Outline

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

• **Given**: one or more sequences
  
  \( x_1, x_2, \ldots, x_t, \ldots \)
  
  \( (y_1, y_2, \ldots, y_t, \ldots) \)
  
  \( (...) \)

• **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

Disk traffic

#bytes

time
Problem #1:

Goal: given a signal (e.g., #packets over time)
Find: patterns, periodicities, and/or compress lynx caught per year (packets per day; temperature per day)
Problem#2: Forecast

Given $x_t, x_{t-1}, \ldots$, 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 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 $L_p$

$L_1$: city-block = Manhattan
$L_2 = Euclidean$
$L_\infty$
Observation #1

Time sequence -> n-d vector
Observation #2

Euclidean distance is closely related to
– cosine similarity
– dot product
Time Warping

- allow accelerations - decelerations
  - (with or without penalty)
- THEN compute the (Euclidean) distance (+ penalty)
- related to the string-editing distance
Time Warping

‘stutters’: 

---

The text in the image is about time warping, which is a phenomenon observed in waveforms. The diagrams illustrate this concept, showing how certain segments of the waveform appear to be delayed or sped up, creating a stuttering effect. The arrows indicate the points of delay or acceleration in the waveform.
Time warping

Q: how to compute it?
A: dynamic programming

\[ D(i, j) = \text{cost to match} \]

prefix of length \( i \) of first sequence \( x \) with prefix of length \( j \) of second sequence \( y \)
Time warping
Time warping

Thus, with no penalty for stutter, for sequences

\[ x_1, x_2, \ldots, x_i,; \quad y_1, y_2, \ldots, y_j \]

https://nipunbatra.github.io/blog/2014/dtw.html
Time warping

VERY SIMILAR to the string-editing distance

no stutter
x-stutter
y-stutter
Time warping

- Complexity: $O(M*N)$ - quadratic on the length of the strings
- Many 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
Outline

• Motivation
• ...
• Linear Forecasting
  – Auto-regression: Least Squares; RLS
  – Co-evolving time sequences
  – Examples
  – Conclusions
Problem#2: Forecast

• Example: give $x_{t-1}$, $x_{t-2}$, …, forecast $x_t$
Forecasting: Preprocessing

MANUALLY:
remove trends

spot periodicities

7 days

Problem#2: Forecast

• Solution: try to express $x_t$ as a linear function of the past: $x_{t-1}, x_{t-2}, \ldots$, (up to a window of $w$)

Formally:

$$x_t \approx a_1 x_{t-1} + \ldots + a_w x_{t-w} + noise$$
(Problem: Back-cast; interpolate)

- Solution - interpolate: try to express $x_t$ as a linear function of the past AND the future:
  $x_{t+1}, x_{t+2}, \ldots x_{t+w_{future}}; x_{t-1}, \ldots x_{t-w_{past}}$
  (up to windows of $w_{past}$, $w_{future}$)

- EXACTLY the same algo’s
Express what we **don’t know** (= “dependent variable”) as a linear function of what we **know** (= “independent variable(s)”).
Linear **Auto** Regression

<table>
<thead>
<tr>
<th>Packs</th>
<th>Sent(t)</th>
</tr>
</thead>
<tbody>
<tr>
<td>43</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td></td>
</tr>
<tr>
<td>72</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>??</td>
<td></td>
</tr>
</tbody>
</table>
Linear Auto Regression

Lag $w = 1$

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

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

$$X_{[N \times w]} \times a_{[w \times 1]} = y_{[N \times 1]}$$

• 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:

- $X_{[N \times w]} \times a_{[w \times 1]} = y_{[N \times 1]}$

Ind-vari1  Ind-vari-w

time
More details:

- \( X_{[N \times w]} \times a_{[w \times 1]} = y_{[N \times 1]} \)

Ind-var1

Ind-var-w

time
More details

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

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

• (Moore-Penrose pseudo-inverse)
• $a$ is the vector that minimizes the RMSE from $y$
More details

• Straightforward solution:

\[
a = (X^T \times X)^{-1} \times (X^T \times y)
\]

- \(a\) : Regression Coeff. Vector
- \(X\) : Sample Matrix

• Observations:
  – Sample matrix \(X\) grows over time
  – needs matrix inversion
  – \(O(N \times w^2)\) computation
  – \(O(N \times w)\) 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: \((X^T X)\)
At the $N+1$ time tick:
More details: key ideas

- Let $G_N = (X_N^T \times X_N)^{-1}$ ("gain matrix")
- $G_{N+1}$ can be computed recursively from $G_N$ 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

\[N = 10^6, \quad w = 1-100\]
EVEN more details:

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

*I x w* row vector
EVEN more details:
EVEN more details:

[w x 1]  [w x (N+1)]  [(N+1) x w]  [w x (N+1)]  [(N+1) x 1]
EVEN more details:

\[ [w \times (N+1)] \quad [(N+1) \times w] \]
EVEN more details:

‘gain matrix’

wxw  wxw  1x1  wxw  wx1  lwx  wxw

SCALAR!
Altogether:

where
I: \( w \times w \) identity matrix
\( \delta \): 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

\[ N = 10^6, \quad w = 1-100 \]
Pictorially:

• Given:
Pictorially:

new point
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’

Independent Variable
eg., #packets sent

Dependent Variable
eg., #bytes sent
Adaptability - ‘forgetting’

Trend change

(R)LS with no forgetting

Dependent Variable
eg., #bytes sent

Independent Variable
eg. #packets sent
Adaptability - ‘forgetting’

Trend change

• RLS: can *trivially* handle ‘forgetting’