Feature engineering

How to make your ML experiments work on real data

In part from: https://medium.com/open-machine-learning-course/open-machine-learning-course-topic-6-feature-engineering-and-feature-selection-8b94f870706a

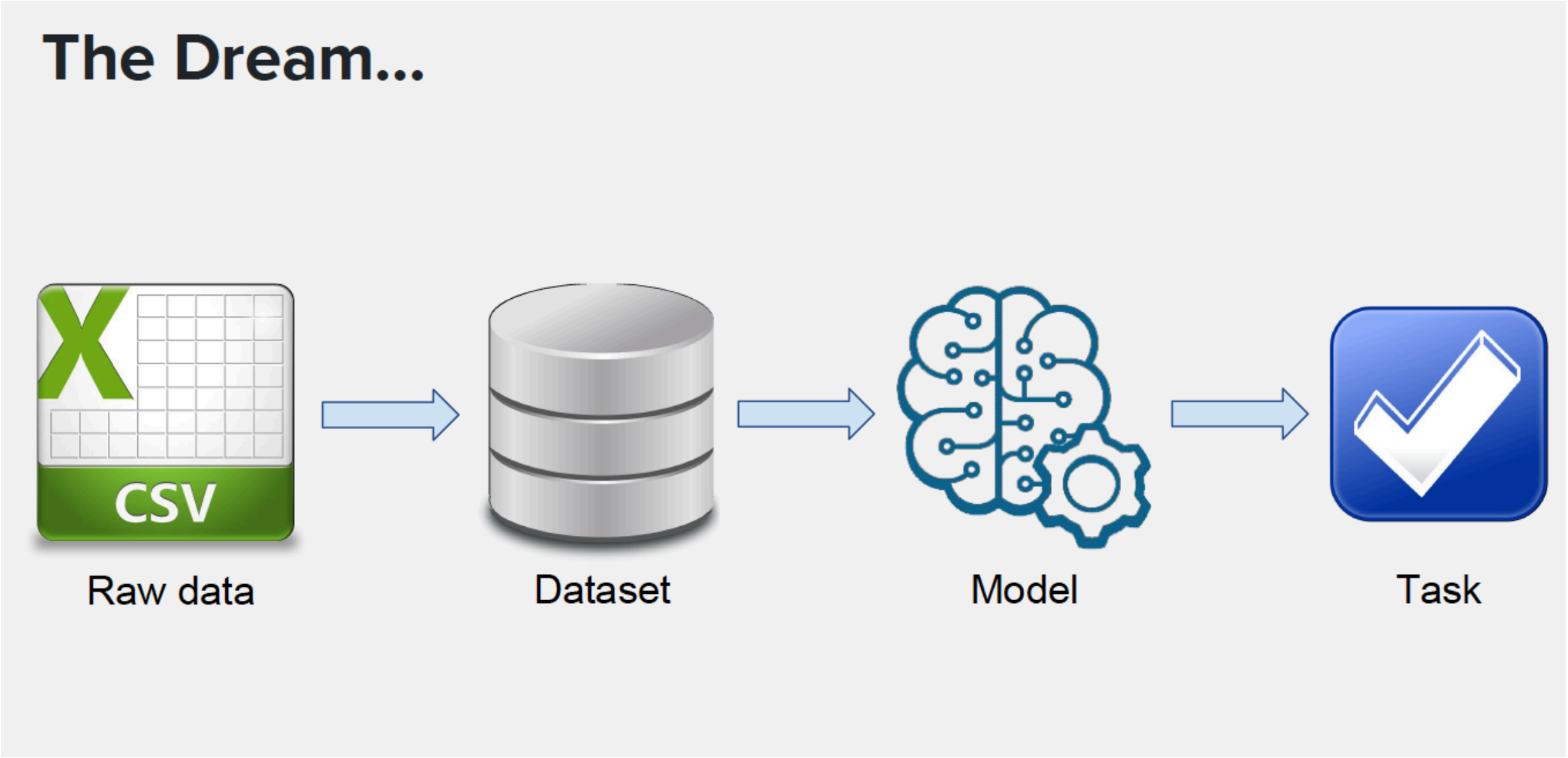
Human's role when applying Machine Learning

- Machine learning provides you with extremely powerful tools for decision making ...
- ... But the role of the developer's decision will still be crucial.
- Your responsibility:
  - setting up the correct problem to be solved/optimized (it's far from straightforward in the real world)
  - choosing a learning algorithm (or a family of algorithms)
  - finding relevant data
  - designing features, feature representation, feature selection . . .

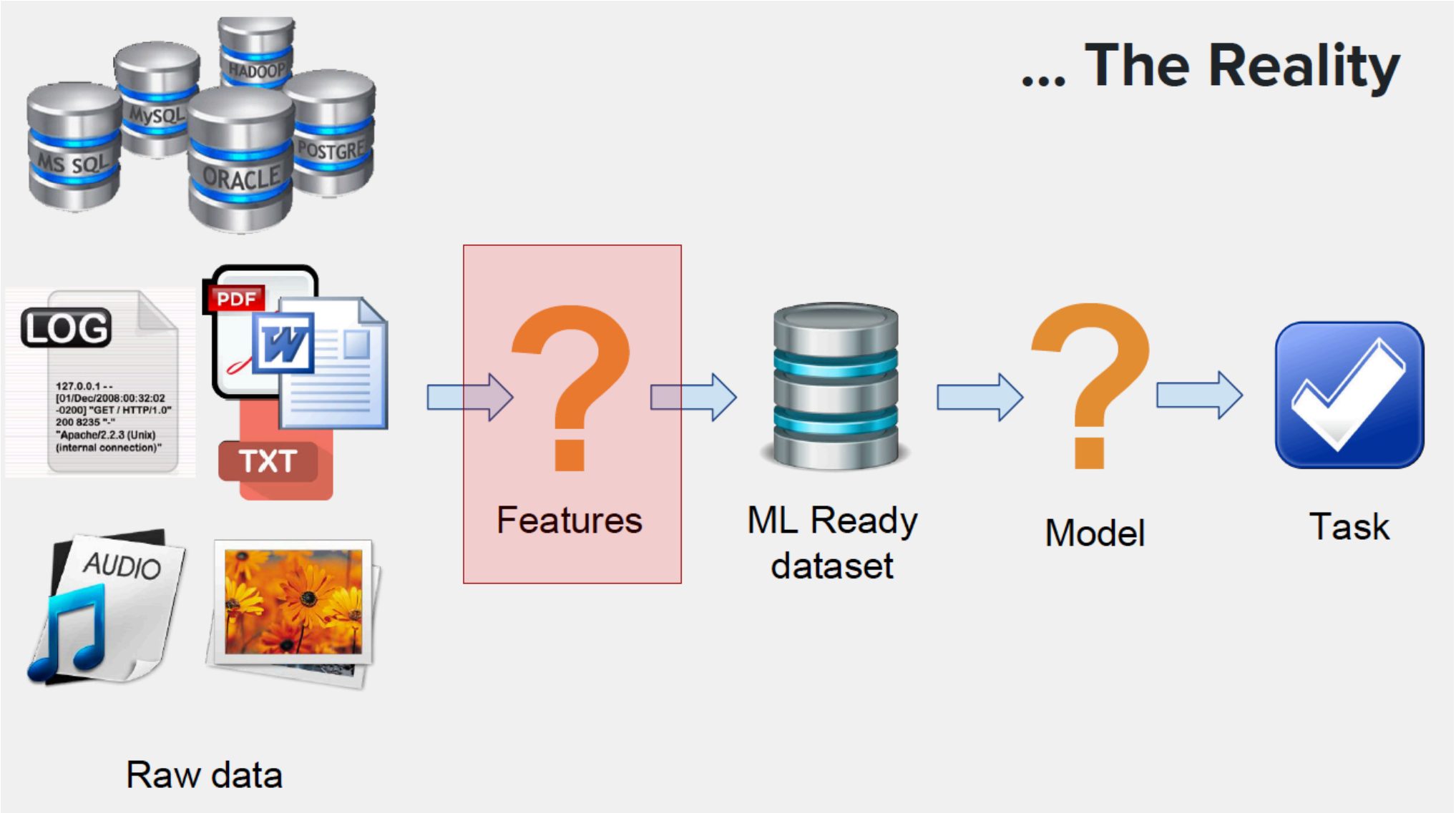
Feature engineering

- Feature engineering - not a formally defined term, just a vaguely agreed space of tasks related to *designing effective feature sets*
- For ML applications two components:
  - First, **understanding the properties of the task you're trying to solve and** how they might interact with the strengths and limitations of the ML model you are going to use
  - Second, experimental work were you **test your expectations** and find out what actually works and what doesn't.

Raw data: the dream



Raw data: the reality



Feature engineering: 4 related tasks

1. **feature identification:** which descriptors are helpful for the task (often not obvious)
2. **feature extraction**: transformation of raw data (e,g, text) into features suitable (e.g., numbers) for modeling;
3. **feature transformation**: transformation of data to improve the accuracy of the algorithm (e.g., normalization, scaling..);
4. **feature selection**: removing unnecessary features.

1. Feature identification

- Given a problem (e.g. image classification, patient disease prediction, predicting successful football players..) often the very first problem is:

Which kind of information would be helpful to accomplish the task? And where can I find it?

- Often this is PRELIMINARY and more crucial than finding and processing the information, once available
- And, this preliminary step is really artwork..

Example

- The transfer fees of football players are getting higher and higher each year.
- Can ML help predicting these values?
  - Problem 1: which information can support the decision system
  - Problem 2: where do I find the data
  - Problem 3 (and ONLY in third position!): which algorithm?

Let’s play

Using domain knowledge and open data sources we find:

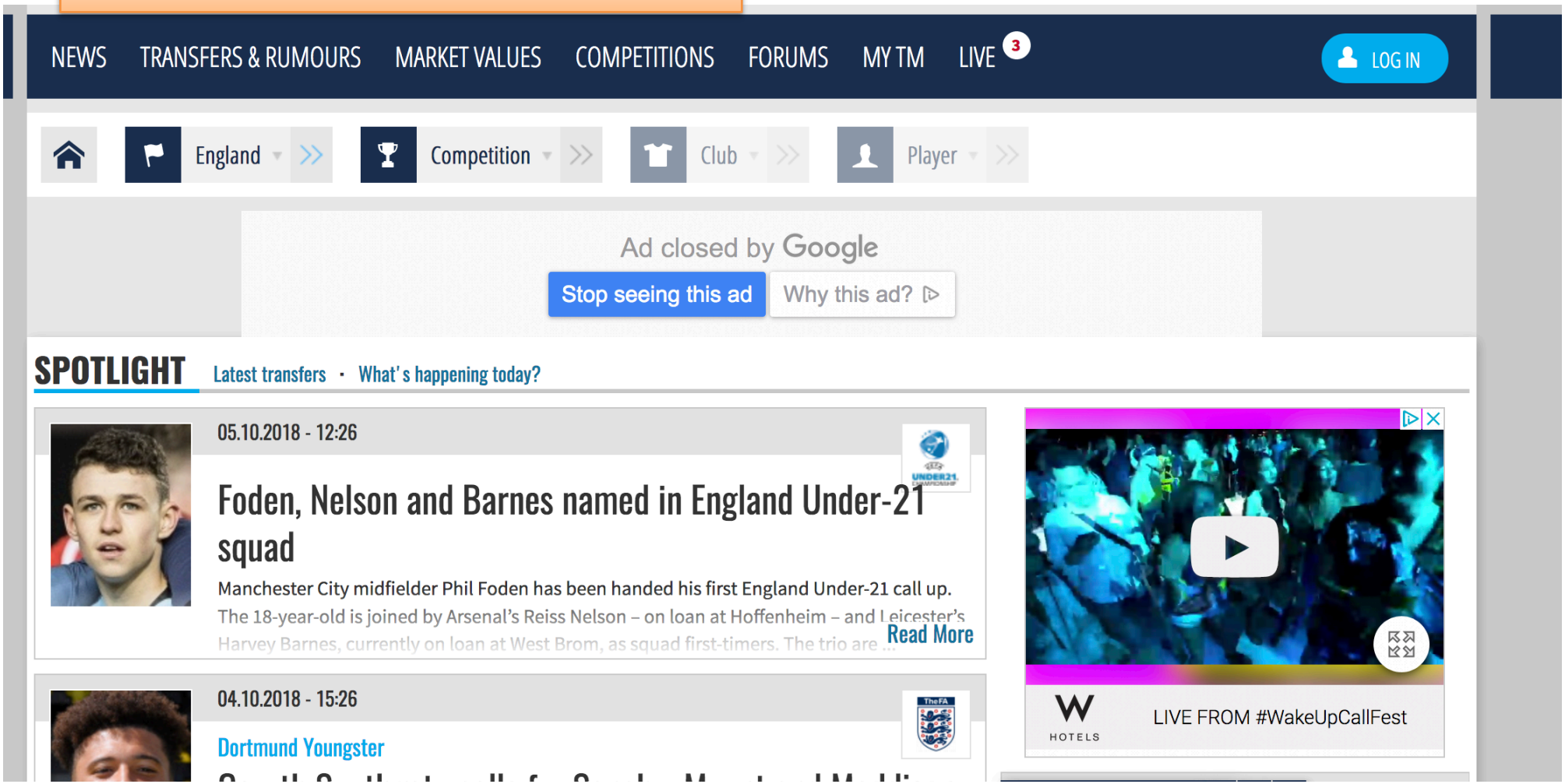
- Features (for each player):
  - **Player's basic information**: team, age, height, weight, . . .
  - **market information**: transfer fee, former team, duration of the contract, when the player joined the team, . . .
  - **performance information**: on-pitch time, actions at the ball, fouls, scores.
- Where can we find these data?
- Data sources: Transfer Market , WhoScored, European Football Database and Garter

