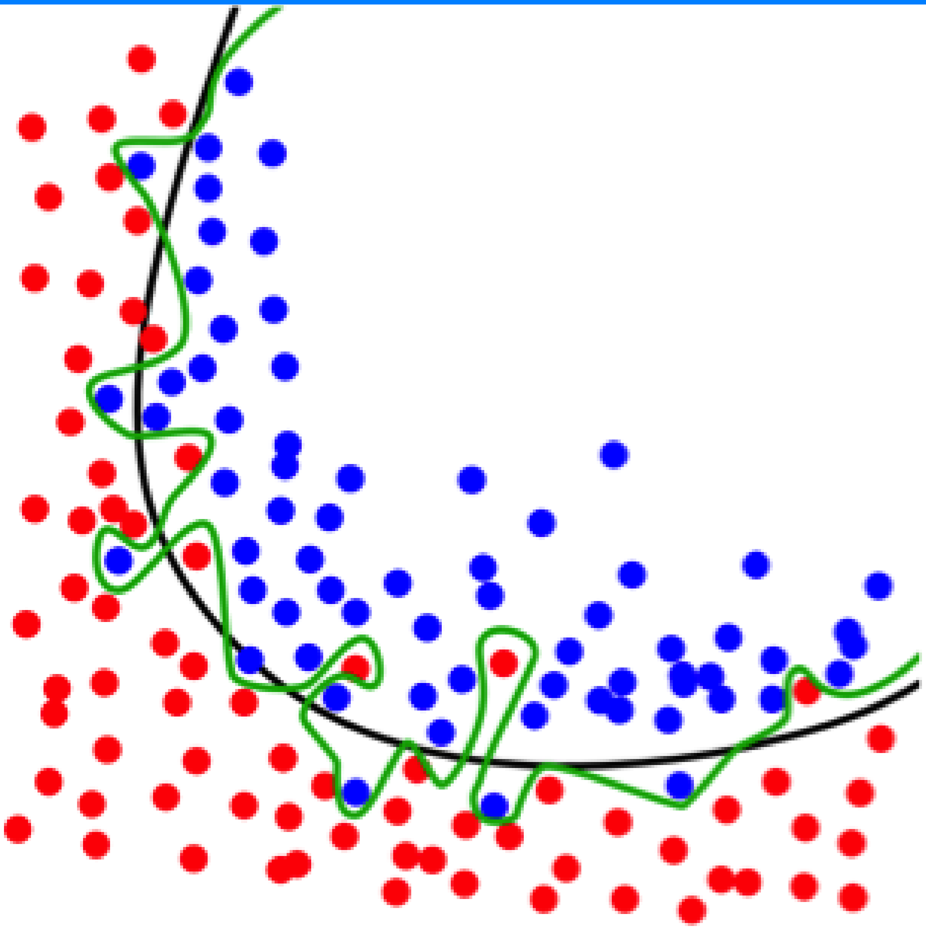
Second Edition

Low Frequency Trading

Note: If only two classes A and B are to be predicted, *there are many ways to choose the separator between class A (good companies, blue) and class B (not good companies, red) when they are placed in the feature space F.*

- If over-fitting occurs, performance on the test set will be poor; i.e. there is little predictive value.

