LUDWIG-
MAXIMILIANS-
UNIVERSITÄT
MÜNCHEN

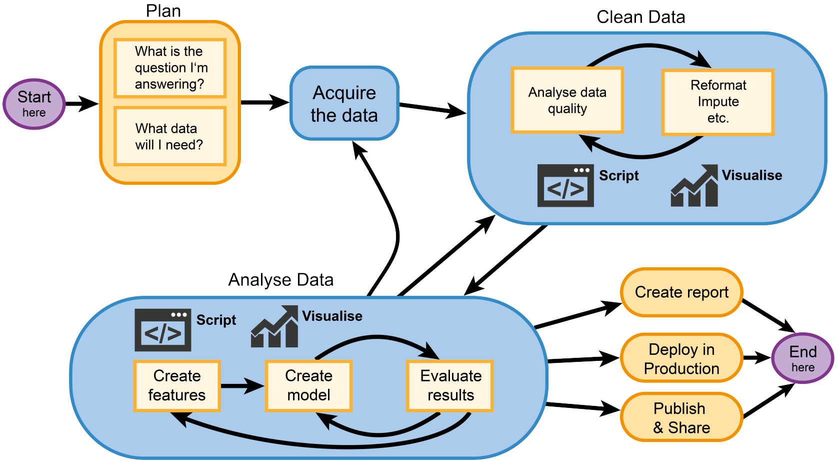
Introduction to Machine Learning

Feature Engineering and Preprocessing

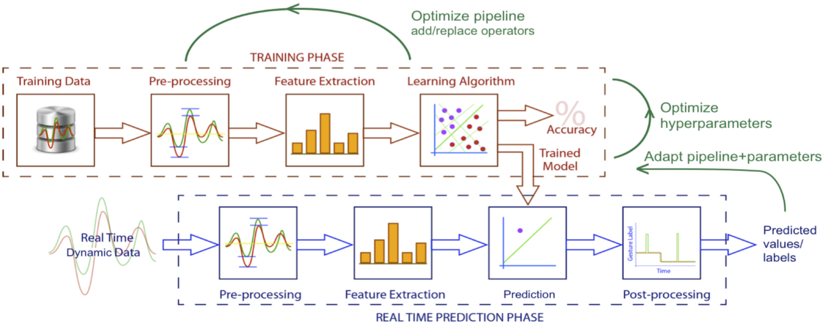
Department of Statistics – LMU Munich

Introduction

MACHINE LEARNING WORKFLOW



MACHINE LEARNING PIPELINES



Choose pipeline structure and optimize pipeline parameters w.r.t. the estimated prediction error on an independent test set, or measured by cross-validation.

IMPORTANT TYPES OF FEATURE ENGINEERING

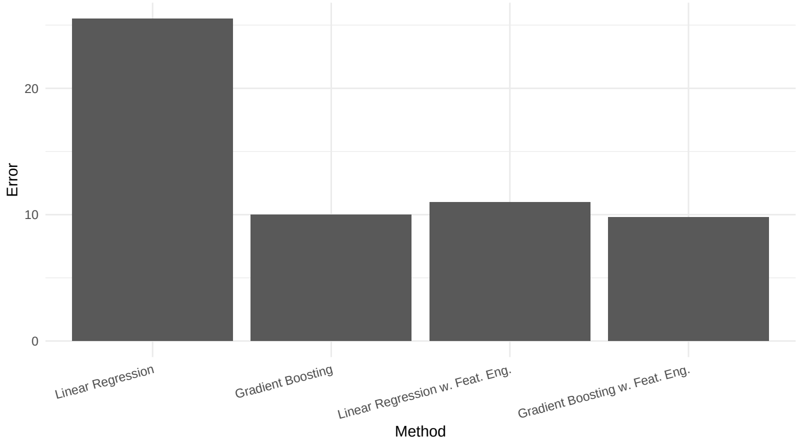
Feature engineering is on the intersection of **data cleaning**, **feature creation** and **feature selection**.

The goal is to solve common difficulties in data science projects, like

- skewed / weird feature distributions,
- (high cardinality) categorical features,
- functional (temporal) features,
- missing observations,
- high dimensional data,
- ...

and to **improve model performance**.

WHY FEATURE ENGINEERING IS IMPORTANT



Choice between a simple **interpretable** model with feature engineering or a complex model without.

FEATURE ENGINEERING AND DEEP LEARNING

One frequent argument for deep learning is the idea of “automatic feature engineering”, i.e., that no further preprocessing steps are necessary.

This is mainly true for special types of data like

- Images
- Texts
- Curves/Sequences

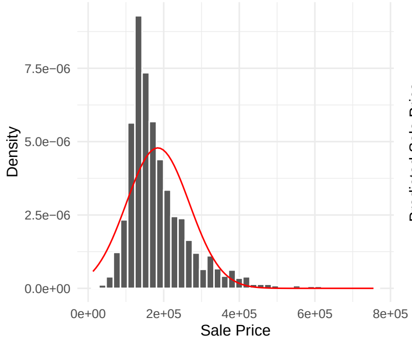
Many feature engineering problems for regular **tabular** data are not solved by deep learning.

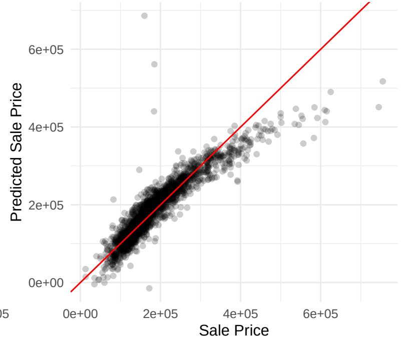
Furthermore, choosing the architecture and learning hyperparameters poses its own new challenges.

Feature/Target Transformation

TARGET TRANSFORMATION

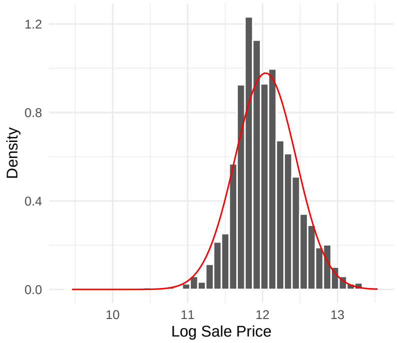
Sometimes using the raw target or raw features is not enough to build an adequate model. For example, the linear model requires a normally distributed target variable. But the house prices do not seems to be normally distributed. A linear model trained on that target overestimates the target variable:

