Université de Montpellier

HAX907X - Apprentissage statistique

Apprentissage de classes déséquilibrées par rééchantillonnage

Étudiants:
SAWADOGO Kader
GERMAIN Marine
LABOURAIL Célia
MARIAC Damien

Encadrant :
BENSAID Bilel

15 octobre 2025

Table des matières

1 Contexte et Objectifs

Nous nous intéressons à l'apprentissage supervisé dans des contextes où les classes sont déséquilibrées. Ce problème est fréquent dans de nombreux domaines, tels que la détection de fraudes, la médecine ou le diagnostic industriel. Un déséquilibre important des classes peut conduire à des modèles biaisés : ils tendent à prédire correctement les classes majoritaires en négligeant les classes minoritaires.

La problématique scientifique que nous étudions est donc la suivante :
"Comment atténuer le déséquilibre de classes pour améliorer la performance des modèles ?"

L'objectif de ce travail est d'étudier des méthodes permettant de résoudre ce déséquilibre. Plus précisément, nous nous concentrons sur trois approches :

1. Le **Random Over-Sampling** (ROS) cité dans l'article *Survey on Deep Learning with Class Imbalance* [3], une méthode de rééchantillonage qui consiste à augmenter artificiellement la proportion de la classe minoritaire.
2. Le **Random Under-Sampling** (RUS), une autre méthode de rééchantillonage, cité également dans l'article [3] qui consiste à réduire la proportion de la classe majoritaire.
3. La **Synthetic Minority Over-sampling Technique** (SMOTE) présenté dans l'article [1] qui génère de nouvelles instances synthétiques pour la classe minoritaire à partir des observations existantes.

2 Les différentes méthodes

Les méthodes data-level agissent directement sur les données d'entraînement pour atténuer le déséquilibre entre classes. Le principe est soit d'augmenter le poids de la classe minoritaire en lui fournissant plus d'exemples (réels ou synthétiques), soit au contraire de réduire le poids de la classe majoritaire en éliminant certains de ses exemples. L'objectif est d'obtenir une distribution de classes plus équilibrée, ce qui force l'algorithme d'apprentissage à prêter autant d'attention à la minorité qu'à la majorité. Ces techniques peuvent toutefois introduire de la variance (sur-apprentissage) ou du biais supplémentaire, il faut donc les appliquer judicieusement.

2.1 Sur-échantillonnage (Over-sampling)

Le sur-échantillonnage consiste à ajouter des copies ou des variantes d'exemples de la classe minoritaire jusqu'à augmenter sa fréquence dans le jeu de données. Dans sa forme la plus simple, le sur-échantillonnage aléatoire ROS duplique aléatoirement des instances minoritaires existantes jusqu'à atteindre un équilibre désiré.

Par exemple, si l'on dispose de 100 exemples minoritaires et 1000 majoritaires, le ROS peut répliquer les minoritaires (éventuellement plusieurs fois chacun) jusqu'à en obtenir 1000, rétablissant ainsi un ratio équilibré. On échantillonne avec remise parmi les indices de la classe minoritaire pour générer de nouvelles instances d'entraînement.

En revanche, la méthode présente des inconvénients. En effet, la duplication augmente le risque de surapprentissage (overfitting), car le modèle apprend alors trop précisément les exemples déjà présents dans la base de données, ce qui nuit à sa capacité à généraliser sur des nouvelles données. De plus, la duplication de données minoritaire augmenterait significativement la taille des données ce qui implique un coût de calcul plus important.

2.2 Sous-échantillonnage (Under-sampling)

À l'inverse, le sous-échantillonnage vise à réduire la proportion de la classe majoritaire en retirant certains de ses exemples du jeu de données. Le sous-échantillonnage aléatoire RUS élimine au hasard des instances de la classe majoritaire jusqu'à atteindre un ratio plus équilibré avec la minorité. Le RUS est utile lorsque l'on dispose d'une grande abondance de données majoritaires, potentiellement redondantes. Plutôt que de tout utiliser, ce qui peut être coûteux et inutile, on peut se permettre d'en élaguer une partie. En réduisant drastiquement le nombre d'exemples majoritaires, on élimine le biais numérique et on accélère l'entraînement (moins de données à parcourir).

Selon l'article de Bee Wah Yap et al. (2013) [2], il est souvent préférable d'utiliser le RUS plutôt que le ROS, à condition que la taille du jeu de données le permette.

