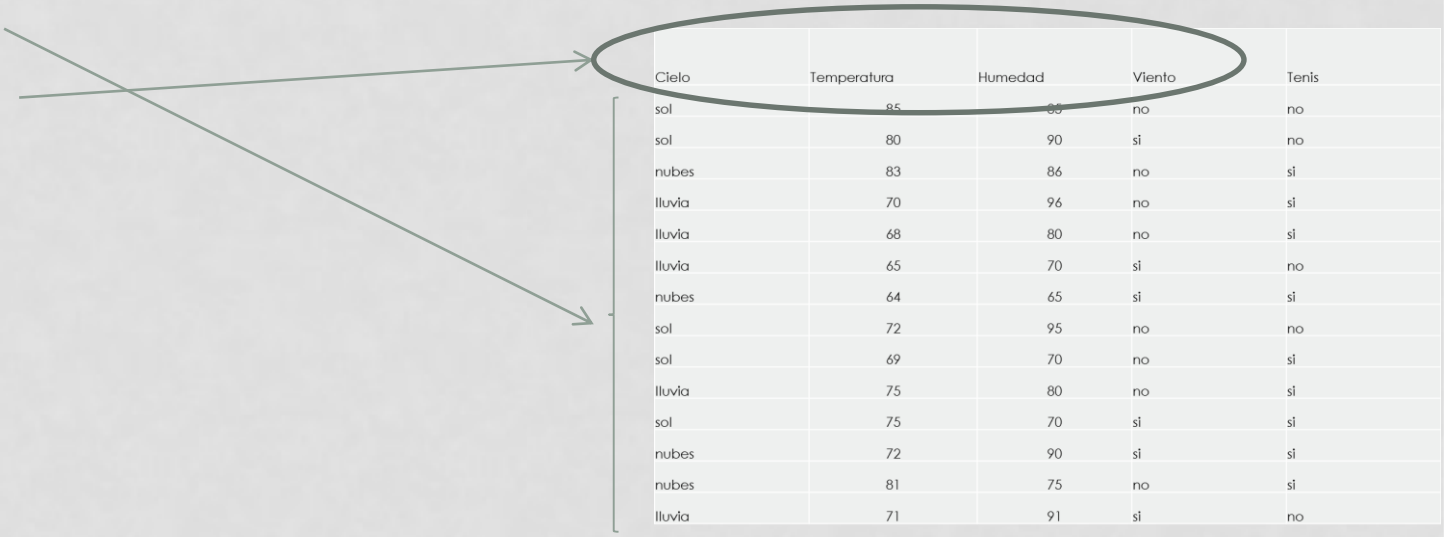
M.L. WORKFLOW / METHODOLOGY:

- Training:
  - Hyper-parameter tuning
- Model evaluation (e.g. train/test or crossvalidation)
- Final model training and using the model for making predictions (deployment)

PREPROCESSING

Transforming the data matrix

- **Preprocessing:**
  - **Instances**
  - **Attributes**

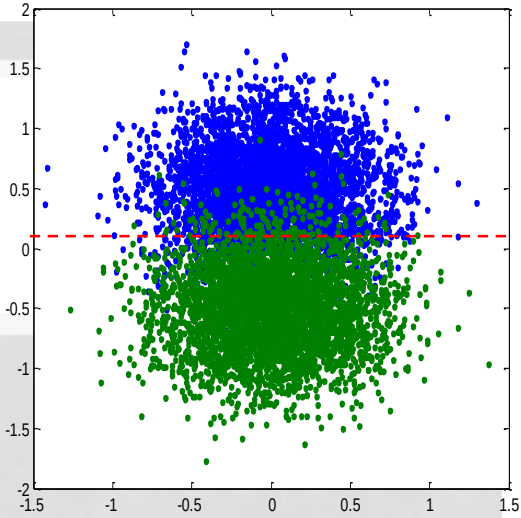


Cielo	Temperatura	Humedad	Viento	Tenis
sol	85	85	no	no
sol	80	90	si	no
nubes	83	86	no	si
lluvia	70	96	no	si
lluvia	68	80	no	si
lluvia	65	70	si	no
nubes	64	65	si	si
sol	72	95	no	no
sol	69	70	no	si
lluvia	75	80	no	si
sol	75	70	si	si
nubes	72	90	si	si
nubes	81	75	no	si
lluvia	71	91	si	no

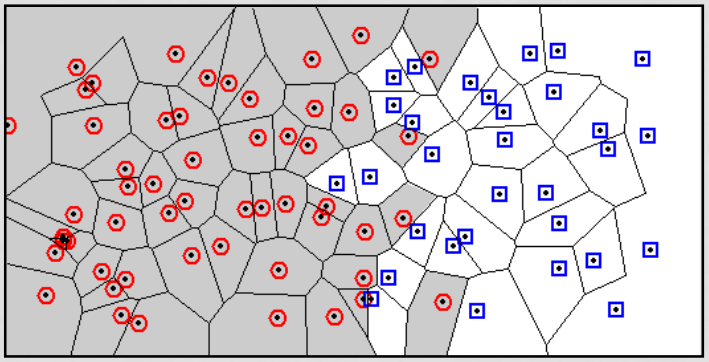
PREPROCESSING

- Instances:
  - Removing outliers
  - Removing noisy instances (**Wilson editing rule**), mainly for KNN
  - Re-sampling in order to balance classes in imbalanced problems (such as **SMOTE** – Synthetic Minority Over-sampling Technique, …) or **ADASYN**
- Attributes:
  - Scaling attributes (useful for methods based on distances: KNN, SVM, Neural networks)
  - Imputation (what to do with missing values?)
  - Categorical attribute encoding into numbers: one-hot-encoding/dummy variables, ...
  - **Attribute selection**
  - Attribute transformation (PCA, ...)
  - Feature engineering (domain dependent, and not automatic: e.g. computing wind speed out of (u,v) wind vector for wind energy prediction)

