Imbalanced Data and Synthetic Generation

Lorenzo Mannocci

a.a. 2026/2027

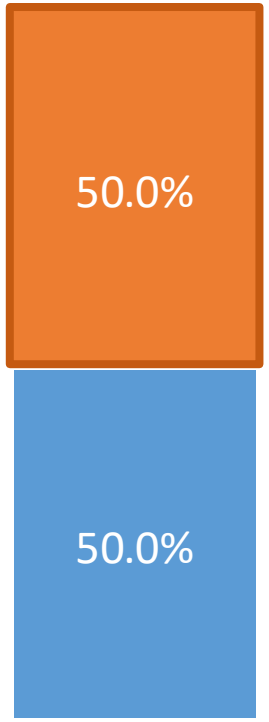


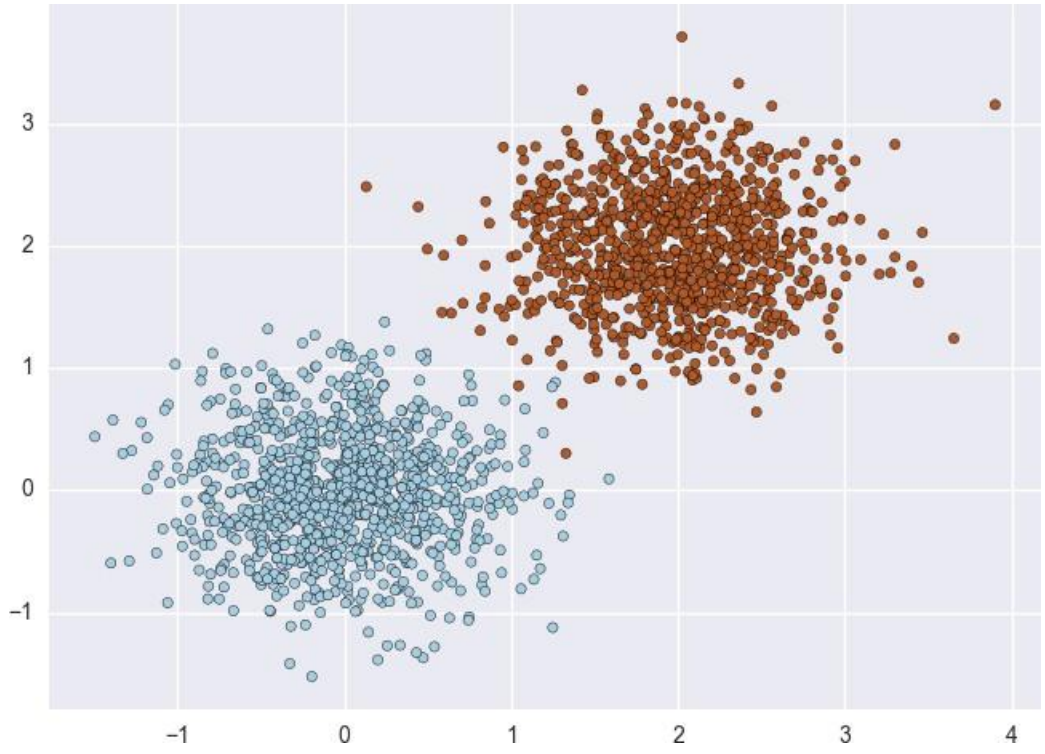
UNIVERSITÀ DI PISA

Imbalanced data

Imbalanced Classes

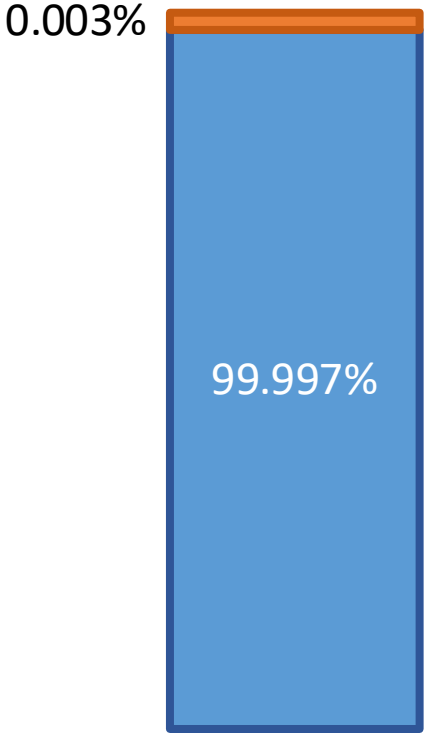
Most classification methods assume classes are reasonably balanced.

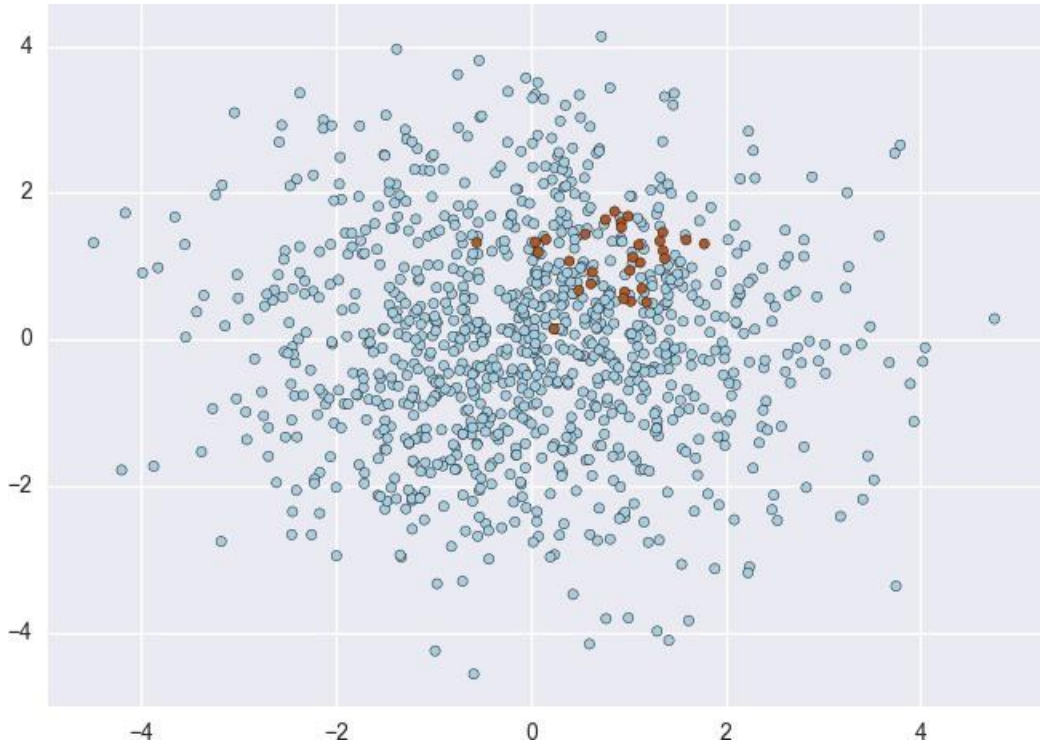




Imbalanced Classes

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.

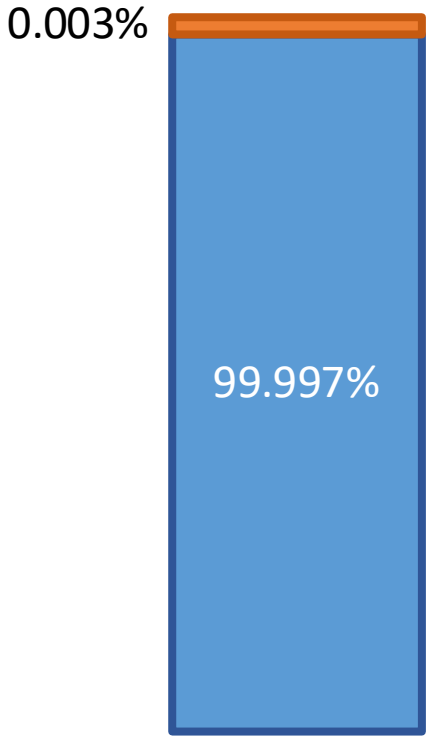
E.g. for 1M example dataset only about 30 represent an event.

