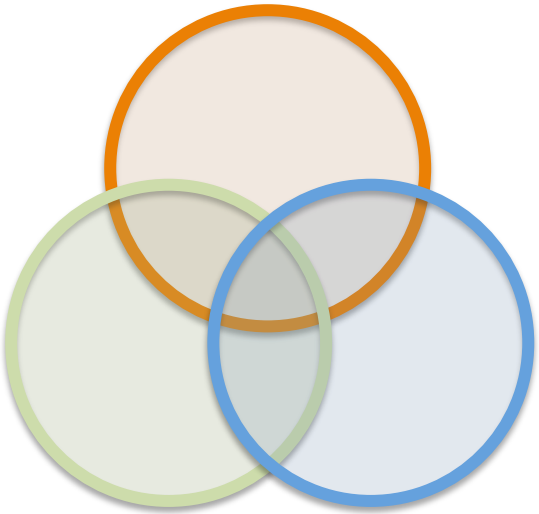
Machine Learning for the Quantified Self

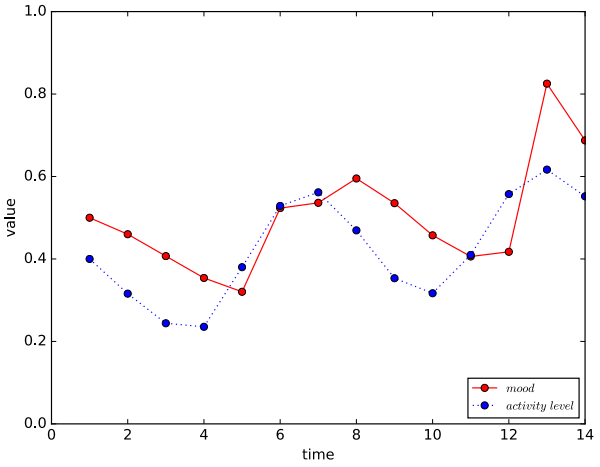


Chapter 8

Predictive Modeling with Notion of Time

Overview

- Previously: we looked at learning algorithms that did not model time explicitly
- Today: we will look at algorithms that consider time explicitly
  - Time series
  - Recurrent neural networks
  - Dynamical systems models



Time Series

- Time series focus on:
  - Understanding periodicity and trends
  - Forecasting
  - Control
- Time series can be decomposed in three components:
  - Periodic variations (daily, weekly, … seasonality)
  - Trend (how the mean evolves over time)
  - Irregular variations (left after we remove the periodic variations and trend)

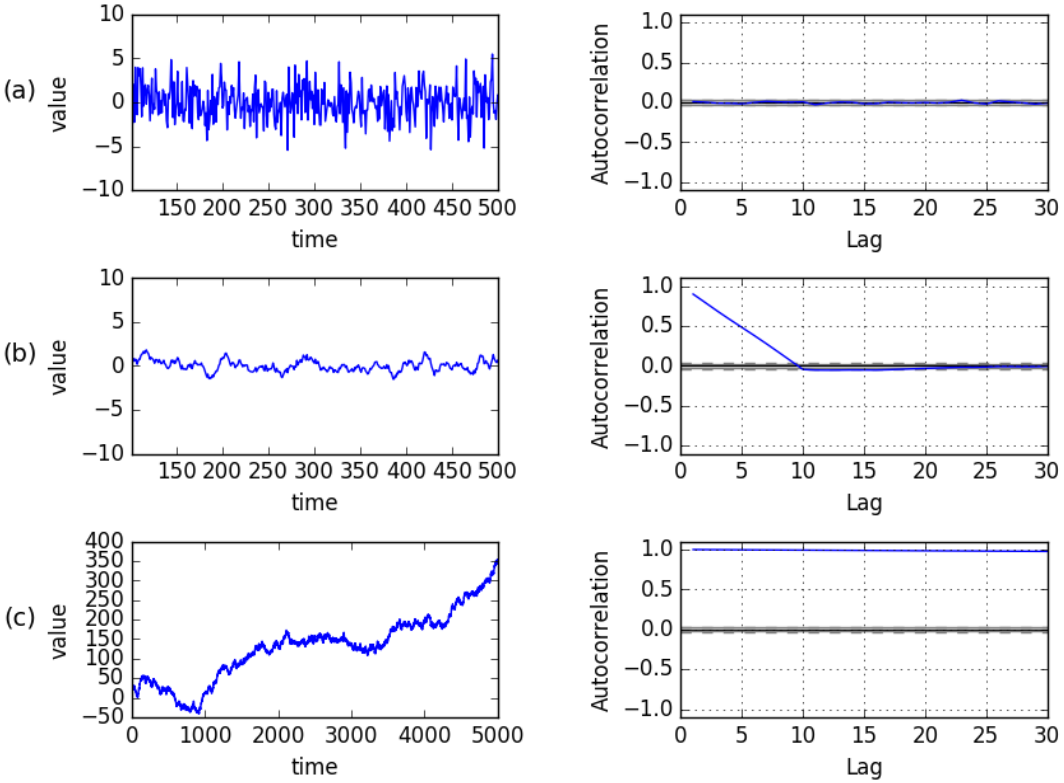
Stationarity (1)

- Important concept: stationarity
  - Trends and periodic variations are removed
  - Variance of the remaining residuals is constant
- Prerequisite or intermediate step for many algorithms
- Additional criterion: the lagged autocorrelation should remain constant (note: x_t represents the value of one attribute):

$$r_\lambda = \frac{\sum_{t=1}^{N-\lambda} (x_t - \bar{x})(x_{t+\lambda} - \bar{x})}{\sum_{t=1}^{N} (x_t - \bar{x})^2}$$

Stationarity (2)

- Autocorrelation represents in how far there is a correlation between a time series and a shifted version of itself (with λ steps)



random

some memory over past time points

cumulative sum of (a) – non-stationary

Filtering and smoothing (1)

- Let us assume our time series of values x_t with a fixed step size Δt
- We can apply a filter to our data, taking q points in the future and past into account:

$$z_t = \sum_{r=-q}^{q} a_r x_{t+r}$$

- This generates a new time series z_t

Filtering and smoothing (2)

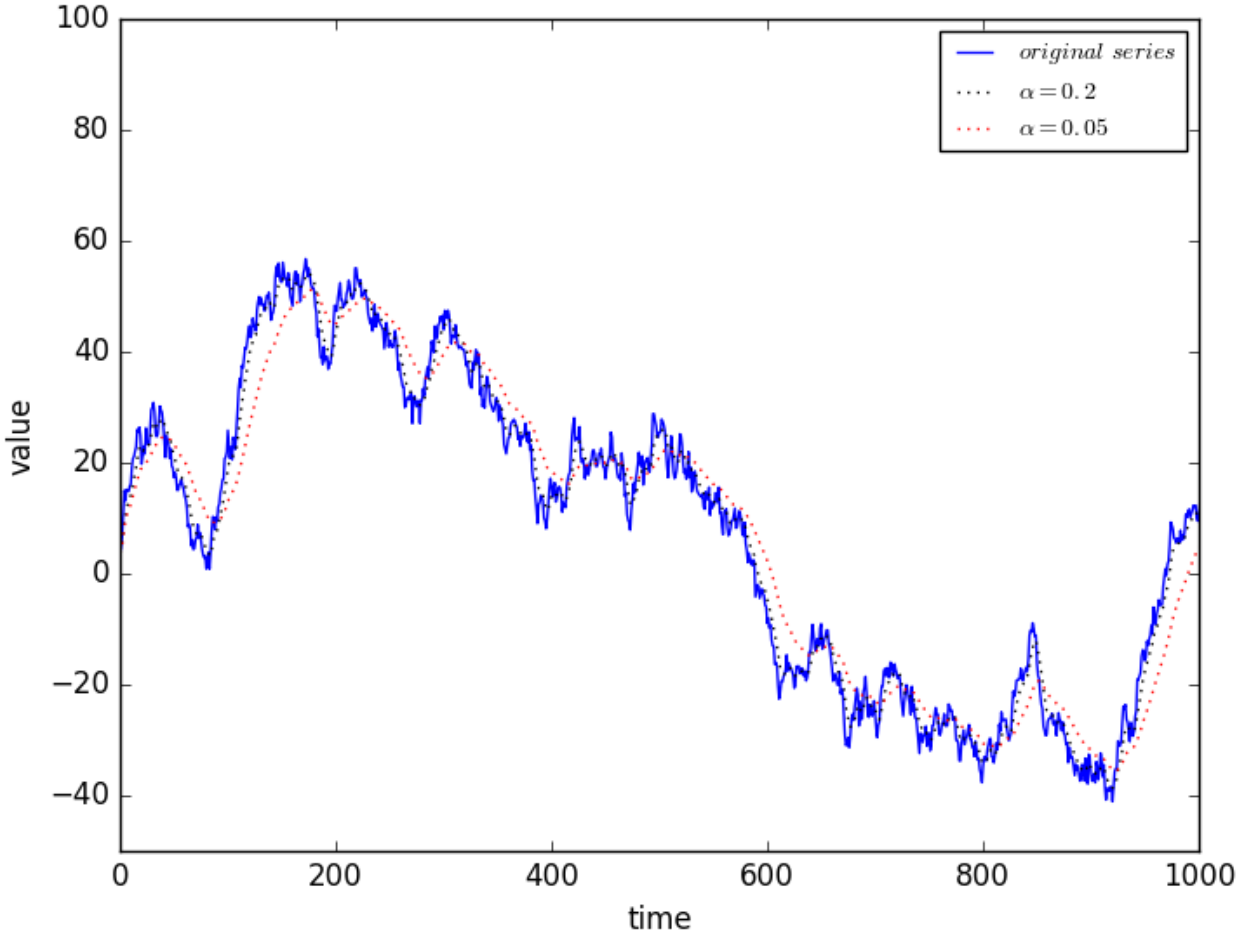
- What could a_r look like?
  - If we take $a_r = (2q+1)^{-1}$ it is just the *moving average*
  - If measurements closer to *t* are more important we can use a *triangular shape*:

$$a_r = \begin{cases} \frac{q-|r|}{q^2} & -q \le r \le q \\ 0 & otherwise \end{cases}$$

  - Or exponential smoothing (only past time points mostly): $a_r = \frac{\alpha(1-\alpha)^{|r|}}{2-\alpha}$

Filtering and smoothing (3)

- Example exponential smoothing



Filtering and smoothing (4)

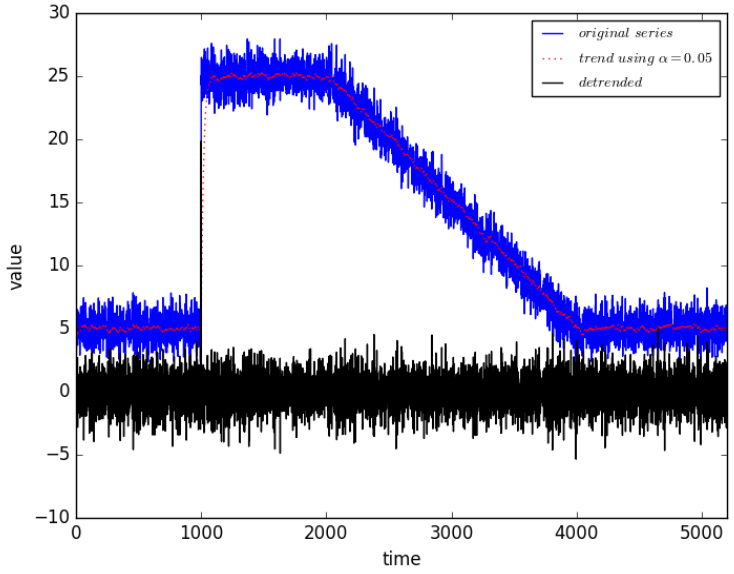
- Now how can we remove a trend?
- Let us take a filter again, but a simple one:

$$z_t = x_t - x_{t-1} = \nabla x_t$$

- This takes the different between the current and previous measurement
  - A long term trend has more or less the same influence on the previous and current time point
- We can apply this operator *d* times (e.g. *d*=2): $\nabla^2 x_t = \nabla x_t - \nabla x_{t-1}$

Filtering and smoothing (5)

- But x_{t-1} might not be a good estimation of the trend, we can therefore also use an exponential smoothing z_t and take $x_t - z_t$



ARIMA (1)

- Of course we would like to forecast, let us turn to ARIMA
  - Assume a measurement at time point *t* is generated by a probability distribution P_t
  - The expected mean is: $\mu(t) = E[P_t]$
  - The auto-covariance function is:

    $$\gamma(t_1, t_2) = E[(P_{t_1} - \mu(t_1))(P_{t_2} - \mu(t_2))]$$

  - A series is stationary when the mean is constant and when the auto-covariance only depends on the time difference $\lambda = t_2 - t_1$

ARIMA (2)

- We assume that the probability distribution at time point *t* is regressed on its own lagged values (*p* past measurements to be precise)
- W_t is the noise we encounter. We can account for the noise by a moving average component (with *q* past values)
- Overall the model becomes:

$$P_t = \phi_1 P_{t-1} + \ldots \phi_p P_{t-p} + W_t + \theta_1 W_{t-1} + \ldots \theta_q W_{t-q}$$

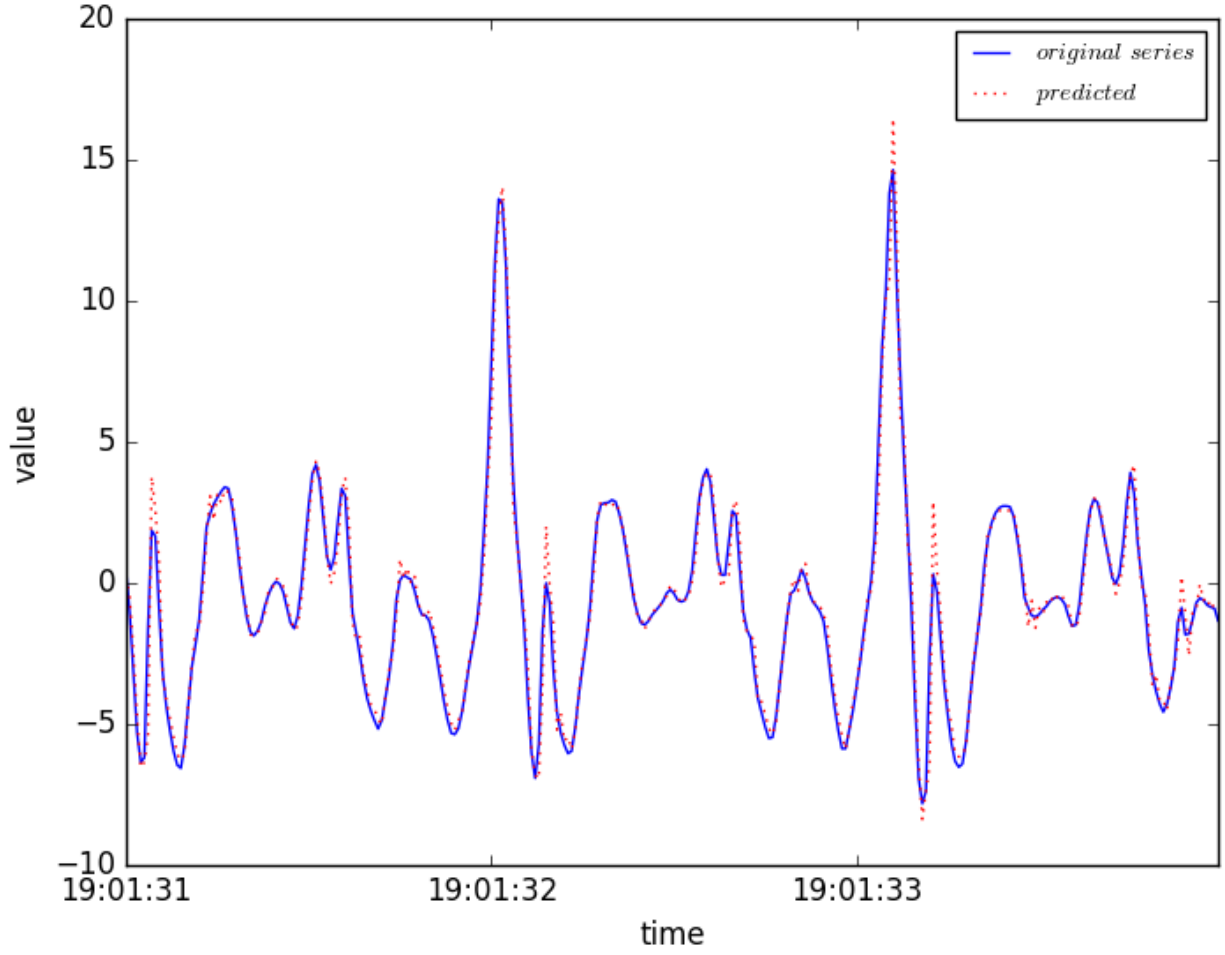
ARIMA (3)