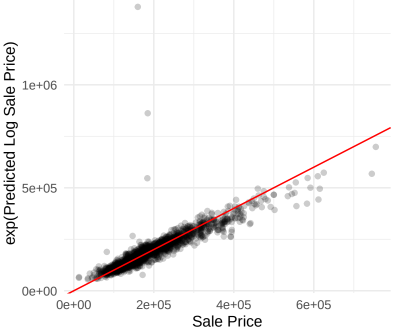




TARGET TRANSFORMATION

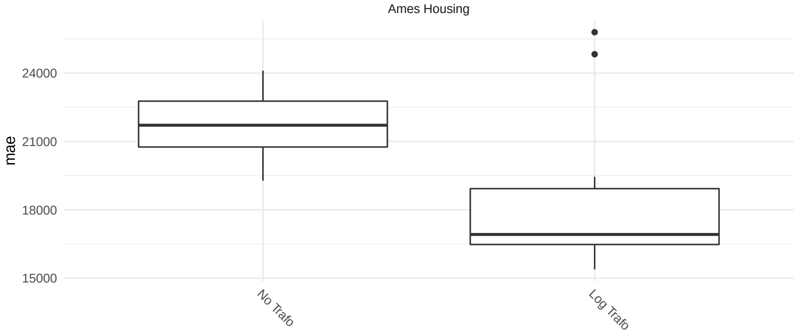
A common trick for skewed distributions is to model the log-transformation:





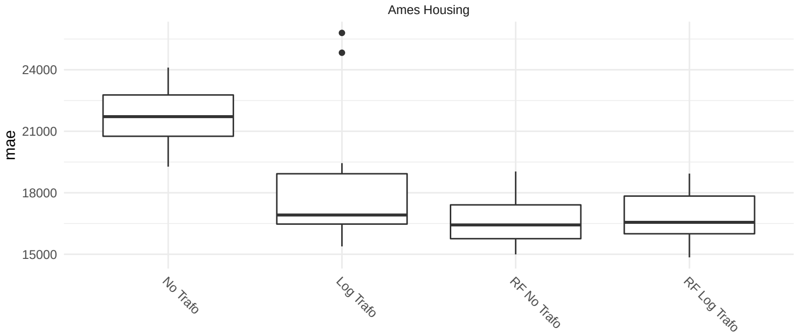
TARGET TRANSFORMATION

Benchmarking the logarithmic transformation against the raw data yields a significant improvement of the mean absolute error:



TARGET TRANSFORMATION

Nevertheless, there are also methods that are able to deal with skewed data:



FEATURE TRANSFORMATIONS

- **Normalization**: The feature is transformed to have a mean of 0 and standard deviation of 1

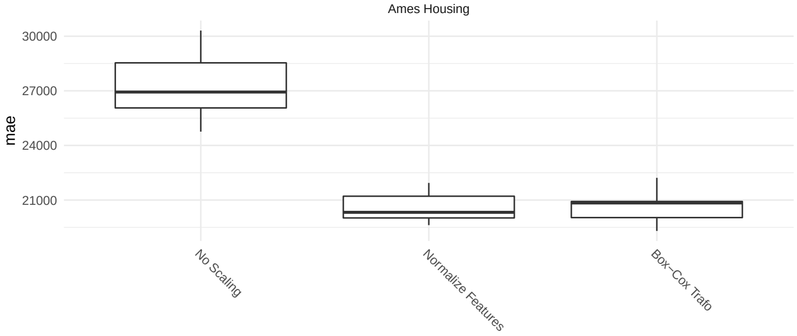
$$z_j^{(i)} = \frac{x_j^{(i)} - \text{mean}(x_j)}{\text{sd}(x_j)}$$

- **Box-Cox Transformation**: Stabilizes variance, makes the data more normal distribution-like

$$z_j^{(i)} = \begin{cases} \frac{\left(x_j^{(i)}\right)^{\lambda} - 1}{\lambda} & \text{if } \lambda \neq 0 \\ \log(x_j^{(i)}) & \text{if } \lambda = 0 \end{cases}$$

FEATURE TRANSFORMATIONS

To illustrate the effect of transforming the features we evaluate a k-NN learner without scaling, with normalization, and with a Box-Cox transformation:

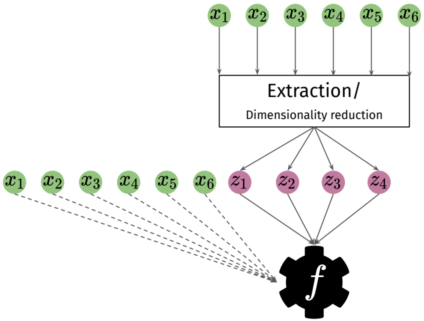


OTHER COMMON TRANSFORMATIONS

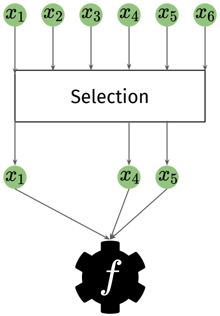
- Polynomials: $x_j \longrightarrow x_j, x_j^2, x_j^3, ...$
- Interactions: $x_j, x_k \longrightarrow x_j, x_k, x_j \times x_k$
- Basis expansions: BSplines, TPB, ...

These transformations are used to improve simple models, e.g. linear regression, and most likely will **not** improve complex machine learning models.

FEATURE EXTRACTION VS. SELECTION



x_1 x_2 x_3 x_4 x_5 x_6
Extraction/
Dimensionality reduction
x_1 x_2 x_3 x_4 x_5 x_6 z_1 z_2 z_3 z_4
f



Feature extraction / dimensionality reduction:

- PCA, ICA, autoencoder, ...

Feature selection:

- Filter, stepwise selection, model-based selection, ...

Categorical Features

CATEGORICAL FEATURES

A categorical feature is a feature with a finite number of discrete (unordered) **levels** $c_1, \ldots, c_K$

- Categorical features are very common in practical applications.
- Except for few machine learning algorithms like tree-based methods, categorical features have to be encoded in a preprocessing step.

Encoding is the creation of a fully numeric representation of a categorical feature.

- Choosing the optimal encoding can be a challenge, especially when the number of levels k becomes very large.

ONE-HOT ENCODING

- Convert each categorical feature to K binary $(1/0)$ features, where K is the number of unique levels.
- One-hot encoding does not lose any information contained in the feature; many models can correctly handle binary features.
- Given a categorical feature x_j with levels $c_1, \dots, c_K$, the new features are

$$\tilde{x}_{j,k} = \mathbb{1}_{[x_j = c_k]} \quad \text{for } k \in \{1, 2, ..., K\},$$

$$\text{with} \quad \mathbb{1}_{[x_j = c_k]} = \begin{cases} 1 & \text{if } x_j = c_k \\ 0 & \text{otherwise} \end{cases}.$$

One-hot encoding is often the **go-to** choice for encoding of categorial features!

