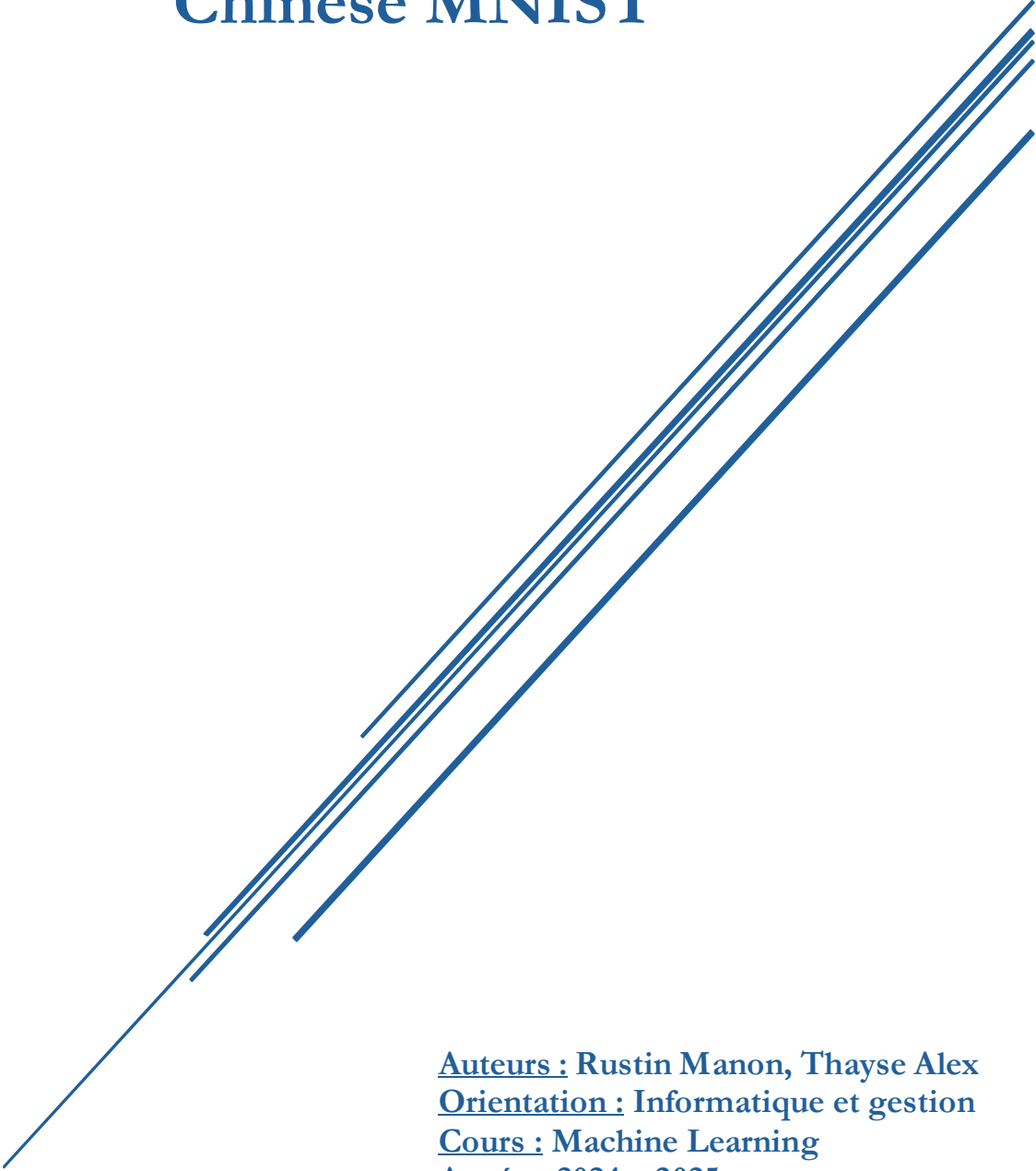


MACHINE LEARNING – Classification supervisée appliquée sur un jeu de données de type Chinese MNIST



Auteurs : Rustin Manon, Thayse Alex
Orientation : Informatique et gestion
Cours : Machine Learning
Année : 2024 – 2025
Professeur : Siebert Xavier
Assistant : Baeckelant Timothy

Table des matières

1. INTRODUCTION.....	1
1. COMPRENDRE LES DONNÉES	1
1.1. DIMENSION ET TYPE DES DONNÉES	1
1.2. DISTRIBUTIONS DES CLASSES.....	1
1.3. CHARGEMENT DES DONNÉES	1
2. ÉVALUATION DES CLASSIFIEURS	2
2.1. MÉTHODE DE VALIDATION	2
2.2. ESSAI DE CLASSIFIEURS.....	2
2.2.1. K plus proches voisins	3
2.2.2. Random Forest	6
2.2.3. Séparateurs à vaste marge (SVM).....	9
2.2.4. Bayes Naïf :	12
2.2.5. Perceptron	14
2.2.6. Multilayer perceptron.....	16
2.2.7. Régression logistique.....	18
2.3. COMPARAISON DES RÉSULTATS	20
3. AMÉLIORATION DES MODÈLES	21
3.1. RÉDUCTION DE LA DIMENSIONNALITÉ	21
3.2. AUGMENTATION DES DONNÉES	21
3.3. ENSEMBLES.....	22
3.3.1. Le Voting.....	22
3.3.2. Le Stacking	23
3.3.3. Comparaison.....	23
3.4. TRANSFER LEARNING	23
3.5. SUPPRESSION CLASSES MAL REPRÉSENTÉES.....	24
3.5.1. Suppression de la classe 1000	24
4. CONCLUSION.....	24
4.1. RÉSUMÉ DES RÉSULTATS	24
4.2. PERSPECTIVES D'AMÉLIORATION	25