Football Player's Performance and Market Value

Miao He[1], Ricardo Cachucho[1], and Arno Knobbe[1,2]

Using on-line infos is far from being easy (often you need scarpers..)

Whoscored: https://www.google.com/



But Google might help you..

kaggle European Soccer Database
www.kaggle.com
Data di aggiornamento: Oct 23, 2016

kaggle Football Events
www.kaggle.com
Data di aggiornamento: Jan 25, 2017

D Football World Cup 2018 Dataset
data.world
Data di aggiornamento: Jun 18, 2018

kaggle International football results from 1872 to 2018
www.kaggle.com
Data di aggiornamento: Jul 11, 2018

kaggle Ultimate 25k+ Matches Football Database -European
www.kaggle.com
Data di aggiornamento: Dec 23, 2016

kaggle Football Delphi
www.kaggle.com
Data di aggiornamento: Aug 16, 2017

kaggle Football World Cup 2018 Dataset
www.kaggle.com
Data di aggiornamento: Jun 14, 2018

kaggle Football Matches of Spanish League

kaggle

European Soccer Database

25k+ matches, players & teams attributes for European Professional Football

Kaggle

5 articoli accademici citano questo set di dati (visualizza in Google Scholar)

Set di dati aggiornato Oct 23, 2016

Autori

Hugo Mathien

Licenza

Database: Open Database, Contents: © Original Authors

Formati di download disponibili dai fornitori

SQLITE , ZIP

Descrizione

The ultimate Soccer database for data analysis and machine learning

What you get:

- +25,000 matches
- +10,000 players
- 11 European Countries with their lead championship
- Seasons 2008 to 2016
- Players and Teams' attributes* sourced from EA Sports' FIFA video game series, including the weekly up
- Team line up with squad formation (X, Y coordinates)
- Betting odds from up to 10 providers
- Detailed match events (goal types, possession, corner, cross, fouls, cards etc...) for +10,000 matches

**16th Oct 2016: New table containing teams' attributes from FIFA !*

Original Data Source:

You can easily find data about soccer matches but they are usually scattered across different websites. A thor
any commercial use of the data. The data was sourced from:

- http://football-data.mx-api.enetscores.com/ : scores, lineup, team formation and events
- http://www.football-data.co.uk/ : betting odds. Click here to understand the column naming system for t

Feature engineering: 4 related tasks

1. **feature identification:** which descriptors are helpful for the task (often not obvious)
2. **feature extraction**: transformation of raw data (e,g, text) into features suitable (e.g., numbers) for modeling;
3. **feature transformation**: transformation of data to improve the accuracy of the algorithm (e.g., normalization, scaling..);
4. **feature selection**: removing unnecessary features.

2. Feature extraction

- In practice, data rarely comes in the form of ready-to-use matrices.
- That's why every task begins with **feature extraction**. Sometimes, it can be enough to read the csv file and convert it into an array, but this is a rare exception.
- Popular types of data from which features can be extracted:
  - Texts
  - Images
  - Geospatial data
  - Date and time
  - Time series, web data, etc.

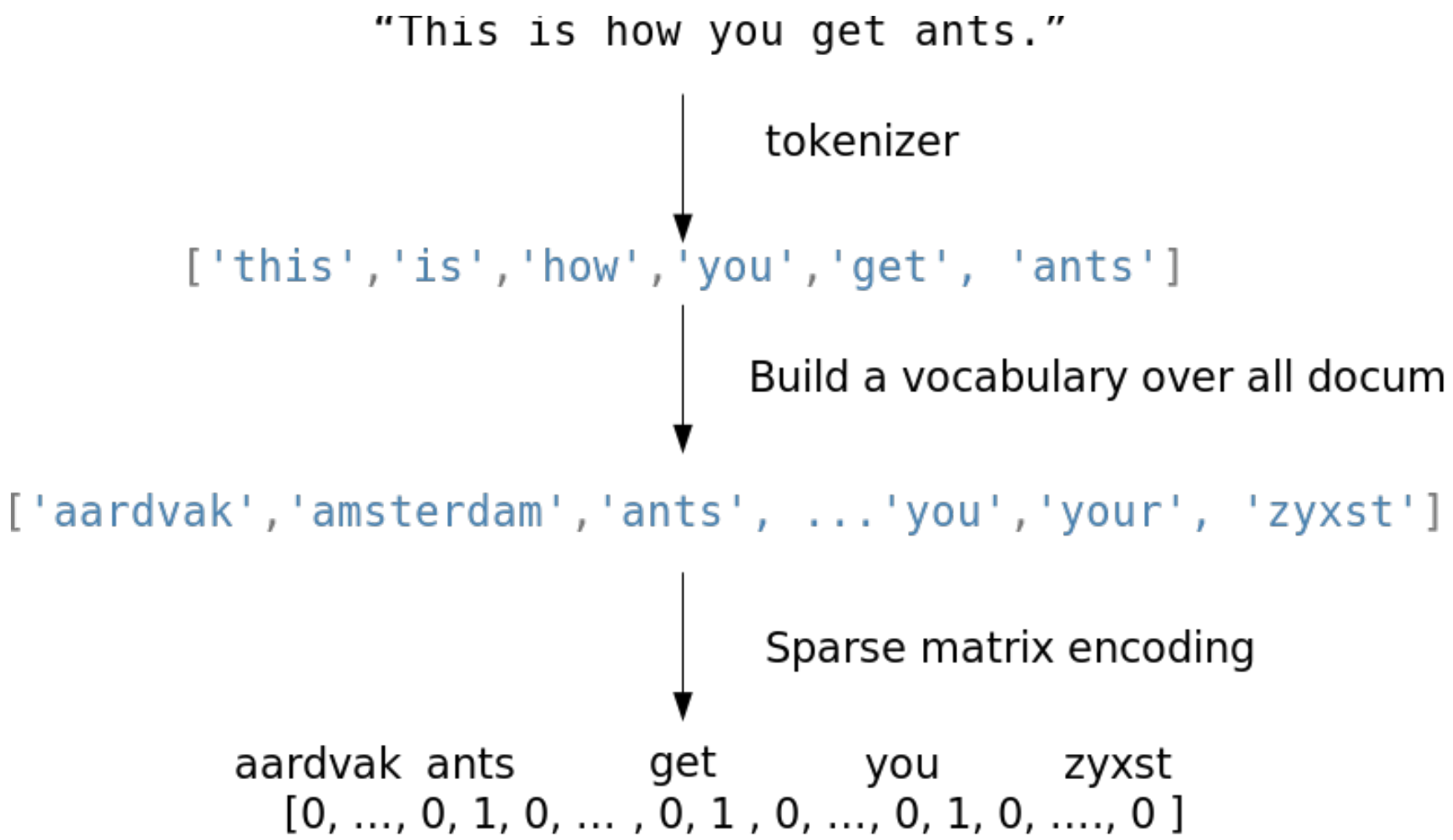
Text (1)

- First step is tokenization, i.e., splitting the text into units (hence, tokens).
  - "Before working with text, one must tokenize it" ➔ before,working,with,text,one,must,tokenize,it
- Next, stemming or lemmatization to normalize data:
  - befor,work,with,text,one,must,token,it
- Finally, text encoding (bag of words is the simplest):
  - Build a vocabulary over all words in all documents:

```
['aardvak','amsterdam','ants', ...'you','your', 'zyxst']
```

  - Encode every document in a **sparse vector** d_i where d_{ij}=1 iff word j of vocabulary is in d_i, else d_{ij}=0

Text (2): Example



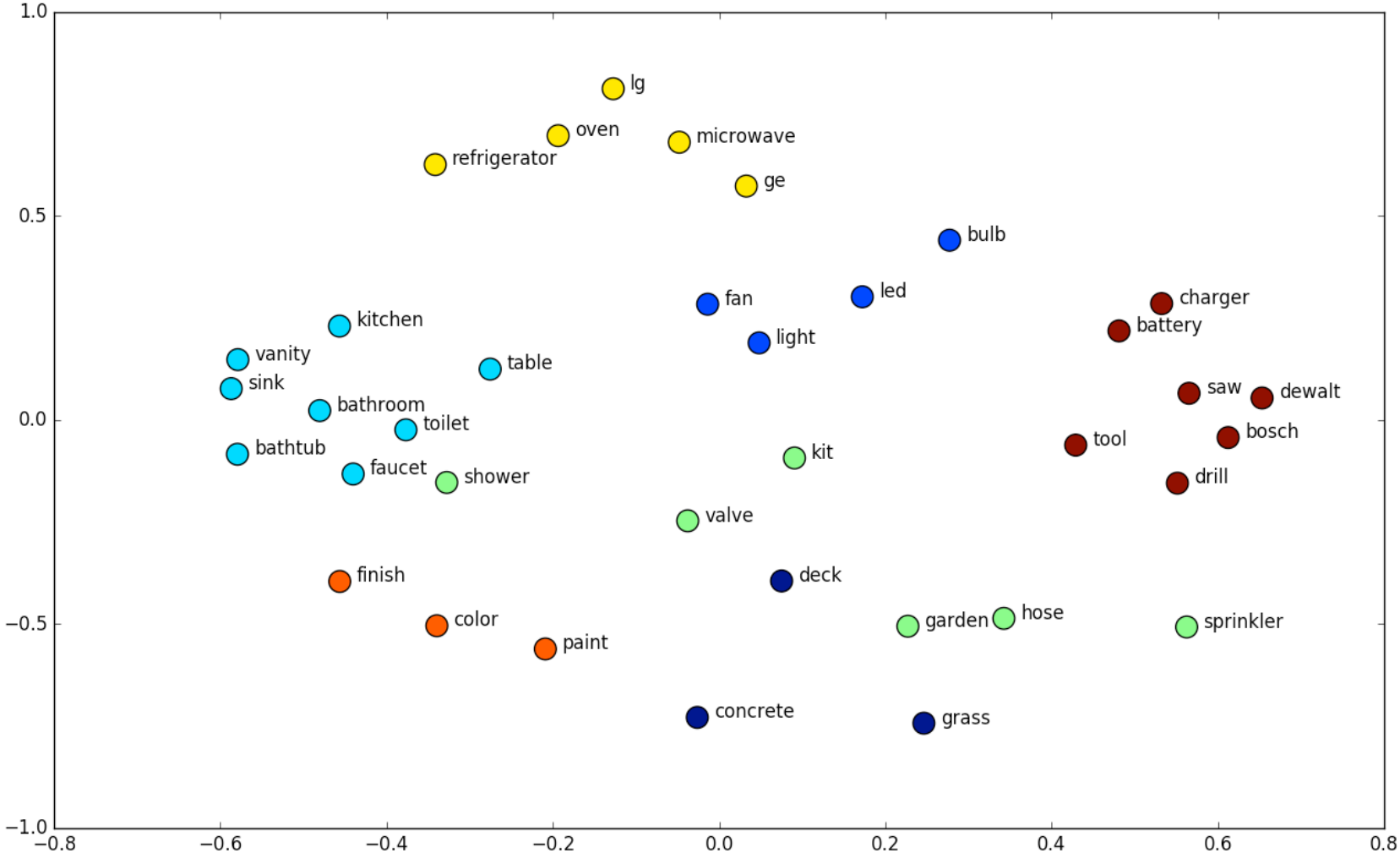
Text (3): embeddings

- Most recent approach to text representation is word embeddings
- More in NLP and W&Social courses, however the idea is that **words**, rather than being represented as a binary value (or a real value) in a "sparse" document vector with |V| dimensions, are represented as "dense" numeric vectors in a "reduced" space
- Words with "close" vectors are similar
- Popular algorithms to obtain word embeddings: Word2vect, Glove, Fastext

