

## Content

1	Organisation .....	3
1.1	PÜ & PL.....	3
1.2	Schedule.....	3
2	Data Preprocessing.....	4
2.1	Handling Errors in Data .....	4
2.2	Resampling Unbalanced Distributions .....	5
2.3	False Predictors.....	5
2.4	Unsupported Data Types .....	5
2.5	High Dimensionality & Feature Subset Selection .....	6
3	Ensemble Methods.....	7
3.1	Voting.....	7
3.2	Bagging.....	8
3.3	Boosting.....	9
3.4	Stacking.....	10
3.5	Regression Ensembles .....	11
3.6	MetaCost Algorithm.....	12
4	Time Series.....	14
4.1	Sequential Pattern Mining.....	14
4.2	Trend Detection.....	15
4.3	Forecasting.....	17
5	Neural Networks and Deep Learning.....	18
5.1	Basics: NN .....	18
5.2	Autoencoders .....	18
5.3	Convolutional Neural Network (CNN) .....	19
6	Anomaly Detection.....	20
6.1	Graphical Approaches.....	20
6.2	Statistical Approaches .....	21
6.3	Distance Based Approaches .....	22

6.4	Challenges .....	24
6.5	Semi-Supervised AD .....	24
7	Hyperparameter Tuning.....	25
7.1	Grid-Search.....	25
7.2	One-After-Another .....	25
7.3	Hill-Climbing Search.....	25
7.4	Local Beam Search.....	25
7.5	Simulated Annealing.....	25
7.6	Random Search.....	26
7.7	Genetic Algorithms.....	27
7.8	Selecting Learners: Meta Learning.....	28
8	Model Validation.....	29
8.1	Z-Test.....	29
8.2	Model Variance .....	30
8.3	Sign Test .....	30
8.4	Wilcoxon Signed-Rank Test.....	30
8.5	Summary .....	31
9	asdf.....	<b>Error! Bookmark not defined.</b>
9.1	Overview .....	<b>Error! Bookmark not defined.</b>

# 1 Organisation

## 1.1 PÜ & PL

Final exam

- 100% of grade
- written exam

Project work

- not graded, but mandatory!
- work on KDD tasks in teams of 6 students

Presentations

- up to three intermediate presentations
- about open questions, problems, current results (numbers!)
- everybody has to present once during those presentations

Final report

- 10 pages
- solutions, results, lessons learned

## 1.2 Schedule

	<b>Lecture (Tuesday)</b>	<b>Exercise (Monday)</b>
21.2.	Introduction & Data Preprocessing	
28.2.	Ensembles	Introduction & Data Preprocessing
7.3.	Time Series	Ensembles
14.3.	Neural Networks & Deep Learning	Time Series
21.3.	Anomaly Reduction	Neural Networks & Deep Learning
28.3.	Hyperparameter Tuning	
4.4.	Easter Break	
11.4.	Easter Break	
18.4.	! KDD Cup Presentation	Hyperparameter Tuning
25.4.	Model Verification and Interpretation	
2.5.	KDD Cup	Model Verification and Interpretation
9.5.	Anomaly Reduction	
16.5.	KDD Cup	Anomaly Reduction
23.5.	KDD Cup	
2.6.	! Submission: Final report and first KDD Cup solutions	
21.6.	! Submission (optional): Final KDD Cup solutions	

## 2 Data Preprocessing

### 2.1 Handling Errors in Data

Data may have errors that make it problematic for the subsequent mining steps; these problems need to be fixed before going further. Sources of erroneous data may be malfunctioning sensors, errors in manual data entry, storage/transmission errors, encoding problems and so on. This can be handled through different methods depending on the type of error:

- Removing data points outside a given interval → requires domain knowledge
- Automatically finding suspicious data points → *Anomaly Detection*

But values can also be missing instead of just being wrong. Treatments for this can be:

- Ignoring records with missing values in training data → this brute force approach does waste some data!
- Replacing missing values with default value, average/median (for numeric data), or most frequent value (for nominals)
- Predicting missing values based on other, complete, training data



The last two treatments are *Imputation Methods*, which are methods for replacing missing data with substituted values.

! Some values may be missing for a good reason. For example, omitted values for alcohol consumption in a questionnaire tend to be higher than average. Values that are only collected under certain conditions (final grade of university degree), values only valid for certain sub-populations (pregnancy y/n) or sensors failing under certain conditions (high temperatures) are also missing for a good reason! In those cases, averaging and imputation causes information loss. In other words: “missing” can be information!

#### Example

Imagine a medical trial checking for side effects of a particular drug. In the trial, there are 50 people who know their blood sugar value. Out of those, 4/5 have an increased blood sugar value.

	side effects	yes (n=58)	no (n=192)
increased blood sugar			
yes (n=40)		30	10
no (n=10)		8	2
-- (n=200)		20	180

Overall, the side effects are moderate (~23%), but people with an increased blood sugar value have a 75% risk of side effects

Assume you handle the missing value for increased blood sugar by filling in the majority value (“yes”). This skews the data distribution!

	side effects	yes (n=58)	no (n=192)
increased blood sugar			
yes (n=240)		50	190
no (n=10)		8	2

Overall, the side effects are moderate (~23%), and even slightly lower (~21%) for people with an increased blood sugar value

## 2.2 Resampling Unbalanced Distributions

Unbalanced distributions of classes may lead to poor performance of the model. For example:

Data set: records of patients who were tested for HIV


Class Distribution: 99.9% negative; 0.01% positive

Learning a decision tree on above data would be without further preprocessing not very effective. This is because it will be hard to find any splitting that significantly improves the purity of the data. Although the model will have very high accuracy (99.9%), recall/precision on positive class would be near 0 as the decision tree would just classify all data as negative.

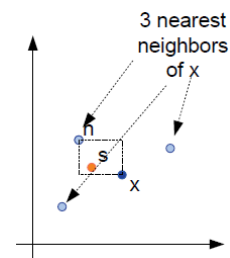
To work around this, rebalancing the data is necessary. Although, evaluation should be done on the unbalanced dataset! Consider an extreme example:

- 1,000 examples of class A
- 10 examples of class B

! Down-sampling would discard 990 examples while up-sampling creates 100 copies of each example of B. Then, it's likely for the classifier to simply memorize the 10 B cases.

 SMOTE (Synthetic Minority Over Sampling Technique) sampling creates synthetic examples of the minority class. It chooses  $n$  among the  $k$  nearest neighbors (within the same class) of  $x$ . Based on each attribute  $a$  it creates a sample  $s$ :

$$s_a \leftarrow x_a + rand(0,1) * (n_a - x_a)$$



## 2.3 False Predictors

~100% accuracy is a great result when learning a model - and a result that should make you suspicious! Including target variables in the attributes while training can lead to this!

Process to recognize false predictors:

- analyzing models (for example: decision trees with only one node are suspicious)
- learn model, inspect model, remove suspect, and repeat until the accuracy drops.
- Analyze attributes: compute correlation of each attribute with label


! There are also strong, but not false predictors; it's not always possible to decide automatically!

## 2.4 Unsupported Data Types

Not every model supports all data types:

Solutions:

- |   |   |
|---|---|
| • some (e.g., ID3) cannot handle numeric data; others (e.g., SVM) cannot nominal data | • converting nominal to numeric data                          |
| • dates are difficult for most learners   | • converting numeric to nominal data (discretization/binning) |
|   | • extract valuable info from dates                            |

 Discretization by entropy, for example, tries to make intra-bin similarity as high as possible by minimizing the entropy in each bin.

! Training and test data have to be equally discretized! Naively applying discretization will lead to different ranges. Correctly discretizing is done by *fitting* only on the training set and transforming the test set using the already fitted discretize. Fitting on both will cause information-/data leakage and may lead to overfitting!

## 2.5 High Dimensionality & Feature Subset Selection

! Datasets with large number of attributes cause (not only a) scalability problem. For example, when training a decision tree: searching all attributes for determining one single split is difficult!

- Learning models gets more complicated in high-dimensional spaces as a higher number of observations are needed for covering a meaningful number of combinations → Combinatorial Explosion
- Distance functions collapse: all distances converge to a constant in high dimensions → Nearest neighbor classifiers are no longer meaningful

To deal with this, only valuable features should be used!

📄 Some basic heuristics are:

- removing nominal attributes which have many identical values or different values
- removing numerical attributes which have little variation, i.e., standard deviation

📄 Filter methods use an attribute weighting criterion ( $\chi^2$ , Information Gain, ...) and select attributes with highest weights → fast but not always optimal.