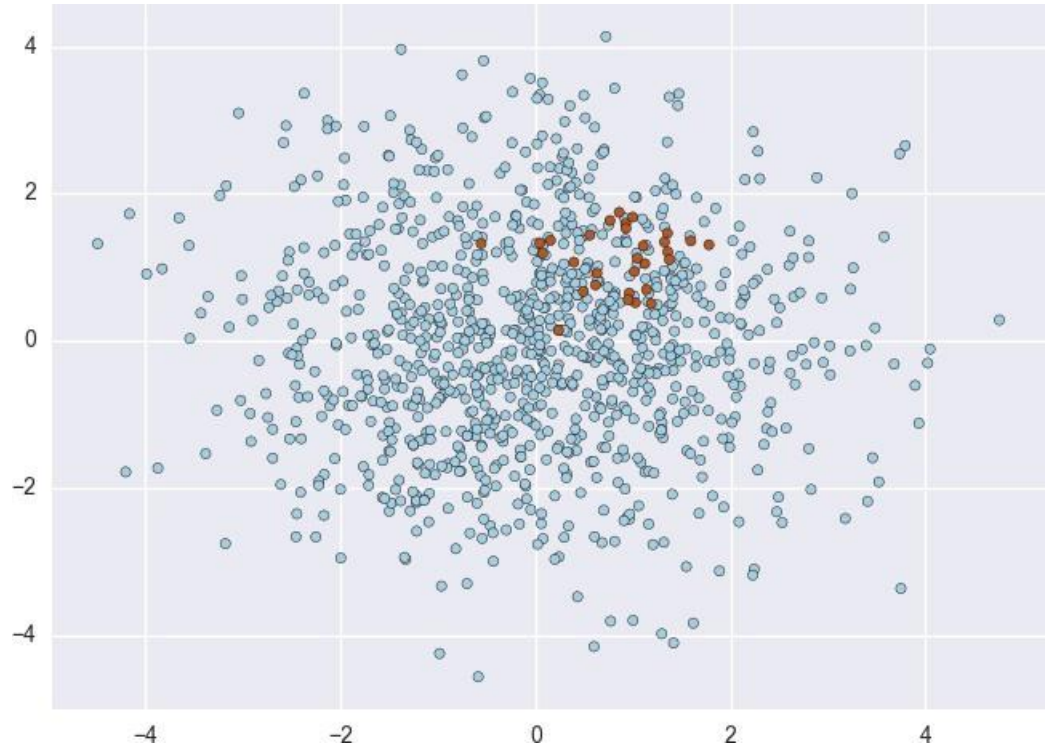
Examples

- 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?

?

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%

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?

?

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 $\rightarrow A_{trivial}$ ~ 50%
  - N=4 $\rightarrow A_{trivial}$ ~ 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

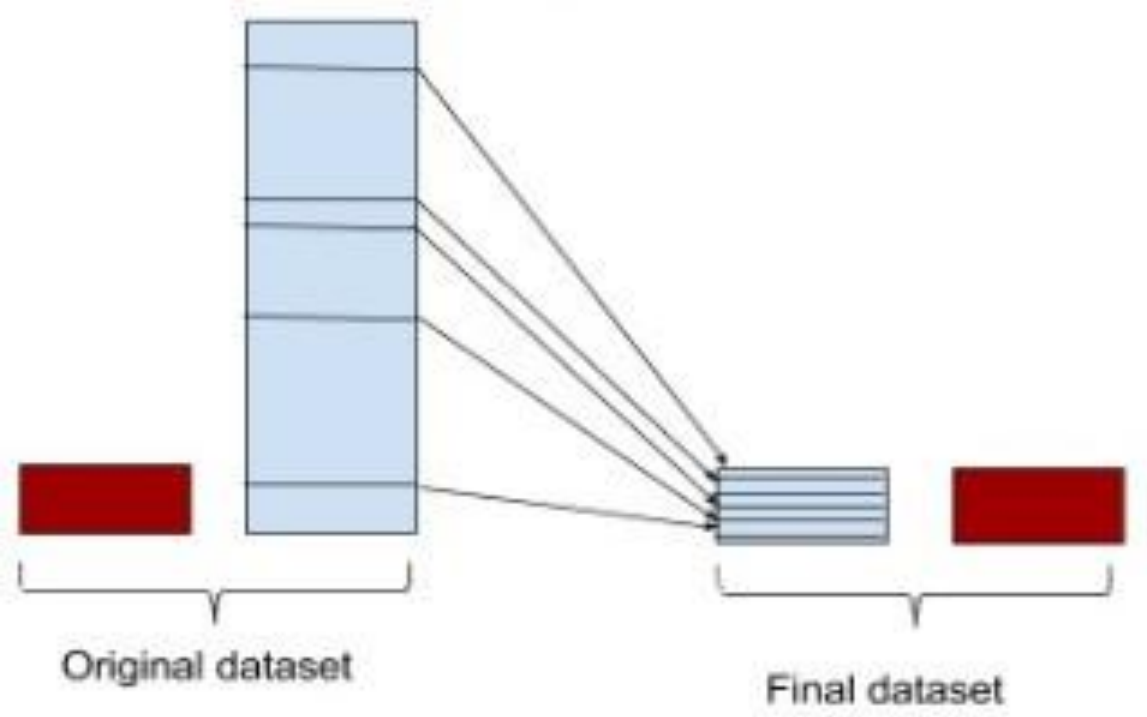
Hadnling Imbalanced Data

- Balance the training set
  - Undersampling the majority class
  - Oversampling the minority class
- 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

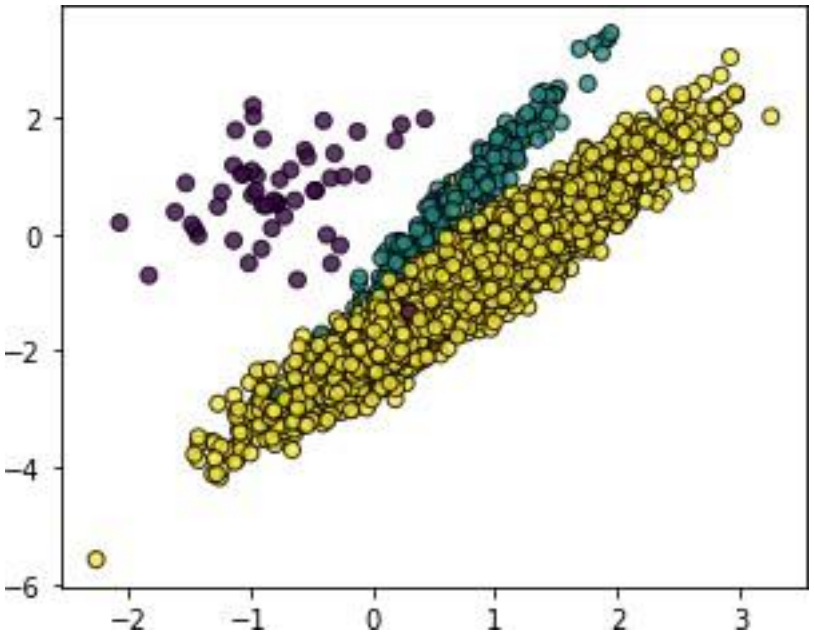
Undersampling Majority Class

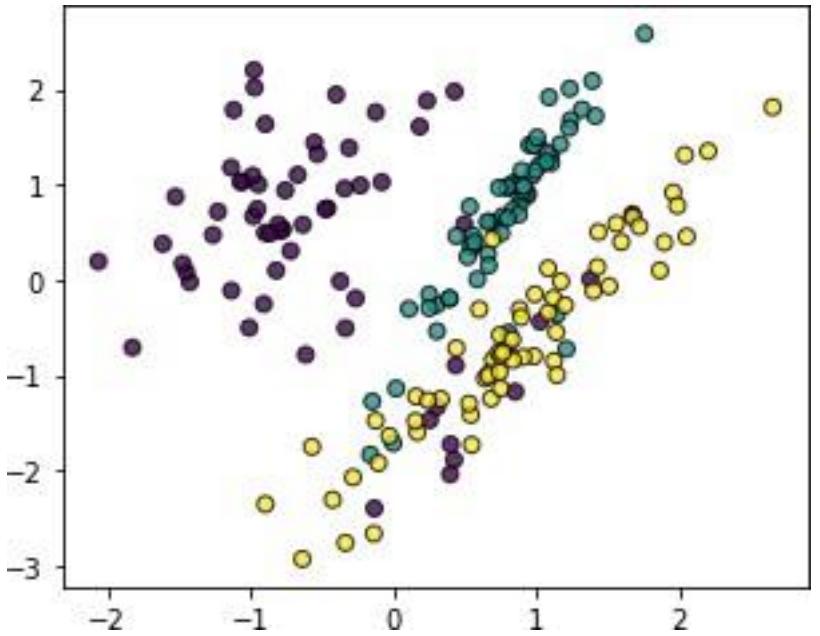
- Random Undersampling
- Neighbor-based approaches, e.g., Condensed Nearest Neighbor, Tomek Links, etc.



