CSE 6242 / CX 4242

Time Series

Mining and Forecasting

Duen Horng (Polo) Chau
Georgia Tech

Slides based on Prof. Christos Faloutsos's materials

Outline

- Motivation
- Similarity search – distance functions
- Linear Forecasting
- Non-linear forecasting
- Conclusions

Problem definition

- **Given**: one or more sequences

  $x_1 , \; x_2 , \; \ldots , \; x_t , \; \ldots$

  $(y_1, y_2, \ldots , y_t , \ldots)$

  $(\ldots)$

- **Find**
  - similar sequences; forecasts
  - patterns; clusters; outliers

Motivation - Applications

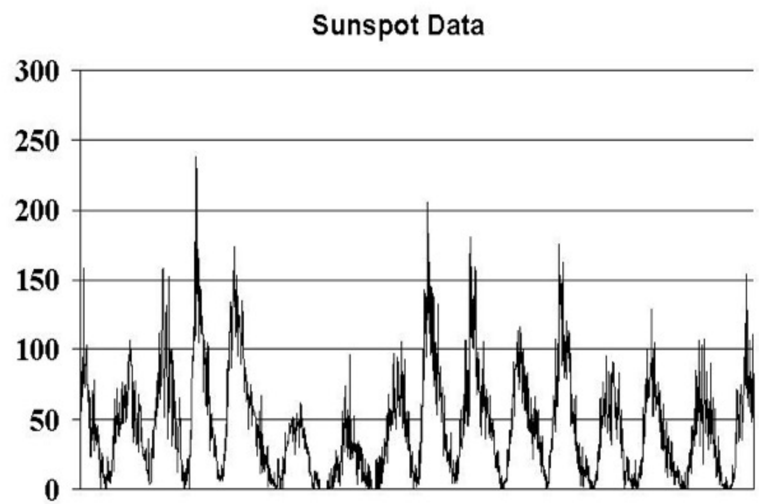
- Financial, sales, economic series
- Medical
  - ECGs +; blood pressure etc monitoring
  - reactions to new drugs
  - elderly care

Motivation - Applications (cont'd)

- 'Smart house'
  - sensors monitor temperature, humidity, air quality
- video surveillance

Motivation - Applications (cont'd)

- Weather, environment/anti-pollution
  - volcano monitoring
  - air/water pollutant monitoring

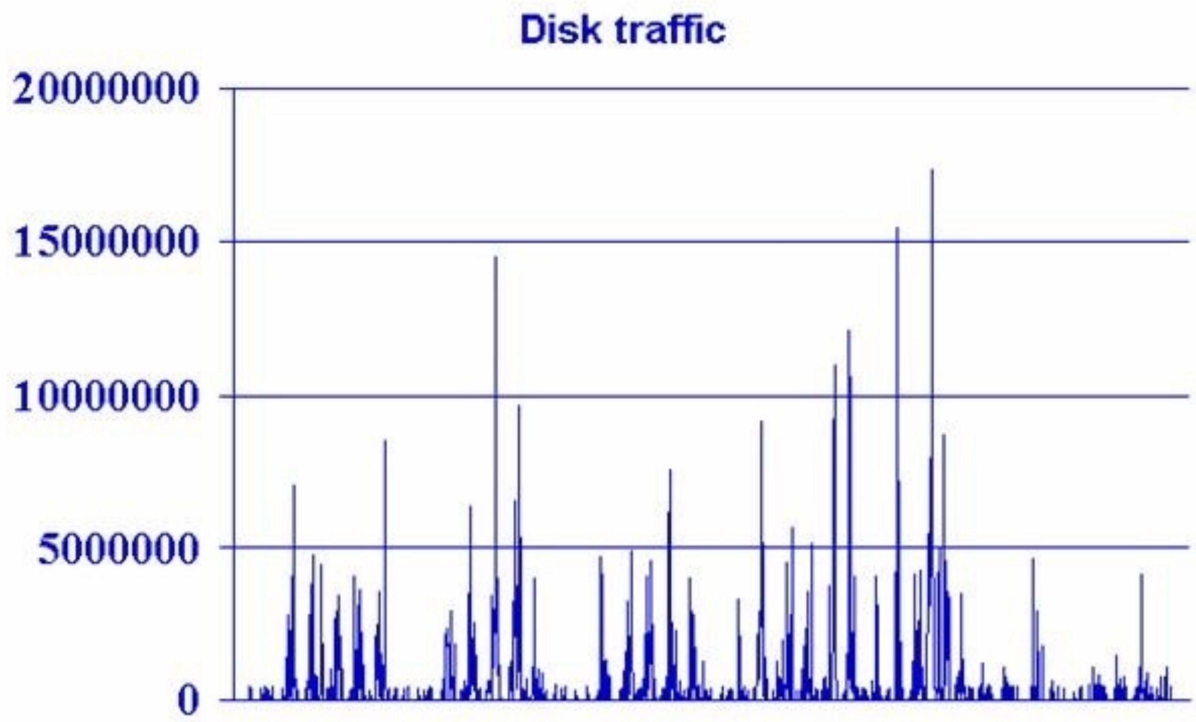


Motivation - Applications (cont'd)

- Computer systems
  - 'Active Disks' (buffering, prefetching)
  - web servers (ditto)
  - network traffic monitoring
  - ...

Stream Data: Disk accesses

#bytes



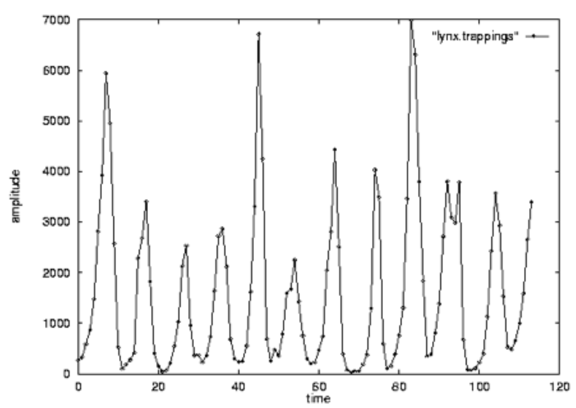
time

Problem #1:

Goal: given a signal (e.g.., #packets over time)

Find: patterns, periodicities, and/or compress

count

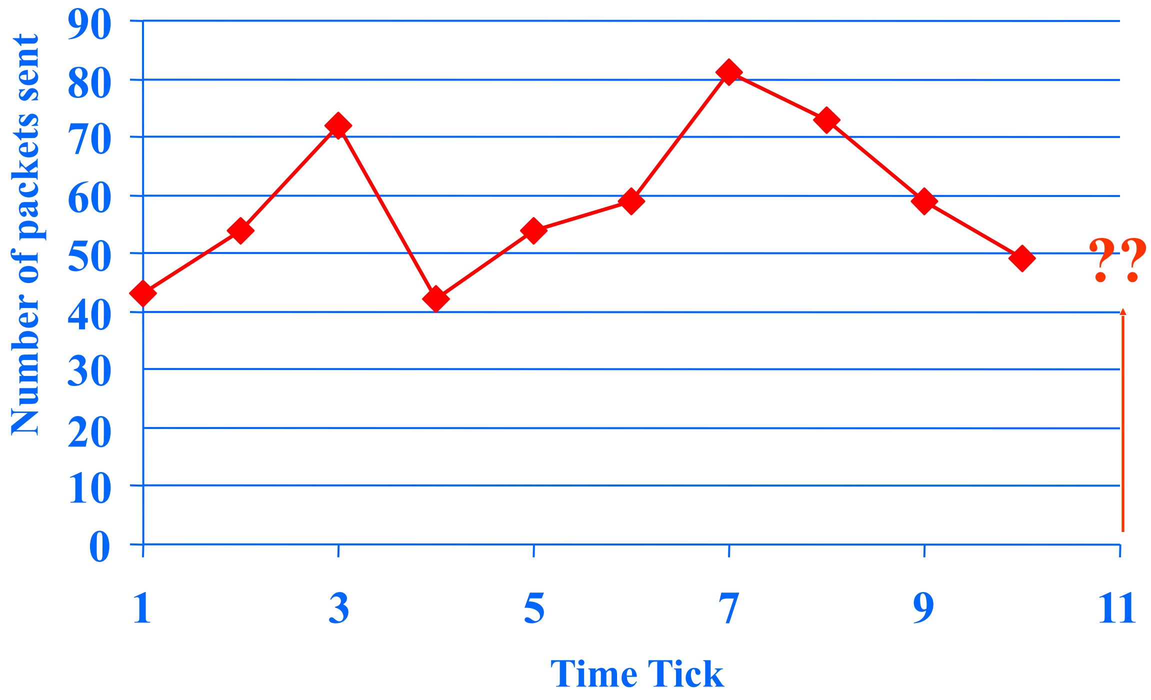


year

lynx caught per year
(packets per day;
temperature per day)