Low Frequency Trading



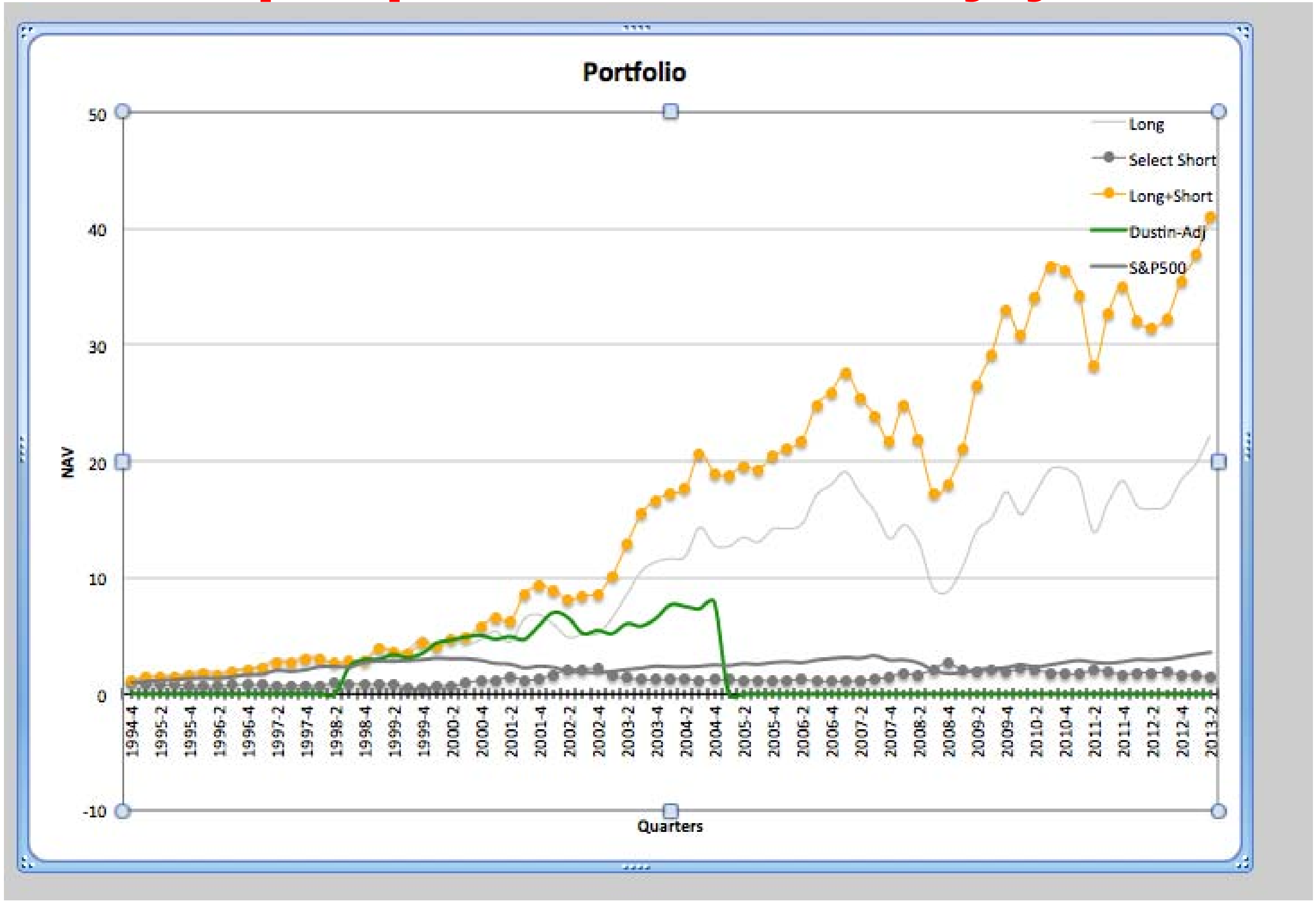
Low Frequency Trading

- Solution: numerous highly technical approaches can be implemented to prevent overfitting

