

## 0.1 Random Forest

### 0.1.1 Definition

**Random Forest** est un algorithme d'apprentissage supervisé utilisé pour la classification et la régression. Il fonctionne en combinant plusieurs arbres de décision construits sur des sous-échantillons aléatoires des données, et en agrégeant leurs prédictions pour améliorer la précision et réduire le surapprentissage. Chaque arbre de décision est construit en sélectionnant aléatoirement un sous-ensemble des caractéristiques à chaque nœud, ce qui introduit de la diversité parmi les arbres et améliore la robustesse du modèle global.

Les critères utilisés pour mesurer la qualité des divisions incluent sans les mêmes que celle d'arbre de décision comme Gini, le gain d'information (entropie) et la réduction de la variance.

Les paramètres clés de l'algorithme Random Forest incluent :

- tous les paramètres de l'arbre de décision, tels que :
  - **Max Depth** : La profondeur maximale de l'arbre, généralement utilisé pour contrôler la complexité de l'arbre et éviter le surapprentissage.
  - **Min Samples Split** : Le nombre minimum d'échantillons requis pour diviser un nœud, généralement fixé à la valeur minimale de 2.
  - **Min Samples Leaf** : Le nombre minimum d'échantillons requis pour être à une feuille, généralement fixé à la valeur minimale de 1.
  - **Criterion** : La fonction utilisée pour mesurer la qualité des divisions (par exemple, Gini, Entropie).
- **N Estimators** : Le nombre d'arbres dans la forêt, varie selon le dataset.

### 0.1.2 Implementation from scratch

Voici notre implémentation de l'algorithme Random Forest en Python.

Implementation Random Forest from scratch

## 0.2 Normalization et Encodage de données

Puisque Random Forest est basé sur des arbres de décision, nous avons appliqué les mêmes techniques de normalisation et d'encodage que pour l'algorithme d'arbre de décision :

- Pour l'encodage des caractéristiques catégorielles, nous avons utilisé **Ordinal Encoding** pour ne pas introduire beaucoup de nouvelles caractéristiques ainsi que pour préserver un ordre entre les valeurs pour permettre à l'algorithme de mieux séparer les données.