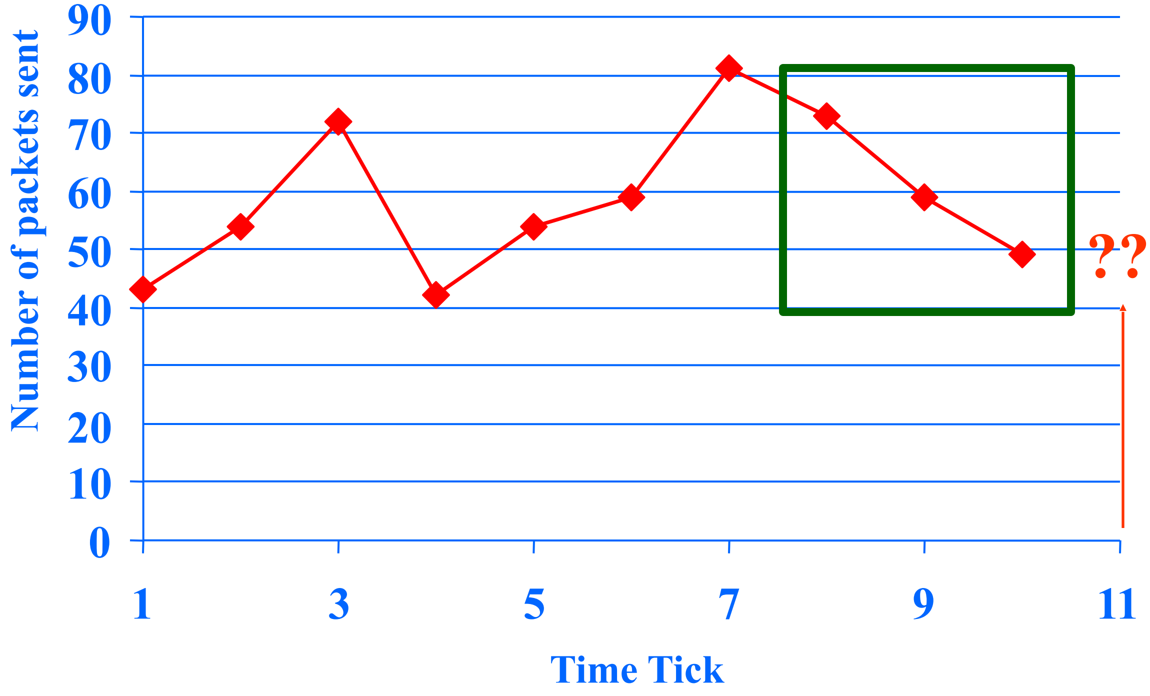
Problem#2: Forecast

Given x_t, x_{t-1}, …, forecast x_{t+1}



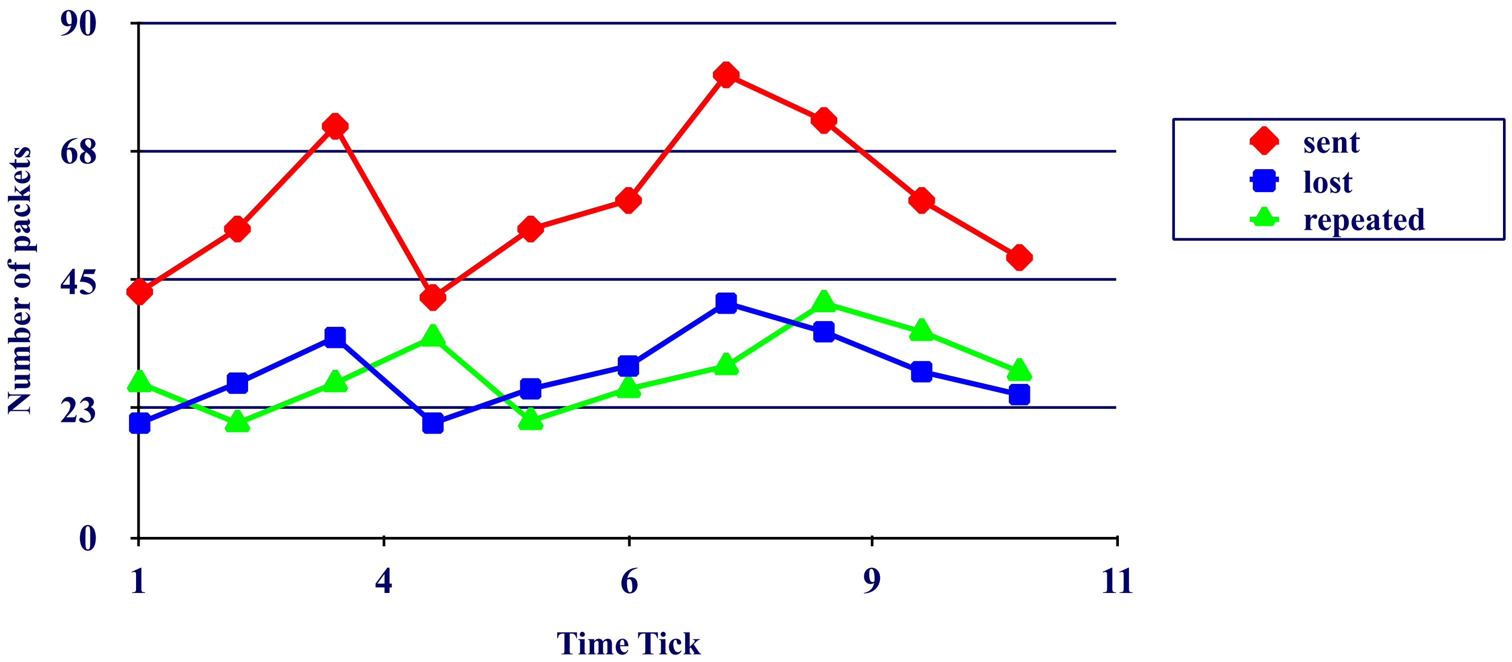
Problem#2': Similarity search

E.g.., Find a 3-tick pattern, similar to the last one



Problem #3:

- Given: A set of **correlated** time sequences
- Forecast 'Sent(t)'



Important observations

Patterns, rules, forecasting and similarity indexing are closely related:

- To do forecasting, we need
  - to find patterns/rules
  - to find similar settings in the past
- to find outliers, we need to have forecasts
  - (outlier = too far away from our forecast)

Outline

- Motivation
- Similarity Search and Indexing
- Linear Forecasting
- Non-linear forecasting
- Conclusions

Outline

- Motivation
- Similarity search and distance functions
  - Euclidean
  - Time-warping
- ...

Importance of distance functions

Subtle, but **absolutely necessary**:

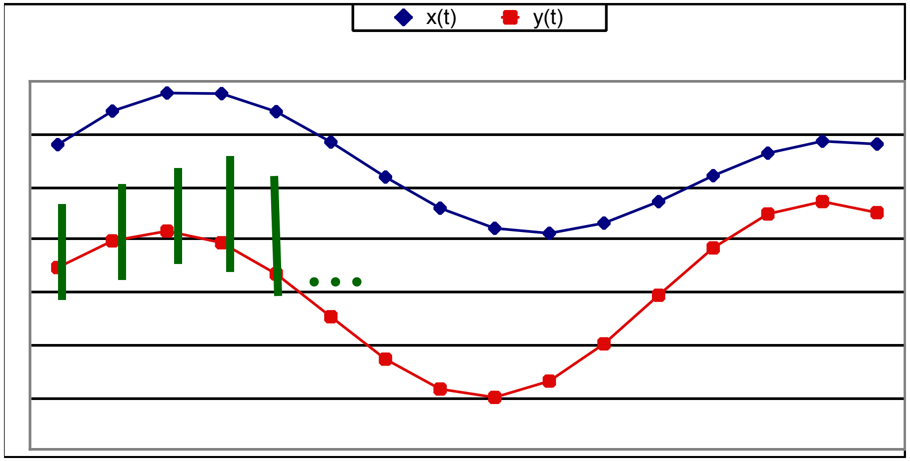
- A ‘must’ for similarity indexing (-> forecasting)
- A ‘must’ for clustering

Two major families

  – Euclidean and Lp norms

  – Time warping and variations

Euclidean and Lp



$$D(\vec{x},\vec{y}) = \sum_{i=1}^{n} (x_i - y_i)^2$$

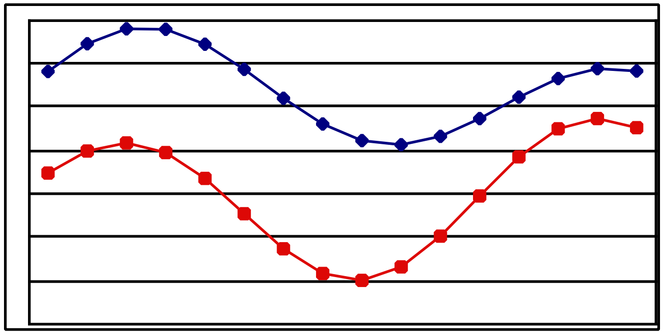
$$L_p(\vec{x},\vec{y}) = \sum_{i=1}^{n} | x_i - y_i |^p$$

