0.1 Knn

0.1.1 Definition

KNN est un algorithme d'apprentissage supervisé utilisé pour la classification et la régression. Il fonctionne en trouvant les 'k' points de données les plus proches dans l'espace des caractéristiques et en utilisant ces points pour prédire la classe ou la valeur de la nouvelle donnée.

Les distances couramment utilisées pour mesurer la proximité entre les points de données incluent la distance euclidienne, la distance de Manhattan et la distance de Minkowski.

Les paramètres clés de l'algorithme KNN incluent :

- **K :** Le nombre de voisins à considérer pour la prédiction.
- **Distance Metric :** La méthode utilisée pour calculer la distance entre les points de données.
- **Weighting :** La manière dont les voisins sont pondérés lors de la prise de décision (par exemple, pondération uniforme ou pondération par distance).

0.1.2 Implemenation from scratch

Voici notre implémentation de l'algorithme KNN en Python avec ces petites modifications :

- Utilisation de la distance carrée $(x-y)^2$ à la place de la distance euclidienne $\sqrt{(x-y)^2}$ pour éviter le calcul de la racine carrée, aussi pour beneficier du calcul matricielle plus rapide avec numpy.
- Cette modification n'affecte pas le résultat final, car la racine carrée est une fonction monotone croissante, donc l'ordre des distances reste le même. autrement dit, si $d_1 < d_2$, alors $\sqrt{d_1} < \sqrt{d_2}$.

`Implemenation KNN from scratch`

0.2 Normalization et Encodage de données

Comme KNN est basé sur la distance entre les points de données, il est crucial de normaliser les caractéristiques pour garantir que toutes les dimensions contribuent équitablement au calcul de la distance. Les techniques que nous avons utilisées sont :

- Pour la normalisation nous avons utilisé **Standarisation (Z-score)** pour centrer et réduire les données et préserver les distances relatives entre les points.
- Pour l'encodage des caractéristiques categorilaes, nous avons utilisé **Dummy Encoding** pour ne pas introduire d'ordre entre les catégories, sauf pour les caractéristiques avec un grand nombre de catégories uniques comme le **GRIDCODE** où nous avons utilisé **Ordinal Encoding** pour ne pas introduire beaucoup de nouvelle caractéristiques.

0.3 Balancement de données

Comme **KNN** est sensible à la distribution des classes dans les données, nous avons exploré différentes techniques de balancement pour équilibrer les classes, nous avons utilisé un model avec $k = 5$ et une distance pondérée et le pour la comparaison des resultats nous avons utilisé le métrique **ROC AUC**. Les techniques de balancement que nous avons testées sont :

- **Original Data :** Utilisation des données telles quelles, sans modification. Les resultats obtenus sont comme suit :

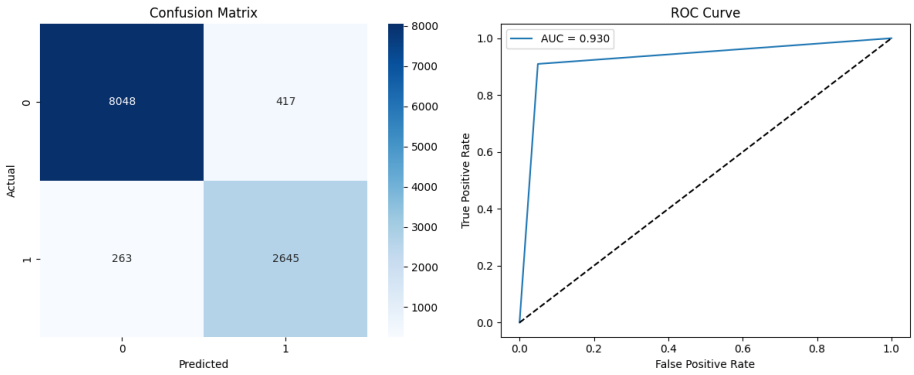

Figure 1: KNN avec données originales

- **Random Under Sampling :** Sous échantillonnage aléatoire des données majoritaires pour équilibrer les classes. Les resultats obtenus sont comme suit :

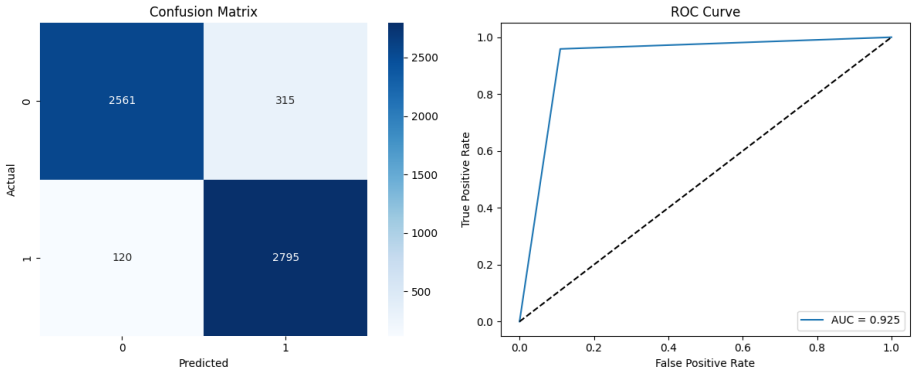

Figure 2: KNN avec données random under sampling

- **Ranodom Over Sampling :** Sur échantillonnage aléatoire des données

minoritaires pour équilibrer les classes. Les resultats obtenus sont comme suit :

