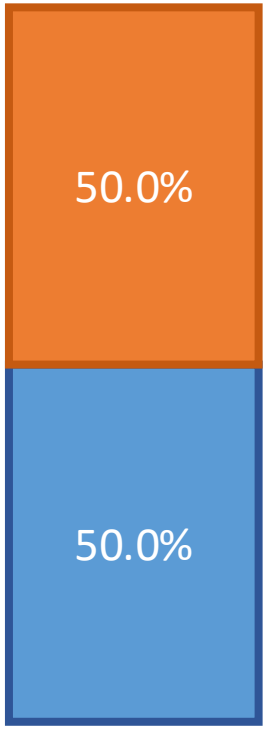
Data Analytics for Digital Health Imbalanced Learning

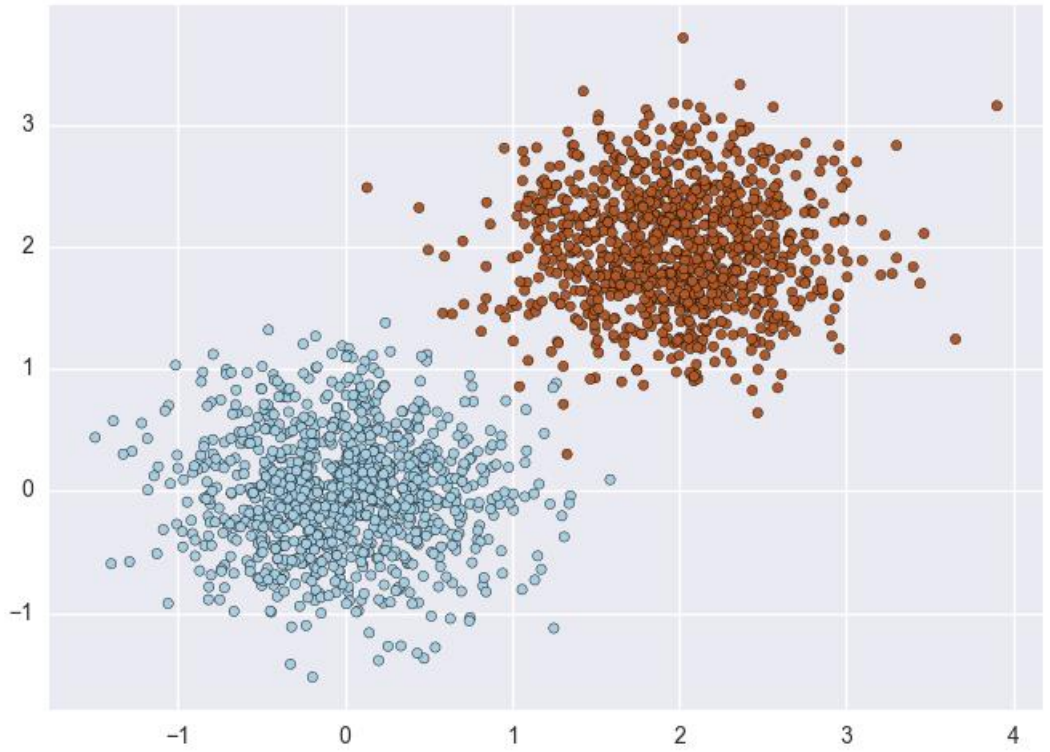
Riccardo Guidotti, Francesca Naretto

IN SUPREMÆ DIGNITATIS 1343 | UNIVERSITÀ DI PISA

Imbalanced Classes

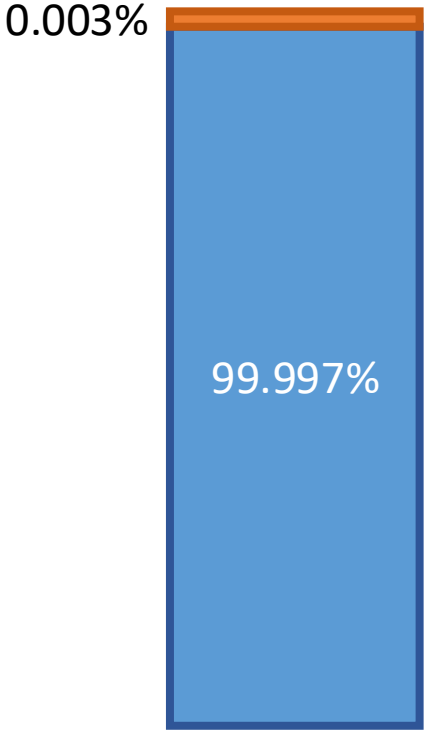
- Most classification methods assume classes are reasonably balanced.

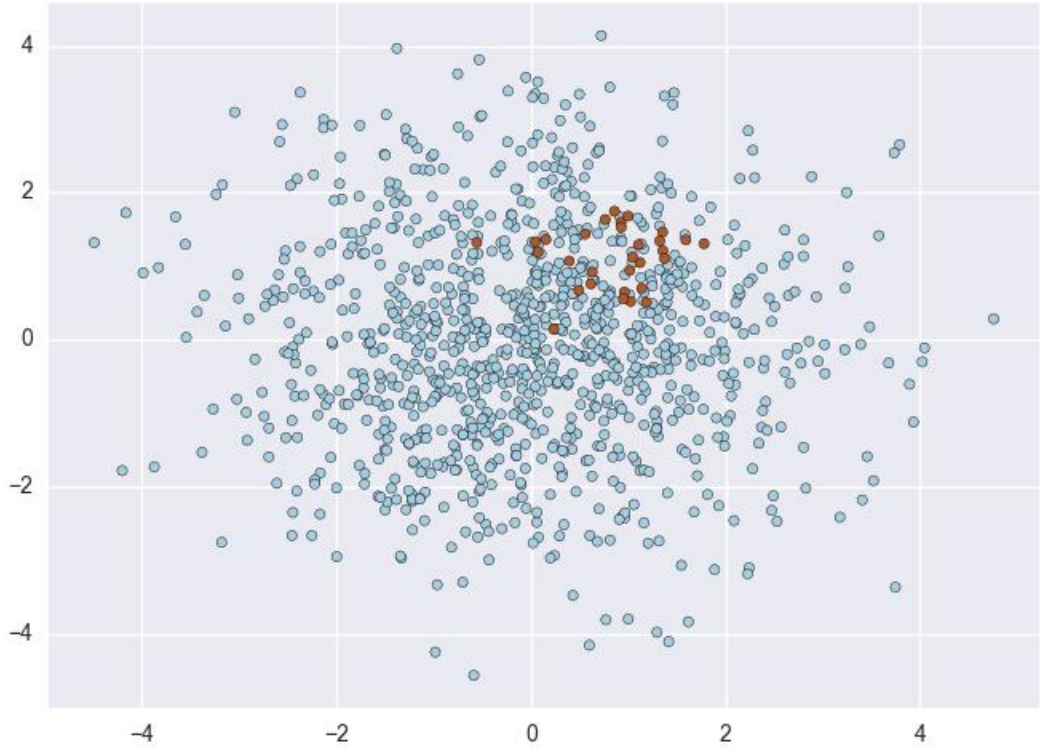




Imbalanced Classes

- In reality it is quite common to have a very popular class and a rare (yet interesting) class.





This occurs when there is a large discrepancy between the number of examples with each class label.

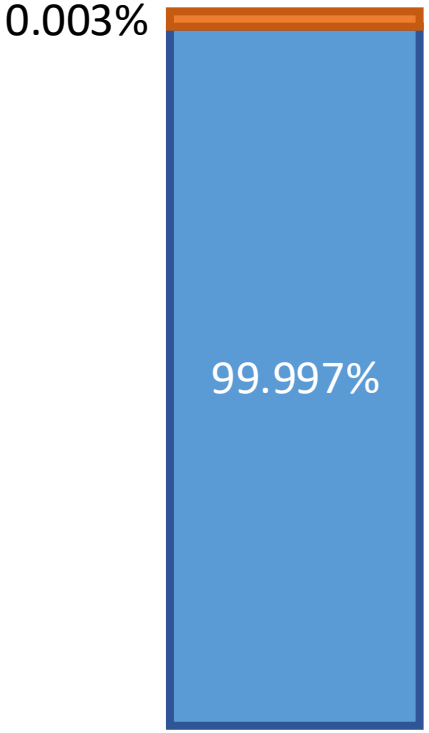
E.g. for 1M example dataset only about 30 represent an event.

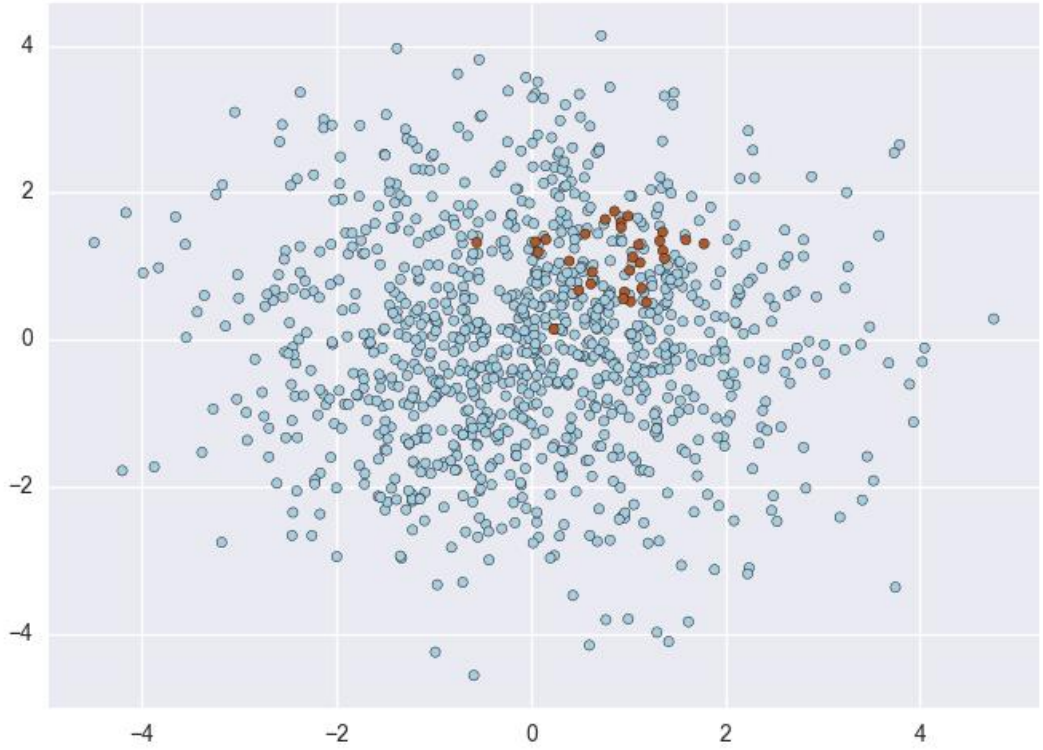
Examples

- About 2% of ***credit card*** accounts are defrauded per year1. (Most fraud detection domains are heavily imbalanced.)
- ***Medical screening*** for a condition is usually performed on a large population of people without the condition, to detect a small minority with it (e.g., HIV prevalence in the USA is ~0.4%).
- ***Disk drive failures*** are approximately ~1% per year.
- ***Factory production defect*** rates typically run about 0.1%.

What happens on classification?

- A classifier that always predict the most common class has an accuracy of 99.997%.





Evaluating Classifiers on Imbalanced Data

- When classes are slightly imbalanced, no balancing is need.
- Yet, take that into consideration when evaluating performances
- Assume the test set contains 100 records

  - Positive cases = 75, Negative cases = 25
  - Is a classifier with 70% accuracy good?
  - No, the trivial classifier (always positive) reaches 75%

  - Positive cases = 50, Negative cases = 50
  - Is a classifier with 70% accuracy good?
  - At least much better than the trivial classifier

Multiclass Problem

- Assume N classes
- If classes are perfectly balanced, a trivial classifier (e.g. majority) will yield $A_{trivial}$ ~100/N % accuracy
- N=2 → $A_{trivial}$ ~ 50%
- N=4 → $A_{trivial}$ ~ 25%
- Goodness of accuracy of a model should be compared against $A_{trivial}$
- E.g., If N=5, an accuracy of 40% would look large

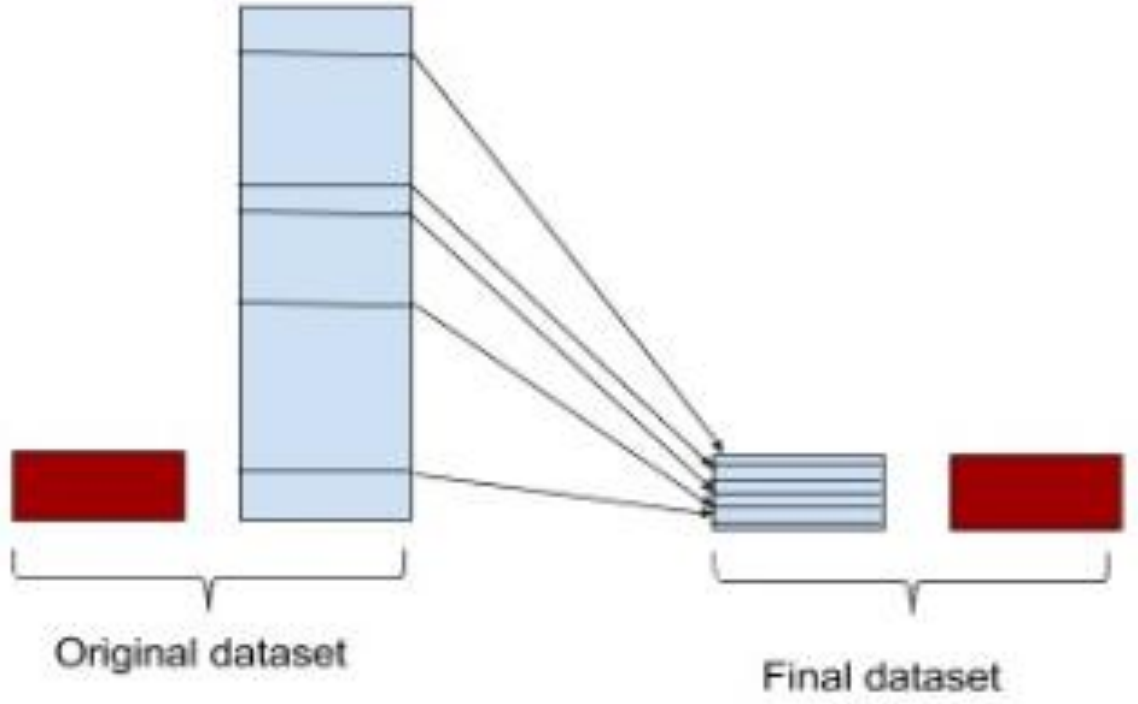
Handling Imbalanced Data

- Balance the training set
  - Undersampling the majority class
  - Oversampling the minority class
- Balance at the algorithm level
  - Adjust the class weight by making the algorithm more sensitive to rare classes
  - Adjust the decision threshold
  - Design new algorithm to perform well on imbalanced data
- Switch to Anomaly Detection
- Do nothing and hope to be lucky

Undersampling

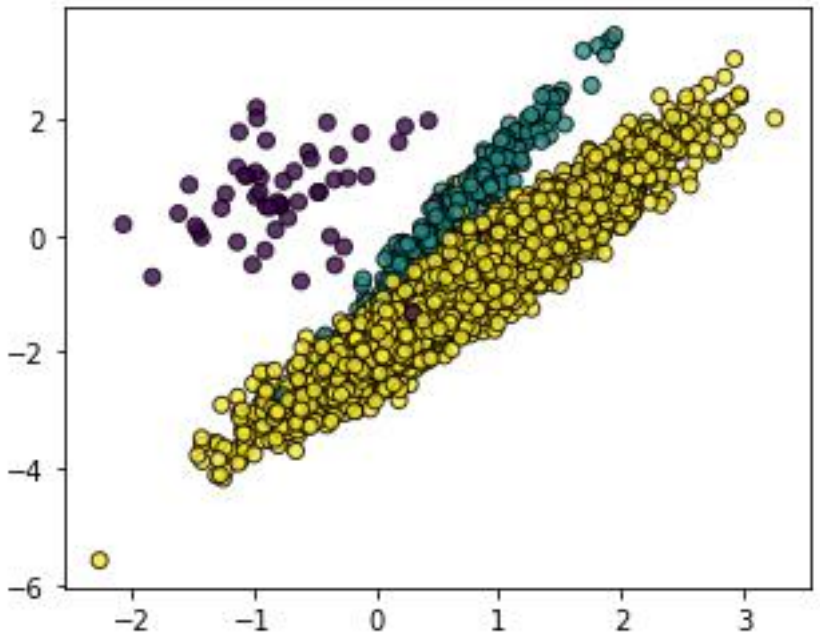
Undersampling the Majority Class

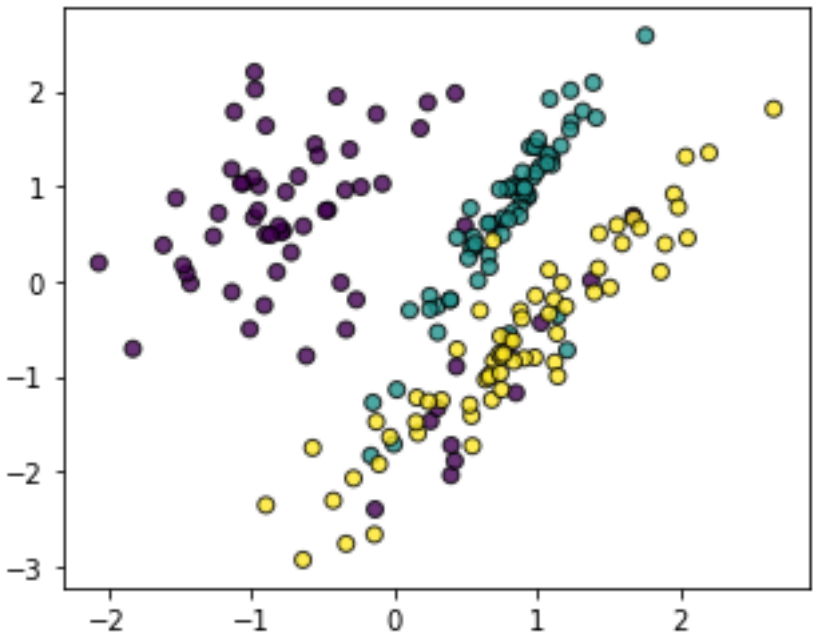
- Random Undersampling
- Neighbor-based approaches, e.g., Condensed Nearest Neighbor, Tomek Links, etc.



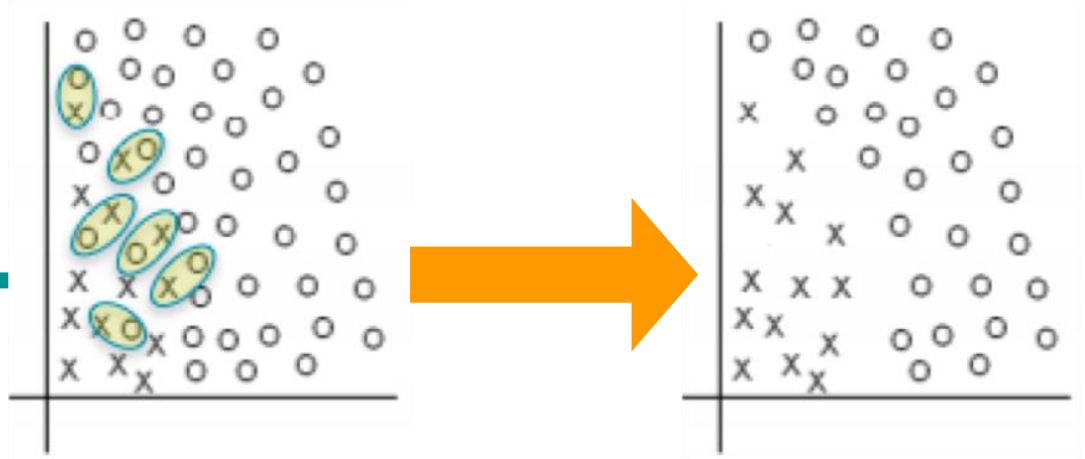
Random Undersampling

- Under-sample the majority class(es) by randomly picking samples with or without replacement.





Tomek Links



- Tomek Links uses a rule to selects the pair of observation (*a, b*) that respect these properties:
  - *a* nearest neighbor is *b*
  - *b* nearest neighbor is *a*.
  - *a* and *b* belong to a different class; minority and majority (or vice versa)
- Tomek Links are used to find samples of data from the majority class that is having the lowest distance with the minority class data (i.e., the data from the majority class that is closest with the minority class data, thus make it ambiguous to distinct), and then remove it.

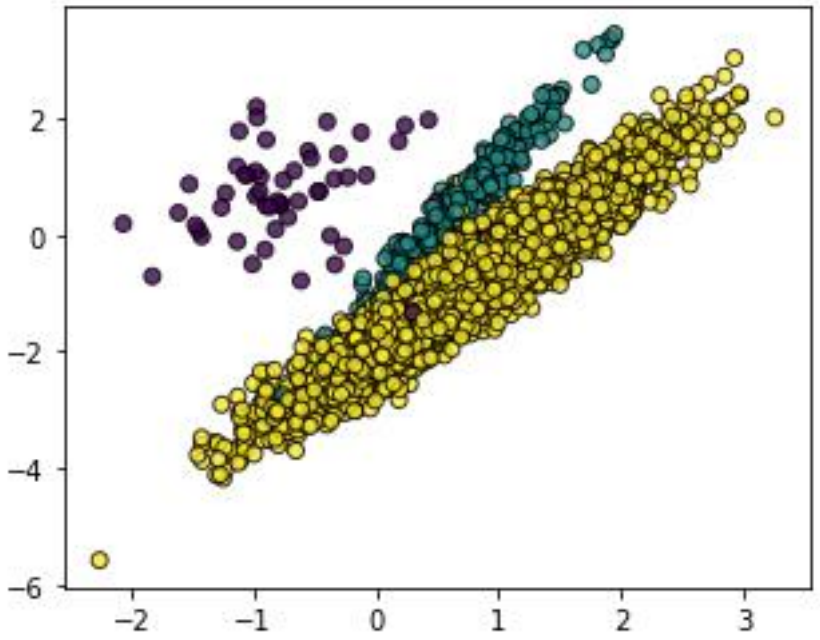
Edited Nearest Neighbor

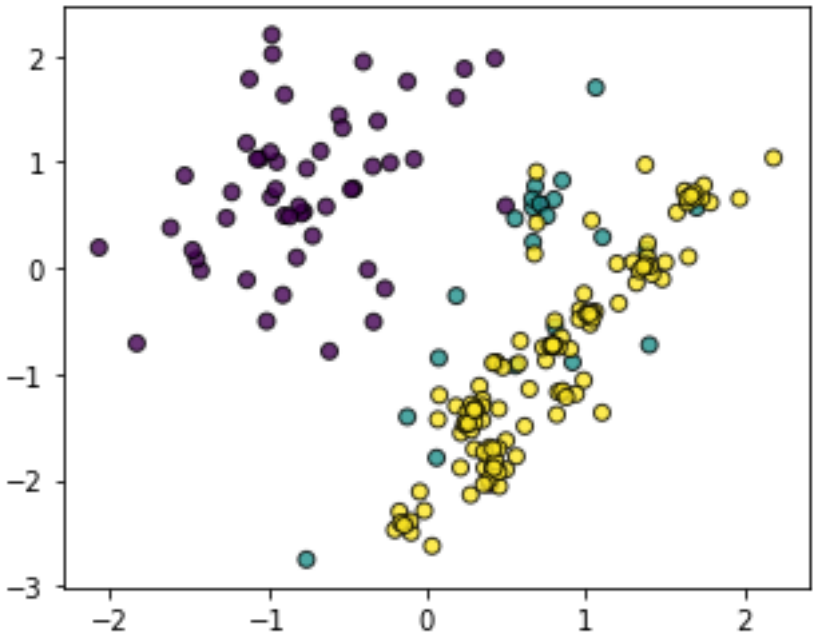
- Main idea: remove instances and neighbors for which a kNN fails.

- The ENN method works as follows
  - First, by finding the kNN of each observation
  - Then by checking whether the majority class from the observation's kNN is the same as the observation's class or not, i.e., if kNN is misclassifying or not
  - Then, if the majority class of the observation's kNN and the observation's class is different, the observation and its kNN are deleted from the dataset.

Condensed Nearest Neighbor

- Perform a smart undersampling by removing majority points having as k-NN a minority point, i.e., identify a subset which, when used as a stored reference set for kNN, correctly classifies all the remaining points in the sample set.





Condensed Nearest Neighbor

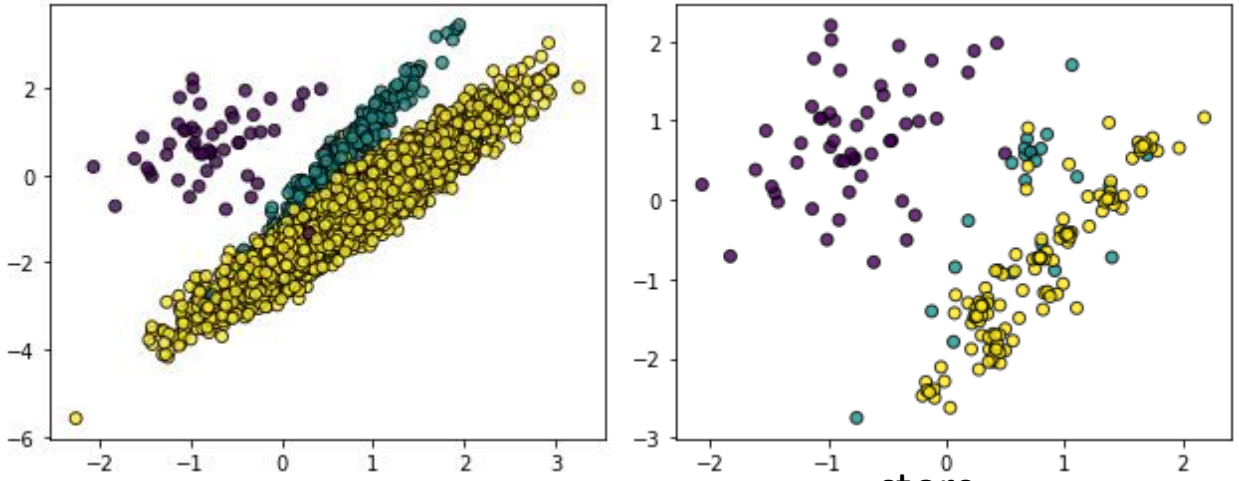
Objective: Enumerate the examples in the dataset X and add them to the "store" only if they cannot be classified correctly by the current contents of the store.

- **Store** ← {}
- REPEAT
  - FOR all *x* in *X* (in random order)
    - Find *x'* in **Store** such that *d(x, x') = min D(x, x'')*
    - IF *class(x)* is not equal *class(x')* THEN
      - Add *x* to **Store**
- UNTIL **Store** does not change

- **Store** used for classification instead of X.

Condensed Nearest Neighbor

P. Hart, “The condensed nearest neighbor rule,” In Information Theory, IEEE Transactions on, vol. 14(3), pp. 515-516, 1968



store

1) The first sample is placed in STORE.

2) The second sample is classified by the NN rule, using as a reference set the current contents of STORE. (Since STORE has only one point, the classification is trivial at this stage.) If the second sample is classified correctly it is placed in GRABBAG; otherwise it is placed in STORE.

3) Proceeding inductively, the ith sample is classified by the current contents of STORE. If classified correctly it is placed in GRABBAG; otherwise it is placed in STORE.

4) After one pass through the original sample set, the procedure continues to loop through GRABBAG until termination, which can occur in one of two ways:

a) The GRABBAG is exhausted, with all its members now transferred to STORE (in which case, the consistent subset found is the entire original set), or

b) One complete pass is made through GRABBAG with no transfers to STORE. (If this happens, all subsequent passes through GRABBAG will result in no transfers, since the underlying decision surface has not been changed.)

5) The final contents of STORE are used as reference points for the NN rule; the contents of GRABBAG are discarded.

Undersampling by Cluster Centroids

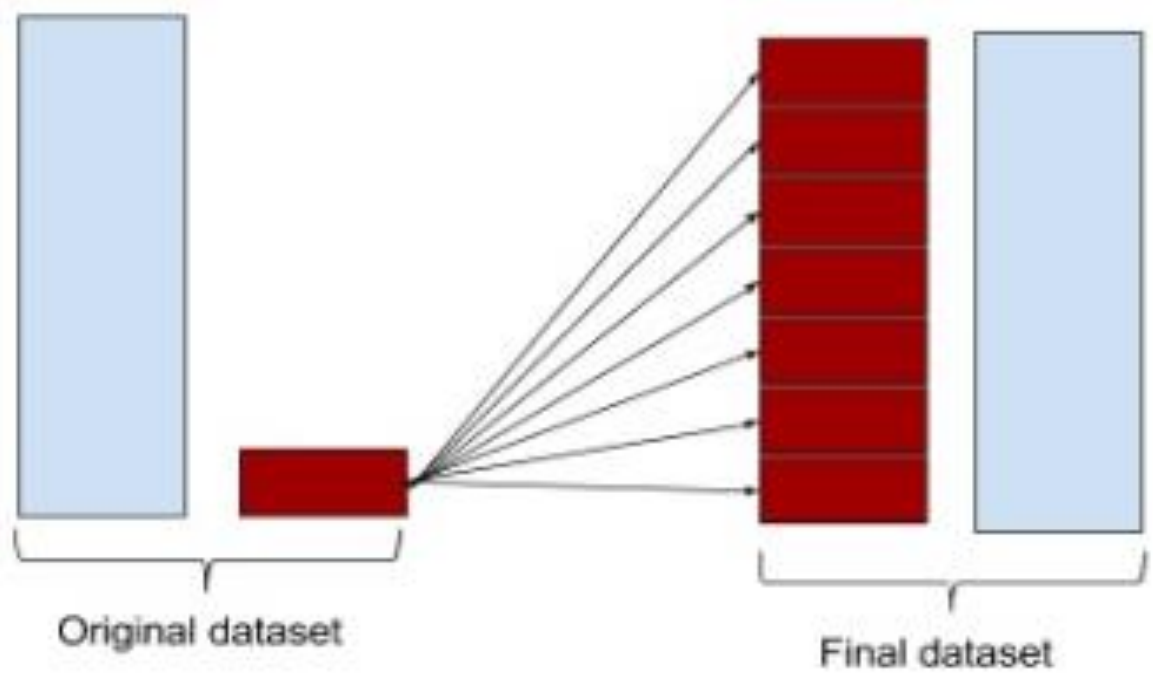
- Undersampling by generating centroids based on clustering methods.

- Under samples the majority class by replacing a cluster of majority samples by the cluster centroid of a KMeans algorithm.
- KMeans keeps K majority samples by fitting it with K cluster to the majority class and using the coordinates of the K cluster centroids as the new majority samples.

Oversampling

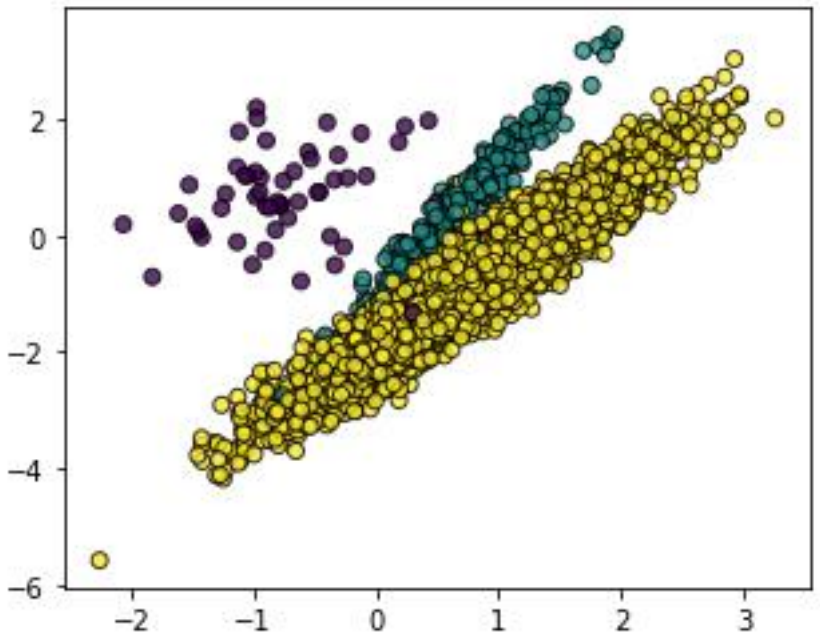
Oversampling the Majority Class

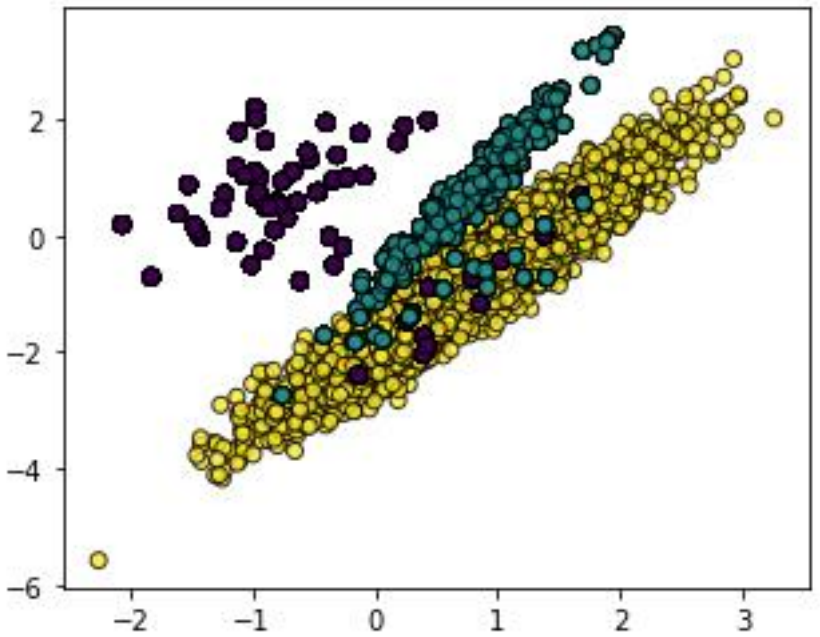
- Random Oversampling
- Synthetic Minority Oversampling Technique (SMOTE)
- Adaptive Synthetic (ADASYN) sampling approach



Random Oversampling

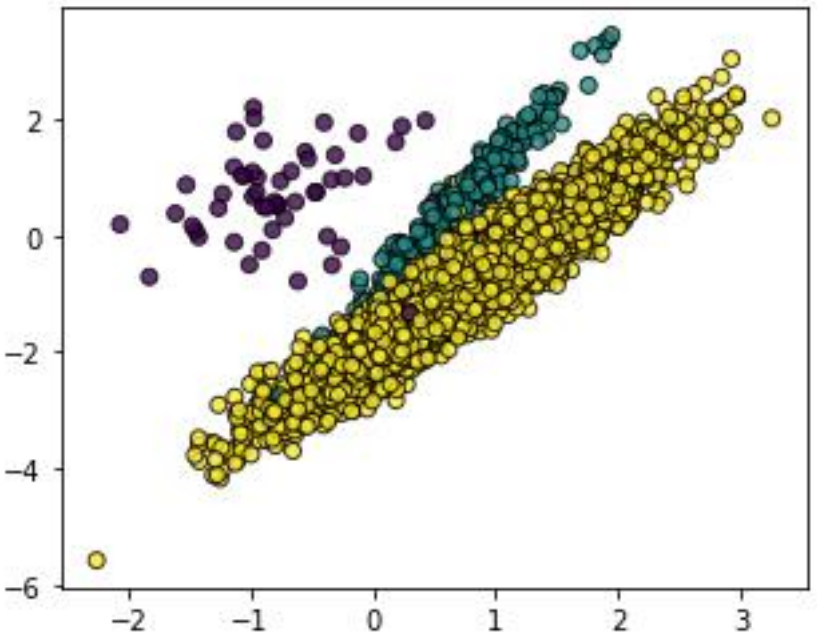
- Over-sample the minority class(es) by picking samples at random with replacement.

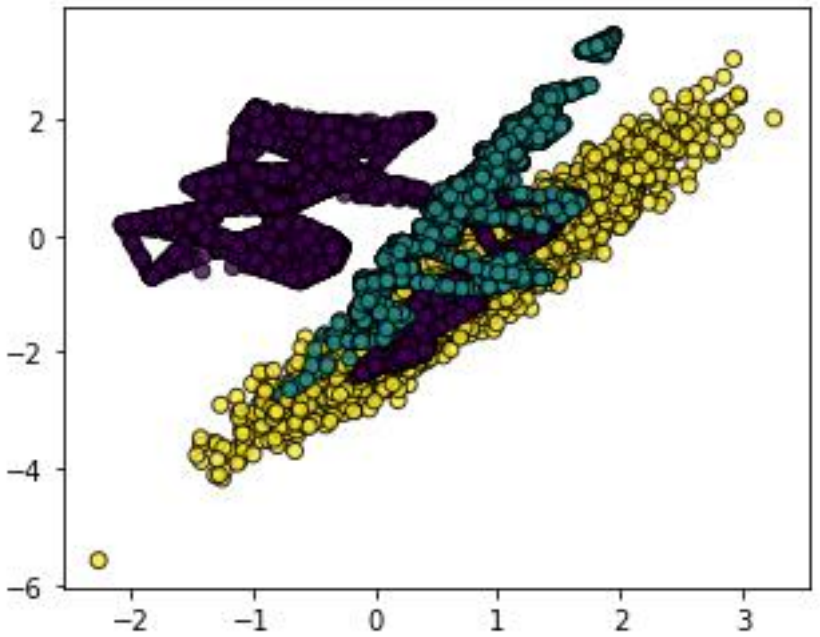




SMOTE Oversampling

- Over-sample the minority class(es) by adding points through interpolation.



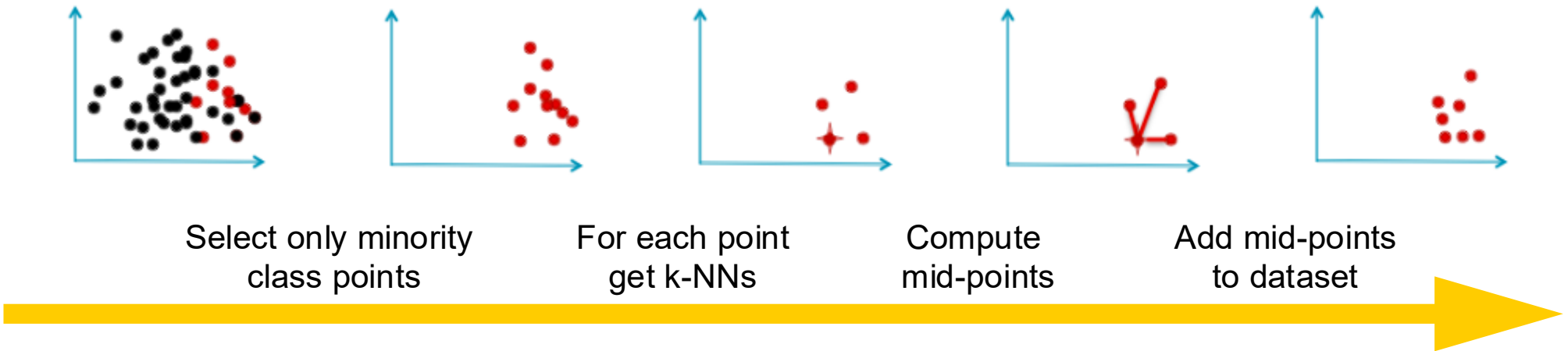


SMOTE

- It operates in the ***"feature space"*** rather than in the "data space", and effectively forces the decision region of the minority class to become more general.
- The minority class is over-sampled by taking each minority class sample and ***introducing synthetic examples along the line segments joining any/all of the k minority class nearest neighbors***.
- Depending upon the amount of over-sampling required, ***neighbors from the k nearest neighbors are randomly chosen*** (by default k=5).
- E.g., if the amount of over-sampling needed is 200%, only two neighbors from the five are chosen and one sample is generated in the direction of each.

SMOTE – Samples Generation

- Take the difference between the feature vector (sample) under consideration and its nearest neighbor.
- Multiply this difference by a random number between 0 and 1, and add it to the feature vector under consideration.
- This causes the selection of a random point along the line segment between two specific features.



SMOTE alternatives

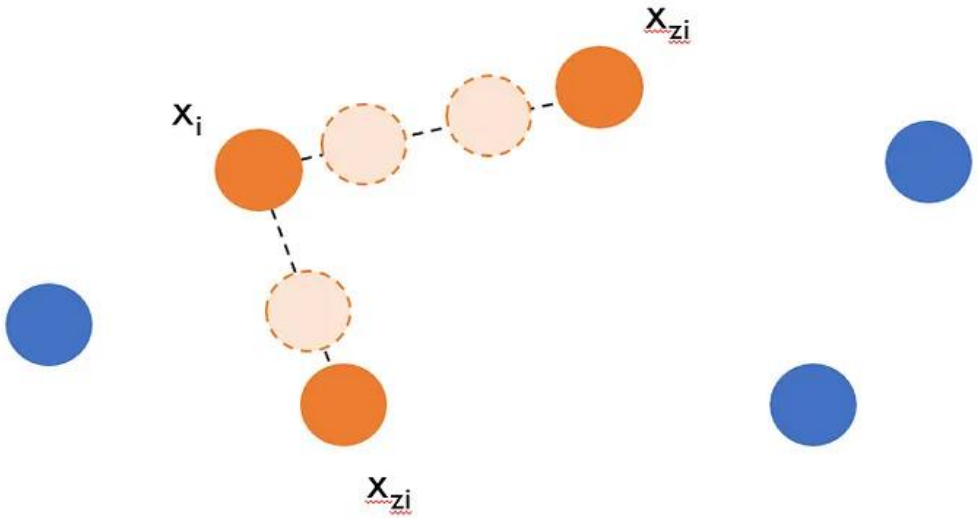
- SMOTENC: Over-sample for continuous and categorical features.
- BorderlineSMOTE: Over-sample using the borderline variant.
- SVMSMOTE: Over-sample using the SVM variant.
- ADASYN: Over-sample using ADASYN.

ADASYN - Adaptive Synthetic

- **Step1:** Calculate the ratio of minority to majority examples using *d= min/maj*
- **Step2:** Calculate the total number of synthetic minority to generate with *G = (maj-min)*β where β is the ratio of minority, i.e., with β=1 is required a perfectly balanced dataset
- **Step3:** Find the kNN of each minority sample and calculate the ratio of *maj* for the neighborhood over *k* as $r_i = maj_i/k$ and normalize it by dividing r_i for the sum of all the r_i.
- **Step4**: (Adaptive step) Calculate the number of synthetic samples to generate per neighborhood as $G_i = Gr_i$

ADASYN - Adaptive Synthetic

- **Step 5:** Generate G_i samples for each neighborhood by taking two minority samples (x_i, y_i) within the neighborhood and generating a synthetic one as SMOTE, i.e., $s_i = x_i + (y_i - x_i)\lambda$ where λ is a random number between 0 and 1.



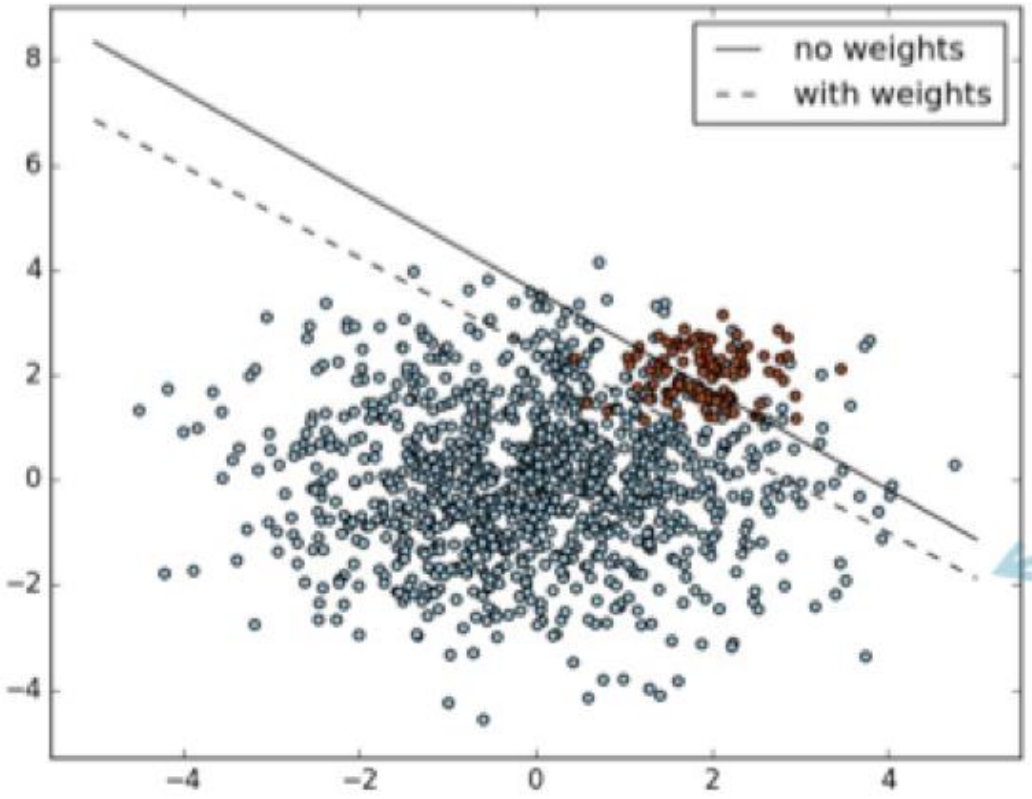
Undersampling & Oversampling

- In the literature there are many solution combining SMOTE with Undersampling approaches and using them in sequence in order to oversample the minority class while undersampling the majority one and mitigating the negative effects of these approaches.

Balancing at the Algorithm Level

Adjust the Class Weight

- The classifier can be trained considering **different costs** to be paid for misclassification errors on minority classes.
- This is generally done using a “class weight’’.



Adjust the Class Weight

- Each outcome with respect to a confusion matrix can be associated to a weight in a corresponding weight (or cost) matrix.
- Thus, the objective of the classification algorithm is to find the model that minimizes the total cost.
  - $\sum_X weight(x)freq(x)$

Confusion Matrix

		Actual	
		Y	N
Predicted	Y	50	7
	N	3	40

Weight Matrix

		Actual	
		Y	N
Predicted	Y	0	95
	N	5	0

Cost = 0.03*5 + 0.07*95

Meta-Cost Sensitive Classifier

- Apply a classifier getting probability of a class label *P(j|x)*
- Compute expected risk of classifying *x* with class *i*:

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

- Consider the train data with the class *i* having lower risk
- Learn a model on the cost-sensitive train data

Adjust the Decision Threshold

- Several classification methods compute scores in terms of probability of belonging to a class, and then assign class.
- Generally we have:
  - Score p > 50% → class = Y
  - Otherwise → class = N

- E.g.: decision trees have p = #positive/#negative cases over each leaf

Adjust the Decision Threshold

- What if we generalize the schema into:
  - Score p > THR% → class = Y
  - Otherwise → class = N
- For each THR (in [0-100]) we get a different set of predictions
- The confusion matrix changes and all indicators derived from it change
  - Accuracy
  - True Positive Rate (TPR)
  - False Positive Rate (FPR)
  - ….

References

- I. Tomek, “Two modifications of CNN,” In Systems, Man, and Cybernetics, IEEE Transactions on, vol. 6, pp 769-772, 2010.
- N. V. Chawla, K. W. Bowyer, L. O.Hall, W. P. Kegelmeyer, “SMOTE: synthetic minority over-sampling technique,” Journal of artificial intelligence research, 321-357, 2002.
- Domingos, Pedro. "Metacost: A general method for making classifiers cost-sensitive." Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining. 1999.
- P. Hart, “The condensed nearest neighbor rule,” In Information Theory, IEEE Transactions on, vol. 14(3), pp. 515-516, 1968.
- Python *imblearn* library: https://imbalanced-learn.readthedocs.io/en/stable/index.html