Words are “projected” onto semantic spaces

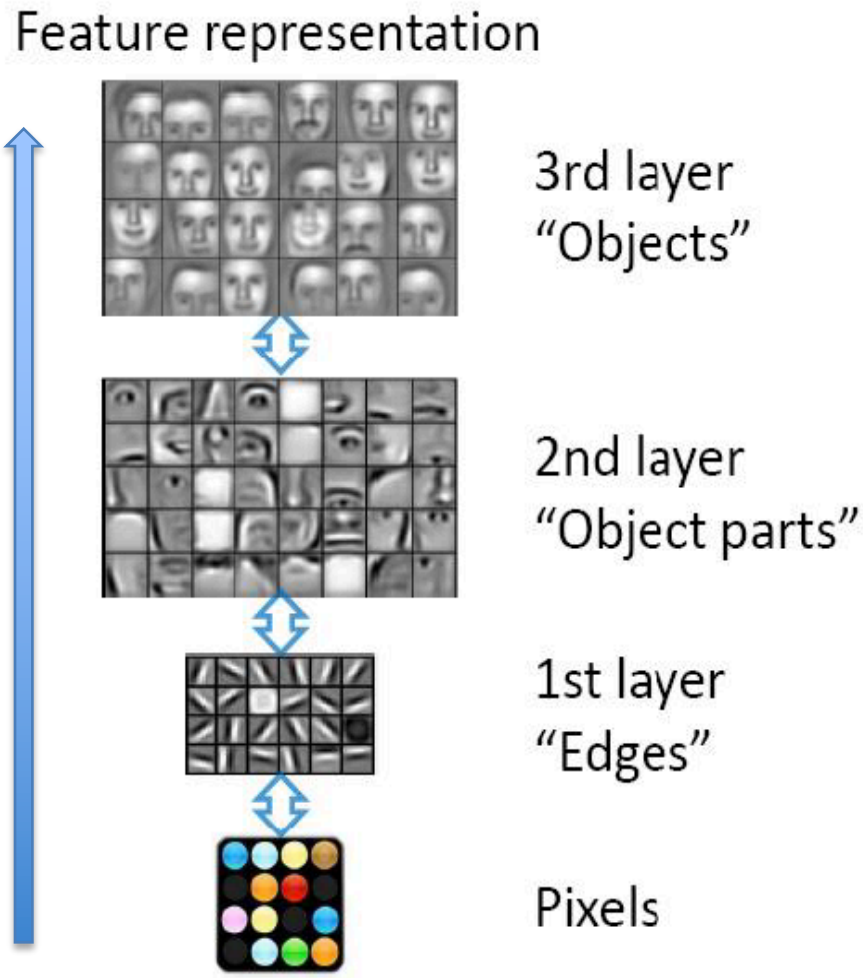
The dimensions are “latent” and automatically learned by looking word contexts. However, the meaning of dimensions is not explicit!
NOTE: We are **unaware** that dimension 1 is, e.g., “royalty”. And, learnt dimensions depends on **the source texts used for learning** – kings, queen and princesses have different vectors if learnt from fairy tales or from gossips newspapers!

Text (4): Example of embeddings



2. Images

- Images can be represented at the level of pixel
- Many popular and (almost) readily available techniques such as Convolutional Networks can be used to progressively extract higher level features (from pixels to edges to semantic elements such as eyes, nose, mouth..)
- Even in this case, the “semantic” of hidden layers is not available!



Images (2)

- Nevertheless, we should not focus too much on neural network techniques. Simpler features are still very useful:
- A more realistic example than cities&mountains: for predicting the popularity of a house rental listing, learn from experts that bright apartments pictures attract more attention, and create a feature such as "the average brightness value of the pixel".

3. Geospatial data

- Geospatial data is often presented in the form of addresses or coordinates (latitude, longitude).
- Depending on the task, you may need two mutually-inverse operations:
  - geocoding (recovering a point of interest from an address)
  - reverse geocoding (recovering an address from a point).
- Both operations are accessible in practice via external APIs from Google Maps or OpenStreetMap.

3. Geospatial data (2)

- CAVEAT:
- Addresses may contain typos, which makes the data cleaning step necessary (see later).
- Coordinates contain fewer misprints, but its position can be incorrect due to **GPS noise or bad accuracy** in places like tunnels, downtown areas, etc.
- If the data source is a mobile device, the geolocation **may not be determined by GPS** but by WiFi networks in the area. While traveling along in Manhattan, there can suddenly be a WiFi location from Chicago.

3. Geospatial data (3)

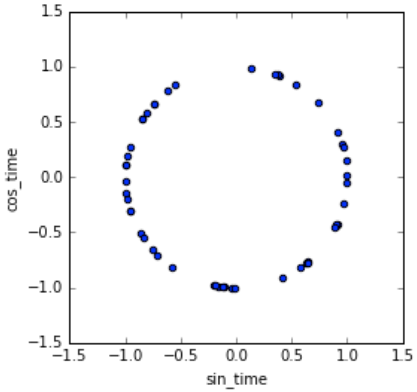
- A point is usually located among infrastructures. Here, you can really unleash your imagination and invent features based on your life experience and domain knowledge: the proximity of a point to the *subway*, the *number of stores* in the building, the distance to the nearest store, the number of *restaurants* around, etc. (e.g. for predicting people movements, for recommending tourist informations..)
- For problems outside an urban environment, you may consider features from more specific sources e.g. the height above sea level, methereological data, etc.

4. **Date and time**

- Days of the week are easy to turned into 7 dummy variables using one-hot encoding (a 7-dimensional binary vector). In addition, we may create a separate binary feature for the weekend “is_weekend”.
- Some tasks may require additional calendar features.
  - For example, cash withdrawals can be linked to a *pay day*; the purchase of a metro card, to the beginning of the month.
  - In general, when working with time series data, it is a good idea to have a calendar with public holidays, abnormal weather conditions, and other important events or anomalyes.

4. **Date and time (2)**

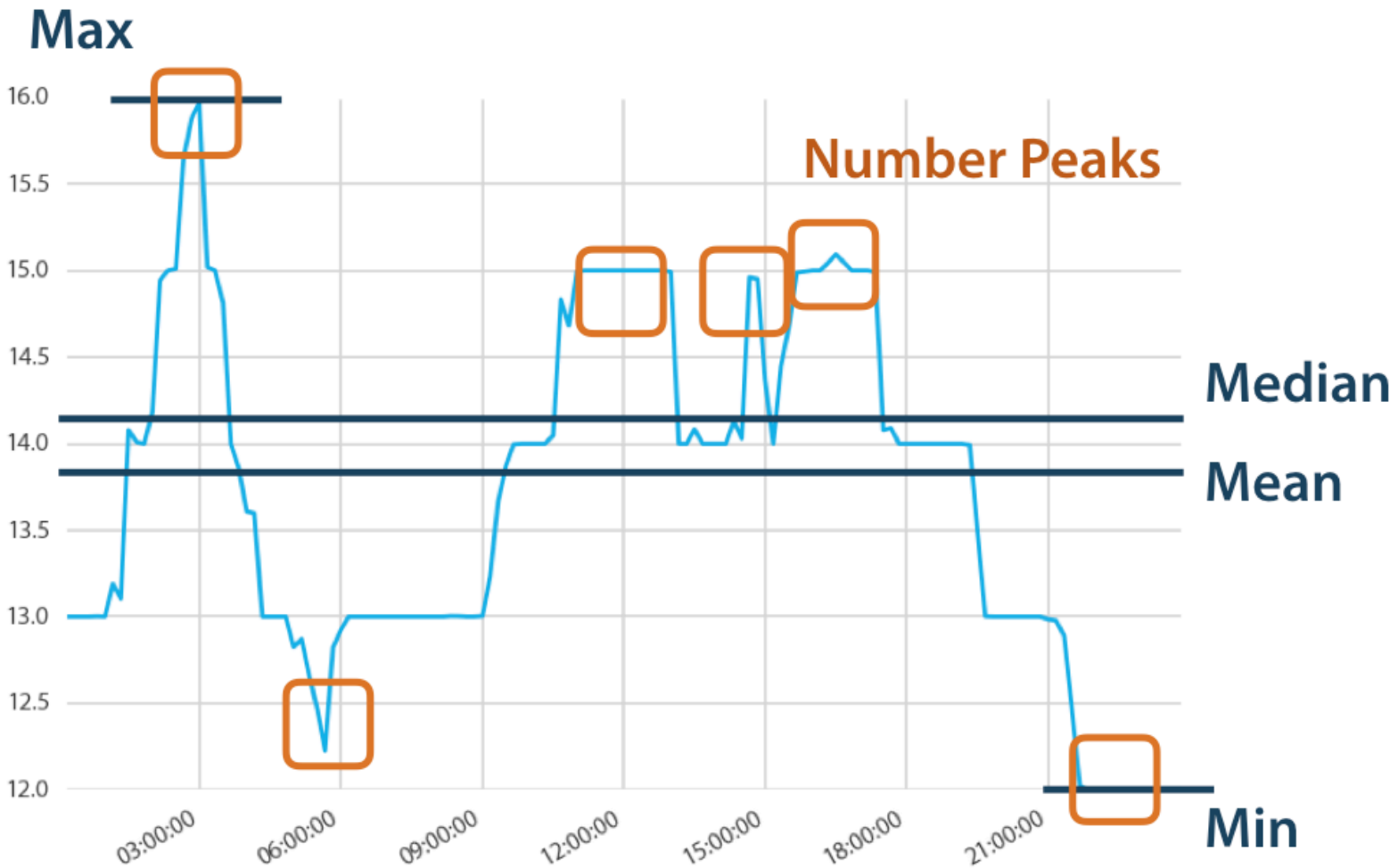
- Dealing with hour (minute, day of the month ...) is not as simple as it seems. If you use the hour as a real variable, we slightly contradict the nature of data: 0<23 while 0:00:00 02.01> 01.01 23:00:00. For some problems, this can be critical.
- There also exist some approaches to such data like projecting the time onto a circle (see e.g., https://ianlondon.github.io/blog/encoding-cyclical-features-24hour-time/).
- This transformation **preserves the distance between points,** which is important for algorithms that estimate distance (kNN, SVM, k-means ...)
- ➔THIS IS AN IMPORTANT POINT: <u>whenever you apply a transformation to your data, need to preserve the relative distance or many ML algorithms won't work properly!</u>



5. Time series

- Sequential data are quite common (e.g. stock market data, personal patient records..)
- The library: http://tsfresh.readthedocs.io/en/latest/ automatically calculates a large number of time series features.
- Further the package contains methods to evaluate the explaining power and importance of such features for classification tasks.