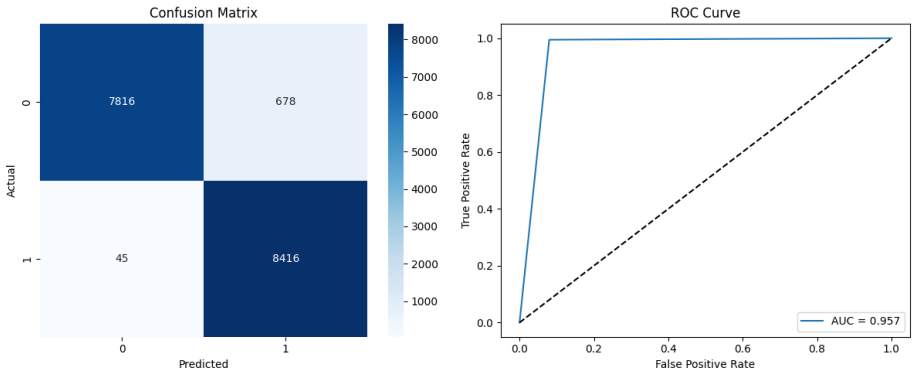

Figure 3: KNN avec données random over sampling

- **Smote Over Sampling :** Sur échantillonnage synthétique des données minoritaires pour équilibrer les classes. Les resultats obtenus sont comme suit :

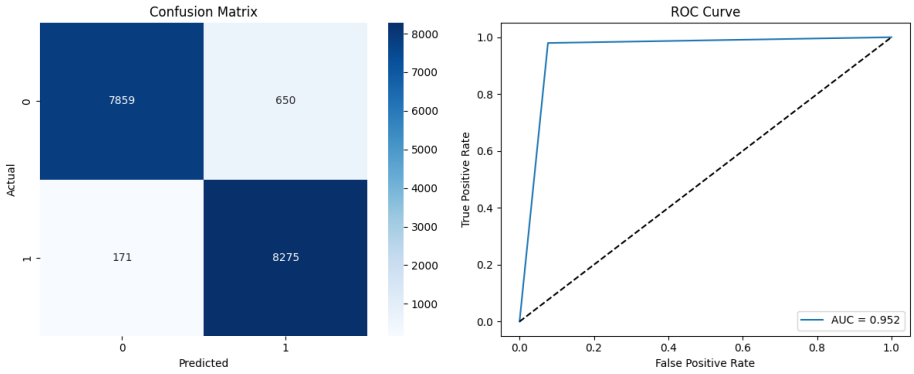

Figure 4: KNN avec données smote over sampling

- **Tomek Under Sampling :** Sous échantillonnage des données majoritaires en éliminant les exemples proches des frontières de décision. Les resultats obtenus sont comme suit :

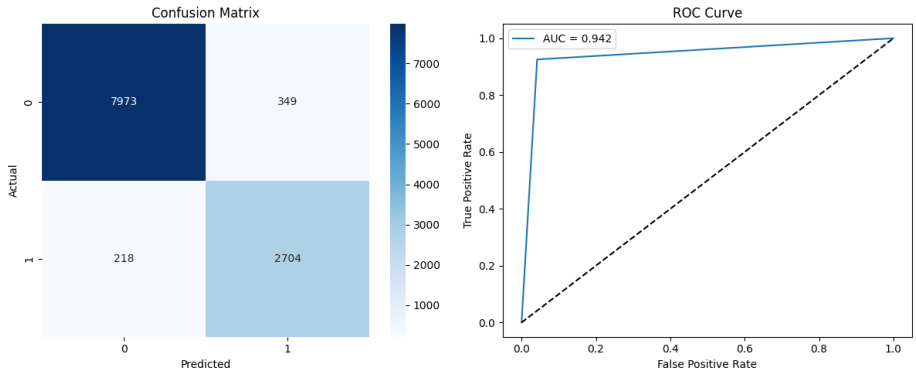

Figure 5: KNN avec données tomek under sampling

À la fin la meilleure technique de balancement pour KNN est **Random Over Sampling** avec les resultats suivants :

- Best sampling method: Random Over-Sampling with ROC AUC: 0.9574
- New dataset sizes: Train=67819, Test=16955, Full=84774
- Class distribution: class 0: 33893, class 1: 33926

0.4 Réglage des paramètres

Pour le réglage des paramètres de l'algorithme KNN, nous avons utilisé la validation croisée (k-folds avec $k = 3$) avec une recherche d'optimisation bayésienne de 30 itérations, la plage des paramètres testés est la suivante :

- **n_neighbors :** [1, 10]
- **weights :** ['uniform', 'distance'] indiquant si tous les voisins ont le même poids ou si les voisins plus proches ont un poids plus élevé.
- **metric :** ['euclidean', 'manhattan'] différentes méthodes pour calculer la distance entre les points de données.

