université PARIS-SACLAY

TIME SERIES :

MACHINE LEARNING, TRAPS & SIGNAL CLASSIFICATION

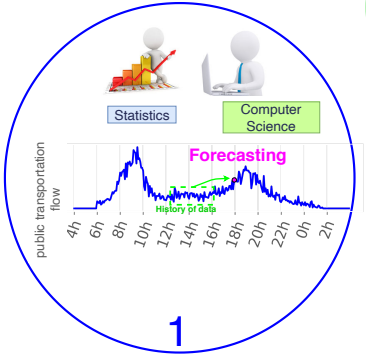
Vincent Guigue

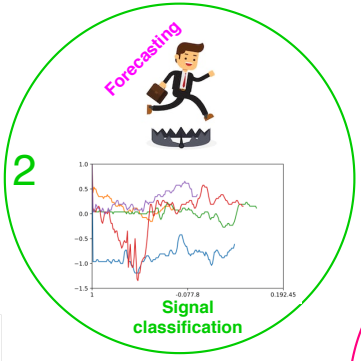
MIA PARIS-SACLAY
EKINOCS

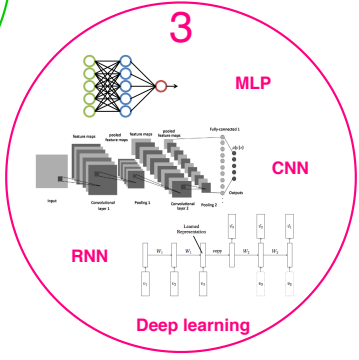
AGROPARISTECH
Institut des Sciences et Industries du Vivant et de l'Environnement

Introduction

Courses organization







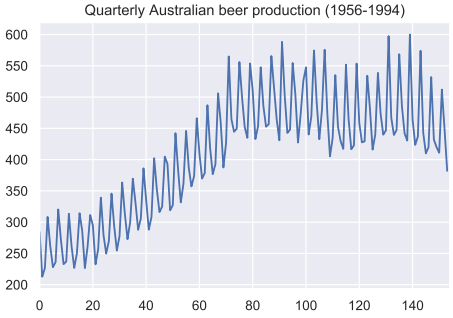
Schedule for course 2

1. Pitfalls in time series forecasting
   - What may (& will) happen to you while manipulating time series
2. Signal classification
   - Look like a standard ML problem...
   - ... But some specificities should be highlighted

Pitfalls in forecasting

Case 1: trend change

Difficult case: what happen when the test distribution diverges from the training one?



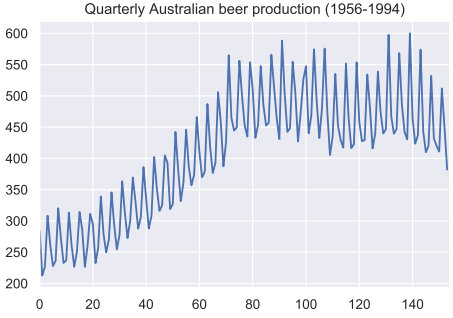
Predicting on long term basis with SARIMA:
trend + season

Autralian beer production: typical series with abrupt change

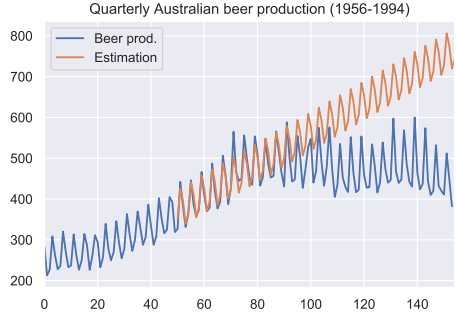
Time Series Nested Cross-Validation, Courtney Cochrane
https://towardsdatascience.com/time-series-nested-cross-validation-76adba623eb9

Case 1: trend change

Difficult case: what happen when the test distribution diverges from the training one?



Autralian beer production: typical series with abrupt change



Predicting on long term basis with SARIMA:
trend + season

Time Series Nested Cross-Validation, Courtney Cochrane
https://towardsdatascience.com/time-series-nested-cross-validation-76adba623eb9

Case 1: Detection & Resolution

\+ A good example: trend may change abruptly in many situations

− Who has ever use the same (SARIMA) model 10 years long without:

- retraining
- feeding the model with real data regularly

Detection

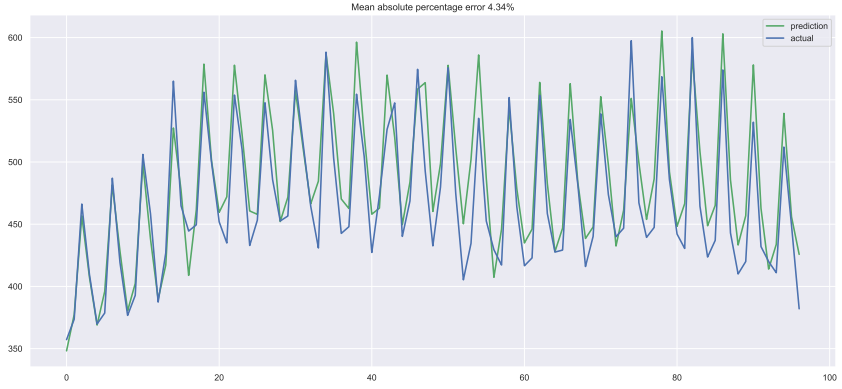
- Human monitoring
- Anomaly detection
- Change detection

Resolution

- Re-train models regularly
- Evaluate your model on recent Data (even in production)
  - Perform anomaly detection, raise alarms

Case 1: getting closer to real case

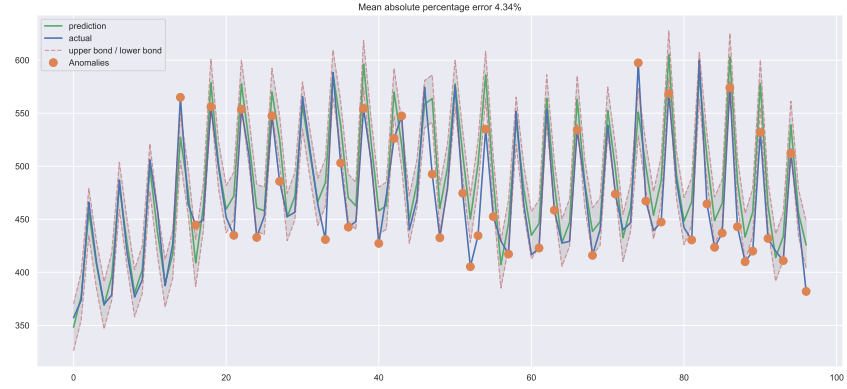
- Keep the same model on a long term basis...
- ... But feed it with new data at each time step