L_1: city-block = Manhattan

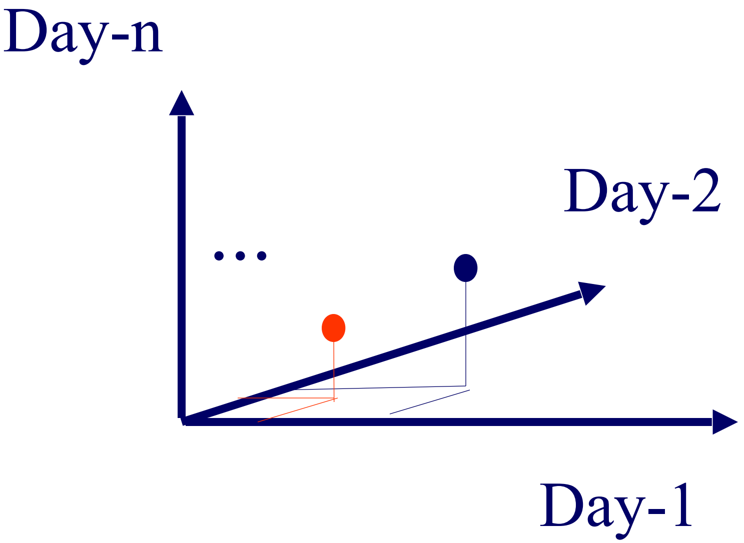
L_2 = Euclidean

L_∞

Observation #1



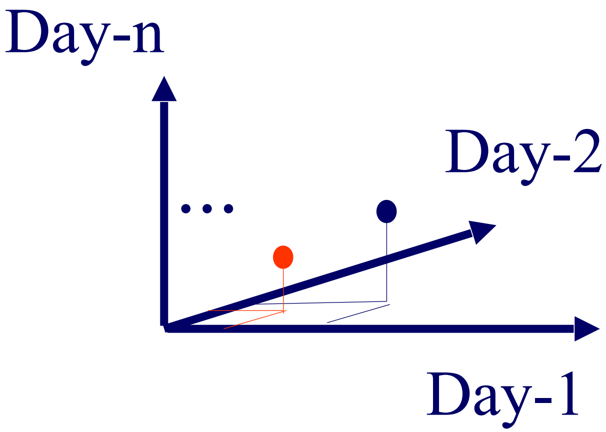
- **Time sequence -> n-d vector**

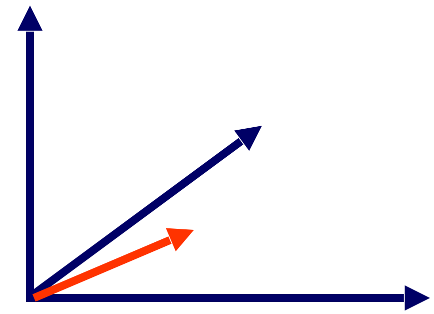


Observation #2

Euclidean distance is closely related to

- cosine similarity
- dot product
- ‘cross-correlation’ function

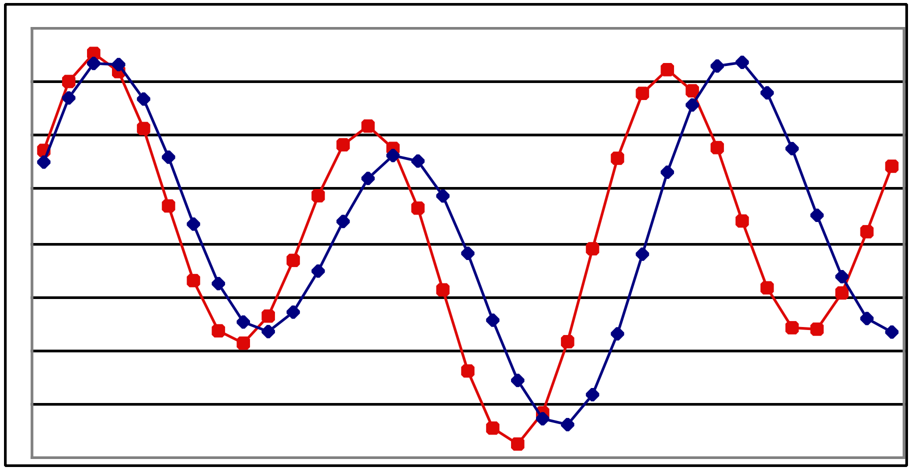




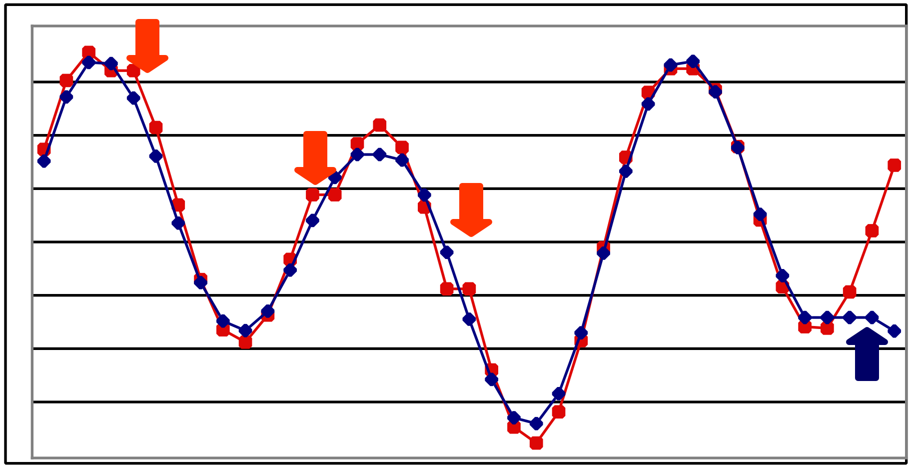
Time Warping

- allow accelerations - decelerations
  - (with or w/o penalty)
- THEN compute the (Euclidean) distance (+ penalty)
- related to the string-editing distance

Time Warping



‘stutters’:



Time warping

Q: how to compute it?

A: dynamic programming

$D(i, j)$ = cost to match prefix of length i of first sequence x with prefix of length j of second sequence y

Time warping

Thus, with no penalty for stutter, for sequences

$$x_1, x_2, \ldots, x_{i,;} \qquad y_1, y_2, \ldots, y_j$$

$$D(i,j) = \left\| x[i] - y[j] \right\| + \min \begin{cases} D(i-1, j-1) & \text{no stutter} \\ D(i, j-1) & \text{x-stutter} \\ D(i-1, j) & \text{y-stutter} \end{cases}$$

Time warping

VERY SIMILAR to the string-editing distance

$$D(i,j) = \left\| x[i] - y[j] \right\| + \min \begin{cases} D(i-1, j-1) & \text{no stutter} \\ D(i, j-1) & \text{x-stutter} \\ D(i-1, j) & \text{y-stutter} \end{cases}$$

Time warping

- Complexity: O(M*N) - quadratic on the length of the strings
- **<u>Many</u>** variations (penalty for stutters; limit on the number/percentage of stutters; …)
- popular in voice processing [Rabiner + Juang]

Other Distance functions

- piece-wise linear/flat approx.; compare pieces [Keogh+01] [Faloutsos+97]
- ‘cepstrum’ (for voice [Rabiner+Juang])
  - do DFT; take log of amplitude; do DFT again!
- Allow for small gaps [Agrawal+95]

See tutorial by [Gunopulos + Das, SIGMOD01]

Other Distance functions

- In [Keogh+, KDD’04]: parameter-free, MDL based

Conclusions

Prevailing distances:

- Euclidean and
- time-warping

Outline

- Motivation
- Similarity search and distance functions
- Linear Forecasting
- Non-linear forecasting
- Conclusions

Linear Forecasting

Forecasting

"Prediction is very difficult, especially about the future."

- Nils Bohr

Danish physicist and Nobel Prize laureate

Outline

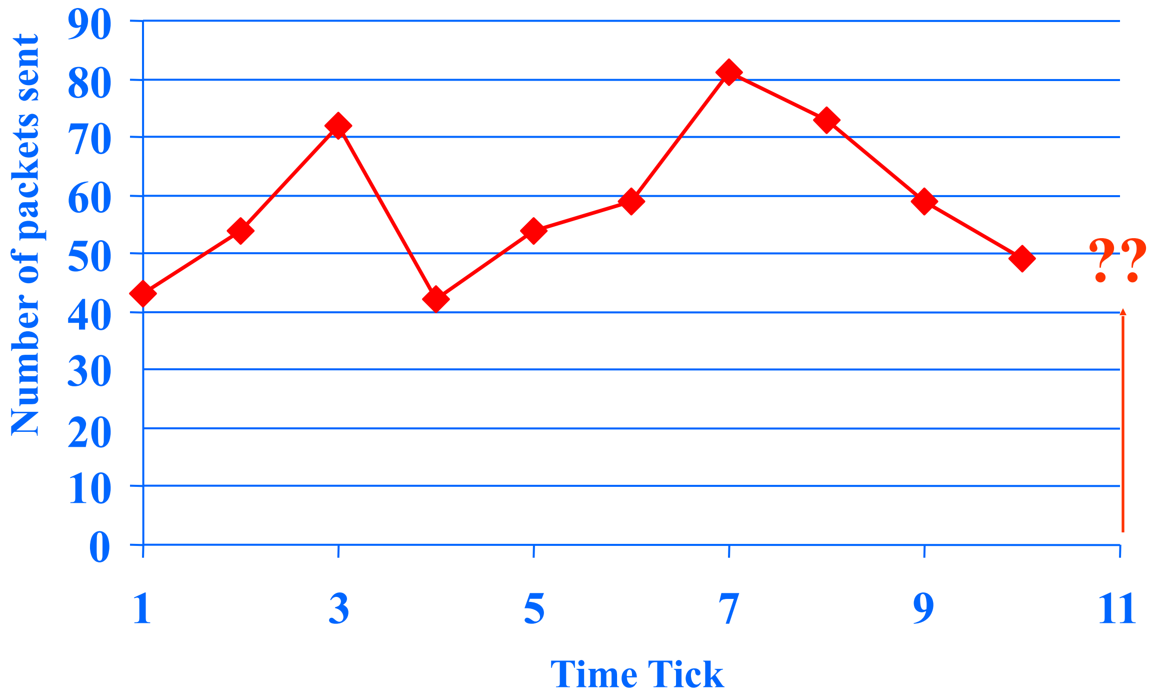
- Motivation
- ...
- Linear Forecasting
  - Auto-regression: Least Squares; RLS
  - Co-evolving time sequences
  - Examples
  - Conclusions

Reference

[Yi+00] Byoung-Kee Yi et al.: *Online Data Mining for Co-Evolving Time Sequences*, ICDE 2000. (Describes MUSCLES and Recursive Least Squares)

Problem#2: Forecast

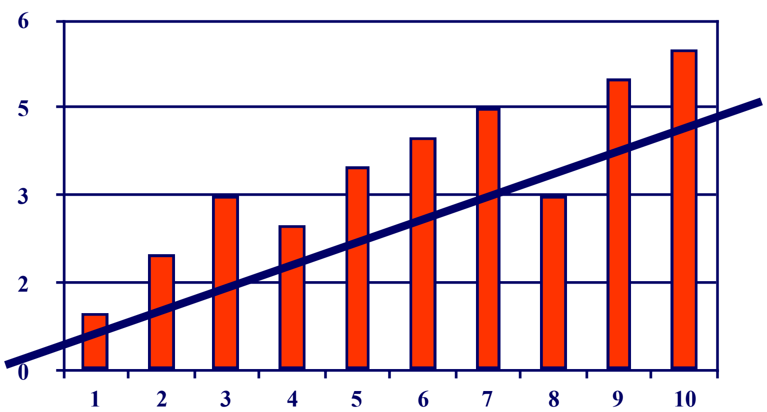
- Example: give x_{t-1}, x_{t-2}, …, forecast x_t



Forecasting: Preprocessing

MANUALLY:

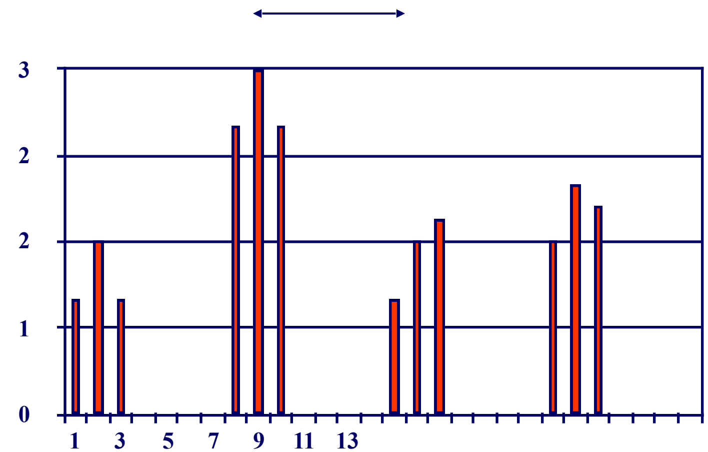
remove trends



time

spot periodicities

7 days



time

Problem#2: Forecast

- Solution: try to express

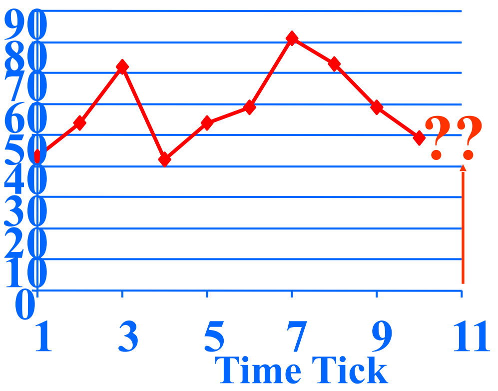
  x_t

  as a linear function of the past: x_{t-1}, x_{t-2}, …,

  (up to a window of w)