ONE-HOT ENCODING: EXAMPLE

Original slice of the dataset:

SalePrice	Central.Air	Bldg.Type
189900	Y	1Fam
195500	Y	1Fam
213500	Y	TwnhsE
191500	Y	TwnhsE
236500	Y	TwnhsE

One-hot encoded:

SalePrice	Central.Air.N	Central.Air.Y	Bldg.Type.1Fam	Bldg.Type.2fmCon	Bldg.Type.Duplex	Bldg.Type.Twnhs	Bldg.Type.TwnhsE
189900	0	1	1	0	0	0	0
195500	0	1	1	0	0	0	0
213500	0	1	0	0	0	0	1
191500	0	1	0	0	0	0	1
236500	0	1	0	0	0	0	1

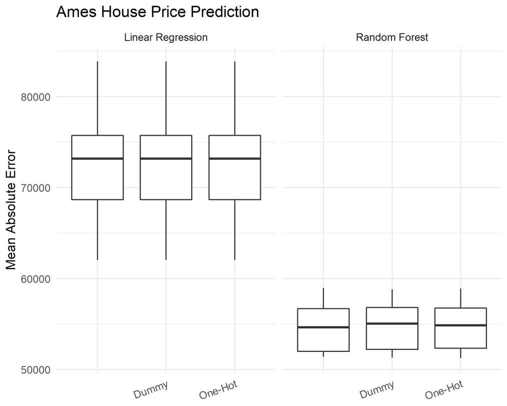
DUMMY ENCODING

- Dummy encoding is very similar to one-hot encoding with the difference that only $K - 1$ binary features are created.
- A **reference** category is chosen that has as all binary features set to 0, i.e.,

$$\tilde{x}_{j,1} = 0, \dots, \tilde{x}_{j,K-1} = 0.$$

- Each feature $\tilde{x}_{j,1}$ represents the **deviation** from the reference category.
- While using a reference category is required for stability and interpretability in statistical models like (generalized) linear models, it is not necessary, rarely done in ML and can even have negative influence on performance.

AMES HOUSING - ENCODING



Result of linear model depends on actual implementation, e.g., R’s ‘lm()‘ produces a **rank-deficient fit** warning and recovers by dropping the intercept.

ONE-HOT ENCODING: LIMITATIONS

- One-hot encoding can become extremely inefficient when the number of levels becomes too large, because one additional feature is introduced for every level.
- Assume a categorical feature with $K = 4000$ levels. When using dummy encoding, 4000 new features are added to the dataset.
- These additional features are very sparse.
- Handling such **high-cardinality categorical features** is a challenge. Possible solutions are
  - specialized methods such as **factorization machines**,
  - **target/impact encoding**,
  - clustering feature levels or
  - feature hashing.

TARGET ENCODING

- Developed to solve limitations of dummy encoding for high cardinality categorical features.
- **Goal**: Each categorical feature $\mathbf{x}$ should be encoded in a single numeric feature $\tilde{\mathbf{x}}$.

- Basic definition for regression by Micci-Barreca (2001):

$$\tilde{\mathbf{x}} = \frac{\sum_{i:\mathbf{x}=k} y^{(i)}}{n_k}, \quad k = 1, \ldots, K,$$

  where n_k is the number of observations of the k'th level of feature $\mathbf{x}$.

TARGET ENCODING - EXAMPLE

Foundation	BrkTil	CBlock	PConc	Slab	Stone	Wood
nk	311	1244	1310	49	11	5

- Encoding for wooden foundation:

house.id	17	893	986	2898	2899
SalePrice	164000	145500	143000	250000	202000
Foundation	Wood	Wood	Wood	Wood	Wood

$$\frac{164000 + 145500 + 143000 + 250000 + 202000}{5} = 180900$$

TARGET ENCODING - EXAMPLE

- For all foundation types:

Foundation	BrkTil	CBlock	PConc	Slab	Stone	Wood
Foundation(enc)	128107	148284	227069	110458	149787	180900

This mapping is calculated on training data and later applied to test data.

TARGET ENCODING FOR CLASSIFICATION

- Extending encoding to binary classification is straightforward, instead of the average target value the relative frequency of the positive class is used.
- Multi-class classification extends this by creating one feature for each target class in the same way as binary classification.

TARGET ENCODING - ISSUES

Problem: Target encoding can assign extreme values to rarely occurring levels.

Solution: Encoding as weighted sum between global average target value and encoding value of level.

$$\tilde{\mathbf{x}} = \lambda_k \frac{\sum_{i:\mathbf{x}=k} y^{(i)}}{n_k} + (1 - \lambda_k) \frac{\sum_{i=1}^{n} y^{(i)}}{n}, \quad k = 1, \dots, K.$$

- λ_k can be parameterized and tuned, but tuning should optimally be done for each feature and level separately (most likely infeasible!).
- Simple solution: Set $\lambda_k = \frac{n_k}{n_k + \epsilon}$ with regularization parameter ϵ.
- This shrinks small levels stronger to the global mean target value than large classes.

TARGET ENCODING - ISSUES

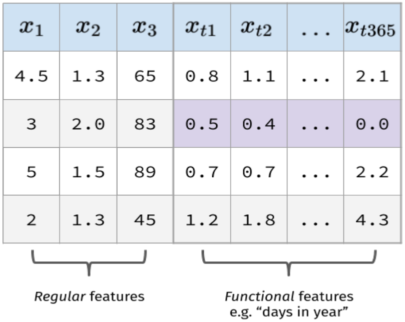
Problem: Label leakage! Information of $y^{(i)}$ is used to calculate $\tilde{\mathbf{x}}$. This can cause overfitting issues, especially for rarely occurring classes.

Solution: Use internal cross-validation to calculate $\tilde{\mathbf{x}}$.

- It is unclear how serious this problem is in practice.
- But: calculation of $\tilde{\mathbf{x}}$ is very cheap, so it doesn't hurt.
- An alternative is to add some noise $\tilde{x}_j^{(n)} + N(0, \sigma_\epsilon)$ to the encoded samples.

Functional Features

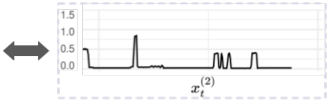
WHAT IS FUNCTIONAL DATA



x_1	x_2	x_3	x_{t1}	x_{t2}	...	x_{t365}
4.5	1.3	65	0.8	1.1	...	2.1
3	2.0	83	0.5	0.4	...	0.0
5	1.5	89	0.7	0.7	...	2.2
2	1.3	45	1.2	1.8	...	4.3

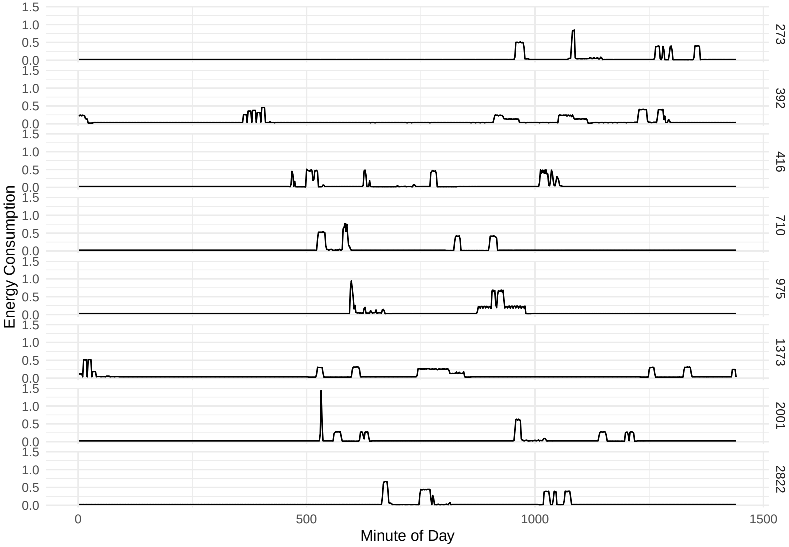
Regular features

Functional features
e.g. “days in year”



Functional or sequence data has a (temporal) order between (some) features.