5. Times series (example)



6. Other domains

- In other domains, you can come up with your own features based on intuition about the nature of the data, based on available information, and on the classification task

Feature engineering: 4 related tasks

1. **feature identification:** which descriptors are helpful for the task (often not obvious)
2. **feature extraction**: transformation of raw data (e,g, text) into features suitable (e.g., numbers) for modeling;
3. **feature transformation**: transformation of data to improve the accuracy of the algorithm (e.g., normalization, scaling..);
4. **feature selection**: removing unnecessary features.

3a) Normalization and changing distribution

- Certain algorithms –and platforms- require specific format for data
- E.g., decision trees allow for categorical data, other methods do not;
- Similarly, there are algorithms that suffer for **unbalanced** scaling of features (e.g. one feature with range [0,1] and others with range [-10000.. +1000..]

Normalization: Scaling and centering

- The reason for centering and scaling is that it places all features **on equal standing**.
- Some ML algorithms, e.g. clustering, project instances onto a multi-dimensional space and examine the distances between different data points. In such methods, features **with large absolute differences** in values will be more important (will "affect" more than others the computation of distance).
- Yet generally such absolute differences in values **reflects nothing more than the metric chosen to measure the variable**, and *a priori* it is unreasonable that one variable should be more important than others

Normalization: Scaling and centering (2)

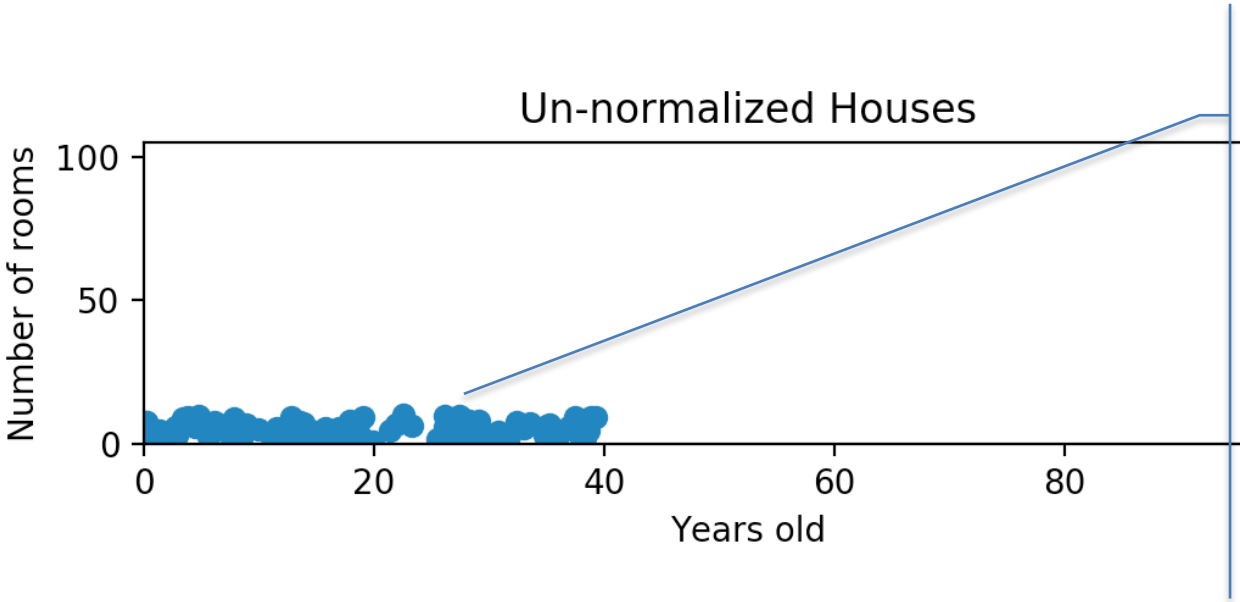
- **Centering** of a real valued feature is done by subtracting its sample **mean** from all values. The equation for calculating the sample mean is (N number of samples):

$$\bar{x} = \sum_{i=1}^{N} x_i$$

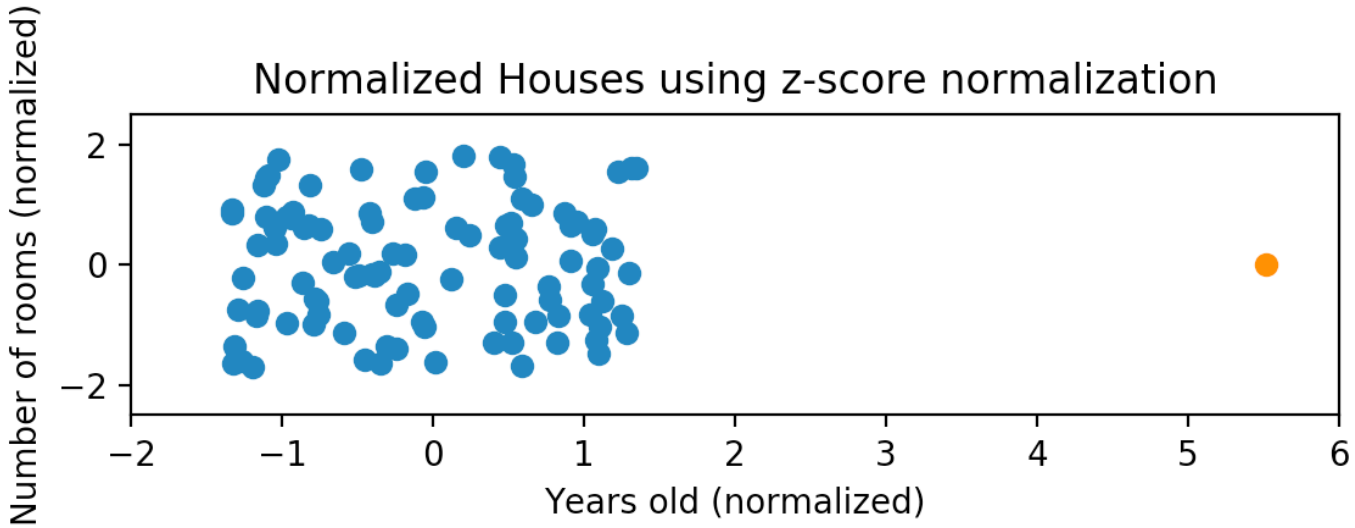
- Scaling of a real valued feature is done by **dividing all its values** by its sample standard deviation:

$$SSD_X = \sqrt{\frac{1}{N-1}\sum_{1}^{N}(x_i - \bar{x})^2}$$

Example



Here , the feature “number of rooms” does not allow any useful separation between datapoints



Normalization: **Changes of Bases**

- Sometimes a ML application involves visual examination of plots of the data or functions of the data, as well as statistical tests.
- A common aim is the generation of a function that would make clearly non-normal data at least “approximately” normal.
- A statistical tests of normality is the ShapiroWilk

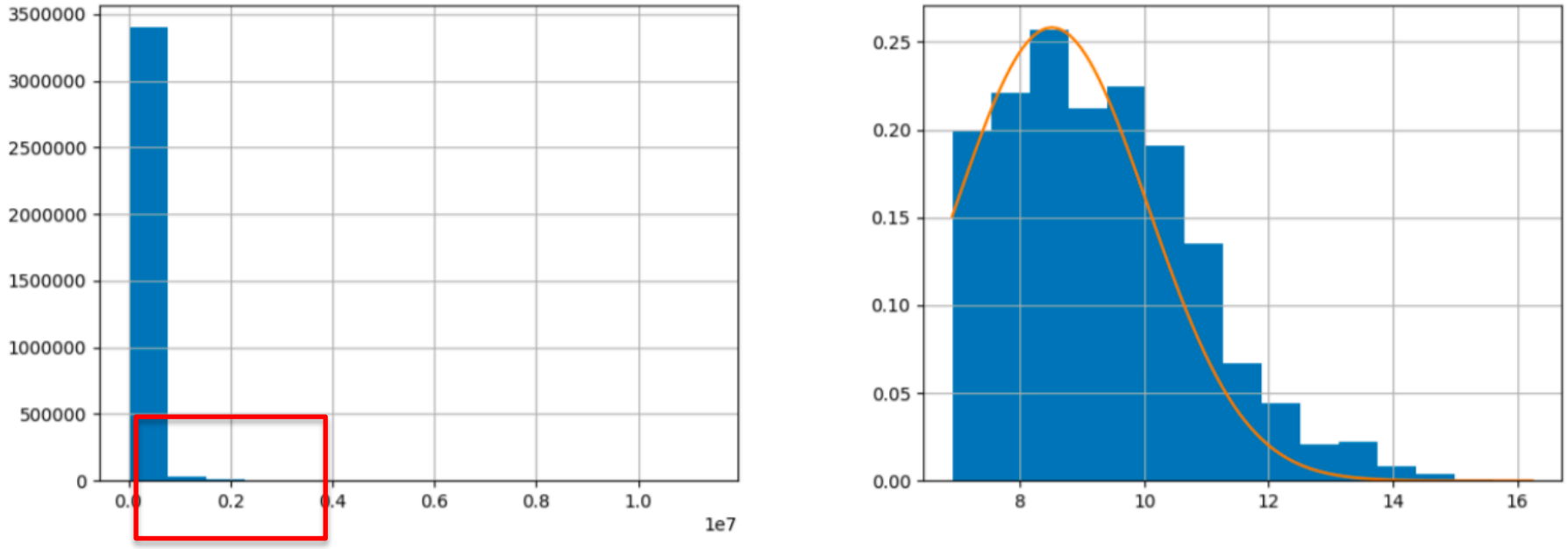
Normalization: **Changes of Bases (2)**

- "skewness" is **asymmetry** in a statistical distribution, in which the curve appears distorted or skewed either to the left or to the right.
- Skewness can be quantified to define the extent to which a distribution differs from a normal (Gaussian) distribution
- Skewness affects the performance of several algorithms (e.g., linear regression, k-nearest neighbour and K-means). See examples https://www.linkedin.com/pulse/question-does-skewness-variable-impact-predictive-data-mosaddar/)

Normalization: **Changes of Bases (3)**

- To reduce the **skewness of the distribution of feature values in a dataset**, we can perform a log transform.
- For more “sparse” distributions, other more complex methods are possible (e.g. *qqnorm*). Some are provided, e.g. in sckitlearn

Example



Original distribution of values, and distribution after applying a log transformation

Normalization: Categorial into numeric

- Certain features can take categorial values (e.g.(Spotify dataset) **artist, track name**..)
- Categories may be nominal (sport, politics, finance..) or ordinal (e.g., dates or week days). Ordinal levels follow a logical order. In nominal categories finding such an order is difficult.
- Some algorithms **do not accept categories**, therefore we need some transformation. Some packages also do not accept categorical variables (SciKitLearn)

Normalization: Categorial into numeric (2)

- **One-hot encoding** is the default way of turning categorical data into numeric. With this method we encode the categorical variable as a **one-hot vector**, i.e. a vector where only one element is non-zero, or *hot*.
- With one-hot encoding, a categorical feature becomes an array whose size is the number of possible choices for that features. With N values, the dimension of the vectors is N