Les deux méthodes de rééchantillonnage ROS et RUS présentent aussi certaines limites. Elles peuvent notamment conduire le modèle à croire que certaines combinaisons de variables expliquent à elles seules la majorité des observations minoritaires, ce qui peut fausser la compréhension réelle des relations entre variables.

2.3 SMOTE (Synthetic Minority Over-sampling Technique)

Plutôt que de copier des instances existantes, SMOTE crée de nouvelles instances minoritaires artificielles en interpolant entre des exemples réels. Concrètement, pour chaque exemple minoritaire original, SMOTE sélectionne aléatoirement l'un de ses k plus proches voisins (minoritaire également), puis génère un nouvel exemple situé aléatoirement le long du segment joignant les deux points dans l'espace des *features*. En répétant ce procédé, on peut synthétiser autant d'exemples minoritaires que souhaité.

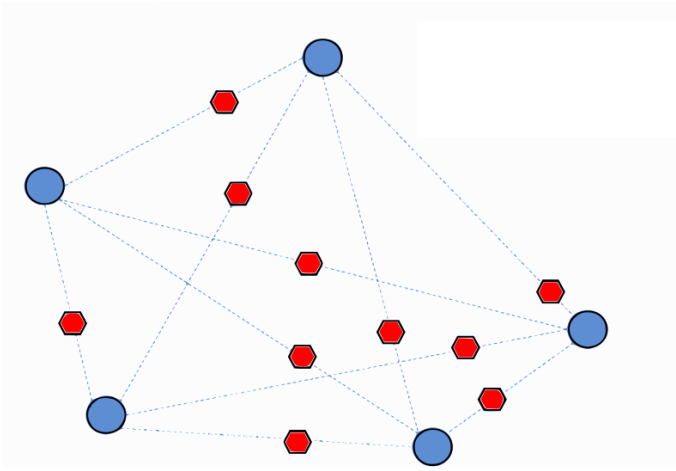

FIGURE 1 – Smote illustration

Les points bleus correspondent à la classe minoritaire dont on souhaite générer de nouveaux exemples, les points rouges sont les points générés par SMOTE.

Typiquement, on se place dans un espace métrique $(\mathbb{R}^d, d)$, avec la distance euclidienne (après normalisation des variables). Soit un jeu d'apprentissage

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}, \quad x_i \in \mathbb{R}^d,\ y_i \in \{0, 1\},$$

et notons $\mathcal{I}_{\min} = \{i : y_i = 1\}$ l'ensemble des indices minoritaires. Pour un point minoritaire x_i, on note $N_k(x_i) \subset \mathcal{I}_{\min}$ l'ensemble de ses k plus proches voisins minoritaires selon d.

SMOTE génère un point synthétique en interpolant *linéairement* entre un point minoritaire x_i et un de ses voisins $x_j \in N_k(x_i)$ choisi au hasard :

$$x_{\text{new}} \;=\; x_i \;+\; \lambda\,(x_j - x_i), \qquad \lambda \sim \mathcal{U}(0,1).$$

En répétant cette opération M fois (réparties sur les $x_i \in \mathcal{I}_{\min}$), on obtient un ensemble $\mathcal{S}$ de M points synthétiques et un jeu équilibré $\mathcal{D}' = \mathcal{D} \cup \{(x,1) : x \in \mathcal{S}\}$. Le nombre M est fixé pour atteindre un ratio de classes cible

$$\rho \;=\; \frac{n_{\min} + M}{n_{\text{maj}}}.$$

Cependant, SMOTE présente plusieurs inconvénients importants. Tout d'abord, son temps de calcul est plus long que celui des méthodes plus simples. Cela s'explique par le fait que l'algorithme doit calculer la distance entre chaque observation minoritaire et tous les autres points pour identifier ses k plus proches voisins. Ce calcul devient rapidement coûteux lorsque la taille du jeu de données augmente, car le nombre de comparaisons croît de manière quadratique (détails en annexe 4.3). En pratique, cela signifie que sur des bases volumineuses ou avec beaucoup de variables, SMOTE peut nécessiter des temps de traitement bien plus élevés, surtout lorsqu'il est utilisé à chaque étape d'une validation croisée.

