More on data preprocessing

Michel.RIVEILL@univ-cotedazur.fr

Outlier detection

https://towardsdatascience.com/a-brief-overview-of-outlier-detection-techniques-1e0b2c19e561

https://machinelearningmastery.com/how-to-use-statistics-to-identify-outliers-in-data/

https://scikit-learn.org/stable/modules/outlier_detection.html

Most common causes of outliers

- The most common or familiar type of outlier is the observations that are far from the rest of the observations or the center of mass of observations.
  - Anomalous events occur relatively infrequently
  - However, when they occur, their impact can be significant and often in a bad way
- Outliers causes
  - Data entry errors (human errors)
  - Measurement errors (instrument errors)
  - Data processing errors (data manipulation or data set unintended mutations)
  - Intentional (dummy outliers made to test detection methods)
  - Sampling errors (extracting or mixing data from wrong or various sources)

  - **Natural (not an error, novelties in data)**

Why removing outliers could be important

- It can be important to identify and remove outliers from data
  - Eliminating outliers allows for more accurate predictions.
  - Outliers distort statistical measures and data distributions
  - Outliers provide a misleading representation of the data.

- Generally, outlier detection is an unsupervised problem
  - We do not know which observations should be considered as "abnormal"

- Outliers could be:
  - **Univariate** can be found by examining a distribution of values in a single feature space
  - **Multivariate** can be found in a n-dimensional space
- Outliers could be
  - Point: single data
  - Contextual: noise in the data
  - Collective: subset of novelties in data

Most popular methods for outlier detection

Some of the most popular methods for outlier detection are:

Z-Score or Extreme Value Analysis (parametric)

Probabilistic and Statistical Modeling (parametric)

Linear Regression Models (PCA, LMS)

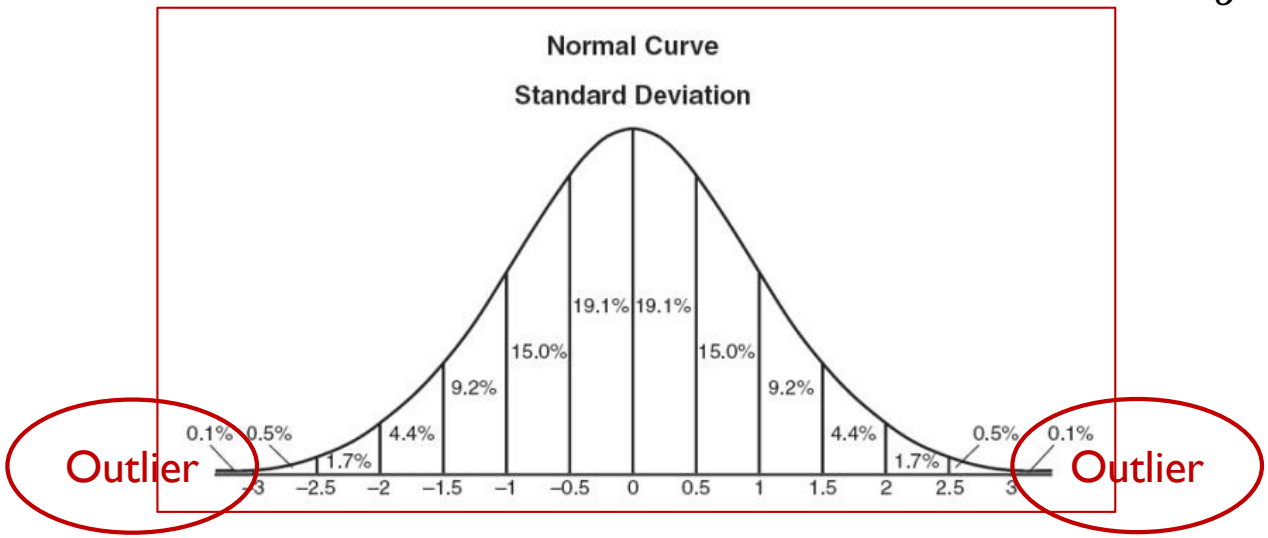
~~Proximity Based Models (non-parametric)~~

Information Theory Models

High Dimensional Outlier Detection Methods (high dimensional sparse data)

z-score (or standard score)

- Remove all data with |z−score| $> 3, z - score = \frac{x-\mu}{\sigma}$



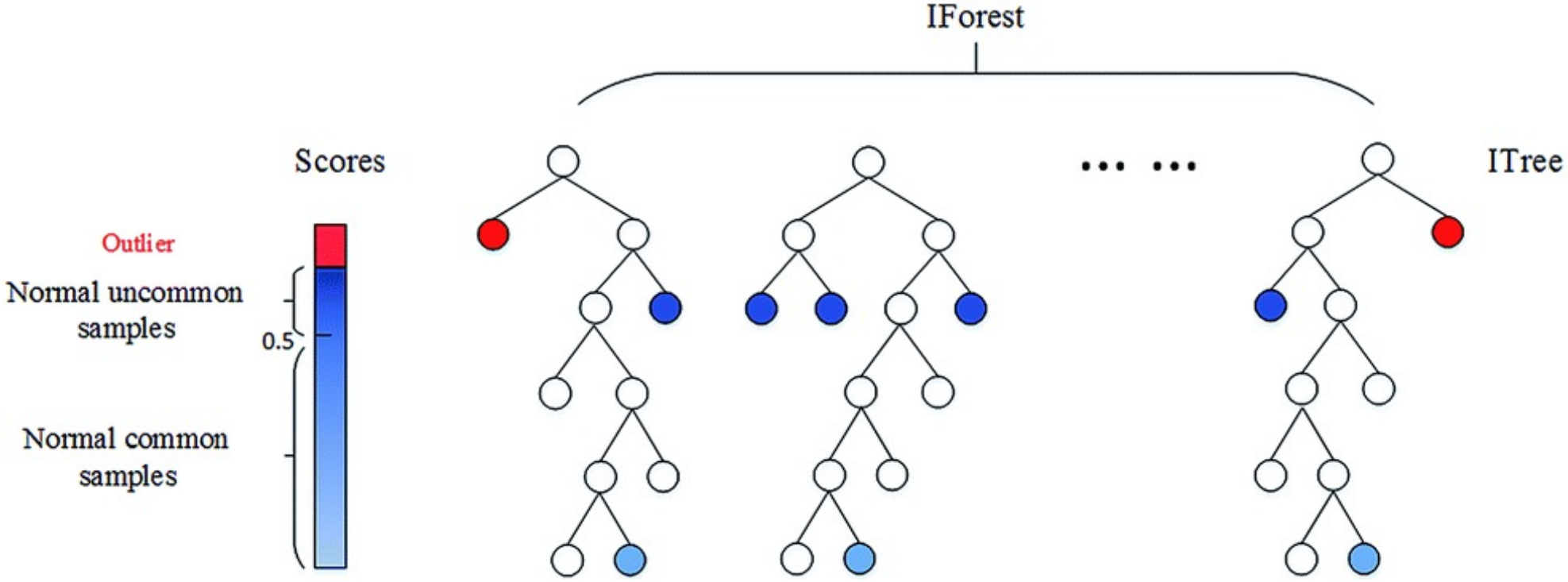
- For univariate
  - → compute z-score for each observation
  - → Remove each observation with |z−score| > 3
- For multivariate
  - Compute the distance between each observation and the mean
  - We now consider the distance as a univariate characteristic

Z-score in Python

```
def detect_outlier(data, multivariate=True, threshold=3):
    if multivariate:
      mean = np.mean(data)
      data = [scipy.spatial.distance.euclidean(mean, d)
                    for d in data]

    mean = np.mean(data)
    std = np.std(data)

    outliers=[d for d in data
                    if np.abs((d-mean)/std))>threshold]

    return outliers
```