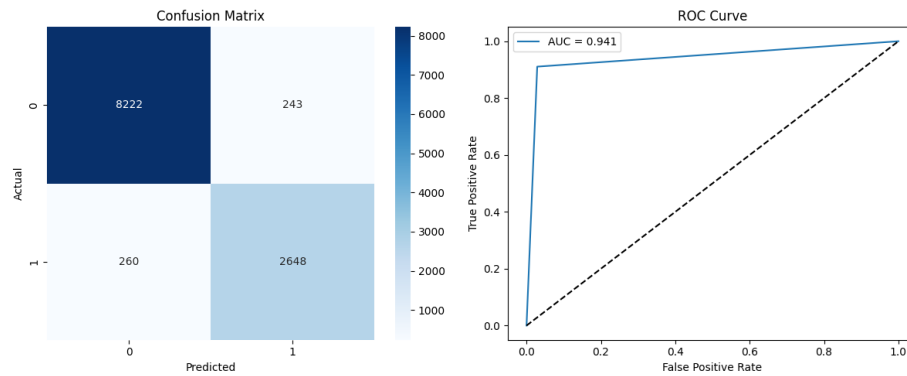


Figure 1: Random Forest avec données originales

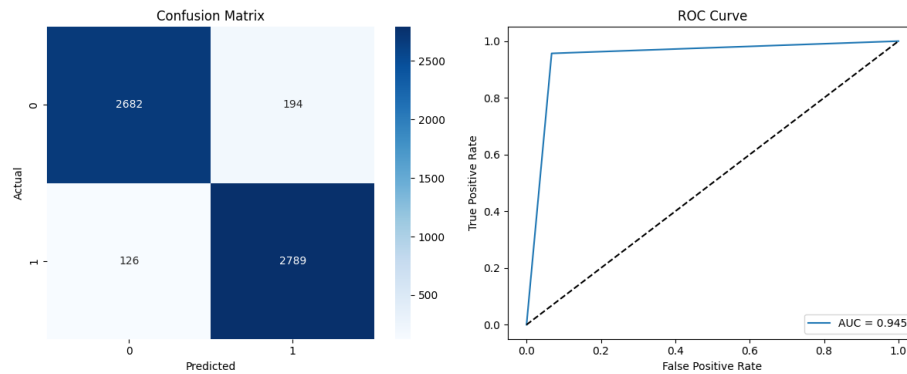


Figure 2: Random Forest avec données random under sampling

### 0.3 Balancement de données

Pour le balancement de classes comme avec les arbres de décision, nous avons utilisé un model de Random Forest de base avec les paramètres importants fixés à 50 pour **n\_estimators** et "gini" pour **criterion** et nous avons utilisé la métrique **ROC AUC** pour l'évaluation. Les techniques de balancement que nous avons testées sont :

- **Original Data :**
- **Random Under Sampling :**
- **Ranodm Over Sampling :**
- **Smote Over Sampling :**
- **Tomek Under Sampling :**