- In addition, we consider differencing with an order *d*
- So how do we find the values for p, q, and d?
  - For p we can look at the correlation between x_t and x_{t-p} (Partial Autocorrelation Function)
  - We can do a grid search over the other parameters and determine the goodness of fit (e.g. using the Akaike Information Criterion)
- And how about the other parameters?
  - We use our data, e.g. for autoregressive component:

$$S = \sum_{t=p+1}^{N} (x_t - \phi_1 x_{t-1} - \cdots - \phi_p x_{t-p})$$

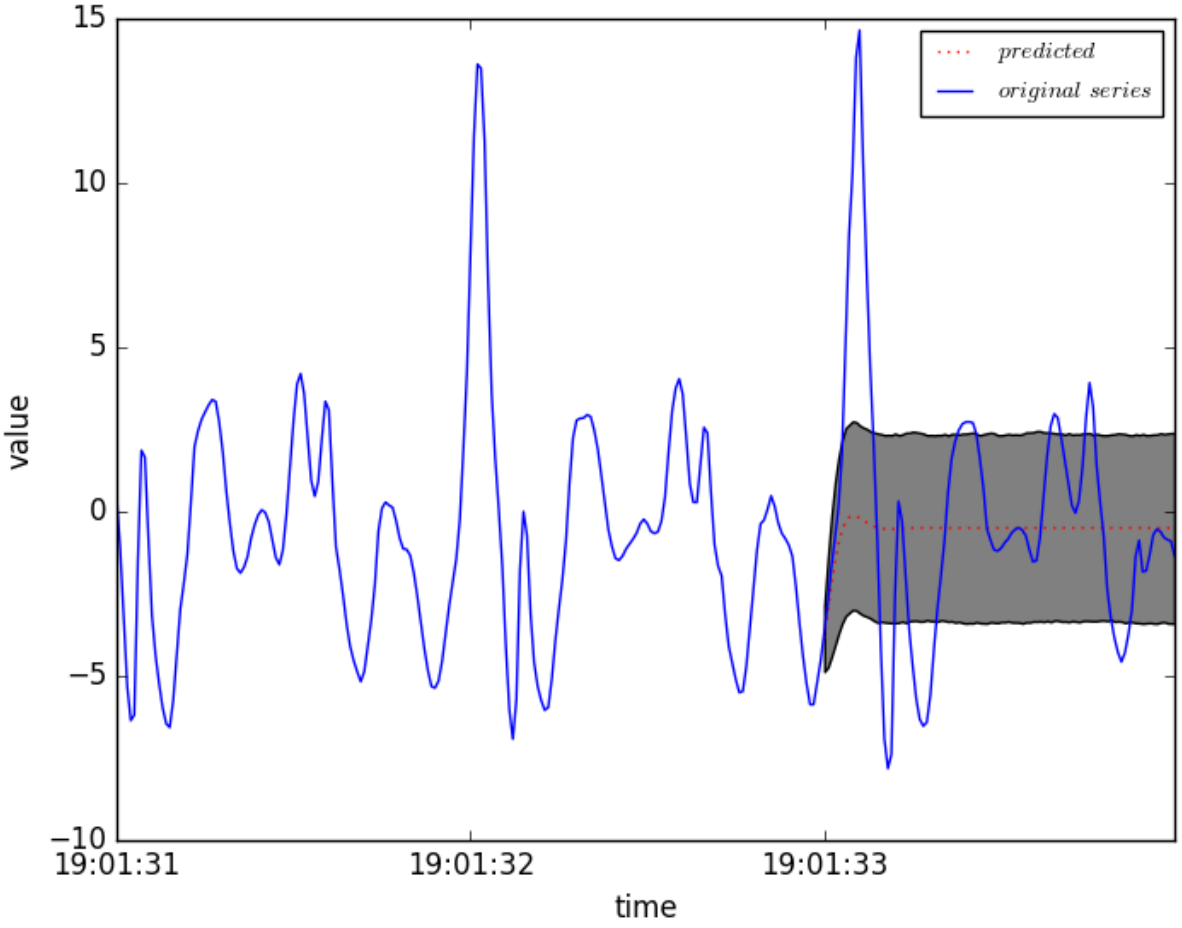
ARIMA - example (1)

- One step ahead prediction (p=3, q=2)



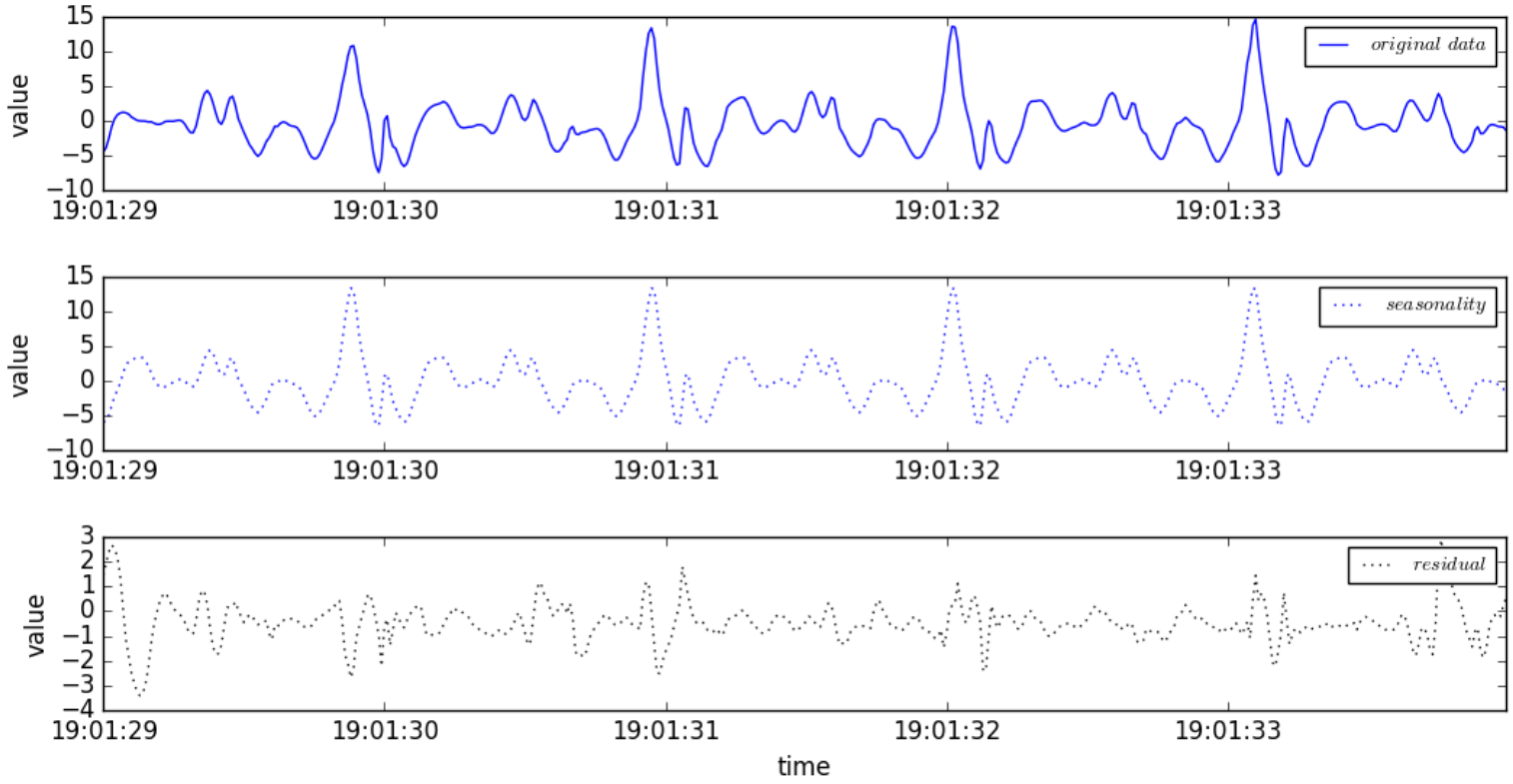
ARIMA - example (2)

- Multiple steps ahead prediction (p=3, q=2)