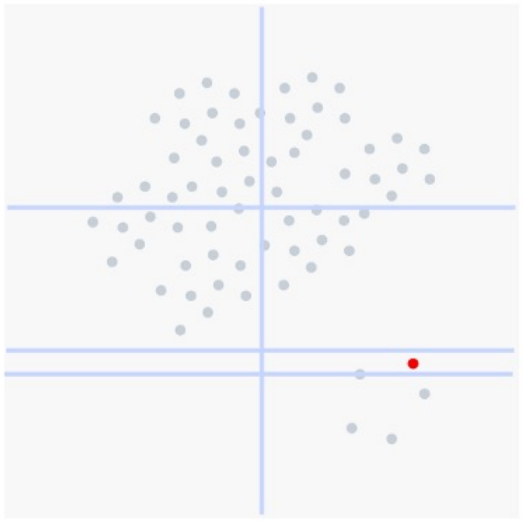
Isolation forest

1. Build a tree
   - We build a decision tree by choosing successively, in a random way, a characteristic and for this characteristic a random division value $\in [min, max]$
2. Build a forest: build randomly many trees
3. The predicted value is the mean of the path length between the root and the leaf
4. Find outlier: the outliers have a shorter path length than the other

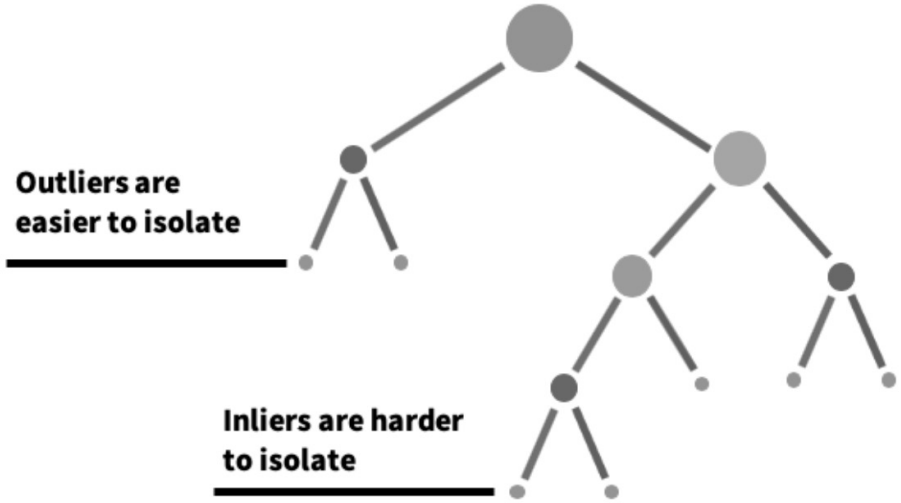


Isolation forest

Isolation of a normal point



Isolation of an anomaly



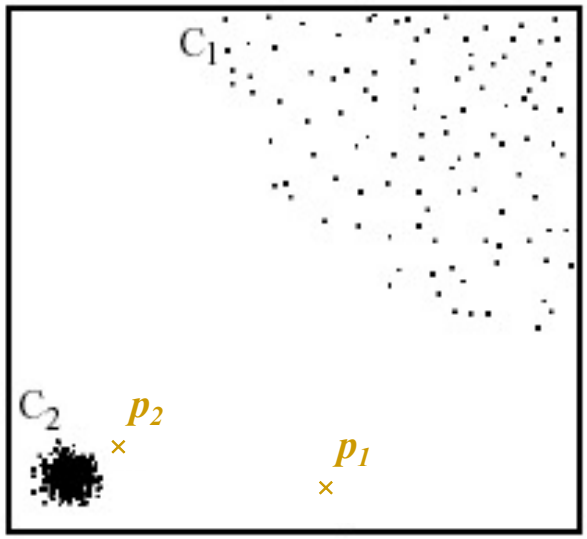
sklearn.ensemble.IsolationForest

- **n_estimators**, *default=100*
- **max_samples**
  - The number of samples to draw from X to train each base estimator.
  - If int, use max_samples
  - If float, use max_samples * n_samples
  - If "auto", use max_samples=min(256, n_samples)
- **Contamination,** *default='auto'*
  - The amount of contamination of the data set, i.e. the proportion of outliers in the data set.
  - Used when fitting to define the threshold on the scores of the samples.
  - If 'auto', the threshold is determined as in the original paper.
  - If float, the contamination should be in the range (0, 0.5].
- **max_features**, *default=1.0*
  - The number of features used to fit each tree
  - If int, then use max_features features.
  - If float, then use max_features * n_features.

- fit(X[, y]), Fit estimator.
- predict([X]), Predict the labels (1 inlier, -1 outlier)

Density-based: Local Outlier Factor (LOF) approach

- A simple approach to identifying outliers is to locate those examples that are far from the other examples in the feature space.
  - For each point, compute the density of its local neighborhood (LOF)
  - Outliers are points with largest LOF value



In the NN approach, p_2 is not considered as outlier, while LOF approach find both p_1 and p_2 as outliers

Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, Jörg Sander, "LOF: identifying density-based local outliers", ACM SIGMOD Volume 29, Issue 2, June 2000, pp. 93–104.

Local Outlier Factor (LOF)

- The local outlier factor LOF, is defined as follows:

$$LOF_k(p) = \frac{\sum_{o \in N_k(p)} \frac{lrd_k(o)}{lrd_k(p)}}{|N_k(p)|}$$

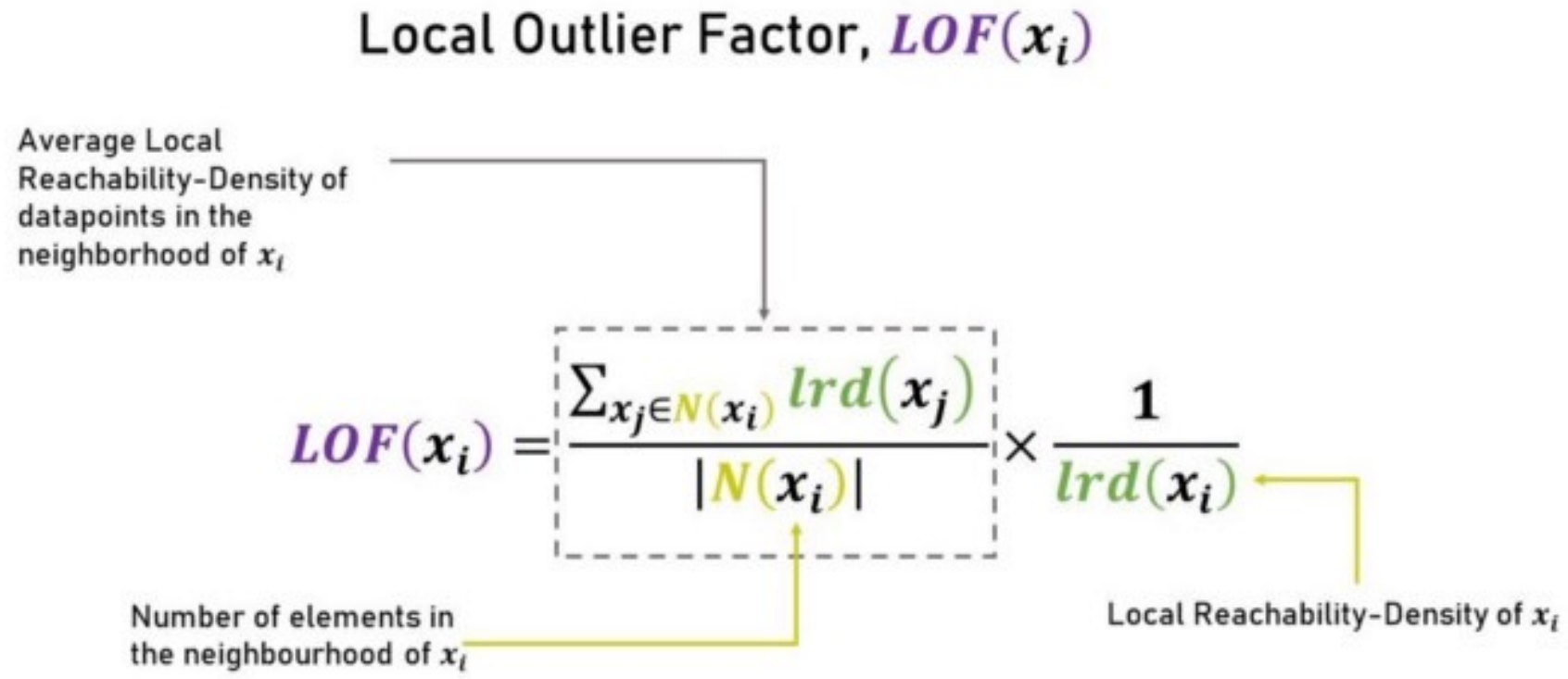
- where $N_k(p)$ is the set of k-nearest neighbors to p