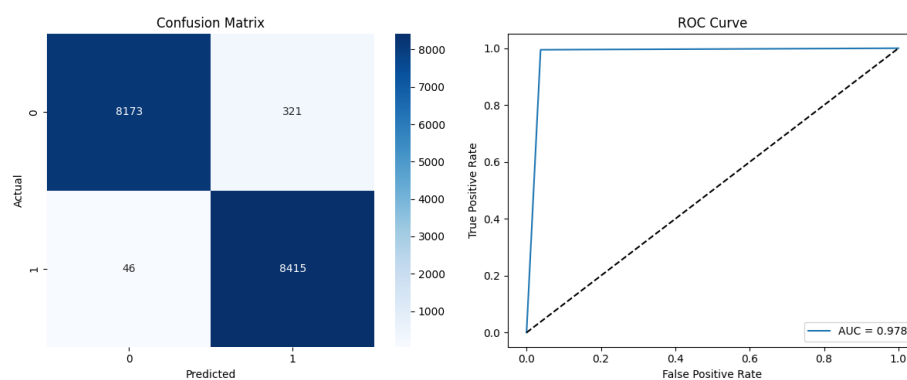


Figure 3: Random Forest avec données random over sampling

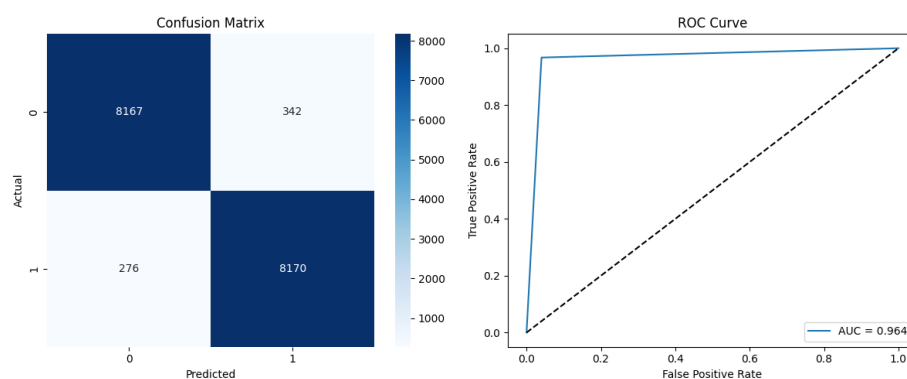


Figure 4: Random Forest avec données smote over sampling

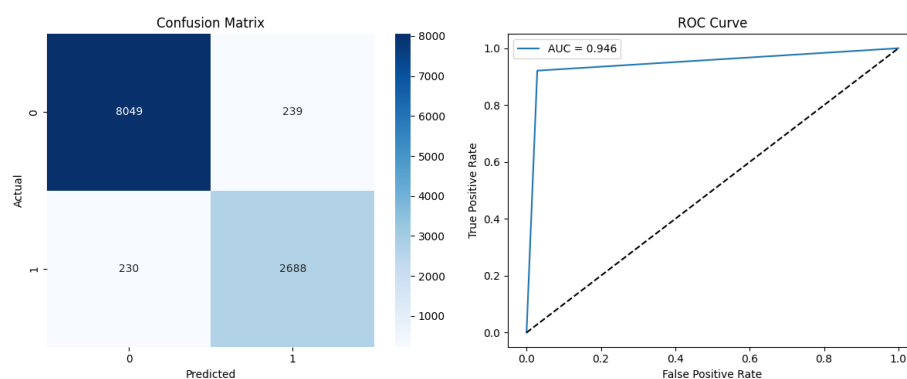


Figure 5: Random Forest avec données tomes under sampling

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

- Best sampling method: Random Over-Sampling with ROC AUC: 0.9784
- 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 Random Forest, nous avons utilisé la validation croisée (k-folds avec  $k = 2$ ) avec une recherche de grille, la plage des paramètres testés est la suivante :

- **max\_depth, min\_samples\_split, min\_samples\_leaf** : les memes valeurs optimales que pour l'arbre de décision
- **n\_estimators** : [20, 30, 50, 100] le nombre d'arbres dans la forêt
- **max\_features** : [None, "sqrt", "log2"] le nombre de caractéristiques à considérer lors de la recherche de la meilleure division pour chaque arbre, None signifie que toutes les caractéristiques sont utilisées
- **criterion** : ["gini", "entropy"] la fonction pour mesurer la qualité d'une division dans chaque arbre

Voici le diagramme d'importance des caractéristiques pour le modèle Random Forest avec les meilleurs paramètres trouvés : Les meilleurs paramètres trouvés sont sont illustrés dans Table ??.

## 0.5 Réduction de dimensionnalité

Comme toujours, Random Forest peut bénéficier de la réduction de dimensionnalité puisque il est basé sur des arbres de décision. Nous avons utilisé l'analyse en composantes principales (PCA) pour réduire la dimensionnalité des données avec une variance expliquée de 90%. Nous avons eu le meme résultat que celui des arbres de decision (car le nombre de caractéristiques est le meme grace au meme type d'encodage). Après avoir appliqué PCA, le nombre de caractéristiques est passé de 38 à 1. 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 également illustrés dans Table ??.