Les meilleurs paramètres trouvés sont sont illustrés dans Table 1.

0.5 Réduction de dimensionnalité

Comme KNN peut être affecté par la malédiction de la dimensionnalité, nous avons appliqué une réduction de dimensionnalité en utilisant l'analyse en composantes principales (PCA) pour réduire le nombre de caractéristiques tout en conservant 90% de la variance des données. Après l'application de PCA, le nombre de caractéristiques est passé de 40 à 16. Nous avons ensuite répété le processus de réglage des paramètres avec les mêmes plages de valeurs. Les

meilleurs paramètres trouvés après la réduction de dimensionnalité sont exactement les mêmes que ceux trouvés avant la réduction de dimensionnalité, les résultat sont illustrés dans Table 1

0.6 Comparaison de resultats

Les resultats finaux obtenus avec KNN après le réglage des paramètres et l'utilisation de la technique de balancement Random Over Sampling sont les suivants :

0.6.1 Réglage de paramèteres

Configuration	**n_neighbors**	**weights**	**metric**
Sans réduction de dimensionnalité	1	uniform	Manhattan
Avec réduction de dimensionnalité	1	uniform	Manhattan

Table 1: Comparaison des performances avec et sans réduction de dimensionnalité

0.6.2 Résultats finaux

Configuration	**Accuracy**	**Precision**	**Recall**	**F1 Score**
Sans réduction de dimensionnalité	0.9706	0.9528	0.9902	0.9711
Avec réduction de dimensionnalité	0.9694	0.9509	0.9898	0.9700

Table 2: Comparaison des performances avec et sans réduction de dimensionnalité

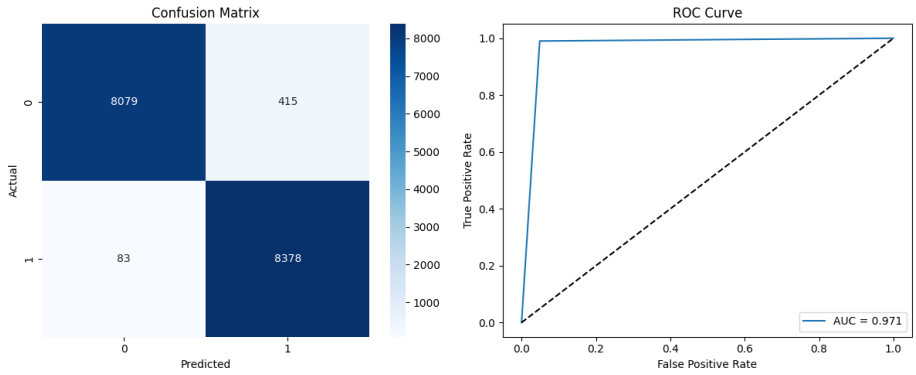

Figure 6: KNN - Final Results with no PCA

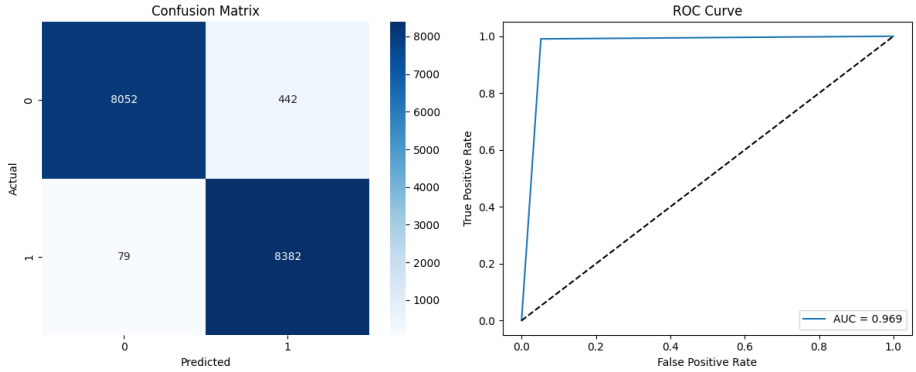

Figure 7: KNN - Final Results with PCA

Nous observons que la réduction de dimensionnalité n'a pas eu un impact significatif sur les performances de l'algorithme KNN dans ce cas, mais elle a permis de réduire significativement le temps de calcul et les ressources nécessaires pour l'entraînement et la prédiction du modèle.