- For example, temperature in °C and °F or textual features such as first and last name

This is done by computing pairwise correlations between attributes and then removing highly correlated attributes. Note that Naive Bayes requires independent attributes and/or will benefit from removing correlated attributes.

📄 *Forward Selection* starts with an empty feature set and then computes the model performance continuously while adding attributes to the feature set until the performance no longer increases → cross validation is advised!

📄 *Backwards elimination* starts with the full feature set and then computes the model performance continuously while removing attributes from the feature set until the performance decreases → cross validation is advised!

Further approaches are Brute Force search and Evolutionary algorithms.

💡 As a summary: simple heuristics are fast but may not be the most effective; brute-force is most effective but the slowest; forward selection, backward elimination, and evolutionary algorithms are often a good compromise.

Feature Subset Selection reduces the width of the dataset - Sampling reduces the height of the dataset.

### 3 Ensemble Methods

💡 Idea: Instead of asking a single learner, combine the predictions of different learners.

📖 Prerequisites for ensembles:

- *Accuracy*: different models can address a problem
- *Diversity*: different models make different mistakes

That means predictions on a new example may differ between the learners. Then, combine their results into a single prediction.

#### 3.1 Voting

This is the most straightforward approach:

- For classification: use the most-predicted label
- For regression: use the average of prediction

This is similar to KNN, where each neighbor can be regarded as an individual classifier.

##### Example

Method	Naïve Bayes	Ripper	KNN	Voting (all 3 combined)
Accuracy-Score	0.71	0.71	0.81	<b>0.91</b>

##### Why does it work?

Suppose there are 25 base classifiers; where each classifier has an accuracy of 0.65, i.e., an error rate  $\epsilon = 0.35$ . Further assume the classifiers and the errors they make are independent (in practice they are not!).

The ensemble classifier makes a wrong prediction if the majority of the classifiers makes a wrong prediction. The probability that, in this case, 13 or more classifiers are wrong is:

$$\sum_{i=13}^{25} \binom{25}{i} * \epsilon^i (1 - \epsilon)^{25-i} \approx 0.06$$

In theory, we can lower the error infinitely by adding more base learners.

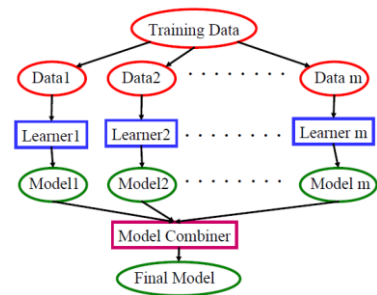
❗ But the formula only holds for independent base learners! It is hard to find many truly independent base learners at a decent level of accuracy.

## 3.2 Bagging

Biases in data samples may mislead classifiers (for example making it overfit). If we had different data sets/samples and trained a model on each of those data sets, only one model would overfit to each noise point. Voting then could address these issues.



Models may differ when learned on different data samples. The idea of bagging is to create new artificial data sets / diverse samples by *sampling with replacement*, learning a model on each sample, and then combining the models.



### Example

Original Data	1	2	3	4	5	6	7	8	9	10
Bagging (Round 1)	7	8	10	8	2	5	10	10	5	9
Bagging (Round 2)	1	4	9	1	2	3	2	7	3	2
Bagging (Round 3)	1	8	5	10	5	5	9	6	3	7

Some examples may appear in more than one set, some examples will appear more than once in a set.

### 3.2.1 Variant: Randomization

Randomize the learning algorithm instead of the input data. Most algorithms can be randomized (weights in neural net; greedy algorithms: pick from  $n$  best options instead of best). Can also be combined with classical bagging.

### 3.2.2 Variant: Random Forests

Random Forests is a variation of bagging. A number of individual decision trees is trained, each on a random subset of examples and also only a random subset of attributes is analyzed for each split. Usually, the individual trees are left unpruned.

### 3.2.3 Variant: Weighted Voting

Some learners provide confidence values (decision trees, naïve bayes) that can be used for weighting the votes. Some models may be rather sure about an example, while others may be indifferent.



### 3.3 Boosting

The idea of *Boosting* is to sequentially train a set of classifiers, one after another, while later classifiers focus on examples that were misclassified by earlier models. Finally, we weight the predictions of the classifiers with their error.

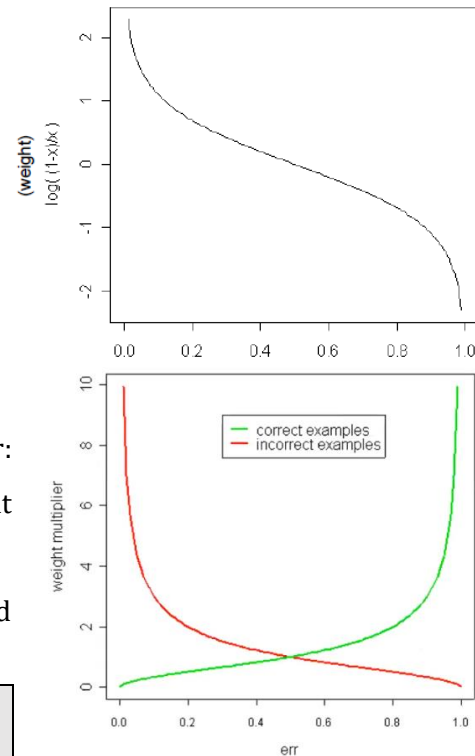
Between training iterations, we increase the weight of incorrectly classified examples so they become more important in the next iterations (misclassification error for these examples count more heavily).

Classifier weights  $\alpha_m$  emphasize differences near 0 or 1. A good classifier gets a highly positive weight, a very bad classifier gets a highly negative weight (inverse of prediction is good again) and classifiers with error near 0.5 (for binary prediction) gets almost no weight.

Samples get weights depending on error rate of classifier:

- Incorrectly identified examples by a classifier that is very good (top left) get high weights
- Correctly classified examples by a very bad classifier (top right) get high weights

Classifier Example	Good	Bad
Correct	low weight	high weight
Incorrect	high weight	low weight



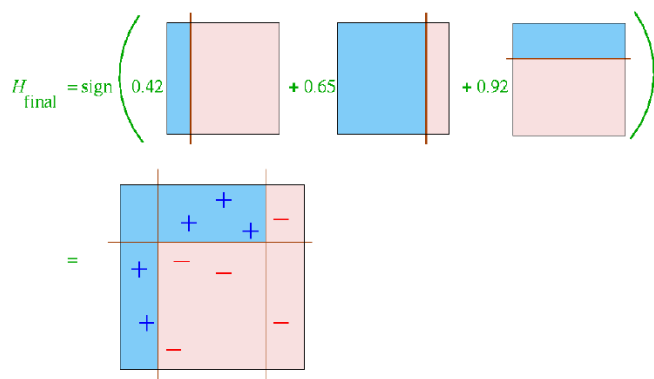
X-Axis: error rate of classifier in a particular iteration

Boosting with 400 splits performs way better than a single decision tree with 400 splits. The reason lies in the complexity of the model space.

#### Hypothesis Space

Each learner has a hypothesis space. It's the total set of possible models that a model can learn. For example, a decision stump (single node decision tree) has a hypothesis space of all possible linear separations of a data set that are parallel to the axis (compare to figure).

The hypothesis space of an ensemble can be larger than that of its base learners.



#### Final Notes

Ensembles have been used to improve generalization accuracy on a wide variety of problems. On average, Boosting provides a larger increase in accuracy than *Bagging*.

- Boosting on rare occasions can degrade accuracy
- Bagging more consistently provides a modest improvement

! Boosting is particularly subject to over-fitting when there is significant noise in the training data subsequent learners over-focus on noise points.

### 3.4 Stacking

Bagging, Boosting, and voting all have the same underlying principle of combining predictions. They either use the majority vote or a weighted average of the predictions.

! *Stacking* will use a classifier to make the final decision. Training material are the class labels of the training data (target) and the (cross-validated) predictions of the ensemble members.