- Most important rule of thumb: keep your separator *simple!*

Performance

Sample portfolios: Thirty years



1994-2013

Sentiment Analysis is a New Application

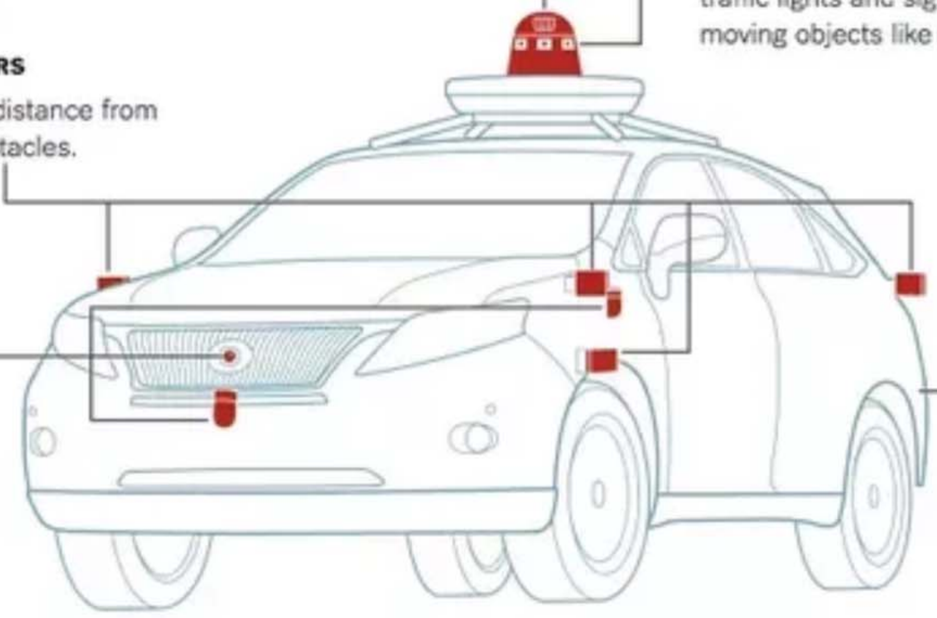
Machine learning tools for predicting a company stock may be extended to include news items (textual analysis of newspapers), analyst rankings of companies, and company sentiments from social networks (twitter, stocktwit, facebook, G+)

8. The latest machines

- Self-driving Ubers:

Self-Driving Ubers

How a Car Drives Itself



LIDAR UNIT

Constantly spinning, it uses laser beams to generate a 360-degree image of the car's surroundings.

RADAR SENSORS

Measure the distance from the car to obstacles.

ADDITIONAL LIDAR UNITS

CAMERAS

Uses parallax from multiple images to find the distance to various objects. Cameras also detect traffic lights and signs, and help recognize moving objects like pedestrians and bicyclists.

MAIN COMPUTER (LOCATED IN TRUNK)

Analyzes data from the sensors, and compares its stored maps to assess current conditions.

By Guilbert Gates | Source: Google | Note: Car is a Lexus model modified by Google.

Self-Driving Ubers

Idealization of input-output:

$$\text{feature vector } \mathbf{x} = (x_1, \ldots, x_p)$$

For example:

$x_1 =$ distance to nearest obstacle in the front of car (0 degrees from front direction)

$x_2 =$ distance to nearest obstacle 10 degrees from front direction.

Self-Driving Ubers

$x_3 =$ distance to nearest obstacle 20 degrees from front direction.

Etc.

Another possible feature vector

$$\mathbf{y} = (y_1, \ldots, y_p)$$

$y_1 =$ direction of motion of object at x_1
$y_2 =$ direction of motion of object at x_2
$y_3 =$ direction of motion of object at x_3

Etc.

Self-Driving Ubers

or

$$\mathbf{z} = (z_1, \ldots, z_p)$$

$z_1 =$ speed of motion of object at x_1
$z_2 =$ speed of motion of object at x_2

Etc.

How to combine information in feature vectors?

Super-feature vector

$$\mathbf{v} = (x_1, \ldots, x_p; y_1, \ldots, y_p; z_1, \ldots, z_p.\ldots.)$$

[a very large vector containing all numbers the machine might be interested in]

Self-Driving Ubers

OR: Use the kernel trick: use a *kernel function* $K_1(\mathbf{x}^{(1)}, \mathbf{x}^{(2)})$ where $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$ are independent copies of **x** (vector of object distances).