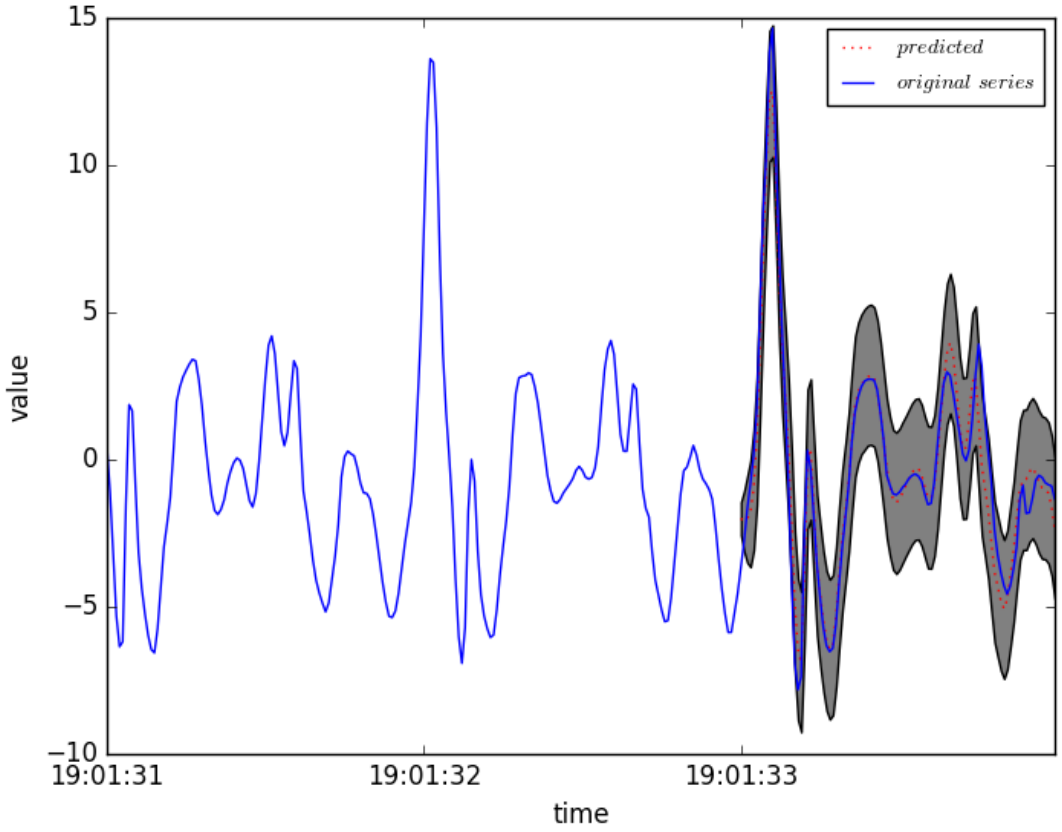
ARIMA - example (3)

- Seasonality decomposition



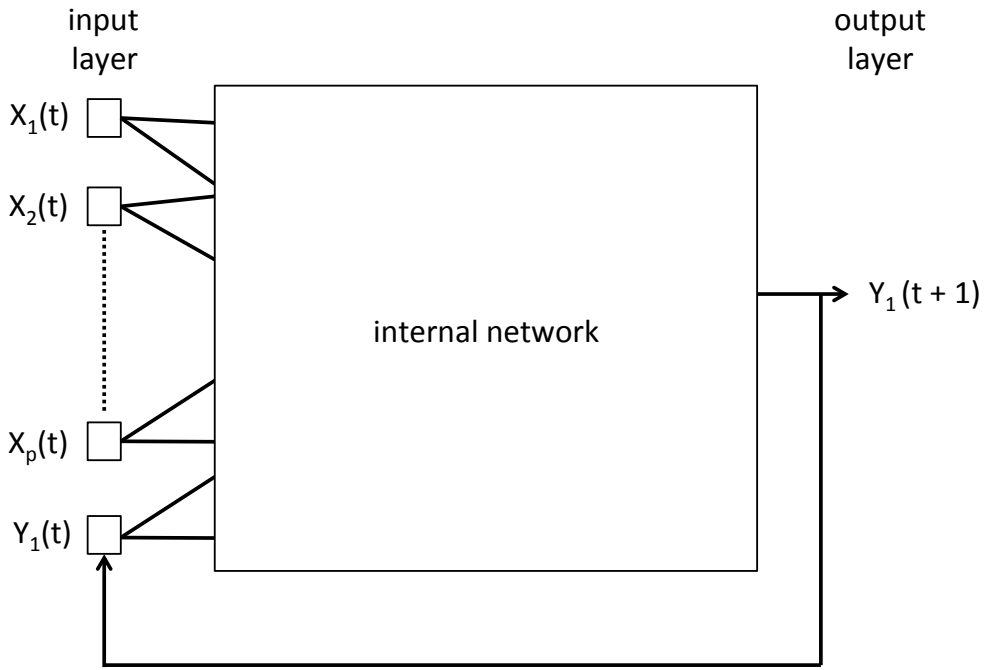
ARIMA - example (4)

- Multiple steps ahead prediction with seasonality (p=3, q=2)



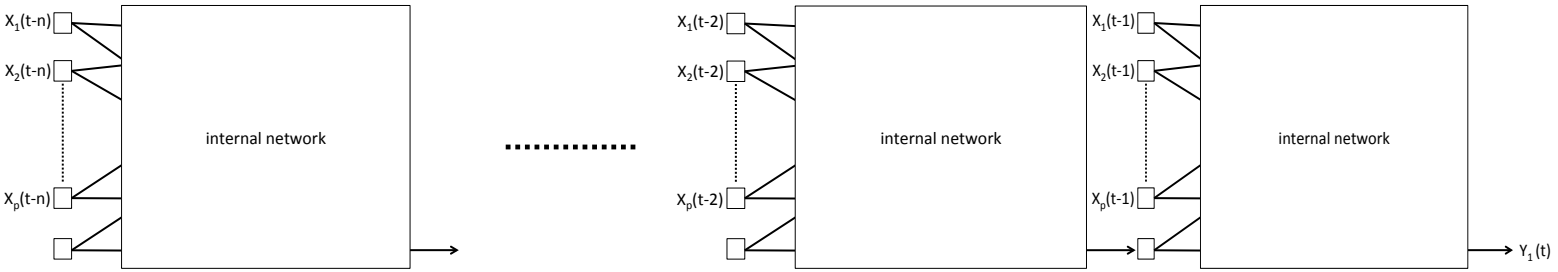
Neural networks with time

- Variant of neural networks that take time component into account explicitly
- Let us consider the simplest variant first:



Recurrent neural networks (1)

- Not very different from our previous neural networks, but cycles make training tricky
  - How do we handle this?
  - We unfold the network, and apply back propagation again



Recurrent neural networks (2)

- How does this work mathematically?
  - Update of regular connections remains the same:

$$\Delta w_{ij} = \eta \delta_j(t) \hat{y}_t^{(i)}$$

$$\delta_j(t) = \begin{cases} \varphi'(v_j(t))(y_t^j - \hat{y}_t^{(j)}) & \text{if } j \text{ is an output node} \\ \varphi'(v_j(t)) \sum_1^k (\delta_k(t) w_{jk}) & \text{otherwise} \end{cases}$$

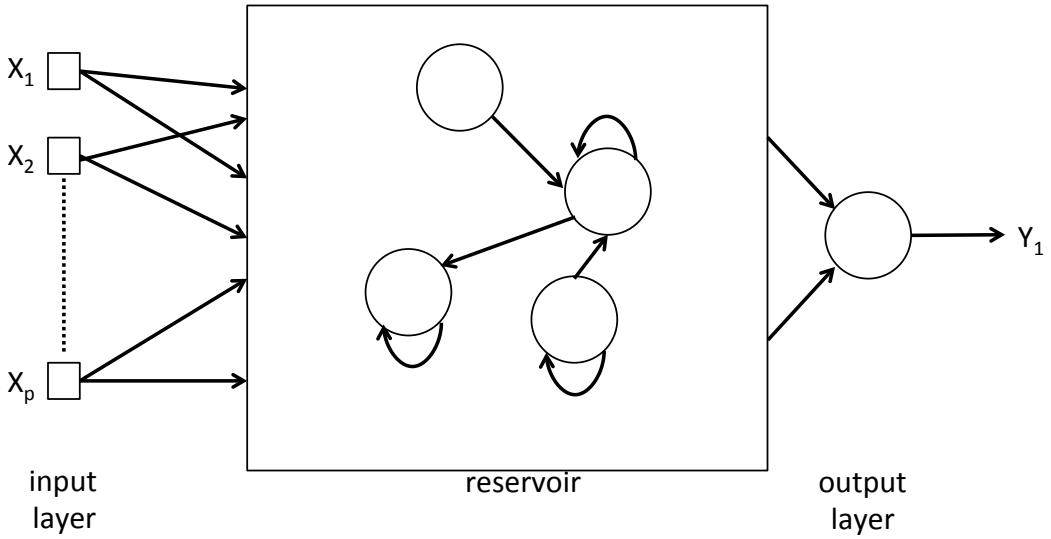
  - For recurrent connections we take:

$$\Delta w_{ij} = \eta \delta_j(t) \hat{y}_{t-1}^{(i)}$$

$$\delta_j(t-1) = \varphi'(v_j(t-1)) \sum_1^k (\delta_k(t) w_{jk})$$

Echo state networks (1)

- Training of RNN's is difficult
- Echo state networks try to tackle this problem
- Have a reservoir of randomly collected neurons



Echo state networks (2)

- We have several matrices of weights:

  $\mathbf{W^{in}}$ is an $n \times p$ matrix for the weights from the input layer to the reservoir.
  $\mathbf{W}$ is the $n \times n$ matrix of the internal weights in the reservoir.
  $\mathbf{W^{out}}$ is the $l \times n$ matrix that specifies the weights to the connections between the reservoir and the output.