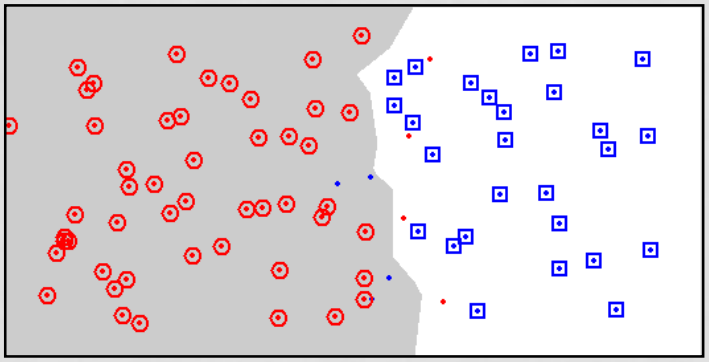
WILSON EDITING RULE: REMOVING NOISY INSTANCES FOR CLASSIFICATION PROBLEMS



- Wilson editing rule: remove instance $\mathbf{x}_i$ if it is classified incorrectly by the majority class of its k neighbours:
  - It removes noisy instances inside a class region
  - It smooths boundaries
- It works well for KNN, but can be used for other methods too



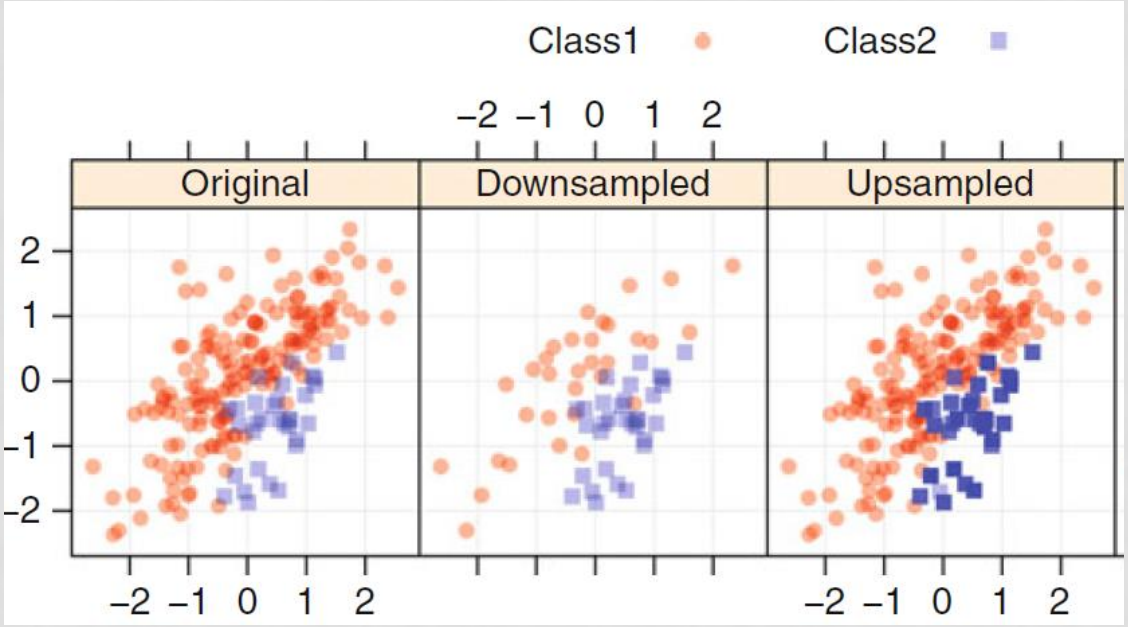
K=7



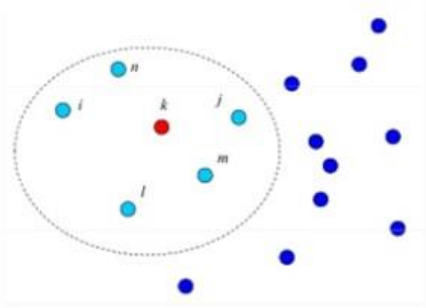
https://imblanced-learn.readthedocs.io/

imblearn.under_sampling.EditedNearestNeighbours

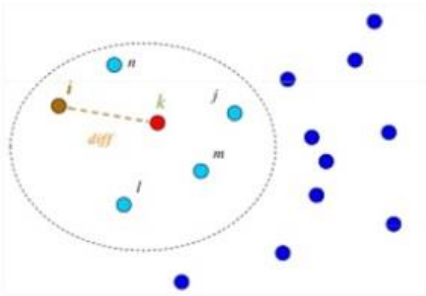
UNDERSAMPLING (DOWNSAMPLING) OVERSAMPLING (UPSAMPLING)



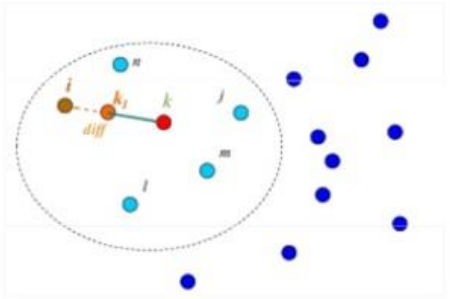
SMOTE (SYNTHETIC MINORITY OVER-SAMPLING TECHNIQUE)



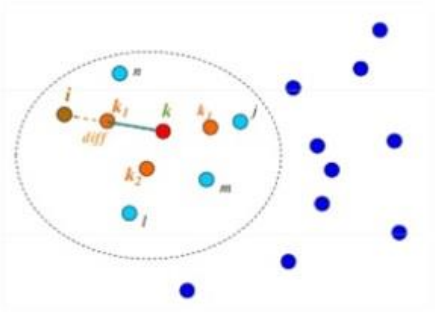
1. For each minority example k compute nearest minority class examples (i, j, l, n, m)



