Handling Noise and Missing Values in Sensory Data

MLQS: Chapter 3

This slide-deck is a modified version of the authors' slides from the MLQS book

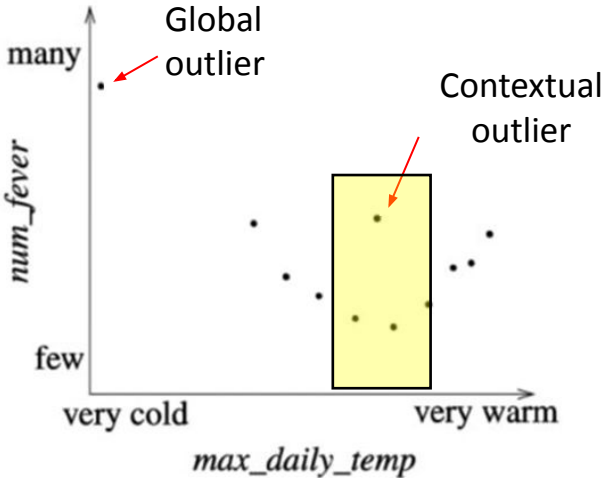
Overview

- Previously: we collected the data
- Today: Removing noise from the data
  - Removal of outliers
  - Imputation of missing values
  - Transform the data to select the most useful information

An outlier is an observation point that is distant from other observations

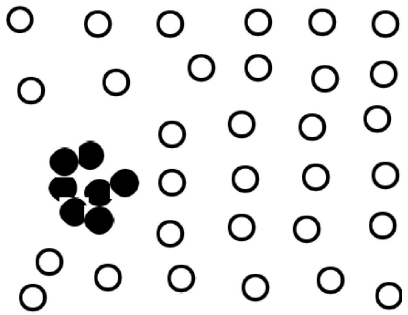
Global outliers

Global outliers
(=point anomaly)



Contextual outliers (=conditional outliers)
Context: grouping based on contextual attributes
(e.g., time/location, personal characteristics)

Collective outliers
(e.g., intrusion)



A subset of data objects deviate significantly from the entire dataset

Removal of outliers (1)

- Causes?
  - Measurement errors
    - Faulty measurement from sensors
    - For example, Arnold with a heart rate of 400 bpm
    - Must check whether a scale is good enough: valid/reliable? (or is there any systematic bias?)
  - Variability of the target values
    - Due to subject variations or random errors
    - For example, Arnold trying to push his limits with a heart rate of 190 bpm (say during a hard running instance)

More details about measurement errors:
https://www.healthknowledge.org.uk/public-health-textbook/research-methods/1a-epidemiology/sources-variation-measurement-control

Removal of outliers (2)

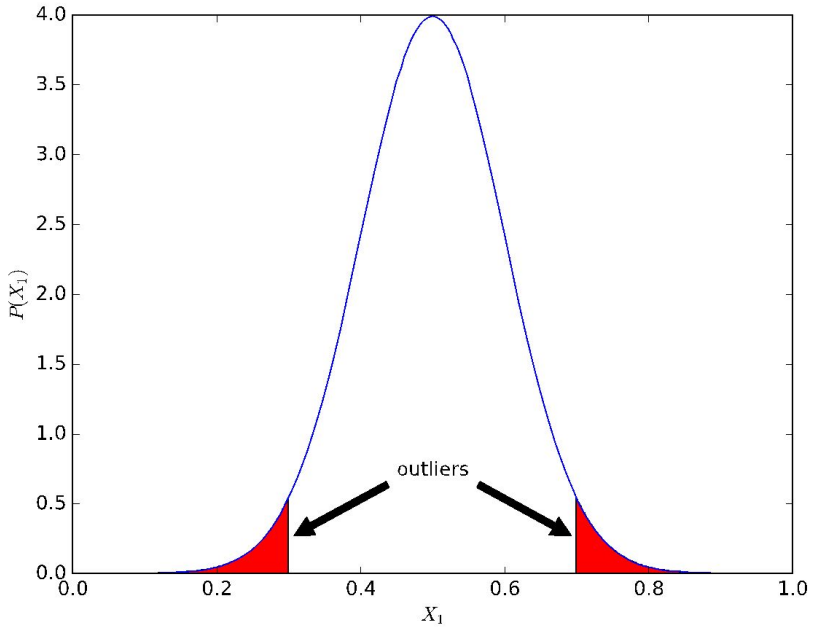
- Difference between measurement and variability outlier?
  - Former generated by *another mechanism*
- How to remove?
  - Domain knowledge (heart rate cannot be over 220)
  - Without domain knowledge (our focus)
- Have to be **cautious** as you do now want to remove valuable information

Removal of outliers (3)

- Categorizing outlier detection methods:
  - Are there **any labeled data**? (supervised vs. unsupervised)
  - **Are there any assumptions about normal data vs. outliers? (our focus)**
- Two types of outlier detection:
  - Distribution based (statistical): we assume a certain distribution of the data
  - Distance based (or proximity-based): we only look at the distance between data points

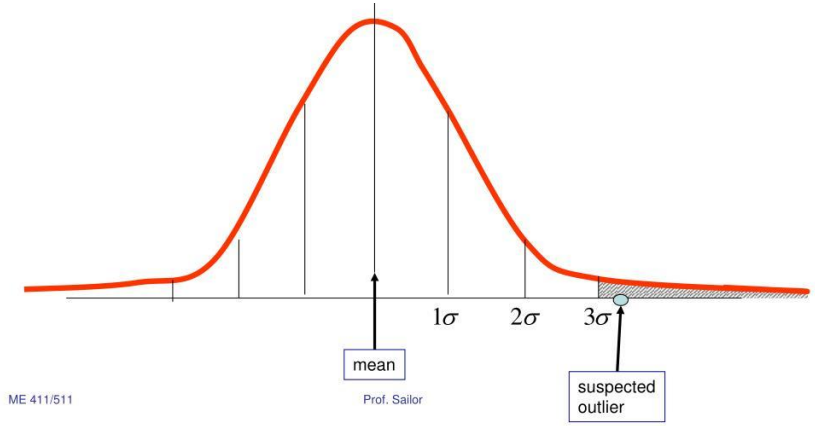
Distribution-based outlier detection (1)

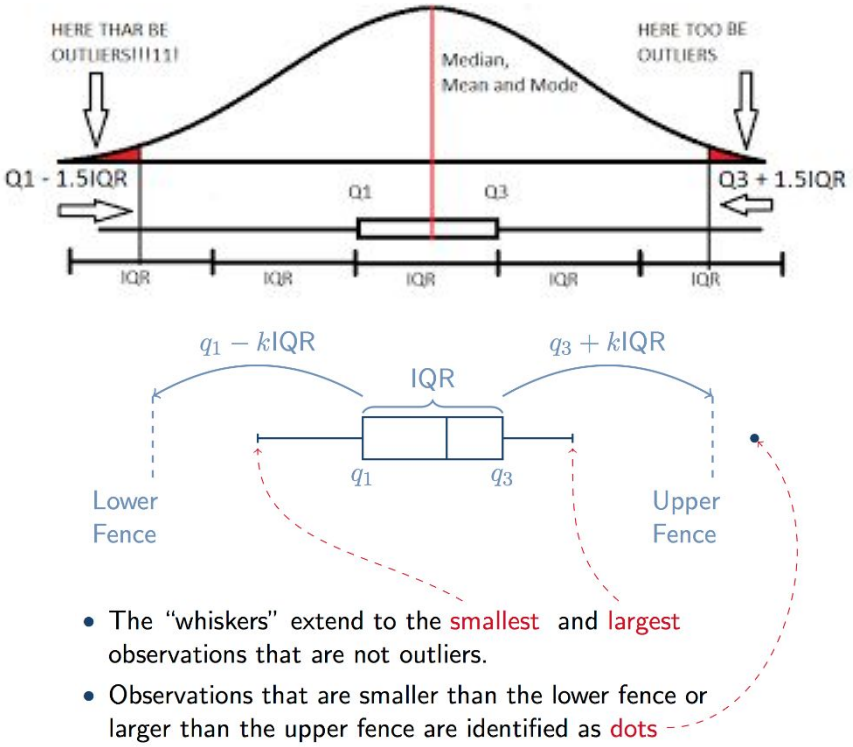
- Assume a normal distribution, single attribute (X_i)



Distribution-based outlier detection (1)

- Assume a normal distribution, single attribute (X_i)
- Values located at the both edges:
  - Commonly used rule-of-thumb: 3*σ from the mean value
  - Alternative: 3*MAD (median absolute deviation) from the median (see Leys, 2013)
- Boxplot: 1.5*IQR is approximately 3*σ (assuming normal distribution)

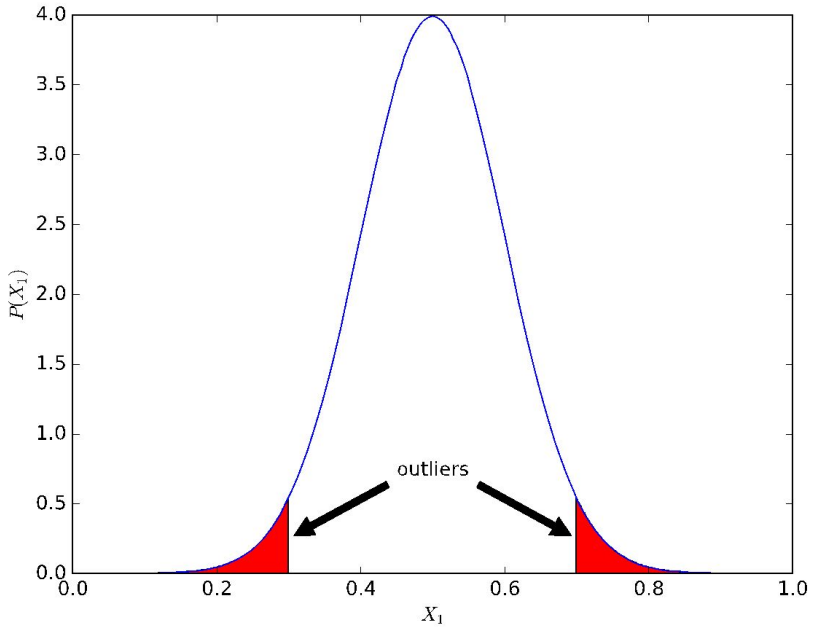




Leys 2013, Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median

Distribution-based outlier detection (1)

- Now let's take a look at Chauvenet's criterion
- Assume a normal distribution, single attribute (X_i)



Chauvenet's criterion (1)

- Take the mean and standard deviation for an attribute *j* in our dataset of size N:

$$\mu = \frac{\sum_{n=1}^{N} x_n^j}{N}$$

$$\sigma = \sqrt{\frac{\sum_{n=1}^{N} (x_n^j - \mu)^2}{N}}$$

Chauvenet's criterion (2)

- Take those values as parameters for our normal distribution
- For each instance *i* for attribute *j* compute the probability of the observation:

$$P(X \leq x_i^j) = \int_{-\infty}^{x_i^j} \frac{1}{\sqrt{2\sigma^2\pi}} e^{-\frac{(u-\mu)^2}{2\sigma^2}} \delta u$$

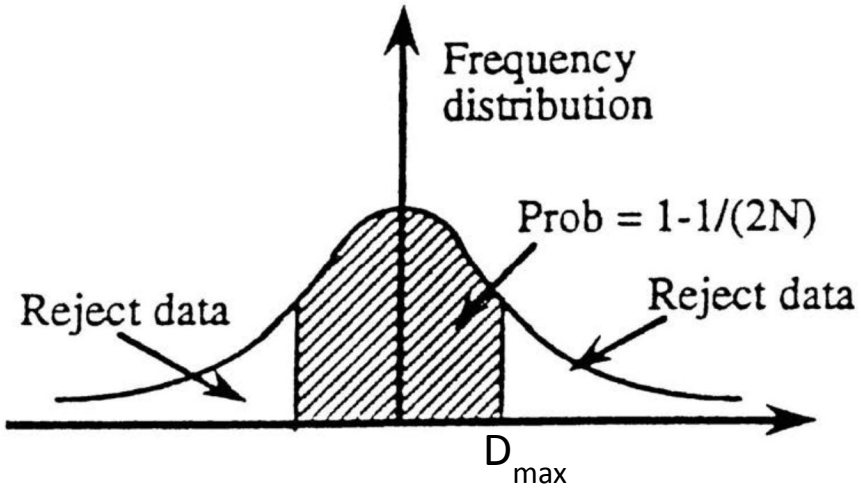
Chauvenet's criterion (3)

- Define it as an outlier when:

$$(1 - P(X \leq x_i^j)) < \frac{1}{c \cdot N}$$

$$P(X \leq x_i^j) < \frac{1}{c \cdot N}$$

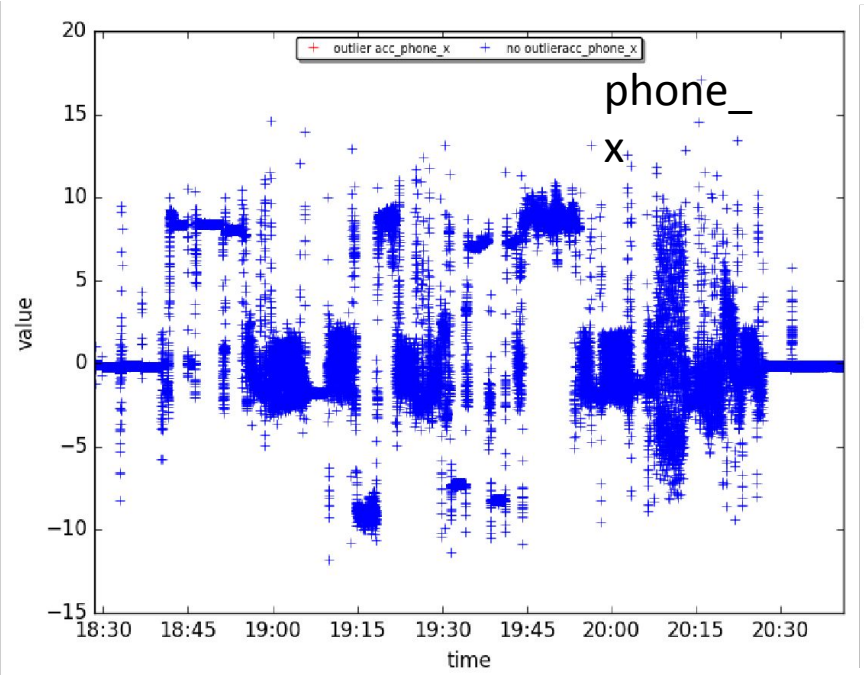
- Typical value for *c* is 2
- N: dataset of size N

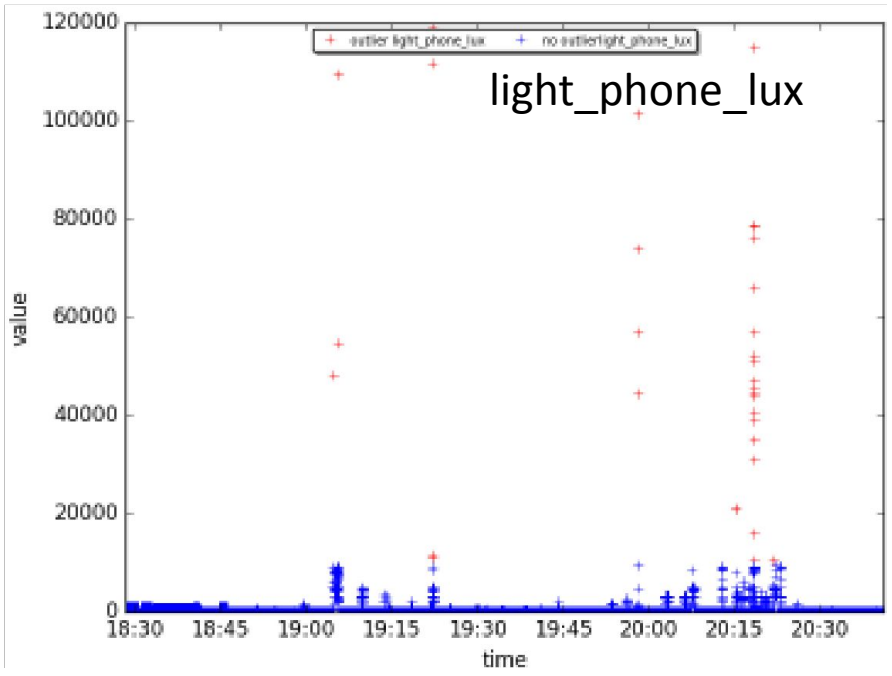


https://www.statisticshowto.com/chauvenets-criterion

Chauvenet's criterion (4)

- CrowdSignals example (c=2):





Winsorizing

- Winsorizing or winsorization is the transformation of statistics by substituting extreme values with less extreme values (e.g., 5%tile or 95%tile values) in the dataset to reduce the effect of possibly spurious outliers
- Consider the data set consisting of:
  - {92, 19, 101, 58, 1053, 91, 26, 78, 10, 13, −40, 101, 86, 85, 15, 89, 89, 28, −5, 41} **(N = 20, mean = 101.5)**
    - The data below the **5th percentile lies between −40 and −5**, while the data above the **95th percentile lies between 101 and 1053**
  - Then a **90% winsorization** would result in the following:
  - {92, 19, 101, 58, 101, 91, 26, 78, 10, 13, −5, 101, 86, 85, 15, 89, 89, 28, −5, 41} **(N = 20, mean = 55.65)**

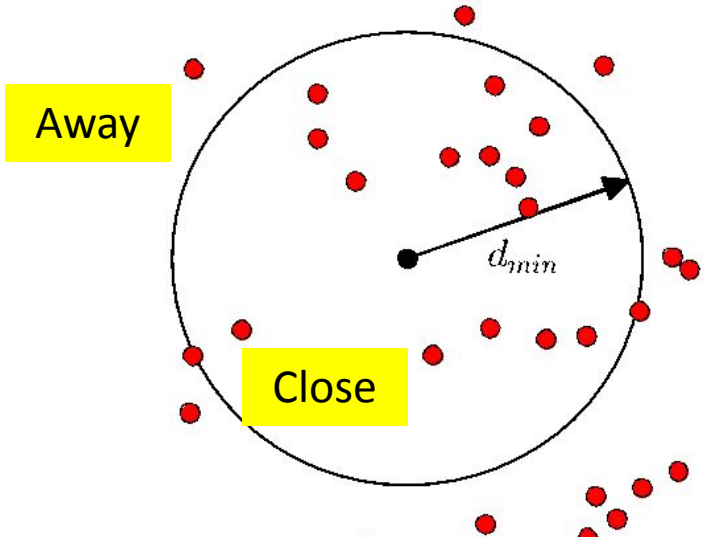
https://en.wikipedia.org/wiki/Winsorizing

Distance-based outlier detection (1)

- Let us move away from **distributions** and just consider the **distance** between points
- Consider the actual distance metrics later (Chapter 5), but e.g. think of Euclidean distance
- Use $d(x_a^j, x_b^j)$ to represent the distance between two values of an attribute *j* (i.e., x_a^j, x_b^j)

Simple distance-based approach (1)

- Points are close if they are within distance d_{min}
- Points are away if they are placed beyond distance d_{min}

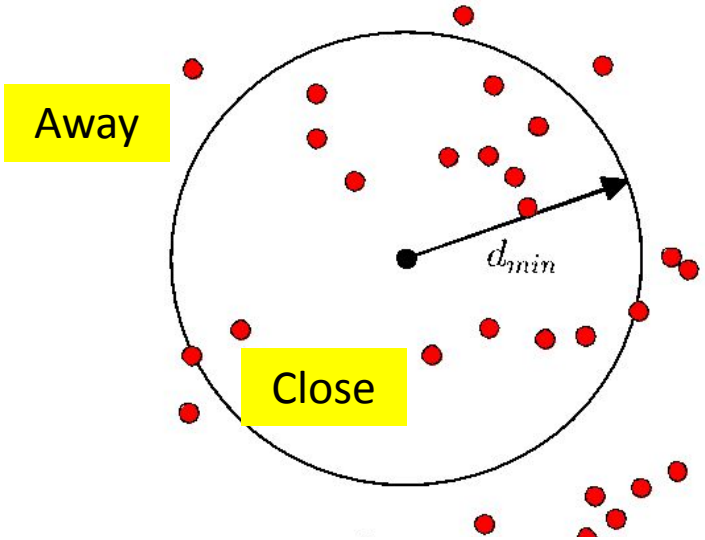


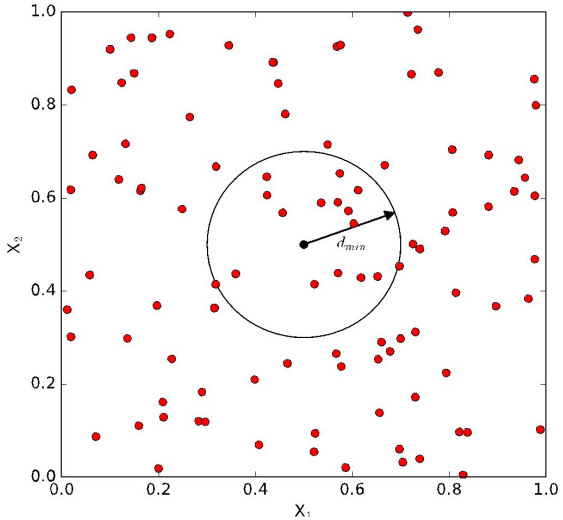
Simple distance-based approach (1)

- A point is an outlier
  - if a fraction f_{min} of N points are away (i.e., outside of d_{min})

  ⇔ this means that at least, (1- f_{min})*N points are close

  ⇔ if there are not enough close points, it's likely the point is an outlier





Simple distance-based approach (2)

- Formal:

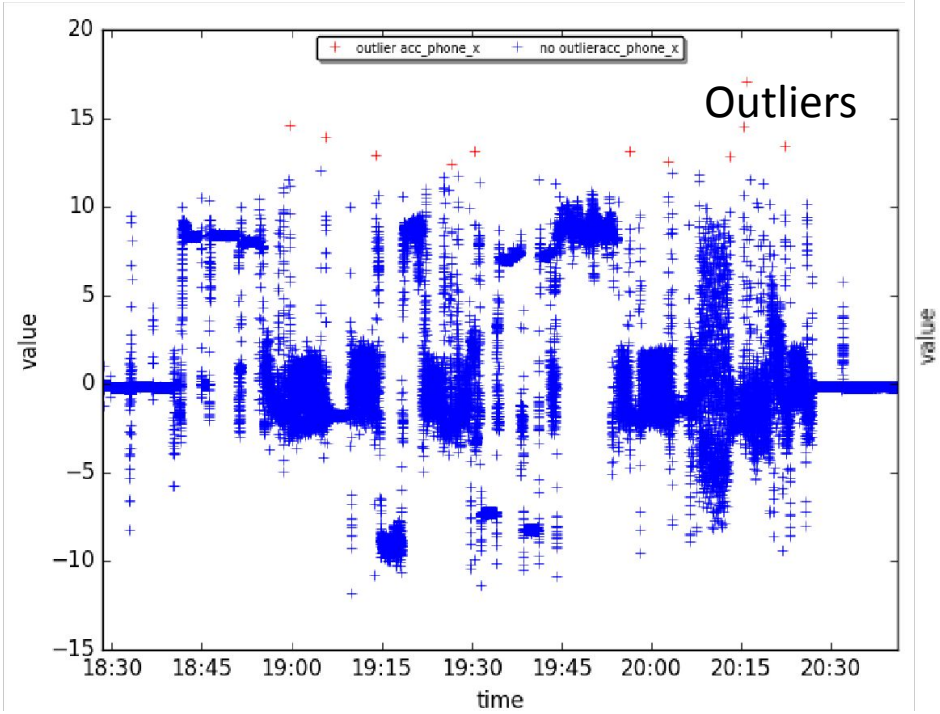
$$outlier(x_i^j) = \begin{cases} 1 & \frac{\sum_{n=1}^{N} d\_over(x_i^j, x_n^j, d_{min})}{N} > f_{min} \\ 0 & otherwise \end{cases}$$

with

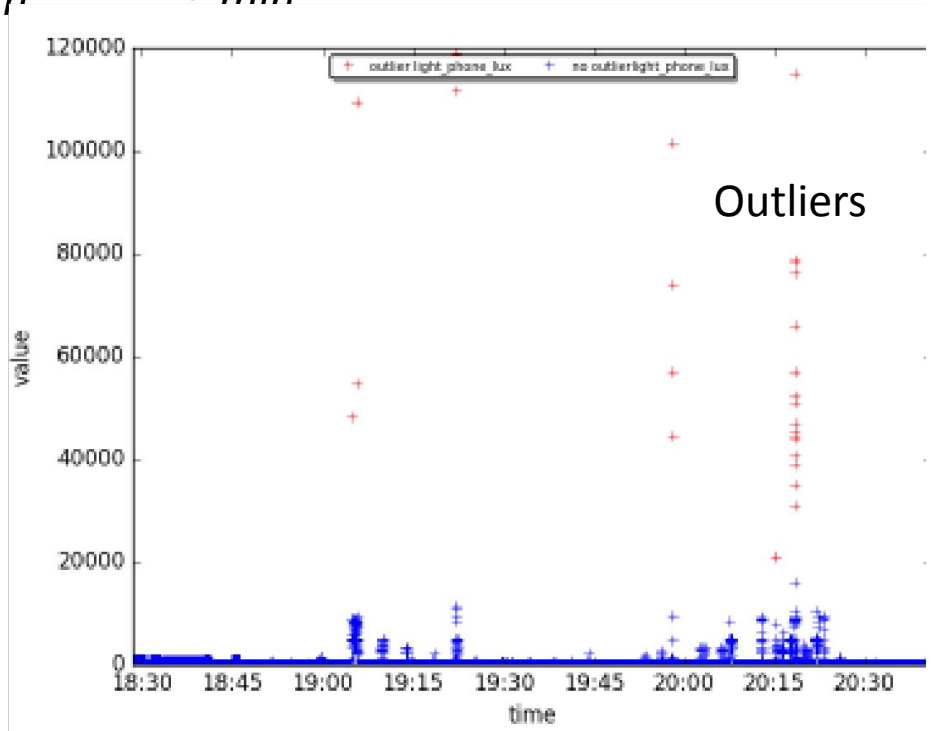
$$d\_over(x, y, d_{min}) = \begin{cases} 1 & d(x,y) > d_{min} \\ 0 & otherwise \end{cases}$$

Simple distance-based approach (3)

- CrowdSignal example (d_{min}=0.1, f_{min}=0.99)



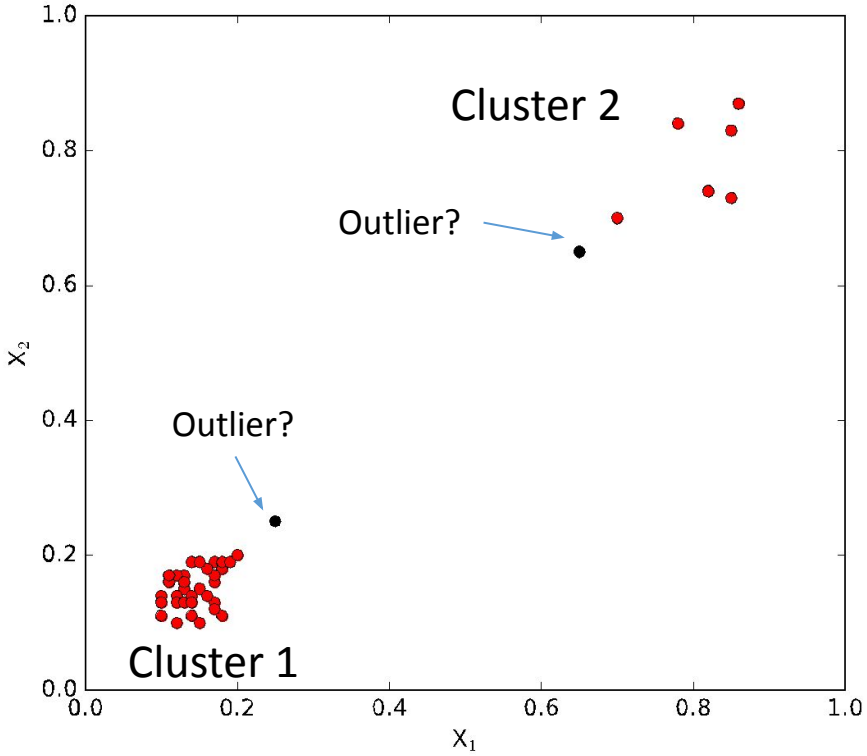
Acc_phone



Light_phone

Distance-based outlier detection (2)

- The previous approach did not take the local density into account, imagine:



Local outlier factor (1)

- Distance based outliers:
  "Outliers are further away from the normal data"
  - Use "distance to k nearest neighbors"

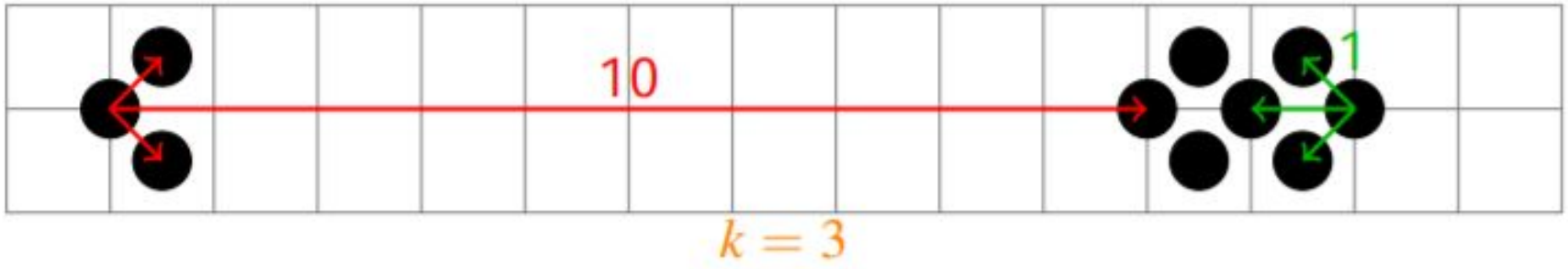
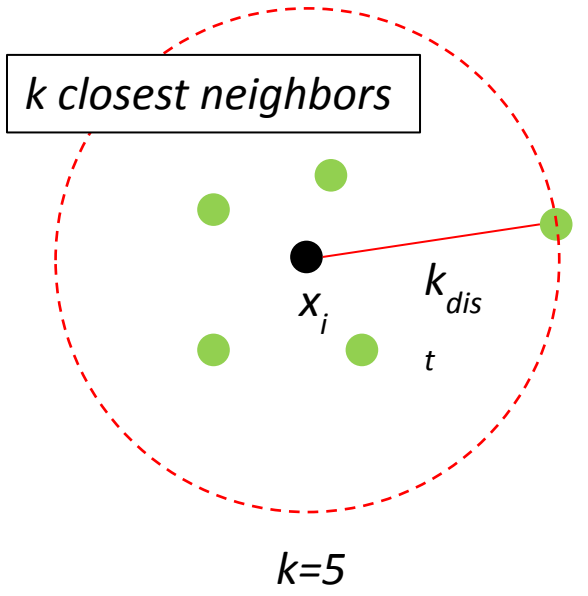


Illustration: https://www.dbs.ifi.lmu.de/Lehre/KDD/SS12/uebung/Tutorial06Outlier.pdf
Overview: https://webdocs.cs.ualberta.ca/~zaiane/courses/cau/slides/cau-Lecture7.pdf

Local outlier factor (1)

- Local outlier factor take this density into account
- First define the distance k_{dist} for a point x_i^j as the largest distance to one of its k closest neighbors:



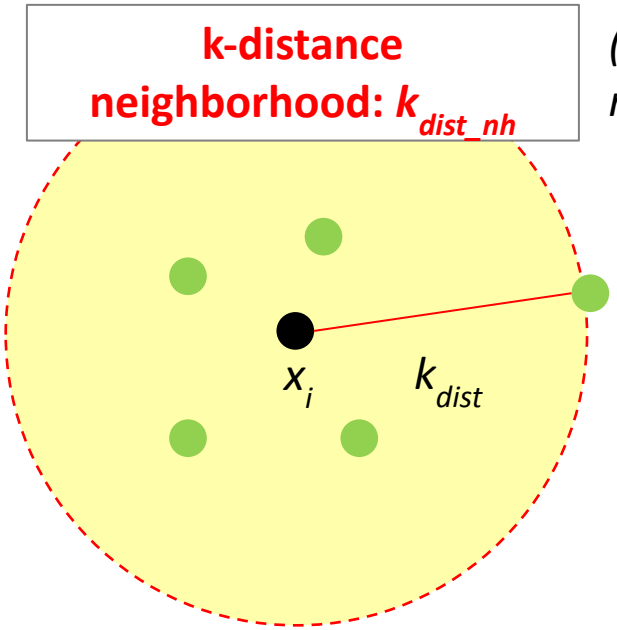
$$|\{x|x \in \{x_1^j, \ldots, x_{i-1}^j, x_{i+1}^j, \ldots, x_N^j\} \wedge d(x, x_i^j) \leq k_{dist}(x_i^j)\}| \geq k$$

$$|\{x|x \in \{x_1^j, \ldots, x_{i-1}^j, x_{i+1}^j, \ldots, x_N^j\} \wedge d(x, x_i^j) < k_{dist}(x_i^j)\}| \leq (k-1)$$

Local outlier factor (2)

- The set of neighbors of x_i^j within k_{dist} is called the **k-distance neighborhood k_{dist_nh}**

$$k_{dist\_nh}(x_i^j) = \{x | x \in \{x_1^j, \ldots, x_{i-1}^j, x_{i+1}^j, \ldots, x_N^j\} \wedge d(x, x_i^j) \leq k_{dist}(x_i^j)\}$$



(= k closest neighbors)

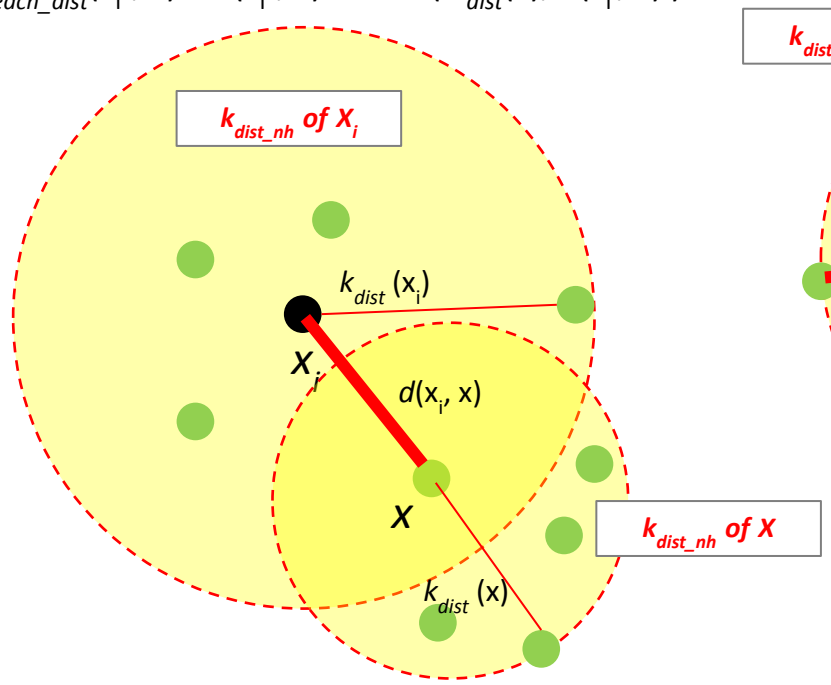
Local outlier factor (2)

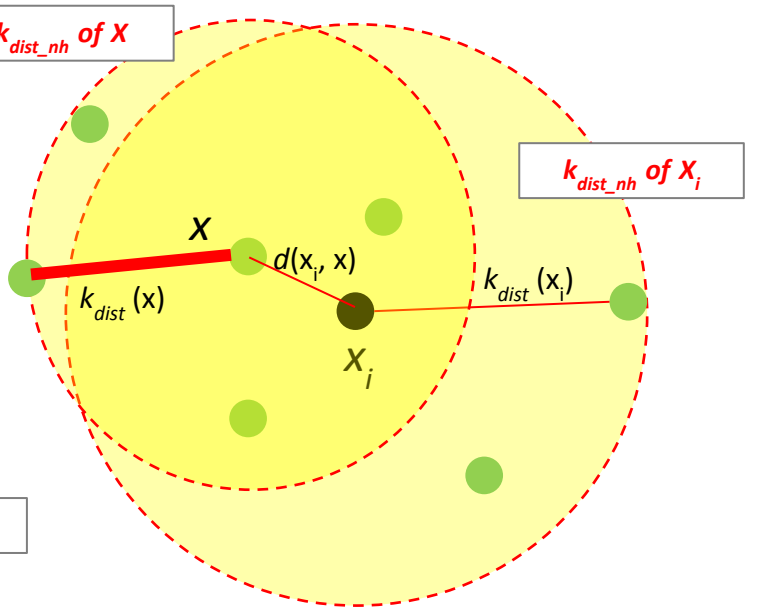
- Define the reachability distance of x_i^j from ***x*** as (x_i^j □ x)

$$k_{reach\_dist}(x_i^j, x) = max(k_{dist}(x), d(x, x_i^j))$$

  - A reachability distance from x is the real distance if the point x_i^j is not among the *k* nearest points of x; otherwise, it's $k_{dist}(x)$

$k_{reach\_dist}(x_i^j, x) = d(x_i^j, x) = \max(k_{dist}(x), d(x_i^j, x))$





$k_{reach\_dist}(x_i, x) = k_{dist}(x) = \max(k_{dist}(x), d(x_i, x))$

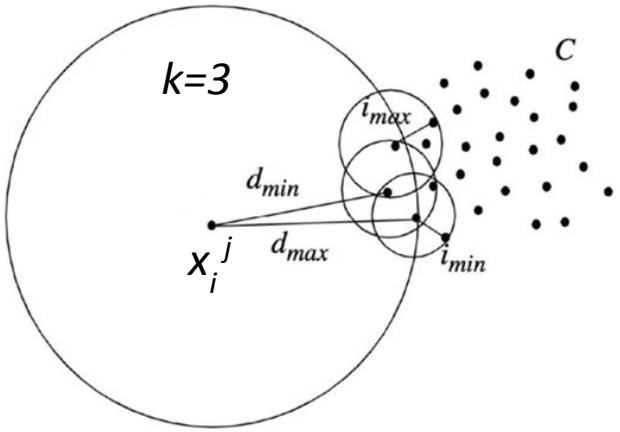
Local outlier factor (3)

- Now we define the local reachability distance of our point x_i^j:

$$k_{lrd}(x_i^j) = 1 / \left(\frac{\sum_{x \in k_{dist\_nh}(x_i^j)} k_{reach\_dist}(x_i^j, x)}{|k_{dist\_nh}(x_i^j)|} \right)$$

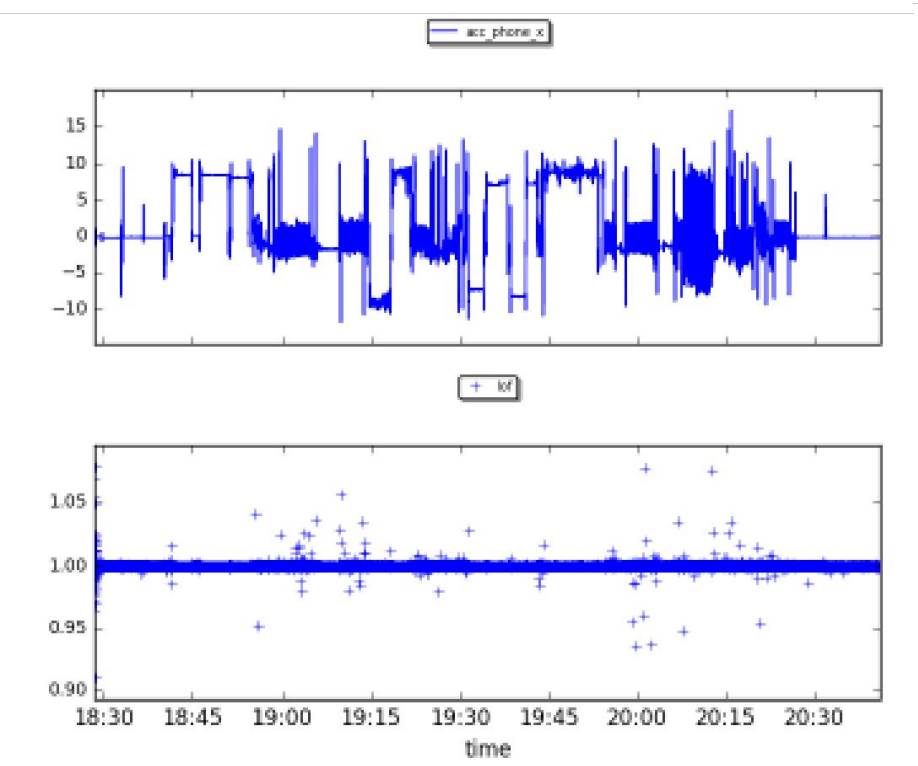
- And we compare this to the neighbors:

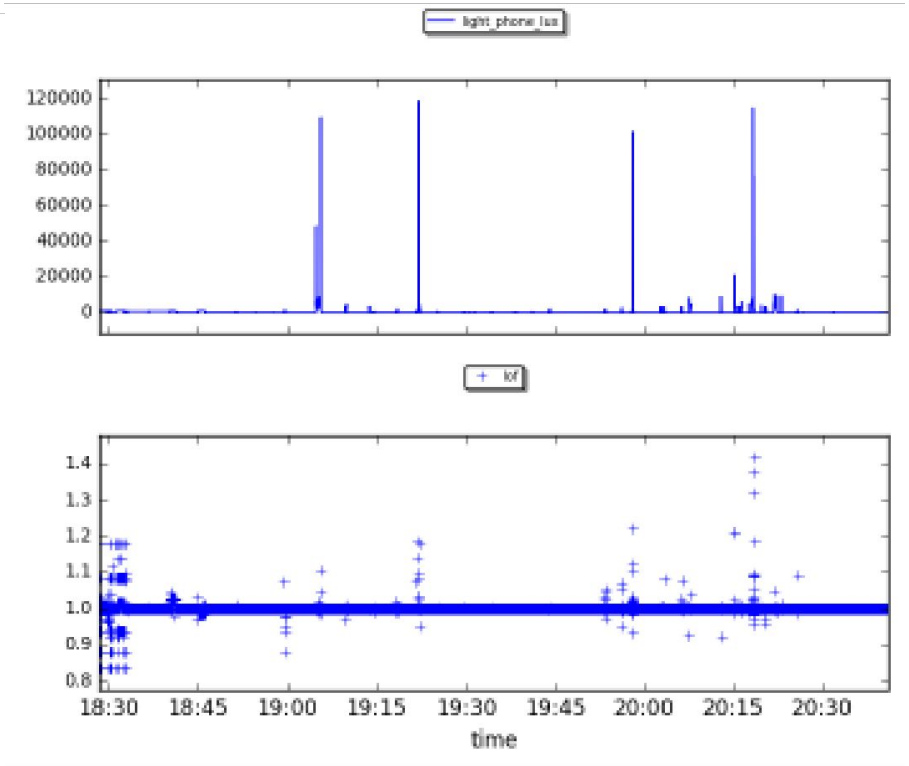
$$k_{lof}(x_i^j) = \frac{\sum_{x \in k_{dist\_nh}(x_i^j)} \frac{k_{lrd}(x)}{k_{lrd}(x_i^j)}}{|k_{dist\_nh}(x_i^j)|}$$



Local outlier factor (4)

- CrowdSignals case (k=5):





Outlier detection

- We remove all elements we have considered to be an outlier
- We replace them with the value missing

Missing values (1)

- Replace missing values by a substituted value (*imputation*)
- What should these values be?
  - mean (numeric)
  - mode (categorical and numeric)
  - median (numeric)

- *Multiple imputation for missing data in epidemiological and clinical research: potential and pitfalls, BMJ 2009; 338*
- *Flexible Imputation of Missing Data, Stef van Buuren, CRC Press*

attribute j of instance i

x_i^j

Missing values (2)

- We can also take more advanced approaches:
  - Use other attribute values in the same instance (Chap 7):

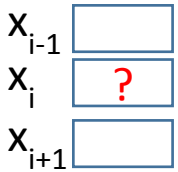
$$x_i^1, \ldots, x_i^{j-1}, x_i^{j+1}, \ldots \quad \Rightarrow \quad x_i^p \rightarrow x_i^j$$

  - Use values of the same attribute from other instances (need an ordered/temporal attribute):

$$x_1^j, \ldots, x_{i-1}^j, x_{i+1}^j, \ldots \quad \Rightarrow \quad x_N^j \rightarrow x_i^j$$

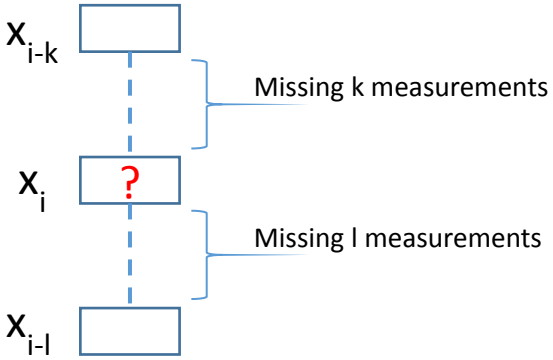
Missing values (3)

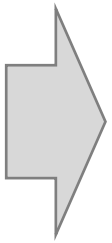
- Some examples of the second case in case of **a single missing measurement**:



$$x_i^j = \frac{x_{i-1}^j + x_{i+1}^j}{2}$$

- In case of multiple missing measurements:





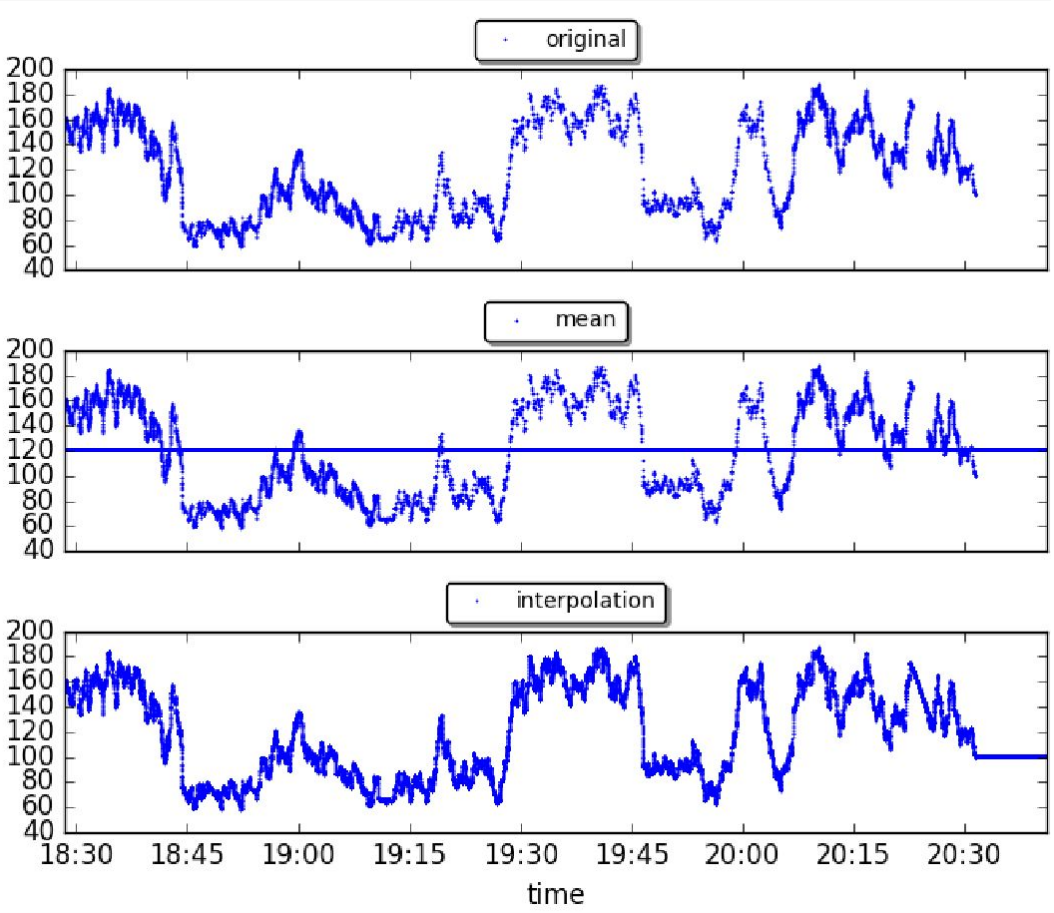
$$x_i^j = x_{i-k}^j + k \cdot \frac{x_{i+l}^J - x_{i-k}^J}{(k+l)}$$

(rate of change)

Here, we assume fixed sampling rate

Missing values (4)

- CrowdSignal example:



Outlier detection + imputation

- Approaches that combine outlier detection and value imputation exist as well
- The Kalman filter is a well-known one:
  - it estimates expected values based on historical data
  - if the observed value deviates too much (i.e., an outlier) we can impute with the expected value

Rudolf E. Kálmán

From Wikipedia, the free encyclopedia

Rudolf Emil Kálmán[3] (May 19, 1930 – July 2, 2016) was an American electrical engineer, mathematician, and inventor. He was most noted for his co-invention and development of the Kalman filter, a mathematical algorithm that is widely used in signal processing, control

Kalman filter (1)

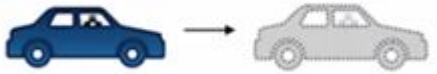
- Assumption
  - Some latent state s_t which can have multiple components
  - Our quantified self data x_t is the measurement result about this state
- For example:
  - s_t is Arnold's presence at a position and velocity
  - x_t is the GPS data and step counter (observation or measurement)
- Key idea of a Kalman filter
  - We know the underlying process model and measurement model; why don't we make use of this "prior knowledge" to better calibrate our data?

Linear Process Model

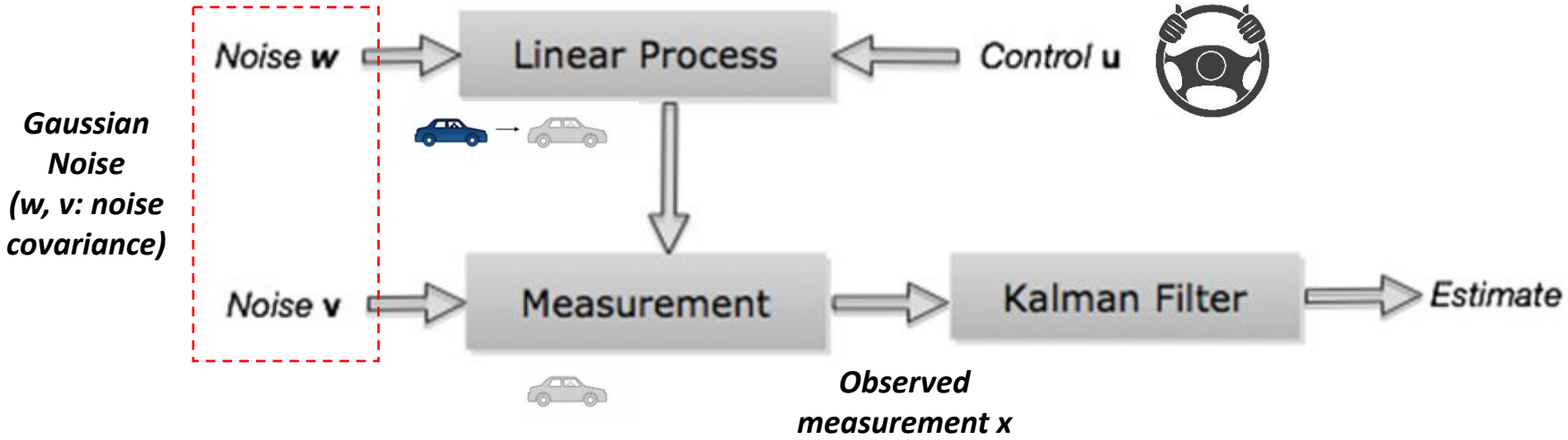
Next value of a state is defined as:

- S_t is system state (e.g., position & speed)
- u_t is a control input state (e.g., acceleration)
- F_t and B_t are matrices (given)
- w_t is white noise (given)

$$s_{t+1} = F_t s_t + B_t u_t + w_t$$



Linear Process (e.g., Motion)



Measurement Model

$$x_t = H_t s_t + v_t$$

Measurement model (e.g., location/speed sensing)

Measurement associated with s_t is:

- v_t is white noise

Kalman filter (3)

$x_t = H_t s_t + v_t$

Measurement model

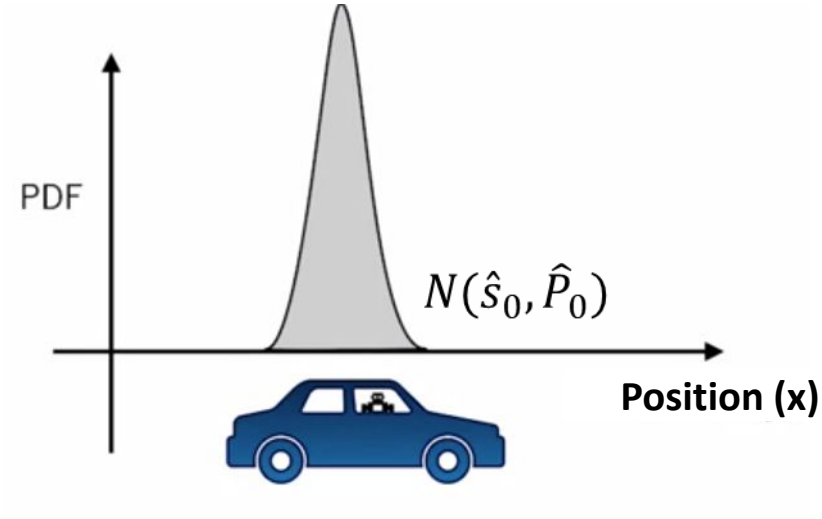
Measurement associated with s_t is:

- v_t is white noise

$\mathbf{x_k} = [1\ \ 0]\ \mathbf{s_k} + \mathbf{v_k}$

$N(\hat{s}_0, \hat{P}_0)$

Measurement model (position observation)



Position

$$\mathbf{s} = \begin{bmatrix} p \\ \frac{dp}{dt} = \dot{p} \end{bmatrix} \qquad \mathbf{u} = a = \frac{d^2p}{dt^2}$$

State

Speed

Acceleration

$$\mathbf{s}_k = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \mathbf{s}_{k-1} + \begin{bmatrix} 0 \\ \Delta t \end{bmatrix} \mathbf{u}_{k-1} + \mathbf{w}_{k-1}$$

Linear Process Model (e.g., Motion)

Kalman filter (3)

- For the noise we assume that:

$$w_t = \mathcal{N}(0, Q_t)$$
$$v_t = \mathcal{N}(0, R_t)$$

- **Phase #1:** Prediction of state and error (using the **process model**)
  - Predict a next state (denoted by a hat):

$$\hat{s}_{t|t-1} = F_t \hat{s}_{t-1|t-1} + B_t u_t$$

  - Estimate prediction error (matrix of variances and co-variances):

$$P_{t|t-1} = \mathbb{E}\left[(s_t - \hat{s}_{t|t-1})(s_t - \hat{s}_{t|t-1})^T\right]$$

$$(\square\ F_t P_{t-1} F_t^T + Q_t)$$

Kalman filter (4)

- **Phase #2:** Correction of state and error (using the **measurement model**)
  - Calculating **the error** based on our prediction of the state

$$e_t = x_t - H^T \hat{s}_{t|t-1}$$

Observed measurement *Predicted measurement*

  - Given this error, we get an **updated prediction** of our state

$$\hat{s}_{t|t} = \hat{s}_{t|t-1} + K_t e_t$$

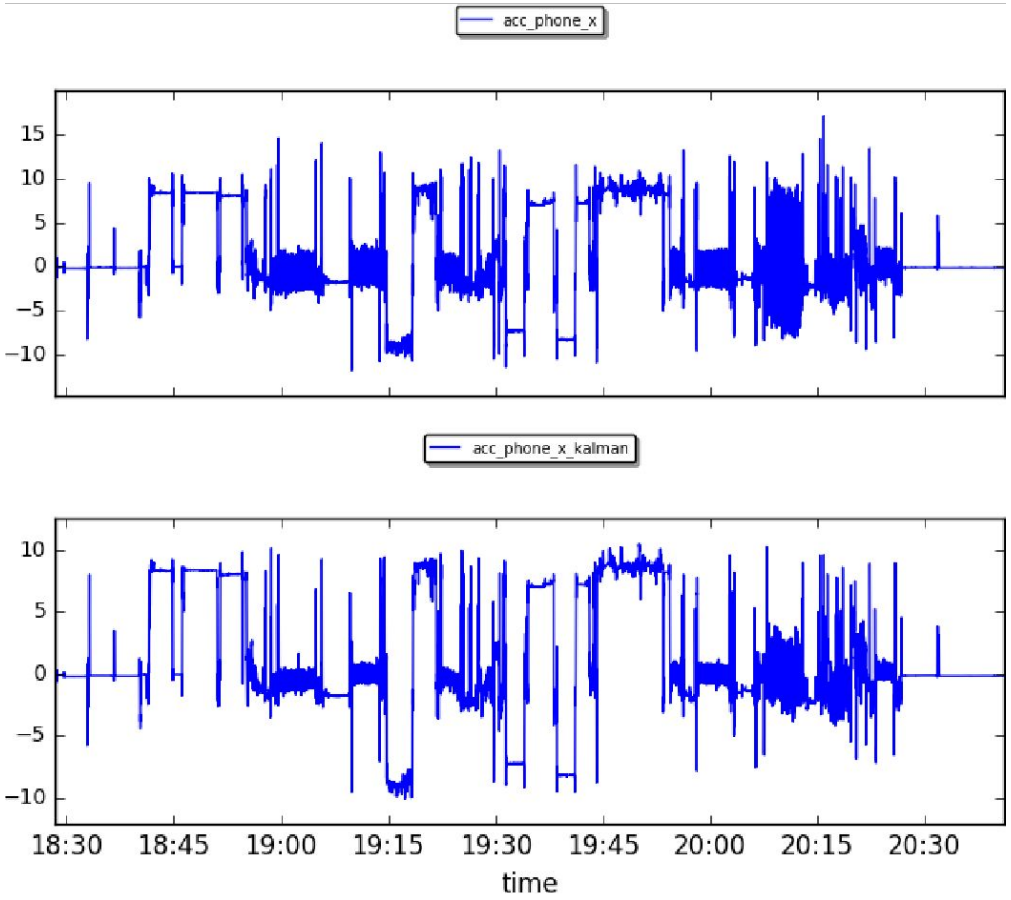
    - *Kalman gain:* K_t is a matrix derived based on an algorithm for which $P_{t|t-1}$ is used (i.e., $\sigma_{predicted} / (\sigma_{predicted} + \sigma_{measured})$

Kalman filter (5)

- Of course the matrices contain models (that we might not always know) but you can even do without (one measurement directly related to one latent state)
- Once we observe a large error, our x_t might be of and we can substitute it by the expected value
- Nice more extensive explanation:
  - http://www.bzarg.com/p/how-a-kalman-filter-works-in-pictures/

Kalman filter (6)

- CrowdSignal example:



Transforming the data (1)

- Even though we have removed the outliers and imputed missing values we could still suffer from noise in our dataset that could distract the learning process
- Approaches exist that filter our this more subtle noise
  - Lowpass filter
  - Principal Component Analysis

Lowpass filter (1)

- Main idea: some data has periodicity (e.g. walking, running)
- You can decompose a series of values into different periodic signals:
  - come with their own frequency
  - we will see about this in Chapter 4
- Some frequencies might be more interesting than others

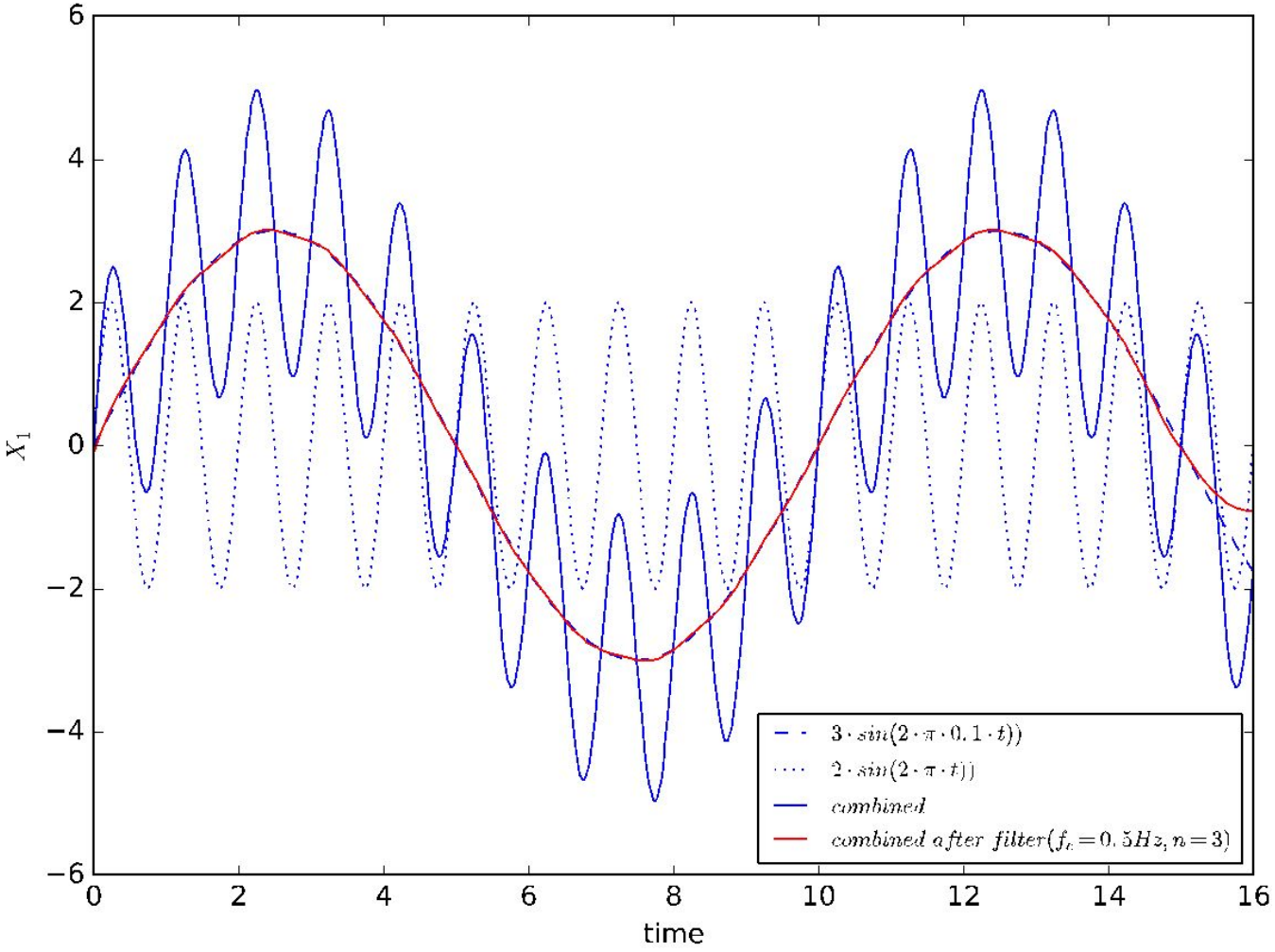
Lowpass filter (2)

- For example: we do not care about running (higher frequency), but we do care about walking
- We can filter out the higher frequency data
- The lowpass filter does exactly this:

$$|G(f)|^2 = \frac{1}{1 + (f/f_c)^{2n}}$$

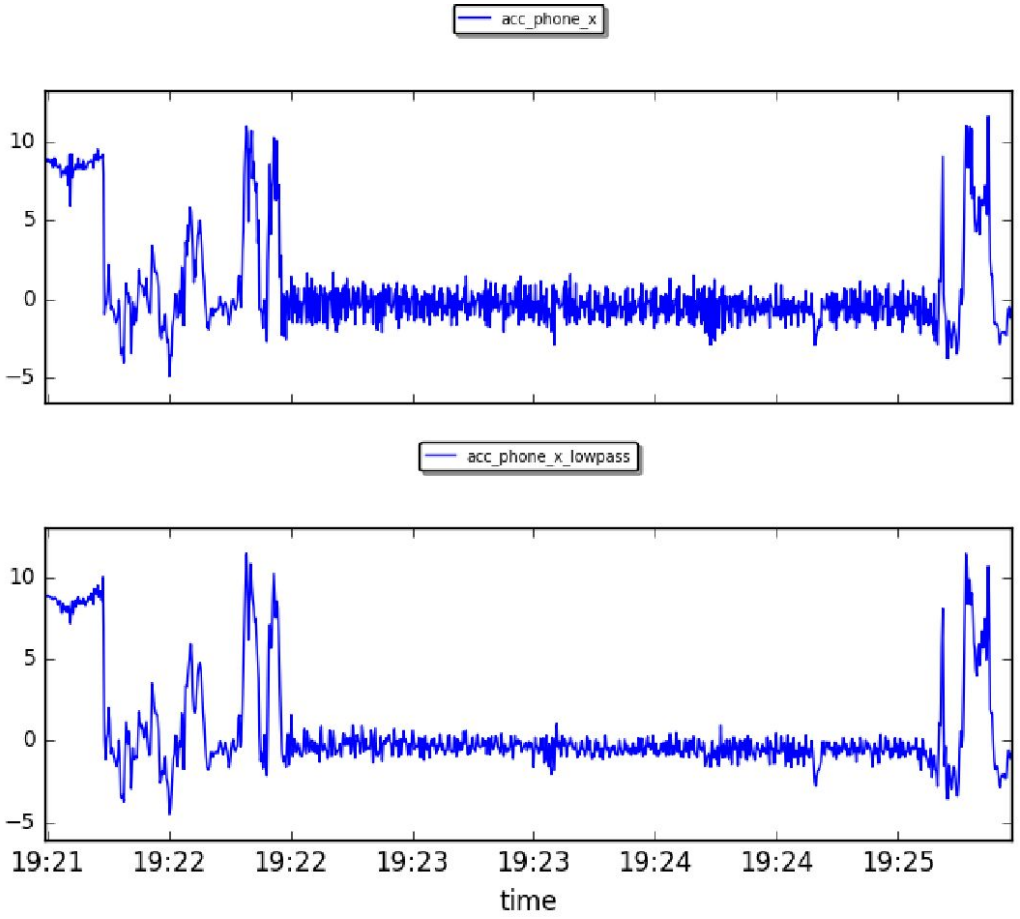
  - $|G(f)|$ is the magnitude of the filter
  - f_c is the cutoff frequency
  - n is the order of the filter

Lowpass filter (3)



Lowpass filter (4)

- CrowdSignal example (f_c=1.5Hz, n=20):



Transforming the data (2)

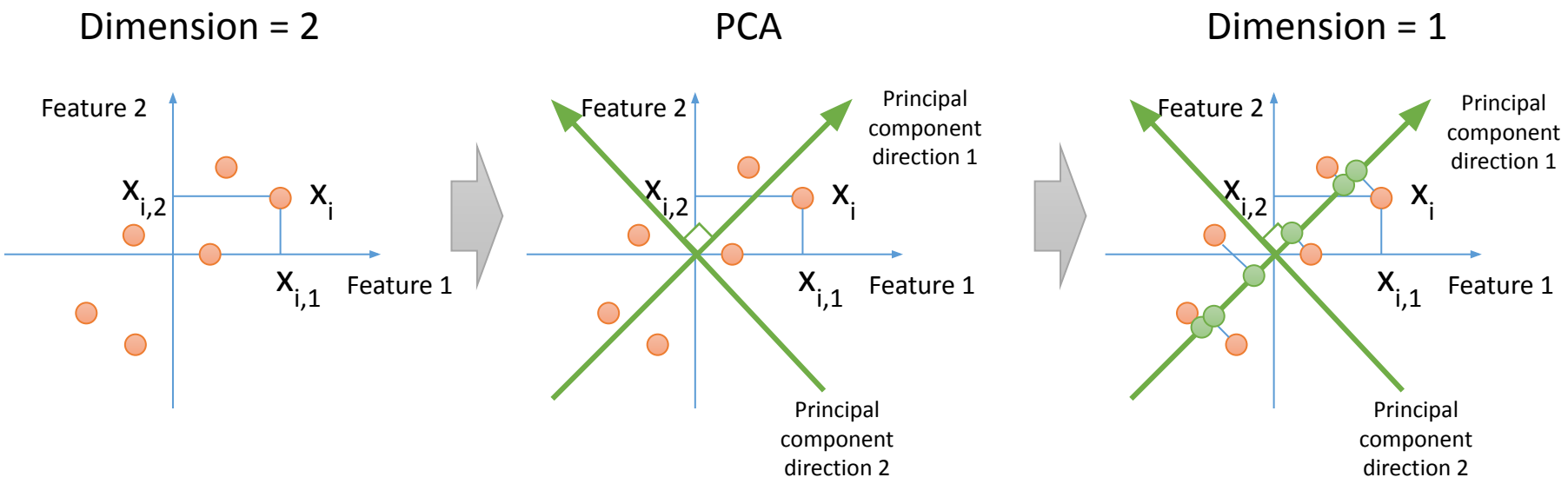
- We can also apply principal component analysis:
  - find new features that explain most of the variability in our data
  - select the number of components based on the explained variance
  - since most are familiar, I will not provide all details, see the book

Dimension Reduction

- Dimension reduction
  - Reduce *n* dimensional data into *k* dimensional data
- Why?
  - Feature extraction
    - Find a better data representation for the machine learning algorithm intended to use, since the original representation might not be the best one
    - Example: finding linear combinations of the original ones, as in PCA (Principal Component Analysis; unsupervised) or LDA (Linear Discriminant Analysis; supervised), as well as nonlinear combinations, like Kernel PCA, t-SNE and autoencoders
  - Feature fusion
    - Combine variables to remove redundant and irrelevant information, which is useful when there are multitude of data collected (e.g., mobile and wearable sensors)

Dimension Reduction

- Principal Component Analysis (PCA):
  reduce *n* dimensional data into *k* dimensional data



(projecting original data to only principal component direction 1)

Dimension Reduction

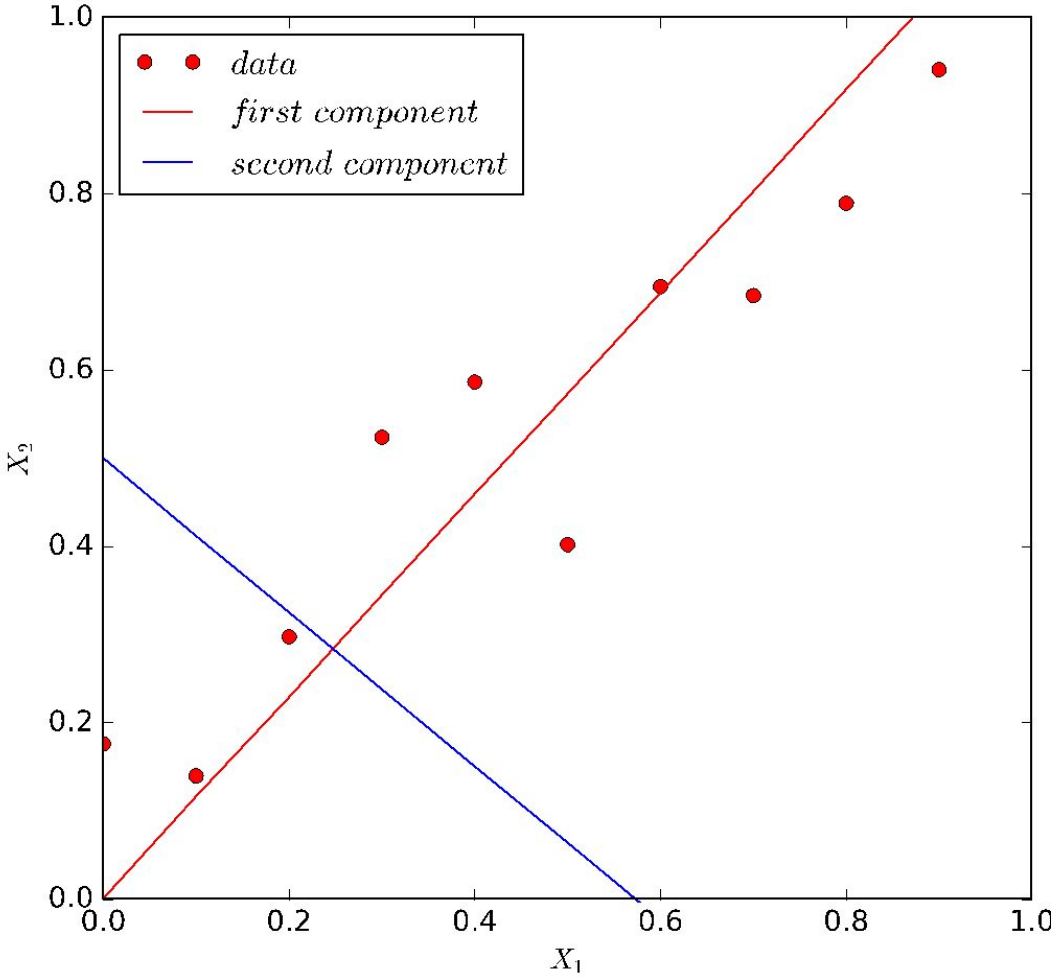
- Principal Component Analysis: reduce "*n*" dimensional data into "*k*" dimensional data
  - Find some orthonormal matrix **P** where **Y=PX** such that $\mathbf{C_Y = 1/(n-1)*YY^T}$ is diagonalized where the rows of **P** are the principal components of **X** (Shlens, 2005)
    - Diagonal terms (signals): large values correspond to interesting dynamics (correlation w/ principal components – think of projection!)
    - Off-diagonal terms (redundancy): large values correspond to high redundancy
  - If we choose **P** to be eigenvectors of $\mathbf{XX^T}$, we can diagonalize $\mathbf{C_Y}$ where $\mathbf{XX^T}$ is a covariance matrix that captures the correlations between all possible pairs of measurements

See Shlens (2005) "A Tutorial on Principal Component Analysis"

Dimension Reduction

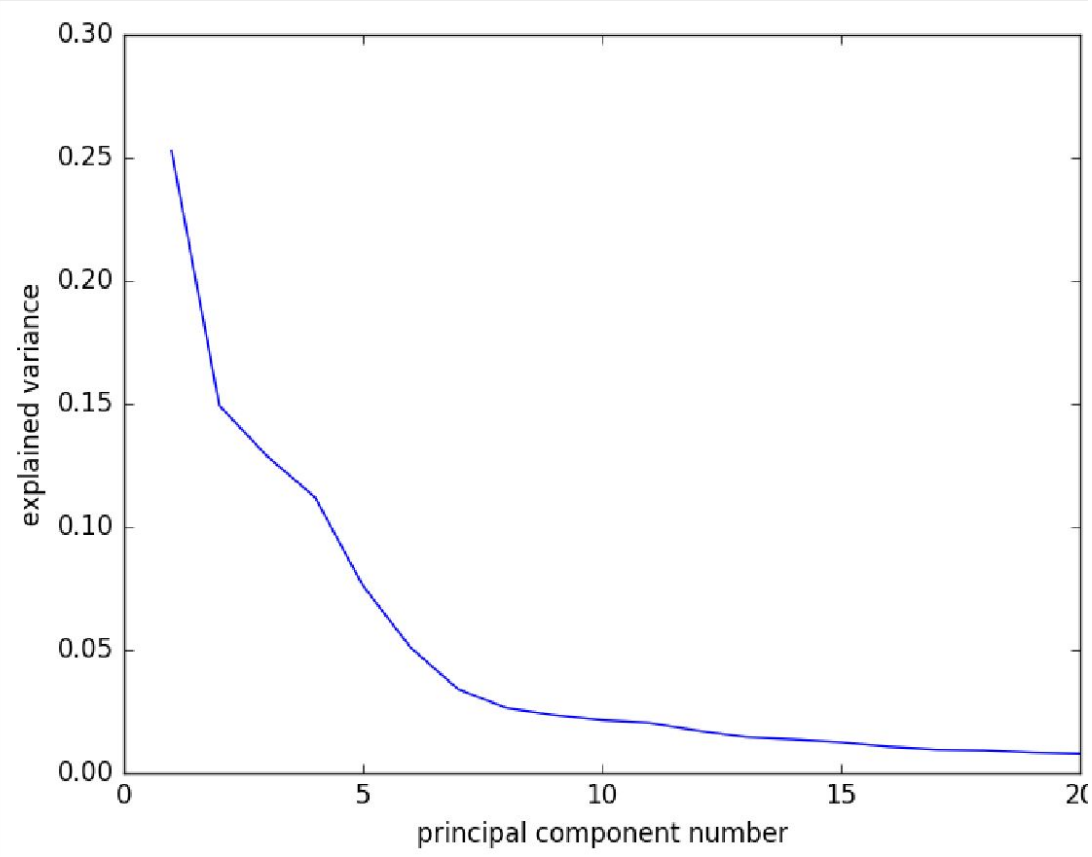
- Good for data compression (if data size is too big; and sparse?) or visualization (if it is 2D or 3D)
- "k" is chosen based on % of variance retained
- A bad use of PCA: use it to prevent over-fitting
  - If we retain most of the variance, this should work; but we may throw away some crucial data (after PCA, it's hard to know which feature is missing)
  - Initially, see how a system performs without PCA; and if you have a reason to believe that PCA will help should you then add PCA
  - Recommendation: apply PCA with "reasons"

Principal component analysis (1)



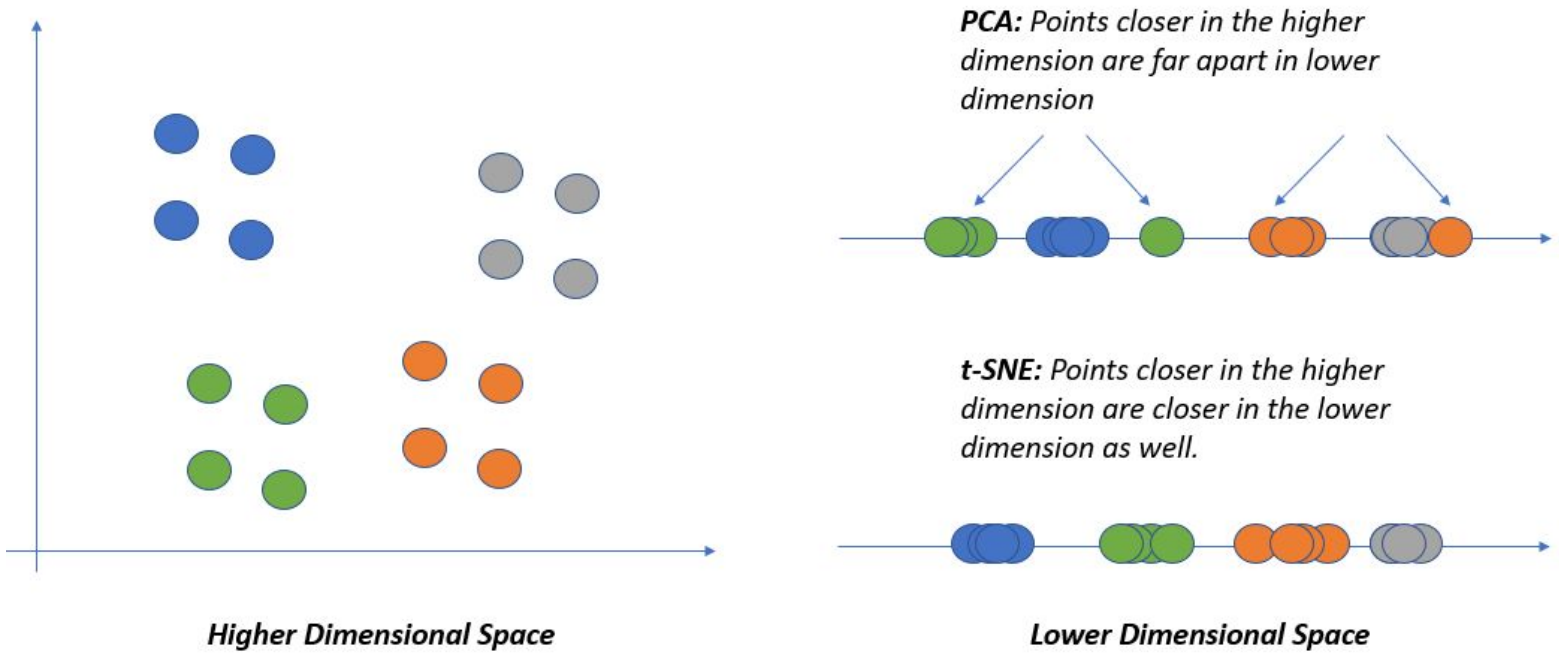
Principal component analysis (2)

- CrowdSignals:



Beyond PCA: t-SNE

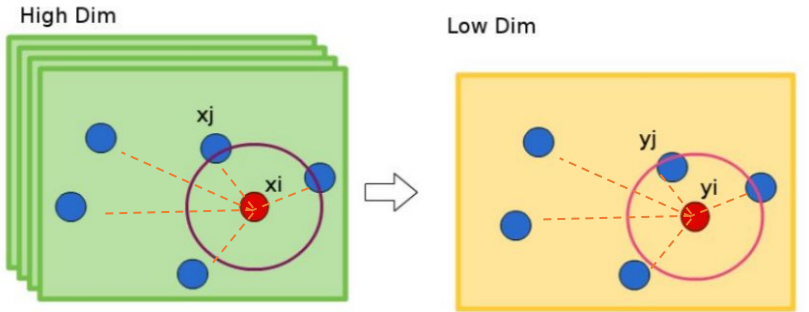
- PCA can't guarantee "distance" similarity in low dimension space
- t-SNE (t-Distributed Stochastic Neighbor Embedding) helps



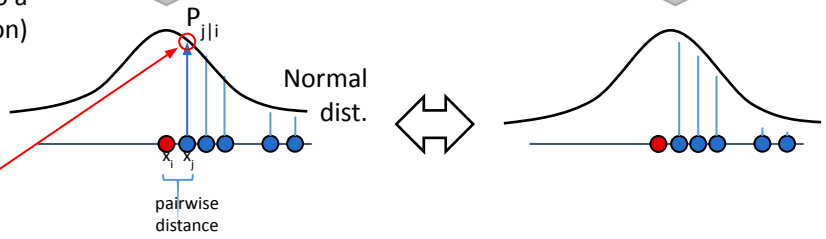
https://medium.com/@prasunbiswas90/improving-convolution-neural-networks-cnn-accuracy-using-t-sne-a4222d306d8

Beyond PCA: t-SNE

- **How t-SNE works?**
  - **Measure pairwise similarity distributions** in both high-dimensional and low-dimensional objects (known as stochastic neighbor embedding)
    - Avoid crowding issues (too many points around the edges) by using t-distribution (less sharp head, more flat tails vs. normal dist.)
  - **Ensure** that these "similarity" distributions are as similar as possible
    - Minimize **KL divergence** between high- and low-dim distributions



Stochastic Neighbor Embedding
(e.g., mapping each pairwise distance to a probability value using a normal distribution)

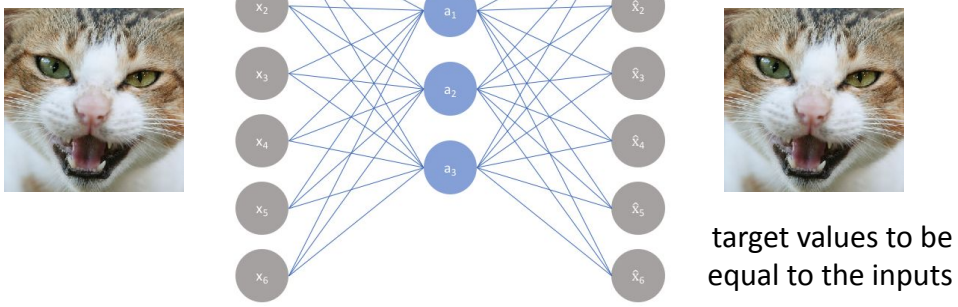


$$P_{j|i} = \frac{\exp\left(-||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||^2/2\sigma_i^2\right)}{\sum_{k\neq i}\exp\left(-||\mathbf{x}^{(i)} - \mathbf{x}^{(k)}||^2/2\sigma_i^2\right)}$$

The probability that point x_i chooses x_j as its neighbor

Autoencoder

- An unsupervised learning algorithm based on a neural network that applies backpropagation for training, by setting the target values to be equal to the inputs

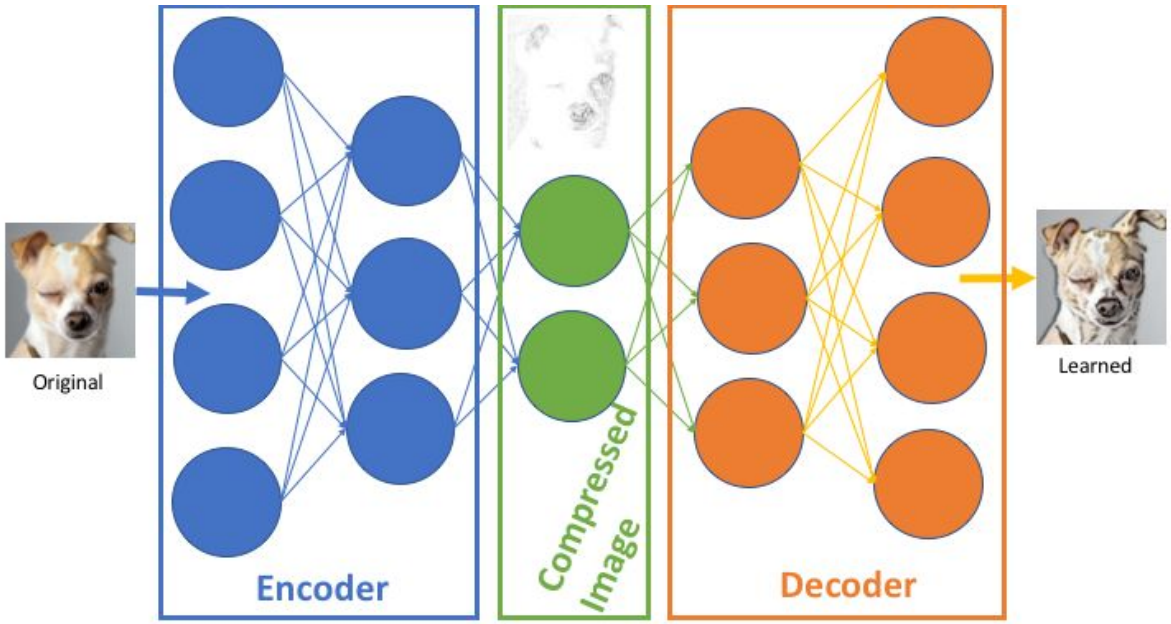


target values to be equal to the inputs

Neural Network

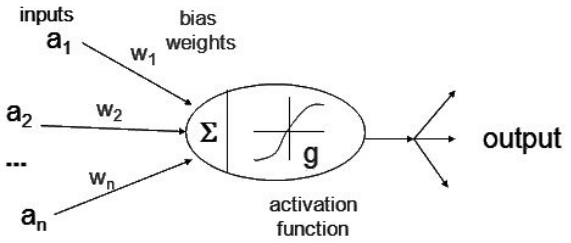
Autoencoder

- An unsupervised learning algorithm based on a neural network that applies backpropagation for training, by setting the target values to be equal to the inputs

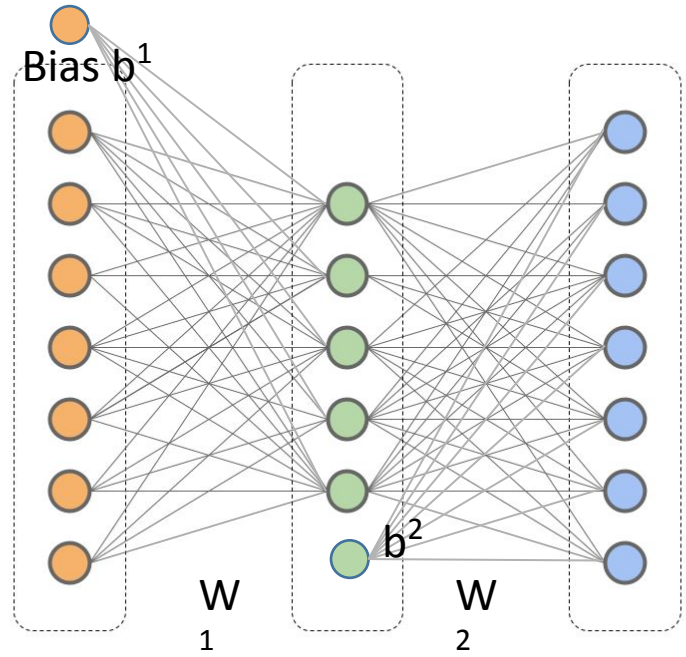


latent feature space

Autoencoder



- Simple autoencoder



Simple autoencoder:

$$y = f(x) = s_1(W^{(1)}x + b^{(1)}),$$
$$r = g(y) = s_2(W^{(2)}y + b^{(2)}),$$

x: input

s_1, s_2: activation function

$W^{(1)}$, $W^{(2)}$: weight matrix

b: bias

r: output

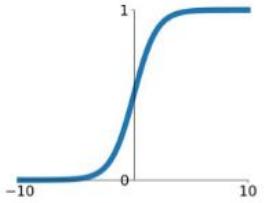
Parameter training w/ loss function

$$\mathcal{J}(W, b; S) = \sum_{x \in S} \mathcal{L}(x, (g \circ f)(x))$$

Autoencoder: Activation Function

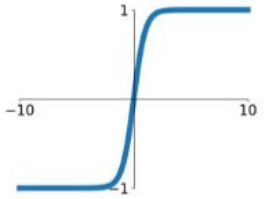
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



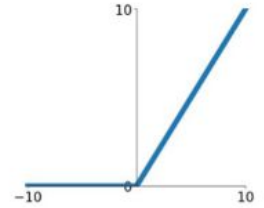
tanh

$$\tanh(x)$$



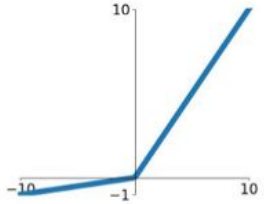
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

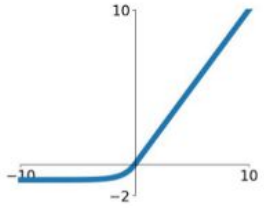


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Autoencoder: Loss Function

Table 1: List of losses analysed in this paper. $\mathbf{y}$ is true label as one-hot encoding, $\hat{\mathbf{y}}$ is true label as +1/-1 encoding, $\mathbf{o}$ is the output of the last layer of the network, $.^{(j)}$ denotes jth dimension of a given vector, and $\sigma(\cdot)$ denotes probability estimate.

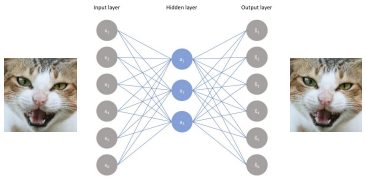
symbol	name	equation
$\mathcal{L}_1$	L_1 loss	$\|\mathbf{y}-\mathbf{o}\|_1$
$\mathcal{L}_2$	L_2 loss	$\|\mathbf{y}-\mathbf{o}\|_2^2$
$\mathcal{L}_1 \circ \sigma$	expectation loss	$\|\mathbf{y}-\sigma(\mathbf{o})\|_1$
$\mathcal{L}_2 \circ \sigma$	regularised expectation loss[1]	$\|\mathbf{y}-\sigma(\mathbf{o})\|_2^2$
$\mathcal{L}_\infty \circ \sigma$	Chebyshev loss	$\max_j \|\sigma(\mathbf{o})^{(j)} - \mathbf{y}^{(j)}\|$
hinge	hinge [13] (margin) loss	$\sum_j \max(0, \frac{1}{2} - \hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})$
hinge^2	squared hinge (margin) loss	$\sum_j \max(0, \frac{1}{2} - \hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})^2$
hinge^3	cubed hinge (margin) loss	$\sum_j \max(0, \frac{1}{2} - \hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})^3$
log	log (cross entropy) loss	$-\sum_j \mathbf{y}^{(j)} \log \sigma(\mathbf{o})^{(j)}$
log^2	squared log loss	$-\sum_j [\mathbf{y}^{(j)} \log \sigma(\mathbf{o})^{(j)}]^2$
tan	Tanimoto loss	$\frac{-\sum_j \sigma(\mathbf{o})^{(j)}\mathbf{y}^{(j)}}{\|\sigma(\mathbf{o})\|_2^2 + \|\mathbf{y}\|_2^2 - \sum_j \sigma(\mathbf{o})^{(j)}\mathbf{y}^{(j)}}$
D_{CS}	Cauchy-Schwarz Divergence [3]	$-\log \frac{\sum_j \sigma(\mathbf{o})^{(j)}\mathbf{y}^{(j)}}{\|\sigma(\mathbf{o})\|_2 \|\mathbf{y}\|_2}$

https://arxiv.org/abs/1702.05659

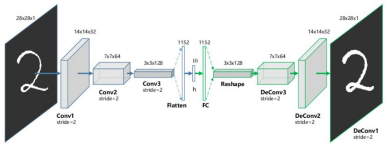
Autoencoder: Taxonomy

Low dimensionality

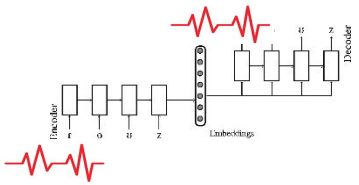
Basic



Convolutional

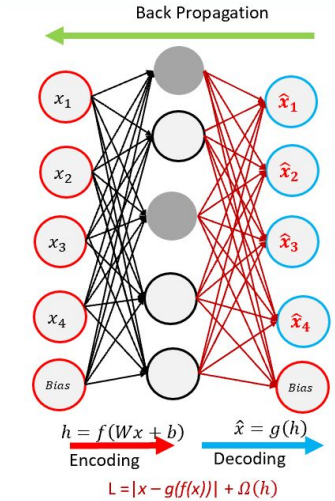


RNN-based



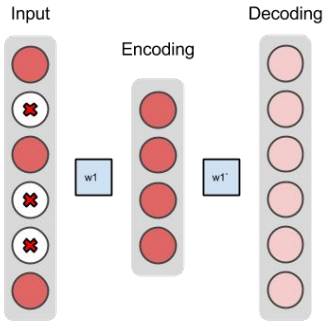
Regularization

Sparse



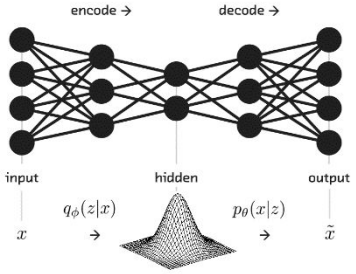
Noise Tolerance

Denoising

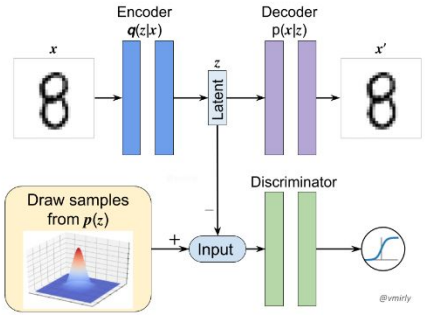


Generative Model

Variational



Adversarial



Example: Outlier detection
(anomaly detection)

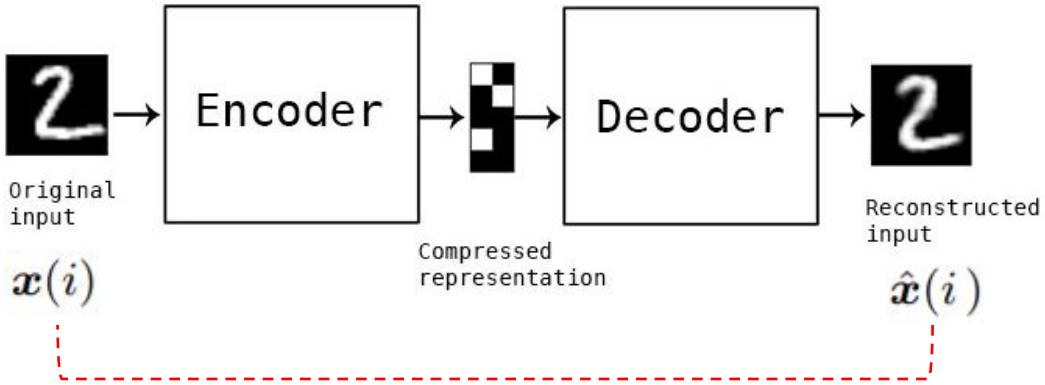
Original Input Data

$$\{\boldsymbol{x}(1), \boldsymbol{x}(2), \ldots, \boldsymbol{x}(m)\}$$

$$\boldsymbol{x}(i) \in \mathbb{R}^D$$ ***(D dimensional sensor data)***

Reconstructed Input Data

$$\{\hat{\boldsymbol{x}}(1), \hat{\boldsymbol{x}}(2), \ldots, \hat{\boldsymbol{x}}(m)\}$$

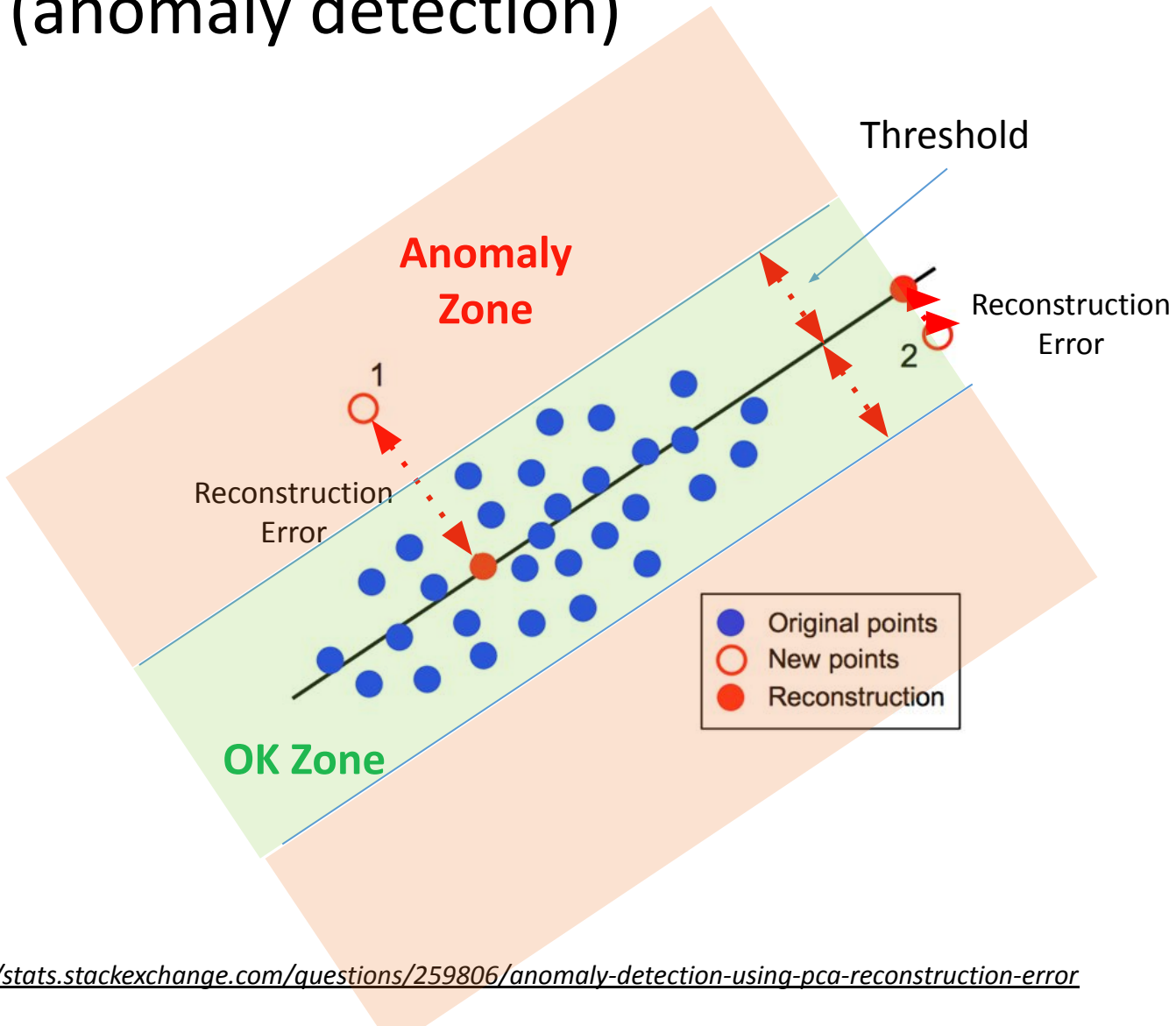


$$Err(i) = \sqrt{\sum_{j=1}^{D} (x_j(i) - \hat{x_j}(i))^2}$$

If (Err(i) > threshold) Anomaly!

https://stats.stackexchange.com/questions/259806/anomaly-detection-using-pca-reconstruction-error

Example: Outlier detection
(anomaly detection)



https://stats.stackexchange.com/questions/259806/anomaly-detection-using-pca-reconstruction-error

Example: Outlier detection
(anomaly detection)

- **Lorenz**: Artificial data simulating the Lorenz system
- **Sat-A & Sat-B**
  - Satellite-A and Satellite-B have D=17 and D=106 dimensional continuous sensor measurements respectively (sensors tended to be correlated)
- Results (average AUC)

	LPCA	AE	dAE	KPCA
Lorenz	0.5104	0.6473	0.7011	0.7045
Sat-A	0.8852	0.8847	0.9354	0.8862
Sat-B	0.9764	0.9763	0.8355	0.7689

LPCA: Linear PCA
AE: AutoEncoder
dAE: Denosing AutoEncoder
KPCA: Kernel PCA

Mayu Sakurada, Takehisa Yairi, Anomaly Detection Using Autoencoders with Nonlinear Dimensionality Reduction, MLSDA'14

Example: Mobility features

- How to summarize mobility data? (a set of GPS points)
- **Displacement Representation (DR)**: Vector of distances between all pairs of adjacent GPS points

Given a set $\mathbf{P}^u = \{p_1^u, p_2^u, \ldots, p_N^u\}$ of N GPS points of a user u. We define the displacement vector for user u as:

$$\mathbf{D}^u = \{d_1^u, d_2^u, \ldots, d_i^u, \ldots, d_{N-1}^u\} \quad (1)$$

with

$$d_i^u = dist(p_i^u, p_{i+1}^u) \quad (2)$$

where *dist()* is a function that computes Haversine distance between two points.

Using Autoencoders to Automatically Extract Mobility Features for Predicting Depressive States, Abhinav Mehrotra, Mirco Musolesi, ACM IMWUT 2018

Example: Mobility features

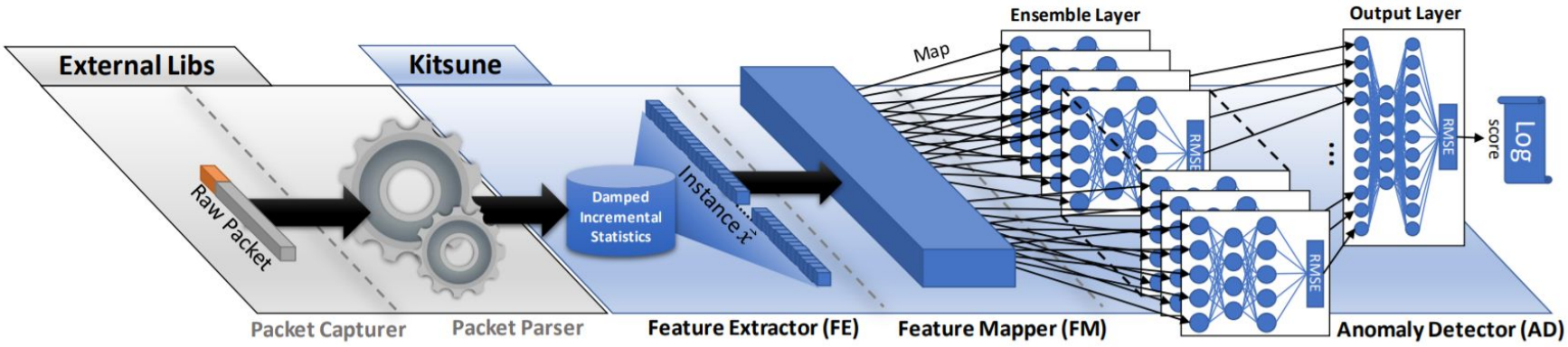
- How to summarize mobility data? (a set of GPS points)
- **Significant Place Representation (SPR)**: the time spent at the top *k* significant places for a given day

More formally, given a set of GPS points $\mathbf{P}^u$, wefi rst cluster them into significant places using the approach presented in [9] for clustering location points. This results in a set $\mathbf{L}^u = \{l_1^u, l_2^u, \ldots, l_i^u, \ldots, l_{M^u}^u\}$ of M^u places [1]. Then based on the overall time a user u spent during the period of data collection at each place in $\mathbf{L}^u$, wefi nd the top k significant places (sorted in a decreasing order of time spent). This results in a set $\mathbf{S}^u = \{s_1^u, s_2^u, \ldots, s_k^u\}$ of k significant places for user u.
We then compute the time spent by the user u at all places in $\mathbf{S}^u$ for a given *day* as follows:

$$\mathbf{T}_{day}^u = \{t_{day,s_1^u}^u, t_{day,s_2^u}^u, \ldots, t_{day,s_k^u}^u\} \quad (6)$$

Using Autoencoders to Automatically Extract Mobility Features for Predicting Depressive States, Abhinav Mehrotra, Mirco Musolesi, ACM IMWUT 2018

Example: **Network Intrusion Detection**



Autoencoder Ensemble

Specifically, the statistics summarize all of the traffic...

- ...originating from this packet's source MAC and IP address (denoted **SrcMAC-IP**).
- ...originating from this packet's source IP (denoted **SrcIP**).
- ...sent between this packet's source and destination IPs (denoted **Channel**).
- ...sent between this packet's source and destination TCP/UDP Socket (denoted **Socket**).

A total of 23 features (capturing the above) can be extracted from a single time window λ (see Table II). The FE extracts the same set of features from a total of five time windows: 100ms, 500ms, 1.5sec, 10sec, and 1min into the past ($\lambda = 5, 3, 1, 0.1, 0.01$), thus totaling 115 features.

Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection
Network and Distributed Systems Security Symposium (NDSS), 2018

Summary

- Outlier
  - ML vs. Distribution-based vs. Distance-based Methods
  - ML: Lack of labels or outlier data makes it challenging (single class classification)
  - Distribution: popular (normality assumption): 3*sigma, Chauvenet's criterion (reject prob = 1/cN) where N is # samples, non-normal dist? (use MAD instead)
  - Distance: local density threshold, local outlier factor
- Imputation
  - Mean/Median/Mode imputation
  - Interpolation-based imputation
  - Model-based imputation
- Kalman Filter (outlier + imputation)
  - Prior knowledge assumption: process & measurement models
- Transformation
  - Loss pass filters (e.g., Butterworth filter) => covered in DSP sections (Week 5)
  - Principal component analysis
  - t-SNE & Autoencoders
- Next: DSP (Week 5) => Feature Extraction (Week 6)