- $\mathbf{W^{in}}$ and $\mathbf{W}$ are randomly set and fixed, we only learn $\mathbf{W^{out}}$
- We compute the output as follows:

$$r_{i+1} = \varphi(\mathbf{W^{in}} x_{i+1} + \mathbf{W} r_i)$$

$$\hat{y}_{i+1} = \varphi_{out}(\mathbf{W^{out}} r_{i+1})$$

Echo state networks (3)

- And we simply learn a W^{out} that minimizes the difference between the actual and predicted y
- How should we create the random reservoir?
  - It should satisfy the echo state property

Definition 8.1. Echo state property: The effect of a previous state r_i and a previous input x_i on a future state r_{i+k} should vanish gradually as time passes (i.e. $k \to \infty$) and not persist or even get amplified.

Echo state networks (4)

- And how do we establish this?
  - Follow this procedure:

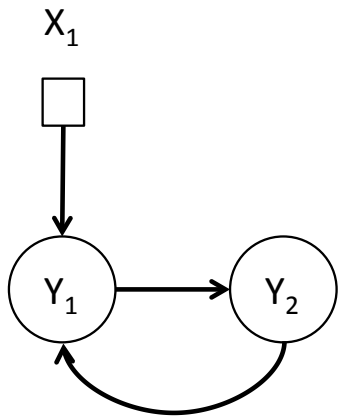
Algorithm 19: Reservoir initialization procedure

1. Randomly initialize an internal weight matrix $\mathbf{W_0}$. $\mathbf{W_0}$ should be sparse and have a mean of 0. The size n reflects the number of training examples N (should not exceed $\frac{N}{10}$ to $\frac{N}{2}$ depending on the complexity)
2. Normalize $\mathbf{W_0}$ to matrix $\mathbf{W_1}$ with unit spectral radius by putting $\mathbf{W_1} = \frac{1}{\rho(\mathbf{W_0})}\mathbf{W_0}$
3. Scale $\mathbf{W_1}$ to $\mathbf{W} = \alpha\mathbf{W_1}$ where $\alpha < 1$, whereby $\rho(W) = \alpha$

Then $\mathbf{W}$ is a network with the echo state property (has always found to be)

Dynamical Systems Models (1)

- The final type of temporal model we will focus on are dynamical systems models
- Express differential equations between attributes and targets



Dynamical Systems Models (2)

- Equations are based on domain knowledge (e.g. scientific papers)
- Models do contain parameters that can be tuned
  - that is where the machine learning will come into play

Dynamical Systems Models (3)

- Let us consider an example model
  - model the relationship between activity and mood (think of the example of Bruce)

$$\hat{y}_{mood}(t+\Delta t) = y_{mood}(t) +$$
$$x_{outside}(t) \cdot (\gamma_1 \cdot (1 - y_{mood}(t)) \cdot pos(y_{activity\ level}(t) - y_{mood}(t)) +$$
$$\gamma_2 \cdot y_{mood}(t) \cdot neg(y_{activity\ level}(t) - y_{mood}(t))) \cdot \Delta t$$
$$\hat{y}_{activity\ level}(t+\Delta t) = y_{activity\ level}(t) +$$
$$(\gamma_3 \cdot (1 - y_{activity\ level}(t)) \cdot pos(sin(\frac{t - \gamma_4 \pi}{\gamma_5})) +$$
$$\gamma_4 \cdot y_{activity\ level}(t) \cdot neg(sin(\frac{t - \gamma_4 \pi}{\gamma_5}))) \cdot \Delta t$$

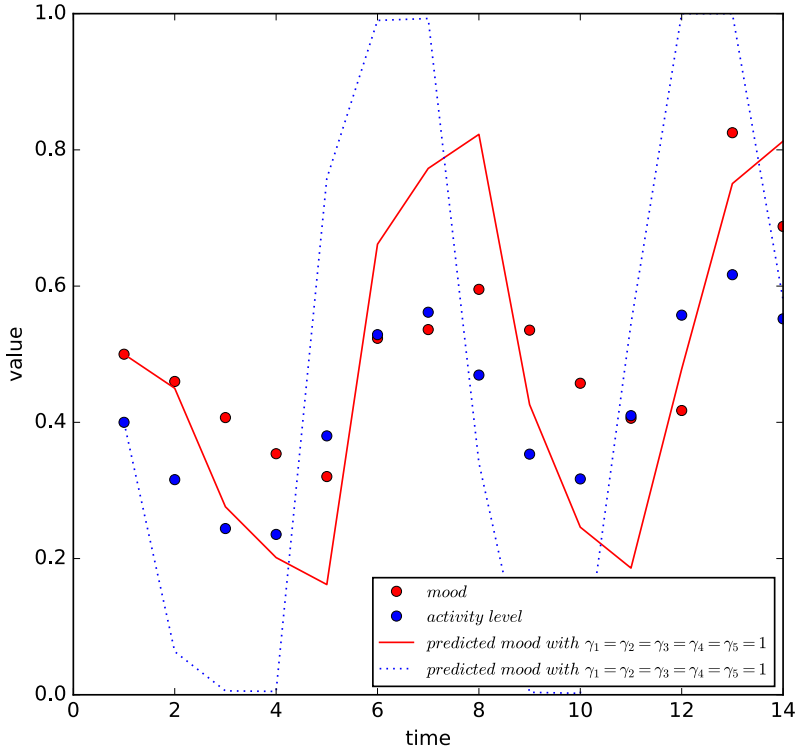
with

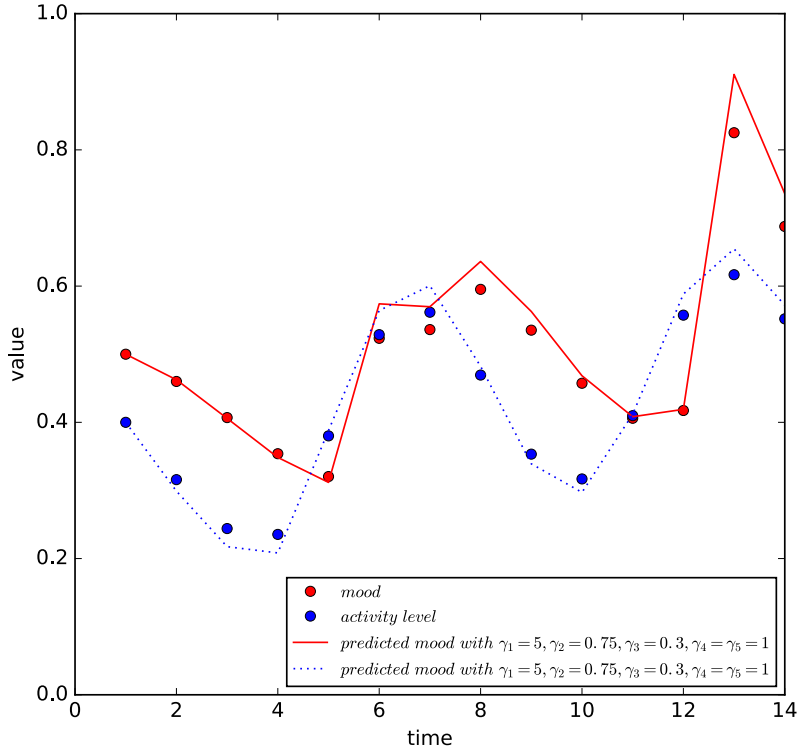
$$pos(v) = \begin{cases} 0 & v < 0 \\ v & otherwise \end{cases}$$

$$neg(v) = \begin{cases} v & v < 0 \\ 0 & otherwise \end{cases}$$

Parameter optimization (1)

- How important are the parameters?