Formally:

$$\boxed{x_t \approx a_1 x_{t-1} + \ldots + a_w x_{t-w} + noise}$$



(Problem: Back-cast; interpolate)

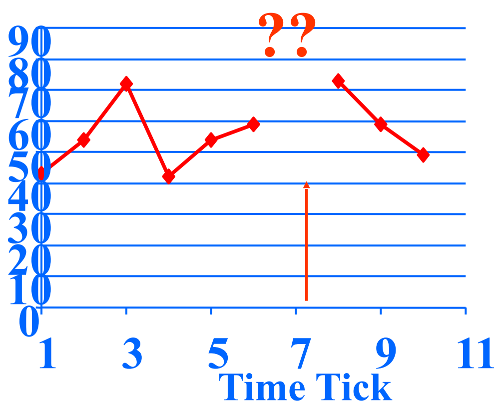
- Solution - interpolate: try to express

  x_t

  as a linear function of the past AND the future:

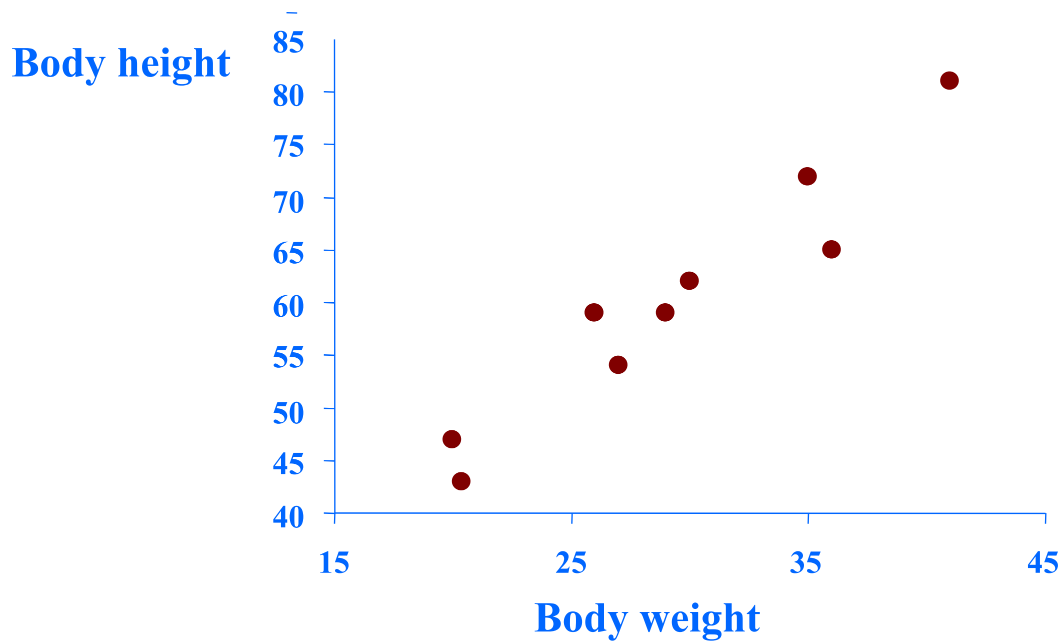
  $x_{t+1}, x_{t+2}, \ldots x_{t+wfuture;}\ x_{t-1}, \ldots x_{t-wpast}$

  (up to windows of w_{past} , w_{future})
- EXACTLY the same algo's



Refresher: Linear Regression

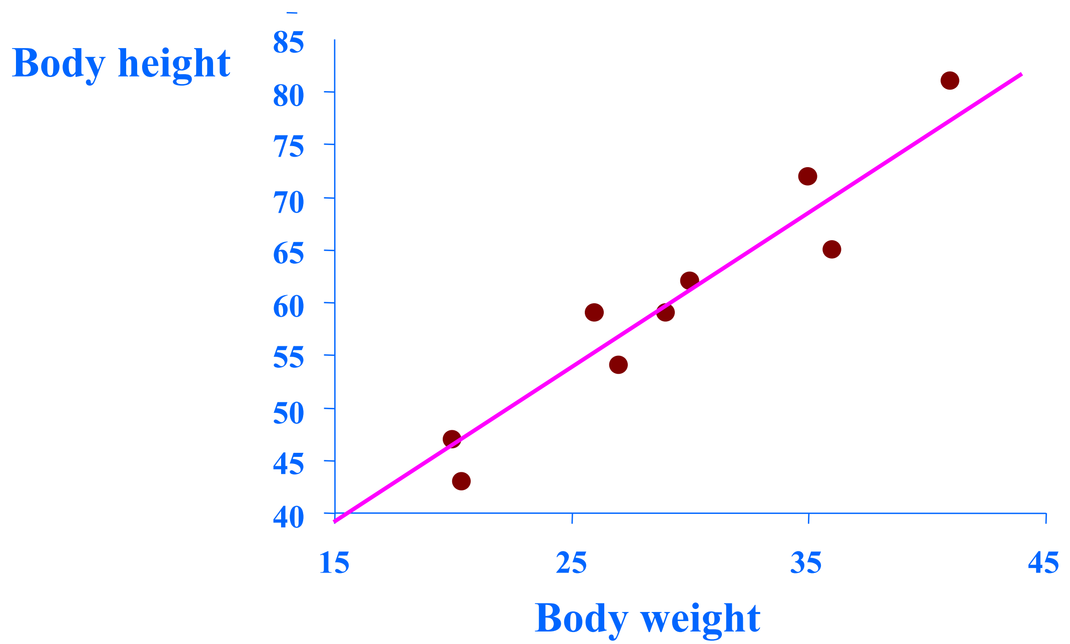
patient	*weight*	*height*
1	27	43
2	43	54
3	54	72
	…	
…		…
N	**25**	**??**



- express what we **don't know** (= "dependent variable")
- as a linear function of what we **know** (= "independent variable(s)")

Refresher: Linear Regression

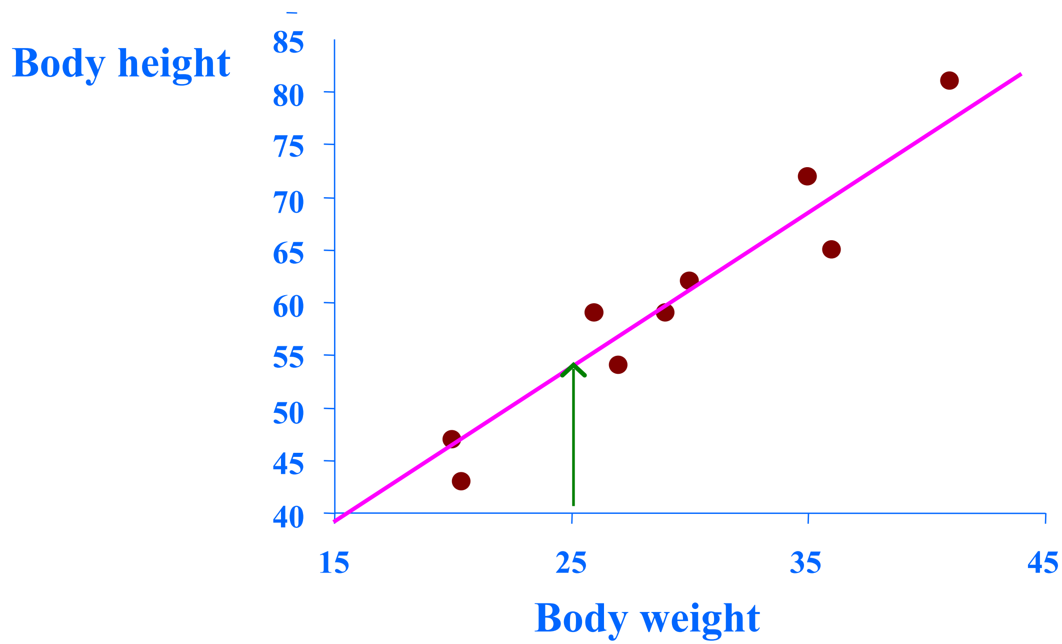
patient	*weight*	*height*
1	27	43
2	43	54
3	54	72
	…	
…		…
N	**25**	**??**



- express what we **don’t know** (= “dependent variable”)
- as a linear function of what we **know** (= “independent variable(s)”)

Refresher: Linear Regression

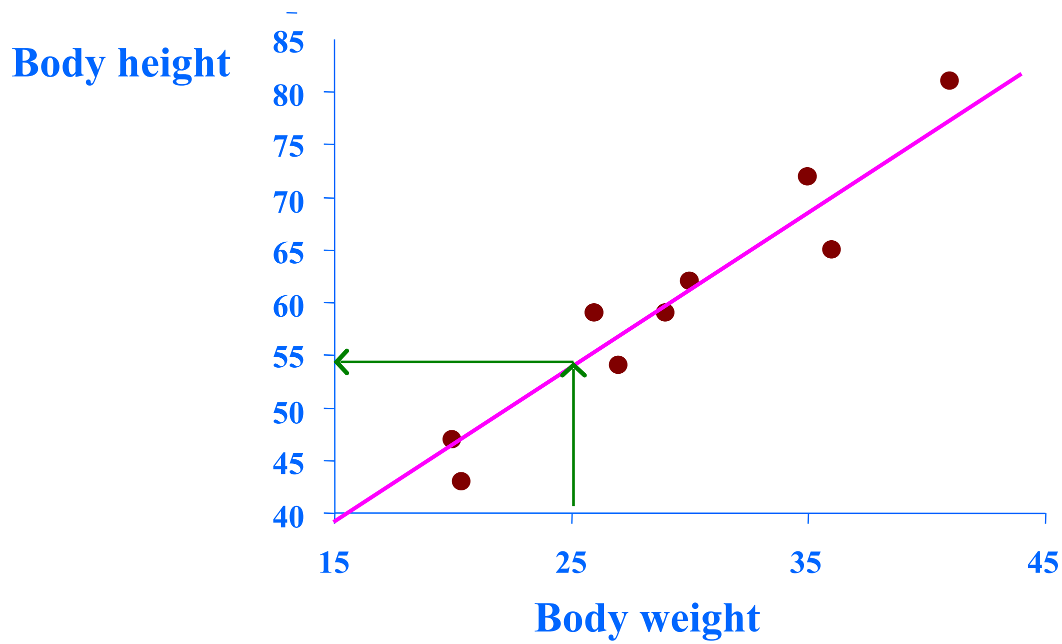
patient	*weight*	*height*
1	27	43
2	43	54
3	54	72
	…	
…		…
N	**25**	**??**