2. Randomly choose an example out of 5 closest points



3. Synthetically generate event k_1, such that k_1 lies between k and i



4. Dataset after applying SMOTE 3 times

The improved version of SMOTE is ADASYN

https://imbalanced-learn.readthedocs.io/

SCALING

- Different attributes may have different ranges (e.g. height: 0m-2m, weight: 0kg-100kg, …)
- The aim is that all attributes have the same range or spread
- Important for some methods such as KNN, Support Vector Machines, and neural networks. Not important for Decision trees.
- If $\mathbf{x_i}$ is an attribute / feature (i.e. a column in a data matrix)
- MinMax / range (normalization):
  - $\mathbf{x_i} = (\mathbf{x_i} - \min(\mathbf{x_i})) \,/\, (\max(\mathbf{x_i}) - \min(\mathbf{x_i}))$
  - New range = 0-1
- Standarization: $\mathbf{x_i} = (\mathbf{x_i} - \text{mean}(\mathbf{x_i})) \,/\, \text{std}(\mathbf{x_i})$

SCALING

- If $\mathbf{x_i}$ is an attribute / feature (i.e. a column in a data matrix)
- MinMax / range (normalization):
  - $\mathbf{x_i} = (\mathbf{x_i} - \min(\mathbf{x_i})) \,/\, (\max(\mathbf{x_i}) - \min(\mathbf{x_i}))$
  - New range = 0-1
- Standarization: $\mathbf{x_i} = (\mathbf{x_i} - \text{mean}(\mathbf{x_i})) \,/\, \text{std}(\mathbf{x_i})$
- Robust scaler: $\mathbf{x_i} = (\mathbf{x_i} - \text{median}(\mathbf{x_i})) \,/\, \text{IQR}(\mathbf{x_i})$
  - Scale features using statistics that are robust to outliers
  - The IQR (interquartile range) is the range between the 1st quartile (25th quantile) and the 3rd quartile (75th quantile)
  - The number of instances between the 1st quantile and the 3rd quantile is 50%

IMPUTATION

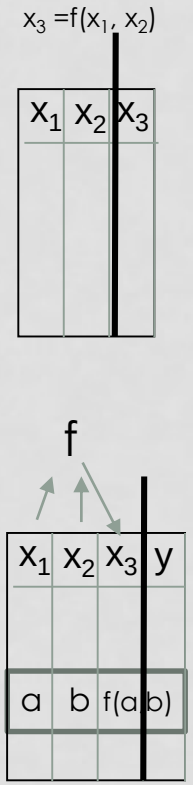
- Imputation = replacing missing values (NA's)
- Some methods are able to deal with NA's (e.g. trees), but some methods aren't (e.g. KNN, SVM, ...)
- Straightforward strategies:
  - Remove instances with NA's
  - Remove attributes with NA's
  - Problem: we are throwing away information if instances (rows) have only a few NA's or if attributes (columns) have a few NA's
    - But attributes containing a large number of NA's (805?) should be removed.
- If possible, it is convenient to understand why there are missing values. E.g.: perhaps values>10000 got a missing value? In that case, perhaps it would be better to replace the missing value by 10000 (rather tan the average)

IMPUTATION

- Automatic methods:
  - Univariate: replace NA's by the mean, median, or mode (categorical attributes):
    - *sklearn.impute.SimpleImputer*
  - Multivariate: use a machine learning method to compute models of an attribute in terms of the other attributes. Use the model to impute each attribute, in turn.
    - *sklearn.impute.IterativeImputer*

MULTIVARIANT IMPUTATION

- The missing value is replaced by the output of a model trained using the remaining attributes.
- Attributes: x_1, x_2 y x_3; and response: y.
- In order to impute x_3 model f can be used:
  - $x_3 = f(x_1, x_2)$
    - Now, x_1 and x_2 act as attributes and x_3 as response.



ENCODING CATEGORICAL VARIABLES: ONE-HOT

- Some machine learning methods are not able to deal with categorical/discrete attributes (and in sklearn, this is the most usual)
- Most commonly used: dummy variables or one-hot-encoding

	Temperature	Color	Target
0	Hot	Red	1
1	Cold	Yellow	1
2	Very Hot	Blue	1
3	Warm	Blue	0
4	Hot	Red	1
5	Warm	Yellow	0
6	Warm	Red	1
7	Hot	Yellow	0
8	Hot	Yellow	1
9	Cold	Yellow	1

	Color	Target	Temp_Cold	Temp_Hot	Temp_Very Hot	Temp_Warm
0	Red	1	0	1	0	0
1	Yellow	1	1	0	0	0
2	Blue	1	0	0	1	0
3	Blue	0	0	0	0	1
4	Red	1	0	1	0	0
5	Yellow	0	0	0	0	1
6	Red	1	0	0	0	1
7	Yellow	0	0	1	0	0
8	Yellow	1	0	1	0	0
9	Yellow	1	1	0	0	0

ENCODING CATEGORICAL VARIABLES: FREQUENCY

- However, one-hot-encoding generates too many columns for variables with many values (many categories).
- Alternatives: integer/label encoding and frequency encoding
- Problem: If the attribute is not ordinal, an arbitrary order is introduced into the new attribute. However, frequency encoding works surprisingly well

Label/integer encoding

Cold:0
Warm:1
Hot:2
Very hot:3

	Temperature	Color	Target	Temp_label_encoded
0	Hot	Red	1	2
1	Cold	Yellow	1	0
2	Very Hot	Blue	1	3
3	Warm	Blue	0	1
4	Hot	Red	1	2
5	Warm	Yellow	0	1
6	Warm	Red	1	1
7	Hot	Yellow	0	2
8	Hot	Yellow	1	2
9	Cold	Yellow	1	0

Frequency encoding: replace category by frequency of category

	Temperature	Color	Target	Temp_freq_encode
0	Hot	Red	1	0.4
1	Cold	Yellow	1	0.2
2	Very Hot	Blue	1	0.1
3	Warm	Blue	0	0.3
4	Hot	Red	1	0.4
5	Warm	Yellow	0	0.3
6	Warm	Red	1	0.3
7	Hot	Yellow	0	0.4
8	Hot	Yellow	1	0.4
9	Cold	Yellow	1	0.2