Parameter optimization (2)

- How do we find appropriate parameter settings?
  - Manual tweaking requires a lot of labor
  - We have data available that allows us to find "the best" parameter values using machine learning
  - Assume λ is a vector of parameter values, then the error we see is

$$E(\lambda) = \sum_{t=1}^{N} (\hat{y}(t) - y(t))^2 \text{ where } \hat{y}(t) \text{ is the prediction of the model given } \lambda$$

Simulated Annealing (1)

- Let us first focus on a very simple algorithm: simulated annealing
  - We randomly move in our parameter space
    - so we randomly select values for our parameters
  - Is that all?
    - Nope, we assume a *temperature* of our process, and a maximum number of iterations (k_{max})
    - We accept parameter vectors that are better
    - Depending on the temperature we can also accept parameter vectors that are worse (exploration)
    - The closer we get to the end (the lower the temperature), the lower the probability of accepting vectors with worse performance

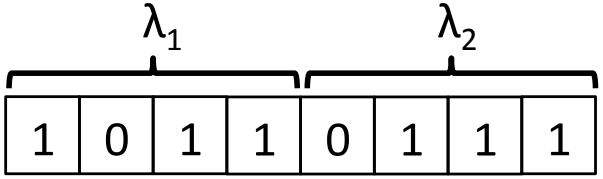
Simulated Annealing (2)

Algorithm 20: Simulated Annealing

$\lambda_{current}$ = random
$E_{prev} = \infty$
Temp = Temp_{init}
for *k from 1 to* k_{max} **do**
  for *i in* λ **do**
    $\lambda'_i = \lambda_i + \text{random}$
  end
  if $E(\lambda') \leq E(\lambda_{current})$ **then**
    $\lambda_{current} = \lambda'$
  else if $e^{\frac{(E(\lambda_{current}) - E(\lambda'))}{k_b Temp}} \geq random(0,1)$ **then**
    $\lambda_{current} = \lambda'$
  $\text{Temp} = \alpha \cdot \text{Temp}$
end
return $\lambda_{current}$

Genetic Algorithms (1)

- Second option is to optimize the parameters using genetic algorithms
- We consider the simple GA
- We encode our parameter values as bits (genotype), these are individuals



- We create a whole population of these individuals

Genetic Algorithms (2)

- From the population, we select parents for a mating pool
- We perform crossover and mutation
- We select a new population for the next generation
- We continue this for a number of generations

Genetic Algorithms (3)

Algorithm 21: Simple Generic Algorithm

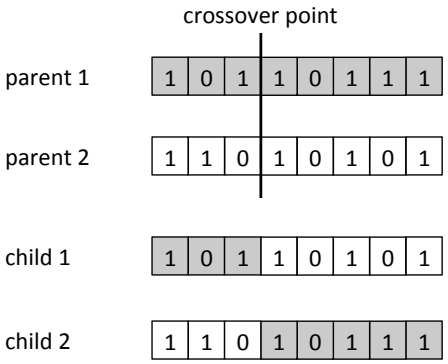
population = random initialization of population with set population size ps
for *i from 1 to max_generations* **do**
    Select ps parents according to equation 8.21
    Select pairs of parents from the individuals we have selected (without replacement)
    Apply crossover to the parents with probability p_c or copy the original parents
    Apply bitwise mutation with probability p_m on the result individuals
    The individuals we have just created become the new population
end
return *fittest individual in the final population*

Genetic Algorithms (4)

- Selection of individuals is done by a roulette wheel:

$$P_i = \frac{(1 - E(\lambda_i))}{\sum_{j=1}^{pop\_size} E(\lambda_j)}$$

- Crossover is straightforward (if we apply it to the parents), select a crossover point:

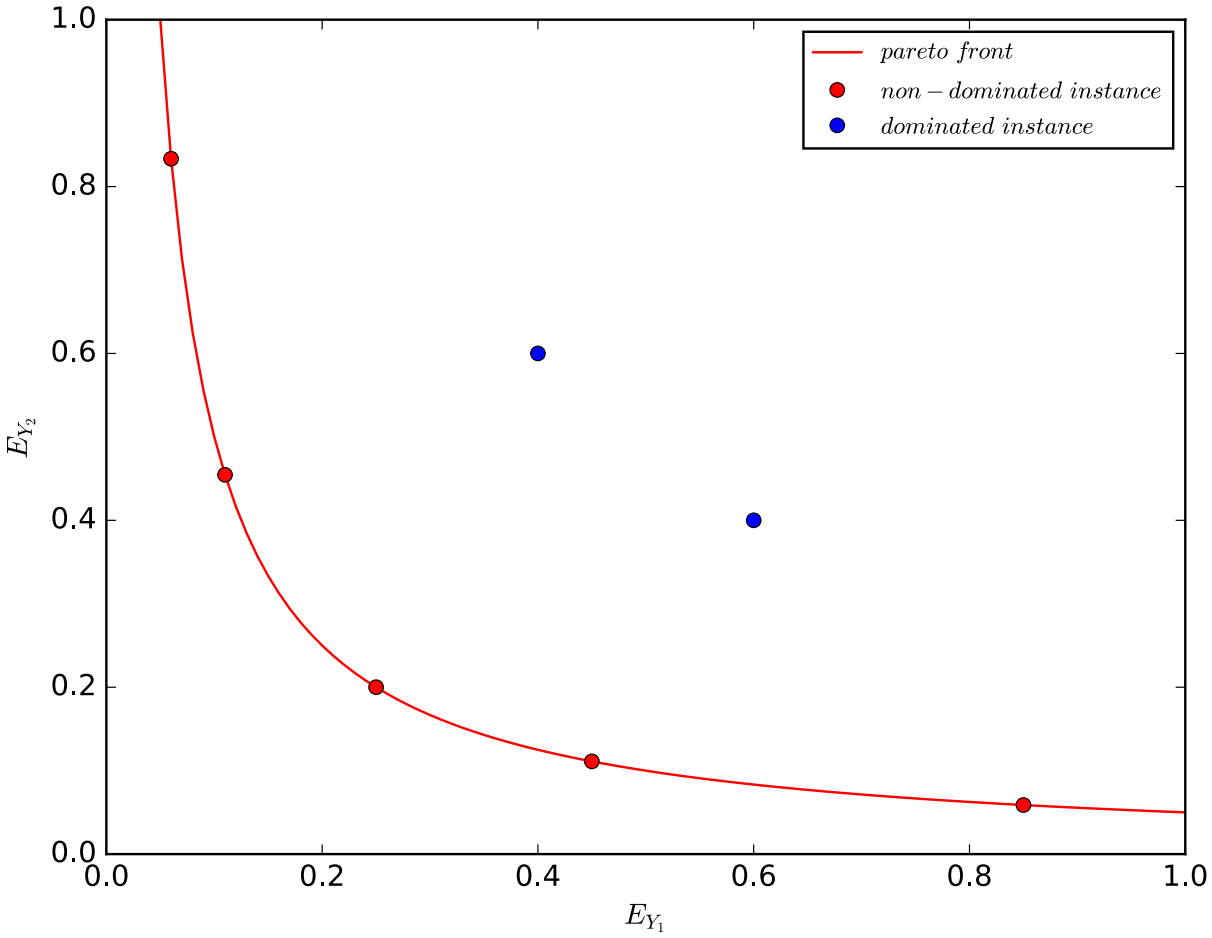


- Mutation is done per bit, with a probability p_m

Multi-Criteria Optimization

- What if we have multiple targets?
- We could just take the average error over all targets (in fact, we have done so)
- We might however be interested in the trade-off between different targets
- This is then a multi-criteria optimization problem
- We call a dynamical systems model with certain parameter values λ a *model instance*

Pareto efficiency (1)



Pareto efficiency (2)

- We assume an error function per target

$$E_{target}(\lambda) = \sum_{t=1}^{N} (\hat{y}_{target}(t) - y_{target}(t))^2$$

- We want to look for non-dominated instances

Definition 8.2. A model instance λ_m is dominated by another model instance λ_n when the mean squared error obtained by model instance λ_n is lower for at least one target and not higher for any of the other targets.