- express what we **don't know** (= "dependent variable")
- as a linear function of what we **know** (= "independent variable(s)")

Refresher: Linear Regression

patient	*weight*	*height*
1	27	43
2	43	54
3	54	72
	…	
…		…
N	**25**	**??**



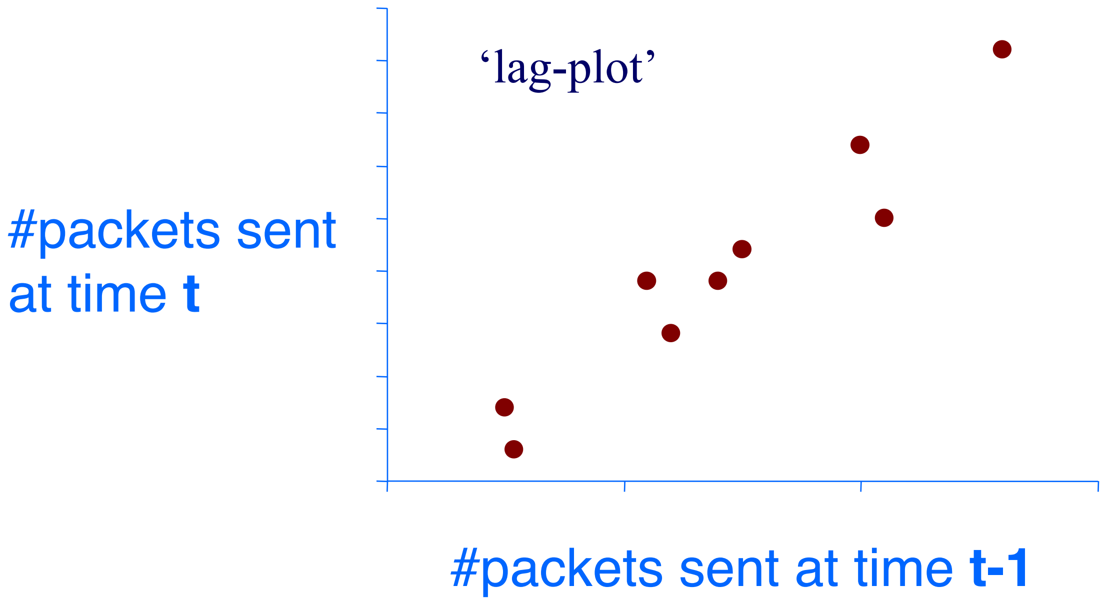
- express what we **don't know** (= "dependent variable")
- as a linear function of what we **know** (= "independent variable(s)")

Linear **Auto** Regression

Time	*Packets Sent(t)*
1	43
2	54
3	72
…	…
N	**??**

Linear Auto Regression

Time	*Packets Sent (t-1)*	*Packets Sent(t)*
1	-	43
2	43	54
3	54	72
	…	
…		…
N	**25**	**??**



Lag *w* = 1

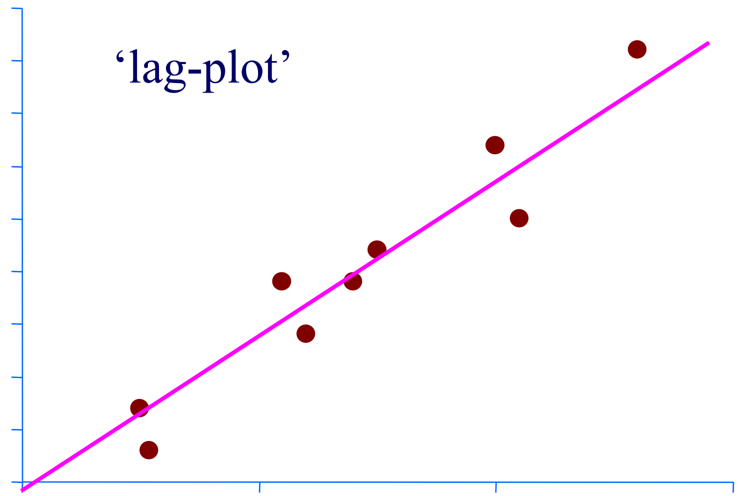
Dependent variable = # of packets sent (S [t])

Independent variable = # of packets sent (S[t-1])

Linear <u>Auto</u> Regression

Time	*Packets Sent (t-1)*	*Packets Sent(t)*
1	-	43
2	43	54
3	54	72
	…	
…		…
N	**25**	**??**

#packets sent at time **t**



#packets sent at time **t-1**

Lag *w* = 1

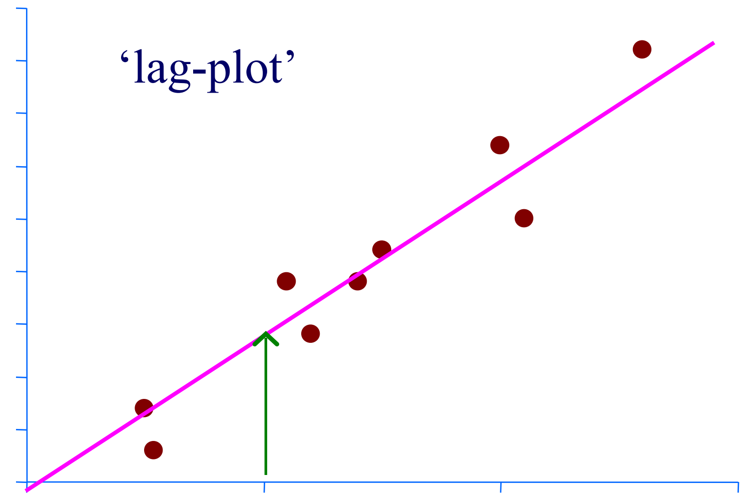
<u>Dependent</u> variable = # of packets sent (S [t])

<u>Independent</u> variable = # of packets sent (S[t-1])

Linear Auto Regression

Time	Packets Sent (t-1)	Packets Sent(t)
1	-	43
2	43	54
3	54	72
	…	
…		…
N	25	??

#packets sent at time **t**



#packets sent at time **t-1**

Lag *w* = 1

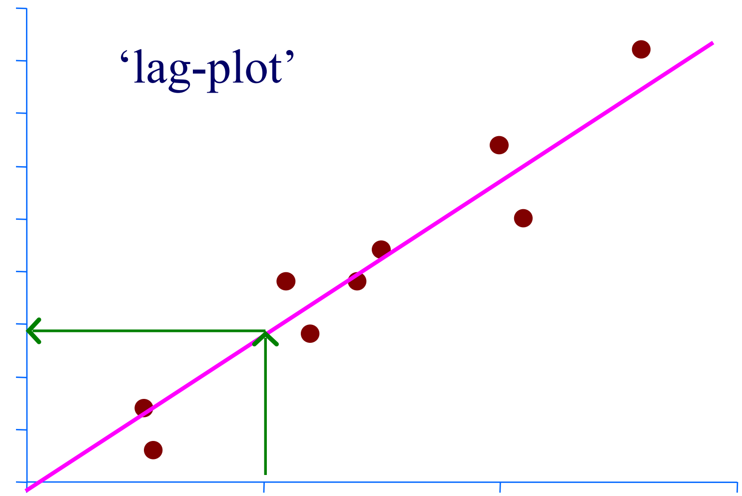
Dependent variable = # of packets sent (S [t])

Independent variable = # of packets sent (S[t-1])

Linear Auto Regression

Time	Packets Sent (t-1)	Packets Sent(t)
1	-	43
2	43	54
3	54	72
	…	
…		…
N	25	??

#packets sent at time **t**



#packets sent at time **t-1**

Lag $w = 1$

Dependent variable = # of packets sent (S [t])

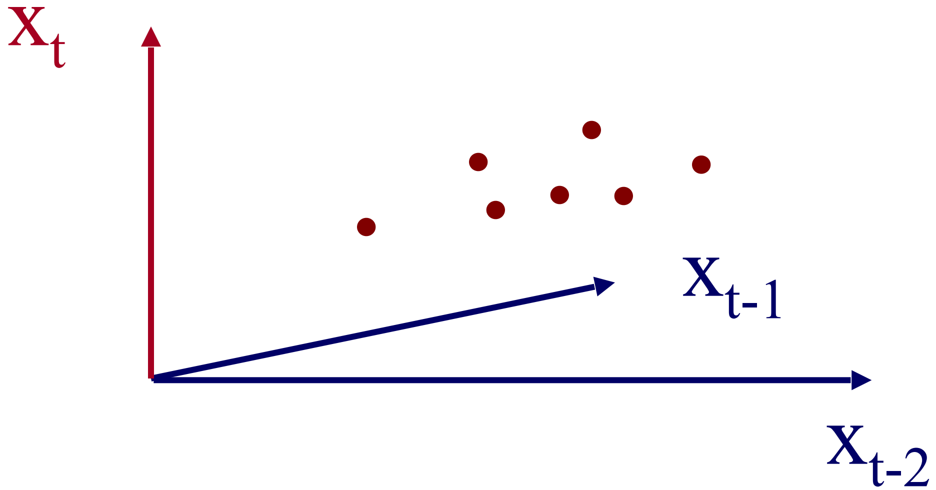
Independent variable = # of packets sent (S[t-1])

Outline

- Motivation
- ...
- Linear Forecasting
  - Auto-regression: **Least Squares; RLS**
  - Co-evolving time sequences
  - Examples
  - Conclusions

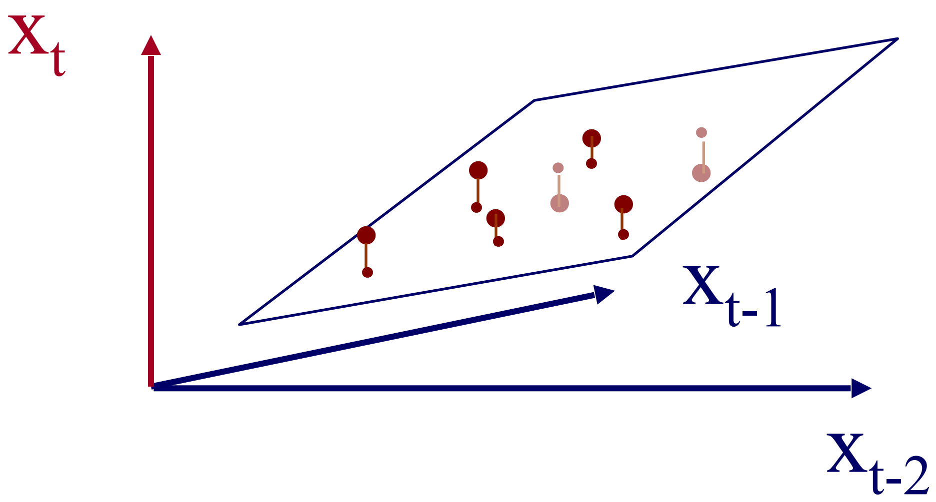
More details:

- Q1: Can it work with window $w > 1$?
- A1: YES!