Difficult to estimate the quality of the prediction

Case 1: getting closer to real case

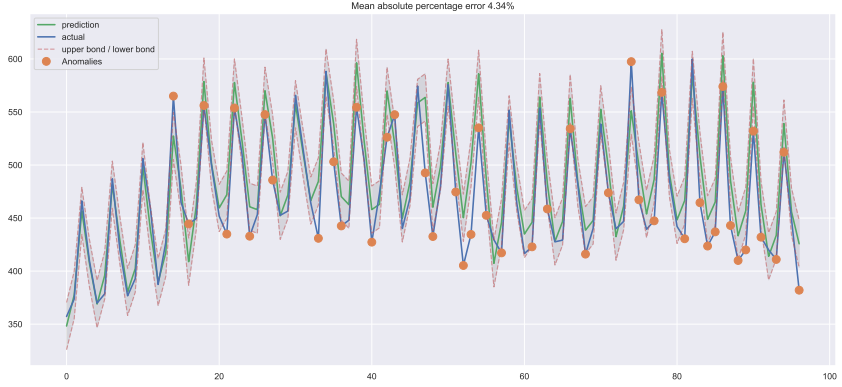
- Keep the same model on a long term basis...
- ... But feed it with new data at each time step



$\Rightarrow$ Adding confidence bounds & alarm detection

Case 1: getting closer to real case

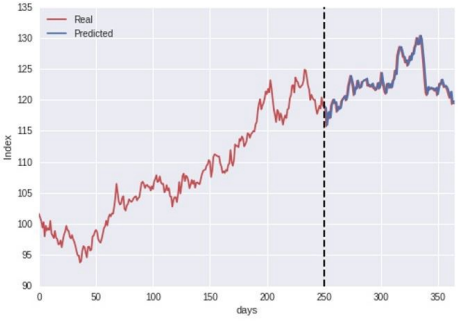
- Keep the same model on a long term basis...
- ... But feed it with new data at each time step

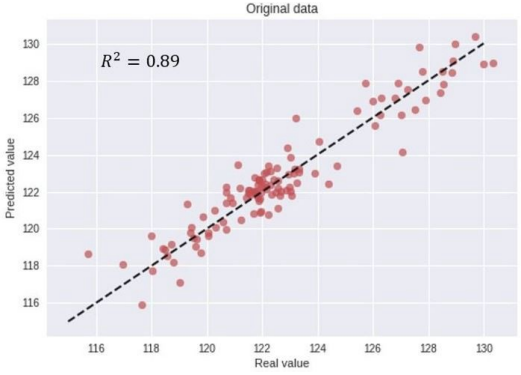


Alterative: Monitoring the results of a strong baseline

$\Rightarrow$ Results will converge between advanced model & baseline

Case 2 : a pretty good forecast





When you look at it from afar

How (not) to use Machine Learning for time series forecasting: Avoiding the pitfalls, Vegard Flovik
`https://towardsdatascience.com/how-not-to-use-machine-learning-for-time-series-forecasting-avo`

Case 2 : a pretty good forecast

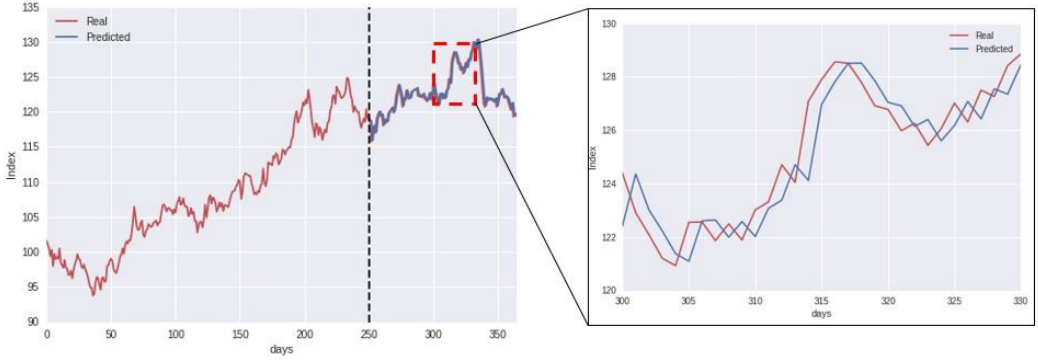
But actually:

- Data are generated from a random walk...

$\Rightarrow$ it can't be predicted !

How (not) to use Machine Learning for time series forecasting: Avoiding the pitfalls, Vegard Flovik
https://towardsdatascience.com/how-not-to-use-machine-learning-for-time-series-forecasting-avo

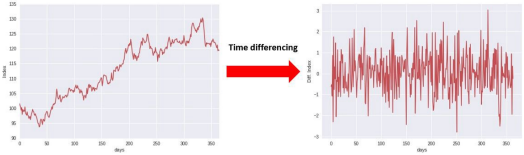
Case 2 : look at the details

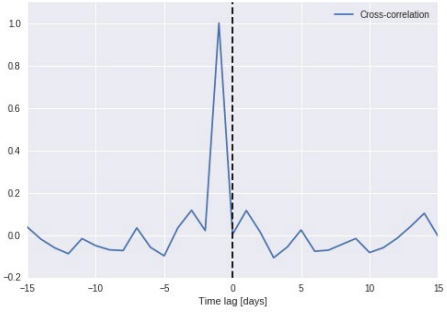


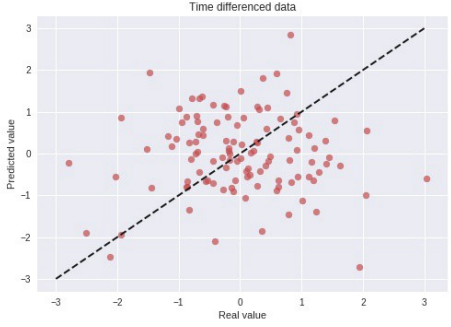
A very basic solution: $\hat{y}_t = y_{t-1}$

Case 2 : what was wrong? How to detect this case? [Statistics]

- Do not use ARMA on non stationary signals
- Measure cross-correlation between y and $\hat{y}$.