- And $$lrd_k(p) = \frac{|N_k(p)|}{\sum_{o \in N_k(p)} reach - dist(p,o)}$$

$$reach - dist_k(p) = \max\{k - dist(o), dist(p,o)\}$$

- Work well for feature spaces with low dimensionality (few features),
  - It can become less reliable as the number of features is increased (the curse of dimensionality)

Local Outlier Factor (LOF)



sklearn.neighbors.LocalOutlierFactor

- **n_neighbors**, *default=20*
  - Number of neighbors to use by default for kneighbors queries. If n_neighbors is larger than the number of samples provided, all samples will be used
- **metric**, *default='minkowski'*
  - The metric is used for distance computation. Any metric from scikit-learn or scipy.spatial.distance can be used.
- **contamination** *'auto' or float, default='auto'*
  - The amount of contamination of the data set, i.e. the proportion of outliers in the data set.
  - When fitting this is used to define the threshold on the scores of the samples.
  - if 'auto', the threshold is determined as in the original paper,
  - if a float, the contamination should be in the range (0, 0.5].

- fit(X[, y]), Fit estimator.
- predict([X]), Predict the labels (1 inlier, -1 outlier)

Minimum Covariance Determinant (MCD)

- Hubert, Mia et al. "Minimum Covariance Determinant and Extensions." *arXiv:* 1709.07045 (2017)

- If the input variables have a Gaussian distribution, then simple statistical methods can be used to detect outliers
  - Define a hypersphere (ellipsoid) that covers the normal data, and data that falls outside this shape is considered an outlier
  - An efficient implementation of this technique for multivariate data is known as the Minimum Covariance Determinant, or MCD for short.

Classical Covariance

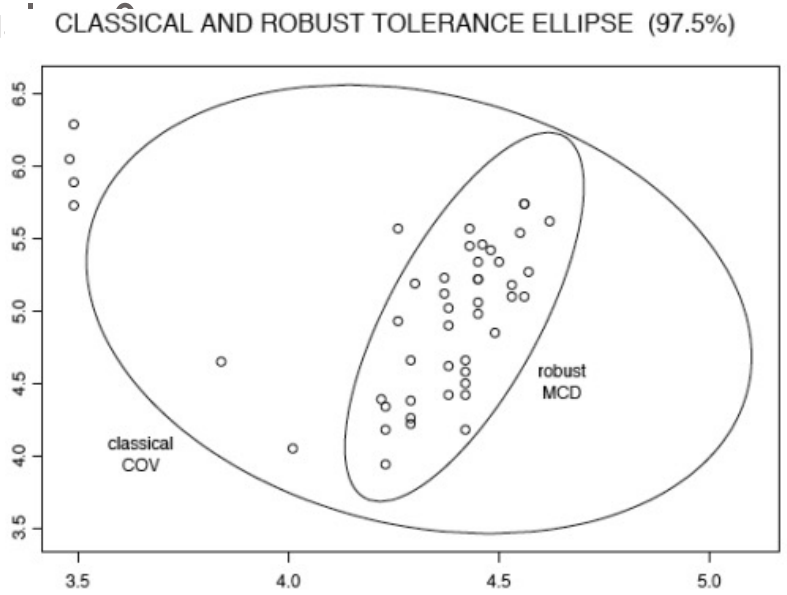
- Assume 10% of samples are outliers

- Algorithm
  - determine all unique sample subsets of size **h = 0.9N**
  - for each subset S
    - compute the covariance matrix of Cs = S
    - compute Vs = determinant (Cs) # Volume
  - choose the subset where Vs is a minimal

Fast- Minimum Covariance Determinant

- choose a random subset of H0 of X, with size h

- repeat
  - Determine covariance S and mean M of the subset H0
  - Determine distances d(Xi) for all Xi relative to H with the Mahalanobis distance
    - $D = [(x - M)S^{-1}(x - M)]^{1/2}$
  - Choose the h smallest distances and create a new subset H1
  - repeat with H0 := H1, until H0 and H1 are equ
- Evaluate from 1 for K times (maybe 500) and determine the selection that had the smallest volume.



sklearn.covariance.MinCovDet

- **store_precision**, *default=True*
  - Specify if the estimated precision is stored.
- **assume_centered,** *default=False*
  - If True, the support of robust location and covariance estimates is computed, and a covariance estimate is recomputed from it, without centering the data.
  - If False, the robust location and covariance are directly computed with the FastMCD algorithm without additional treatment.
- **support_fraction**, *default=None*
  - The proportion of points to be included in the support of the raw MCD estimate.
  - If None, the minimum value of support_fraction will be used within the algorithm: [n_sample + n_features + 1] / 2. Range is (0, 1).

- fit(X[, y]), Fit estimator.
- predict([X]), Predict the labels (1 inlier, -1 outlier)

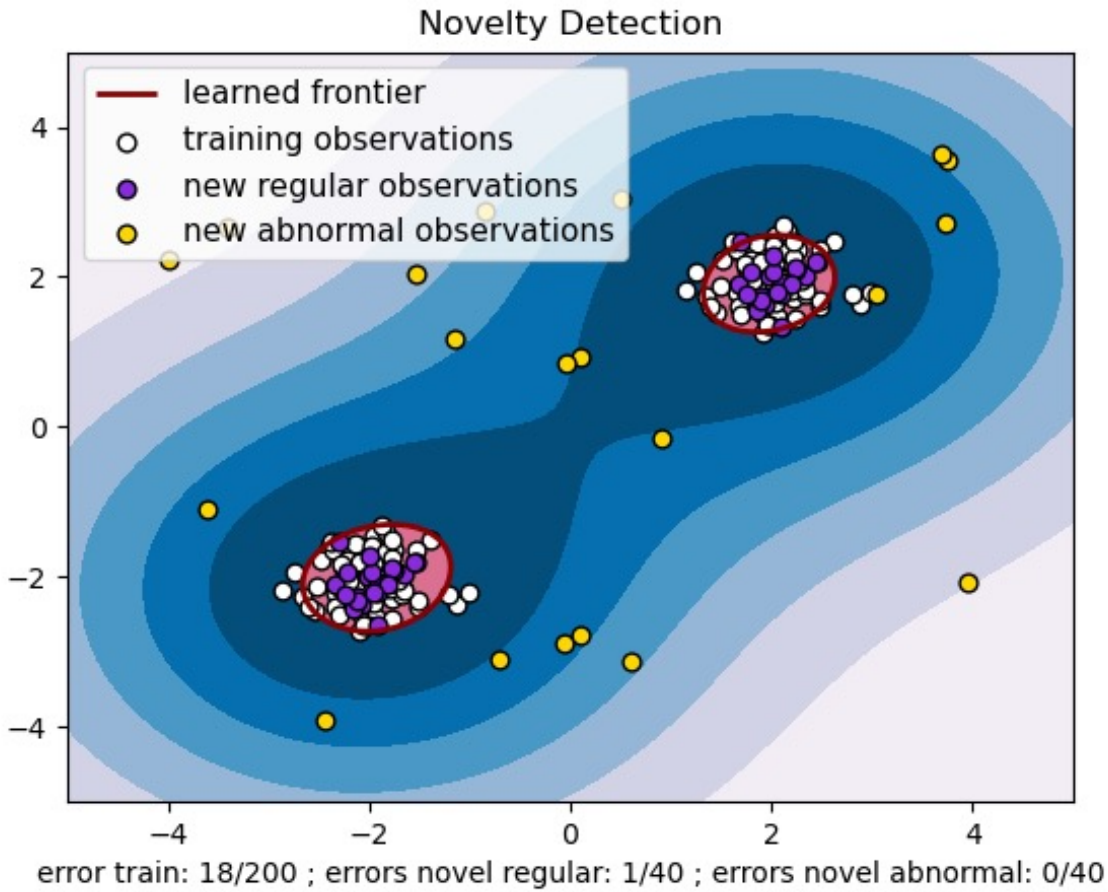
One-class SVM

- SVMs are max-margin methods,
  - They do not model a probability distribution.
  - The idea is to find a function that is positive for regions with high density of points, and negative for small densities.