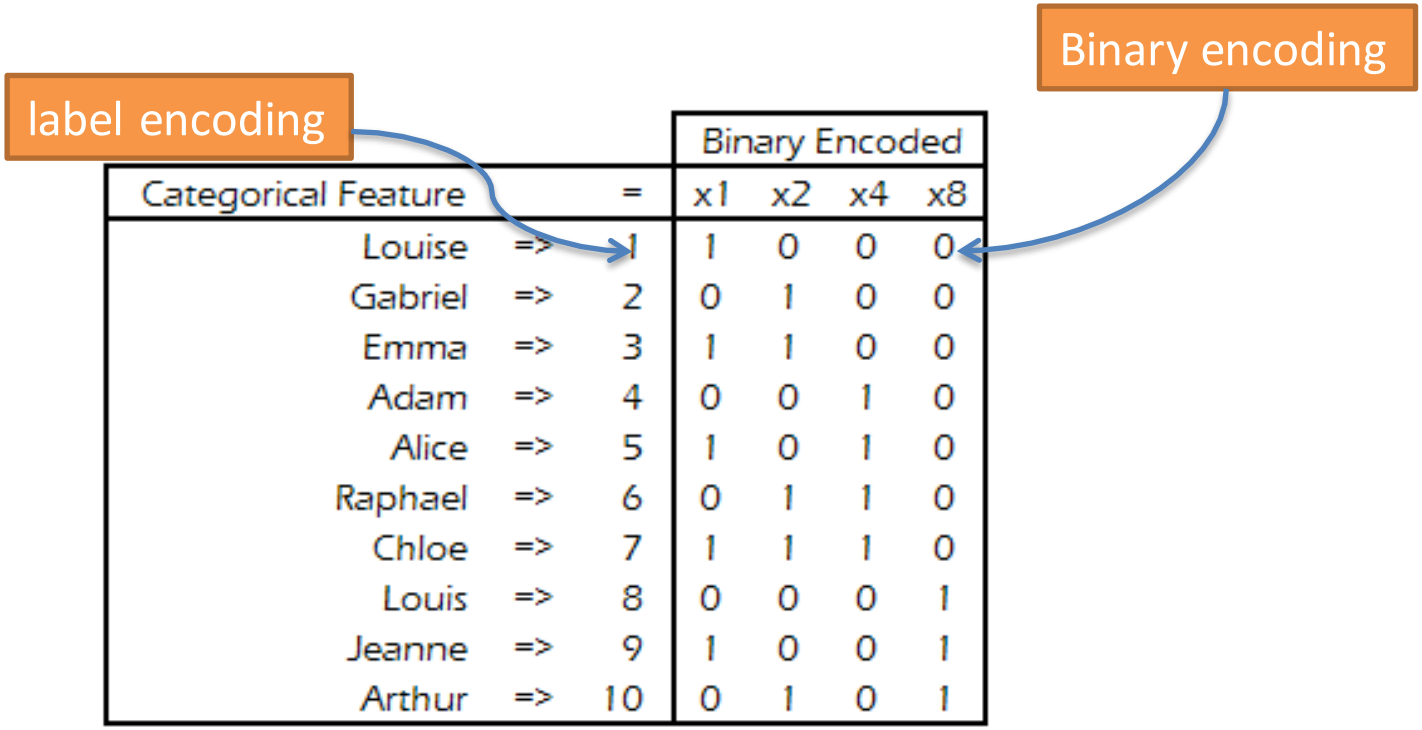
One hot encoding

color	color_red	color_blue	color_green
red	1	0	0
green	0	0	1
blue	0	1	0

Normalization: Categorial into numeric (3)

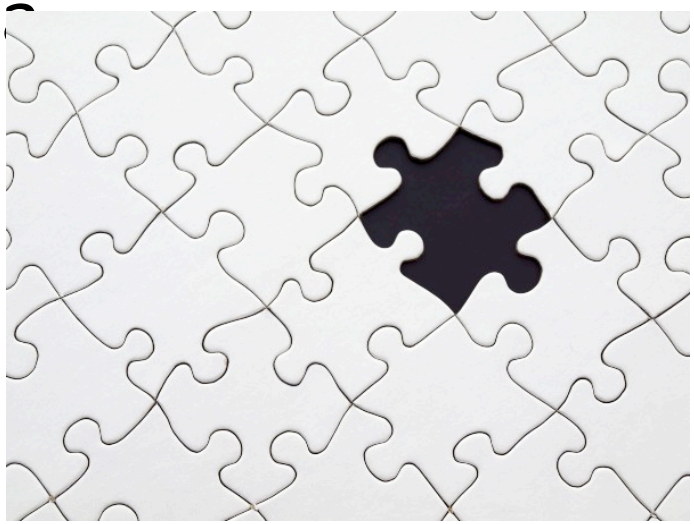
- However, if N is large, one-hot encoding may be a bad idea. Another approach to encoding categorical values is to use a technique called **label encoding**. Label encoding is simply converting each categorical **value** to a number.
- BUT, in those algorithms where the "weight" of each attribute value matters (SVM, regressions..), label encoding introduces an *unjustified* bias towards higher values
- An intermediate alternative is **label binarization** wich introduces log2(N) values.

Example (label and binary encoding)



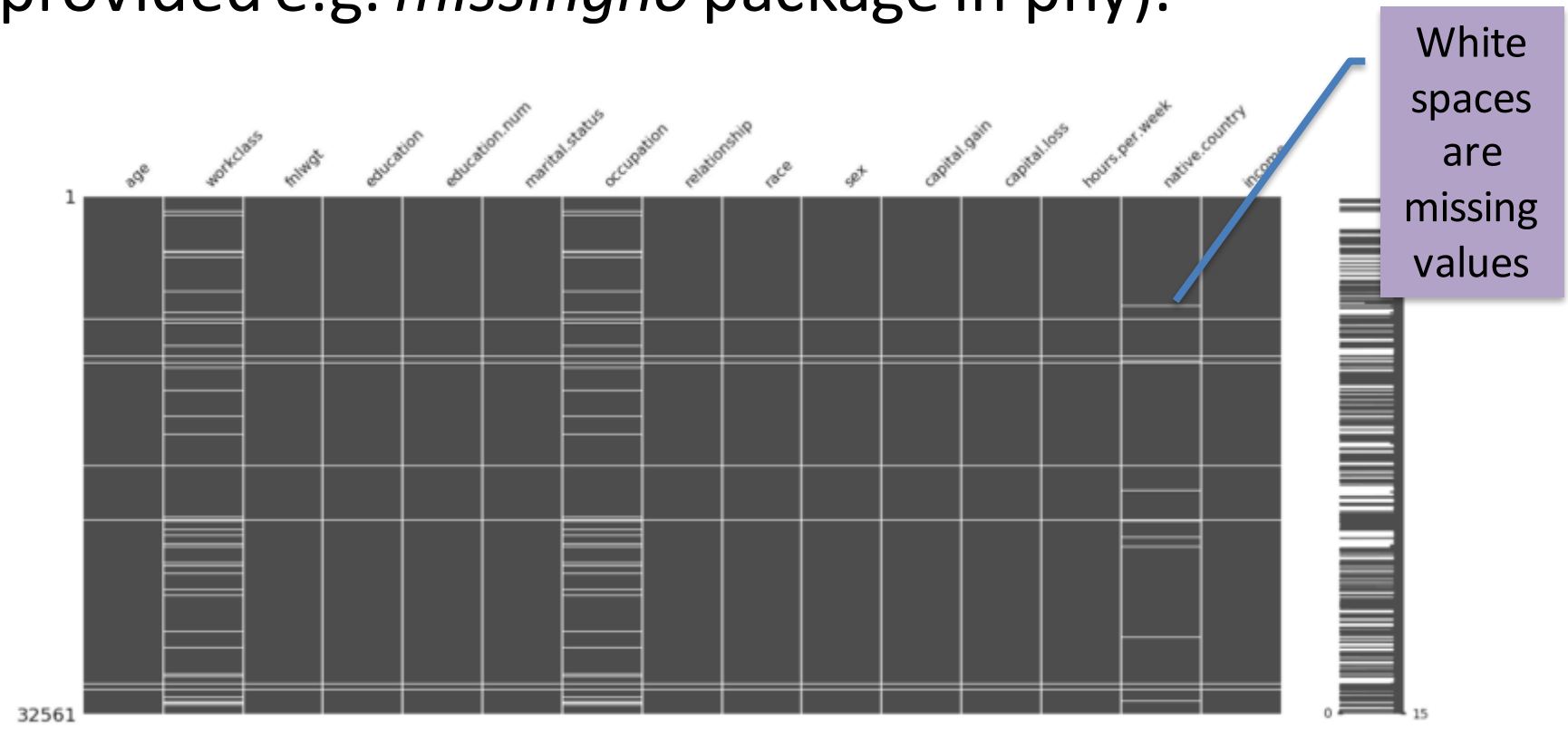
3b. Missing values

- Real-world data often has missing values.
- Data can have missing values for a number of reasons such as observations that were not recorded and data corruption.
- Handling missing data is important as many machine learning algorithms do not support data with missing values.



Missing values (2)

- The first thing to do is count how many you have and try to visualize their distributions (methods are provided e.g. *missingno* package in phy).



Missing values (3)

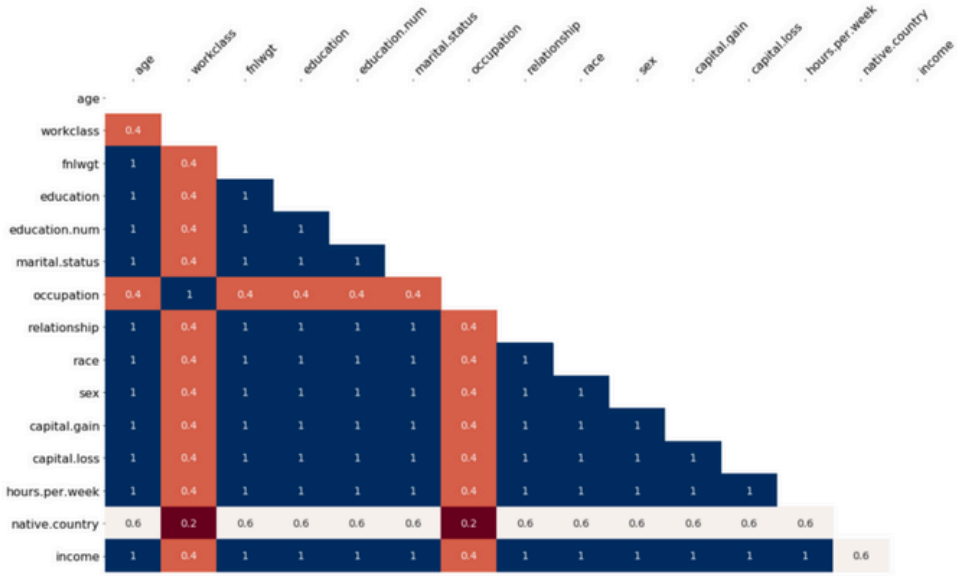
- The simplest thing to do is REMOVAL of instances with missing values (if missing <10%), or removal of attribute (if missing >50%)
- For *numerical* values, a standard and often very good approach is to replace the missing values with mean, median or mode in the entire distribution of values for a given feature
- With *categorical* values, the standard is to replace with the most probable value (although it might be dangerous..)

Missing values: other methods (4)

- **Regression imputation**: A regression model is estimated to predict observed values of a feature x_j based on other features $(x_k..x_n)$, and that model is then used to impute values in cases where that variable is missing.

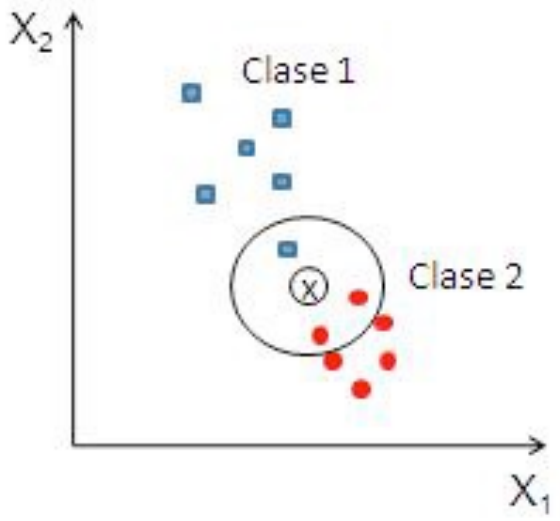
$$x_j = w_k x_k + \cdots w_n x_n$$

- Correlation matrixes among features can help designing the regression model (what are the most helpful features that could predict the missing value of a feature)



Missing values: other methods (5)

- Imputation with K-Nearest Neighbours:
  If j-th feature x_j is missing on instance x_i, we can consider the *K* most similar instances that have no missing value in *j*-th feature.
- Then impute the missing value with the most frequent item (the *mode*) amongst the *j*-th feature of these *K* patterns.