## 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 :

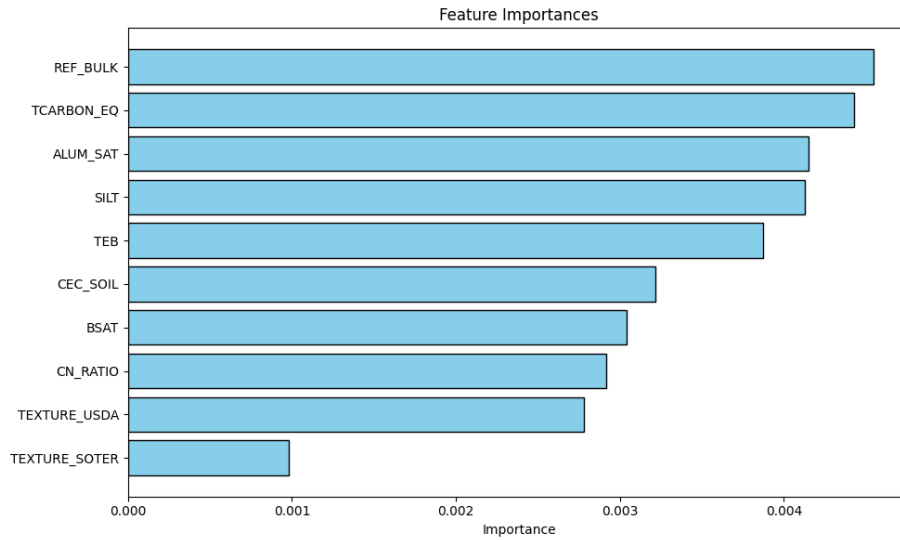


Figure 6: Random Forest - Feature Importance

Configuration	Criterion	N Estimators	Max Features
Sans réduction de dimensionnalité	gini	100	log2
Avec réduction de dimensionnalité	gini	100	log2

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

On observe que la réduction de dimensionnalité a eu un impact négatif sur les performances de l'algorithme Random Forest dans ce cas, contrairement à l'arbre de décision. Cependant, elle a permis de réduire significativement le temps de calcul et les ressources nécessaires pour l'entraînement et la prédiction du modèle en réduisant tous les caractéristiques du dataset en une seule composante. On remarque aussi que les paramètres optimaux restent les mêmes avant et après la réduction de dimensionnalité.

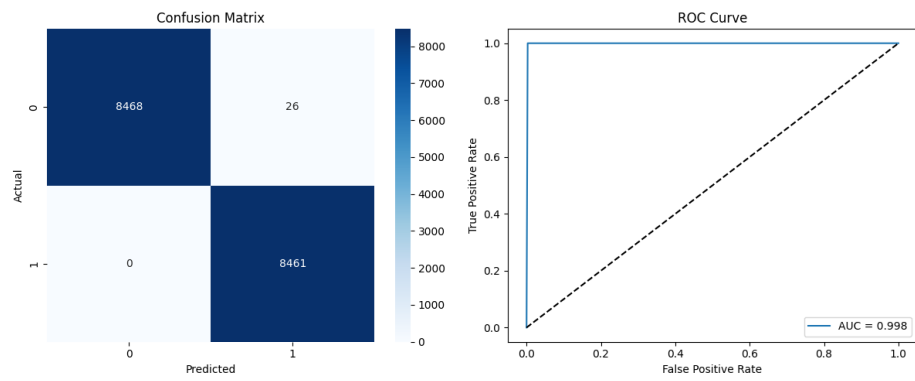


Figure 7: Random Forest - Final Results with no PCA

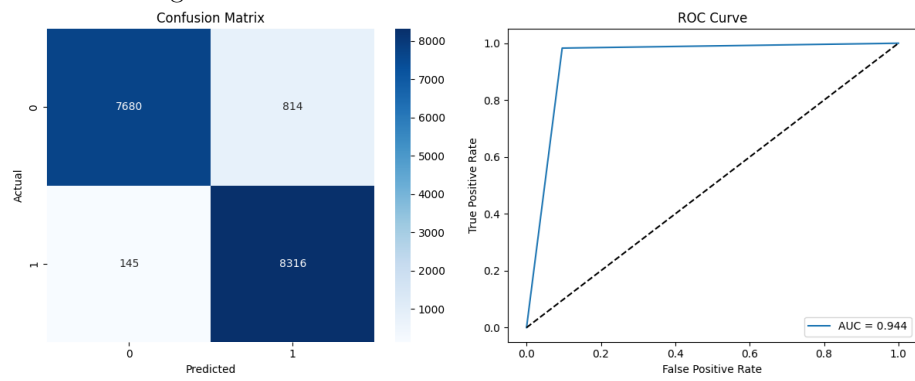


Figure 8: Random Forest - Final Results with PCA

<b>Configuration</b>	<b>Accuracy</b>	<b>Precision</b>	<b>Recall</b>	<b>F1 Score</b>
Sans réduction de dimensionnalité	0.9985	0.9969	1	0.9985
Avec réduction de dimensionnalité	0.9434	0.9108	0.9829	0.9455

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