Random Undersampling

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

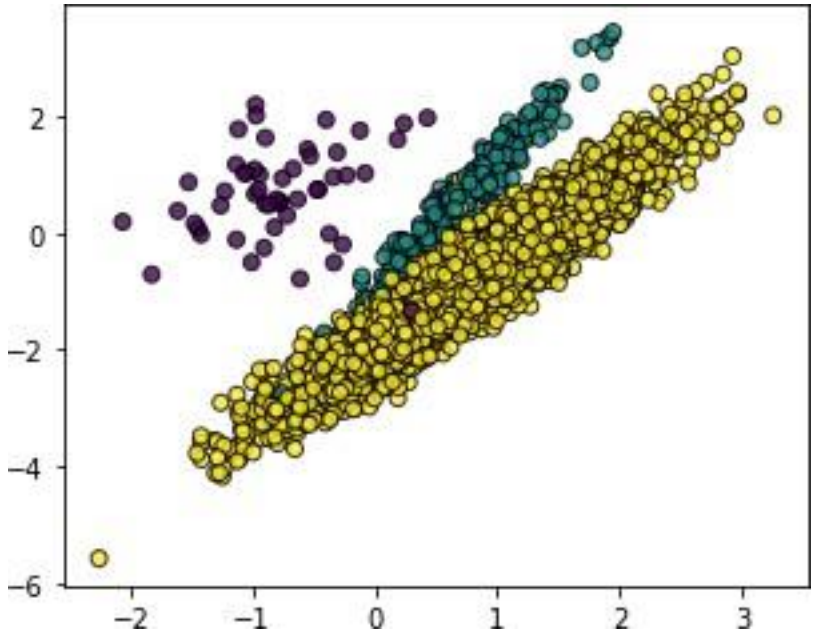


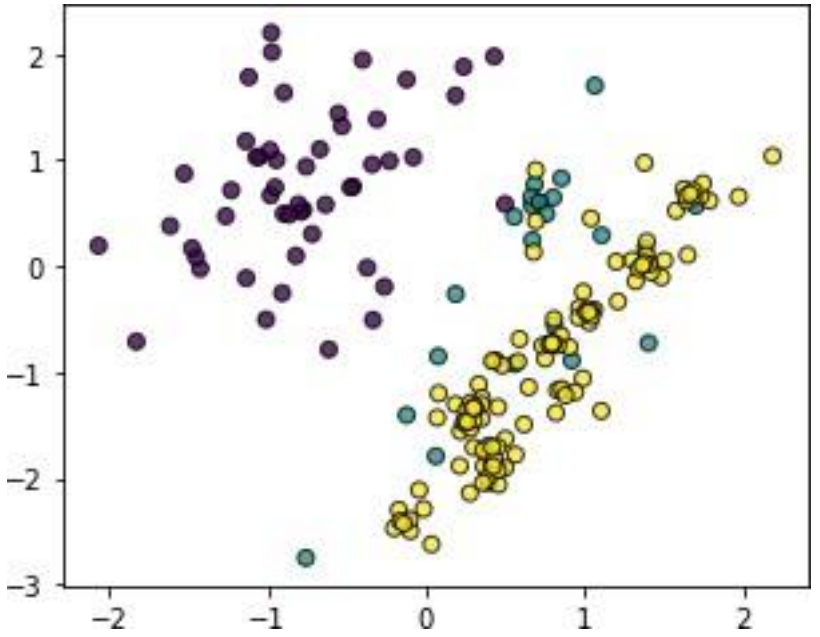


Condensed Nearest Neighbor

Condensed Nearest Neighbor (CNN)

Performs a smart undersampling by reducing the size of the majority class by keeping only the most informative instances needed to preserve the decision boundary.





Condensed Nearest Neighbor (CNN)

The algorithm maintains a subset of the original dataset, called the condensed set S.

1. Initialize the condensed set. Start with all minority-class observations and a small initial selection of majority-class observations (a common initialization in imbalanced-learn).

2. Classify the remaining observations. For each majority-class observation not currently in S, predict its class using a 1-NN classifier trained on S.

3. Identify misclassified observations. If an observation is correctly classified, it is not added to S. If it is misclassified, add it to S, because the current subset does not adequately represent that region.

4. Repeat. Continue checking observations and adding misclassified ones until a complete pass adds no new observations.

5. Return the condensed set. The final subset contains all retained minority observations and the majority observations needed to represent the classification structure.

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

Condensed Nearest Neighbor (CNN)

1. Imagine a majority-class observation located deep inside a region containing many other majority observations.
   - Its nearest neighbor in S will probably belong to the same class. Therefore, it is correctly classified and does not need to be added.
2. Now consider a majority-class observation close to the minority region.
   - If its nearest neighbor in S belongs to the minority class, it will be misclassified. CNN therefore adds it to S, improving the representation of that region.

Consequently, CNN tends to:

- Remove redundant majority observations in homogeneous regions.
- Retain informative majority observations, often near class boundaries.
- Preserve minority observations when used as an undersampling method.

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

Condensed Nearest Neighbor (CNN)

Strenghts

- Reduces majority-class storage and speeds up later k-NN classification.
- Keeps informative samples close to the decision boundary.
- Does not synthesize artificial observations.

Limits

- Sensitive to noisy or mislabeled examples, which can be retained.
- The result depends on order, metric, and the chosen nearest-neighbor rule.
- The retained set may still be imbalanced and may remove useful density information.

Best understood as boundary preservation rather than random deletion.

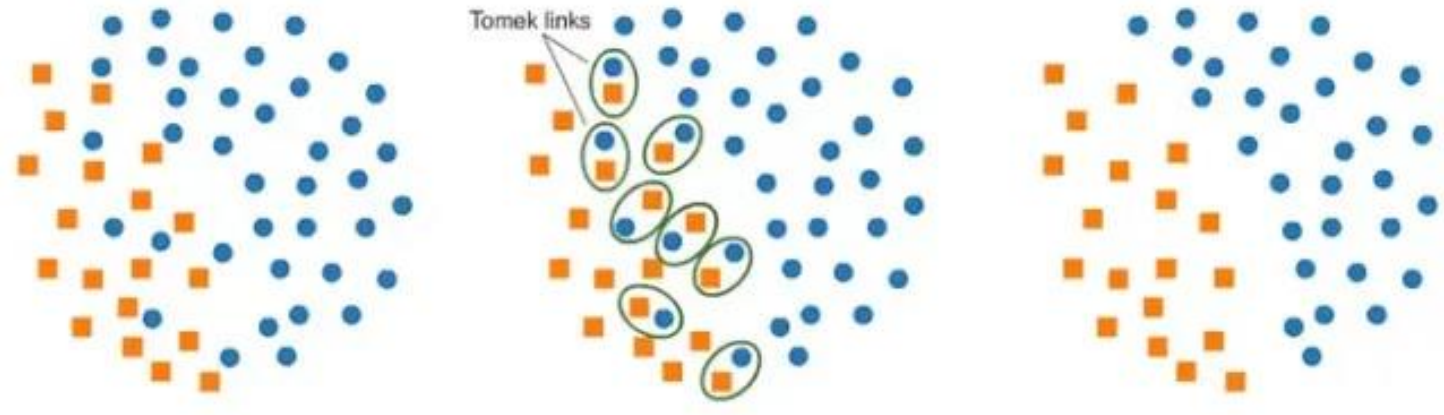
Tomek’s Links

Tomek's Links

A Tomek Link is a pair of instances belonging to different classes that are each other's nearest neighbor.
Formally, two samples x_i and x_j form a Tomek Link if:

1. $y_i \neq y_j$ (different classes)
2. x_j is the nearest neighbor of x_i
3. x_i is the nearest neighbor of x_j

In other words, they are mutual nearest neighbors from opposite classes.



Tomek's Links

A Tomek Link indicates that the samples lie in an overlapping or ambiguous region between the two classes.

Input

Training dataset D

Procedure

1. Compute the nearest neighbor of every sample.
2. Identify all Tomek Links
3. For each Tomek Link:
    a) usually remove the majority-class sample.
4. Return the cleaned dataset.

Tomek's Links

Advantages

- Very simple.
- Removes borderline majority examples.
- Cleans noisy datasets.
- Preserves the minority class.
- Clearer decision boundaries, enabling the model to learn more effectively.
- Frequently combined with oversampling methods (e.g., SMOTE)

Limits

- Does **not** balance the dataset by itself.
- Removes relatively few samples.
- May remove informative boundary points.
- Performance depends on the distance metric.

Tomek's Links

SMOTE Generate synthetic minority samples. Some synthetic samples may overlap with majority samples.

Tomek Links Remove majority samples (and optionally synthetic minority samples) involved in Tomek Links.

The result is:

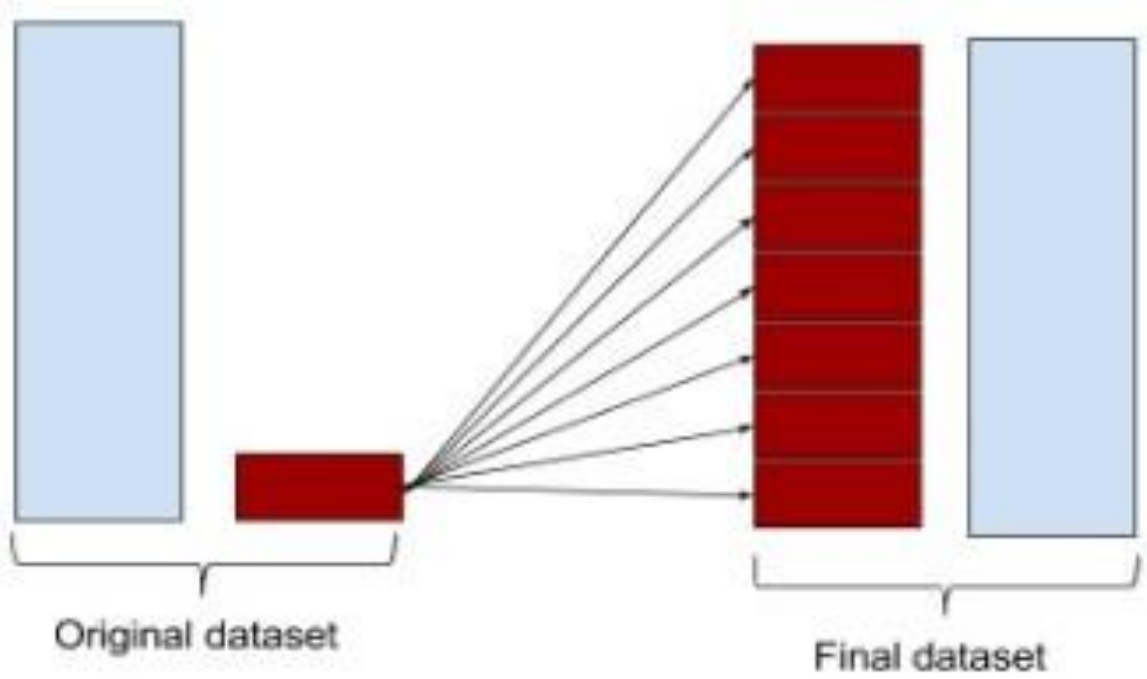
- more balanced data,
- cleaner decision boundaries,
- fewer ambiguous training examples.

This combination, often called SMOTE + Tomek, is implemented directly in the *imbalanced-learn* library and is one of the standard preprocessing pipelines for imbalanced classification.

Oversampling

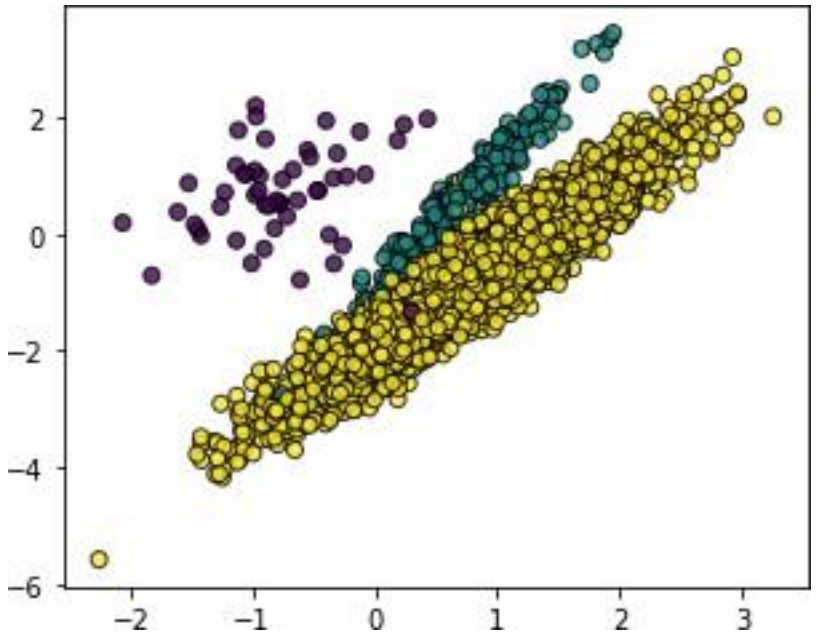
Oversampling Majority Class

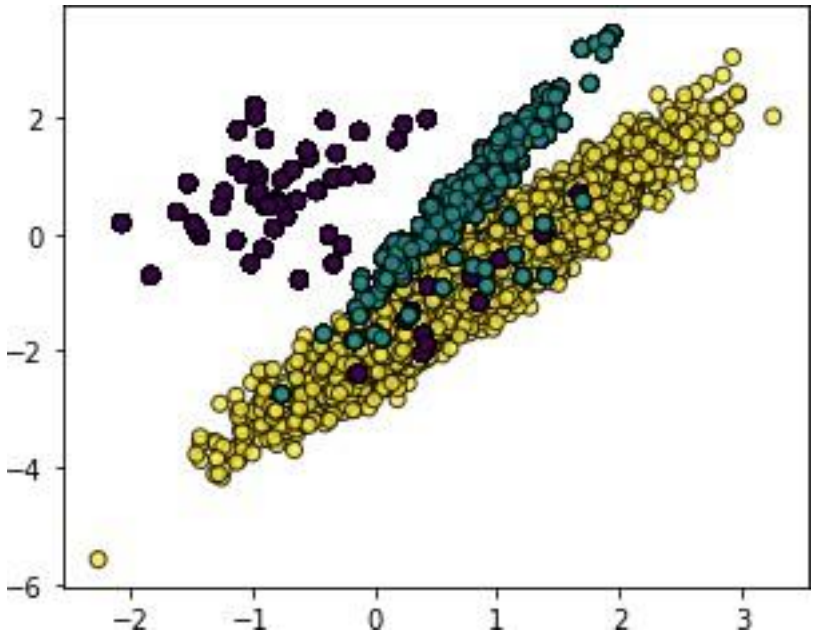
- Random Oversampling
- Synthetic Minority Oversampling Technique (SMOTE)
- Adaptive Synthetic (ADASYN) sampling approach



Random Oversampling

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

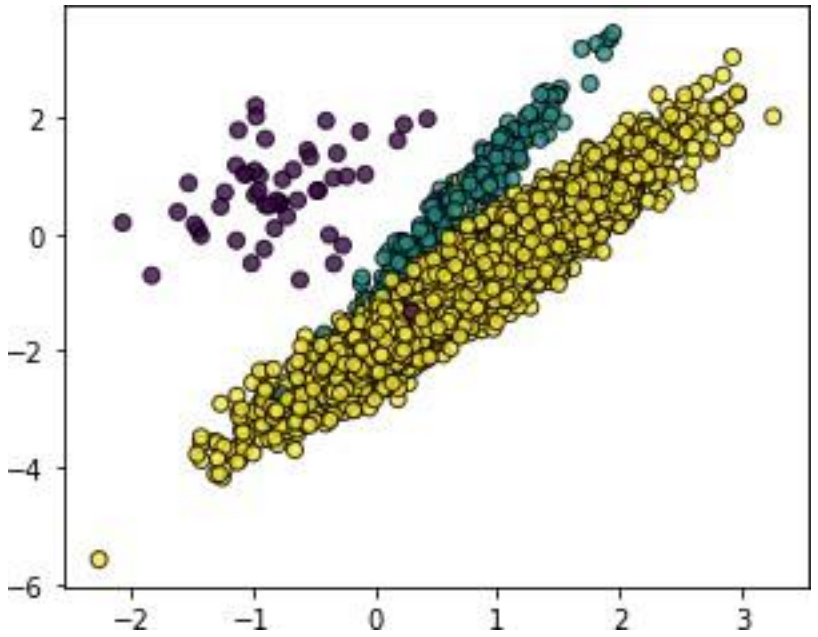


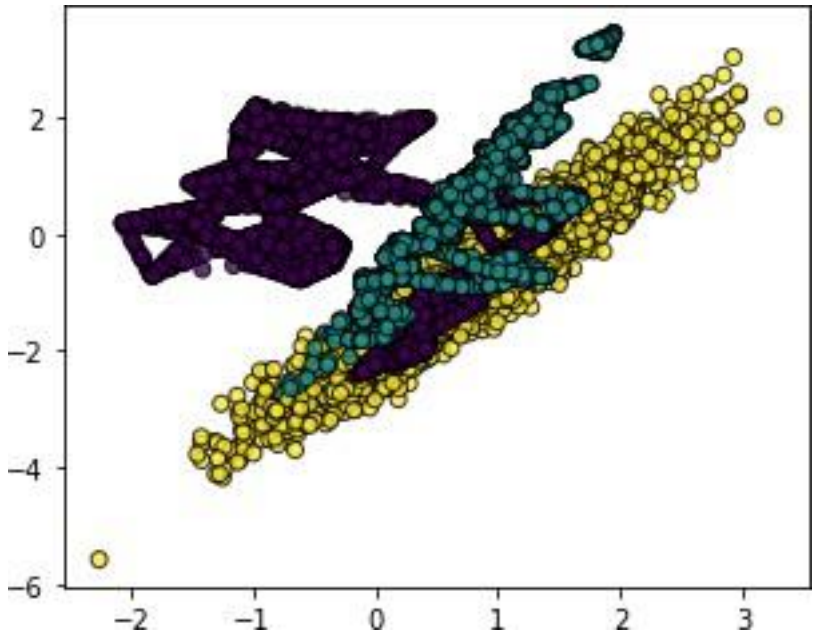


SMOTE

SMOTE Oversampling

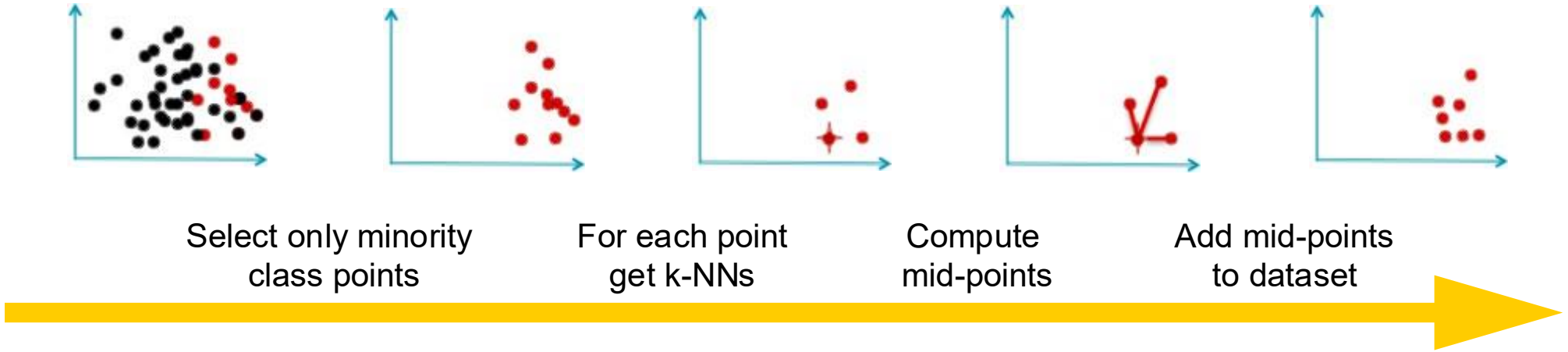
Over-sample the minority class(es) by adding points through interporlation.





SMOTE Oversampling

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



SMOTE Oversampling

- 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.

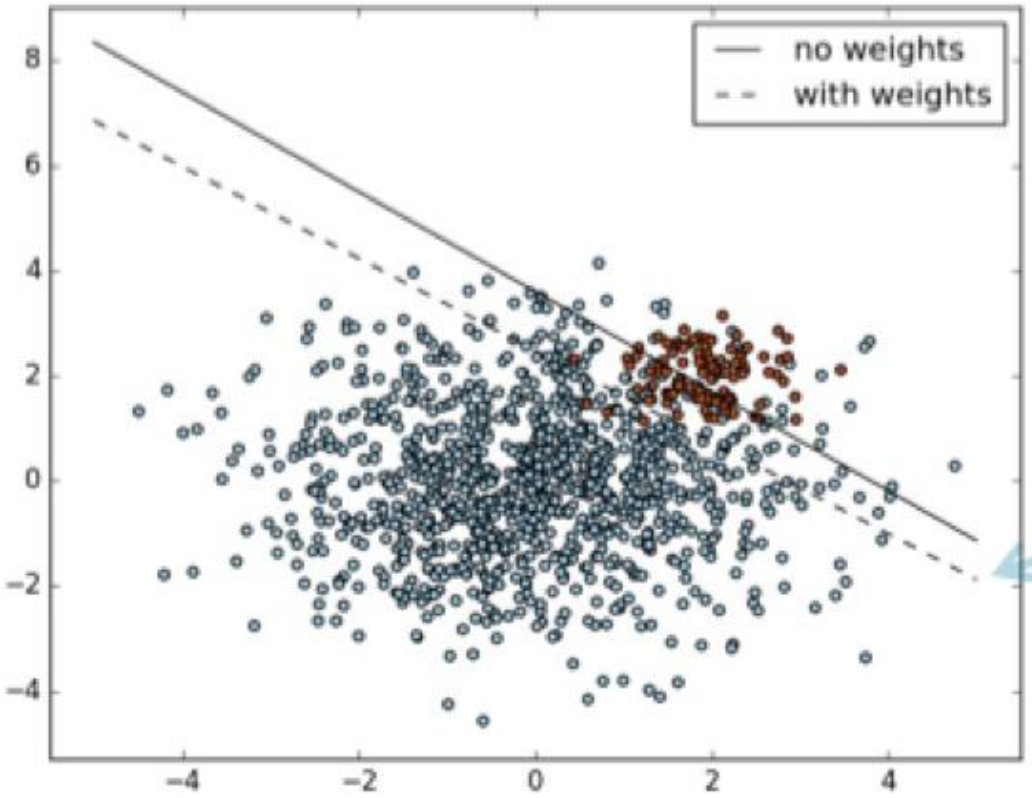
SMOTE Alternatives

- 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.

Class weights

Class Weights

- 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".



Class Weights

- 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

Synthetic data generation

Discriminative vs Generative Models

Given a distribution of inputs X and labels Y.

Discriminative models

- Learn conditional distribution $p(Y|X)$
- Learns decision boundary between classes.
- Limited scope. Can only be used for classification tasks.
- E.g., Logistic regression, SVM…

Generative models

- Learn the joint distribution $p(Y, X)$
- Learns actual probability distribution of data.
- Can do both generative and discriminative tasks.
- E.g. Naïve Bayes, Gaussian Mixture Model…

Synthetic Data Generation

Recent methods learn the minority distribution instead of simply interpolating between existing observations.

Method	How are new samples generated?
Random Oversampling	Duplicate existing observations
SMOTE	Interpolate between neighboring minority samples
GAN	Learn the overall data distribution through adversarial training
CTGAN	Learn the distribution of tabular data while accounting for mixed data types and rare categories

Generative Adversarial Network

Generatvive Adversarial Networks

Generate new observations that follow the same statistical distribution as the original data.

"GANs and the variations that are now being proposed is the most interesting idea in the last 10 years in ML, in my opinion"

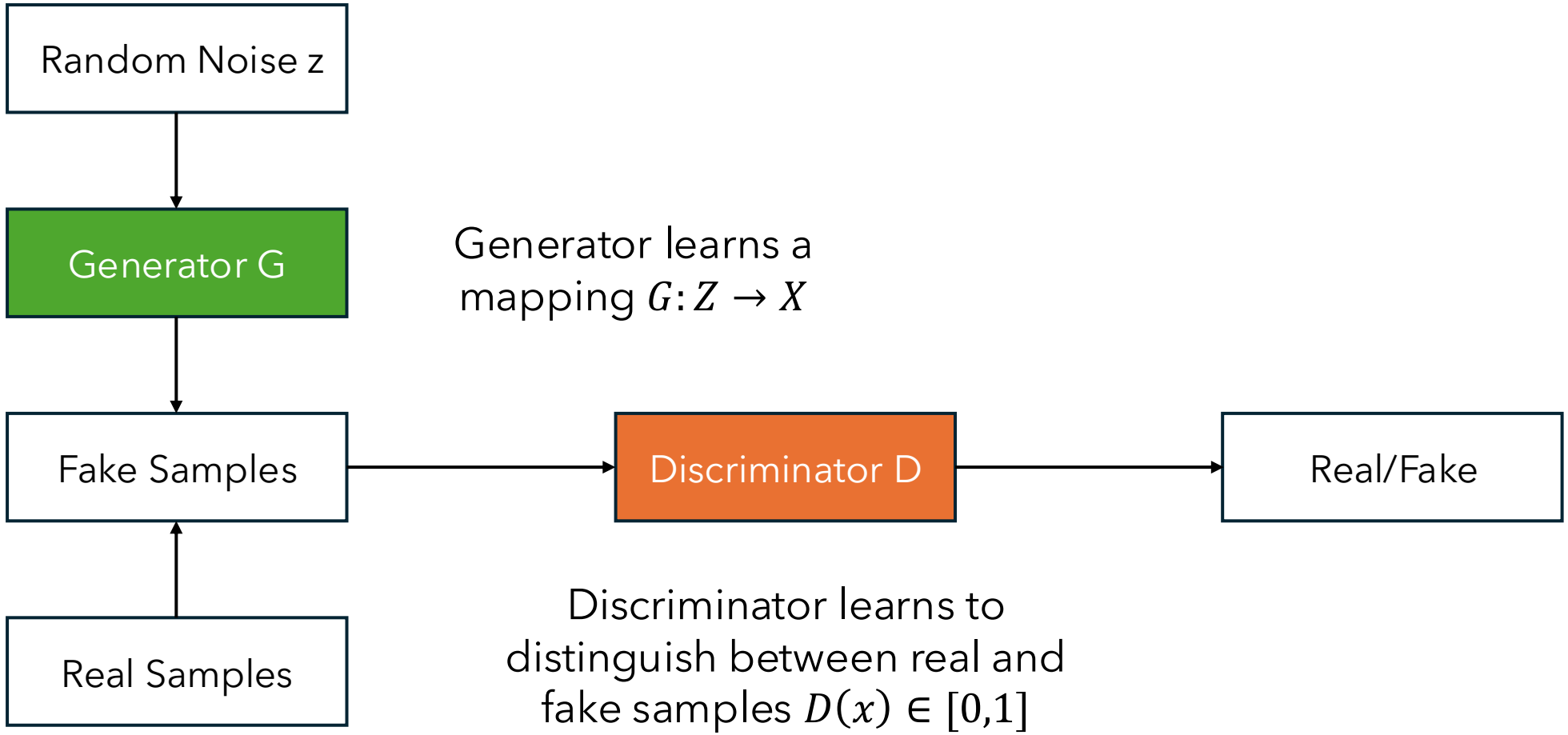
–Yann LeCun



Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2020). **Generative adversarial networks**. *Communications of the ACM*, 63(11), 139-144.

Generative Adversarial Networks

We try to learn the underlying the distribution from which our dataset comes from.



Both networks improve by competing against each other.

Original GAN Objective

The Generator and the Discriminator are trained through a **minimax game**.
The Discriminator (D) tries to **maximize** the objective, while the Generator (G) tries to **minimize** it: $\min_G \max_D V(D,G)$

The value function is: $V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{z\sim p(z)}\left[\log\left(1 - D\left(G(z)\right)\right)\right]$

Where:

- $\mathrm{x}\sim p_{data}$: real samples from the training data
- $\mathrm{z}\sim p_{data}$: random latent noise
- $\mathrm{G(z)}$: synthetic samples generated by the Generator
- $\mathrm{D(X)}$: probability that x is classified as real

The Discriminator D tries to **maximize** $V(D,G)$, while the Generator G tries to **minimize** it.

In practice, the Generator and the Discriminator are optimized separately.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2020). **Generative adversarial networks**. *Communications of the ACM*, 63(11), 139-144.

GAN: Discriminator

We convert the maximization problem into a minimization problem by changing the sign of the objective:

$$L_D = -\mathbb{E}_{x\sim p_{\text{data}}}[\log D(x)] - \mathbb{E}_{z\sim p(z)}[\log(1 - D(G(z)))]$$

Real samples

For every real sample x, the discriminator outputs $D(x) \in [0,1]$.

- If the discriminator correctly identifies a real sample, $D(x) \approx 1. \rightarrow \log(1) = 0$, so the loss becomes very small.
- If instead $D(x) \approx 0 \rightarrow \log(0) \rightarrow -\infty$, so the loss becomes very large.

The discriminator is rewarded for assigning high probabilities to real samples.

Fake samples

For fake samples we want:

- $D(G(z)) \approx 0 \rightarrow 1 - D(G(z)) \approx 1 \rightarrow \log(1) = 0.$ Therefore, the loss is minimized.
- If instead the discriminator believes fake samples are real, $D(G(z)) \approx 1$, sothe loss becomes very large.

The discriminator is rewarded to classify generated samples as fake.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2020). **Generative adversarial networks**. *Communications of the ACM*, 63(11), 139-144.

GAN: Generator

Generator tries to maximise $D(G(z))$, i.e., it want fake samples to be classified as real.

When we optimize the Generator, we keep the Discriminator fixed:

$$L_G = E_{(z \sim p(z))}\left[log\left(1 - D(G(z))\right)\right]$$

Fake samples

For fake samples Generator want:

- $D(G(z)) \approx 1 \rightarrow -\log(1) = 0$. It wants the discriminator to believe that generated samples are real.

Notice that only fake samples appear here.

The generator never sees real samples directly.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2020). **Generative adversarial networks**. *Communications of the ACM*, 63(11), 139-144.

GAN: Generator and Saturation Problem

At the beginning of training, the Generator usually produces very poor samples. As a consequence, the Discriminator can easily recognize them as fake. Therefore: $D\big(G(z)\big) \to 0$

With the original Generator loss, this situation can lead to very small gradients.
A small gradient means that the parameters of the Generator change only slightly during optimization.

This creates a problem:
the Generator is performing badly, but it receives only a weak signal telling it how to improve.
This phenomenon is called gradient saturation.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2020). **Generative adversarial networks**. *Communications of the ACM*, 63(11), 139-144.

GAN: Generator and Saturation Problem

$D(G(z))$	**Interpretation**	**Original Generator loss** $\log\left(1 - D\left(G(z)\right)\right)$	**Learning signal**
0.001	Discriminator is almost certain the sample is fake	≈ -0.001	Very weak
0.01	Sample is clearly recognized as fake	≈ -0.010	Weak
0.10	Generator is starting to fool the discriminator	≈ -0.105	Stronger
0.50	Discriminator is uncertain	≈ -0.693	Good
0.90	Generator is very successful	≈-2.303	Strong objective effect

When $D\left(G(z)\right) \longrightarrow 0$, which is common at the beginning of training, the original Generator objective changes very little. As a result, the gradient passed back to the Generator can become very small.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2020). **Generative adversarial networks**. *Communications of the ACM*, 63(11), 139-144.

GAN: Generator and Non-Saturating Loss

To avoid the saturation problem, the Generator objective is modified. Instead of minimizing: $log\left(1 - D\left(G(z)\right)\right)$, the Generator minimizes: $log\left(D\left(G(z)\right)\right)$
This is called the non-saturating Generator loss.
The objective remains the same, the difference is in the strength of the gradient.

As a result, the Generator receives a stronger learning signal when its samples are poor.

$$L_G^{\mathrm{NS}} = -E_{(z\sim p(z))}\left[log\left(D\left(G(z)\right)\right)\right]$$

Non-saturating Generator loss

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2020). **Generative adversarial networks**. *Communications of the ACM*, 63(11), 139-144.

Challeneges of generating tabular data

Age	Income	Education	Country	Default
35	42,000	Bachelor	Italy	No
51	87,000	Master	France	No
22	25,000	High School	Italy	Yes

Can we simply train a standard GAN to generate rows of this table?

Challenge 1 – Mixed data types

A row may simultaneously contain continuous, binary, ordinal and categorical variables.
A neural network naturally produces continuous numerical outputs, while something such as *Country = Italy* is discrete.

Challenge 2 – Multimodal continuous distributions

Continuous variables are not necessarily Gaussian or unimodal.
For example, *income* might contain several groups: $p(income)$could have peaks corresponding roughly to low-, medium-, and high-income populations. A simple normalization of the entire column may make this distribution difficult for the Generator to learn.

Challenge 3 – Imbalanced categorical variables

During ordinary random training, the GAN sees Italy much more frequently than Iceland.
Consequently, the Generator may learn common categories well but poorly represent rare ones.

Conditional Tabular GANs

Conditional Tabular GANs (CTGAN)

Standard GAN: $G(z)$The Generator receives only random noise.

A Conditional GAN also receives some information c: $G(z, c)$
So instead of saying: "Generate a sample.", we can say: "Generate a sample satisfying condition c."

$c = Country = Iceland$ and the Generator learns to generate observations compatible with that condition.

It adapts conditional GANs specifically to the statistical characteristics of tabular data.

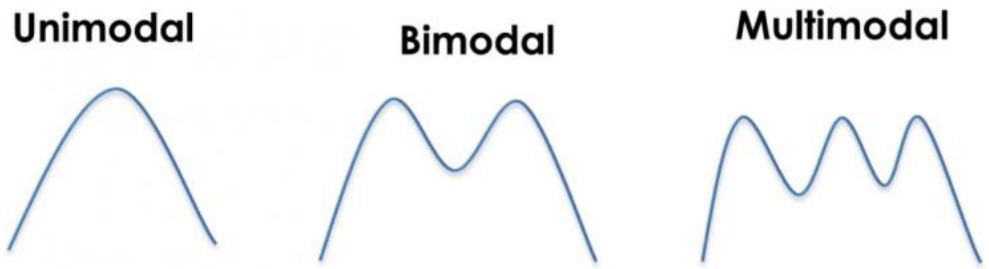
1. Mode-specific normalization: Handles complex/multimodal continuous variables.
2. Conditional generator: Allows generation conditioned on categorical values.
3. Training-by-sampling: Ensures that rare categories receive sufficient attention during training.

Xu, L., Skoularidou, M., Cuesta-Infante, A., & Veeramachaneni, K. (2019).
Modeling tabular data using conditional gan. *Advances in neural information processing systems*, 32.

Conditional Tabular GAN (CTGAN)

1. Mode-specific normalization

Handles complex/multimodal continuous variables.



Take a continuous variable such as income with several modes.

Instead of representing income simply as one normalized number,
CTGAN first models its distribution as a mixture of modes.

This makes multimodal distributions easier for the neural network to reproduce.

Conditional Tabular GANs

2. Conditional generator

Allows generation conditioned on categorical values.

$$Education = \{High\ School, Bachelor, Master\}$$

CTGAN can select: *Education* = *Master* and provide this information to the Generator.

The Generator therefore learns:

$$p(x \mid Education = Master)$$

rather than only the unconditional distribution $p(x)$.

Conditional Tabular GANs

3. Training-by-sampling
Ensures that rare categories receive sufficient attention during training.

If *Master* represents only 3% of the dataset, ordinary random sampling rarely selects it.

CTGAN deliberately samples categorical values during training so that rare categories are adequately represented.

CTGAN uses a log-frequency-based sampling strategy rather than simply making all categories equally frequent.

Conditional Tabular GANs

CTGAN learns the distribution of tabular data through adversarial training. Once trained, the Generator can produce new synthetic observations.

Generation phase

1. Sample a random noise vector z and a conditional vector c.
2. The trained Generator produces a synthetic observation:

$$x_{\text{synthetic}} = G(z, c)$$

Repeat to generate the desired number of synthetic rows.

For imbalanced-classification practical, CTGAN must be trained only on minority-class observations and then generate additional rows. This is a particular application of CTGAN, not a requirement of the algorithm.

Variational Autoencoder

A Different Way to Generate Data

CTGAN generates data through an adversarial game between a Generator and a Discriminator.

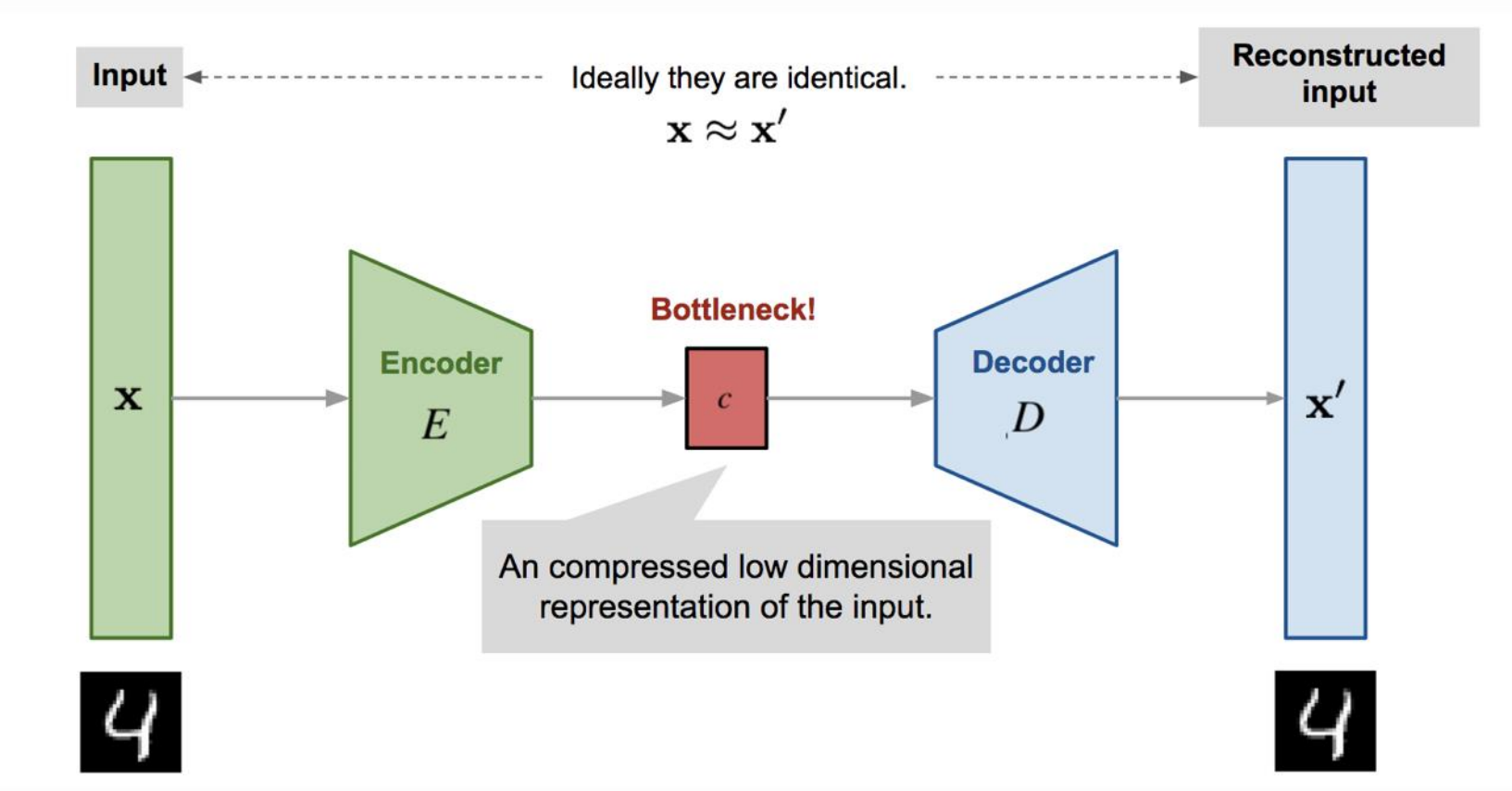
Another approach is to learn a structured latent representation of the data and generate new samples by decoding points from that latent space.