Table des Figures

Figure 1 : Evolution de l'erreur en fonction du nombre de voisins.....	3
Figure 2 : KNN - Rapport de classification pour $k = 1$	4
Figure 3 : KNN – Courbe ROC pour $k = 1$	4
Figure 4 : KNN – Rapport de classification pour $k = 5$	4
Figure 5 : KNN – Courbe ROC pour $k = 5$	4
Figure 6 : KNN – Matrice de confusion pour le modèle optimisé.....	5
Figure 7 : Evolution de l'erreur en fonction du nombre d'arbres.....	6
Figure 8 : Random Forest – Rapport de classification pour $n = 350$	7
Figure 9 : Random Forest – Courbe ROC pour $n = 350$	7
Figure 10 : Random Forest – Matrice de confusion pour $n = 350$	8
Figure 11 : Evolution de l'erreur en fonction du paramètre C	9
Figure 12 : SVM – Rapport de classification pour $C = 0.01$	10
Figure 13 : SVM – Courbe ROC pour $C = 0.01$	10
Figure 14 : SVM – Matrice de confusion pour $C = 0.01$	10
Figure 15 : Chiffre 10.....	11
Figure 16 : Chiffre 1000.....	11
Figure 17 : Chiffre 9.....	11
Figure 18 : Chiffre 10 000 000.....	11
Figure 19 : Bayes Naïf – Rapport de classification.....	13
Figure 20 : Bayes Naïf – Courbe ROC.....	13
Figure 21 : Evolution de l'erreur en fonction du paramètre σ	14
Figure 22 : Perceptron – Rapport de classification.....	15
Figure 23 : Perceptron – Courbe ROC.....	15
Figure 24 : Perceptron – Matrice de confusion.....	15
Figure 25 : Evolution de l'erreur en fonction du nombre de couches cachées.....	16
Figure 26 : MLP – Rapport de classification.....	17
Figure 27 : MLP – Courbe ROC.....	17
Figure 28 : MLP – Matrice de confusion.....	17
Figure 29 : Evolution de l'erreur en fonction du paramètre C	18
Figure 30 : Régression logistique – Rapport de classification.....	19
Figure 31 : Régression logistique – Courbe ROC.....	19
Figure 32 : Régression logistique – Matrice de confusion.....	19
Figure 33 : Variance expliquée cumulée.....	21

Table des Tableaux

Tableau 1 : Récapitulatif des hyperparamètres.....	3
Tableau 2 : Comparaison des erreurs sur les données.....	3
Tableau 3 : Récapitulatif des hyperparamètres pour Random Forest.....	6
Tableau 4 : Comparaison des erreurs sur les données.....	6
Tableau 5 : Récapitulatif des hyperparamètres pour SVM	9
Tableau 6 : SVM - Comparaison des erreurs sur les données.....	9
Tableau 7 : Récapitulatif des hyperparamètres pour Bayes Naïf.....	12
Tableau 8 : Bayes Naïf - Comparaison des erreurs sur les données.....	12
Tableau 9 : Récapitulatif des hyperparamètres du Perceptron.....	14
Tableau 10 : Perceptron – Comparaison des erreurs sur les données.....	14
Tableau 11 : Récapitulatif des hyperparamètres du Multilayer Perceptron.....	16
Tableau 12 : MLP – Comparaison des erreurs sur les données.....	16
Tableau 13 : Récapitulatif des hyperparamètres pour la régression logistique.....	18
Tableau 14 : Comparaison des différentes méthodes.....	20
Tableau 15 : Comparaison de la précision sur les données initiales et les données augmentées.....	22
Tableau 16 : Comparaison des précisions sur les données augmentées avant et après suppression de la classe 1000.....	24

1. Introduction

Le Machine Learning, et en particulier la classification, est devenu un outil clé dans divers domaines pour prendre des décisions éclairées à partir des données. Il est utilisé dans des applications variées telles que le diagnostic médical à partir d'images, les systèmes de recommandation de produits ou la détection de fraudes dans les transactions bancaires. L'identification du classifieur le plus adapté est essentielle pour optimiser les performances d'un système et ainsi garantir des prédictions précises.

Ce projet vise à identifier le meilleur algorithme de classification pour notre jeu de données, en testant différents modèles et en appliquant des méthodes d'optimisation. L'objectif est d'améliorer la précision du modèle tout en relevant les défis associés aux données d'images et des risques de surapprentissage.

1. Comprendre les Données

1.1. Dimension et type des données

Pour obtenir un bref aperçu du jeu de données utilisé, nous examinons sa taille, ainsi que le type de variables qu'il contient. Dans notre cas, les données se composent de 15000 instances et de 5 attributs :

- *suite_id* : Identifiant de la personne j ;
- *sample_id* : Identifiant relatif à l'essai i écrit par la personne j du chiffre k ;
- *code* : Code numérique associé au chiffre (ex. : code 1 pour le chiffre 0).
- *value* : Représente la valeur numérique du chiffre (ex. : 9) ;
- *character* : Le caractère chinois correspondant (ex. : 九).

Ici, chaque échantillon est une image de taille standardisée, 64×64 pixels.

Les données ne contiennent aucune valeur manquante, ce qui facilite leur traitement ultérieur.

1.2. Distributions des classes

La distribution des classes dans le jeu de données considéré est uniforme. Chaque classe est représentée par le même nombre d'échantillons (1000 échantillons par chiffre). Cette répartition permet de réduire le risque que le modèle privilégie certaines classes au détriment des autres.

Néanmoins, certaines classes peuvent être visuellement similaires (traits ou structures proches), ce qui pourrait entraîner une baisse de performances des différents modèles (diminution des scores de précision ou de rappel). Par ailleurs, les classes associées à des caractères complexes pourraient également causer des confusions dans les prédictions du modèle.

Nous devons donc approfondir les analyses (ex. : exploration des matrices de confusion) et éventuellement effectuer une transformation des données.

1.3. Chargement des données

Dans notre projet, nous avons choisi de conserver une taille d'image de 28×28 pixels pour les images de Chinese MNIST, plutôt que de les redimensionner à 64×64 pixels, pour plusieurs raisons pratiques et techniques. Tout d'abord, en augmentant la taille des images à 64×64 , nous passons de 784 pixels à 4096 pixels par image, ce qui augmente considérablement la complexité du modèle sans offrir d'informations supplémentaires pertinentes. Nous avons en effet testé l'ensemble des modèles pour ces dimensions, et les résultats ne furent que peu concluants (sans méthode d'ensembles, la précision la plus élevée obtenue fut de 71,5% pour Random Forest). Nous en concluons que l'augmentation de la dimension introduit plutôt plus de bruit et de détails inutiles qui peuvent nuire aux performances du modèle.

En choisissant de travailler avec des images de 28x28 pixels, nous avons opté pour un compromis optimal entre la taille des données et la capacité de notre modèle à extraire les caractéristiques utiles pour la classification. Cette taille est suffisante pour capturer les motifs clés des chiffres Chinese MNIST tout en restant relativement légère pour éviter les problèmes liés à une dimensionnalité trop élevée. De plus, cela permet de réduire le risque de surapprentissage, car le modèle a moins de données à apprendre tout en étant suffisamment informatif pour effectuer une bonne classification.

Nous précisons que nous avons choisi un redimensionnement de 28x28 pixels car il s'agit de valeurs standards largement utilisées dans les jeux de données tels que celui considéré dans ce projet. Ceci peut donc permettre une comparaison facile avec d'autres travaux de recherche. Comme il s'agit d'une taille largement validée par la communauté scientifique, et que la taille 64x64 ne nous semble pas appropriée pour les méthodes testées dans le cadre de ce projet¹, nous sélectionnons 28x28 pixels.

2. Évaluation des Classifieurs

2.1. Méthode de validation

Pour évaluer la performance des modèles, nous avons utilisé une validation croisée à 5 folds. Cela permet de maximiser l'utilisation des données disponibles, de réduire la variance des estimations et ainsi d'éviter le risque de surapprentissage. Nous avons mesuré la performance à l'aide des métriques suivantes : précision, rappel, F1-score et AUC-ROC, afin de tenir compte du déséquilibre des classes dans le jeu de données.