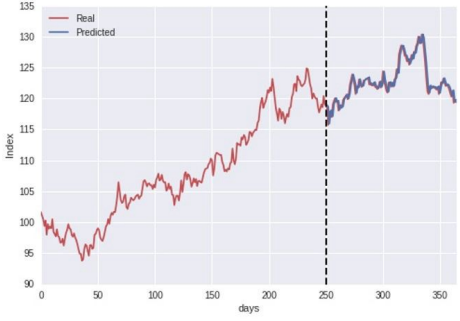






Case 2 : what would you do to detect this case? [Computer science]

Always compare a predictor to a strong baseline



- Choose the good baselines
  - Linear? $\Rightarrow$ not sufficient
  - Time series $\Rightarrow$ at least
    - Previous value: $\hat{y}_t = y_{t-1}$
    - Moving average: $\hat{y}_t = \frac{1}{T} \sum_i y_{t-i}$

1. Compare the results in CV...
2. And the standard deviation on the results.

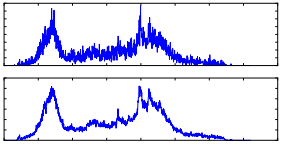
Baselines: a more comprehensive list

General baseline for time series:

- Previous value: $\hat{y}_t = y_{t-1}$
- Moving average: $\hat{y}_t = \frac{1}{T} \sum_i y_{t-i}$
- Predict always the most frequent value (rarely efficient on time series)

Seasonal data:

- Take one averaged season
  - e.g. in public transportation: average weekday or average Monday = prediction for every Monday
  - Strong baseline on specific dataset
  - **Very** strong baseline beyond $T + 1$



One Wednesday / averaged Wednesday on a particular subway station

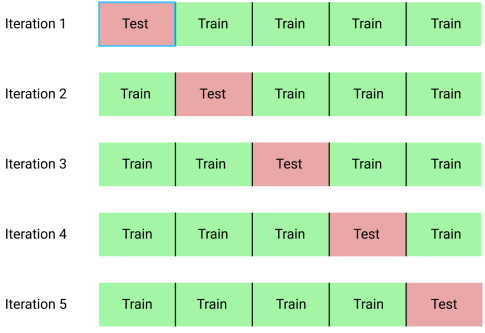
ML classical baseline

- Linear model
- K-nn : all $y_{t-1} = \alpha$ lead to $y_t = \beta$

Case 3 : In-sample evaluation

Classical block cross-validation = seeing the future

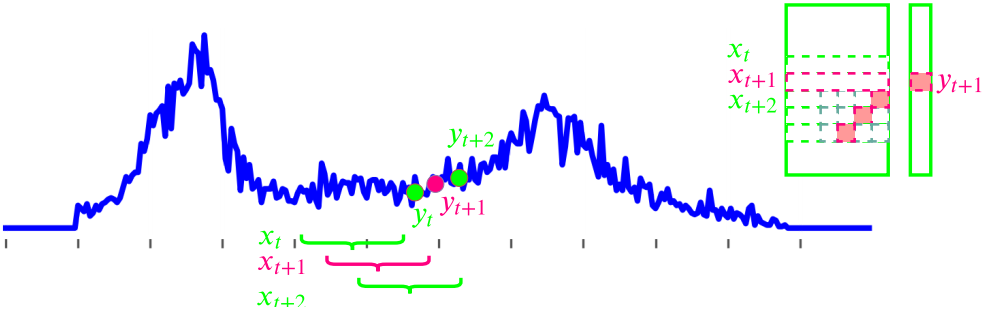
- cheating on any unpredictable (/ not modeled) trends
  - Difficult prediction problem turns into to a simpler interpolation problem
- side effect at the beginning/end of each test split (cf next slide)



Leonard J. Tashman, Int. Jour. of Forecasting, 2000
Out-of-sample tests of forecasting accuracy: an analysis and review

Case 3 : In-sample evaluation & information leak

Shuffled cross validation: test samples are surrounded by training ones

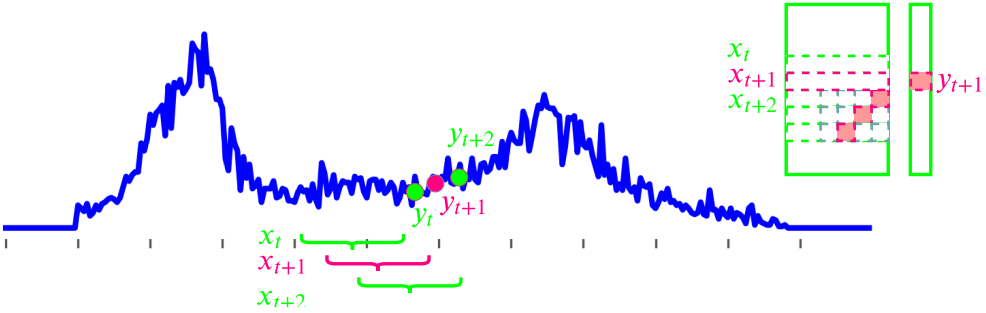


- Target x_{t+1}: find the most correlated points $\{x_c\}$ in the training set
- Add criterion to detect the better one regarding temporal evolution
- Predict $x_c^\star[end]$

$\Rightarrow$ it can even work to predict at $T + N$!

Case 3 : In-sample evaluation & information leak

Shuffled cross validation: test samples are surrounded by training ones



The only thing evaluated in this procedure is:

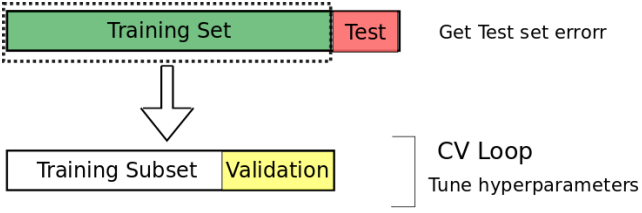
Is y_{t+1} close to y_t most of the time?

... And it is often the case !

$\Rightarrow$ You may obtain a prediction accuracy better that the intrinsic noise level!

Case 3 : In-sample / Out-of-sample evaluation [single series]

Switching to out-of-sample evaluation:

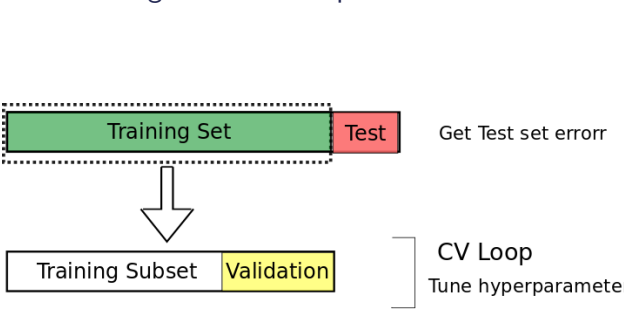


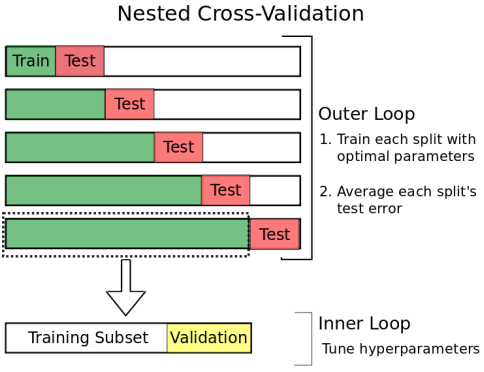
Note: to optimize your parameters, you have to use a validation set at the end of the training set.