ENCODING CATEGORICAL VARIABLES: TARGET MEAN ENCODING

- Target mean encoding (AKA impact encoding), replaces category value by the average of the target variable for that category value (C) (regression) or the probability of class 1, p_c (two-class classification).
- This way, some information about the target is introduced as an input attribute.
- Of course, testing data is encoded using the encodings from the training data

C

	Temperature	Color	Target
0	Hot	Red	1
1	Cold	Yellow	1
2	Very Hot	Blue	1
3	Warm	Blue	0
4	Hot	Red	1
5	Warm	Yellow	0
6	Warm	Red	1
7	Hot	Yellow	0
8	Hot	Yellow	1
9	Cold	Yellow	1

Information

$Label_C = p_c$

	Temperature	Color	Target	Temperature_mean_enc
0	Hot	Red	1	0.750000
1	Cold	Yellow	1	1.000000
2	Very Hot	Blue	1	1.000000
3	Warm	Blue	0	0.333333
4	Hot	Red	1	0.750000
5	Warm	Yellow	0	0.333333
6	Warm	Red	1	0.333333
7	Hot	Yellow	0	0.750000
8	Hot	Yellow	1	0.750000
9	Cold	Yellow	1	1.000000

ENCODING CATEGORICAL VARIABLES: TARGET MEAN ENCODING

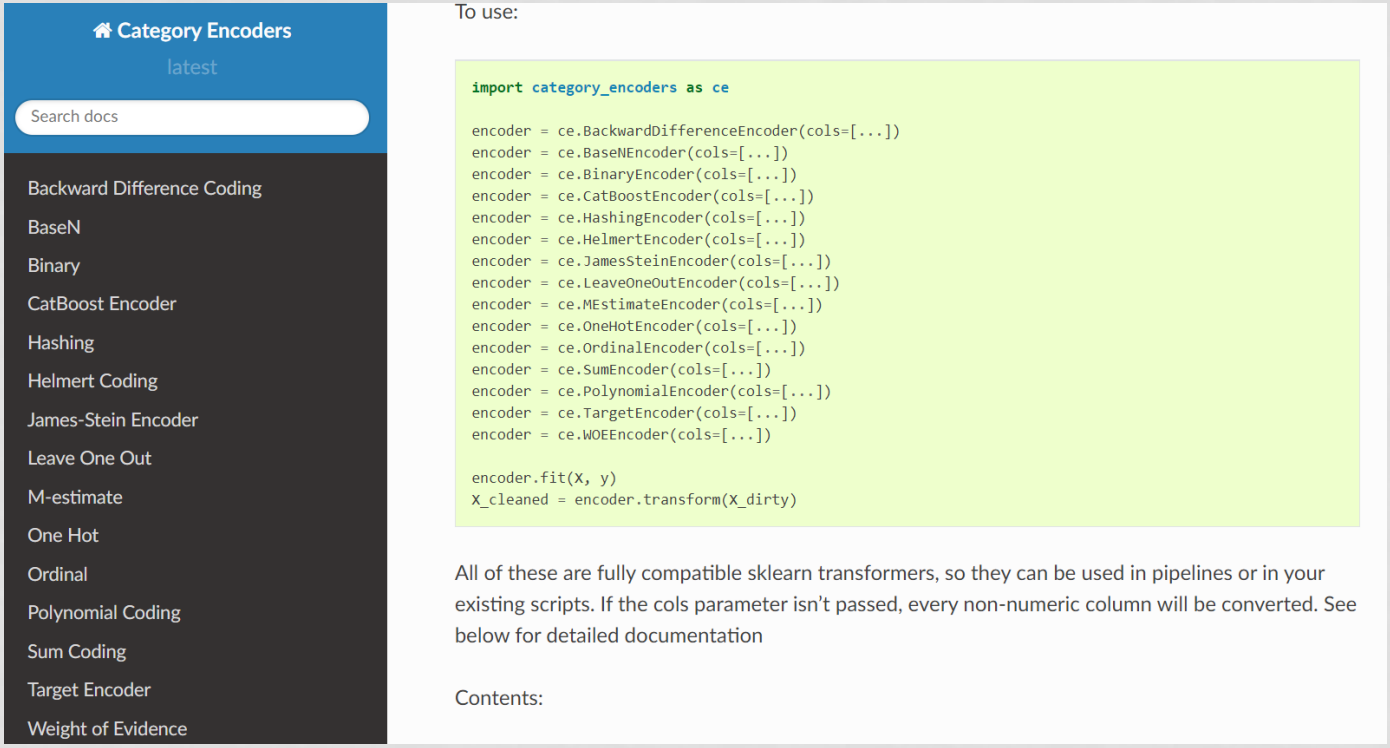
- However, for category values with very few instances, average estimation is going to be inaccurate.
- Smoothing is typically applied, so that minority categories are encoded with values similar to the global average.
- $Label_C = \frac{n_c * p_c + \alpha * p_{global}}{n_c + \alpha}$
- n_c: instances ∈ category C
- p_c : class1 probability of instances ∈ C
- p_{global} : prior class1 probability of the response variable on the complete training set (or average, in case of regression)
- α = regularization coefficient (5 is recommended)

For "very hot":

$Label_C = \frac{1 * 1.0 + \alpha * 7/10}{1 + \alpha} = 0.75$ for alpha=5

CATEGORICAL ENCODING FOR SCIKIT-LEARN

- https://github.com/scikit-learn-contrib/category_encoders
- http://contrib.scikit-learn.org/category_encoders/index.html