Now form another kernel function $K_2(\mathbf{y}^{(1)}, \mathbf{y}^{(2)})$ where $\mathbf{y}^{(1)}$, $\mathbf{y}^{(2)}$ are independent copies of variable **y** (vector of object directions).

Etc.

Training (the car machine)

Example of training:

Now take lots of example measured values of high dimensional super-vectors **v** in many different situations:

$$\mathbf{v}^{(1)}, \mathbf{v}^{(2)}, \ldots, \mathbf{v}^{(N)}.$$

Desired machine output for input **v**:

$$f(\mathbf{v}) \begin{cases} > 0 & \text{if brakes should be applied} \\ \leq 0 & \text{if brakes should not be applied} \end{cases}$$

Self-Driving Ubers

For SVM machine, if **v** is the current measured super-vector, then there is an appropriate kernel function $K(\mathbf{v}^{(1)}, \mathbf{v}^{(2)})$ defined on independent copies $\mathbf{v}^{(1)}, \mathbf{v}^{(2)}$ of the super-vector **v**, such that

$$f(\mathbf{v}) = \sum_{i=1}^{N} a_i K(\mathbf{v}, \mathbf{v}^{(N)}) + a_0.$$

Self-Driving Ubers

Note this gives a *curved* (non-plane) separation between the two regions, $f(\mathbf{v}) > 0$ and $f(\mathbf{v}) \leq 0$,

i.e., the separation is $f(\mathbf{v}) = 0$.

Self-Driving Ubers

The coefficients a_i can be obtained using a linear algebra algorithm from the $k \times k$ *kernel matrix* **K** with entries

$$\mathbf{K}_{ij} = K(\mathbf{v}^{(i)}, \mathbf{v}^{(j)})$$

$(i, j = 1, \ldots, N)$.

Self-Driving Ubers

The good news: if we have already derived a good kernel matrix $\mathbf{K}_x$ for the **x** variables, $\mathbf{K}_y$ for the **y** variables, $\mathbf{K}_z$ for the **z** variables, etc., then the *correct* full kernel matrix incorporating all of this information is the sum matrix

$$\mathbf{K} = \mathbf{K}_x + \mathbf{K}_y + \mathbf{K}_z + \dots$$

integrating all of the variables **x**, **y**, **z** *automatically.*

Again the utility of the kernel trick!

More about the kernel trick: Can we use just any function $K(\mathbf{v}, \mathbf{w})$ above?

Almost - the function $K(\mathbf{v}, \mathbf{w})$ must be *positive definite*, i.e., if take any fixed set of measured super-vectors $\mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(k)}$, the *kernel matrix*

$$\mathbf{K}_{ij} = K(\mathbf{v}^{(i)}, \mathbf{v}^{(j)})$$

is positive definite, i.e. has only non-negative eigenvalues.

Self-Driving Ubers

Lots of functions work though. For example

$K(\mathbf{v}, \mathbf{w}) = \mathbf{v} \cdot \mathbf{w}$ (linear kernel)

$K(\mathbf{v}, \mathbf{w}) = e^{-|\mathbf{v}-\mathbf{w}|^2/\sigma^2}$ (Gaussian kernel) with width σ

$K(\mathbf{v}, \mathbf{w}) = (1 + \mathbf{v} \cdot \mathbf{w})^d$ (polynomial kernel)

See:

Self-Driving Ubers

More generally: can let

$$K(\mathbf{v}, \mathbf{w}) = \Phi(\mathbf{v})\Phi(\mathbf{w}) \qquad (1)$$

for any continuous function Φ: $\mathbb{R}^m \longrightarrow \mathbb{R}^n$
($m =$ dimension of $\mathbf{v}$)

Can show: using this kernel K in (1) is equivalent to mapping *all* feature vectors **v** that we encounter into $\Phi(\mathbf{v})$ in *all* ML operations.