Un autre problème souvent mentionné dans la littérature concerne la sensibilité de SMOTE aux points aberrants. Si une observation minoritaire isolée se trouve très proche d'un point appartenant à la classe majoritaire, l'algorithme risque d'utiliser cette proximité pour créer de nouvelles données synthétiques incorrectes. Cela peut introduire du bruit et perturber l'apprentissage du modèle. Pour limiter ce problème, il est possible d'ignorer les observations minoritaires dont les k plus proches voisins appartiennent principalement à la classe majoritaire, afin d'éviter de générer des points non représentatifs.

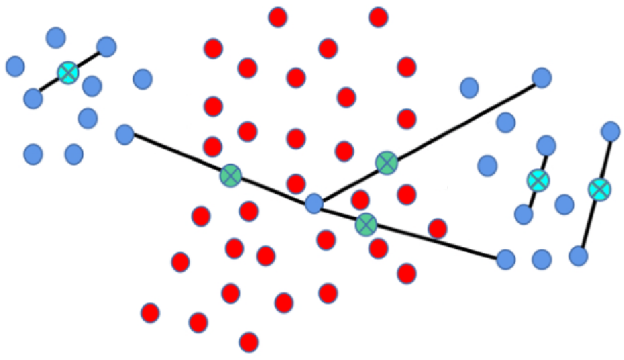

FIGURE 2 – Smote et points aberrants

Dans cette illustration, les points bleux représentent la classe minoritaire, les rouges la classe majoritaire et les points bleus avec une croix sont les points générés par SMOTE. On remarque que certains points synthétiques sont créés dans la zone de la classe majoritaire du à un point mal classé, ce qui peut perturber l'apprentissage du modèle.

Plusieurs variantes de SMOTE ont été développées pour corriger ces défauts :

- **SMOTE-NC (Nominal and Continuous)** : s'adapte aux jeux de données contenant à la fois des variables continues et qualitatives.
- **Borderline-SMOTE** (Hui Han, Wen-Yuan Wang et Bing-Huan Mao, 2005) : se concentre sur les points situés à la frontière entre classes, considérés comme plus informatifs pour le modèle.

3 Conclusion

Les méthodes ROS, RUS et SMOTE sont des approches différentes pour corriger le déséquilibre de classes. Elles ne se comparent pas directement, car leur efficacité dépend du jeu de données et du contexte d'application. En effet, on voit sur le tableau 1 le récapitulatif des 3 méthodes avec leur points forts et point faibles.
En réponse à la problématique posée, il n'existe pas de méthode réellement meilleure qu'une autre : le choix doit être adapté aux données et au modèle utilisé. L'objectif est d'obtenir un jeu de données plus équilibré afin d'améliorer les performances du modèle.

Méthode	**Principe**	**Avantages**	**Inconvénients**
ROS	Duplication aléatoire de la classe minoritaire	- Simplicité - Conservation des données	- Surapprentissage - Volume important de données
RUS	Suppression aléatoire de la classe majoritaire	- Rapidité - Réduction du biais - Moins de données à traiter	- Perte d'information - Moins de représentativité
SMOTE	Création d'exemples synthétiques de la classe minoritaire (interpolation entre exemples réels)	- Données synthétiques variées - Moins de surapprentissage que ROS	- Coût calcul élevé - Sensible aux valeurs abberrantes - Risque de bruit

TABLE 1 – Résumé des méthodes

Il existe néanmoins des variantes et des combinaisons de ces méthodes qui peuvent être explorées pour tirer parti de leurs avantages respectifs. Par exemple, on peut combiner SMOTE avec un léger sous-échantillonnage de la classe majoritaire pour obtenir un compromis entre diversité et réduction du biais.
Nous avons exploré trois méthodes qui agissent au niveau des données pour atténuer le déséquilibre de classes. Mais il existe aussi des approches au niveau de l'algorithme, qui modifient la fonction de coût ou le processus d'apprentissage pour accorder plus d'importance à la classe minoritaire. Ces techniques peuvent être combinées avec les méthodes présentées ici pour maximiser les chances d'obtenir un modèle performant et robuste face au déséquilibre.

4 Annexe

4.1 Implémentation et résultats

Exemple d'application : Détection de fraudes par carte bancaire

L'objectif de cette application est de mettre en place un modèle de détection de fraudes à partir du jeu de données *Credit Card Fraud Detection* disponible sur *Kaggle* et sur le github associé [4]. Ce jeu contient 284 807 transactions, dont seulement 492 sont frauduleuses (soit environ 0,17 %). Le fort déséquilibre entre les classes rend la tâche difficile, car un modèle naïf qui prédirait toujours "non fraude" atteindrait déjà un score proche de 99,8 %. Pour dépasser cette limite, plusieurs stratégies de gestion du déséquilibre ont été appliquées et comparées en utilisant une régression logistique.

Régression logistique sans traitement

Dans un premier temps, nous avons entraîné le modèle sur l'ensemble complet, sans traitement particulier du déséquilibre. Les résultats montrent une accuracy très élevée (0,9991), mais trompeuse, puisque le rappel (sensitivity) des fraudes n'est que de 0,61. Autrement dit, la majorité des fraudes ne sont pas détectées, ce qui rend le modèle inutilisable en pratique malgré une performance apparente.

RUS

Afin de corriger ce problème, nous avons testé le RUS, qui consiste à équilibrer les classes en réduisant volontairement le nombre de transactions normales pour égaler celui des fraudes. Dans ce cas, le modèle obtient une *balanced accuracy* d'environ 0,956 avec un rappel de 0,959. Le modèle apprend donc à bien reconnaître les fraudes, mais au prix d'une perte importante d'information, puisque la majorité des transactions légitimes est écartée.

ROS

La seconde approche est le ROS, qui duplique aléatoirement les fraudes afin de créer un jeu équilibré. Cette méthode conserve toutes les données disponibles et améliore le rappel (0,977), ce qui signifie que presque toutes les fraudes sont détectées. La *balanced accuracy* atteint 0,949. Toutefois, l'inconvénient de cette méthode est le risque de surapprentissage, car les exemples de fraude sont répétés artificiellement sans ajouter de diversité.

SMOTE

Enfin, nous avons appliqué la méthode *SMOTE (Synthetic Minority Oversampling Technique)*, qui génère artificiellement de nouveaux exemples de fraude en interpolant entre des observations existantes. Cette méthode dépasse les performances des précédentes, avec une accuracy globale de 0,981, un rappel de 0,992 et une spécificité de 0,970. Elle offre donc le meilleur compromis entre la détection des fraudes et la limitation des faux positifs.

4.2 Algorithmes

Algorithm 1: RUS

Input: Dataset D avec classes C_{maj} et C_{min}
Output: Dataset équilibré D'
$n_{min} \leftarrow |C_{min}|$;
Sélectionner aléatoirement n_{min} échantillons de C_{maj};
Construire $D' = C_{min} \cup$ sous-échantillon(C_{maj});
return D';

Algorithm 2: ROS

Input: Dataset D avec classes C_{maj} et C_{min}
Output: Dataset équilibré D'
$n_{maj} \leftarrow |C_{maj}|$;
while $|C_{min}| < n_{maj}$ **do**
    Dupliquer aléatoirement un échantillon de C_{min};
Construire $D' = C_{maj} \cup C_{min}$ (après duplication);
return D';

Algorithm 3: SMOTE (Synthetic Minority Over-sampling Technique)

Input: Dataset D avec classes C_{maj} et C_{min}, nombre de voisins k
Output: Dataset équilibré D'
foreach $x_i \in C_{min}$ **do**
    Trouver les k plus proches voisins de x_i dans D;
    if *tous les voisins appartiennent à* C_{maj} **then**
        Marquer x_i comme **bruit** (non utilisé);
    else if *la majorité des voisins appartiennent à* C_{maj} **then**
        Marquer x_i comme **danger**;
    else
        Marquer x_i comme **safe** (non utilisé);
foreach x_i *marqué comme* ***danger*** **do**
    Sélectionner aléatoirement un voisin minoritaire x_{voisin} parmi ses k plus proches voisins;
    Générer un nombre aléatoire $\lambda \sim U(0, 1)$;
    Créer un point synthétique :

$$x_{\text{new}} = x_i + \lambda \cdot (x_{voisin} - x_i)$$

    Ajouter x_{new} à C_{min};
Construire $D' = C_{maj} \cup C_{min}$ (avec points synthétiques générés);
return D';

4.3 Coût algorithmique de SMOTE

On note n le nombre total d'observations, $n_{\min}$ le nombre d'observations minoritaires, d la dimension, k le nombre de voisins à récupérer, et M le nombre de points synthétiques à générer. On suppose l'espace $\mathbb{R}^d$ standardisé et une distance euclidienne.

Étape kNN (dominante) [5]
Pour chaque point minoritaire x_i, on calcule sa distance à *tous* les points du jeu.

- **Coût d'une distance** : $O(d)$.
- **Coût pour comparer à n points** : $O(nd)$.
- **Coût pour $n_{\min}$ points minoritaires** : $O(n_{\min}\, n\, d)$.

La sélection des k plus petites distances parmi n peut se faire en temps linéaire amorti (par sélection partielle) ou en $O(n \log n)$ via tri, mais *dans tous les cas* le calcul des distances $O(nd)$ domine lorsque d et n sont suffisamment grands. Ainsi, le coût de l'étape kNN en naïf est

$$T_{\text{kNN, naïf}} = O(n_{\min}\, n\, d).$$

Étape de génération (négligeable devant kNN)

Chaque point synthétique s'obtient par :

$$x_{\text{new}} = x_i + \lambda\,(x_j - x_i), \qquad \lambda \sim \mathcal{U}(0,1),$$

soit une interpolation vectorielle en d composantes $\Rightarrow O(d)$ par point. Pour M points générés :

$$T_{\text{gen}} = O(M\, d).$$

Bilan temps.

$$\boxed{T_{\text{SMOTE}} \;=\; O(n_{\min}\, n\, d) \;+\; O(M\, d)}$$

En pratique, le terme $O(n_{\min}\, n\, d)$ (recherche des voisins par force brute) domine.

4.4 Coût algorithmique de ROS et RUS

On note n le nombre total d'observations, $n_{\min}$ le nombre d'observations minoritaires, n_{maj} le nombre d'observations majoritaires, d la dimension, et M le nombre de points à ajouter (ROS) ou à retirer (RUS). On suppose les données stockées en matrice dense $\mathbb{R}^{n \times d}$.

ROS (Random OverSampling). On échantillonne avec remise des indices de la classe minoritaire pour créer M duplicatas.

- **Tirage d'indices (avec remise)** : $O(M)$.
- **Copie des M vecteurs (dimension d)** : $O(Md)$.

Ainsi, le coût total (naïf) de ROS est

$$T_{\text{ROS}} = O(Md).$$

En pratique, le terme mémoire/copie $O(Md)$ domine (aucun calcul de distance).

RUS (Random UnderSampling). On choisit aléatoirement (sans remise) n_{keep} points parmi la classe majoritaire et on forme un nouveau jeu de données $X' = X_{\min} \cup X_{\text{maj,keep}}$, de taille $n_{\text{cible}} = n_{\min} + n_{\text{keep}}$ (typiquement $n_{\text{cible}} \leq n$).

— **Sélection d'indices (sans remise)** : $O(n_{\text{keep}})$ (via un tirage aléatoire ou un shuffle linéaire).

— **Assemblage / copie des lignes retenues** : $O(n_{\text{cible}}\, d)$.

Le coût total s'écrit donc

$$T_{\text{RUS}} = O(n_{\text{cible}}\, d) \quad \text{avec} \quad n_{\text{cible}} \leq n,$$

souvent dominé par l'assemblage mémoire. Si l'on ne matérialise que des *masques d'indices* et que l'entraînement en aval accepte un sous-échantillonnage par vue, le coût pratique se réduit au tirage $O(n_{\text{keep}})$.

Bilan. Contrairement à SMOTE (dominé par la recherche kNN), ROS et RUS n'impliquent aucun calcul de distance. Les coûts sont essentiellement linéaires en la taille copiée :

$$\boxed{T_{\text{ROS}} = O(Md)\,, \qquad T_{\text{RUS}} = O(n_{\text{cible}}\, d)}$$

avec, en pratique, une domination par les opérations de copie/assemblage mémoire.

Références

[1] Nitesh V Chawla et al. “SMOTE : synthetic minority over-sampling technique”. In : *Journal of artificial intelligence research* 16 (2002), p. 321-357.

[2] Merwan Chelouah. “Méthodes de rééquilibrage des classes en classification supervisée”. In : ().

[3] Justin M Johnson et Taghi M Khoshgoftaar. “Survey on deep learning with class imbalance”. In : *Journal of big data* 6.1 (2019), p. 1-54.

[4] MARIAC Damien, GERMAIN Marine, LABOURAILLE Célia, SAWADOGO Kader. *Nearest Neighbors*. `https://github.com/DamienMariac/classes-desequilibre`. 2025.

[5] scikit-learn developers. *Nearest Neighbors*. `https://scikit-learn.org/stable/modules/neighbors.html`. Accessed Oct. 2025. 2025.