  - SVM separates two classes using a hyperplane with the largest possible margin.
  - One-Class SVM uses a hypersphere to encompass all of the instances.
  - The "margin" as referring to the outside of the hypersphere -- so by "the largest possible margin", we mean "the smallest possible hypersphere".

Bernhard Schölkopf, John C. Platt, John C. Shawe-Taylor, Alex J. Smola, Robert C. Williamson, "Estimating the Support of a High-Dimensional Distribution", Neural Computation, Volume 13, Issue 7, July 2001, pp. 1443–1471.

One-class SVM

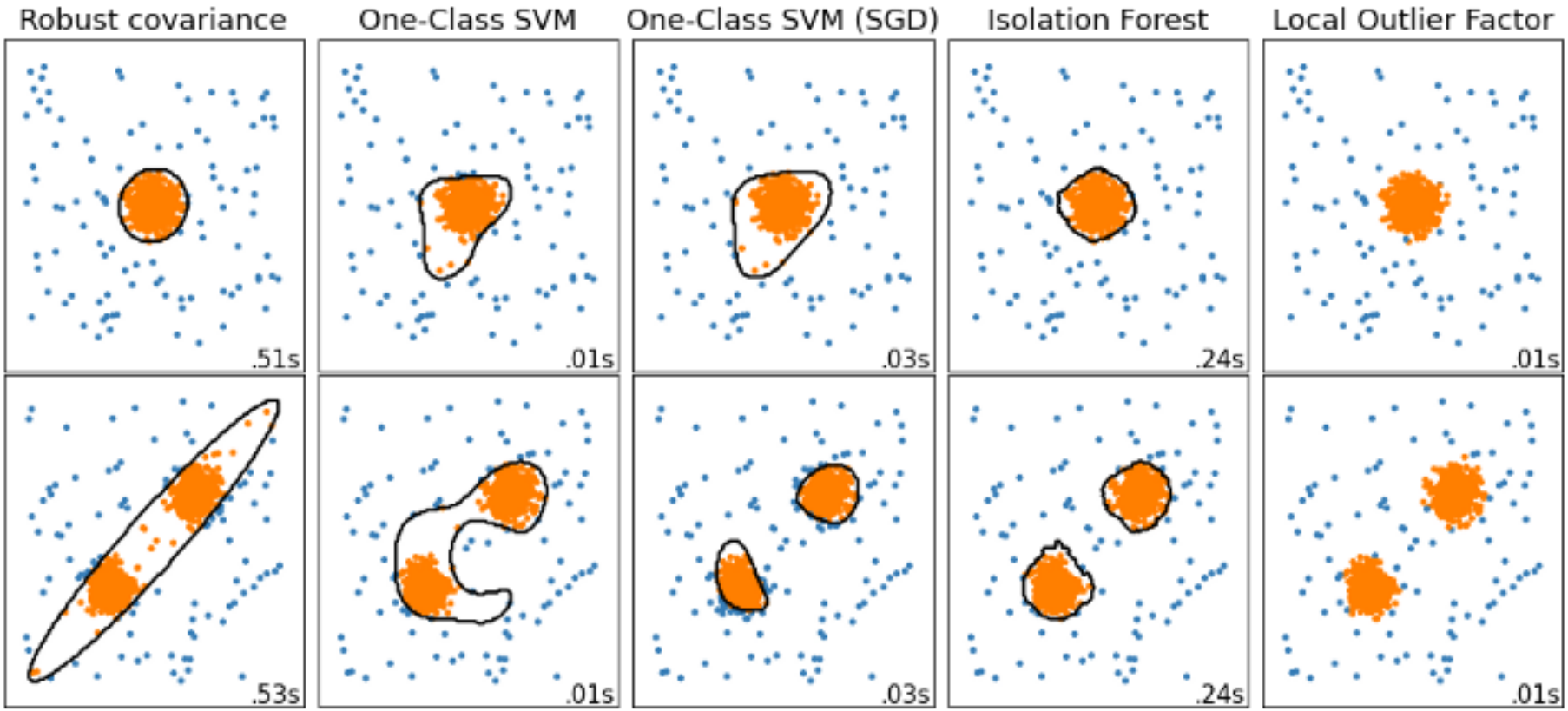


sklearn.svm.OneClassSVM

- **Kernel**, *default='rbf'*
  - Specifies the kernel type to be used in the algorithm. I
- **Gamma**, *default='scale'*
  - if gamma='scale' (default) is passed then it uses I / (n_features * X.var()) as value of gamma,
  - if 'auto', uses I / n_features.
- **nu**, *default=0.5*
  - An upper bound on the fraction of training errors and a lower bound of the fraction of support vectors.
  - Should be in the interval (0, I].

- fit(X[, y]), Fit estimator.
- predict([X]), Predict the labels (I inlier, -I outlier)

Benchmarking



Lab1

- Test some algorithms on a dataset that contains outliers
  - Choose the classifier that you think is preferable for this job.

1. Without taking any precautions
2. By eliminating outliers with one of the following approaches:
   - With Isolation Forest (IF)
   - With Local Outlier Factor (LOF)
   - With Minimum Covariance Determinant (MCD)

   - Use a gridsearch to find the right hyperpameters.

- Submit your work in the form of an executable and commented notebook at lms.univ-cotedazur.fr

Handling Missing Data

MISSING DATA

- Missing data are observations that were originally planned but were not made
- Missing data can occur for many reasons:
  - Participants may not respond to questions (legitimately or illegitimately),
  - Collection or recording equipment malfunctioned,
  - Subjects may withdraw from studies before they are completed, data entry oversights may occur.

Types of Missing data

There are 3 major categories of missing values:

- **Missing Completely at Random(MCAR):**
  - no relationship between the missing data and any value
- **Missing at Random(MAR):**
  - missing data is random but there is a systematic relationship between it and the other data,
  - Example:
    - Women are less likely to talk about their age and weight than men.
    - Men are less likely to talk about their salary and emotions than women.
- **Missing Not at Random(MNAR):**
  - The most difficult of the absence situations.
  - The absence is not random, and there is a systematic relationship between the missing value, the observed value, and the absence itself

Detecting missing data

- In Python, missing data is usually indicated by NaN.
- Count the NaN under an entire DataFrame (pandas)
  - df.isnull().sum().sum()
- Count the NaN under an entire array (numpy)
  - np.count_nonzero(np.isnan(data))

Treating missing data: Deletion Methods

- Deletes the missing values from a dataset. and conduct analysis using only complete cases.
- Advantage: Simplicity
- Disadvantage: loss of data if we discard all incomplete cases.
- NOTE: deleting some observations changes the distribution of the different variables.