Self-Driving Ubers

After this mapping just use the simple linear kernel function $K(\mathbf{v},\mathbf{w}) = \mathbf{v} \cdot \mathbf{w}$ (the one that gives a linear boundary between the two categories), *but always replacing* $\mathbf{v} \longrightarrow \Phi(\mathbf{v})$ *and* $\mathbf{w} \longrightarrow \Phi(\mathbf{w})$ before starting the calculation.

Self-Driving Ubers

Φ is called a *feature map*.

When is a feature map useful? When a linear separation does not work between the blue and red dots:

See:

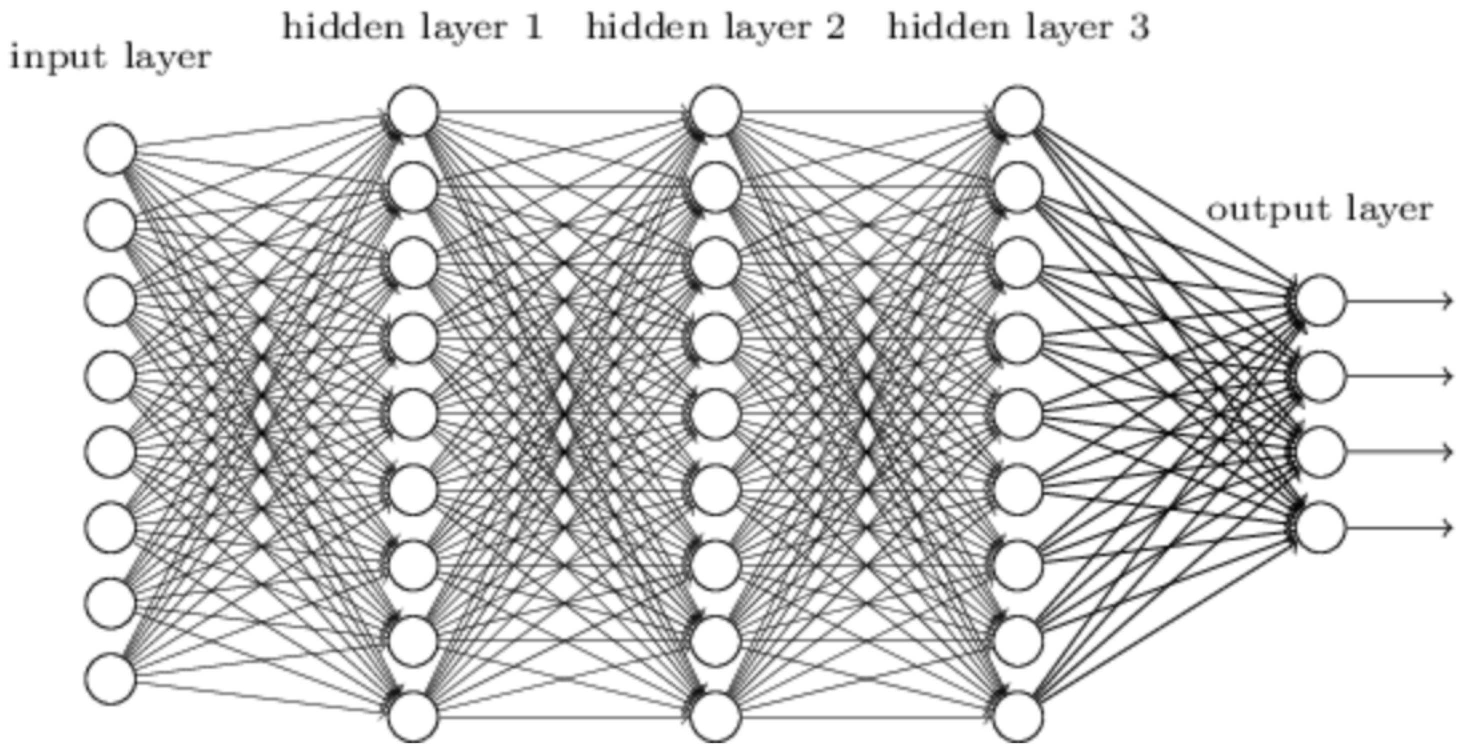
https://www.youtube.com/watch?v=3liCbRZPrZA

Deep Networks

- Deep neural networks:

Replace single middle (hidden) layer of neurons by 3 or more hidden layers.