So far, generation was based on competition. Now we move to a different idea: learn a probabilistic latent space and generate by sampling from it.

From autoencoders to generative models

An Autoencoder is a neural network composed of two parts:

- Encoder: compresses the input into a lower-dimensional latent representation.
- Decoder: reconstructs the original input from that representation.



The model is trained so that the reconstructed sample is as similar as possible to the original one.

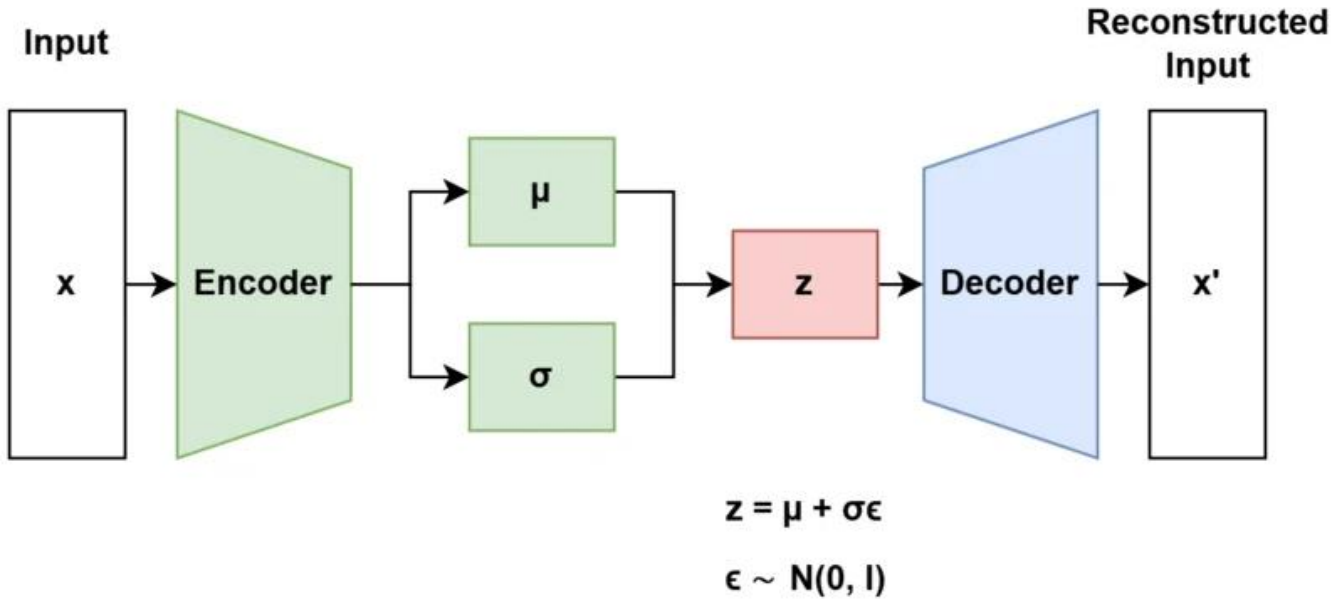
The problem is that a standard Autoencoder does not necessarily organize the latent space in a way that allows us to sample new meaningful points.

What is a Variational Autoencoder?

A Variational Autoencoder (VAE) is a probabilistic generative model.
Instead of encoding each input into a single point in the latent space, the Encoder learns a probability distribution.

For each observation, the Encoder produces:

- a mean, μ
- a standard deviation, σ



These parameters define a latent distribution.

A latent vector z is then sampled from this distribution and passed to the Decoder.

Kingma, D. P., & Welling, M. (2013). **Auto-encoding variational bayes**. *arXiv preprint* arXiv:1312.6114.

Sampling in the latent space

For each input x, the Encoder estimates a latent distribution:

$$z \sim N(\mu, \sigma^2)$$

A latent vector z is sampled from this distribution and used by the Decoder.

However, directly sampling z would make backpropagation difficult.
VAEs therefore use the reparameterization trick:

$$z = \mu + \sigma \cdot \varepsilon$$

Where $\varepsilon\ N(0,1)$, i.e., is the random component, which is separated from the parameters learned by the Encoder.

As a result, the model can still be trained using gradient descent.

Kingma, D. P., & Welling, M. (2013). **Auto-encoding variational bayes**. *arXiv preprint* arXiv:1312.6114.

VAE Loss function

VAE loss contains two components:

1. Reconstruction Loss measures how well the Decoder reconstructs the original input.
The objective is: Reconstructed data ≈ Original data
A low reconstruction loss means that the Decoder preserves the information contained in the original sample.

2. KL Divergence encourages the latent distributions learned by the Encoder to remain close to a standard normal distribution.
This regularizes the latent space and makes it smooth and continuous.

Reconstruction loss preserves information

$$L_{VAE} = L_{reconstruction} + L_{KL}$$

KL divergence makes the latent space suitable for generation

There is a **trade-off**:

- if we only cared about reconstruction, the latent space could become irregular and difficult to sample from.
- If we only cared about the KL term, all latent representations would become too similar and we would lose information about the input.

Kingma, D. P., & Welling, M. (2013). **Auto-encoding variational bayes**. *arXiv preprint* arXiv:1312.6114.

Generating new data with a VAE

After training, the Encoder is no longer required to generate new observations.

To generate synthetic data:

1. Sample a random latent vector z from the latent distribution.
2. Pass z through the Decoder.
3. The Decoder produces a new synthetic observation.

Because the latent space has been regularized during training, nearby latent points should generate similar and realistic observations.

Also in this case, the VAE must be trained only on the minority class samples, if the goal is oversampling.

Xu, L., Skoularidou, M., Cuesta-Infante, A., & Veeramachaneni, K. (2019).
Modeling tabular data using conditional gan. *Advances in neural information processing systems*, 32.

Tabular Variational Autoencoder (TVAE)

It adapts the VAE framework to the specific characteristics of tabular datasets.

Tabular data are challenging because they often contain:

- continuous variables
- categorical variables
- multimodal numerical distributions
- variables with very different scales

TVAE introduces specialized data transformations and output representations so that the VAE can model these heterogeneous attributes effectively.

Xu, L., Skoularidou, M., Cuesta-Infante, A., & Veeramachaneni, K. (2019).
Modeling tabular data using conditional gan. *Advances in neural information processing systems*, 32.

References

1. I. Tomek, **Two modifications of CNN**, In Systems, Man, and Cybernetics, IEEE Transactions on, vol. 6, pp 769-772, 2010.
2. 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.
3. 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.
4. P. Hart, **The condensed nearest neighbor rule**, *Information Theory, IEEE Transactions* on, vol. 14(3), pp. 515-516, 1968.
5. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2020). **Generative adversarial networks**. *Communications of the ACM*, 63(11), 139-144.
6. Xu, L., Skoularidou, M., Cuesta-Infante, A., & Veeramachaneni, K. (2019). **Modeling tabular data using conditional gan**. *Advances in neural information processing systems*, 32.
7. Kingma, D. P., & Welling, M. (2013). **Auto-encoding variational bayes**. *arXiv preprint* arXiv:1312.6114.