Nous avons effectué les tests selon deux configurations :

1. Lorsque les données ne sont pas choisies aléatoirement, par exemple en prenant les 10 premières personnes.
2. Lorsque les données sont sélectionnées de manière aléatoire, en respectant la proportion des classes.

Pour chacune de ces configurations, nous avons tracé des graphiques montrant l'évolution de l'erreur d'entraînement et de l'erreur de test en fonction d'un paramètre clé de la méthode concernée. Ces graphiques permettent d'observer la convergence des modèles, ainsi que l'impact de la manière dont les données sont choisies sur la performance.

Une recherche d'hyperparamètres a été réalisée via une *grid search* combinée avec la validation croisée afin d'optimiser les modèles. Les performances finales ont été comparées sur les données de test en utilisant les mêmes métriques.

En complément, nous avons généré des matrices de confusion pour les différents modèles afin de mieux comprendre leurs erreurs. La matrice de confusion nous permet de visualiser le nombre de prédictions correctes et incorrectes pour chaque classe, en distinguant les *vrais positifs* (VP), *faux positifs* (FP), *vrais négatifs* (VN) et *faux négatifs* (FN). Ces informations sont essentielles pour analyser en détail les performances.

2.2. Essai de classifieurs

Après diverses analyses sur les méthodes des k plus proches voisins et de Random Forest présentes respectivement en Annexe 1 et en Annexe 2, nous avons choisi d'analyser les différents classifieurs sur les données séparées aléatoirement en données d'entraînement et en données test en raison de résultats significativement meilleurs (erreur plus faible, précision en validation croisée plus élevée). Par ailleurs, ces analyses nous ont guidé vers une valeur de données d'entraînement de 60%, ce qui fait que l'ensemble test comprend 40% des données initiales. En effet, les résultats n'étaient pas significativement beaucoup plus élevés au-delà de ces valeurs, et ils prenaient beaucoup plus de temps – indicateur fondamental à prendre

¹ Dans le cadre d'un projet où nous testerions des méthodes d'apprentissage plus avancées, notamment dans le domaine du deep learning, des résolutions plus élevées telles que 64x64 pixels seraient peut-être plus adaptées.

en compte également. Cette première approche nous permet d'avoir une idée de comparaison entre les différents classifieurs que nous testons ci-dessous.

2.2.1. K plus proches voisins

L'algorithme des k-plus-proches-voisins (k-nn) est une méthode d'apprentissage automatique non paramétrique reconnue pour sa simplicité et sa flexibilité. Il ne repose sur aucune hypothèse préalable concernant la distribution des données, ce qui lui permet de traiter efficacement des données numériques dans des tâches de classification et de régression. Son principe repose sur l'identification des k points les plus proches d'un échantillon donné selon une métrique de distance, telle que la distance euclidienne. La prédiction est ensuite réalisée par vote majoritaire dans le cas de la classification.

Les hyperparamètres sélectionnés pour cette méthode sont repris dans le Tableau 1.

Hyperparamètres	Valeur(s) testée(s)	Description
Nombre de voisins (k)	1 à 20	Nombre de voisins pris en compte pour la précision.
Métrique de distance	Distances euclidienne, Manhattan, Chebychev, Minkowski	Type de distance utilisée pour mesurer la similarité entre les points.
Pondération	Uniforme, Distance	Poids attribué à chaque voisin lors du calcul.

Tableau 5 : Récapitulatif des hyperparamètres

Résultats obtenus

Pour évaluer les performances du modèle, nous traçons l'erreur sur les données d'entraînement et l'erreur sur les données de test en fonction du paramètre clé de cette méthode, k le nombre de voisins. La Figure 1 présente les résultats lorsque les données d'entraînement et les données de test sont générés aléatoirement. Le Tableau 2 nous donne quant à lui les valeurs chiffrées, ainsi que la précision.

k	Erreur d'entraînement	Erreur de test	Précision
1	0.0000	0.2875	0.7125
2	0.1698	0.3533	0.6467
3	0.1677	0.3307	0.6693
4	0.1949	0.3293	0.6707
5	0.1988	0.3202	0.6798
6	0.2184	0.3273	0.6727
7	0.2253	0.3288	0.6712
8	0.2381	0.3318	0.6682
9	0.2453	0.3338	0.6662
10	0.2562	0.3368	0.6428

Tableau 6 : Comparaison des erreurs sur les données

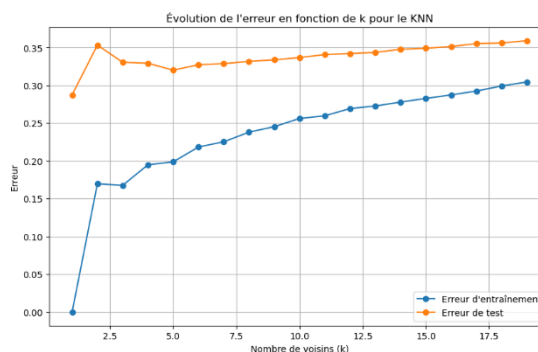


Figure 3 : Evolution de l'erreur en fonction du nombre de voisins

Pour un nombre de voisin égal à 1, l'erreur d'entraînement est nulle, ce qui signifie que le modèle a parfaitement mémorisé les données d'entraînement. Il s'agit d'un signe classique de surapprentissage. L'erreur de test est quant à elle assez élevée (0.2875), indiquant que le modèle ne généralise pas bien aux nouvelles données.

Par après, à mesure que le nombre de voisins augmente, l'erreur d'entraînement augmente également (résultat attendu – le modèle devient moins flexible et moins susceptible de surapprendre). On peut remarquer que l'erreur de test diminue initialement jusqu'à augmenter à nouveau, ce qui suggère qu'un nombre de voisins trop faible conduit à un surapprentissage (variance élevée) tandis qu'un nombre de voisins trop élevé conduit à un sous-apprentissage. Par ailleurs, la précision augmente jusqu'à un nombre de voisins égal à 5 avant de rediminuer. Néanmoins, cette précision est relativement faible pour toutes les valeurs de k, ce qui semble indiquer que le modèle des k plus proches voisins n'est pas adapté pour ce jeu de données.

Pour trouver les meilleurs paramètres du modèle, nous obtenons les résultats suivants avec Grid Search :

- Meilleur k : 1 avec une précision de 0,7125 ;
- Meilleurs paramètres: {'metric': 'Manhattan', 'n_neighbors': 1, 'weights': 'uniform'}.

Comme un nombre de voisins égal à 1 nous semble se rapprocher d'un surapprentissage – et qu'un nombre de voisins égal à 5 paraît être un choix alternatif pertinent, nous analysons plus en détail le rapport de classification ainsi que la courbe AUC-ROC pour les deux cas (Figures 2 à 5).