Deep Networks



Input layer has neural activations that form a vector **x** :

Deep Networks

$$\mathbf{x} = (x_1, \ldots, x_p)$$

Hidden layer 1 activations:

$$\mathbf{y} = (y_1, \ldots, y_n)$$

Hidden layer 2 activations:

$$\mathbf{z} = (z_1, \ldots, z_n)$$

$$\vdots$$

Deep Networks

Final (output) layer activations:

$$\mathbf{q} = (q_1, \ldots, q_n).$$

Feedforward function from **x** to **y** is *almost* a linear map:

$$\mathbf{y} = \mathbf{W} \cdot \mathbf{x},$$

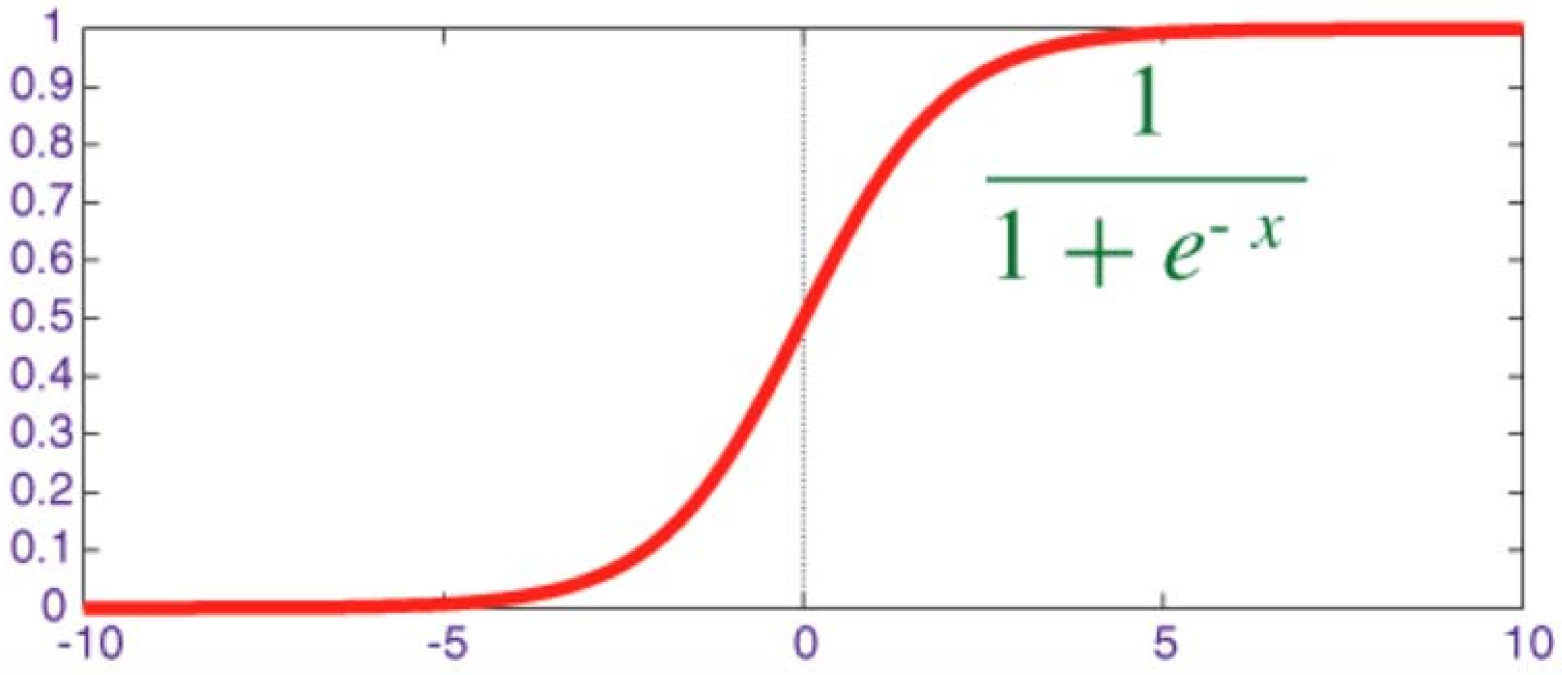
where matrix entries

$w_{ij} = \mathbf{W}_{ij}$ = connection strength from neuron x_i to neuron y_j

Deep Networks

Almost: now apply a *sigmoid* function $\phi(y) = \frac{1}{1+e^{-x}}$ that for each component y of **y** looks like

Deep Networks



.

Note ϕ is bounded to prevent your neurons y_i from burning out. Thus we actually have:

Deep Networks

$$\mathbf{y} = \phi(\mathbf{W} \cdot \mathbf{x}).$$

Now repeat same map from **y** to **z** :

$$\mathrm{z} = \phi(\mathbf{V} \cdot \mathbf{y}),$$

where **V** is now matrix of weights from **y** layer to **z** layer, etc.