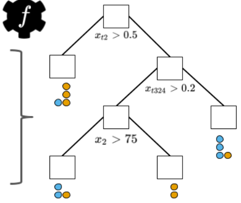
FUNCTIONAL DATA - EXAMPLE: ENERGY USAGE



HANDLING FUNCTIONAL FEATURES - 3 WAYS

1. Ignore the structure and let the model figure it out.

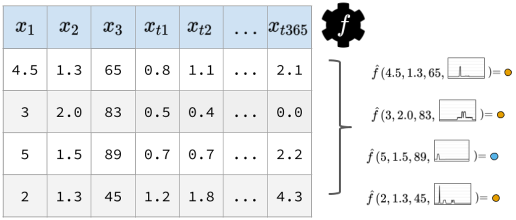
x_1	x_2	x_3	x_{t1}	x_{t2}	...	x_{t365}
4.5	1.3	65	0.8	1.1	...	2.1
3	2.0	83	0.5	0.4	...	0.0
5	1.5	89	0.7	0.7	...	2.2
2	1.3	45	1.2	1.8	...	4.3



- Can be quite difficult for the model to implicitly learn and utilize the structure.
- Requires complex models that can model high-level interactions between features, e.g., (boosted) trees.
- No **translation invariance**, i.e., structure has to be learned for every position in the timeseries seperately.

HANDLING FUNCTIONAL FEATURES - 3 WAYS

2. Use specialized machine learning algorithms that can handle functional features.



x_1	x_2	x_3	x_{t1}	x_{t2}	...	x_{t365}
4.5	1.3	65	0.8	1.1	...	2.1
3	2.0	83	0.5	0.4	...	0.0
5	1.5	89	0.7	0.7	...	2.2
2	1.3	45	1.2	1.8	...	4.3

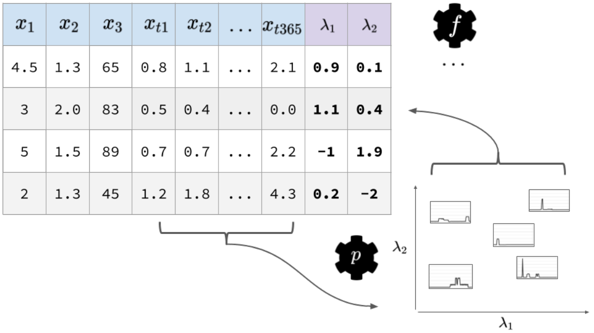
f
$\hat{f}(4.5, 1.3, 65,$ $)=$
$\hat{f}(3, 2.0, 83,$ $)=$
$\hat{f}(5, 1.5, 89,$ $)=$
$\hat{f}(2, 1.3, 45,$ $)=$

- Extensions of existing algorithms, e.g., functional k-nearest neighbor using **dynamic time warping** as distance metric.
- **Theoretical** correct/optimal way to handle functional features.
- **But:** Less flexibility in model choice and (oftentimes) less efficient implementations.

Note: Deep neural networks (both CNNs and RNNs) can be used for functional data. But depending on the overall data structure not always the best choice.

HANDLING FUNCTIONAL FEATURES - 3 WAYS

3. Use feature engineering to extract information from functional structure.



x_1	x_2	x_3	x_{t1}	x_{t2}	...	x_{t365}	λ_1	λ_2
4.5	1.3	65	0.8	1.1	...	2.1	**0.9**	**0.1**
3	2.0	83	0.5	0.4	...	0.0	**1.1**	**0.4**
5	1.5	89	0.7	0.7	...	2.2	**-1**	**1.9**
2	1.3	45	1.2	1.8	...	4.3	**0.2**	**-2**

- Large number of possible **extractors**, both simple and complex.
- Allows application of any standard machine learning algorithms.
- Thus, very flexible approach.

FEATURE EXTRACTION FOR FUNCTIONAL DATA

Simple descriptive statistics:

- min, max, mean, variance, ...

More complex transformations:

- Fourier Transformation
- Functional Principal Components
- Wavelets
- Spline Coefficients
- ...

Problem: Often unclear which of these techniques to use.

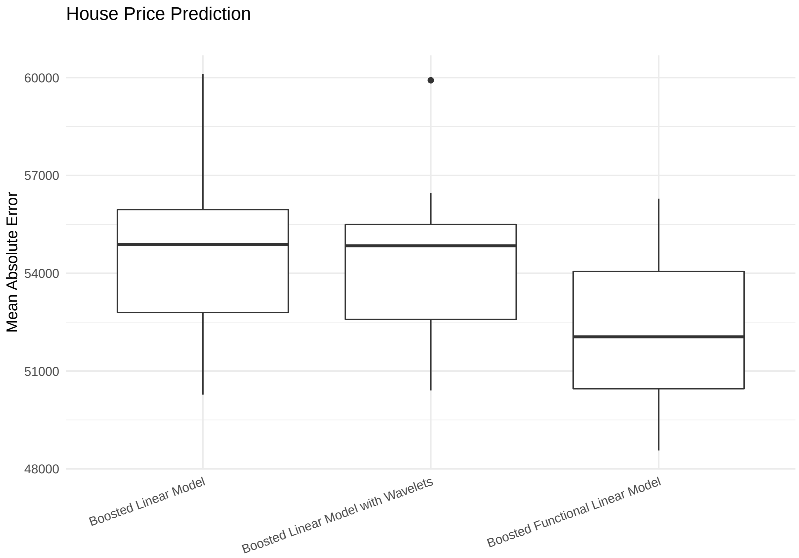
Solution: Extract a large number of features and use feature selection methods, or include extraction strategies in pipeline definition and use tuning strategies (can get quite expensive!)

EXAMPLE - AMES HOUSING

house.id	mean.energy	var.energy	max.energy	...
2463	0.053	0.008	0.509	...
2511	0.048	0.011	0.849	...
2227	0.071	0.007	0.459	...
526	0.047	0.009	0.775	...
195	0.070	0.010	0.805	...

- Some features are easily interpretable and domain knowledge can help to define meaningful extractions.
- More complex features, e.g. wavelets, allow to capture more complex structures, but are not interpretable anymore.