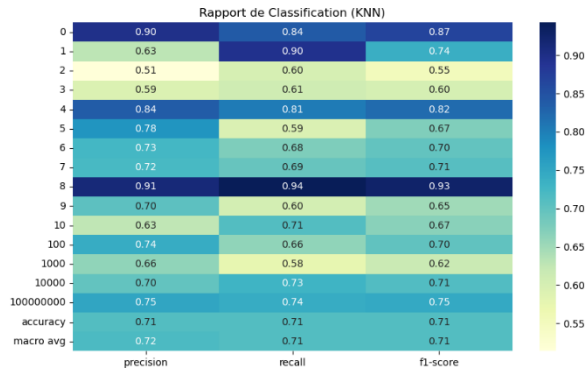


Figure 7 : KNN - Rapport de classification pour $k = 1$

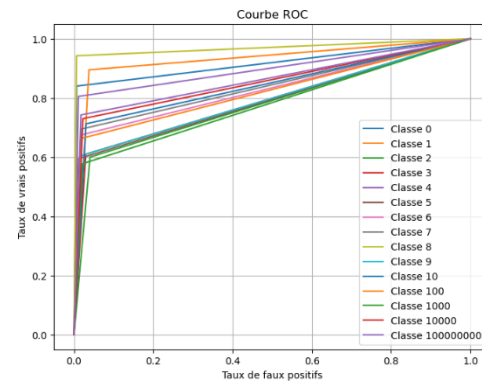


Figure 3 : KNN – Courbe ROC pour $k = 1$

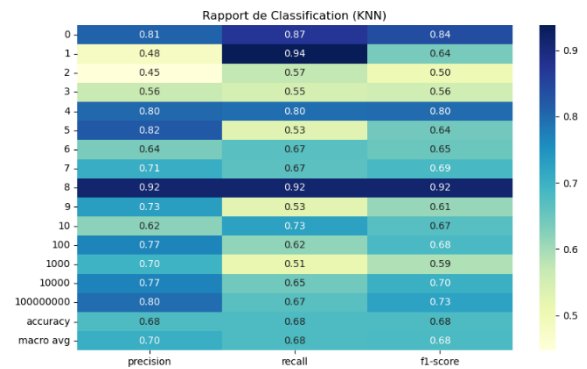


Figure 4 : KNN – Rapport de classification pour $k = 5$

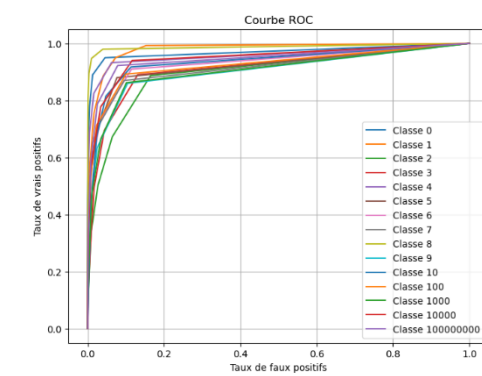


Figure 5 : KNN – Courbe ROC pour $k = 5$

Nous remarquons que la précision moyenne est meilleure pour $k = 1$ (0.72) que pour $k = 5$ (0.70) mais que cette différence n'est pas significative. De plus, cela ne signifie pas que le modèle pour $k = 1$ est préférable à tout point de vue. En effet, lorsque le nombre de voisins est aussi faible, le modèle s'adapte davantage aux données d'entraînement, ce qui peut mener à un surapprentissage. Un nombre de voisins plus élevé rend le modèle plus robuste, mais au prix d'une légère perte de précision.

Si on s'intéresse aux valeurs moyennes, la précision, le F1-score et le rappel sont sensiblement proches entre les modèles où $k = 1$ et $k = 5$.

Les courbes ROC apportent davantage d'informations pour conclure sur les paramètres optimaux du modèle. Pour $k = 1$, les courbes ROC sont moins bonnes (l'aire sous la courbe est plus faible dans la majorité des classes). Ceci peut s'expliquer par le fait que ce modèle est plus sensible au bruit et aux variations dans les données – les courbes paraissent donc moins lisses. Le modèle montre donc des performances plus élevées sur les données d'entraînement, mais il ne généralise pas correctement (surapprentissage). Pour le modèle où $k = 5$, les courbes ROC sont plus stables, même si la précision est légèrement inférieure. La stabilité est meilleure dans l'identification des classes pour ce modèle – c'est-à-dire que ce dernier est moins sensible aux points aberrants dans les données (la décision finale est basée sur plusieurs voisins) et généralise mieux sur les données de test.

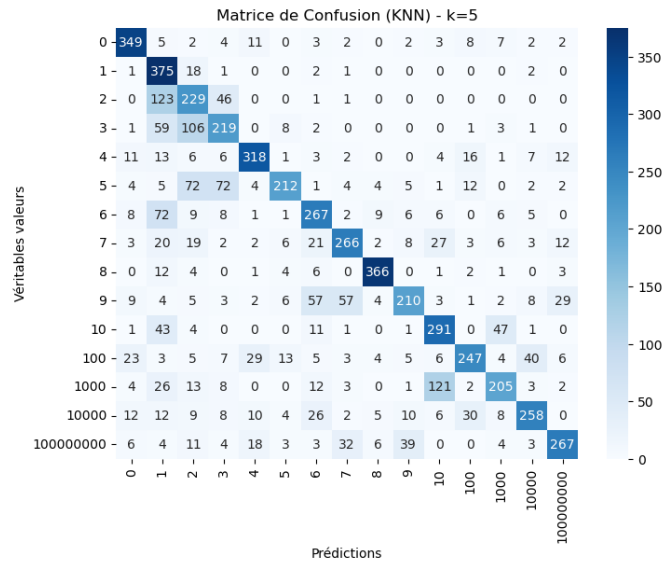


Figure 6 : KNN – Matrice de confusion pour le modèle optimisé

L'analyse de la matrice de confusion pour $k = 5$ est représentée sous forme de heatmap à la Figure 6. Les valeurs situées sur la diagonale représentent les prédictions correctes (vrais positifs), tandis que les autres cases indiquent des erreurs de classification. Certaines classes sont très bien identifiées, comme la classe 1 avec 375 exemples correctement classés ou la classe 8 avec 366 exemples. Ces performances montrent que l'algorithme parvient à identifier ces classes de manière fiable.

Certaines classes présentent cependant des confusions. Par exemple, la classe 2 est souvent confondue avec la classe 1 (123 cas). On remarque aussi que la classe 5 est assez mal représentée en comparaison avec les autres classes, puisqu'on ne recense que 212 vrais positifs sur 395 exemples. C'est également le cas pour la classe 100. Ce déséquilibre affecte négativement les performances globales et indique que l'algorithme est biaisé en faveur des classes majoritaires.