Listwise deletion

- Preferred when there is a Missing Completely at Random case (MCAR)
- Entire rows(which hold the missing values) are deleted

id	Gender	Age	result
1	Male	20	Positive
~~2~~	~~Female~~		~~Negative~~
3	Female	35	Positive
~~4~~		~~23~~	~~Negative~~
5	Female	24	Positive
6	Male	26	Positive

- Remove observation
  - Pandas: df.dropna(axis=0)
  - Numpy: data[~np.isnan(data).any(axis=1), :] # remove the first dimension

Deletion Method, example

Observation=10,variable=4,15% missing values → 6 missing values

Obser-vation	Case 1				Case 2				Case 3			
	y_1	y_2	y_3	y_4	y_1	y_2	y_3	y_4	y_1	y_2	y_3	y_4
1	NA	NA	NA	NA	NA				NA			
2	NA	NA			NA				NA			
3					NA					NA		
4					NA					NA		
5					NA						NA	
6					NA						NA	
7												
8												
9												
10												

Eliminate obs. 1, 2.
Keep 8*4=32 data.
20% loss

Eliminate variable 1.
Keep 4*4=16 data.
60% loss

Eliminate obs 1-6.
Keep 4*4=16 data.
60% loss

30

Treating missing data: Imputation Methods

- Replace missing value by a value

1. Random sample from existing values
   - Replace the missing value by the value of a randomly selected observation
   - Example:

Obs	1	2	3	4	5	6	7	8	9	10
var_1	3.4	3.9	2.6	1.9	2.2	3.3	1.7	2.4	2.8	3.6
var_2	5.7	4.8	4.9	6.2	6.8	5.6	5.4	4.9	5.7	?

   - Select randomly a complete observation → say 3
   - Replace var_2^{10} by var_2^3 i.e. replace ? By 4.9

   - Advantage: We can use complete case analyses
   - Disadvantage: it may change the distribution of data

Dealing with Missing Data Imputation Methods

- Replace missing value by a value

2. Mean/Min/Max/Previous Substitution
   - Replace missing value with the sample mean (or min, max, previous substitution). Then, run analyses as if all complete cases
   - Example:

Obs	1	2	3	4	5	6	7	8	9	10
var_1	3.4	3.9	2.6	1.9	2.2	3.3	1.7	2.4	2.8	3.6
var_2	5.7	4.8	4.9	6.2	6.8	5.6	5.4	4.9	5.7	?

   - Select randomly a complete observation → say 3
   - Replace var_2^{10} by the mean (5.55), the min (4.8), the max (6.8), the previous (5.7)

   - Advantage: We can use complete case analyses
   - Disadvantage: it may change the distribution of data

Imputation methods in python
Single imputation strategy

- Impute missing data by a new one, based on « columns » observation

- from sklearn.impute import SimpleImputer
  - **Strategy,** The imputation strategy.
    - "mean", replace missing values using the mean along each column. Can only be used with numeric data.
    - "median", replace missing values using the median along each column. Can only be used with numeric data.
    - "most_frequent", replace missing using the most frequent value along each column. Can be used with strings or numeric data. If there is more than one such value, only the smallest is returned.
    - "constant", replace missing values with fill_value. Can be used with strings or numeric data.

Imputation methods in python
Multiple imputation strategy

- Modeling each variable with missing values as a function of other variables in a round-robin mode.

- from sklearn.impute import IterativeImputer
  - **Estimator:** estimator to use at each step of the round-robin imputation
  - **n_nearest_features:** number of other features to use to estimate the missing values of each feature column.
  - **initial_strategy:** Which strategy to use to initialize the missing values. Same as the strategy parameter in SimpleImputer.
  - **imputation_order**
    - 'ascending': From features with fewest missing values to most.
    - 'descending': From features with most missing values to fewest.
    - 'roman': Left to right.
    - 'arabic': Right to left.
    - 'random': A random order for each round.

Imputation methods in python
Multiple imputation strategy

K_Nearest Neighbor Imputation

- The KNN algorithm helps to impute missing data by finding the closest neighbors
  - use Euclidean distance metric
  - Imputs missing data with non-missing values from neighbors.

- from sklearn.impute import KNNImputer
  - **n_neighbors:** Number of neighboring samples to use for imputation.
  - **Weights**: Weight function used in prediction.
    - 'uniform' : All points in each neighborhood are weighted equally.
    - 'distance' : weight points by the inverse of their distance.

Lab2

- Test some algorithms on a dataset that contains missing values
  - Choose the classifier that you think is preferable for this job.

1. with removal of missing data
2. with imputation of missing data by a value

- If necessary use a gridsearch to find the right hyperpameters.

- Submit your work in the form of an executable and commented notebook at lms.univ-cotedazur.fr

Working with Unbalanced dataset

Unbalanced dataset

- The models tend to favor the majority class.
- If a gradient descent algorithm is used -> the majority class dominates the gradient
- And in most cases, the likelihood is dominated by the majority class.
- It is therefore important to "counterbalance" these effects.
- The strategies focus on:
  - Modifying the dataset
  - Modifying the cost function
- They are generally not used at the same time.
- Whatever the approach, a suitable metric must be chosen

Adapted metric

- In the context of unbalanced datasets, it is very important to use adapted metrics.

- Indeed, in the case of a dataset with an anomaly rate of 0.01%
  - A model that classifies all data as anomaly free will have an accuracy of 0.99%.
  - This is of no interest because it will be very difficult to compare the various models.

How to evaluate the classification

- In dataset 0.5 % patients have cancer
  - y = 1, the patient has cancer
  - y = 0, the patient does not have cancer

Model 1	Model 2
Step 1: I train a logistic regression Step 2: I predict with my model → 1% patients have cancer For 1.000 patients, I detect only 4 patients with cancer and also 6 false positive	def predictCancer(x) return 0 0 % patients have cancer For every 1,000 patients, no patient has cancer.

Model 1 better than model 2 ?

How to evaluate the classification

0.5 % of patient have cancer	Model 1 1 % of patient have cancer			Model 2 0 % of patient have cancer		
Matrice de confusion		1	0		1	0
	1	4	6	1	0	0
	0	1	989	0	5	995
ACC	$\frac{4+989}{1000} = 0.993$			$\frac{995}{1000} = 0.995$		
TP	4			0		
TP+FP	10			0		
TP+FN	5			5		
Precision	TP / (TP+FP) = 4/10 = 0.4			0/0 = NaN≈0		
Recall	TP / (TP+FN) = 4/5 = 0.8			0/5 = 0.0		
F1 score	**2*P*R/(P+R) = 2*0.4*0.8 / (0.4+0.8) = 0.64 / 1.2 = 0,53**			**NaN≈0**		

More adapted metric

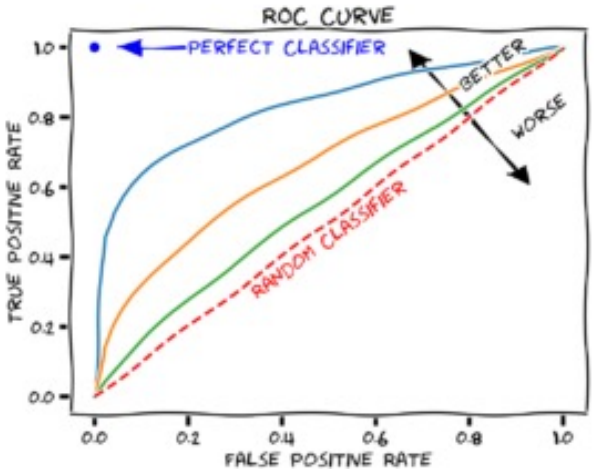
- Precision
  - This is the rate of true positives. It gives the proportion of data that really belong to the "Positve" class among those that have been classified in the "positive" class
- Recall
  - Proportion of data well classified by the classification algorithm among the data of the "Positive" class
- F1-score
  - It is a criterion that allows to make a compromise between precision and recall