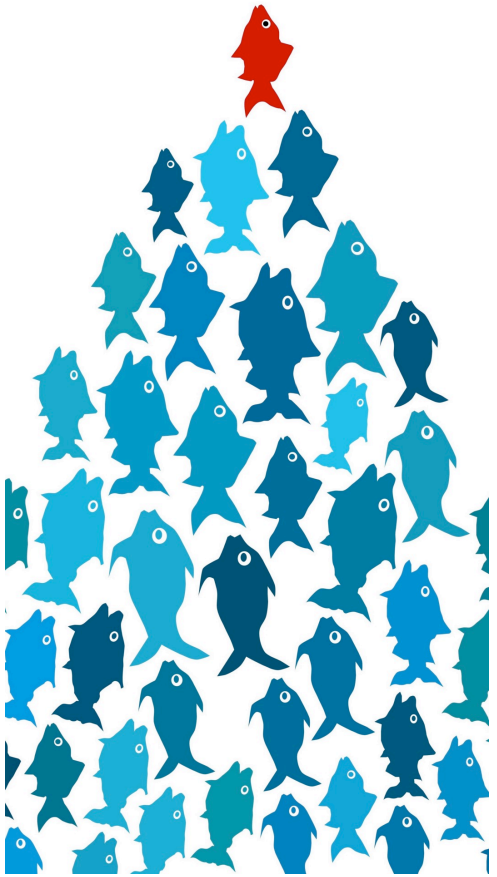


3c. Data Augmentation

- Data augmentation refers to methods that **add more features** to available data
- For image datasets, you can rotate, scale, translate, intepolate..
- For other types of datasets, you can add new features that can be inferred from the available data
- For example, in a database of football matches, you may want to add for each team the time elapsed between the current match and the last victorious match

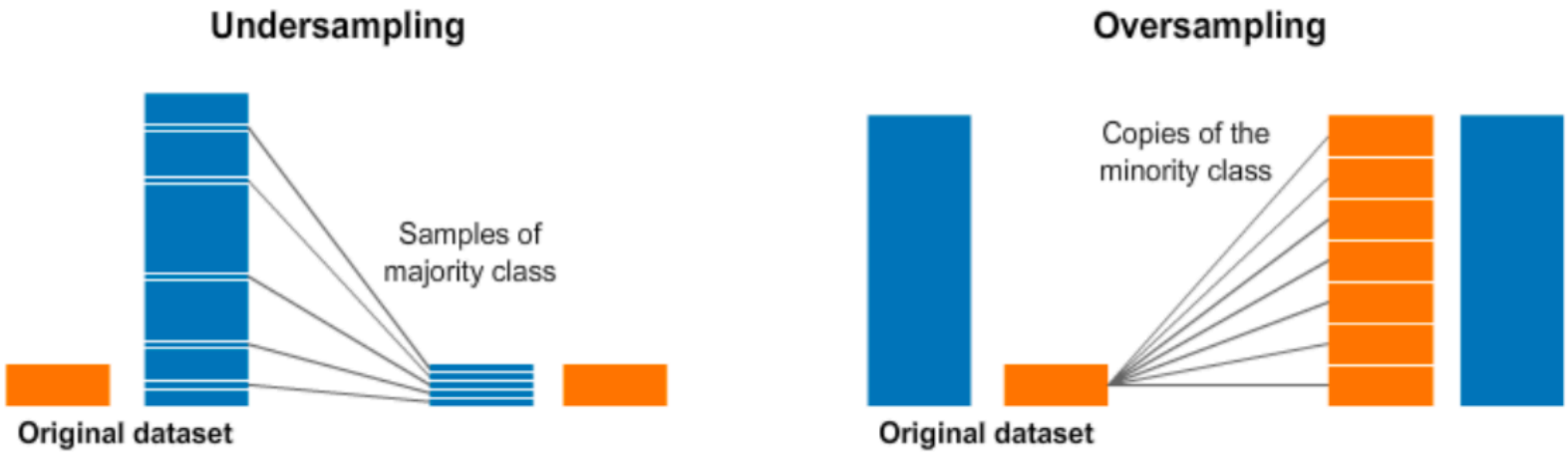
3d. Inbalanced categories

- Class imbalance is when each class does not make up an equal portion of your data-set
- For example, suppose you have two classes — A and B.
- Class A is 90% of your data-set and class B is the other 10%, but you are most interested in identifying instances of class B.
- You can reach an accuracy of 90% by simply predicting class A every time, but this provides a useless classifier for your intended use case.



Imbalanced categories: sampling

- **Sampling:** A simple way to fix imbalanced data-sets is simply to balance them, either by **oversampling** instances of the minority class or **undersampling** instances of the majority class.

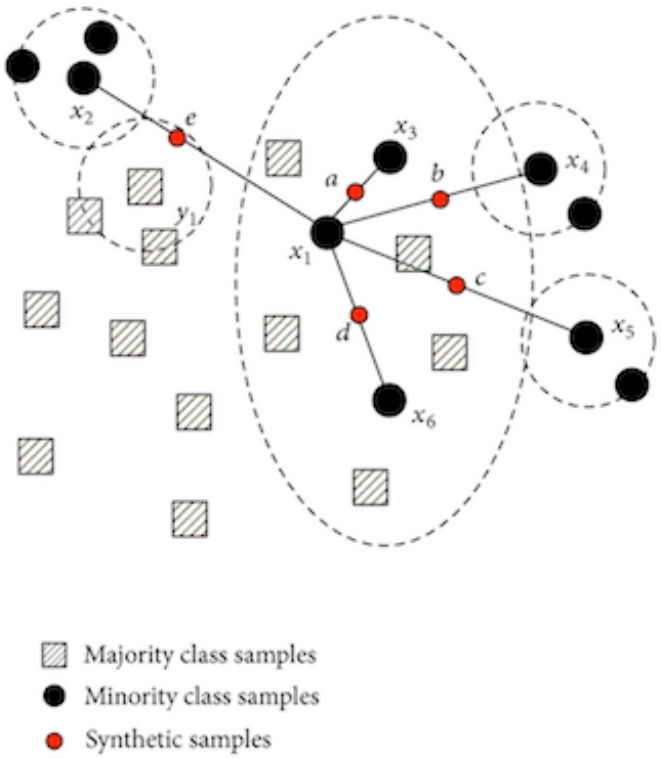


undersampling and oversampling

Inbalanced categories (2)

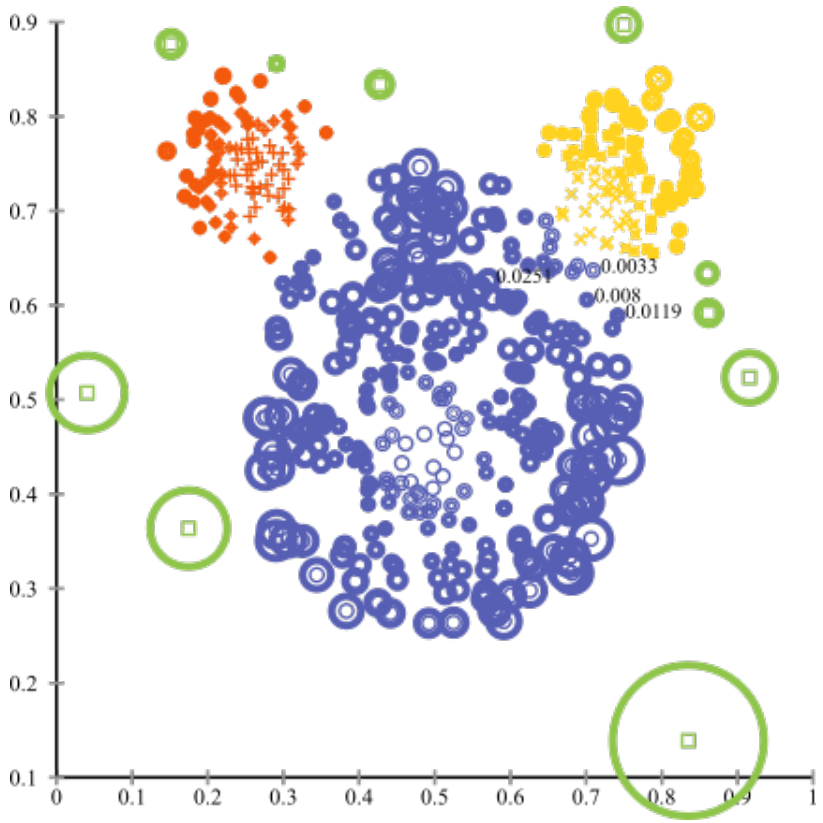
- A more powerful sampling method is **SMOTE** Synthetic Minority Oversampling TEchnique , which creates new instances of the minority class by forming convex combinations of neighboring instances. https://www.researchgate.net/publication/220543125_SMOTE_Synthetic_Minority_Over-sampling_Technique

- As the graphic shows, it effectively draws lines between minority points in the feature space, and samples along these lines.



Inbalanced categories (3)

- **Anomaly Detection:** we assume that there is a "normal" distribution(s) of data-points, and anything that sufficiently deviates from that distribution(s) is an anomaly.
- When we reframe our classification problem into an anomaly detection problem, we treat the majority class as the "normal" distribution of points, and the minority as anomalies.
- We can also simply **ignore** anomalies (however, it depends on application: if anomalies are, e.g. fraudolent behaviors, then this is exactly what we may be looking for!)



Inbalanced categories (4)

- **Cost-sensitive Learning** In regular learning, we treat all misclassifications equally (regardless of the class which is misclassified), which causes issues in imbalanced classification problems, as there is no extra reward for identifying the minority class over the majority class.
- Cost-sensitive learning changes this, and uses a function **C(p, t)** (usually represented as a matrix) that specifies the **cost** of misclassifying an instance of class **t** as class **p**.
- The algorithm, in the attempt of minimizing the cost of wrong decisions, will pay more attention to the minority elements

	Actual Positive $y_i = 1$	Actual Negative $y_i = 0$
Predicted Positive $c_i = 1$	C_{TP_i}	C_{FP_i}
Predicted Negative $c_i = 0$	C_{FN_i}	C_{TN_i}

Feature engineering: 4 related tasks

1. **feature identification:** which descriptors are helpful for the task (often not obvious)
2. **feature extraction**: transformation of raw data (e,g, text) into features suitable (e.g., numbers) for modeling;
3. **feature transformation**: transformation of data to improve the accuracy of the algorithm (e.g., normalization, scaling..);
4. **feature selection**: removing unnecessary features.