En termes de métriques, les classes bien représentées comme 0 ou 8 bénéficient d'une précision, d'un rappel et d'un f1-score élevés car leurs prédictions sont majoritairement correctes (peu de faux négatifs et de faux positifs). En revanche, les classes comme 2 ou 3 souffrent d'une précision et d'un rappel faibles (respectivement 0.45-0.57 et 0.56-0.55), car elles sont classées à tort comme des autres classes. La classe 1, quant à elle, montre un rappel élevé (car il y a peu de faux négatifs) mais une précision très faible (0.48) à cause des nombreux faux positifs.

Analyse des classes problématiques

Certaines classes diminuent considérablement les performances du modèle puisqu'elles sont souvent mal identifiées. La classe 100, par exemple, est confondue avec pratiquement toutes les autres classes (ce qui implique que le rappel est faible), et ce de manière conséquente – on ne recense que 247 vrais positifs. Néanmoins, peu de classes sont classées à tort comme étant la classe 100, ce qui confère à cette classe une précision relativement élevée de 0.77. Ces observations sont similaires pour les classes 1000, 10000, 1000000000, ce qui leur confère un rappel assez faible mais une précision relativement acceptable.

Limites identifiées pour ce modèle

Le jeu de données considéré contient des images de 28x28 pixels, ce qui signifie un espace de 784 dimensions. En haute dimension, les points de données ont tendance à être plus dispersés, ce qui rend les voisins moins significatifs. K-nn est donc sensible à la dimensionnalité du jeu de données.

Par ailleurs, k-nn repose uniquement sur des distances entre les points dans l'espace des caractéristiques. Il ne peut pas capturer des relations complexes entre des pixels voisins.

2.2.2. Random Forest

L'algorithme de Random Forest est une méthode d'apprentissage automatique supervisée qui utilise un ensemble d'arbres de décision pour améliorer la précision et contrôler le surapprentissage. Il repose sur la création de multiples arbres de décision indépendants à partir de sous-échantillons des données d'entraînement. Pour les tâches de classification, la prédiction finale est déterminée par un vote majoritaire des arbres. Cette méthode réduit le risque de surapprentissage, est robuste face aux données bruitées et peut gérer des données de grande dimension, bien qu'elle puisse être coûteuse en termes de calcul et moins interprétable en raison du grand nombre d'arbres.

Les hyperparamètres sélectionnés pour cette méthode sont repris dans le Tableau 3.

Hyperparamètres	Valeur(s) testée(s)	Description
Nombre d'arbres	10 à 1000	Nombre d'arbres dans la forêt
Profondeur maximale	5 à 50, None	Profondeur maximale des arbres
Critère	Gini, entropie	Fonction de mesure de la qualité des divisions
Nombre de caractéristiques	Sqrt, log2, None	Nombre de caractéristiques à considérer pour trouver la meilleure division

Tableau 7 : Récapitulatif des hyperparamètres pour Random Forest

Résultats obtenus

Nombre d'arbres	Erreur d'entraînement	Erreur de test	Précision
10	0.0040	0.3585	0.6415
50	0.0000	0.2583	0.7417
100	0.0000	0.2433	0.7567
150	0.0000	0.2358	0.7642
200	0.0000	0.2308	0.7692
250	0.0000	0.2333	0.7667
300	0.0000	0.2307	0.7693
350	0.0000	0.2307	0.7693
400	0.0000	0.2273	0.7727
700	0.0000	0.2253	0.7747
1000	0.0000	0.2243	0.7757

Tableau 8 : Comparaison des erreurs sur les données

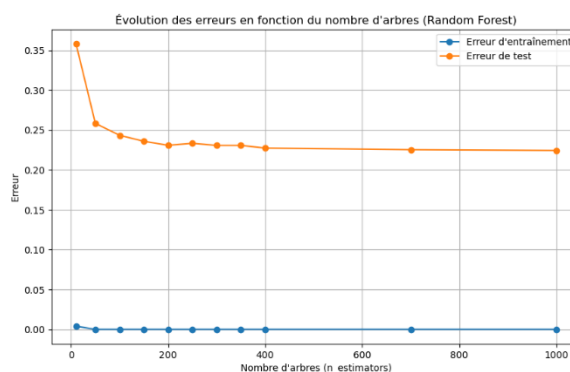


Figure 7 : Evolution de l'erreur en fonction du nombre d'arbres

Pour évaluer les performances du modèle, nous traçons l'erreur sur les données d'entraînement et l'erreur sur les données de test en fonction du paramètre clé de cette méthode, le nombre d'arbres. La Figure 7 présente les résultats lorsque les données d'entraînement et les données de test sont générés aléatoirement. Le Tableau 4 nous donne quant-à-lui les valeurs chiffrées, ainsi que la précision en validation croisée.

En observant nos valeurs, nous constatons que l'erreur d'entraînement devient nulle dès que le modèle atteint 50 arbres. Cela montre que notre modèle s'ajuste parfaitement aux données d'entraînement. Cependant, cette erreur d'entraînement nulle peut également indiquer un risque de surapprentissage, où notre modèle risque de trop bien s'ajuster aux données d'entraînement.

Nous notons également que l'erreur de test diminue à mesure que nous augmentons le nombre d'arbres, passant de 0,3585 avec 10 arbres à 0,2253 avec 700 arbres. Cette diminution indique que notre modèle améliore sa précision sur des données inconnues, ce qui reflète une meilleure généralisation. Toutefois, nous observons que cette réduction ralentit à partir d'environ 400 arbres : ajouter plus d'arbres au-delà de ce seuil n'apporte que de faibles améliorations.

En ce qui concerne la précision, nos résultats montrent une progression constante, allant de 64,15 % avec 10 arbres à 77,57 % avec 1000 arbres. Cependant, ici aussi, l'amélioration de la précision devient minimale après 400 arbres. Cela suggère que nous n'avons pas besoin d'un nombre excessif d'arbres pour obtenir de bonnes performances, surtout si nous devons tenir compte du temps et des coûts de calcul supplémentaires.

L'analyse des données montre que, dans la plage 350-700, le modèle atteint un bon équilibre entre deux objectifs : maximiser la généralisation sur des données inconnues et minimiser le coût de calcul (entre autres le temps). On remarque d'ailleurs sur la Figure 8 que l'erreur de test stagne à partir d'un certain nombre d'arbres pour devenir pratiquement constante.

Pour trouver les meilleurs paramètres du modèle, nous obtenons les résultats suivants avec Grid Search :

- Meilleurs paramètres : {'max_depth' : None, 'max_features' : 'sqrt', 'n_estimators' : 500} ;
- Meilleure précision : 0,7760 ;

La recherche 'Grid Search' nous donne un nombre d'arbres maximal puisque c'est pour un nombre d'arbres maximal testé que la précision est la plus élevée. Néanmoins, nous devons prendre en compte le temps de calcul pour choisir une réponse optimale. Comme l'erreur sur les données test ne diminue plus de manière significative à partir d'un nombre d'arbres de 350, nous optons – de manière un peu arbitraire – pour ce nombre d'arbres-là, qui nous semble être un bon compromis².