Leonard J. Tashman, Int. Jour. of Forecasting, 2000
Out-of-sample tests of forecasting accuracy: an analysis and review

Case 3 : In-sample / Out-of-sample evaluation [single series]

Switching to out-of-sample evaluation:

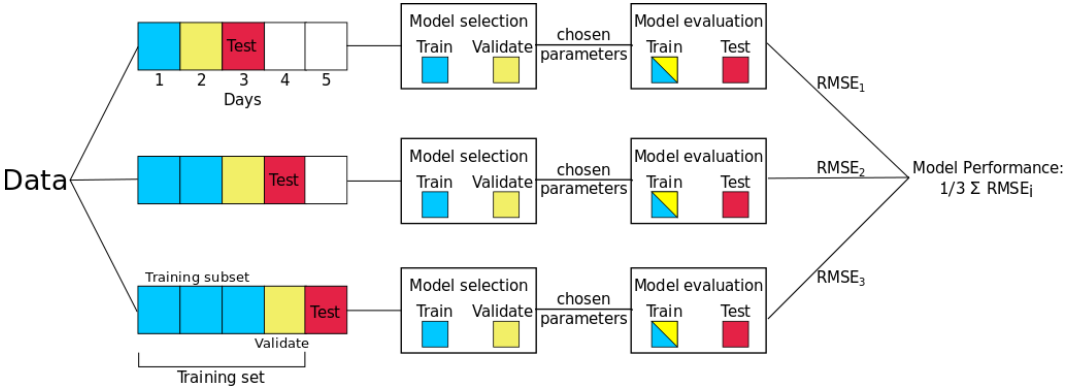




Note: to optimize your parameters, you have to use a validation set at the end of the training set.

Leonard J. Tashman, Int. Jour. of Forecasting, 2000
Out-of-sample tests of forecasting accuracy: an analysis and review

Case 3 : Complete Nested Cross-Validation procedure [single series]



Good news: it is already implemented in scikit-learn. $\Rightarrow$ just use it!

```
from sklearn.model_selection import TimeSeriesSplit
```

Case 3 : Nested Cross Validation over a population

Interdependent samples

e.g. evolution of the temperature in multiple cities / sales in different shops

- Apply the Nested CV on all samples
- Make sure that Train/test frontier corresponds to an absolute time-stamp

Independent samples

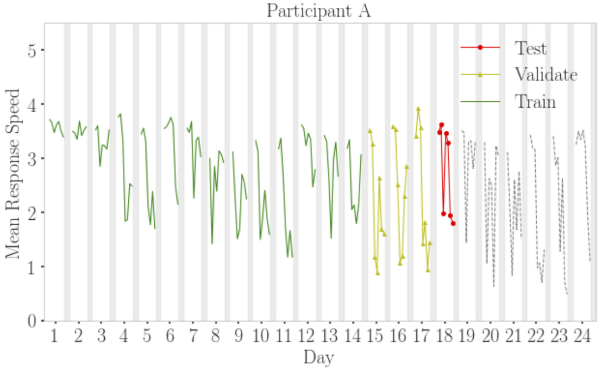
e.g. Patient response to a treatment

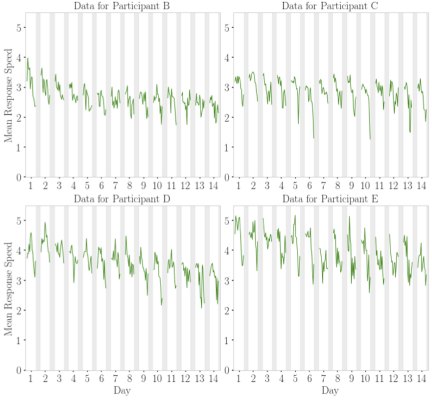
Case 3 : Nested Cross Validation over a population

Interdependent samples

Independent samples

e.g. Patient response to a treatment





Case 3 : Detect temporal information leak

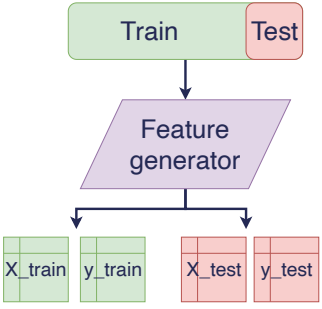
Nested cross validation **is supposed to** enable you to detect information leak...

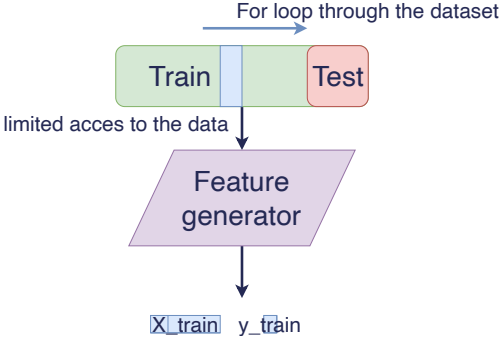
One case remains critical

Creating a **leaking feature** may be difficult to detect.

e.g.

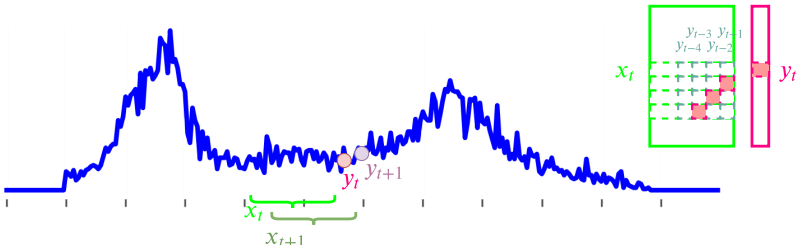
- An aggregation of values overflowing in the future of the sample
- Uncontrolled feature (from tsfresh)





⇒ Even if we don't like for loop, even if it is less convenient...

Case 4 : Normalization of the lag variables (not the contextual ones)

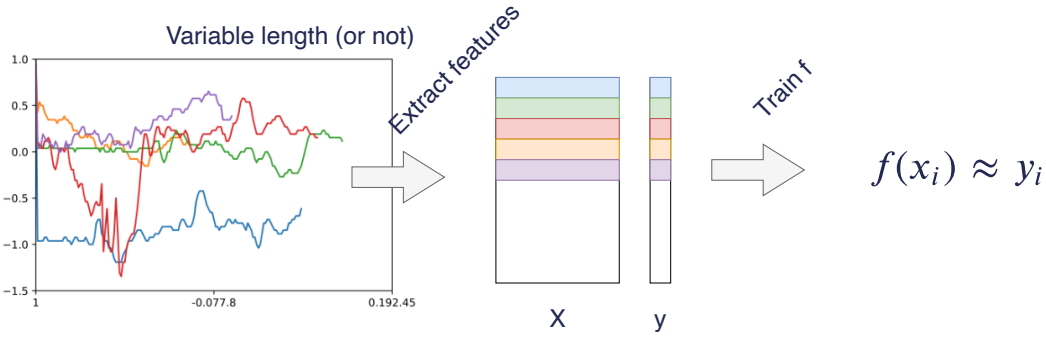


As in any machine learning use case, dealing with time series requires normalization:

- Normalization by columns
  - ⇒ Great impact of future measurements on the data
  - ⇒ Destruction of the temporal dependencies
- Normalization by line
  - Impact of the future measurements on the data
  - ... But limited impact
    - Normalizing by the max
    - Even better: normalizing by the 99% percentile
    - Very stable information that could have been given by an expert

Signal classification

A (very) different problem

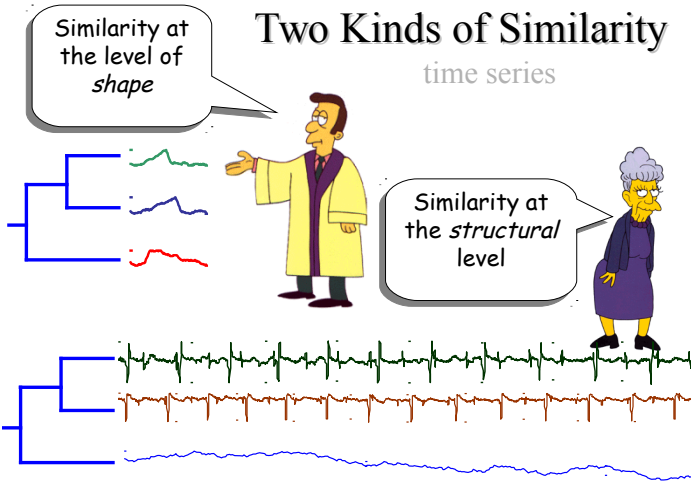


$\Rightarrow$ Much closer to traditional machine learning

Classifier = defining a similarity

How to compute a **relevant similarity/distance** between signals?

Two main kinds of similarities



Keogh Eammonn, Tutorial, 2019
Introduction to time series mining

- Long signals : standard descriptors
- Short signals : more likely to depends on the shape

Long signal processing chain

Feature engineering

- Depending on the application ⇒ discussions with experts
- Local statistics computations
  - Statistic moment
  - Frequency / power spectral density...
  - `tsfresh`
    - Hundreds of features...
    - ... & test of relevant ones

Classification

Whatever the length of the signal ⇒ constant representation

Standard ML similarities

- no more signal specificity

Standard ML (XGboost...)

Short & variable length signals: the weakness of Euclidian metric

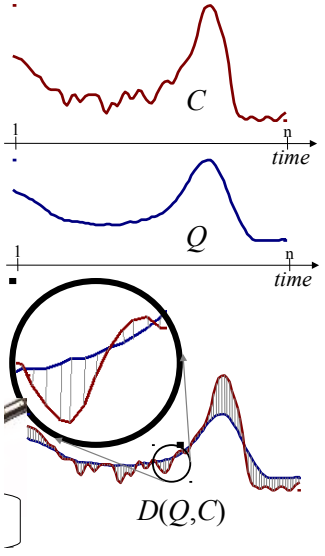
Euclidian distance:

$$d(C, Q) = \sqrt{\sum_t (C_t - Q_t)^2}$$

- Not adapted to variable length signal

  padding / cutting
- Not robust to noise
- (Definitely) not robust to shape variation/shift/scale...

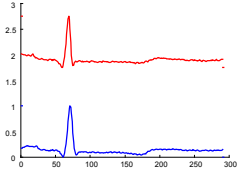
$\Rightarrow$ ... Must be tested anyway (cheap & sometimes efficient)

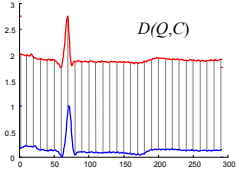


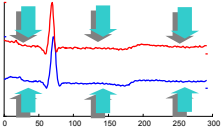
Handmade solution to address euclidian weaknesses

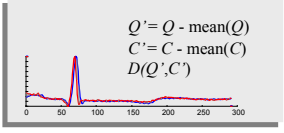
As Euclidian metric is cheap... We can try to compensate its weaknesses:

Transformation I: Offset Translation









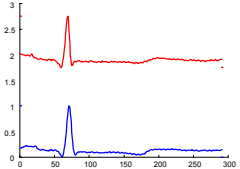
Remark: Q' and C' now have mean = 0

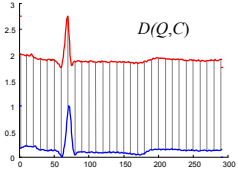
From **feature engineering** to **metric engineering**.

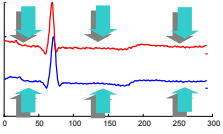
Handmade solution to address euclidian weaknesses

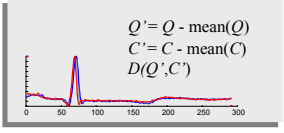
As Euclidian metric is cheap... We can try to compensate its weaknesses:

Transformation I: Offset Translation



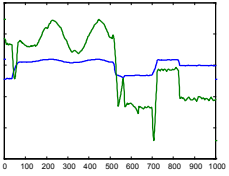


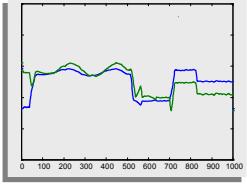




Remark: Q' and C' now have mean = 0

Transformation II: Amplitude Scaling





Z-score of Q
(i.e. zero-mean and one-std)

$Q'' = (Q - \text{mean}(Q)) / \textbf{std}(\boldsymbol{Q})$
$C'' = (C - \text{mean}(C)) / \textbf{std}(\boldsymbol{C})$
$D(Q'',C'')$

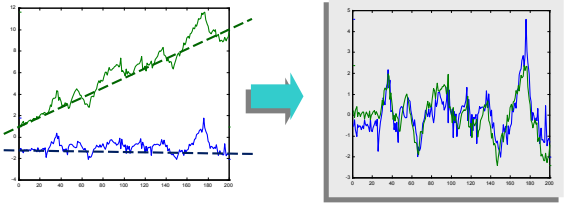
Or Procrust alignment procedure

From **feature engineering** to **metric engineering**.

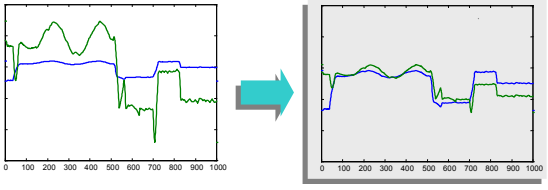
Handmade solution to address euclidian weaknesses

As Euclidian metric is cheap... We can try to compensate its weaknesses:

Transformation III: Linear Trend



Transformation II: Amplitude Scaling



Z-score of Q
(i.e. zero-mean and one-std)

$Q'' = (Q - \text{mean}(Q)) / \textbf{std}(\boldsymbol{Q})$

$C'' = (C - \text{mean}(C)) / \textbf{std}(\boldsymbol{C})$

$D(Q'', C'')$

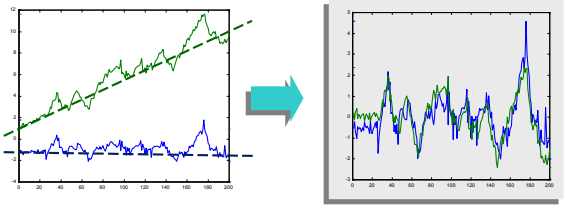
Or Procrust alignment procedure

From **feature engineering** to **metric engineering**.