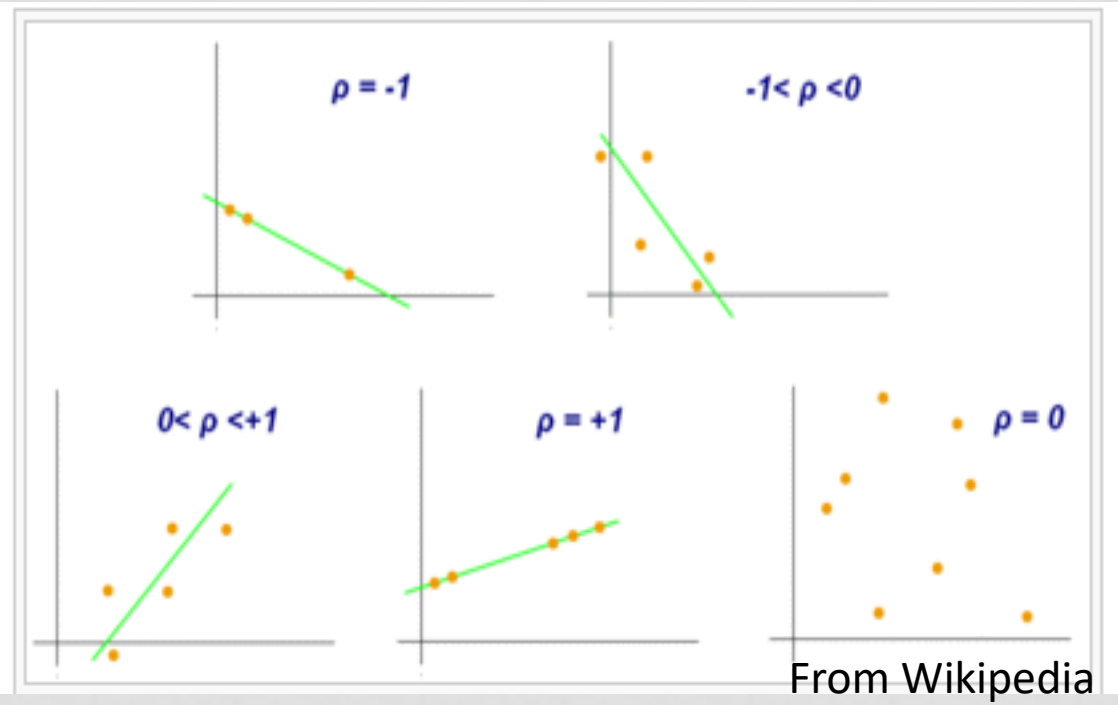


Feature selection: filter methods

- **Filter**: feature importance is evaluated for each attribute individually, using a simple statistical / mathematical method. Then, features are ranked and the worse ones are discarded.
- Given input attributes A_1, A_2, ..., A_n, each A_i is evaluated **individually**, computing its correlation or dependency with the response variable, independently of the rest of attributes (i.e. attributes are considered individually, rather than subsets)
- An attribute A_1 is correlated with the class, if knowing its value implies that the class can be predicted more accurately
  - For instance, car speed is correlated with having an accident. But the Social Security Number of the driver is not.
  - For instance, salary is (inversely) correlated with credit default
- How to evaluate / rank attributes (attribute/class correlation):
  - Linear correlation
  - Chi-square
  - Mutual information
  - …
- Once evaluated and ranked, the worst attributes can be removed (according to a threshold)

LINEAR CORRELATION

- Pearson coefficient $-1 <= r <= +1$
- $Covariance(X,Y) = E((X-\mu_X)(Y-\mu_Y))$
- $Correlation(X,Y) = r = \frac{Cov(X,Y)}{\sigma_X \sigma_Y}$



From Wikipedia

FILTER IN SCIKIT

- sklearn.feature_selection.SelectKBest

`f_classif`

ANOVA F-value between label/feature for classification tasks.

`mutual_info_classif`

Mutual information for a discrete target.

`chi2`

Chi-squared stats of non-negative features for classification tasks.

`f_regression`

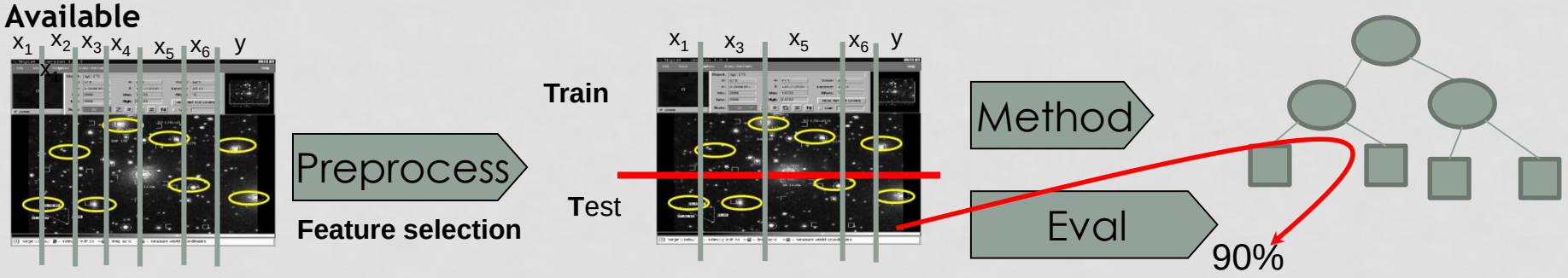
F-value between label/feature for regression tasks.

`mutual_info_regression`

Mutual information for a continuous target.

HOW TO DO PREPROCESSING CORRECTLY?

- You may think the following workflow is correct, but the problem is that there might be some "information leakage" from the test partition to the training partition (i.e. the model will "know" a bit about the test partition)
- If the evaluation is to be correct, the model should be completely independent from the test partition, but if we do preprocessing, before splitting into train and test, we are going to use information for the preprocessing (which indirectly ends up in the model), that later will belong to the test partition.
- Otherwise, the model will know something about the testing partition and **model evaluation (90%) will be too optimistic**.