Attributes				Class	$C_1$	$C_2$	...	$C_{n_c}$	$C_1$	$C_2$	...	$C_{n_c}$	Class
$x_{11}$	...	$x_{1n_a}$		$t$	$t$	$t$	...	$f$	$t$	$t$	...	$f$	$t$
$x_{21}$	...	$x_{2n_a}$		$f$	$f$	$t$	...	$t$	$f$	$t$	...	$t$	$f$
...	...	...		...	...	...	...	...	...	...	...	...	...
$x_{n_e1}$	...	$x_{n_en_a}$		$t$	$f$	$f$	...	$t$	$f$	$f$	...	$t$	$t$

training set
predictions of the classifiers
training set for stacking

! Stacking tends to overfit to a base classifier that performs very well on the training set. In order to prevent this one can:

- Split the data set: use one portion for base classifiers and another for the meta model
- Cross-validate base classifiers:
  - Train each base classifier on 90% of training data
  - Make predictions for the remaining 10% (→ features for meta learner)
  - Repeat 10 times

#### Example

Method	Naïve Bayes	Ripper	KNN	Stacking
Accuracy-Score	0.71	0.71	0.81	<b>0.86</b>

#### 3.4.1 Variant: Keep Original Attributes

Here the meta classifier gets both, the original attributes (previously used for training only the base learners) and the base learners predictions as features. This allows the identification of blind spots of individual base learners.

#### 3.4.2 Variant: Confidence Values

If learners provide confidence values, those can be used by the meta learner. This often further improves the results.

## 3.5 Regression Ensembles

Most ensemble methods also work for regression with slight adaptations to their algorithms:

- Voting → use average
- Bagging → use average or weighted average
- Stacking → learn regression model as meta learner
- Boosting → *additive regression*

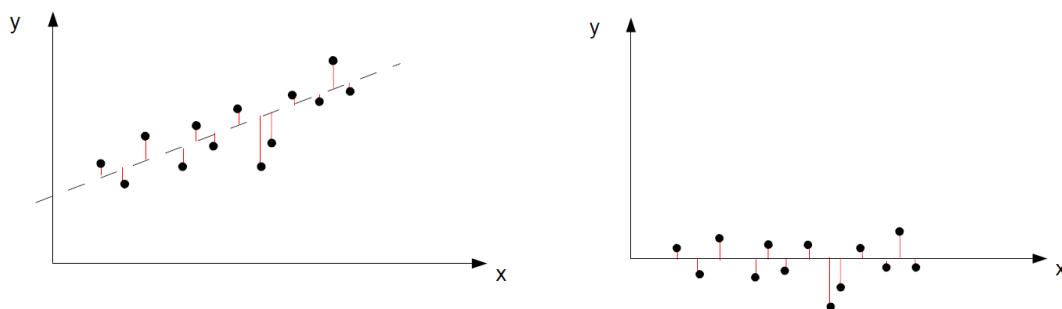
### Additive Regression

Boosting can be seen as a greedy algorithm for fitting additive models. The same kind of algorithm holds for numeric prediction:

- Build standard regression model
- Gather residuals (the difference between the observed value and the estimated value of the quantity of interest, for example: sample mean)
- Learn model that predicts the residuals
- repeat

To predict, simply sum up the weighted individual predictions from all models.

! Additive and Linear Regression are not a good match:




First model: left; Predicting error terms (residuals): right

### XGBoost

💡 Currently wins most Kaggle competitions. Is actually additive regression using regression trees. It tries to overcome the overfitting problem by introducing a penalty term for the size of the tree. This favors smaller trees over larger ones.

### 3.6 MetaCost Algorithm

Most classifiers aim at reducing the number of errors, where all errors are regarded as being equally important. In reality, misclassification costs differ (e.g., airplane warning system; bank fraud detection)!


 The MetaCost Algorithm uses a lot of ideas from ensemble learning and instance weighting. It creates classifications on the training set (→ Bagging) multiple times. It learns a number of base classifiers by pulling different samples out of the data.

It then estimates each class's probability (80% class A; 20% class B) by the fraction of votes that it receives from the ensemble. With that, a *conditional risk* can be assigned to each sample. This risk addresses the probability of being misclassified and the cost of doing so.

The conditional risk  $R(i|x)$  is the expected cost of predicting that sample  $x$  belongs to class  $i$ :

$$R(i|x) = \sum_j P(j|x) * C(i, j)$$

- $C(i, j)$  is the misclassification cost of classifying an example of class  $j$  as class  $i$
- $P(j|x)$  is the probability of above misclassification happening

 The goal is to *relabel the training examples* with their “optimal” classes (label with the lowest risk). This will make a defensive, low-risk-predicting classifier that minimizes costs.

### 3.7 Exercises

#### Y/N Question

- With the Voting mechanism, predictions of multiple classifiers or regressors are combined (using the mode or the average) to find a final prediction. Correct?

#### Answer / Solution

Yes.

#### Y/N Question

- Bagging is like voting, but instead of using differing classifiers or regressors, we split the dataset into  $n$  parts and vote over the results from the classifiers/regressors applied to the individual datasets. Correct?

#### Answer / Solution

Not completely. We do not simply split the dataset, but we use sampling with replacement to create multiple datasets.

#### Y/N Question

- Boosting is like voting, but instead of doing a majority vote (mode/average), we use a classifier to make the final decision. Correct?

#### Answer / Solution

No, this is what stacking does. The idea of Boosting is to train a set of classifiers where later classifiers are forced to focus more on the misclassifications of previous classifiers by assigning these examples a higher weight.

**Y/N Question**

- The Random Forest algorithm is an application of the Stacking mechanism. Correct?

**Answer / Solution**

No, it is an application of the Bagging mechanism.

**Y/N Question**

- The MetaCost algorithm is an ensemble mechanism that tries to relabel the training examples in such a way that the "risk" of the predictions is minimized. Correct?

**Answer / Solution**


Yes.

## 4 Time Series

Many classical data mining problems have variants that respect the time dimension:

- Frequent pattern mining → sequential pattern mining
- Classification → predicting sequences of nominals
- Regression → predicting the continuation of a numeric series

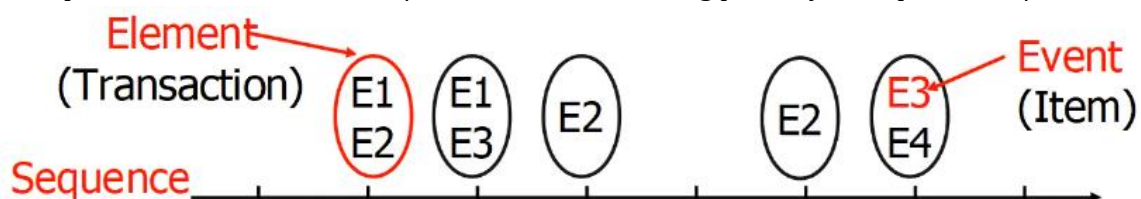
### 4.1 Sequential Pattern Mining

 Sequential pattern mining is about finding frequent subsequences in a set of sequences.

Applications for this are in Web Usage Mining, for example, where website owners try to improve the users navigation path (less clicks to reach goal). Recommendation systems can be improved by the time dimension (mobile phone → charger; and not the other way around).

#### 4.1.1 Sequences, Elements, and Items


A sequence consists of elements/transactions containing possibly multiple events/items.




 Formally, a sequence is an ordered list of elements (transactions):

$$s = \langle e_1 e_2 \dots e_n \rangle$$

The length of a sequence  $|s|$  is given by the number of elements in the sequence.


 Each element is attributed to a specific contains a collection of events (items):

$$e_i = \{i_1, i_2, \dots, i_k\}$$


 A  $k$ -sequence is a sequence that contains  $k$  events (items). This can be across elements:

valid 2-sequences	$\langle \{1\} \{2\} \rangle, \langle \{3, 4\} \rangle$
valid 3-sequences	$\langle \{1\} \{2, 3\} \rangle, \langle \{4, 5, 6\} \rangle, \langle \{7\} \{8\} \{9\} \rangle$

#### 4.1.2 Subsequences

 A sequence  $\langle a_1 a_2 \dots a_n \rangle$  is *contained* in another sequence  $\langle b_1 b_2 \dots b_m \rangle$  with  $m \geq n$  if there exists integers  $i_1 < i_2 < \dots < i_n$  such that  $a_1 \subseteq b_{i_1}, a_2 \subseteq b_{i_2}, \dots, a_n \subseteq b_{i_n}$ .