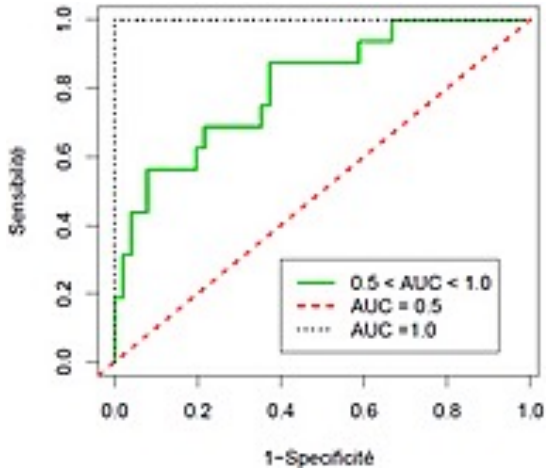
- G-mean
  - G-mean allows to make a compromise between sensitivity/recall and selectivity
    - Sensitivity/recall = $\frac{TP}{TP+FP}$ Selectivity = $\frac{TN}{TN+FN}$
    - G-mean = $\sqrt{sensitivity * selectivity}$
- AUC (precision-recall)

https://support.sas.com/resources/papers/proceedings17/0942-2017.pdf

More adapted metric : ROC AUC

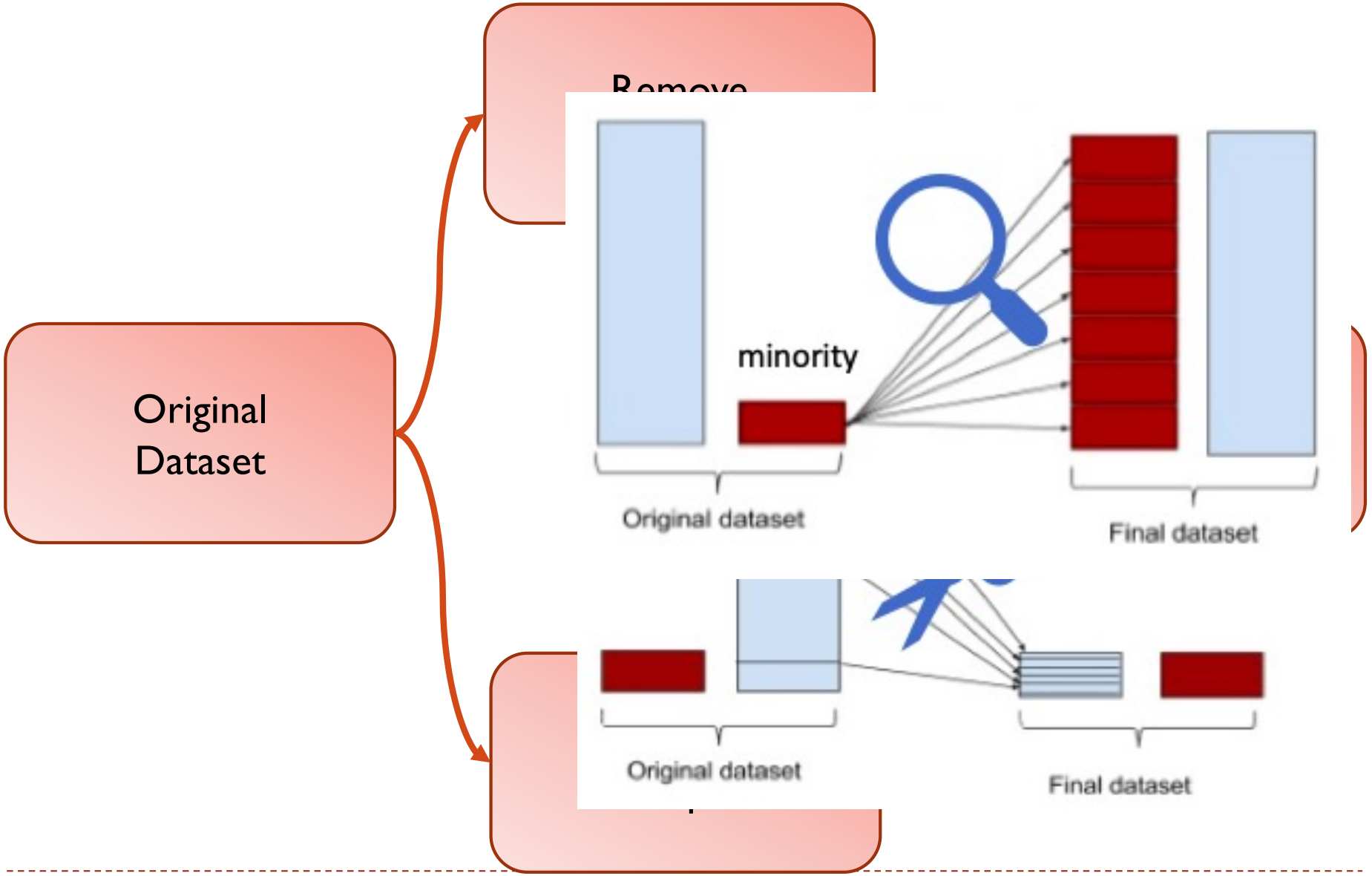
- The ROC (Receiver Operating Characteristic) curve
  - represents sensitivity (recall) versus 1 - specificity for all possible thresholds
  - Sensitivity (recall) is the ability of the test to detect patients
  - Specificity is the ability of the test to detect non-diseased patients.





  - How to interpret its AUC?
    - An AUC of 0.5 (50%) indicates that the model is non-informative
    - An increase in the AUC indicates an improvement in discriminatory abilities, with a maximum of 1.0 (100%)

Dataset strategies



Basic dataset strategies
Random Under Sampling

- **Remove samples from the majority class: RUS [Kubat97]**
  - Sample the data records from majority class (Randomly, Near miss examples, Examples far from minority class examples (far from decision boundaries)
  - Introduce sampled data records into the original data set instead of original data records from the majority class
  - Usually results in a general loss of information and overly general rules

```
from imblearn.under_sampling import RandomUnderSampler
rus = RandomUnderSampler(sampling_strategy= ???)
X_res, y_res = rus.fit_resample(X, y)
```

  - Sampling strategy
    - Float: specify the ratio $\frac{minority\ class}{majority\ class}$ (only for binary classification)
    - String: the number of samples in the different classes will be equalized
      - ‘minority’: resample only the minority class – only under sampling
      - 'all': resample all classes
      - ‘auto’

Basic dataset strategies
Random Over Sampling

- **Over-sampling the rare class: ROS** [Ling98]
  - Make the duplicates of the rare events until the data set contains as many examples as the majority class => balance the classes
  - Does not increase information but increase misclassification cost

```
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(sampling_strategy= ???)
X_res, y_res = ros.fit_resample(X, y)
```

  - Sampling strategy
    - Float: specify the ratio $\frac{minority\ class}{majority\ class}$ (only for binary classification)
    - String: the number of samples in the different classes will be equalized
      - 'majority': resample only the majority class – only over sampling
      - 'all': resample all classes
      - 'auto'

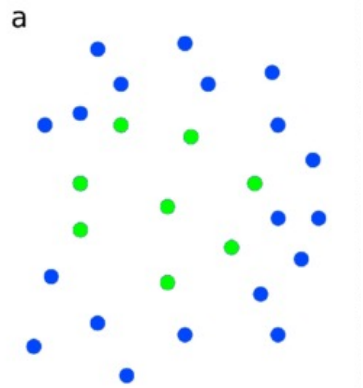
Basic dataset strategies
Generating artificial anomalies

- SMOTE (Synthetic M**artificial**inority Over-sampling TEchnique) [Chawla02]
  - new rare class examples are generated inside the regions of existing rare class examples
  - Artificial anomalies are generated around the edges of the sparsely populated data regions [Fan01]
  - Classify synthetic outliers vs. real normal data using active learning [Abe06]

```
from imblearn.over_sampling import SMOTE
smote = SMOTE(sampling_strategy=???, k_neighbors=5)
X_res, y_res = smote.fit_resample(X, y)
```