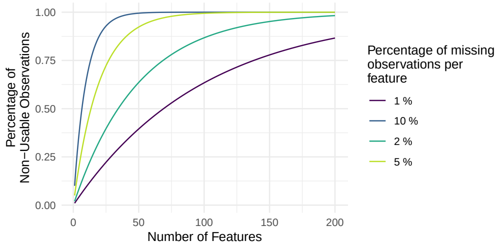
COMPARISON


House Price Prediction
Mean Absolute Error
60000
57000
54000
51000
48000
Boosted Linear Model
Boosted Linear Model with Wavelets
Boosted Functional Linear Model

Imputation

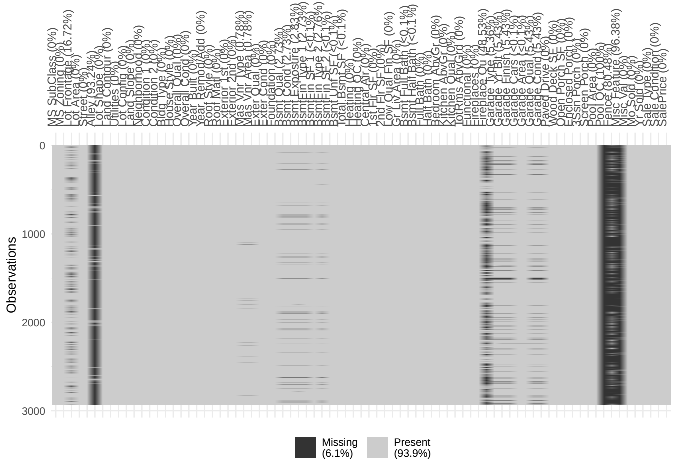
MOTIVATING EXAMPLE

- Assume each feature in your dataset has 2 % missing values.
- The missing values are randomly distributed over the observations.
- How many rows can be used if all observations that contain at least a missing value is dropped?



With 100 features and 2 % missing values, only 13 % of our data can be used.

VISUALIZING MISSING VALUES

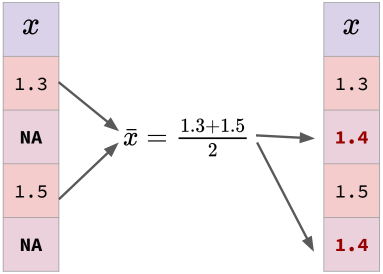


VISUALIZING MISSING VALUES

- Remove observations that contain missing values.
  But: Could lead to a very small dataset.
- Remove features that contain mostly missing values.
  But: Can lose (important) information.
- Use models that can handle missing values, e.g., (most) tree-based methods
  But: Restriction in model choice.
- **Imputation**
  $\rightarrow$ Replace missing values with **plausible** values.

SIMPLE IMPUTATION METHODS

A very simple imputation strategy is to replace missing values with univariate statistics of the feature, e.g. mean or median:



SIMPLE IMPUTATION METHODS

The statistic used to impute the missing values has to match the type of the feature:

- Numeric features: mean, median, quantiles, mode, ...
- Categorical features: mode, ...

Alternatively, missing values can be encoded with new values

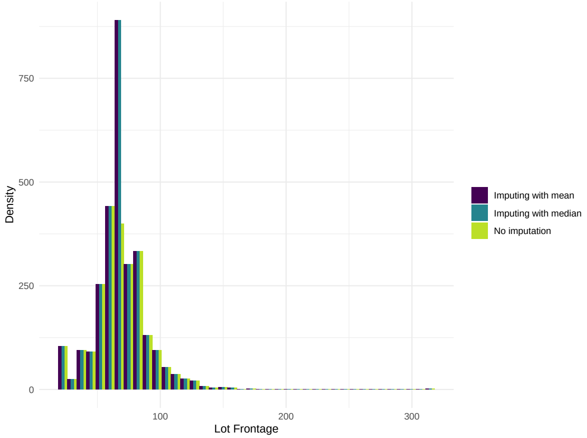
- Numeric features: 2*max, ...
- Categorical features: MISS, ...

Note: This is especially useful for tree-based methods, as it allows to separate observations with missing values in a feature.

Note: Encoding numeric values **out-of-range** for models estimation global feature effects is usually a very bad idea.

DISADVANTAGE OF CONSTANT IMPUTATION

By imputing a feature with one value we shift the distribution of that feature towards a single value.



IMPUTATION BY SAMPLING

A way out of this problem is to sample values to replace each missing observation from

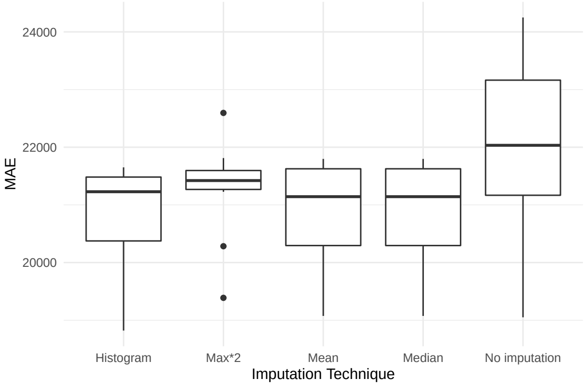
- the empirical distribution or histogram for a numeric feature.
- the relative frequencies of levels for a categorical feature.

This ensures that the distribution of the features does not change much.

To ensure that the information about which values are missing is not lost, it is important to add binary indicator features.

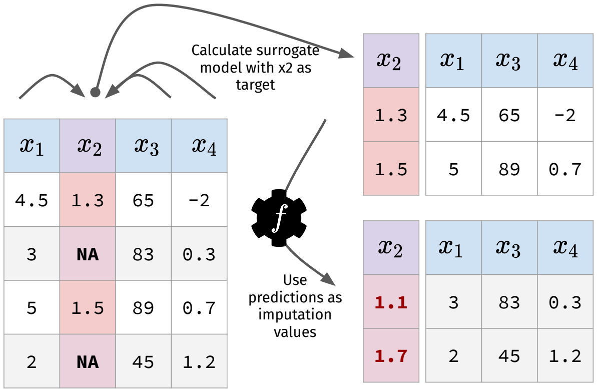
BENCHMARK OF SIMPLE IMPUTATION

To illustrate the effect of imputation on the performance we evaluate a linear model on the Ames housing dataset. Evaluation is done with a 10-fold cross-validation:



MODEL-BASED IMPUTATION

Instead of imputing a single value or sampling values it is desirable to take advantage of structure and correlation between features.


Calculate surrogate model with x2 as target
Use predictions as imputation values
f

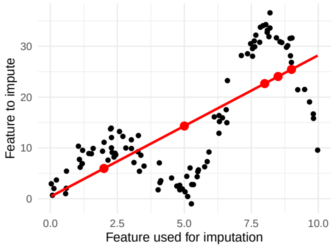
x_1	x_2	x_3	x_4
4.5	1.3	65	-2
3	**NA**	83	0.3
5	1.5	89	0.7
2	**NA**	45	1.2

x_2	x_1	x_3	x_4
1.3	4.5	65	-2
1.5	5	89	0.7

x_2	x_1	x_3	x_4
1.1	3	83	0.3
1.7	2	45	1.2

MODEL-BASED IMPUTATION: DRAWBACKS

- Choice of surrogate model has high influence on imputed values:

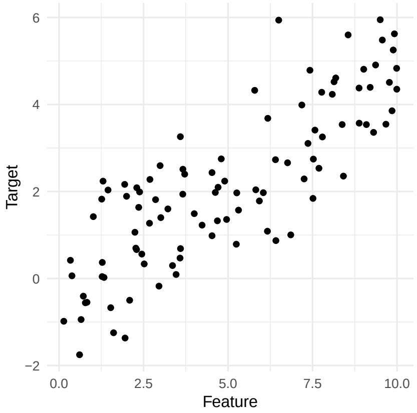


- Surrogate model should be able to handle missing values itself, otherwise imputation **loop** may be necessary.
- Surrogate model hyperparameter can be tuned and may be different for each feature to impute.