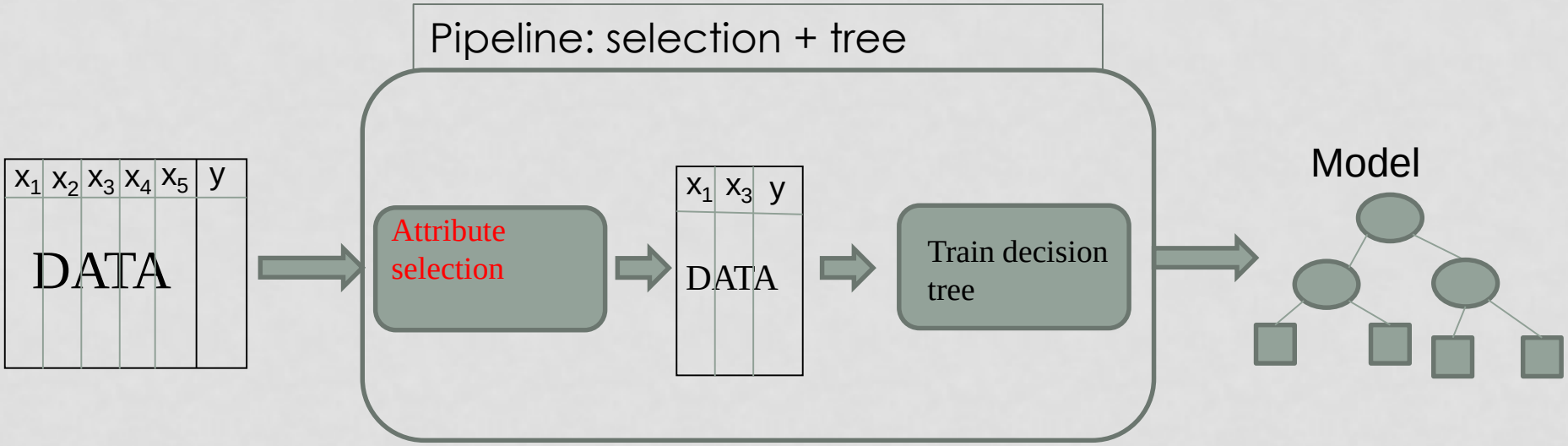


HOW TO DO PREPROCESSING CORRECTLY?

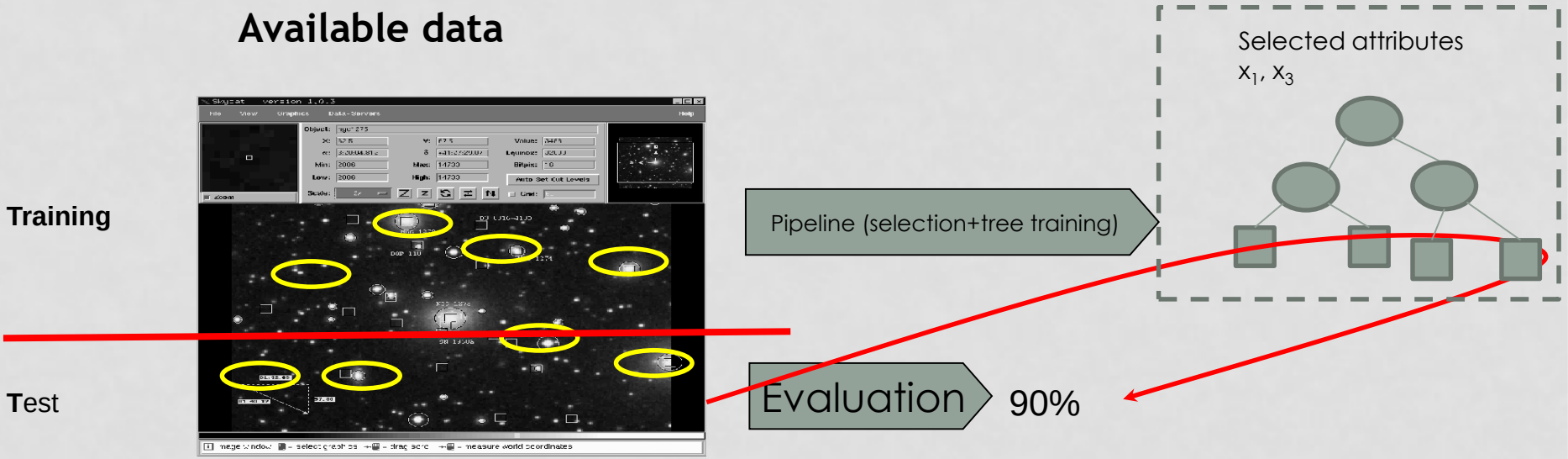
- We shouldn't use test data for training the model, in any way

HOW TO DO PREPROCESSING CORRECTLY?

- Put preprocessing inside the training method
  - pipeline
- E.g. for attribute selection:

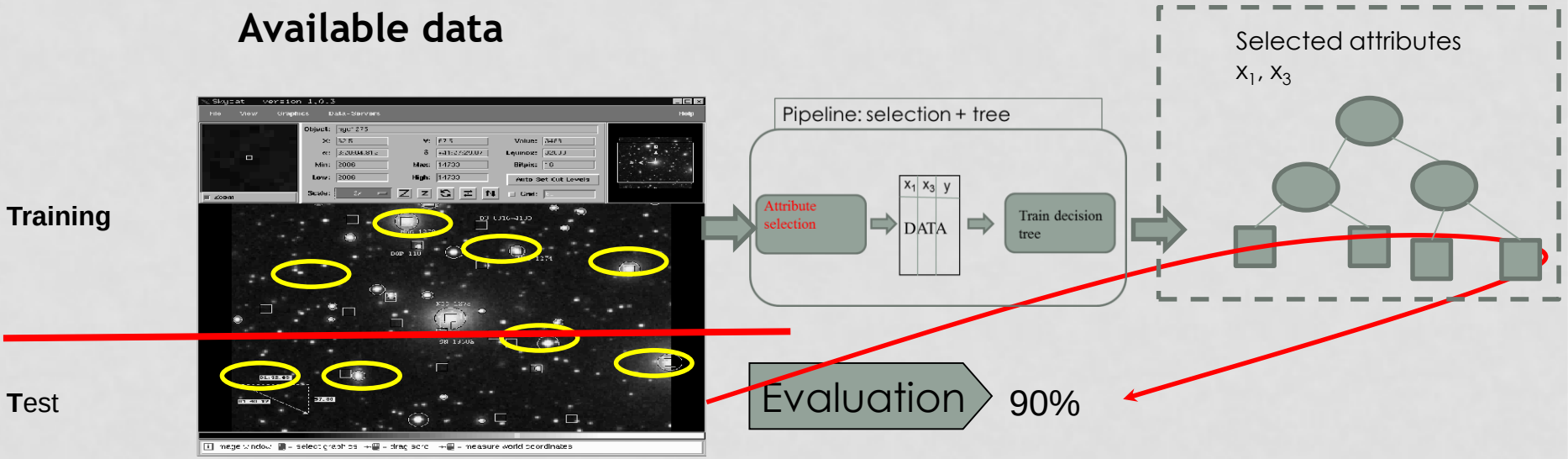


HOW TO DO PREPROCESSING CORRECTLY?