Feature selection

- How many? Are there enough? Are there too many?
- For any ML task, you can easily come up with dozens of features and extract them from various external sources.
- However, the **number** and **complexity** of NEEDED features often depend on the specific task addressed
- For example, if you need to distinguish city landscapes from mountain landscapes you don't need pixel features (a color

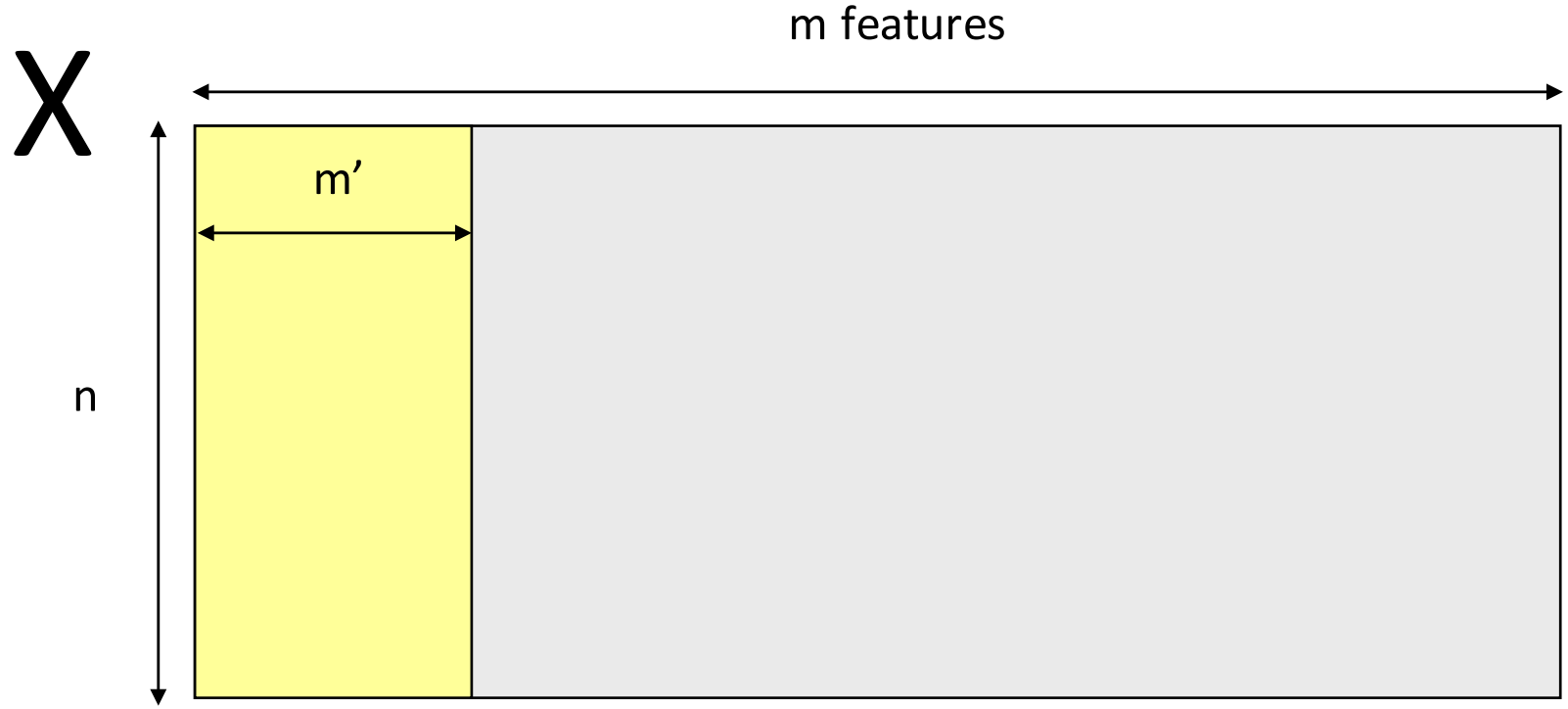
Feature selection

- In many practical cases, one may come out with hundreds –and sometimes more – potentially useful features (so the “too many” is the most frequent case)
- Not easy to say what is truly useful, nor if some features are correlated
  - Adding many potentially correlated features can decrease model performance
  - Too many features make models less interpretable and less generalizable
- So, we need **automatic tools for feature selection (filtering)**

Feature Selection

The abundance of data in contemporary datasets demands development of clever algorithms for detecting feature importance

If we have too many features, it is hard to select those that truly impact on performance (this is specially true for deep learning and neural algorithms)

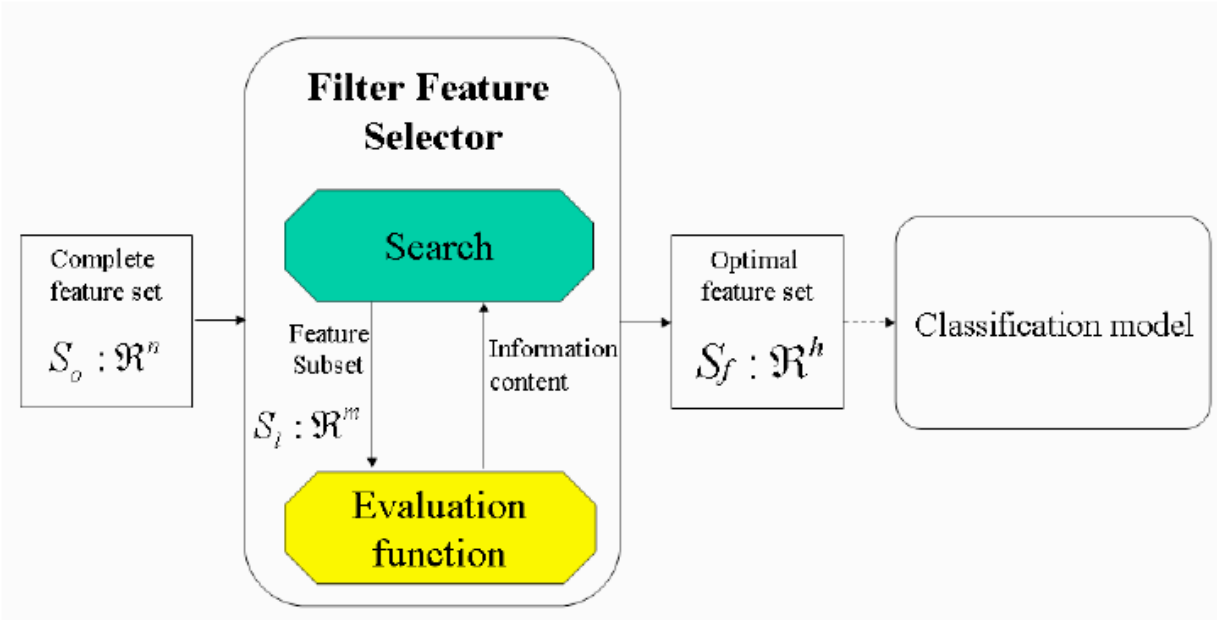


4. Feature selection

- Since exhaustive search for optimal feature subset is infeasible in most cases, many search strategies have been proposed in literature, often classified in three types:
  - Filter methods
  - Wrapper methods
  - Embedded and hybrid methods

a) Filter methods

- Filter methods select features based on a performance measure *regardless of the employed data classification algorithm*.
- Only <u>after</u> the best features are found, the ML algorithms can use them.



Filter methods (2)

- We can roughly classify the developed measures for feature filtering into: *information, distance, consistency, similarity, and statistical measures*.
- Furthermore, *univariate* feature filters evaluate (and usually rank) a **single feature**, while *multivariate* filters evaluate an entire feature subset.

A list of common filter methods

Name	Filter class	Applicable to task
Information gain	univariate, information	classification
Gain ratio	univariate, information	classification
Symmetrical uncertainty	univariate, information	classification
Correlation	univariate, statistical	regression
Chi-square	univariate, statistical	classification
Inconsistency criterion	multivariate, consistency	classification
Minimum redundancy, maximum relevance (mRmR)	multivariate, information	classification, regression
Correlation-based feature selection (CFS)	multivariate, statistical	classification, regression
Fast correlation-based filter (FCBF)	multivariate, information	classification
Fisher score	univariate, statistical	classification
Relief and ReliefF	univariate, distance	classification, regression
Spectral feature selection (SPEC) and Laplacian Score (LS)	univariate, similarity	classification, clustering
Feature selection for sparse clustering	multivariate, similarity	clustering
Localized Feature Selection Based on Scatter Separability (LFSBSS)	multivariate, statistical	clustering
Multi-Cluster Feature Selection (MCFS)	multivariate, similarity	clustering
Feature weighting K-means	multivariate, statistical	clustering
ReliefC	univariate, distance	clustering

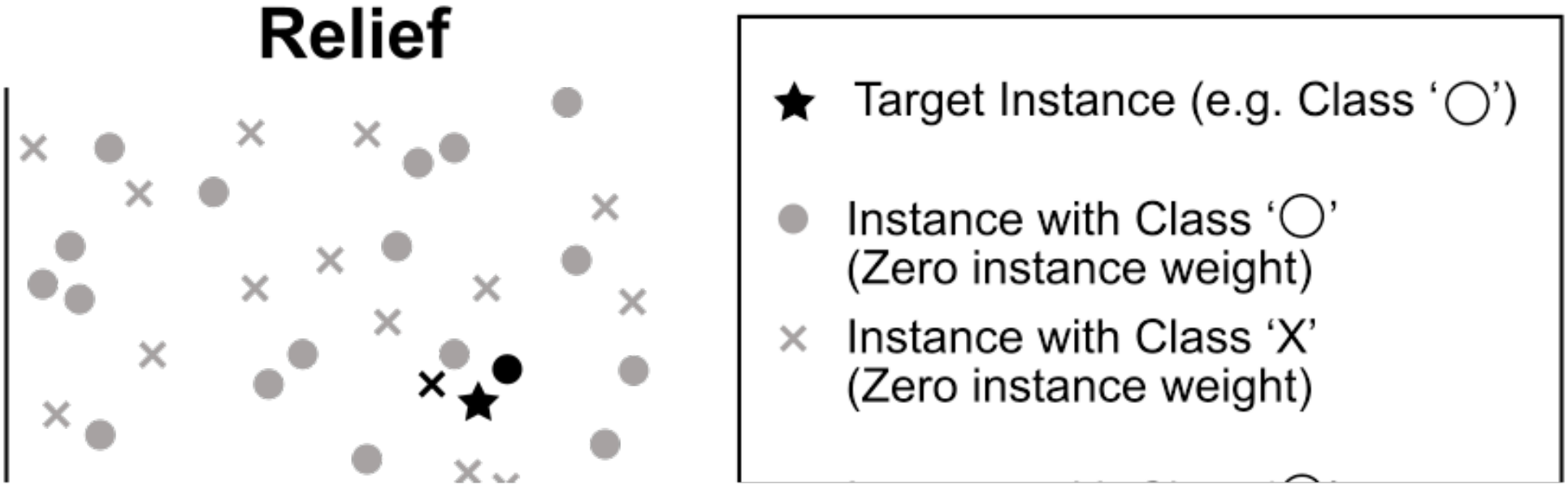
Examples of filters

- **Information gain** (information, univariate) YOU KNOW!
- **Relief(F)** (distance, univariate): consider all features as independent ones and estimate the relevance (quality) of a feature based on its ability to distinguish instances located **near** each other.
  - the algorithm iteratively selects a random instance and then searches for its two nearest neighbors: a **nearest hit** (from the **same class**, e.g., negative) and a **nearest miss** (from the different class).
  - For each feature x_i the estimation of its quality (weight W_i) is updated depending on the differences between the current instance and its nearest hit and miss along the corresponding attribute axis.

$$W_i = W_i - (x_i - \text{nearHit}_i)^2 + (x_i - \text{nearMiss}_i)^2$$

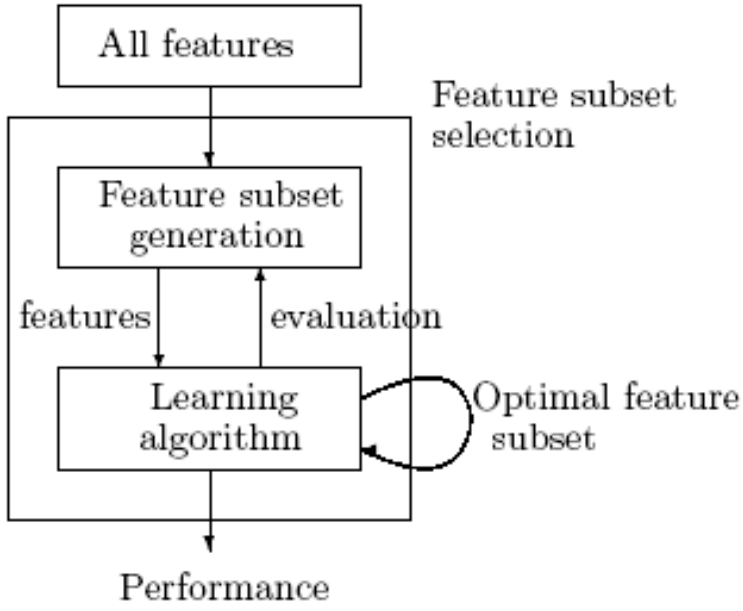
  - Several measures to compute difference (euclidean distance, Manhattan distance..)
  - An implementation is described in http://www.ijird.com/index.php/ijird/article/viewFile/81772/63106

Example (Relief)



b) Wrappers

- Wrappers consider feature subsets by the quality of the performance on a specific ML algorithm, which is taken as a “black box” evaluator.