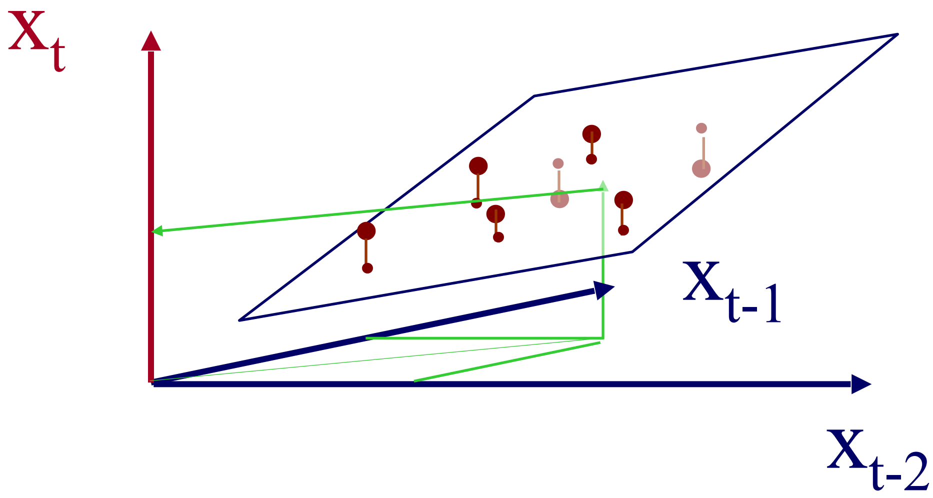
More details:

- Q1: Can it work with window $w > 1$?
- A1: YES! (we'll fit a hyper-plane, then!)



More details:

- Q1: Can it work with window $w > 1$?
- A1: YES! (we'll fit a hyper-plane, then!)



More details:

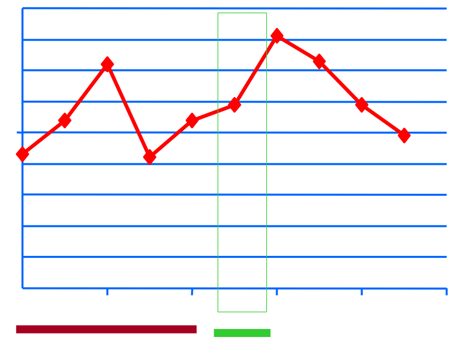
- Q1: Can it work with window $w > 1$?
- A1: YES! The problem becomes:

$$\mathbf{X}_{[N \times w]} \times \mathbf{a}_{[w \times 1]} = \mathbf{y}_{[N \times 1]}$$

- OVER-CONSTRAINED
  - **a** is the vector of the regression coefficients
  - **X** has the N values of the w indep. variables
  - **y** has the N values of the dependent variable

More details:

- $\mathbf{X}_{[N \times w]} \times \mathbf{a}_{[w \times 1]} = \mathbf{y}_{[N \times 1]}$



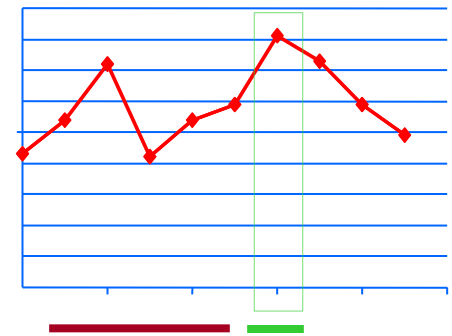
Ind-var1 Ind-var-w

time

$$\begin{bmatrix} X_{11}, X_{12}, \cdots, X_{1w} \\ X_{21}, X_{22}, \ldots, X_{2w} \\ \vdots \\ \vdots \\ \vdots \\ X_{N1}, X_{N2}, \ldots, X_{Nw} \end{bmatrix} \times \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_w \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ \vdots \\ \vdots \\ y_N \end{bmatrix}$$

More details:

- $\mathbf{X}_{[N \times w]} \times \mathbf{a}_{[w \times 1]} = \mathbf{y}_{[N \times 1]}$



Ind-var1 Ind-var-w

time

$$
\begin{bmatrix}
X_{11}, X_{12}, \cdots, X_{1w} \\
X_{21}, X_{22}, \ldots, X_{2w} \\
\vdots \\
\vdots \\
\vdots \\
X_{N1}, X_{N2}, \ldots, X_{Nw}
\end{bmatrix}
\times
\begin{bmatrix}
a_1 \\ a_2 \\ \vdots \\ a_w
\end{bmatrix}
=
\begin{bmatrix}
y_1 \\ y_2 \\ \vdots \\ \vdots \\ \vdots \\ y_N
\end{bmatrix}
$$

More details

- Q2: How to estimate $a_1, a_2, \ldots a_w = \mathbf{a}$?
- A2: with Least Squares fit

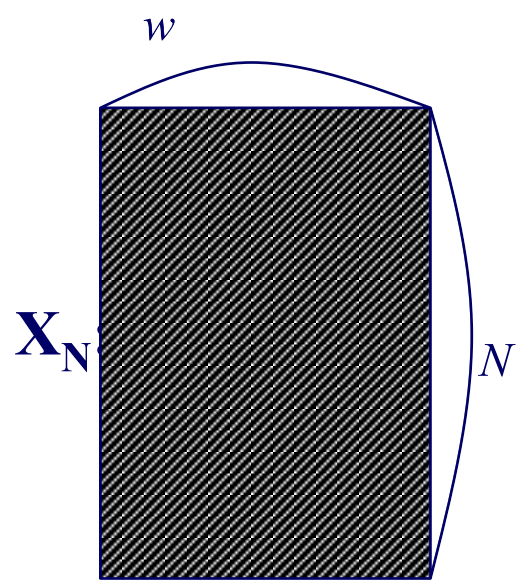
$$\boxed{\mathbf{a} = (\mathbf{X}^T \times \mathbf{X})^{-1} \times (\mathbf{X}^T \times \mathbf{y})}$$

- (Moore-Penrose pseudo-inverse)
- **a** is the vector that minimizes the RMSE from **y**

More details

- Straightforward solution:

$$\boxed{\mathbf{a} = (\mathbf{X}^T \times \mathbf{X})^{-1} \times (\mathbf{X}^T \times \mathbf{y})}$$



a : Regression Coeff. Vector
X : Sample Matrix

- Observations:
  - Sample matrix X grows over time
  - needs matrix inversion
  - $\mathbf{O}(N \times w^2)$ computation
  - $\mathbf{O}(N \times w)$ storage

Even more details

- Q3: Can we estimate **a** incrementally?
- A3: Yes, with the brilliant, classic method of "Recursive Least Squares" (RLS) (see, e.g., [Yi+00], for details).
- We can do the matrix inversion, WITHOUT inversion! (How is that possible?!)

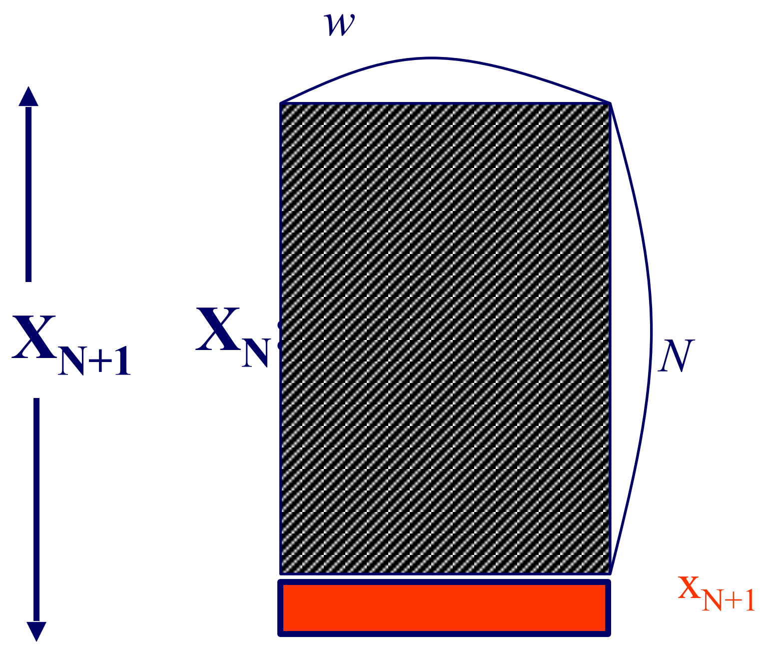
Even more details

- Q3: Can we estimate **a** incrementally?
- A3: Yes, with the brilliant, classic method of **“Recursive Least Squares” (RLS)** (see, e.g., [Yi+00], for details).
- We can do the matrix inversion, WITHOUT inversion! (How is that possible?!)
- A: our matrix has special form: $(X^T X)$

SKIP

More details

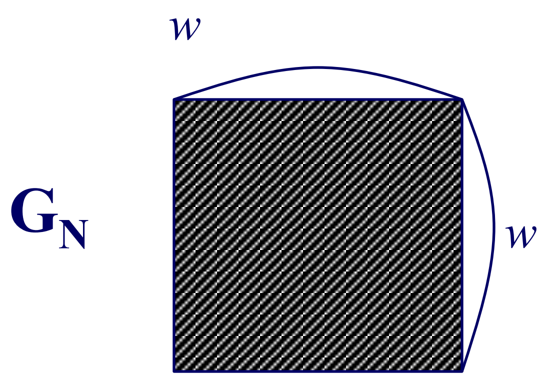
At the *N+1* time tick:



SKIP

More details: key ideas

- Let $\mathbf{G}_N = (\mathbf{X}_N^T \times \mathbf{X}_N)^{-1}$ ("gain matrix")
- $\mathbf{G}_{N+1}$ can be computed recursively from $\mathbf{G}_N$ without matrix inversion



Comparison:

- Straightforward Least Squares
  - Needs huge matrix (**growing** in size) $O(N \times w)$
  - Costly matrix operation $O(N \times w^2)$
- Recursive LS
  - Need much smaller, fixed size matrix $O(w \times w)$
  - Fast, incremental computation $O(1 \times w^2)$
  - **no matrix inversion**

$$\boxed{N = 10^6, \quad w = 1\text{-}100}$$

SKIP

EVEN more details:

$$G_{N+1} = G_N - [c]^{-1} \times [G_N \times x_{N+1}{}^T] \times x_{N+1} \times G_N$$

1 x *w* row vector

$$c = [1 + x_{N+1} \times G_N \times x_{N+1}{}^T]$$

Let's elaborate
(VERY IMPORTANT, VERY VALUABLE!)