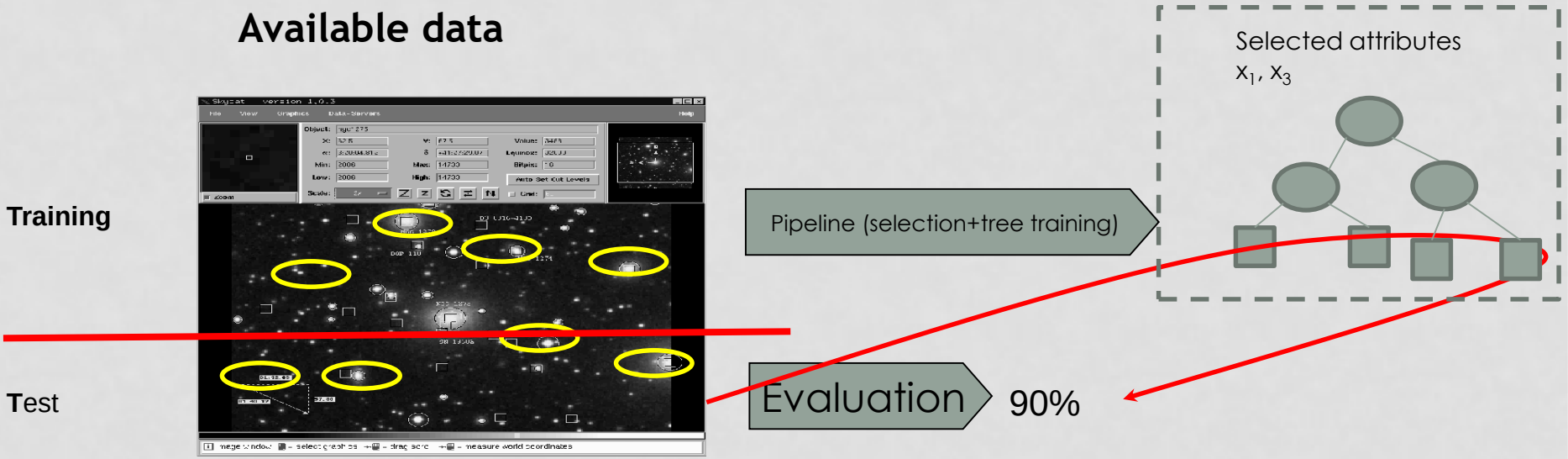
- Which attributes are selected is decided with the training partition only, and kept for use during testing
- Now, the trained model is not just the tree, but also the attributes selected during training

HOW TO DO PREPROCESSING CORRECTLY?



- Which attributes are selected is decided with the training partition only, and kept for use during testing

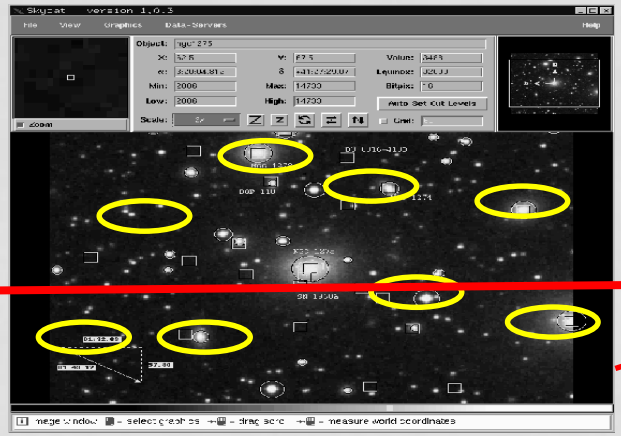
HOW TO DO PREPROCESSING CORRECTLY?



- Which attributes are selected is decided with the training partition only, and kept inside the model for use during testing
- The same thing is done for other preprocessing tasks:
  - For attribute scaling, max(xi), min(xi) are computed using training data only, and kept for use during testing
  - For imputation, mean(xi) is computed with training data, and used during testing.

AN EXAMPLE WITH IMPUTATION USING THE MEAN

Available data



Train

Test

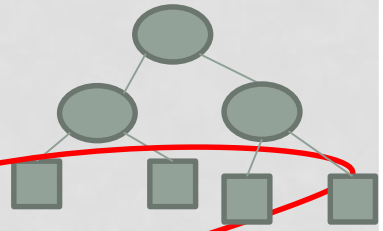
pipeline (imputation+tree)

(reimputation)

Eval

90%

Means of X_1, X_2

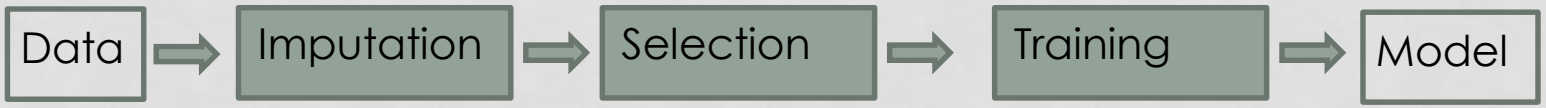


HOW TO DO PREPROCESSING CORRECTLY?