Data sequence b	Subsequence a	Contain?
$\langle \{2,4\} \{3,5,6\}, \{8\} \rangle$	$\langle \{2\} \{3, 5\} \rangle$	Yes
$\langle \{1,2\} \{3,4\} \rangle$	$\langle \{1\} \{2\} \rangle$	No
$\langle \{2,4\} \{2,4\}, \{2,5\} \rangle$	$\langle \{2\} \{4\} \rangle$	Yes

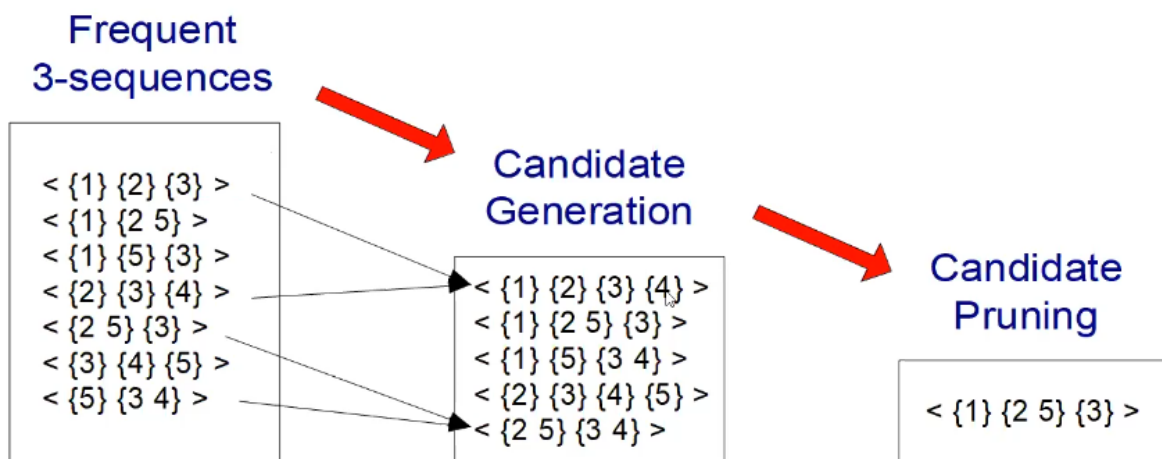
 The *support* of a subsequence  $w$  is defined as the fraction of data sequences that contain  $w$ . A *sequential pattern* is a frequent subsequence (i.e., a subsequence whose support  $\geq \text{minsup}$ ).

### 4.1.3 Generalized Sequential Pattern Algorithm (GSP)

💡 Given a database of sequences and a user-specified minimum support threshold,  $\text{minsup}$ , find all subsequences with support  $\geq \text{minsup}$ .

1. Make the first pass over the sequence database  $D$  to yield all the 1-element frequent subsequences
2. Repeat:
  - a. Candidate Generation: merge pairs of frequent subsequences found in the previous pass to generate candidate sequences that contain 1 item more than last time
  - b. Candidate Pruning: Prune candidate  $k$ -sequences that contain infrequent  $(k - 1)$ -subsequences
  - c. Support Counting: make a new pass over  $D$  to find the support for these new candidate sequences

#### Example



## 4.2 Trend Detection


💡 Given a time series, find out what the general trend is (rising, falling). Possible obstacles are random, seasonal and cyclical effects that affect the analysis.

Random Effects	No real pattern or reason for why this effect occurs.
Seasonal Effects	Occur regularly each year. E.g., ice cream sales in summer
Cyclical Effects	Recurring, cyclical effects of different period lengths.

Trend curve estimation can be done by various techniques, such as:


- Least-Squares Method: Minimizes the sum of squared errors.
- Moving-Average Method: Averages last two consecutive points for new prediction.

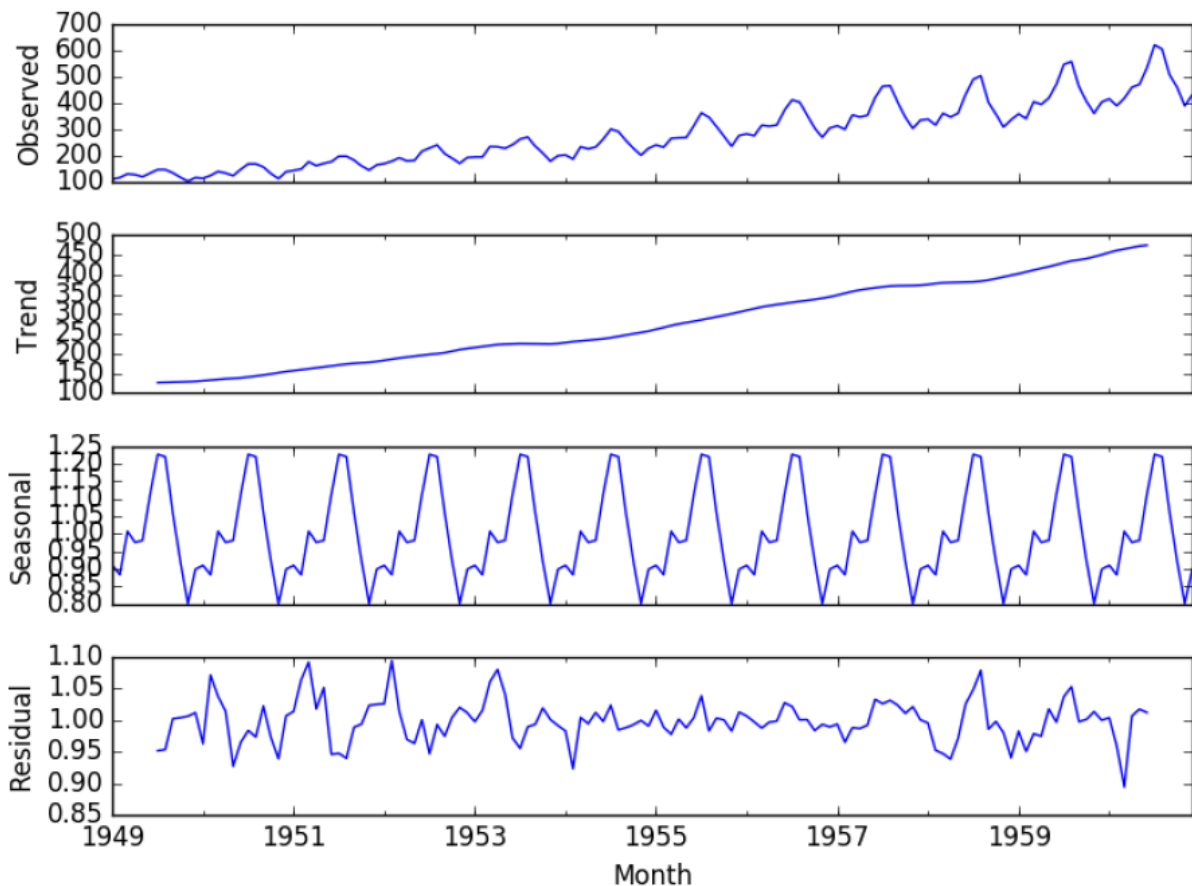
## 4.2.1 Component Model of Time Series

 A time series can consist of four components:

- Long-term trend  $T_t$
- Cyclical effect  $C_t$
- Seasonal effect  $S_t$
- Random variation  $R_t$

These components can either be added together in an *additive model* or multiplied with each other in a *multiplicative model*. Some methods work more smoothly for a certain model.

 To find out more about a certain component the other ones have to be filtered out.



If we know the periodicity of seasonal or cyclical effects, we can isolate these effects from one another. To measure the seasonal effects, we compute seasonal indexes. These indices are computed by the overall deviation for each season. This can be done by:


1. Computing the trend  $\hat{y}$  (i.e., linear regression)
2. For each time period, computing the ratio  $\frac{y_t}{\hat{y}_t}$  (assuming multiplicative model)
3. For each season (or other relevant period), computing the average of  $\frac{y_t}{\hat{y}_t}$

This gives us the average deviation for that season:


$$\frac{y_t}{\hat{y}_t} = \frac{T_t * S_t * R_t}{T_t} = S_t * R_t$$




## 4.2.2 Fourier Transformation

 Views time series as a sum of sine waves with different periodicity and different amplitudes. The frequencies of those sine waves are called spectrum. Fourier transformation allows to transform between spectrum and series.

## 4.2.3 Smoothing

 The key idea of *moving average* is to calculate the upcoming value as the average of the last  $n$  values. This smoothes the data and eliminates random movements. But it will lose data at the beginning or end of a series due to the calculation.


 Often, moving averages are also used for the trend instead of a linear function.