- Formally:

$$dominated(\lambda_m, \lambda_n) = \begin{cases} 1 & \exists i \in 1, \ldots, q : E_i(\lambda_m) > E_i(\lambda_n) \wedge \\ & \forall j \in 1, \ldots, q : E_j(\lambda_m) \geq E_j(\lambda_m) \\ 0 & \text{otherwise} \end{cases}$$

NSGA-II (1)

- The NSGA-II algorithm can find these non-dominated model instances
- NSGA = Non-Dominated Sorting Genetic Algorithm
- It has a population of solutions (like the simple GA)
- We create fronts from the population

NSGA-II (2)

Algorithm 22: Finding Pareto Fronts

```
find_pareto_fronts(P):
used = []
i = 0
F = [] while |P| > 0 do
    P' = P[1] // The first model instance in the population
    for p ∈ P ∧ p ∉ P' do
        P' = P' ∪ {p}
        for q ∈ P' ∧ p ≠ q do
            if dominated(q, p) then
                P' = P' \ {q}
            else if dominated(p, q) then
                P' = P' \ {p}
        end
    end
    Add P' to F
    P = P \ P'
    i = i + 1
end
return F
```

NSGA-II (3)

- We would like to get a nice spread of solutions across the Pareto Front
- We compute the distance of points to other points in the front and sort the points based in this distance (highest distance first)
- Points on the boundary are set to infinite distance

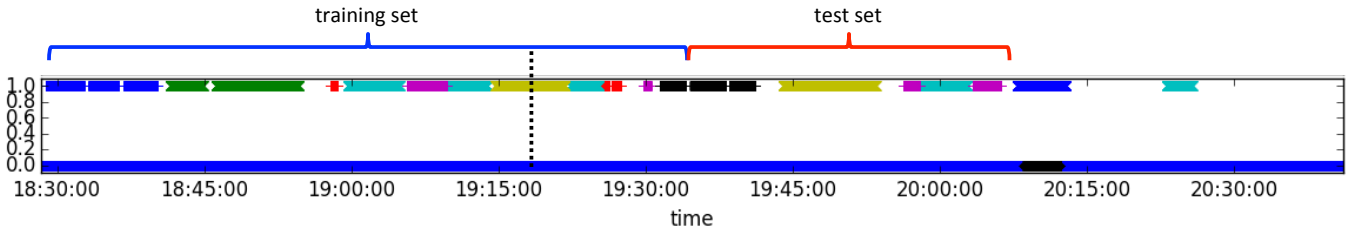
NSGA-II (4)

Algorithm 24: NSGA-II main loop

$R_t = P_t \cup C_t$ // Take the parent and child population
$F_1, \ldots, F_f = \text{find\_pareto\_fronts}(R_t)$
$P_t = \emptyset$
$i = 1$
while $|P_{t+1}| < |P_t|$ **do**
    if $|P_{t+1}| - |P_t| \geq |F_i|$ **then**
        $P_{t+1} = P_{t+1} \cup F_i$
        $i = i + 1$
    else
        $d = \text{distance\_assignment}(F_i)$
        $F_{sorted} = sort(F_i, d)$
        $P_{t+1} = P_{t+1} \cup \{F_{sorted}[1]\} \cup \cdots \cup \{F_{sorted}[|P_{t+1}| - |P_t|]\}$
end
Create C_{t+1} using crossover and mutation
$t = t + 1$

Case study

- Let us move to the CrowdSignals again
- Recall from last time: how did we tune the parameters?
  - Cross validation
  - Does that make sense now?
  - Nope it does not…..



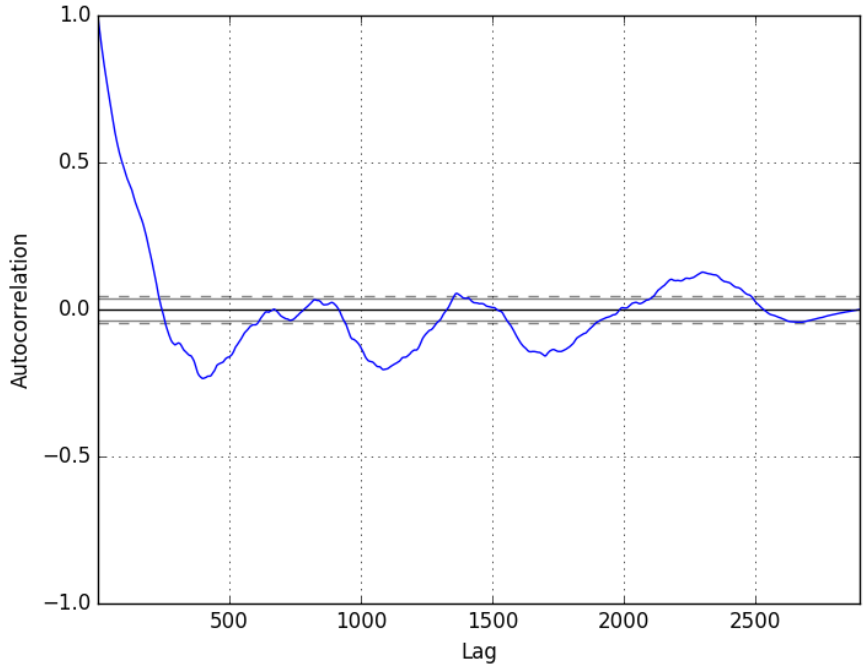
Algorithms used

- What techniques do we use?
  - Note: normalize values to [0,1] and [-0.9,0.9]/ [0.1, 0.9]

Algorithm	Variant description	Parameters varied
Echo State Network (ESN)	Randomly connected reservoir of neurons with tanh activation function with the output being fed back into the reservoir	Number of neurons in reservoir: $\{500, 1000, 5000\}$ α: $\{0.4, 0.6\}$
Recurrent Neural Network (RNN)	Recurrent neural network with one layer of hidden neurons with a sigmoid activation function and sigmoid output nodes	number of hidden neurons: $\{50, 100\}$ maximum iterations over the entire dataset: $\{250, 500\}$
Time series	ARIMAX algorithm using Bayesian inference	p: $\{0, 1, 3, 5\}$ q: $\{0, 1, 3, 5\}$ d: $\{0, 1\}$

Time series

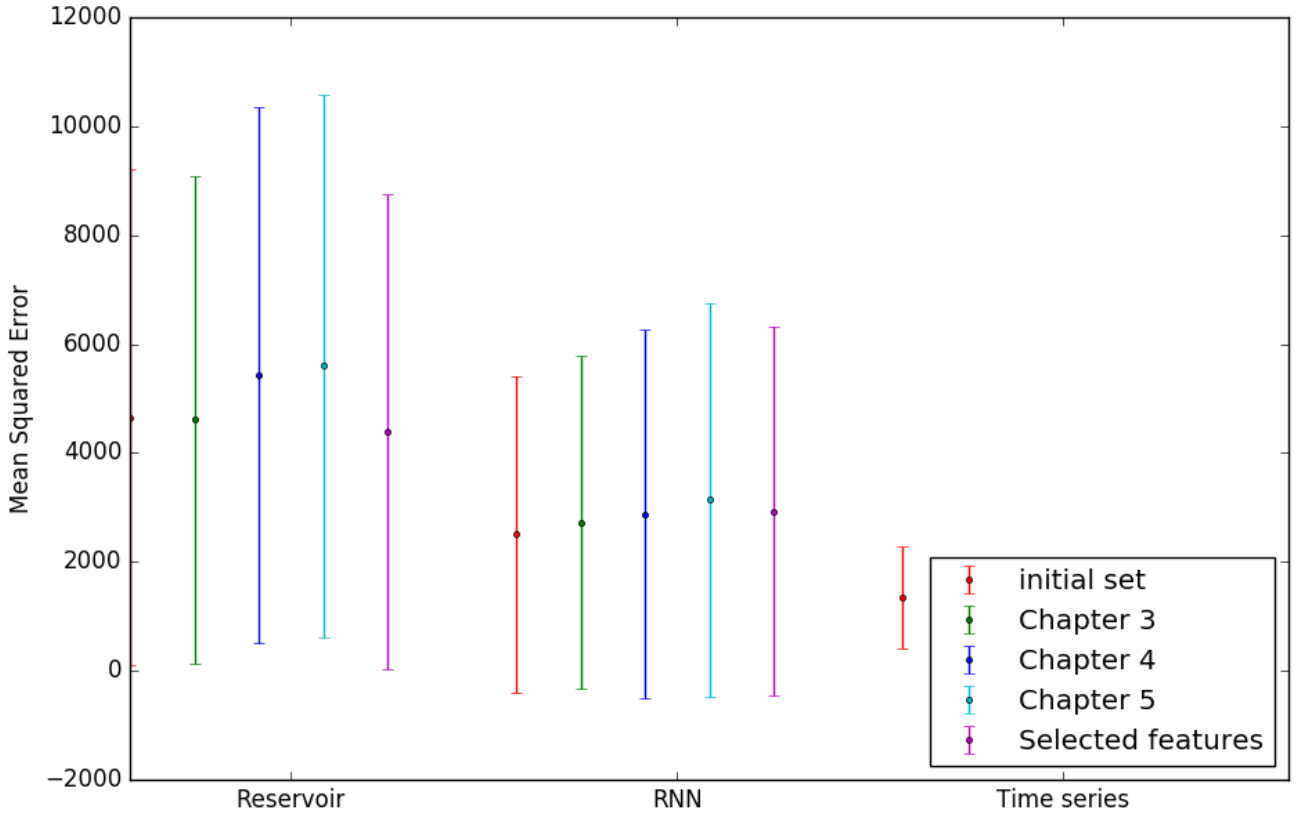
- Autocorrelations we observe:



- Stationary time series (Dickey-Fuller test)

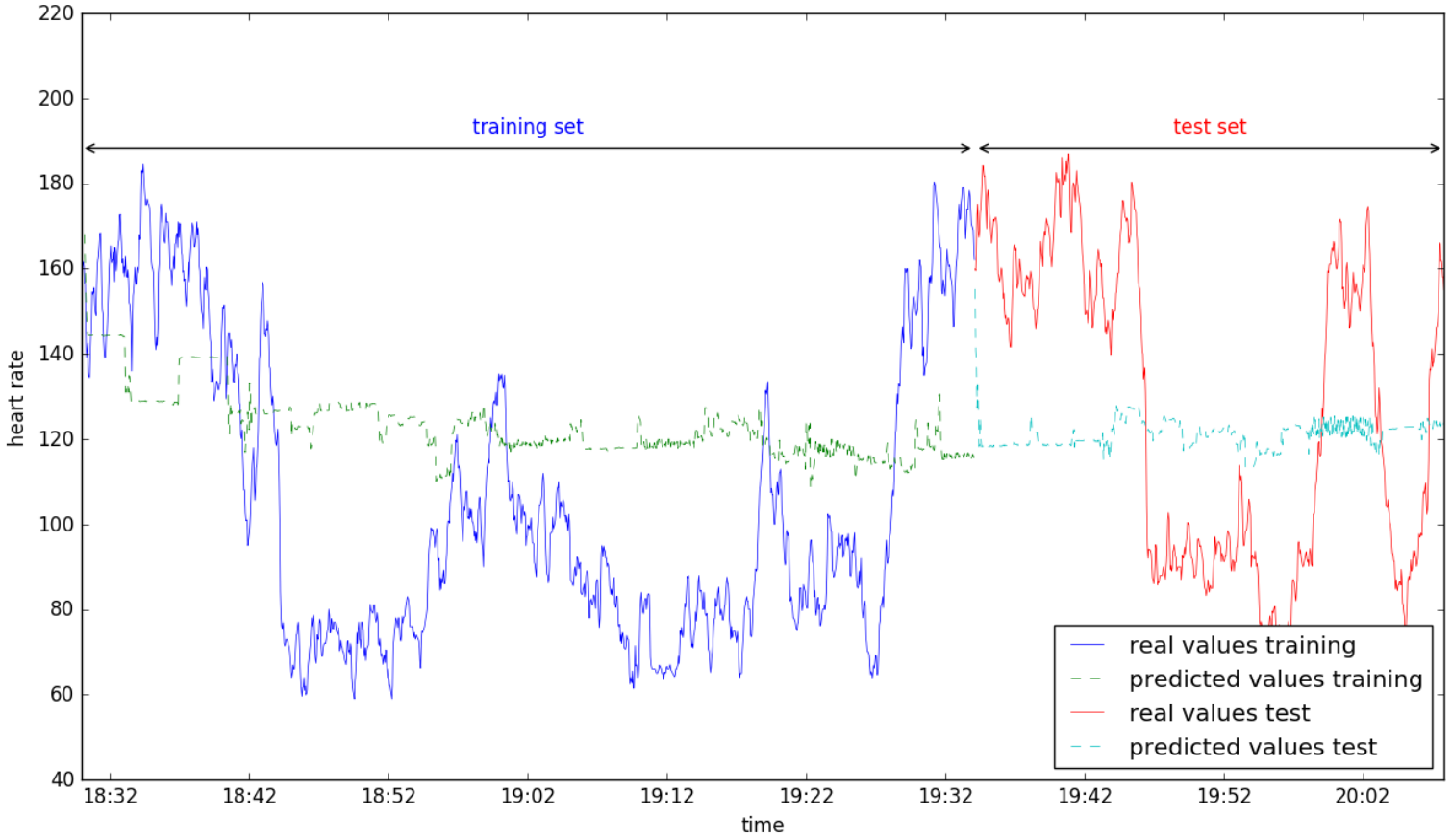
Results (1)

- Results:



Results (2)

- Time series:



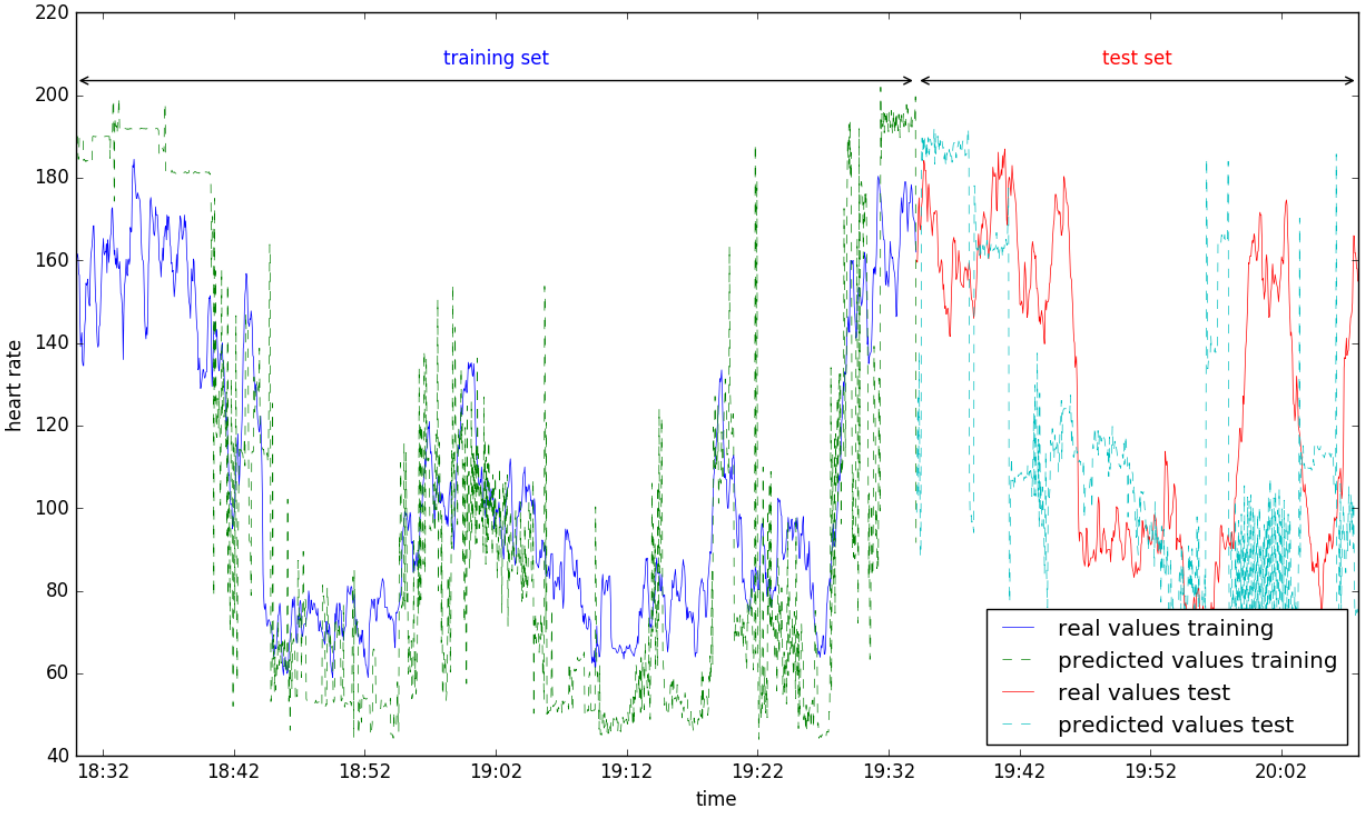
Results (3)

- Time series feature importance:

Approach	β
MA(1)	1.8606
MA(2)	2.2943
MA(3)	2.0588
MA(4)	1.2991
MA(5)	0.4741
acc_phone_x	-0.2286
acc_phone_y	-0.3136
acc_phone_z	0.1977
acc_watch_x	0.1259
gyr_phone_y	-0.1105
gyr_phone_z	-0.1282
labelOnTable	0.2781
labelSitting	-0.1099
mag_phone_y	-0.1317
press_phone_pressure	0.1173

Results (4)

- Recurrent Neural Network:



Results (5)

- Echo State Network: