Big Data & Data Science
TP05 – Apprentissage Automatique Supervisé — Données Déséquilibrées
TP noté (Contrôle Continu)

Vous devez soumettre vos solutions de ce travail sur Moodle, en respectant la date limite fixée par votre enseignant(e).
Vous devez travailler en binôme ou trinôme, mais une seule soumission par groupe est requise.
La triche ne sera pas tolérée et sera sanctionnée.

NB : Veillez à ajouter à chaque étape des commentaires internes expliquant le code et interprétant les résultats. Cela est obligatoire et sera pris en compte dans l'évaluation. L'absence de ces commentaires à une étape entraînera une perte de points correspondante.

Exercice 1 — Effet du déséquilibre des classes sur la prédiction de la qualité du vin (Facile)
(5 points)

Objectif d'apprentissage

Comprendre comment le déséquilibre des classes influence les performances d'un classificateur et analyser pourquoi les métriques standards comme la précision globale (accuracy) peuvent être trompeuses dans de tels contextes.

Consignes

- Veuillez intégrer vos réponses dans le notebook Python : TP05_Exercice1&2_TODO.ipynb

Nous utilisons le jeu de données *Wine Quality* partagé.

1. Charger le jeu de données *Wine Quality* (version vin rouge ou vin blanc) à partir du UCI Machine Learning Repository ou de sklearn.datasets. (https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv)
2. Transformer les scores de qualité originaux en une tâche de classification binaire en utilisant un seuil σ. Prendre différentes valeurs de σ ∈ {6, 7, 8} :
   • Attribuer aux échantillons ayant un score de qualité ≥ σ l'étiquette « good » (*classe positive*),
   • Attribuer à tous les autres l'étiquette « not good » (*classe négative*). Cela crée naturellement un jeu de données déséquilibré, car les vins de haute qualité sont moins fréquents.
3. Diviser les données en ensembles d'entraînement et de test.
4. Entraîner un modèle de régression logistique pour prédire si un vin est de bonne qualité.
5. Évaluer le modèle à l'aide des métriques suivantes pour les différentes valeurs de σ :
   - Accuracy,

- Précision, Rappel (Recall), F1-score,
- ROC-AUC.

6. Analyser les résultats et expliquer pourquoi la précision (accuracy) peut être trompeuse en présence de déséquilibre de classes.
7. Visualiser la matrice de confusion pour appuyer votre analyse.

Exercice 2 — Rééchantillonnage (5 points)

Il s'agit de la suite de l'exercice 1: Continuez à intégrer vos réponses dans le même notebook Python : TP05_Exercice1&2_TODO.ipynb

1. Afficher le ratio de déséquilibre des classes avant le rééchantillonnage.
2. Appliquer une technique de rééchantillonnage telle que RandomOverSampler ou SMOTE pour rééquilibrer les classes.
3. Réentraîner et réévaluer le modèle en utilisant les mêmes métriques.
4. Comparer et interpréter les performances avant et après rééchantillonnage.

Exercice 3 — Rééquilibrage des données et validation croisée (Intermédiaire) (5 points)

Objectif d'apprentissage

Mettre en œuvre et comparer différentes stratégies de rééquilibrage, et analyser leur impact sur les performances et la robustesse du modèle.

Consignes

Intégrez vos réponses dans le notebook TP05_Exercice3_ImbalancedData_TODO .ipynb

1. Télécharger le jeu de données `breast_cancer :`
   from sklearn.datasets import load_breast_cancer
   `X, y = load_breast_cancer(return_X_y=True)`

2. Vérifier le problème de déséquilibre et calculer le ratio d'imbalance.

3. Expérimenter les méthodes suivantes de gestion du déséquilibre :
   - Sans rééquilibrage (baseline),
   - RandomOverSampler,
   - SMOTE,
   - RandomUnderSampler.

4. Entraîner un classificateur Random Forest pour chaque méthode.

5. Évaluer tous les modèles à l'aide d'une validation croisée stratifiée et comparer :
   - F1-score,
   - AUC,
   - Temps d'entraînement.

6. Discuter des compromis entre précision, rappel et surapprentissage (overfitting) pour chaque stratégie.

7. Visualisation : Créer un graphique en barres comparatif des F1-scores et des valeurs AUC pour toutes les méthodes de rééquilibrage.

(Optionnel : Points Bonus) Algorithme de validation croisée StratifiedKFold

Objectif

Comprendre et reproduire le mécanisme interne de la validation croisée stratifiée en implémentant votre propre version de StratifiedKFold en Python. Cet exercice vise à approfondir la compréhension du partitionnement des données en préservant les proportions de classes dans chaque pli (fold).

Consignes

1. Implémenter une fonction my_stratified_kfold(X, y, K, random_state) qui :
   - Divise le jeu de données en K plis,
   - Garantit que chaque pli conserve la même distribution de classes que le jeu de données original,
   - Retourne une liste de paires d'indices (indices d'entraînement, indices de test).
2. Tester votre implémentation sur un petit jeu de données synthétique (par exemple, 10 échantillons avec 2 classes).
3. Comparer vos résultats avec sklearn.model_selection.StratifiedKFold pour vérifier la justesse.

Livrables

- Votre implémentation Python de my_stratified_kfold.
- Une courte explication (3–5 phrases) décrivant comment votre méthode maintient l'équilibre des classes.
- Une capture d'écran ou impression comparant votre sortie avec celle de StratifiedKFold de scikit-learn.

Algorithme 1 : StratifiedKFold(X, y, K)

Entrées :

X : matrice de caractéristiques de taille N × d

y : vecteur d'étiquettes de taille N

K : nombre de plis souhaités

Sortie : ensemble de K paires (indices d'entraînement, indices de test)

Étapes :

1. Regrouper les échantillons par classe
   Créer une partition C où chaque sous-ensemble Ci contient les indices des échantillons appartenant à la classe i.
2. Mélanger dans chaque classe
   Pour chaque classe i ∈ C : mélanger aléatoirement les indices dans Ci (avec une graine aléatoire fixe pour la reproductibilité).

3. Diviser chaque classe en K sous-ensembles
   Pour chaque classe i : Diviser Ci en K parties de taille approximativement égale : Ci = {Ci,1, Ci,2, …, Ci,K}.
4. Construire les plis stratifiés
   Pour k = 1 à K :
   - Former le pli de test F_test_k en concaténant Ci,k pour toutes les classes.
   - Former le pli d'entraînement F_train_k avec tous les échantillons restants.
5. Retourner les plis : Retourner la liste des paires (F_train_1, F_test_1), …, (F_train_K, F_test_K).

Utiliser numpy ou pandas pour manipuler efficacement les indices, et tester le code sur des distributions d'étiquettes équilibrées et déséquilibrées.

Exercice 4 — Apprentissage sensible aux coûts et calibration du seuil avec les arbres de décision (Avancé) (5 points)

Objectif d'apprentissage

Explorer des techniques avancées pour gérer le déséquilibre de classes sans rééchantillonnage, en utilisant le pondération des classes (class weighting) et l'optimisation du seuil de décision avec un classificateur Decision Tree.

Consignes

Intégrez vos réponses dans le notebook TP05_Exercice4_ImbalancedData_TODO .ipynb

1. Charger un jeu de données binaire déséquilibré (par exemple, *Breast Cancer dataset* de sklearn.datasets, modifié pour accroître le déséquilibre).

2. Entraîner un DecisionTreeClassifier en utilisant le paramètre class_weight :
   - D'abord avec le modèle par défaut (aucune pondération),
   - Puis avec class_weight='balanced'.

3. Évaluer les modèles à l'aide de :
   - la courbe ROC,
   - la courbe Précision–Rappel.

4. Déterminer le seuil de décision optimal qui maximise le F1-score ou le rappel selon le contexte d'application (diagnostic médical, détection de fraude, etc.).

5. Comparer les performances avant et après l'ajustement du seuil, et discuter de l'impact sur les faux positifs et faux négatifs.

6. Visualiser la structure de l'arbre de décision pour interpréter comment la pondération des classes influence les frontières de décision apprises.

Extension optionnelle (Points Bonus)

- Implémenter une fonction de coût personnalisée où les faux négatifs sont pénalisés dix fois plus que les faux positifs.
- Utiliser cette fonction pour calculer une métrique d'évaluation basée sur le coût et la comparer au F1-score standard.

GOOD LUCK !