 *Exponential smoothing* views new data points  $S_t$  as an average of the new observation  $y_t$  and the previously smoothed value  $S_{t-1}$ :

$$S_t = \alpha y_t + (1 - \alpha)S_{t-1}$$

$\alpha$  determines the importance of the new observation with regards to the previously smoothed value. This takes all the history of previous values into account, discounted by  $\alpha$ .

## 4.3 Forecasting

 *Moving averages* can also be used for prediction by using the average of the last  $n$  values to predict the next one:  $y_t = \frac{1}{n}(y_{t-1} + \dots + y_{t-n})$

 A more sophisticated version, *Autoregressive models*, learns weights from the data to change the impact of certain values:  $y_t = \delta_1 y_{t-1} + \delta_2 y_{t-2} + \dots + \delta_n y_{t-n} + \beta + \epsilon_t$

The same holds for exponential smoothing; this works well for short windows but at some point, the predictions will diverge.

 *Double exponential smoothing* adds a trend component for its predictions:

$$S_t = \alpha y_t + (1 - \alpha)(S_{t-1} + b_{t-1})$$

$$b_t = \beta(S_t - S_{t-1}) + (1 - \beta)b_{t-1}$$

Where  $S_t - S_{t-1}$  describes the change of the estimate and  $b$  is the exponentially smoothed times series of those changes.  $S$  is called level smoothing,  $b$  is called trend smoothing.

 *Triple exponential smoothing (Holt Winters Method)* introduces a seasonal component:

$$S_t = \alpha(y_t - c_{t-L}) + (1 - \alpha)(S_{t-1} + b_{t-1})$$

$$b_t = \beta(S_t - S_{t-1}) + (1 - \beta)b_{t-1}$$

$$c_t = \gamma(y_t - S_t) + (1 - \gamma)c_{t-L}$$

$L$  is the cycle length of the seasonality.

## 4.4 Examples / Questions

### Example Task

If using simple exponential smoothing, will the peak demand for a Saturday likely be overpredicted or underpredicted? Why?

### Answer / Solution

Simple exponential smoothing puts much weight on very recent observations, so it would likely overpredict the demand for the Saturday.

### Example Task

Would the Holt-Winters method solve the problem described earlier? If yes, which cycle length should be chosen?

### Answer / Solution

As it is a cyclic time series, Holt-Winters method is a perfect fit. It explicitly introduces a term for cyclic predictions. A cycle length of 7 would make sense.

## 5 Neural Networks and Deep Learning

### 5.1 Basics: NN

[...]

### 5.2 Autoencoders

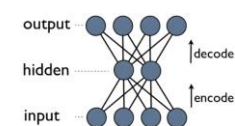
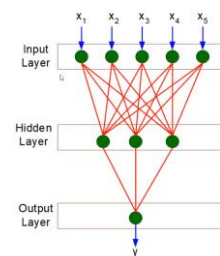
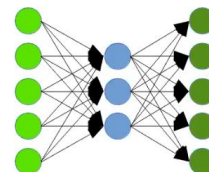
💡 We generally assume, that data can be described with fewer variables without losing much information ( $\rightarrow$  *dimensionality reduction*).

In neural nets, the hidden layer is usually smaller than the input layer in terms of nodes. It compresses the input. Because the output can be predicted from the values at the hidden layer, less features should be sufficient to predict the output.

📄 Autoencoders apply above idea. By using the same values for input and target, they train a model for predicting an example of itself.

Additionally, by adding random noise to the input, the model can learn to remove noise from an instance.


By adding more hidden layers after finishing to train the previous one, we get *Stacked Autoencoders*. They can capture more complex hidden variables and/or denoising patterns, i.e., are more powerful.




📄 If we assume that the hidden layer captures a compressed version of the majority of the data, we then can use the *deviation between output and input* as a measure for outlier score.

## 5.3 Convolutional Neural Network (CNN)

CNNs are comprised of two special components: the convolutional layers and pooling layers.

 **Convolutional layers** are responsible for extracting features from input data, typically images, by applying convolution operations. A convolution involves sliding a small filter (also called a kernel) over the input data and computing element-wise multiplications followed by a sum. The resulting sum is placed in the corresponding position of an output feature map. The filters have learnable weights that are adjusted through the training process. These weights determine the pattern or feature that the filter is designed to detect in the input data.


 **Pooling layers** are used to reduce the spatial dimensions of the feature maps while retaining important information. The most common type of pooling is *max pooling*, where a window (usually 2x2 or 3x3) slides over the input feature map, and the maximum value within the window is selected as the representative value for that region. Pooling layers do not have any learnable weights. The pooling operation, whether it's max pooling or average pooling, simply involves selecting the maximum (or average) value from a local region of the input feature map.

1	0	2	3
4	6	6	8
3	1	1	0
1	2	2	4

→

6	8
3	4

## 5.4 Word Embeddings

 *Word2Vec* is a framework for learning word vectors. Instead of counting how often each word  $w$  occurs near a particular word  $x$  a classifier is trained on a binary prediction task. Is  $w$  likely to show up near  $x$ ? The learned classifier weights will be used as the word embeddings.

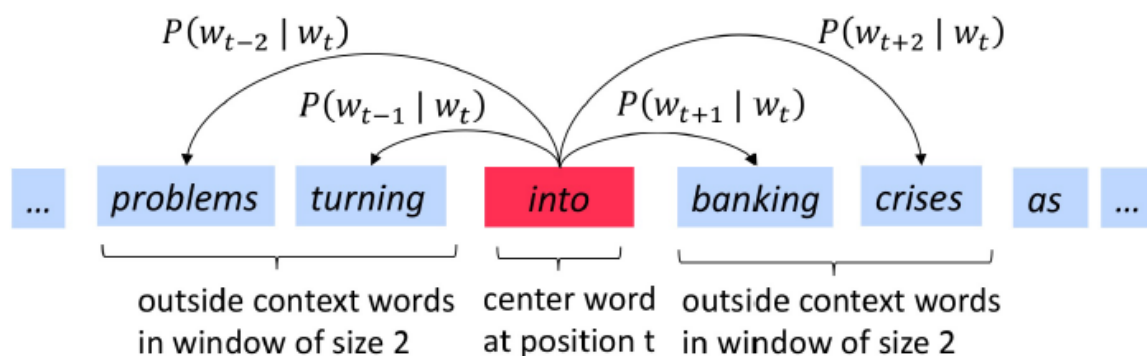
Given a large corpus of text, every word in a fixed vocabulary is represented by a vector. Use the similarity of the word vectors for  $w_t$  (center word) and  $w_{t \pm j}$  (context of size  $t$ ) to calculate the probability of  $w_{t \pm j}$  given  $w_t$ .

**CBOW (Continuous Bag of Words)**

- Predict the **center word** from the **context words**

**Skip-gram**

- Predict the **context words** from the **center word**



### Example

You have a corpus of 1,000 product reviews that you shall use to train a sentiment predictor. As the number of unique words in these product reviews is 150 and hence rather small, you are unsure whether to use One-Hot-Encoding or Word2Vec to encode the words in your product reviews.

1. Name some advantages for both of the approaches.


2. Which one would you prefer?
3. Would it change your opinion if you knew that your sentiment predictor will never encounter words that are not in the training corpus?

### Answer / Solution

One-Hot Encoding	Word2Vec
(+) Simple encoding scheme where it is clear what is meant by a given vector	(-) Word is represented by a vector of latent features without clear meaning
(+) No training data necessary to encode words; any word or token can be encoded	(-) Needs a lot of training data or a pre-trained model has to be used. How to deal with unknown words?
(-) Vector size is equivalent to the size of the vocabulary	(+) Vector size is fixed (typically set to something between 100 and 1000)
(-) No semantic relation between words	(+) Similar words are in similar "regions"

Especially because of the last point, Word2Vec is expected to be the better solution for this problem. But if the vocabulary size is fixed, we might not need the semantic similarity between vectors as this similarity could be encoded in the model (if the training data is expressive enough).

## 6 Anomaly Detection

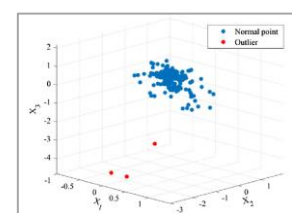
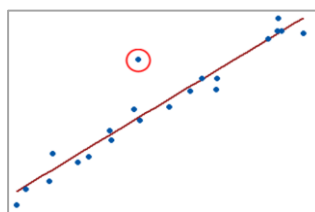
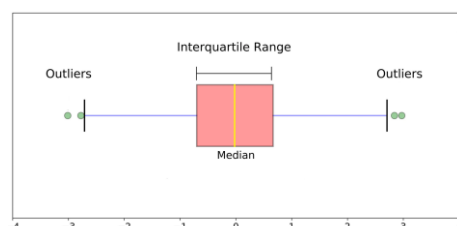
 *Anomaly (or Outlier) Detection* Is about automatically identifying data points that are somehow different from the rest. For this we assume that there are considerably more normal observations than outliers in the data. Anomalies can take different forms:

- Wrong data values (errors)
- Unusual observations (outliers or anomalies)
- Observations not in line with previous observations (novelties)

These can be identified in either a supervised or an unsupervised manner. The more common, unsupervised approach involves utilizing a dataset containing both regular and potentially exceptional data points. In this scenario, the objective is to calculate an outlier score for each individual data point to pinpoint these exceptional instances. On the other hand, in a supervised context, a clean dataset is employed to train a model that can recognize outliers within a separate test dataset.

### 6.1 Graphical Approaches

Boxplot (1D), Scatter plot (2D), Spin plot (3D).



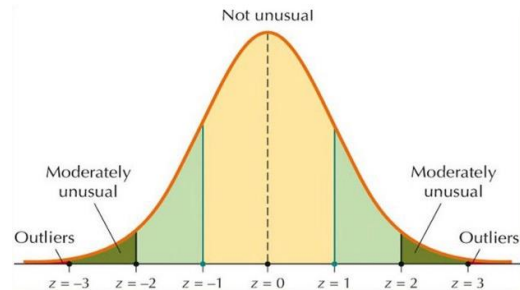
Benefits are the use case of getting a first look and the ability to visualize the results of other (often black box) detection models.

Limitations of these approaches are their time consuming and subjective nature. Detecting what an outlier is in these methods is often subject to interpretation,

## 6.2 Statistical Approaches

A potential approach is to assume a parametric model describing the distribution of the data (e.g., normal distribution).

Another is to apply a statistical test that depends on data distribution, parameter of the distribution (e.g., mean, variance) and a number of expected outliers (confidence limit).



### 6.2.1 Median Absolute Deviation (MAD)

MAD can be used for outlier detection. All values that are  $k * MAD$  away from the median are considered to be outliers:

$$MAD = \text{median}_i (X_i - \text{median}_j (X_j))$$

Asks how much, on average, do the observations deviate from the median?

#### Example

Given the data: [0, 1, 1, 3, 5, 7, 42].

Calculate the median (= 3).

Calculate the deviations from this median: [3, 2, 2, 0, 2, 4, 39].

Calculate MAD from deviations (= 2).

Choose  $k$  (= 3).

Calculate allowed interval:

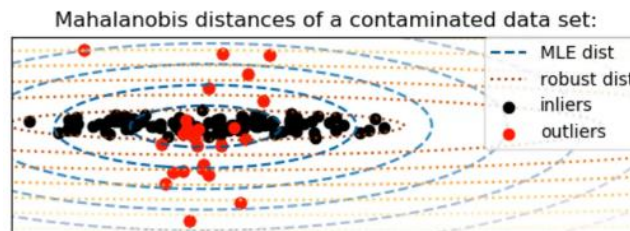
$$[\text{median} - (k * MAD); \text{median} + (k * MAD)]$$

$$[3 - (3 * 2); 3 + (3 * 2)] = [-3; 9]$$

Therefore, 42 is an outlier.

## 6.2.2 Elliptic Curves

For multi-dimensional datasets, above ways of calculating safety margins can be done as well.



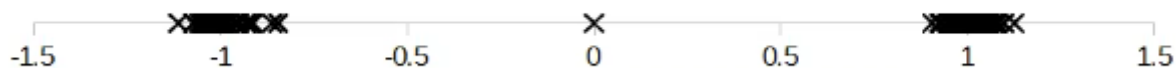
By fitting elliptic curves, we get these safety margins across multiple dimensions.

## 6.2.3 Limitations

- Most of these tests are for a single attribute (*univariate*).
- For high dimensional data, it may be difficult to estimate the true distribution.
- In many cases the distribution also may not be known.

## 6.3 Distance Based Approaches

So far, we have only looked at extreme values only; but outliers can occur at non-extremes as well. In that case, methods like IQR (Boxplot) fail.



### 6.3.1 Nearest-Neighbor Based

- Data points, for which there are fewer than  $p$  neighboring points within a distance  $d$ .
- Top  $n$  data points whose distance to the  $k^{th}$  nearest neighbor is greatest
- Top  $n$  data points whose average distance to the  $k^{th}$  nearest neighbors are greatest

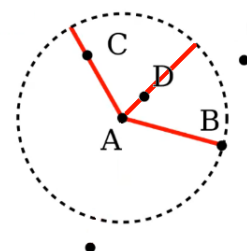
### 6.3.2 Density Based

#### Local Outlier Factor (LOF)

For each point, compute the density of its local neighborhood (defined by KNN for example). If its higher than the average density, the point is cluster. If its lower, then its an outlier. The LOF of a point is the ration of the average density of a points neighbors to the density of the point itself. Outliers then are points with large LOF values.

LOF uses a concept called *reachability distance*. This means that all points within the  $k$ -neighborhood have the same distance. For example:  $rd_3(A, B) = rd_3(A, C) = rd_3(A, D)$ .

For anything outside of this neighborhood, the actual distance is used.



Note that this makes the distance asymmetric, as  $rd_3(A, B) \neq rd_3(B, A)$  may happen, if A is outside of B's neighborhood.

The local density of a points neighborhood, or average reachability distance, is the average distance from a points neighbors back to it. So, in this example:

$$avgrd_3(A) = \frac{1}{3} * (rd_3(B, A) + rd_3(C, A) + rd_3(D, A))$$

Density is defined as the inverse; the larger the avg. reachability distance, the parser the region in which the data point lies. Local Reachability Density:

$$lrb_k(A) = \frac{1}{avgrd_k(A)}$$

The LOF of a point then is the relation of density of the neighbors ( $P$ ) to  $A$ :


$$LOF_k(A) = \frac{\sum_P \frac{lrb_k(P)}{lrb_k(A)}}{|N_k(A)|} = \frac{\sum_P lrb_k(P)}{|N_k(A)| * lrb_k(A)}$$

With  $|N_k(A)|$  being the amount of neighbors A has, as there might be more than  $k$  if there is a tie in distances.


Also, possible approach: **DBSCAN** (Center Points, Border Points, Outliers)

### 6.3.3 Clustering Based

- Cluster the data into groups of different density using algorithm of choice
- Choose points in small clusters as candidate outliers
- Compute the distance between candidate points and non-candidate clusters
- If candidate points are far from all other non-candidate points, then they are outliers

 *Clustering Based LOF* assigns a score based on the size of the cluster a data point is in and the distance of the data point to the next large cluster.


### 6.3.4 Model Based

 Idea: there is a model underlying the data; points deviating from the model are outliers.

- Learn a model for each attribute given all other attributes
- Use this model to predict an expected value
- Calculate the deviation between actual and predicted value to identify outliers

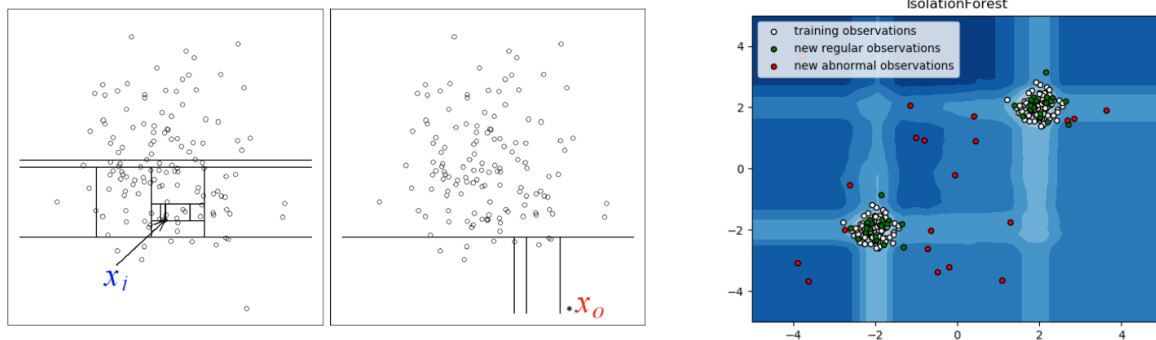
When calculating the deviation (via Euclidian distance (RMSE) for example) we can also assign weights to attributes given the strength of the pattern learned (RRSE).

### 6.3.5 Isolation Forests

 Isolation Forests are random forests of isolation trees. Isolation trees are decision trees that only have leaves with one example each. Not a model-based approach!



The idea being, that outliers will end up higher in the isolation tree. Across a set of random trees, the average path length then can be used as an outlier score.



*Less splits are necessary to isolate  $x_o$  than for  $x_i$ .*

Data points in dense areas need more tests: i.e., they end up deeper in the trees. Data points far away from the rest have a higher probability to be isolated earlier.

## 6.4 Challenges

! A large number of attributes may cause problems. As many detection approaches use distance measures, meaningless attributes obscure the distances. In practice, one could perform dimensionality reduction via PCA or feature subset selection first.

! Attributes also may have different scales. Hence, different attributes may have different contributions to outlier scores. Therefore, values should be normalized! Z-Transformation.

📄 Evaluation of Outlier Detection is usually done on a labeled data subset, where known outliers are labeled as such. Typical Measures are F1-Score or Area under ROC curve.

## 6.5 Semi-Supervised AD

Approaches discussed so far were unsupervised. In semi-supervised AD, experts manually label some data points as being outliers or not.


Using these examples to train a classifier for detecting outliers does not work:

- Outliers are not a class
- Outliers are scarce → unbalanced dataset



## 7 Hyperparameter Tuning

### 7.1 Grid-Search

 Performs a brute force search that is guaranteed to find the best parameter setting but in most practical cases too slow.

For example, given  $n$  binary parameters, this would take  $2^n$  evaluations. For analog/continuous parameters this is even higher!

### 7.2 One-After-Another

! Given  $n$  parameters with  $m$  degrees of freedom, *grid search* would take  $m^n$  evaluations.

By employing the following procedure, this can be reduced to  $n^m$ :

```
1 for i in parameters:
2     try all options for parameters[i]
3     fix best performing setting for parameters[i]
```

Does not guarantee finding the optimal solution but will probably find one near-optimal.

Note that interaction effects make parameter tuning hard, i.e., changing one parameter may change the optimal settings for another one.

### 7.3 Hill-Climbing Search

Also known as *greedy local search*. Will always search in the direction of the steepest ascend.

Basic hill climbing is the starting point for more sophisticated algorithms such as:

	p=0	p=1	p=2
q=0	0.5	0.4	0.1
q=1	0.4	0.3	0.2
q=2	0.1	0.2	0.7

- *Stochastic hill climbing*: random selection among the uphill moves & the selection probability can vary with the steepness of the uphill move.
- *First-choice hill climbing*: generating successors randomly until a better one is found, then pick that one.
- *Random-restart hill climbing*: run hill climbing with different starting seeds; tries to avoid getting stuck in a local optimum

### 7.4 Local Beam Search

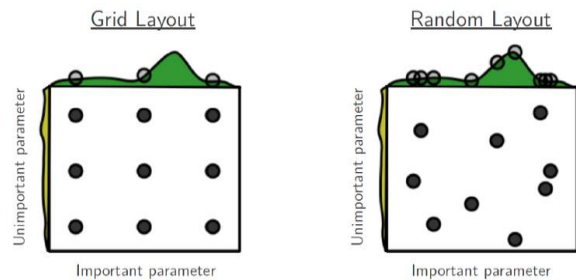
[...]

### 7.5 Simulated Annealing

Escapes local maxima by allowing bad moves but gradually decreases size and frequency [...]

## 7.6 Random Search

Previous Approaches used fixed grids. Some Hyperparameters are pretty sensitive while others have little influence, which is hard to know upfront. Hence, grid search may easily miss best parameters, while random search often yields better results.



## 7.7 Successive Halving

1. Sample set of hyperparameter configurations
2. Evaluate performances of current configurations
3. Sort by performance and throw out bottom half
4. Go back to 2. and repeat until one config remains

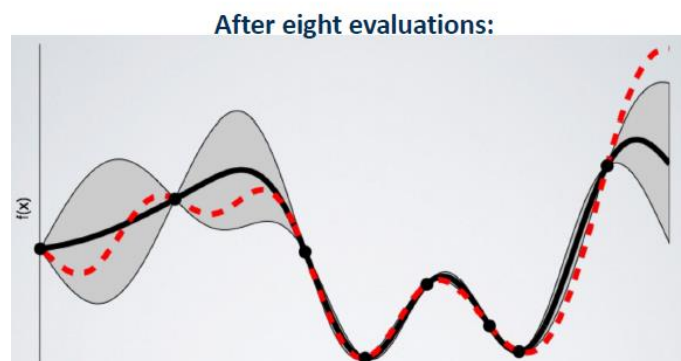
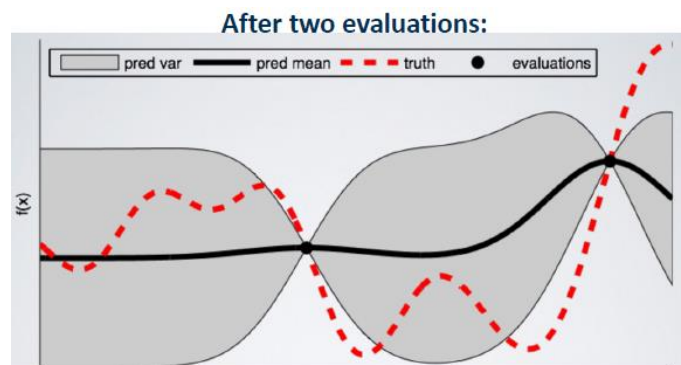
### Example

*"You have a budget of 8 runs over the complete dataset. Describe how this would be realized."*

→ First run with all 256 configs ( $2^8$ ) but only using  $\frac{1}{256}$  of the dataset to train. Then, the second run with the 128 best configs and  $\frac{1}{128}$  of the training data. Continue until only 2 configs (8<sup>th</sup> run) are left and pick the better one.

## 7.8 Bayesian Optimization

1. Build a surrogate probability model of the objective function
2. Find the hyperparameters that perform best on the surrogate
3. Apply these hyperparameters to the true objective function
4. Update the surrogate model incorporating the new results
5. Repeat steps 2-4 until max. iterations or time is reached




### Example

*"You have a budget of 8 runs over the complete dataset. Describe how this would be realized."*

→ Refine the surrogate model with 8 runs and take the best config based on the surrogate.

## 7.9 Genetic Algorithms

 *Genetic Algorithms* are inspired by the process of evolution and involve the use of a population of individuals (or solutions). The main concept includes creating new individuals through crossover of existing ones, introducing random mutations, and selecting the best solutions from each generation by their fitness score (akin to "survival of the fittest").

### 7.9.1 Standard Genetic Algorithm (SGA)

#### Components

- individuals: a hyperparameter setting
- fitness function: performance of a hyperparameter setting
- a crossover method: create a new setting from two others
- a mutation method: change one parameter
- survivor selection

#### Algorithm

```
1 select parents for the mating pool
2 shuffle the mating pool
3 for each consecutive pair apply crossover with probability  $p_c$ , otherwise copy their parents
4 for each offspring apply mutation with probability  $p_m$  for each bit
5 replace population with offspring
```


### 7.9.2 1-Point Crossover

Choose a random point on the two parents, split them at this crossover point. Create children by exchanging tails.

Hyperparameter/ Solution	hp1	hp2	hp3	hp4	hp5	hp6	hp7
s1	true	0.87	0.75	0.01	sgd	0.05	0.72
s2	false	0.75	0.83	0.04	adam	0.04	0.53

Hyperparameter/ Solution	hp1	hp2	hp3	hp4	hp5	hp6	hp7
s1'	true	0.87	0.75	0.01	adam	0.04	0.53
s2'	false	0.75	0.83	0.04	sgd	0.05	0.72



### 7.9.3 Mutation

Alter each gene independently with probability  $p_m$  (the mutation rate)

Hyperparameter/ Solution	hp1	hp2	hp3	hp4	hp5	hp6	hp7
s1	true	0.87	0.75	0.01	sgd	0.05	0.72
s2	false	0.75	0.83	0.04	adam	0.04	0.53

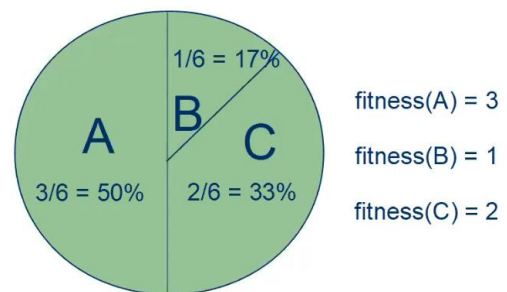
  

Hyperparameter/ Solution	hp1	hp2	hp3	hp4	hp5	hp6	hp7
s1	true	0.87	0.75	0.01	sgd	0.05	0.72
s2	false	0.75	0.86	0.04	adam	0.04	0.53

### 7.9.4 Selection

Better individuals get a higher chance to reproduce.  
The chance is proportional to their fitness.