(b) Wrapper method

Wrappers (2)

- Thus, for classification tasks, a wrapper will evaluate subsets of features based on the classifier performance (e.g. Naïve Bayes or Decision Forest or Neural Networks).
- The evaluation is repeated for each subset, and the subset generation is dependent on the search strategy, in the same way as with filters.
- Wrappers are much slower than filters in finding sufficiently good subsets because they depend on the the considered algorithm.

Wrappers (3)

- **Wrapper methods:**
- recursive feature elimination
- sequential feature selection algorithms
- genetic algorithms

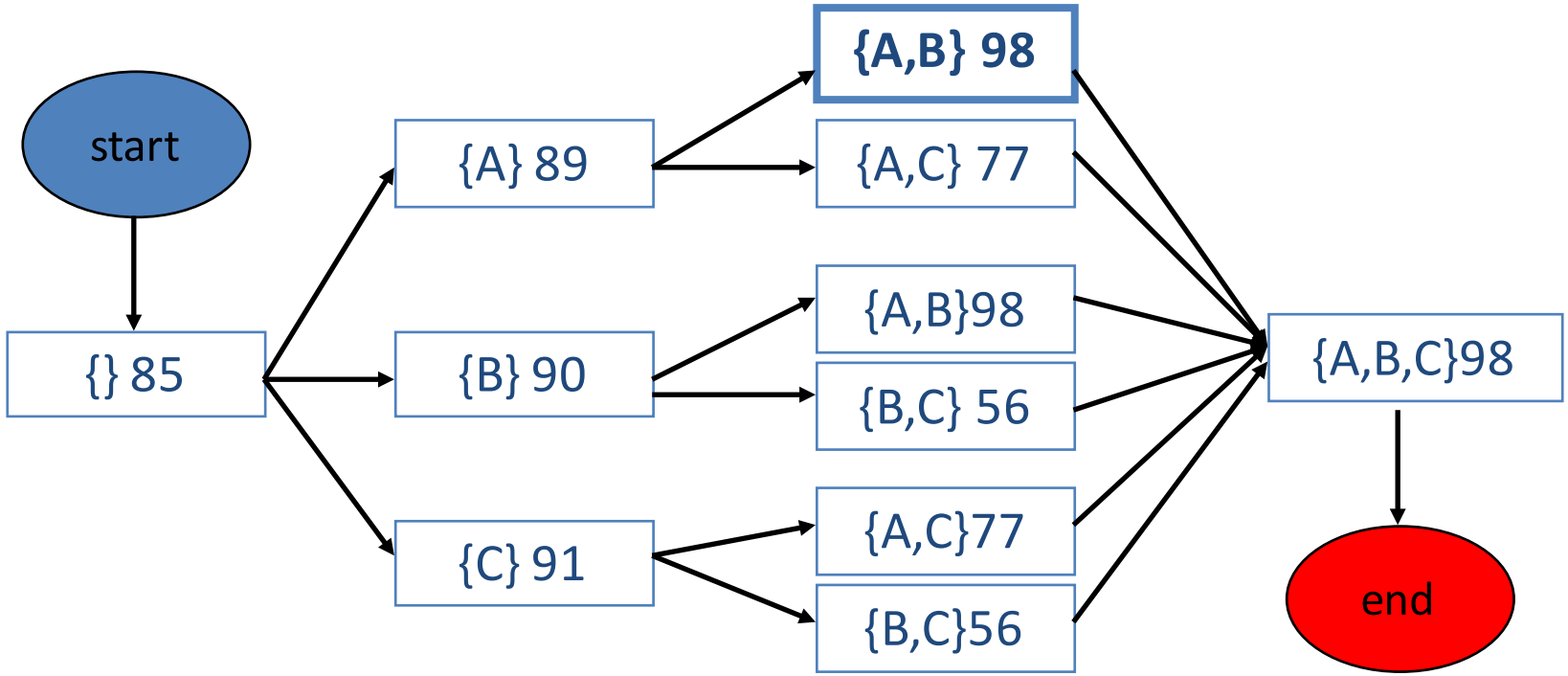
Wrappers (sequential)

The task: Say we have features A, B, C and a classifier *M.* We want to predict T (the class) given the smallest possible subset of features {A,B,C}, while achieving maximal performance (accuracy)

FEATURE SET	CLASSIFIER	PERFORMANCE
{A,B,C}	*M*	**98%**
{A,B}	*M*	**98%**
{A,C}	*M*	77%
{B,C}	*M*	56%
{A}	*M*	89%
{B}	*M*	90%
{C}	*M*	91%
{.}	*M*	85%

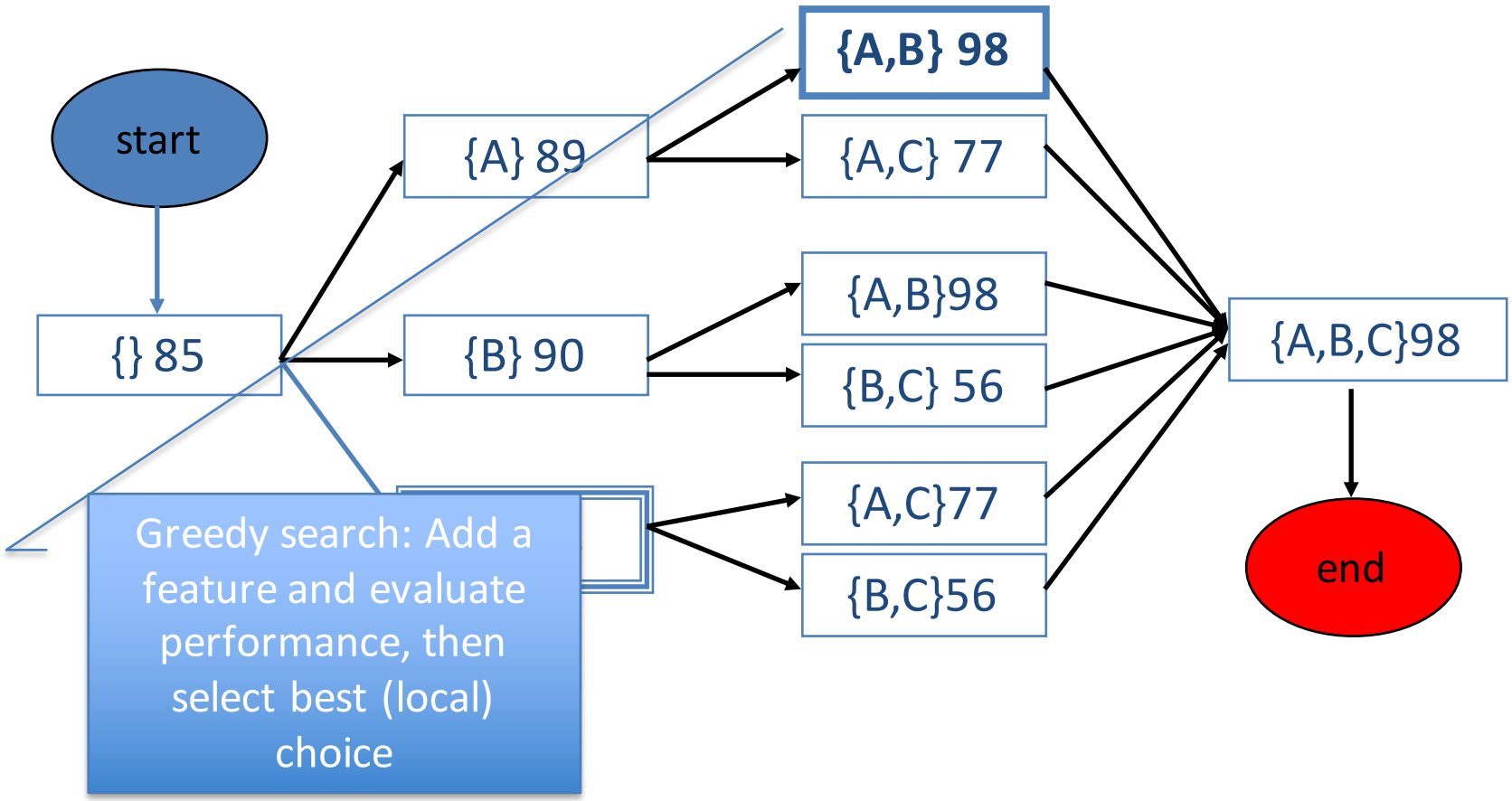
Wrappers (sequential)

The set of all subsets of features is the *power set* and its size is $2^{|V|}$. Hence for large *V* we cannot do this procedure exhaustively; instead we rely on *heuristic search* of the space of all possible feature subsets.



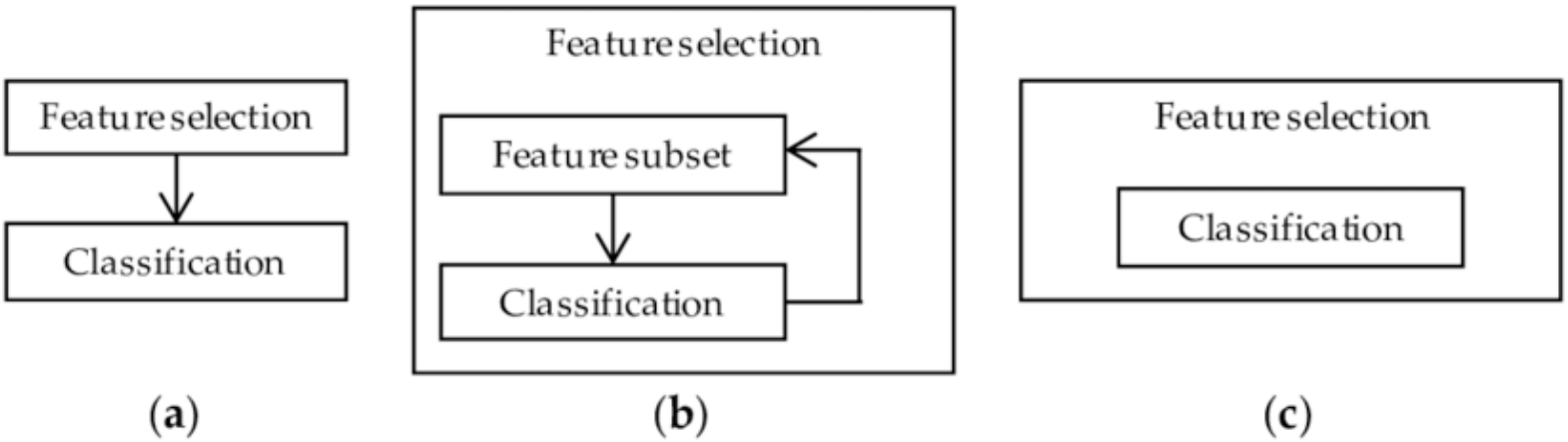
Wrappers (5)

A common example of heuristic search is hill climbing: keep adding features one at a time until no further improvement can be achieved.



3) Embedded methods

- Embedded methods perform feature selection during the modelling algorithm's execution.
- In contrast with filter (a) and wrapper (b) approaches, in embedded methods (c) the features selection part can not be separated from the learning part.
- Most embedded methods are model-dependent, they depend on the class of algorithms chosen



Embedded methods (2)

- Most common methods are (*advanced readings*)**:**
- **Perturbation-based approaches** (Lundberg and Lee, 2017: A Unified Approach to interpreting model decisions)
- **Gradient approaches** (Selvaraju et al., 2016: Grad-cam: Why say you that? Visual explantions from deep networks via gradient-based localization)