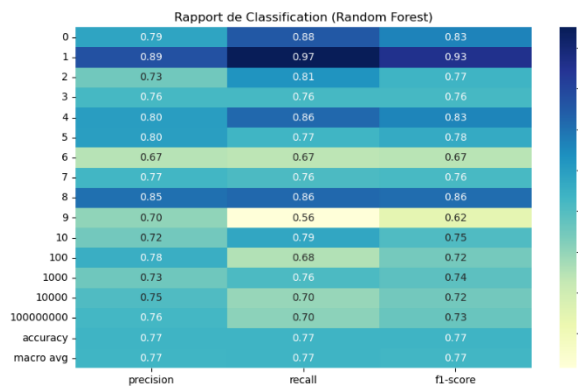


Figure 8 : Random Forest – Rapport de classification pour n = 350

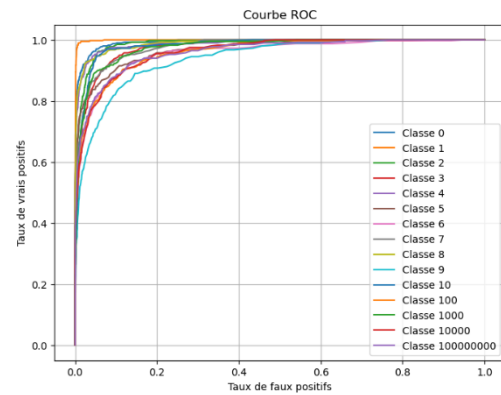


Figure 9 : Random Forest – Courbe ROC pour n = 350

Dans le rapport de classification pour un nombre d'arbres égal à 350 (Figure 9), les résultats montrent des performances plutôt positives entre les classes, et meilleures que pour k-nn. Par exemple, les classes 1, 4 et 8 semblent bien représentées par le modèle puisqu'elles ont toutes une précision, un rappel et un F1-score globalement élevés (supérieurs à 0,8). En revanche, d'autres classes comme 6 et 9 obtiennent des résultats bien plus faibles, ce qui indique que le modèle a plus de difficultés à faire des prédictions correctes.

Concernant la courbe ROC, on remarque que certaines courbes comme celles des classes 1 et 10 sont proches du coin supérieur gauche – indiquant que le modèle distingue bien ces classes. Cependant, d'autres courbes définissent une aire bien inférieure, notamment pour la classe 9. Ces résultats sont confirmés par le rapport de classification, si l'on se concentre majoritairement sur la valeur du rappel. Globalement, les courbes sont quand même assez proches du coin supérieur gauche, par ailleurs l'AUC-ROC du modèle optimisé est de 0,9739.

² Sur python, nous avons testé l'analyse pour 500 arbres et les résultats étaient très similaires à ceux obtenus pour 350 arbres. Pour 250 arbres en revanche, certaines différences nous semblaient encore significatives, notamment dans la matrice de confusion – bien que la précision ne soit pas très différente.

La précision globale du modèle est de 0,77 ce qui signifie que certaines classes causent encore une certaine baisse de performance puisque seulement 77% des prédictions sont correctes. Analysons plus en détails la matrice de confusion (Figure 11) pour mieux comprendre les performances sous-jacentes.

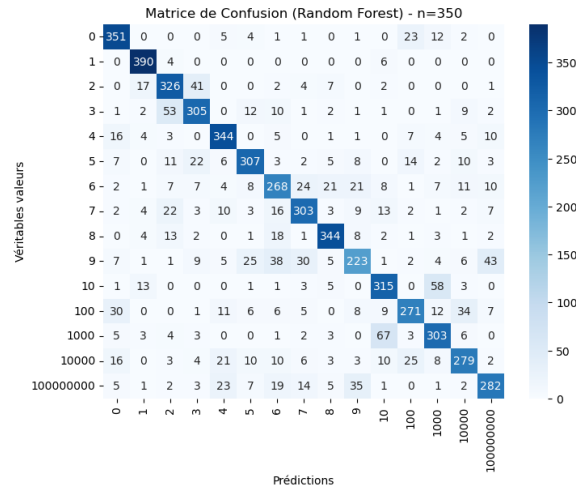


Figure 10 : Random Forest – Matrice de confusion pour n = 350

L'allure générale de la matrice semble d'un premier abord bien meilleure que celle relative à k-nn. En effet, il semble y avoir une majorité de vrais positifs (diagonale en bleu foncé) et beaucoup moins de faux positifs (le reste de la matrice en bleu clair – magnitude comprise entre 0 et 60).

Comme pour knn, la classe 1 est la classe qui comprend un nombre conséquent de vrais positifs (390 sur 400). Le modèle est excellent pour reconnaître cette classe – le rappel est de 97%. C'est également le cas des classes 0 et 8. Ceci explique le rappel élevé des classes 0, 1 et 8 dans le rapport de classification. En revanche, les classes 9 et 10000 sont les classes pour lequel le modèle fait le plus de mauvaises prédictions puisqu'une grande partie des données de test appartenant à ces classes sont mal identifiées (respectivement 223 sur 400 et 279 sur 400). Ces classes ont donc quant à elles un très mauvais rappel.

Analyse des classes problématiques

Nous réalisons ici que la baisse de performances semble liée à des classes ayant à la fois une mauvaise précision et en même temps un mauvais rappel, ces classes sont d'ailleurs bien trop souvent classées à tort. C'est notamment le cas des classes 6, 9 et 100 dont les F1-score respectifs s'élèvent à 0.67, 0.62 et 0.72.

Limites identifiées pour ce modèle

Le modèle de Random Forest nécessite une quantité significative de mémoire et de temps pour construire un grand nombre d'arbres avec des ensembles de données aussi volumineux que Chinese MNIST.

Par ailleurs, le modèle Random Forest n'a pas de mécanisme intégré pour gérer les variations comme la rotation, la mise à l'échelle ou les déformations. Le modèle est donc très limité en termes de robustesse face à des données qui diffèrent légèrement de l'ensemble d'entraînement.

2.2.3. Séparateurs à vaste marge (SVM)

Les séparateurs à vastes marges cherchent à maximiser la distance entre la frontière de l'hyperplan de décision et les points les plus proches de chaque classe, appelés vecteurs de support. Ce principe repose sur l'idée qu'un hyperplan avec une marge large offre une meilleure généralisation en réduisant le risque de surapprentissage. Mathématiquement, l'hyperplan est défini par l'équation

$$\langle w, x \rangle + b = 0$$

où w est le vecteur normal et b le biais. L'optimisation vise à maximiser la distance entre les deux marges et donc à minimiser

$$L(w) = \frac{\|w\|^2}{2}$$

sous la contrainte que tous les points soient « du bon côté », c'est-à-dire que $y_i \cdot (\langle w, x_i \rangle + b) \geq 1$. Pour les données non linéairement séparables, des fonctions noyau comme le RBF ou le polynôme sont utilisées pour projeter les données dans des espaces de caractéristiques de dimension supérieure.