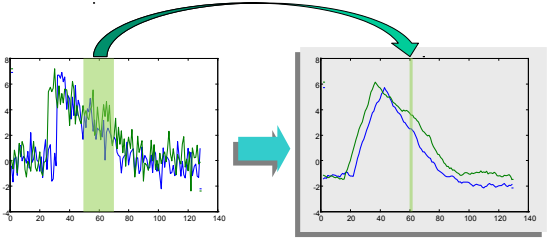
Handmade solution to address euclidian weaknesses

As Euclidian metric is cheap... We can try to compensate its weaknesses:

Transformation III: Linear Trend



Transformation IV: Noise

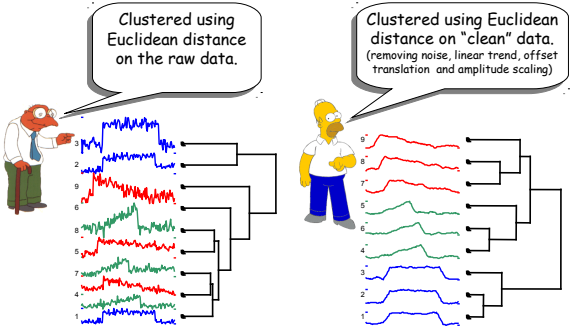


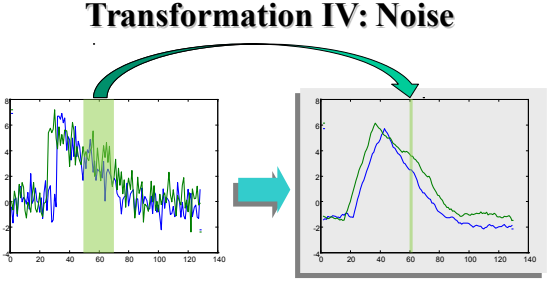
Low pass / band-pass / Butterworth, Bessel...
Or a simple moving average.

From **feature engineering** to **metric engineering**.

Handmade solution to address euclidian weaknesses

As Euclidian metric is cheap... We can try to compensate its weaknesses:



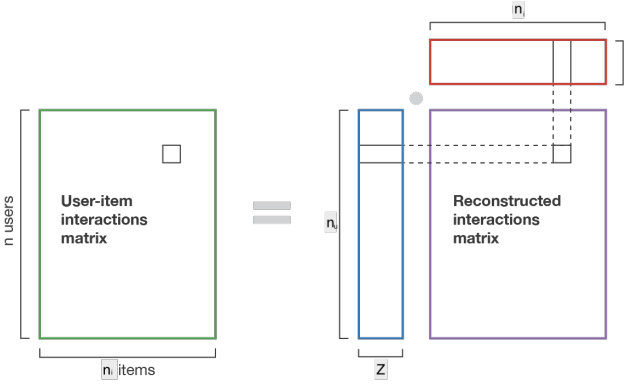


Low pass / band-pass / Butterworth, Bessel...
Or a simple moving average.

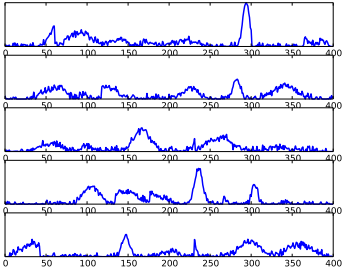
From **feature engineering** to **metric engineering**.

Matrix factorization as a filtering procedure

A elegant way to bridge with recommender systems!



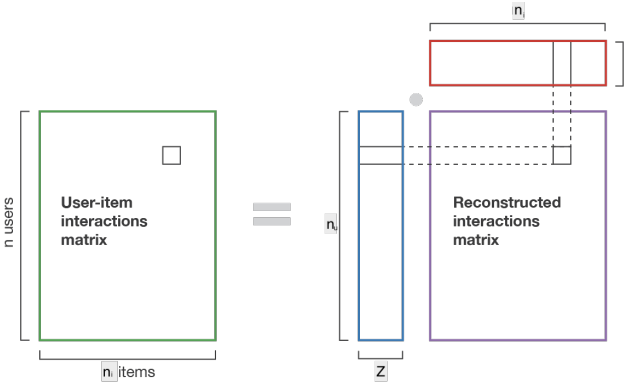
Interactions $\Rightarrow$ signals



Examples of signals including patterns

Matrix factorization as a filtering procedure

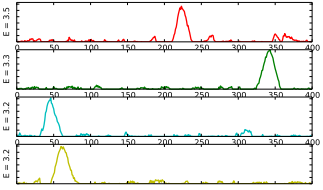
A elegant way to bridge with recommender systems!

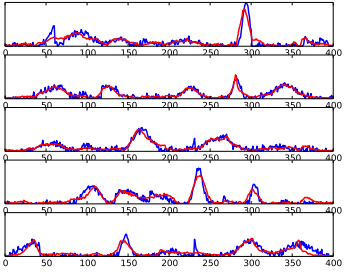


Interactions $\Rightarrow$ signals

$\Rightarrow$ Collaborative Filtering

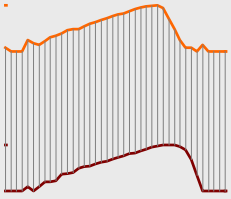
Parts of the dictionary:





DTW : Dynamic Time Wrapping

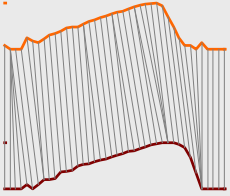
When Euclidian metric (& variations) fail:



Fixed Time Axis

Sequences are aligned "one to one".

Greatly suffers from the misalignment in data.



"Warped" Time Axis

Nonlinear alignments are possible.

Can correct misalignments in data.