Outliers

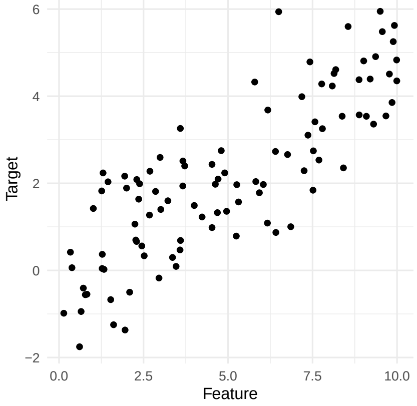
EXAMPLE LINEAR MODEL

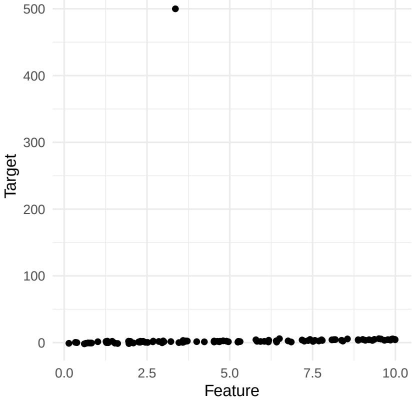
The following data has a clear linear dependency:



EXAMPLE LINEAR MODEL

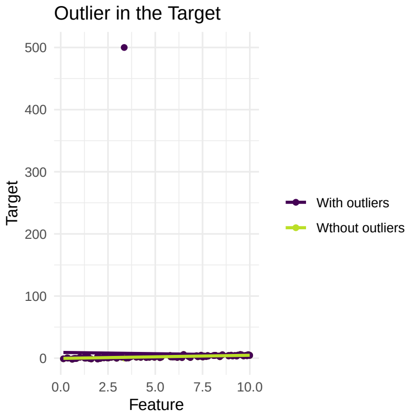
Adding a single outlier does not change the linear dependency, there is just one wrong value:

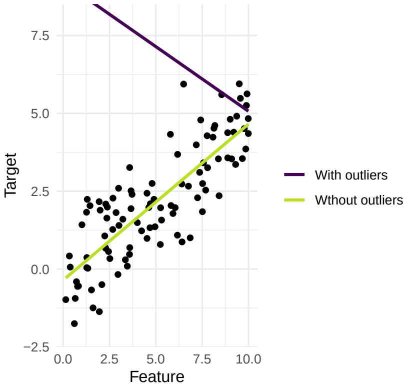




But how does this single value affect a model trained on that data?

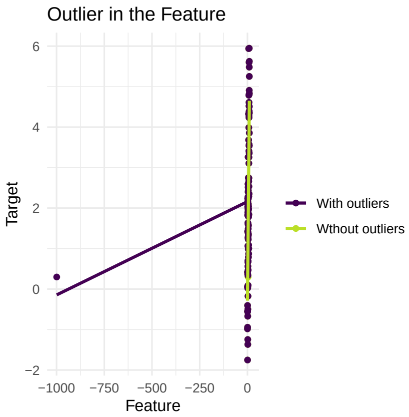
EXAMPLE LINEAR MODEL: OUTLIER IN TARGET

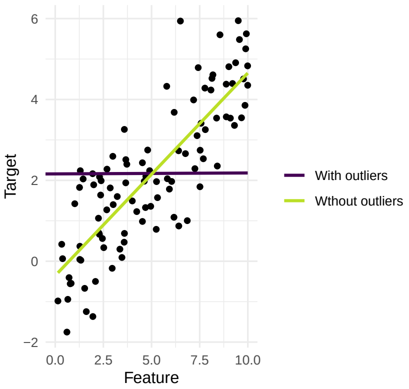




EXAMPLE LINEAR MODEL: OUTLIER IN FEATURE

One observations with a feature value of -999 (could be a wrongly coded missing value).





SOLUTIONS

- Make the model less **sensitive** regarding outliers
  But: Errors still need to be measured properly.

- Remove observations containing outliers
  But: How to detect these outliers?

TEACH A MODEL TO HANDLE OUTLIERS

Most machine learning models are trained by minimizing a loss function $L(y, f(\mathbf{x}))$.

For example, a linear model is trained by minimizing quadratic errors, i.e., L2-loss:

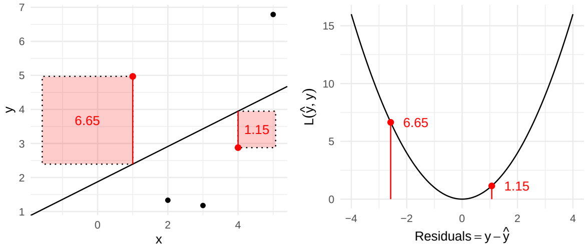
$$\hat{\theta} = \arg\min_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right) = \arg\min_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} \left(y^{(i)} - \boldsymbol{\theta}^\top \mathbf{x}^{(i)}\right)^2$$

QUADRATIC LOSS

The **Mean Squared Error** (L2-Loss) averages the squared distances between the target variable y and the predicted target $f(\mathbf{x})$.

$$MSE = \frac{1}{n} \sum_{i=1}^{n} \left(y^{(i)} - f\left(\mathbf{x}^{(i)}\right)\right)^2$$

Observations with large residuals heavily influence the MSE:

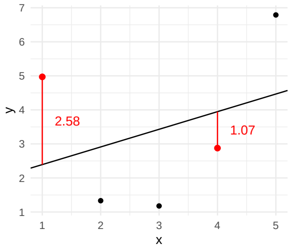


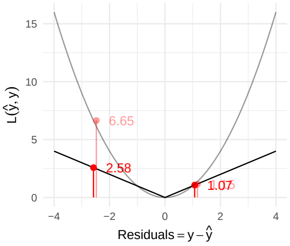
ABSOLUTE LOSS

An alternative is to optimize the **Mean Absolute Error (L1-Loss):**

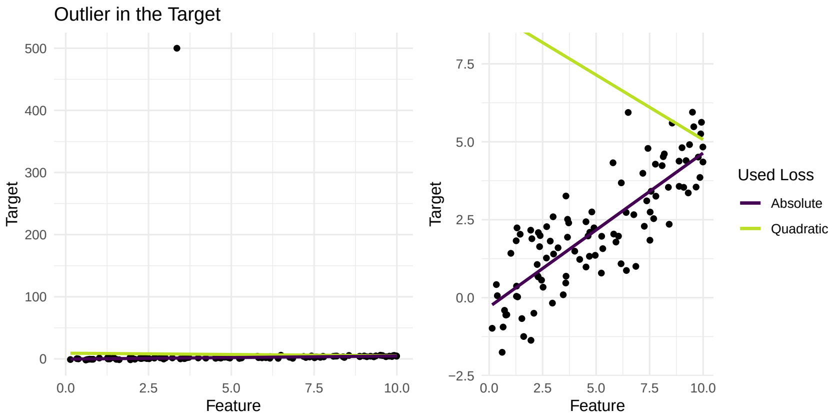
$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|y^{(i)} - f\left(\mathbf{x}^{(i)}\right)\right|$$