- Two types of pre-processing:
  - No information-leakage: this kind of preprocessing CAN be done on the complete dataset (before splitting into train/test)
    - Remove ID attribute because we know it is not useful for classification
    - Create dummy variables
    - Removing constant columns
  - Information-leakage:
    - Important (because the response variable values of the test partition are used for the preprocessing):
      - Feature selection: remove attribute $\mathbf{x_4}$ because its values are not correlated with the class
    - Less important (because the resp. var. of the test partition is not used, although the features of the test partition are used
      - Scaling
      - Imputation

PIPELINES IN SCIKIT LEARN

- Doing preprocessing correctly involves using pipelines: sequences of preprocessing and training steps.
- For example, we might want to do:
  1. Imputation (to remove missing values)
  2. Attribute selection (to select the most relevant features)
  3. Model training



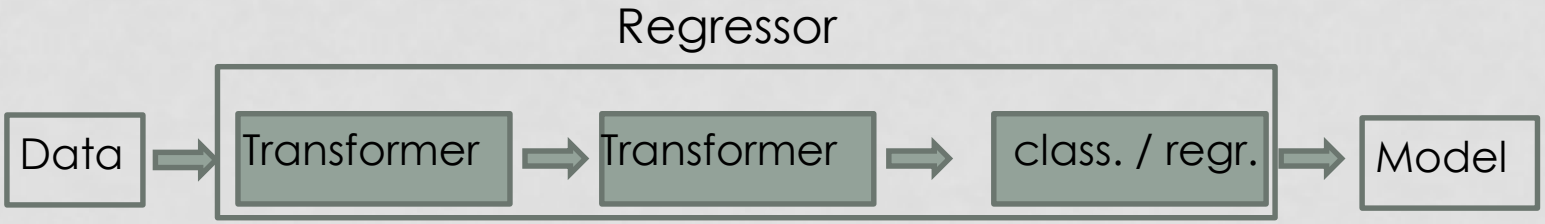
- Pipelines in sklearn are sequences of estimators: an **estimator** in sklearn is either a **transformer** (or pre-processing method) or a **classifier/regressor** (or training method)

PIPELINES IN SCIKIT-LEARN

https://scikit-learn.org/stable/modules/compose.html#pipeline

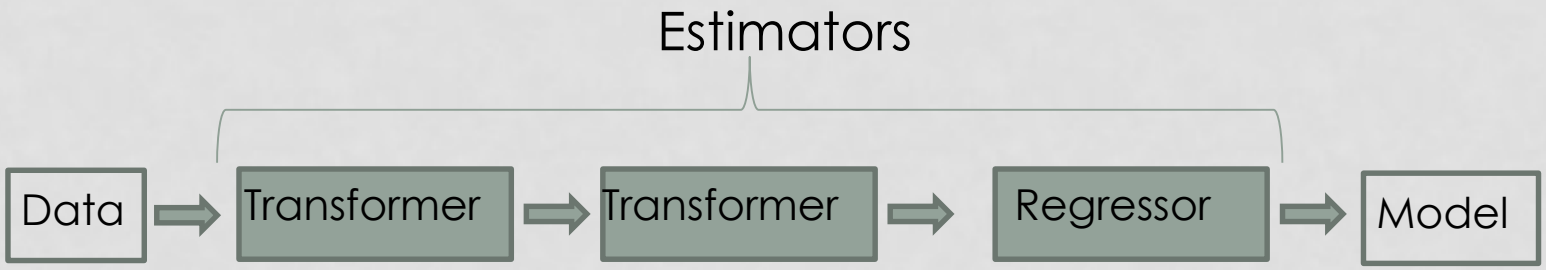
- Transformers (preprocessing): feature selection, principal component analysis, binarizer, scaler, imputer, category encoder, …
- Classifier / regressors (model training methods): decision trees, knn, svm, …
- A sequence of transformers IS a transformer:

- A sequence of Transformers ended in a classifier or regressor IS a classifier or regressor



PIPELINES IN SCIKIT LEARN

- Pipeline: a sequence of estimators (transformers and classifier/regression)
- Transformer: preprocessing. They have two methods:
  - **.fit** (applied to training data)
  - **.transform** (typically applied to testing data)
- Classifier / regressor: decision trees, knn, SVM's, … Two methods:
  - **.fit** (for training models on training data)
  - **.predict** (typically, for computing predictions on testing data)



PIPELINE FOR FEATURE SCALING WITH KNN

```
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsRegressor
from sklearn.pipeline import Pipeline

m_boston = load_boston()
X = m_boston.data
y = m_boston.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

scaler = StandardScaler()
knn = KNeighborsRegressor()

regr = Pipeline([
    ('scaling', scaler),
    ('knn', knn)
])

regr.fit(X_train, y_train)
prediction = regr.predict(X_test)
```

PIPELINE FOR IMPUTATION AND FEATURE SCALING WITH KNN

```
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.neighbors import KNeighborsRegressor
from sklearn.pipeline import Pipeline

m_boston = load_boston()
X = m_boston.data
y = m_boston.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

scaler = StandardScaler()
imp = SimpleImputer(missing_values=np.nan, strategy='mean')
knn = KNeighborsRegressor()

regr = Pipeline([
    ('imp-mean', imp),
    ('scaling', scaler),
    ('knn', knn)
])

regr.fit(X_train, y_train)
prediction = regr.predict(X_test)
```

WHY USE PIPELINES?

1. Avoiding information leakage: test data should never be used for training, in any way
2. Clear coding: a pipeline clearly states your preprocessing and training methods
3. Hyper-parameter tuning: each step in the pipeline has its own hyper-parameters. Pipelines make possible to tune all of them

Pipeline persistence

- Trained / fit pipelines can be saved into a file in pickle format, to be used later
- Caution! if the version of sklearn changes, or a different architecture is used (e.g. saving in Windows10 and loading in Linux), this may lead to unexpected results

```
from joblib import dump, load
dump(pca_sel_knn, 'pca_sel_knn.joblib')
pca_sel_knn = load('pca_sel_knn.joblib')
```