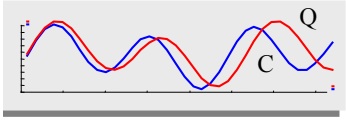
- Intrinsically adapted to small shift & scale variations
- Expensive $\mathcal{O}(2N^2)$... but not unaffordable

DTW : Dynamic Time Wrapping

Idea: comparing all pairing...
Too expensive

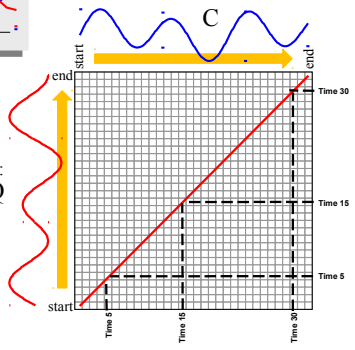
(1) Computing all individual distance between points $\mathcal{O}(N^2)$
(2) Assumming that both extremities are liked

How is DTW Calculated? I



The **Euclidean distance** works only on the diagonal of the matrix
The sequence of comparisons performed:
•Start from pair of points (0,0)
•After point (i,i) move to (i+1,i+1)
•End the process on (n,n)

We create a matrix the size of |Q| by |C|, then fill it in with the distance between every pair of point in our two time series.



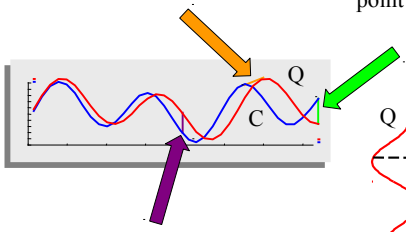
DTW : Dynamic Time Wrapping

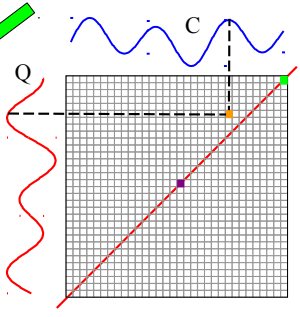
Idea: comparing all pairing...
Too expensive

(1) Computing all individual distance between points $\mathcal{O}(N^2)$
(2) Assumming that both extremities are liked

How is DTW Calculated? I

We create a matrix the size of |Q| by |C|, then fill it in with the distance between every pair of point in our two time series.





- The **DTW distance** can "freely" move outside the diagonal of the matrix
- Such cells correspond to temporally shifted points in the two time series

DTW : Dynamic Time Wrapping

Idea: comparing all pairing...

Too expensive

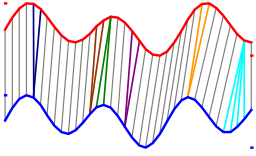
(1) Computing all individual distance between points $\mathcal{O}(N^2)$
(2) Assumming that both extremities are liked
(3) Searching for the best pairing...

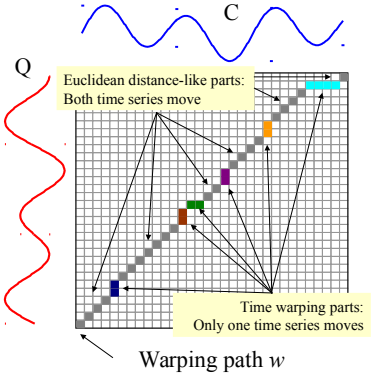
How is DTW Calculated? II

Every possible warping between two time series, is a path through the matrix.

The **DTW distance** can "freely" move outside the diagonal of the matrix

The only constraints:
•Start from pair of points (0,0)
•After point (i,j), **either "i" or "j" increase by one, or both of them**
•End the process on (n,n)



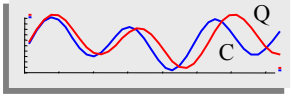


DTW : Dynamic Time Wrapping

Idea: comparing all pairing...
Too expensive

(1) Computing all individual distance between points $\mathcal{O}(N^2)$
(2) Assumming that both extremities are liked
(3) Searching for the best pairing...
... with a dynamic prog. approach $\mathcal{O}(N^2)$

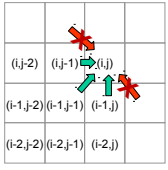
How is DTW Calculated? II



Definition of DTW as recursive function:

$\gamma(i,j)$ = cost of best path reaching cell (i,j)
$= d(q_i,c_j) + \min\{ \gamma(i\text{-}1,j\text{-}1), \gamma(i\text{-}1,j), \gamma(i,j\text{-}1) \}$

Idea: best path must pass through (i-1,j), (i-1,j-1) or (i,j-1)



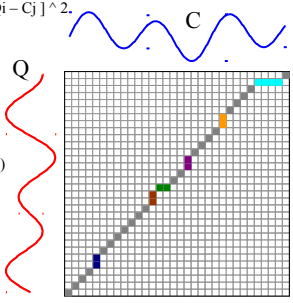
We want the best warping path:

$$\sum w_k \, i$$

w_k = cost of the k-th points comparison

Alternatives:
- $w_k = | Qi - Cj |$
- $w_k = [Qi - Cj] \hat{} 2$.

(i,j) = position of w_k

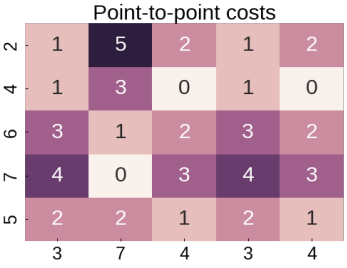


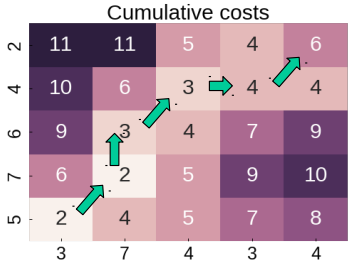
Remark: w can be longer than "n"

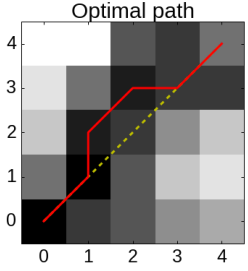
DTW : Results

t = < 3, 7, 4, 3, 4 >
q = < 5, 7, 6, 4, 2 >

$$\gamma(i,j) = d(q_i,c_j) + \min\{ \gamma(i\text{-}1,j\text{-}1), \gamma(i\text{-}1,j), \gamma(i,j\text{-}1) \}$$

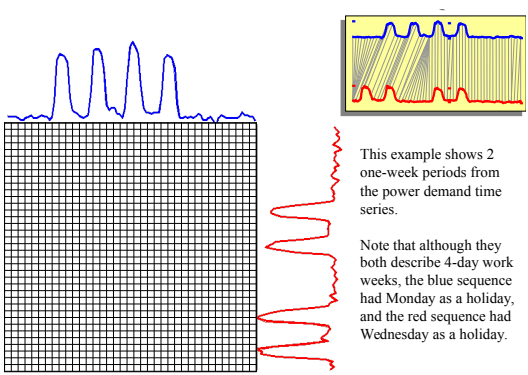






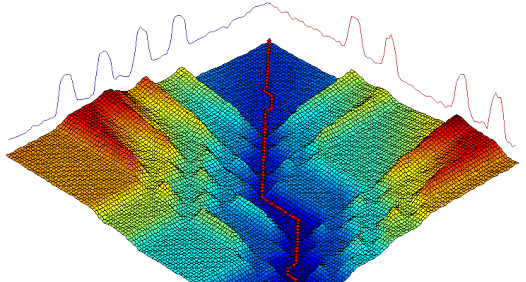
Computing both the **distance** and the **best pairing**

DTW : Results



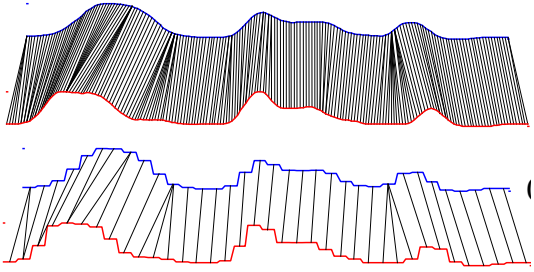
This example shows 2 one-week periods from the power demand time series.