Observations with large errors do not heavily influence the MAE that much:



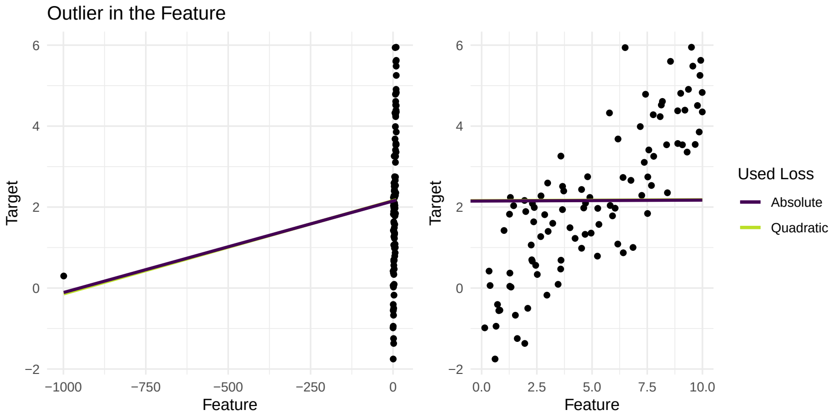


EXAMPLE LINEAR MODEL: TARGET OUTLIER



The model becomes more **robust** in regards to the outlier.

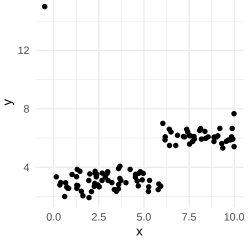
EXAMPLE LINEAR MODEL: FEATURE OUTLIER



But: Using a robust loss function is not always sufficient!

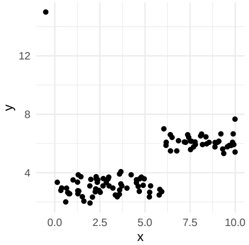
TREE-BASED-MODELS

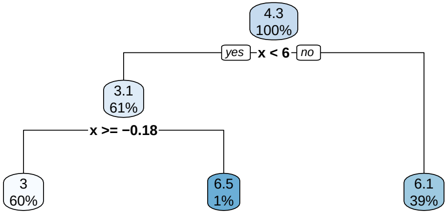
Trees are able to isolate outliers in separate terminal nodes:



TREE-BASED-MODELS

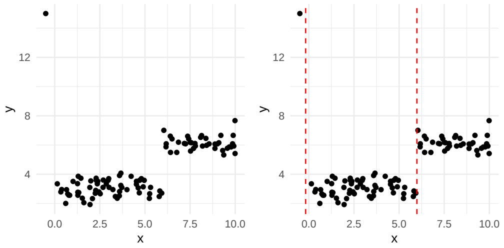
Trees are able to isolate outliers in separate terminal nodes:

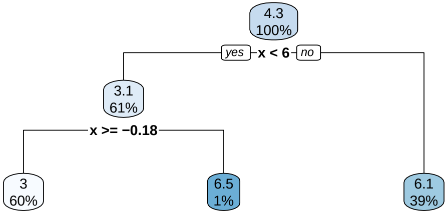




TREE-BASED-MODELS

Trees are able to isolate outliers in separate terminal nodes:





OUTLIER DETECTION

A different way to handle outliers is to remove them completely. But they must be detected first.

- Different values than most observations, but when is it **different enough** to conclude it is an outlier. **Outlier or extreme value?**
- Even when all single feature values are **normal**, an observation still can be an outlier over multiple features.
- Difference between wrong value, e.g., missing encoding error, or really occurring outlier.

OUTLIER DETECTION

First very simple approach: Remove observations if they are too big or too small.

- Construct a lower bound l and upper bound u to indicate which values are outliers.
- Remove values that are outside the interval $[l, u]$.
- $[l, u]$ can be defined by domain knowledge or by looking at empirical distributions of features.

Z-SCORE

Assume a feature is normally distributed. The transformation

$$z_j^{(i)} = \frac{x_j^{(i)} - \bar{x}_j}{\mathrm{sd}(x_j)}$$

is standard normal distributed. It is therefore very easy to calculate how likely it is to observe a value within a given interval.

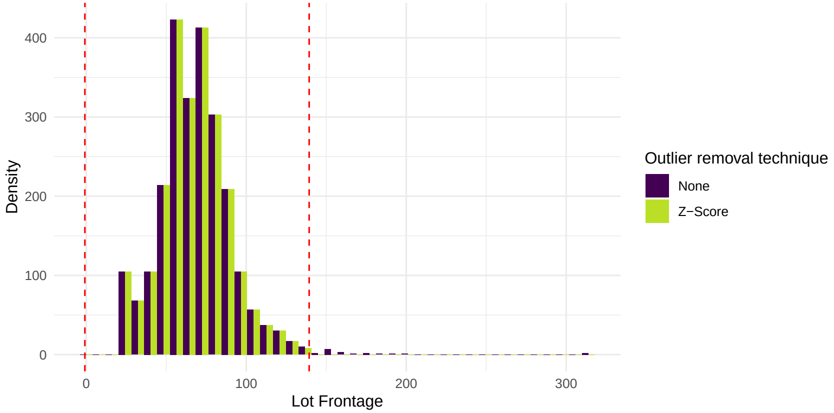
Remove observation *i* if:

$$z_j^{(i)} \notin [-3, 3] \;\text{ or }\; x_j^{(i)} \notin [-3\,\mathrm{sd}(x_j) + \bar{x}_j, 3\,\mathrm{sd}(x_j) + \bar{x}_j]$$

The probability of such an observation is 99.7 %, and it is therefore very unlikely to observe a value outside of $[-3, 3]$.

Problem: Features are often not normally distributed.

Z-SCORE



Density
400
300
200
100
0
0
100
200
300
Lot Frontage
Outlier removal technique
None
Z−Score

REMARKS

Advantages

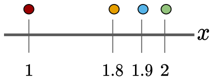
- Very intuitive and easy to use.
- Implementation is very fast.

Disadvantages

- Only takes single features into account.
- Unclear how to choose ranges ($[l, u], ...$).