Each layer encodes more abstract information about the input information **x** (image with pixel intensites x_i).

Deep Networks

Finally, the output layer (with enough layers) tells you whether the painting you showed to the first layer **x** is a Da Vinci or not.

Deep Networks

The trick: you need lots of layers.

Deep Networks

x (first) layer encodes visual pixels of painting

y (second) layer encodes directionalities of edges at nearby pixels

z (third) layer encodes presence of shapes (circles, triangles) at nearby pixels.

$\vdots$

Each successive layer encodes higher levels of abstract information.

Deep Networks

$\vdots$

q (last) layer encodes identity of the painter

(after sufficient training of your network machine!).

*Who knew that iterated maps (applying matrix multiplications and the sigmoid function ϕ repeatedly to get from input **x** to output **q**) had such power!*

Final remark

Just as standard computers can emulate our ability to perform precise calculational tasks (and even linguistic tasks)

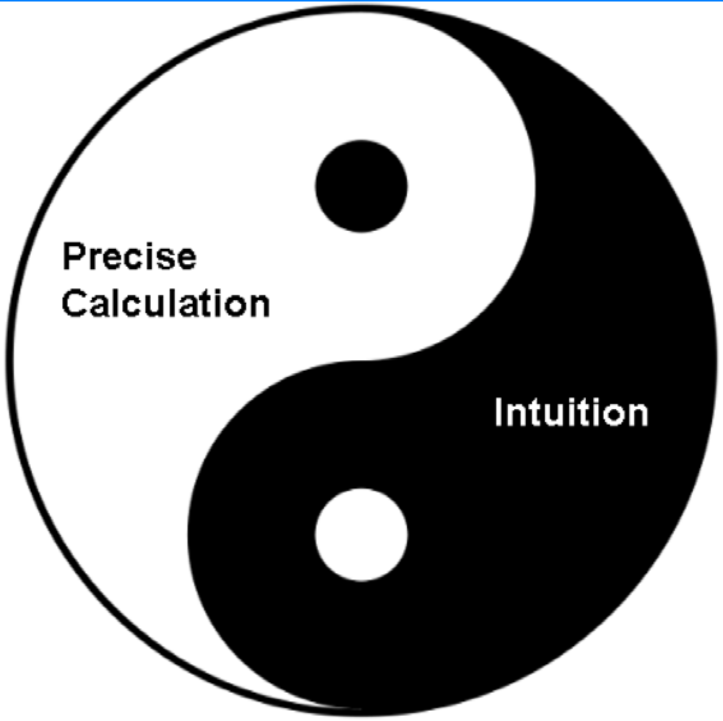
Machine learning (e.g. in a neural network or SVM or elsewhere) emulates our *intuition*.