Note that although they both describe 4-day work weeks, the blue sequence had Monday as a holiday, and the red sequence had Wednesday as a holiday.

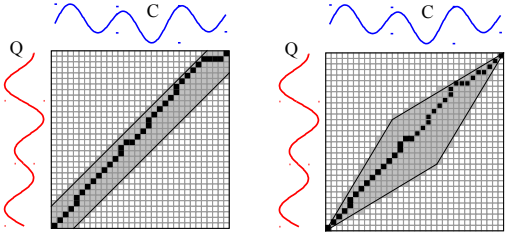


DTW : Improvements

Subsampling



Reduction of pairing investigations



⇒ Great computational cost reduction!

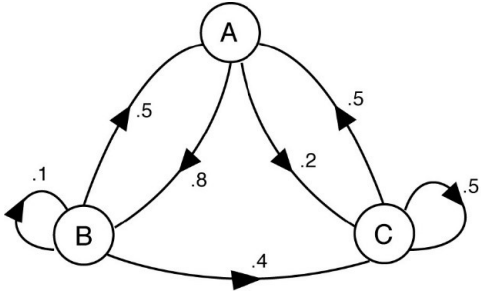
Markov model

The bayesian way to deal with time series

Markov Modeling :

- Adapted to discret signals
  - Quantization
- Adapted to variable length signals
- Fast/easy to learn (max likelihood)
- Given a series s, compute $p(s|\lambda)$

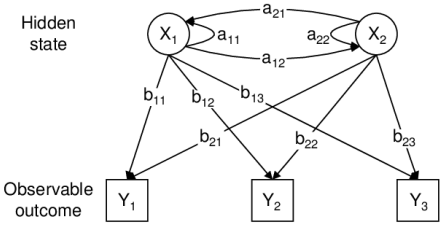
1. Train one model λ_c per class
2. Classif: $\arg\max_c p(s|\lambda_c)$



Parameters : $\lambda = (\Pi, A)$
Π = initial state probabilities
A = Stochastic transition matrix

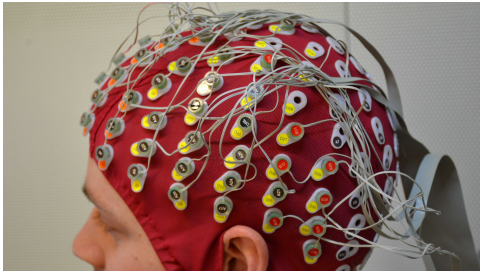
Hidden Markov Models

- Adapted to var. length signals
- Adapted to both discrete/continuous signals
- Ideal to introduce hypotheses on hidden states
- Can detect abrupt changes / anomalies in signals
- More costly to train & exploit
- Less convenient (hypotheses required to initialize the model)



Parameters : $\lambda = (\Pi, A, B)$
Π = initial state probabilities
A = Stochastic transition matrix
B = Emission probability distribution

Multivariate time series classification



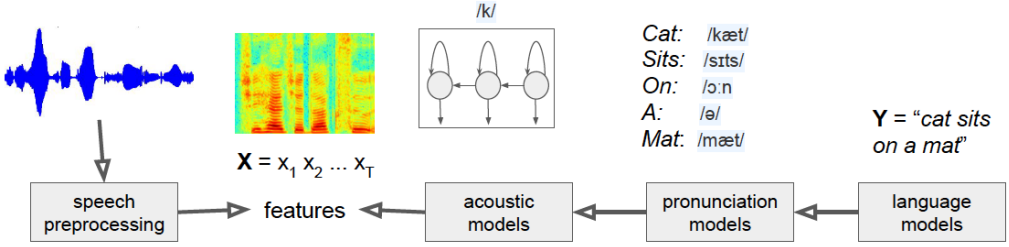
- EEG
  - Brain Computer Interface
- ECG
  - Internet of Things
- Radar signals

Often noisy signals

1. Baseline: concatenate the channels + standard signal classification
   - Poor results with noise
   - Require filtering & feature engineering ++
2. State of the art = Riemannian geometry
   One signal: $X_i \in \mathbb{R}^{C \times T}$
   Distance on covariance matrices: $d_{\mathcal{R}}(X_i X_i^T, X_j X_j^T)$
   $\Rightarrow$ Not easy to implement, but A. Barachant provides all libraries in python.

A Barachant et al., Trans. Biomed. Eng., 2011
Multiclass brain–computer interface classification by Riemannian geom

Classification of signal parts



- Event detection
- Speech reco. (phoneme classification)
- Handwriting reco.
- Sliding window approaches

$\Rightarrow$ Most applications move to deep learning

Financial time series analysis

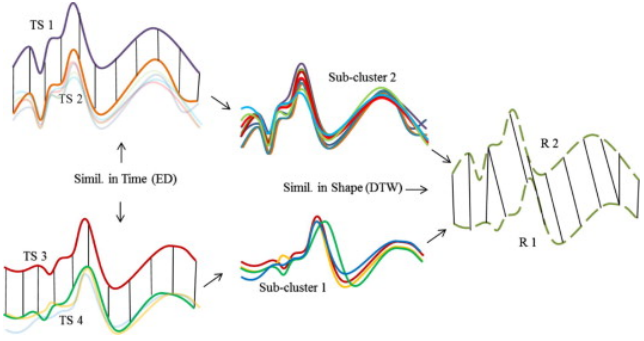
Lot of works on previous approaches... But still a robust event detection approach:

Buy/sell alarms are still based on short/long term moving average crossing

Clustering of signals / signal parts

- Interesting in both classification & forecasting schemas
  - pattern detection
- Identifying operating regime in a time series
  - Ensemble approaches / specific dev regarding one regime
  - Measure confidence / regime
- Aggregating signals = a way to tackle noise

Standard algorithms (e.g. k-means)
Specific distance (e.g. DTW)
In a co-movement group, DTW can identify precursors



Conclusion

Conclusion

Standard ML is required to deal with time series but not sufficient

- In time series forecasting, evaluation is more dangerous than in standard ML
  1. Use dedicated cross-validation approaches
  2. Beware of malicious features

- When computing the distance between signals, you must:
  1. try an Euclidian distance
  2. always keep in mind DTW