Les hyperparamètres sélectionnés pour cette méthode sont repris dans le Tableau 5.

Hyperparamètres	Valeur(s) testée(s)	Description
Type de noyau	Linéaire, RBF, polynomial	Transformation des données pour gérer les cas non linéaires.
Paramètre C	10^{-3} à 10^3	Contrôle le compromis entre marge large et classification correcte.
Paramètre γ	0,001 ; 0,01 ; 0,1 ; « scale »	Définit l'influence d'un seul point sur la décision dans les noyaux non linéaires.
Degré du noyau	2, 3	Relatif la complexité du noyau polynomial.

Tableau 5 : Récapitulatif des hyperparamètres pour SVM

Résultats obtenus

Nombre d'arbres	Erreur d'entraînement	Erreur de test	Précision
0.001	0.0561	0.252	0.7480
0.0046	0.0152	0.2198	0.7802
0.0215	0.0026	0.2193	0.7807
0.1	0.0002	0.2238	0.7762
0.4642	0.0000	0.2252	0.7748
2.1544	0.0000	0.2252	0.7748
46.4159	0.0000	0.2252	0.7748

Tableau 6 : SVM - Comparaison des erreurs sur les données

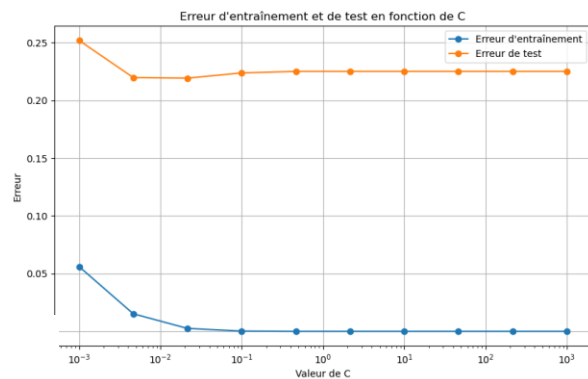


Figure 11 : Evolution de l'erreur en fonction du paramètre C

En sortie du Grid Search, les meilleurs paramètres obtenus indiquent un coefficient C égal à 0.01^3 , un noyau de type polynomial de degré 3, et une valeur de γ fixée à « 0.1 ». L'effet du paramètre C sur les erreurs d'entraînement et de test est repris à la Figure 12.

³ Lorsque nous avons effectué les tests en 64x64 pixels, nous avons obtenu un C = 10. Ceci s'explique par le fait qu'avec des images 64x64, le modèle doit être plus flexible pour s'adapter à la complexité des données, d'où un C plus élevé (moins de régularisation). En 28x28, il y a moins de bruit et moins de surajustement, ce qui favorise un C plus faible (régularisation plus forte) pour éviter le surapprentissage.

L'évolution des erreurs en fonction de C (Tableau 5) révèle une tendance typique : les erreurs sont initialement assez élevées (0.252) et diminuent progressivement jusqu'à $C = 0.1$. Au-delà, l'erreur d'entraînement continue de diminuer, mais l'erreur de test commence à augmenter, avant de stagner, indiquant un surapprentissage du modèle. Une valeur de C égale à 0.1 nous semble donc être un bon compromis entre une marge suffisamment large et bonne généralisation des données, tout en évitant un surapprentissage.

Concernant les précisions, le modèle optimisé nous permet d'atteindre, dans cette première approche de l'analyse, une précision de 78.4%, ce qui nous semble meilleur que les méthodes précédemment testées.

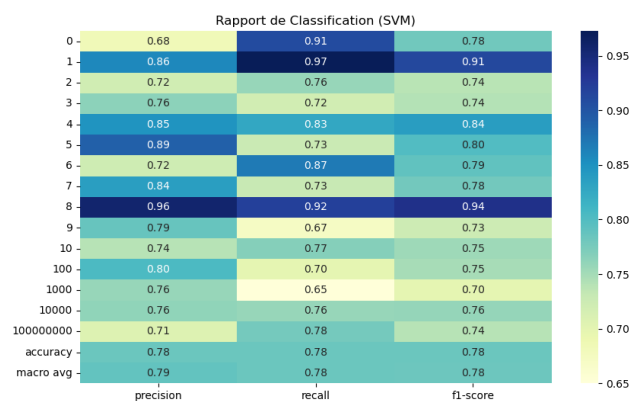


Figure 12 : SVM – Rapport de classification pour C = 0.01

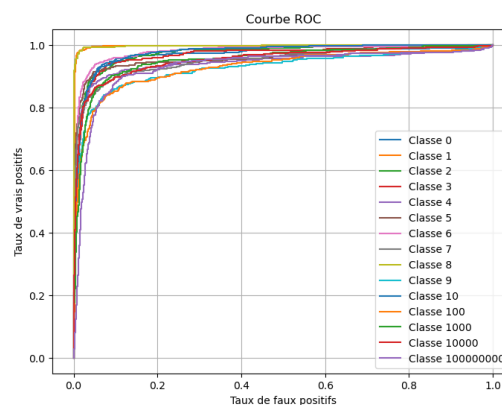


Figure 13 : SVM – Courbe ROC pour C = 0.01

Le rapport de classification relatif au modèle optimisé (Figure 13) met en avant que les classes 1, 4 et 8 sont très bien représentées puisqu'elles présentent une précision, un rappel et un F1-score relativement élevés. Les classes 1 et 8 sont très bien identifiées puisque leur rappel est très élevé (respectivement 0.97 et 0.92) en comparaison avec les autres classes. Les courbes AUC-ROC (Figure 14) confirment par ailleurs que le modèle est performant pour ces deux dernières classes puisque les courbes relatives à ces classes sont très proches du coin supérieur gauche – ce qui signifie qu'elles ont un taux de vrais positifs très élevé pour un taux de faux positifs relativement faible.

Certaines classes semblent cependant baisser les performances du modèle, notamment les classes 9 et 1000 qui présentent un faible rappel (respectivement de 0.67 et 0.65). Elles présentent donc un faible nombre de vrais positifs. Leur précision est cependant acceptable (entre 75 et 80%). Les courbes AUC-ROC relatives à ces classes montrent que l'aire sous-tendue à ces courbes est la plus faible en comparaison avec les courbes relatives aux autres classes.