Yin and Yang

Intuition is what takes an input situation consisting of feature vectors **x**, **y**, **z** representing our Uber car's current environment and then tells it what to do next.

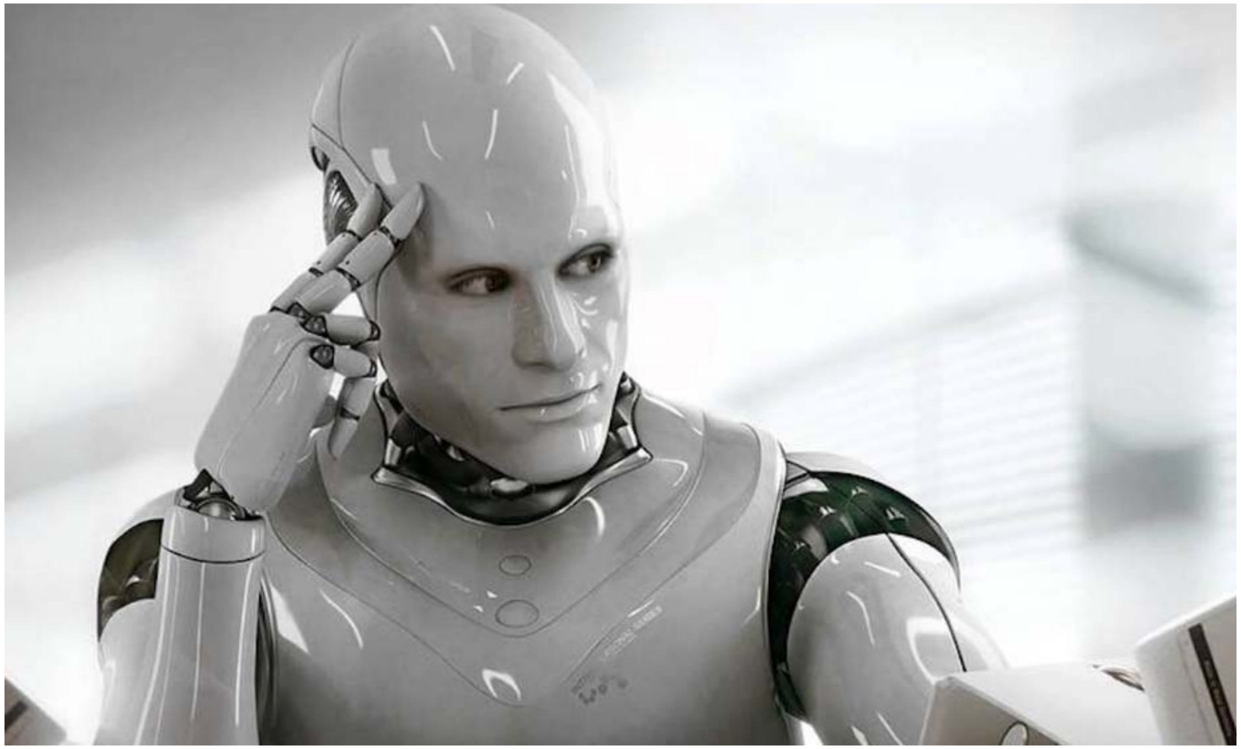
Intuition will also be emulated by computers, using machine learning -- already our precise calculational abilities have been emulated (very well!) by standard computers.

Yin and Yang



The combination of precise calculation and intuitive thinking are the *yin* and *yang* of our thinking, and will also be the *yin* and *yang* of the thinking of future Artificial Intelligences.

Yin and Yang



https://medium.com/fluxx-studio-notes/ai-virtual-assistants-and-chat-bots-before-now-and-in-the-future-df979529ad5f