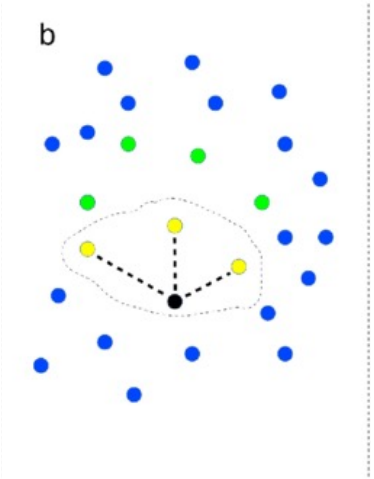
  - Sampling strategy
    - Float: specify the ratio $\frac{minority\ class}{majority\ class}$ (only for binary classification)
    - String: the number of samples in the different classes will be equalized
      - 'majority': resample only the majority class – only over sampling
      - 'all': resample all classes
      - 'auto'
  - **Neighbor**
    - number of nearest neighbours to used to construct synthetic samples

SMOTE



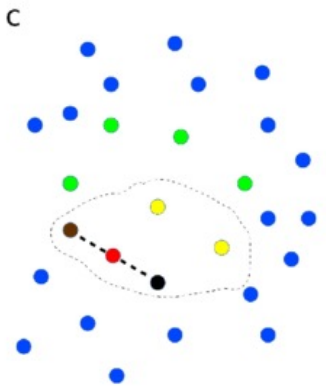
SMOTE starts from a set of positive (green points) and negative (blue points) examples

SMOTE



It then selects a positive example (black) and its k nearest neighbors among the positives (yellow points, with k = 3),

SMOTE



- Finally one of the k nearest neighbours is randomly selected (brown point)
- a new synthetic positive example is added, by randomly generating an example (red point) along the straight line that connects the black and brown points.

b and c is repeated for all the positives, by adding each time a new synthetic example similar (in an Euclidean sense) to the other positive examples.

Algorithm strategies

- Main advantage
  - Does not change the data and calculation time

- Class weights:
  - Instead of repeating samples (or removing samples)

    Re-weigth the loss function
  - Works for most model

  - Same effect as over-sampling (add samplers)
    - But not as expensive → dataset size remain the same

    - Generally `class_weight` parameter of the model
      - 'balanced': adjust weights inversely proportional to class frequencies
      - {class_label: weight}

Others algorithm strategies

- **Not All Classification Errors Are Equal**
  - **Cancer Diagnosis Problem**: Consider a problem where a doctor wants to determine whether a patient has cancer or not. It is better to diagnose a healthy patient with cancer and follow-up with more medical tests than it is to discharge a patient that has cancer.
  - **Fraud Detection Problem**: Consider the problem of an insurance company wants to determine whether a claim is fraudulent. Identifying good claims as fraudulent and following up with the customer is better than honoring fraudulent insurance claims.

  - It's possible to use `class_weight` parameter in order to penalize more FP as FN

Others algorithm strategies

- By modifying the cost function, when it can be done
  - in neural networks by implementing a custom loss
  - by implementing the desired model (linear and logistic regression)
  - This is unfortunately not possible with sklearn.

- Cost matrix: A matrix that assigns a cost to each cell in the confusion matrix.
  - | Actual Negative | Actual Positive
  - Predicted Negative | C(0,0), TN | C(0,1), FN
  - Predicted Positive | C(1,0), FP | C(1,1), TP

  - Total cost = C(0,1) * FN + C(1,0) * FP

Cost Matrix

- Example
  - In an insurance claim,
    - the costs for a false positive maight be monetary cost of follow-up with the customer to the company
    - The costs for a false negative might be the cost of the insurance claim
  - In a cancer diagnostic,
    - The cost of a false positive might be the monetary cost of performing subsequent test, whereas what is the equivalent euro cost for letting a sick patient go home and get sicker
  - Generally, cost matrix is defined by a domain expert or economist
- For example we may have a dataset with a 1 to 100 ration of examples in the minority class, this ratio can be inverted and used as the cos of misclassifications erroers

	Actual Negative	Actual Positive
Predicted Negative	0	100, FN
Predicted Positive	1, FP	0, TP

- mean false error (MFE)
  - $MFE = FPE + FNE$
- squared mean false error (SMFE)
  - $SMFE = FPR^2 + FNE^2$

- FPE : mean false positive error
  - $FPE = \frac{1}{N}\sum_{i=1}^{N}\sum_{n}\frac{1}{2}(d_n^{(i)} - y_n^{(i)})^2$
- FNE : mean false negative error
  - $FPE = \frac{1}{P}\sum_{i=1}^{N}\sum_{n}\frac{1}{2}(d_n^{(i)} - y_n^{(i)})^2$

Others algorithm strategies

- Focal loss:
  - Modification of the cross entropy loss
  - Give more weight to the examples that are difficult to classify (false negatives) compared to the false positives, easier to classify.
  - $p_t = \begin{cases} p \; if \; y = 1 \\ 1 - p, otherwise \end{cases}$
  - $\alpha_t = \begin{cases} \alpha \; if \; y = 1 \\ 1 - \alpha, otherwise' \end{cases}$
  - $FL = -\alpha(1-p)^{\gamma} CE_{loss}$
    - *α is a class weight*
    - *If p → 1, classe facile à classifier alors le facteur de modulation → 0 afin de réduire impact de cette classe*
  - $FL(p_t) = -\alpha_t(1-p_t)^{\gamma}\log(p_t)$, γ is a hyper-parameters

Lab3

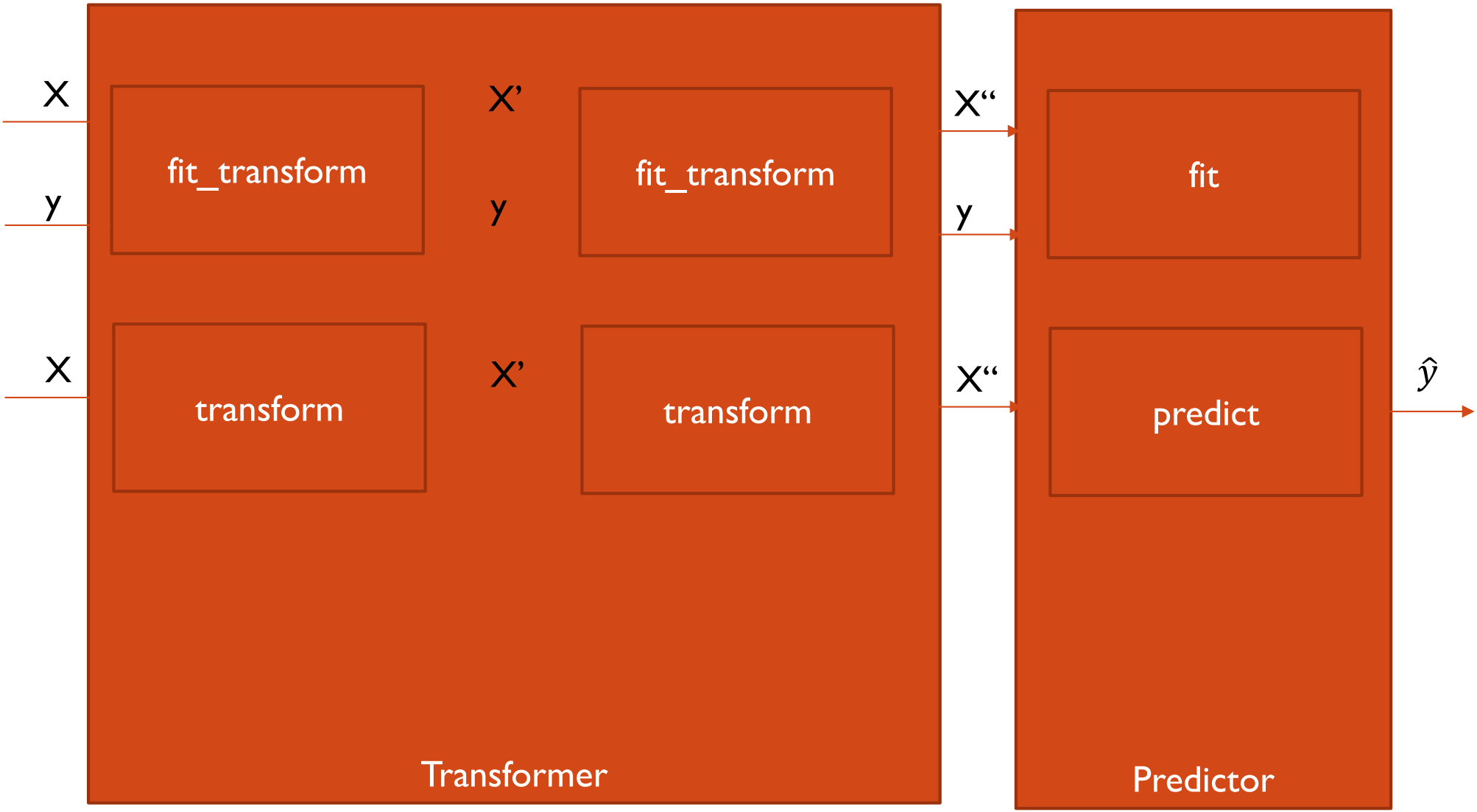
- Test some algorithms on an unbalanced dataset
  - Choose the classifier that you think is preferable for this job.

1. Without taking any precautions
2. With modification of the dataset by Over sampling or Under sampling or SMOTE
3. Without modification of the dataset by weight

- If necessary use a gridsearch to find the right hyperpameters.

- Submit your work in the form of an executable and commented notebook at lms.univ-cotedazur.fr

Sklearn pipeline



Sklearn pipeline, build custom transformer

- Implements a transform that replaces missing values with the most frequent value
- Note: a transform does not change the number of elements in the dataset

```
class myImputer (BaseEstimator, TransformerMixin):
    def __init__(self, strategy="most_frequent"):
        self.strategy = strategy
        self.sample = SimpleImputer(strategy=self.strategy)
    def fit(self, X, y=None):
        return self.sample.fit(X, y)
    def transform(self, X):
        return self.sample.transform(X)
```

Sklearn pipeline, build custom predictor

- implements an estimator that performs logistic regression

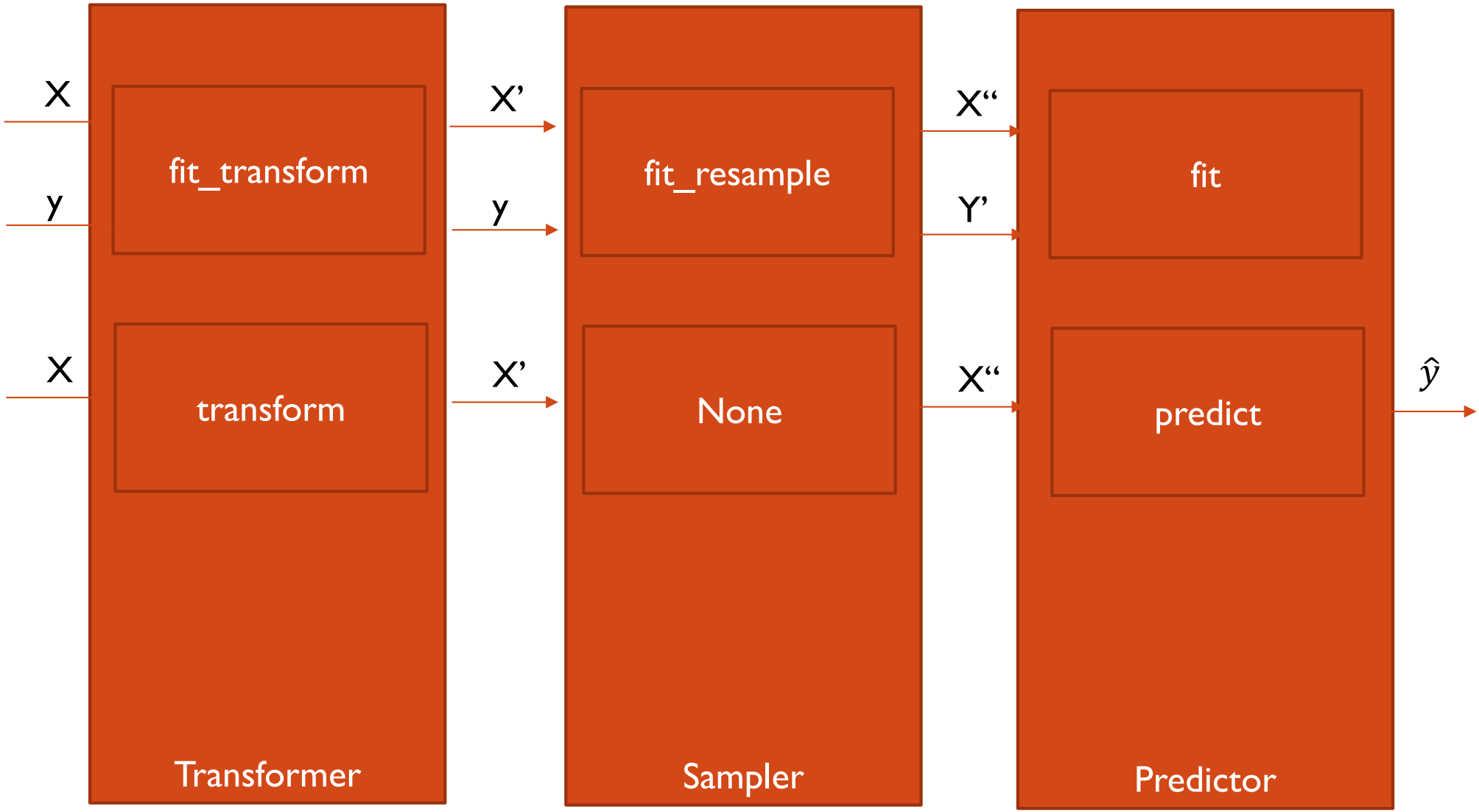
```
class myLogisticRegression(BaseEstimator):
    def __init__(self, penalty='none'):
        self.penalty = penalty
        self.sample = LogisticRegression(penalty=self.penalty)
    def fit(self, X, y):
        self.sample.fit(X, y)
    def predict(self, X):
        return self.sample.predict(X)
```

Imblearn pipeline

- The elements of the sklearn pipelines, do not allow to change the number of elements in the dataset

- the imblearn library, which offers various sampling methods, offers a new type of elements that allow to modify the number of elements in the dataset only when building the predictor

Imblearn pipeline



Imblearn, build sampler

```
class myDataCleaner(BaseEstimator):
    def fit_resample(self, X, y):
        data = np.concatenate((X, y), axis=1)
        # remove rows with NaN
        data = data[~np.isnan(data).any(axis=1), :]
        return X_[:,:-1], y[:,-1]
```

Build pipeline (put all together)

```
# Two step pipeline
# Step 1: handle missing data
# Step 2: predictor
pipeline = Pipeline([('missing_data', SimpleImputer()),
                     ('clf', myLogisticRegression())])
```

Pipeline parameters for GridSearchCV

```
# Two strategies for missing data
# 1. Impute them
# 2. Remove them
parameters = [{'missing_data': [myImputer()],
               'missing_data__strategy': ['mean', 'median', 'most_frequent'],
               'clf': [myLogisticRegression()],
               'clf__penalty': ['none', 'l2'],
              },
              {'missing_data': [myDataCleaner()],
               'clf': [myLogisticRegression()],
               'clf__penalty': ['none'], #'l2'],
              }]
```

GridSearchCV

```
grid = GridSearchCV(pipeline, parameters, cv=2,
                    scoring="f1_micro", refit=True,
                    verbose=2)

grid.fit(X, y)

y_pred = grid.predict(X)
print("Best: {:.2f} using {}".format(
    grid.best_score_,
    grid.best_params_
))

print('Test set score: ' + str(grid.score(X, y)))
```