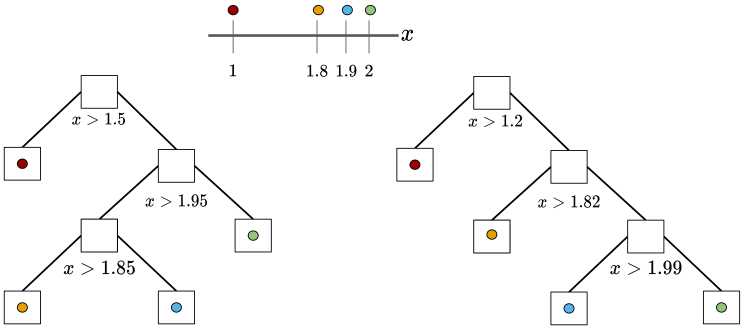
ISOLATION FOREST

- The isolation forest randomly creates splits by sampling from the range of a random feature until we have nodes with just one observation.
- Outliers are more likely to be separated since it is more likely to choose a split point between the outlier and its closest neighbor.
- For example, the probability of splitting the red point is 80 %:



- Therefore, the more distant a point is, the sooner it is separated in a terminal node.

ISOLATION FOREST



path length(🔴) = 1	path length(🟠 🔵 🟢) = 8/3	path length(🔴) = 1	path length(🟠 🔵 🟢) = 8/3
path length(🟠) = 3	path length(🔴 🔵 🟢) = 6/3	path length(🟠) = 2	path length(🔴 🔵 🟢) = 7/3
path length(🔵) = 3	path length(🔴 🟠 🟢) = 6/3	path length(🔵) = 3	path length(🔴 🟠 🟢) = 6/3
path length(🟢) = 2	path length(🔴 🟠 🔵) = 7/3	path length(🟢) = 3	path length(🔴 🟠 🔵) = 6/3

ISOLATION FOREST

$$\text{score}(\text{🔴}) = \exp_2\left(-\frac{\text{average path length}(\text{🔴})}{\text{average path length}(\text{🟡🔵🟢})}\right) \approx 0.77$$

$$\text{score}(\text{🟡}) = \exp_2\left(-\frac{\text{average path length}(\text{🟡})}{\text{average path length}(\text{🔴🔵🟢})}\right) \approx 0.45$$

$$\text{score}(\text{🔵}) = \exp_2\left(-\frac{\text{average path length}(\text{🔵})}{\text{average path length}(\text{🔴🟡🟢})}\right) \approx 0.35$$

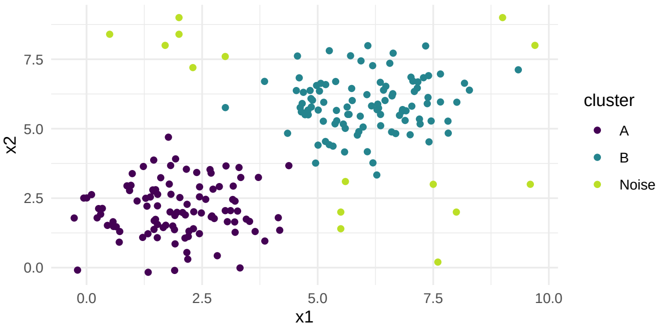
$$\text{score}(\text{🟢}) = \exp_2\left(-\frac{\text{average path length}(\text{🟢})}{\text{average path length}(\text{🔴🟡🔵})}\right) \approx 0.45$$

- Score close to 1 indicates outliers
- Score smaller than 0.5 corresponds to a typical observation
- The whole dataset seems to have just normal observations if all scores are around 0.5

DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a clustering algorithm that allows **undecided** observations that do not fit to any cluster.

These **undecided** noise observations can be understood as outliers.



Practical Feature Enginerring

FEATURE ENGINEERING IN PRACTICE

There are generally two ways of doing feature engineering in practice:

1. Manual Feature Engineering

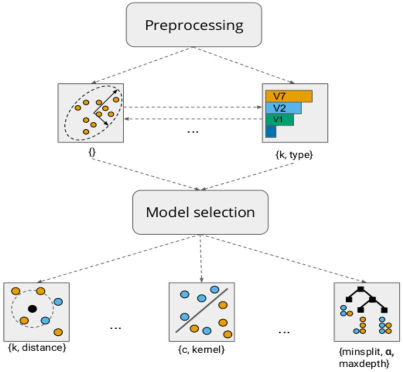
Trial and error with educated guesses what methods might be important/useful to apply to the data

2. (Semi-)Automatic Feature Engineering

Define a set of feature engineering operations and let an optimizer search for a well working pipeline

Important: It is crucial for both approaches that this process is embedded in a nested cross-validation loop.

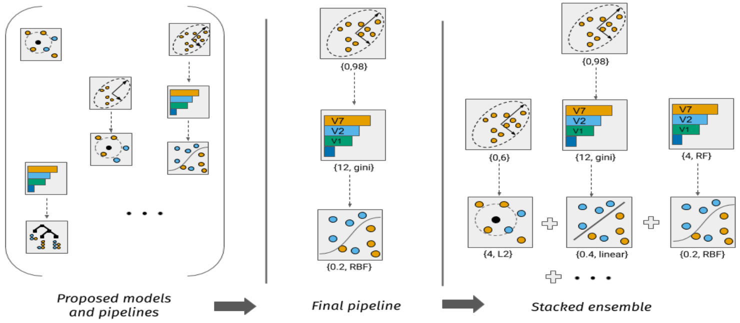
(SEMI-)AUTOMATIC FEATURE ENGINEERING



- Define a space of possible operations from the previous chapters and let an optimizer search an optimal pipeline
- If model choice and hyperparameters are included in the search, this process is called **Auto**matic **M**achine **L**earning (AutoML)

STACKING AND AUTOML

- AutoML approaches create a large number of models while searching for an optimal pipeline
- To further boost the pipeline performance, multiple pipelines can be **stacked** in a post-processing step



FEATURE ENGINEERING AND DOMAIN KNOWLEDGE

- Some forms of feature engineering are very hard to automate.
- Information that is not present in the data can not be found automatically.
- This is why we still need humans with domain knowledge to find optimal models.

DOMAIN KNOWLEDGE: EXAMPLE

A simple example can be spatial information hidden in categorical features:

Neighborhood	Mitchel	NridgHt	MeadowV	BrDale	Greens	NPkVill
n	114	166	37	30	8	23

The **Neighborhood** feature does not directly include which houses are close to each other across different neighborhoods.

Some manual preprocessing / feature engineering is still required to enrich this data with actual spatial information.

HUMAN-IN-THE-LOOP APPROACHES

With better software it becomes easier for humans to integrate such knowledge.

Examples:

- Features could be tagged as **spatial** and trigger an automatic GoogleMaps API query, adding longitude and latitude to the data.
- Feature groups can be tagged, e.g., revenues of different departments, where adding or averaging multiple features can be beneficial.

Random or exhaustive combination of features quickly becomes infeasible due to exponential growth in possible combinations with more features.