SKIP

EVEN more details:

$$a = [X_{N+1}{}^{T} \times X_{N+1}]^{-1} \times [X_{N+1}{}^{T} \times y_{N+1}]$$

SKIP

EVEN more details:

$$a = [X_{N+1}{}^{T} \times X_{N+1}]^{-1} \times [X_{N+1}{}^{T} \times y_{N+1}]$$

[w x 1] [w x (N+1)] [(N+1) x w] [w x (N+1)] [(N+1) x 1]

SKIP

EVEN more details:

$$a = \boxed{[X_{N+1}{}^{T} \times X_{N+1}]^{-1}} \times [X_{N+1}{}^{T} \times y_{N+1}]$$

[w x (N+1)]　　　　[(N+1) x w]

SKIP

EVEN more details:

$$a = [X_{N+1}^{T} \times X_{N+1}]^{-1} \times [X_{N+1}^{T} \times y_{N+1}]$$

'gain matrix'

$$G_{N+1} \equiv [X_{N+1}^{T} \times X_{N+1}]^{-1}$$

$$\overset{\text{wxw}}{G_{N+1}} = \overset{\text{wxw}}{G_N} - \overset{\text{1x1}}{[c]^{-1}} \times [\overset{\text{wxw}}{G_N} \times \overset{\text{wx1}}{x_{N+1}^{T}}] \times \overset{\text{1xw}}{x_{N+1}} \times \overset{\text{wxw}}{G_N}$$

SCALAR!

$$c = [1 + x_{N+1} \times G_N \times x_{N+1}^{T}]$$

SKIP

Altogether:

$$G_0 \equiv \delta\, I$$

where
I: w x w identity matrix
δ: a large positive number

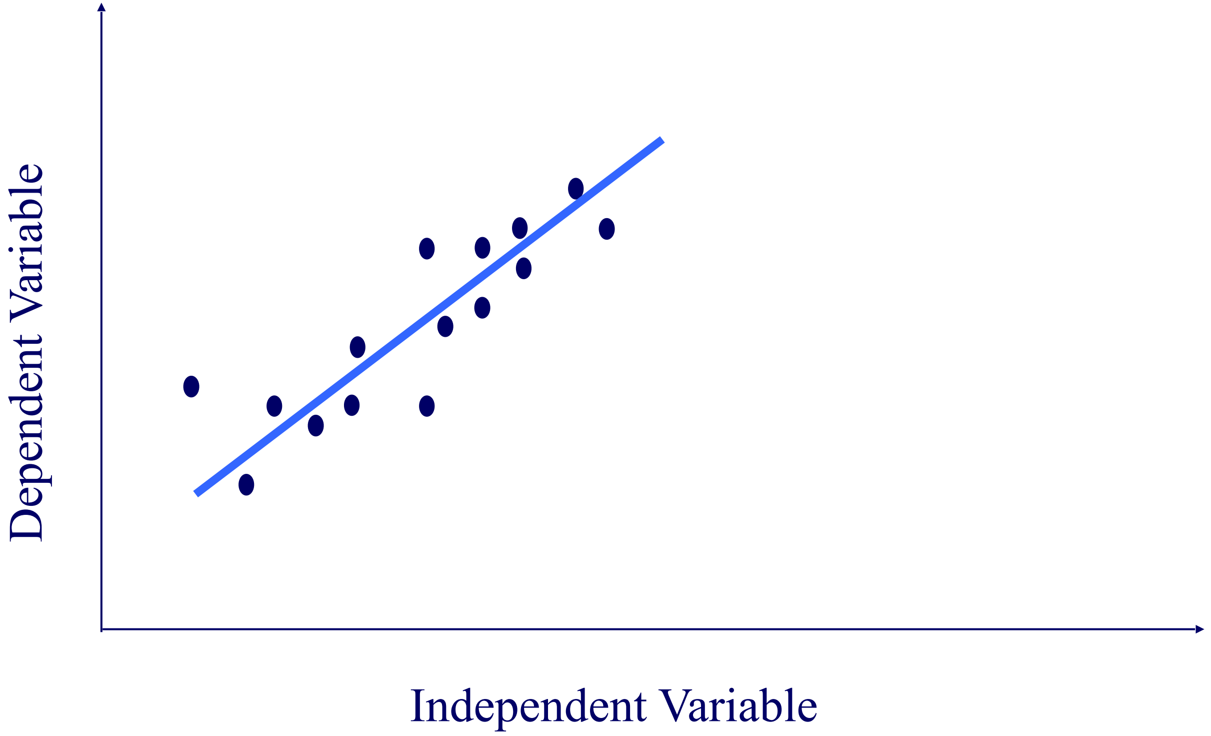
Comparison:

- Straightforward Least Squares
  - Needs huge matrix (**growing** in size) $O(N\times w)$
  - Costly matrix operation $O(N\times w^2)$
- Recursive LS
  - Need much smaller, fixed size matrix $O(w\times w)$
  - Fast, incremental computation $O(1\times w^2)$
  - **no matrix inversion**

$$\boxed{N = 10^6, \quad w = 1\text{-}100}$$

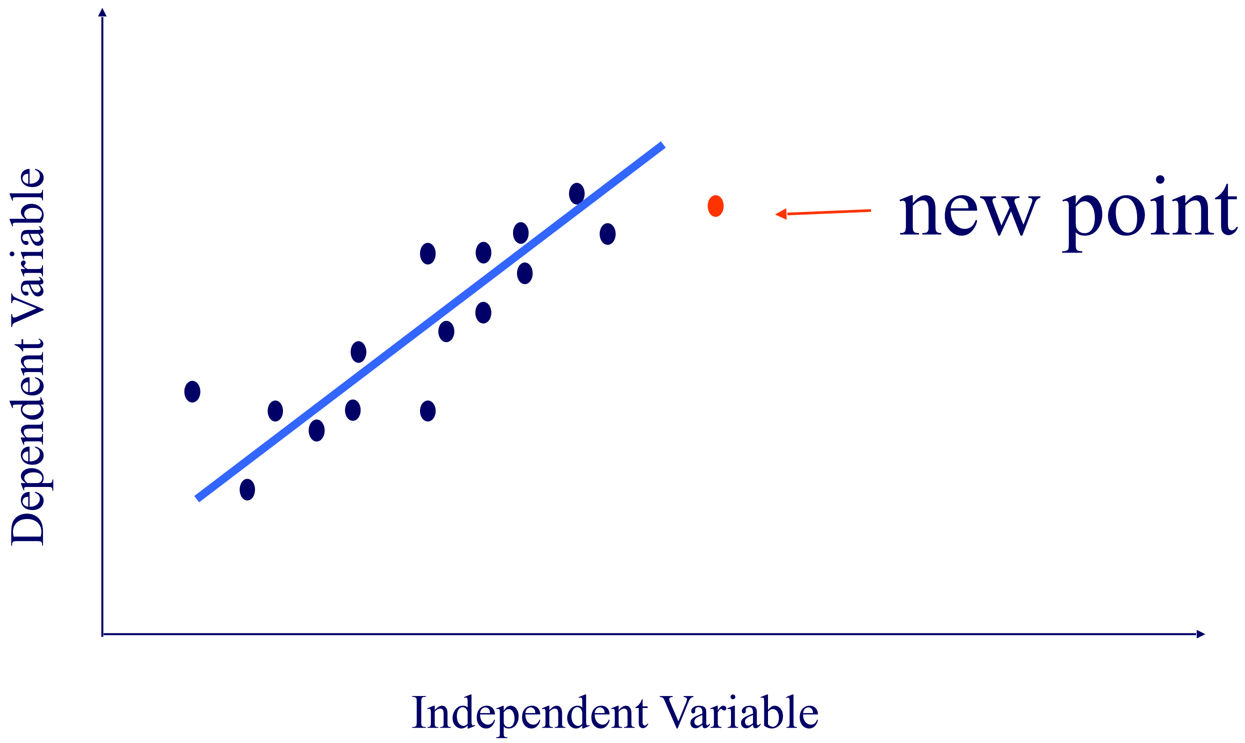
Pictorially:

- Given:



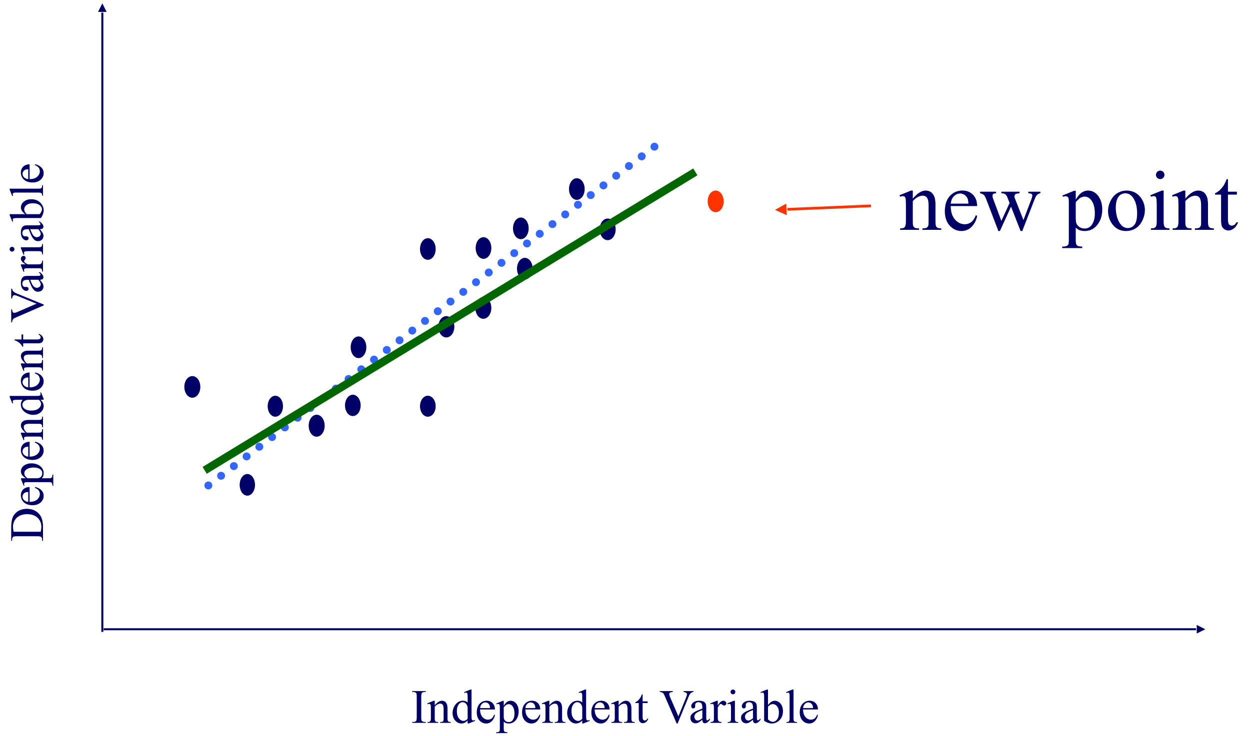
Pictorially:

•



Pictorially:

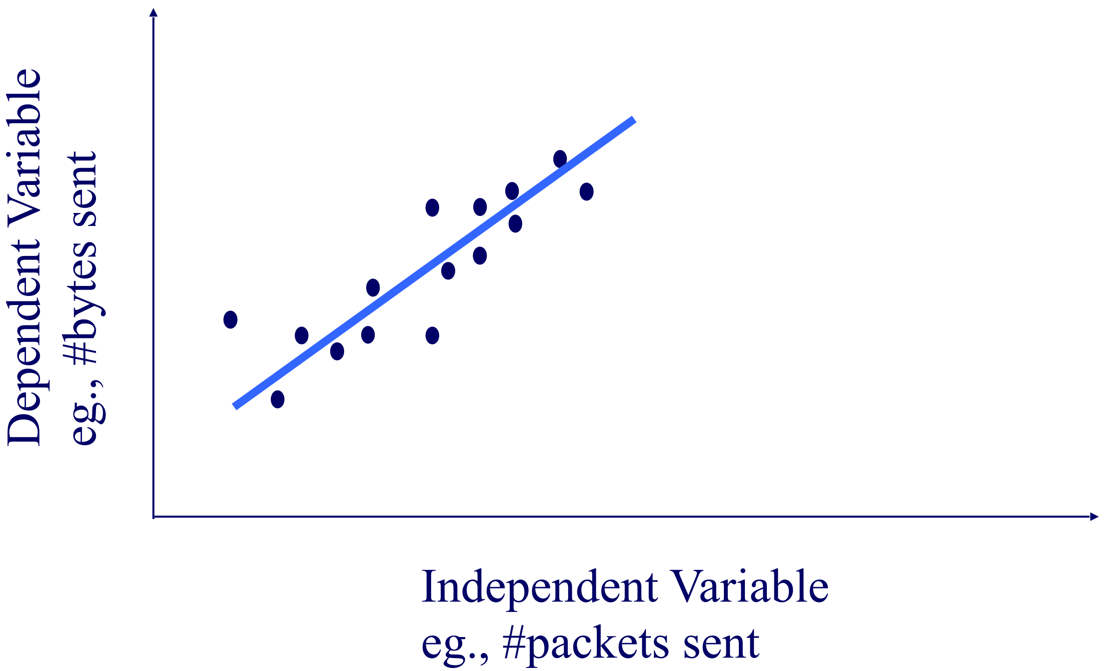
RLS: quickly compute new best fit



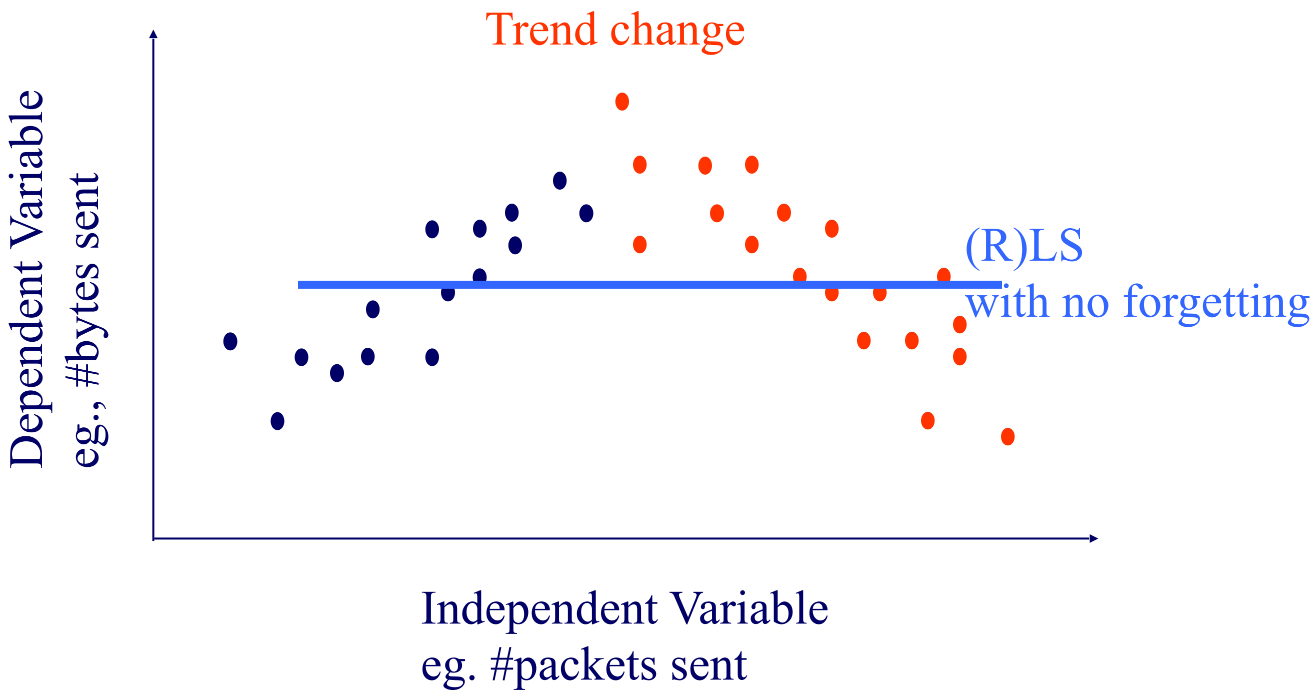
Even more details

- Q4: can we 'forget' the older samples?
- A4: Yes - RLS can easily handle that [Yi+00]:

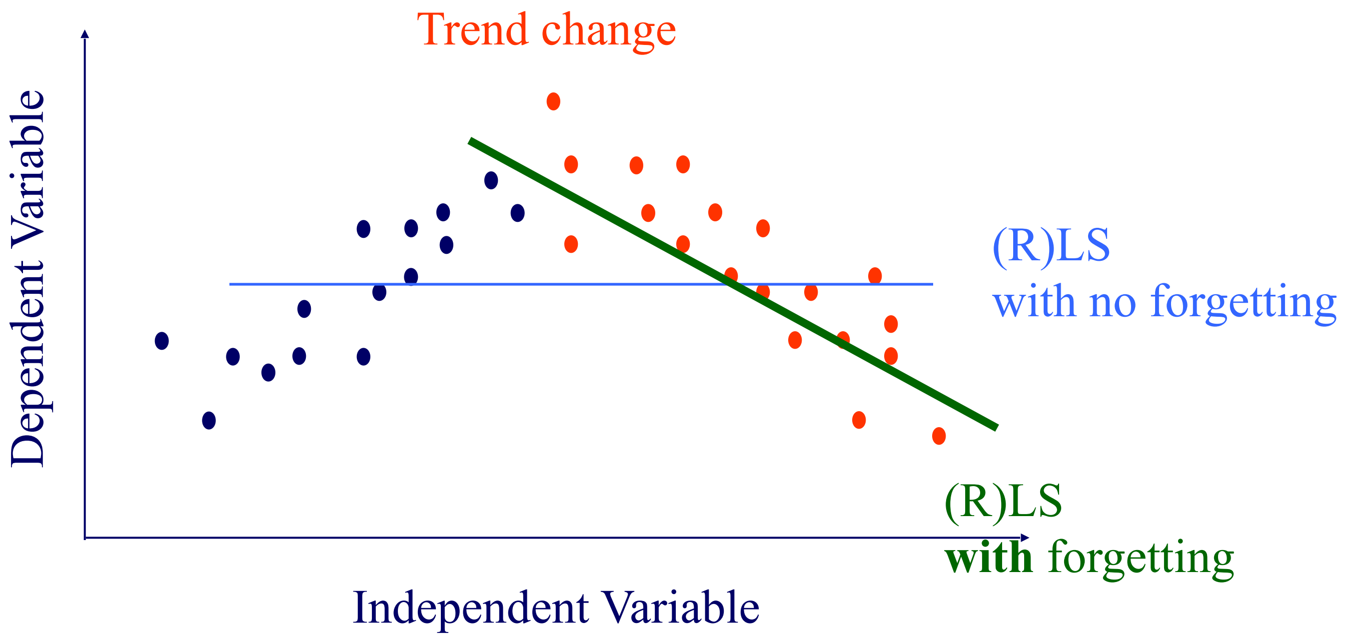
Adaptability - ‘forgetting’



Adaptability - ‘forgetting’



Adaptability - ‘forgetting’



• RLS: can *trivially* handle ‘forgetting’