Implementation: roulette wheel; spin  $n$  times to select individuals.



### 7.9.5 Exploration vs. Exploitation

💡 Crossover-Operations are more explorative, making big jumps to an area somewhere in between two parent areas. Mutation-Operations are rather exploitative, by creating random small diversions, thereby staying near the parent.

## 7.10 Selecting Learners: Meta Learning

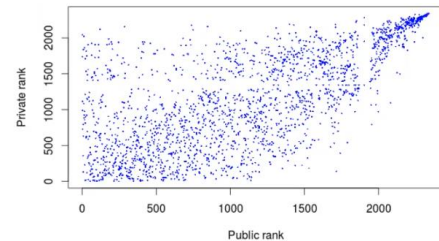
So far, we have looked at finding good hyperparameters for a fixed learner. A similar problem though, is selecting a learner for the task at hand.

Selecting a Learner by Meta Learning, also known as AutoML, involves training a regression model to predict the expected accuracy or other metrics of various machine learning approaches using individual datasets. This technique considers features such as the number of instances and attributes, the distribution of nominal and numerical attributes, and entropy and skewness of classes to determine the most suitable machine learning approach for a given dataset.

## 8 Model Validation

! Just because a model has been performing good on a known test set, this does not necessarily mean it will perform equally well on other unseen data (see Kaggle Data).

Possible causes are that the “official” test data simply has different characteristics than the “local” test data by, for example, an unlucky split train-test-split.



! Furthermore, tuning hyperparameters against test partition or selecting the “best” approach on based on test partition, could lead to overfitting. Or using the local test partition in feature construction would lead to overfitting. → So, when is a model really better than another one?

### 8.1 Z-Test

The old classification model  $M_0$  had an error rate of 0.35 and the new model  $M_1$  one of 0.3. Both are evaluated on the same test set  $T$ . What might be suitable indicators to definitely say one model is better?

- size of test set
- model complexity
- model variance

#### Variant A

Size of test set  $|T| = 40$ . Therefore, a single error contributes 0.025 to the error rate. I.e.,  $M_1$  got two more examples right than  $M_0$ .

💡 The smaller the difference in the error, the larger  $|T|$  should be. But how likely is the error of  $M_1$  lower just by chance?

We can easily construct the binomial distribution given sample size  $n$  and error  $p$  to see that what probability of observing an error of 0.3 or 0.35 is.

$$\mu = n * p$$

$$\sigma = \sqrt{\frac{p * (1 - p)}{n}}$$

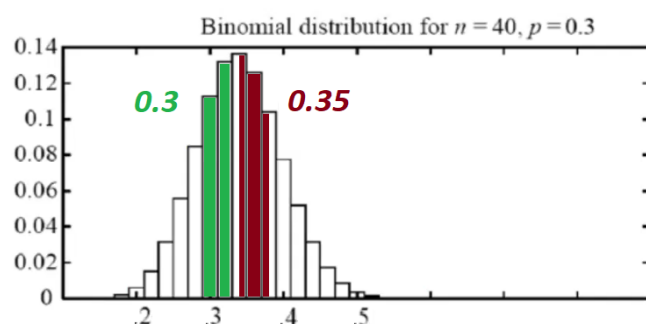
This gives us a confidence interval; with a probability of 95%, we observe error rates in the interval [0.175, 0.442].

Therefore, with only 40 samples, we cannot say whether  $M_1$  or  $M_0$  is better.

For  $n = 2000$ , the 95%-interval now is only [0.28, 0.32]

#### Variant B

Size of test set  $|T| = 2000$ . Then, a single error only contributes 0.0005 to the error rate. I.e.,  $M_1$  got 100 more examples right than  $M_0$ .



💡 The binomial distribution with parameters  $n$  and  $p$  is the discrete probability distribution of the number of successes in a sequence of  $n$  independent experiments, each asking a yes-no question, and each with its own Boolean-valued outcome: success (with probability  $p$ ) or failure (with probability  $q = 1 - p$ ).

## 8.2 Model Variance

💡 What happens if you repeat an experiment on a different test set, or on a different training set, or with a different random seed? With what variance do the results change? Was I lucky?

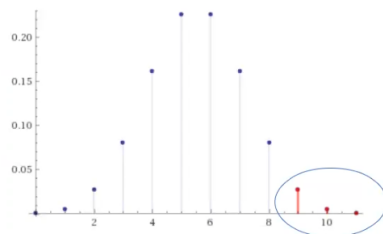
Often, training data differs. Even if you use cross or split validation during development, you might still train a model on the entire training data later.

## 8.3 Sign Test

Model M outperforms model S in  $\frac{9}{12}$  cases, model S outperforms model M in  $\frac{2}{12}$  cases, and one is tied. What is the likelihood of M outperforming S in  $\geq 9$  cases out of 11 (ties removed) simply by chance?

Analogy: what is the likelihood of  $\geq 9$ x heads in 11 coin tosses?

Problem	M	S
1	0.09	0.11
2	0.71	0.72
3	0.77	0.69
4	0.21	0.44
5	0.37	0.37
6	0.85	0.92
7	0.62	0.65
8	0.58	0.55
9	0.79	0.89
10	0.12	0.16
11	0.09	0.15
12	0.19	0.24
Avg.	0.45	0.49



The sign test shows that the probability is 0.03, therefore not concluding significance at  $p < 0.05$ .

## 8.4 Wilcoxon Signed-Rank Test

Puts more emphasis on clear wins and less on close cases.

Approach:

- Rank results by absolute difference
- Sum up ranks for positive (green,  $R^+$ ) and negative (red,  $R^-$ ) outcomes, remove ties
- Using significance tables look up result

Problem	M	S	Delta	Rank
1	0.09	0.11	-0.02	3
2	0.71	0.72	-0.01	2
3	0.77	0.69	0.08	10
4	0.21	0.44	-0.23	12
5	0.37	0.37	0	1
6	0.85	0.92	-0.07	9
7	0.62	0.65	-0.03	4.5
8	0.58	0.55	0.03	4.5
9	0.79	0.89	-0.1	11
10	0.12	0.16	-0.04	6
11	0.09	0.15	-0.06	8
12	0.19	0.24	-0.05	7
Avg.	0.45	0.49		

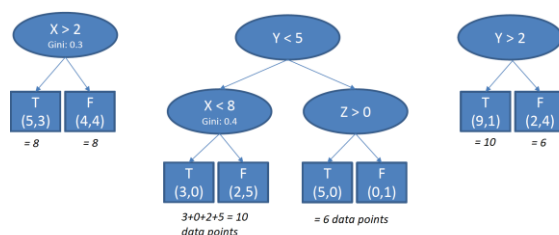
$$R^+ = 62.5 \quad \text{and} \quad R^- = 14.5$$

## 8.5 Feature Importance

Formula

$$I(F) = \frac{1}{\text{total no. of trees}} * \sum_{\text{trees with } F} \sum_{\text{nodes } (n) \text{ with } F} p(n) * GINI$$

Example



$$\begin{aligned}
 I(X) &= \frac{1}{3} * \left( 1 * 0.3 + \frac{10}{16} * 0.4 \right) \\
 &= \frac{1}{3} * (0.3 + 0.25) \\
 &= \frac{0.55}{3} = \mathbf{0.1833}
 \end{aligned}$$

## 8.6 Summary

In summary, the simple z-test proves to be reliably accurate when dealing with datasets that exceed a threshold of 30 observations. However, it's important to note that the sign test lacks the ability to differentiate between cases with large and small margins. This is where the Wilcoxon signed-rank test comes into play. Unlike the sign test, the Wilcoxon signed-rank test is applicable even when dealing with small sample sizes, such as just a handful of datasets. Additionally, the Wilcoxon signed-rank test considers both large and small margins, providing a more comprehensive perspective on the data analysis process.