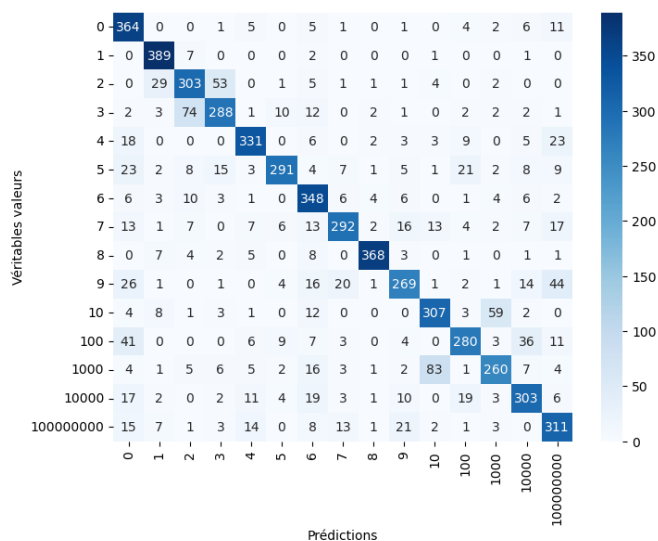


Figure 14 : SVM – Matrice de confusion pour C = 0.01

La matrice de confusion relative à SVM pour un paramètre C égal à 0.01 est représentée à la Figure 15.

La matrice de confusion nous permet de mieux comprendre comment le modèle se comporte avec les différentes classes et de mettre en évidence les erreurs de classification. On remarque que les classes 9, 100 et 1000 ont un nombre assez faible de vrais positifs, ce qui entraîne un rappel faible pour ces classes. Cela signifie que ces classes sont souvent mal classées et font baisser les performances globales du modèle. Par exemple, la classe 1000 est souvent incorrectement classée comme étant 10, avec 83 faux positifs. Ce problème n'est pas nouveau et avait déjà été observé avec les modèles k-NN et Random Forest. En regardant de plus près les caractéristiques visuelles des classes concernées, on constate que les classes 10 et 1000 sont très similaires (respectivement Figures 16 et 17), ce qui complique leur distinction pour le modèle. Cette confusion visuelle rend la tâche du classifieur beaucoup plus difficile, car il est plus compliqué de différencier des classes qui se ressemblent beaucoup sur le plan visuel.

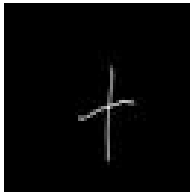


Figure 15 : Chiffre 10

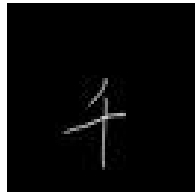


Figure 16 : Chiffre 1000



Figure 17 : Chiffre 9

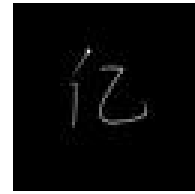


Figure 18 : Chiffre
10 000 000

La classe 9 est quant à elle surtout confondue à la classe 10 000 000. Bien que les similitudes ne soient pas aussi flagrantes qu'entre les classes 10 et 1000, on retrouve quand même des caractéristiques similaires entre les classes concernées (Figures 18 et 19). Il suffit que certaines personnes écrivent un peu trop différemment pour que les chiffres très similaires soient confondus – c'est ce qui semble se produire pour les trois premiers modèles.

Limites identifiées pour ce modèle

L'entraînement d'un modèle SVM, notamment avec des noyaux non linéaires comme le polynomial, peut devenir très coûteux en termes de temps et de ressources. En effet, la complexité de calcul des SVM est généralement quadratique par rapport au nombre d'exemples d'entraînement, ce qui signifie que plus le jeu de données est grand, plus le temps de calcul nécessaire pour entraîner le modèle augmente de manière exponentielle. Pour un jeu de données considéré Chinese MNIST, composé de 15 000 images d'entraînement, l'entraînement est assez long.

De plus, le modèle SVM rencontre des difficultés de scalabilité avec de grandes données comme celui considéré ici. Il est en effet mieux adapté à des jeux de données plus petits ou à faible dimensionnalité, car son coût computationnel augmente rapidement avec la taille du jeu de données.

Enfin, une autre limite majeure des SVM pour notre jeu de données Chinese MNIST est leur difficulté à gérer des problèmes de classification multi classe. Par défaut, les SVM sont des classifieurs binaires, ce qui signifie qu'ils sont conçus pour séparer deux classes. Sur notre jeu de données, qui comprend quinze classes distinctes, il est nécessaire d'étendre le modèle à plusieurs classes, ce qui affecte les performances du modèle.

2.2.4. Bayes Naïf :

Le classifieur de Bayes Naïf est un algorithme de classification supervisée basé sur le Théorème de Bayes supposant une hypothèse d'indépendance conditionnelle des attributs. Ce modèle permet une implémentation et des calculs rapides.

Le Théorème de Bayes exprime que :

$$\underbrace{P(Y = y|X = x)}_{a \text{ posteriori}} \times \underbrace{P(X = x)}_{\text{évidence}} = \underbrace{P(X = x|Y = y)}_{\text{vraisemblance}} \times \underbrace{P(Y = y)}_{a \text{ priori}}$$

Avec un vecteur d'attributs $\mathbf{X} = \{X_1 \dots X_p\}$ et Y la variable aléatoire correspondant à la classe.

L'hypothèse d'indépendance conditionnelle est « naïve » car elle suppose que, lorsqu'on connaît les étiquettes, les attributs sont indépendants. Cependant, cette hypothèse est fautive dans de nombreux cas. Dans le nôtre, elle n'est pas respectée car les pixels adjacents dans une image sont corrélés. Néanmoins, le classifieur bayésien naïf peut quand même être efficace. En effet, l'indépendance conditionnelle réduit la complexité en réduisant le nombre de paramètres du modèle, ce qui limite le surapprentissage. En revanche, cet algorithme rencontre des problèmes lorsque les attributs des données sont fortement corrélés ou identiques à d'autres.

Le type de classifieur bayésien naïf utilisé est le gaussien car il est plus approprié pour des données continues supposées suivre une distribution normale. Dans notre cas, les images de chiffres manuscrits peuvent être considérées comme des données continues car chaque pixel peut être considéré comme une variable numérique qui représente l'intensité lumineuse.

Il n'y a qu'un hyperparamètre sélectionné pour cette méthode. Celui-ci est repris dans le Tableau 6.

Hyperparamètres	Valeur(s) testée(s)	Description
Ajustement de la variance (var_smoothing)	$10^{-9}, 10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^0$	Ajustement de la variance pour éviter les problèmes de variances très petites ou nulles

Tableau 7 : Récapitulatif des hyperparamètres pour Bayes Naïf

Résultats obtenus

Variance	Erreur d'entraînement	Erreur de test	Précision
10^{-9}	0.7604	0.7742	0.2258
10^{-8}	0.7563	0.7730	0.2270
10^{-7}	0.7529	0.7690	0.2310
10^{-6}	0.7486	0.7642	0.2358
10^{-5}	0.7434	0.7592	0.2408
10^{-4}	0.7364	0.7538	0.2462
10^{-3}	0.7294	0.7472	0.2528
10^{-2}	0.7251	0.7435	0.2565
10^{-1}	0.7393	0.7587	0.2413
10^0	0.8149	0.8322	0.1678

Tableau 8 : Bayes Naïf - Comparaison des erreurs sur les données

Pour évaluer les performances de ce modèle, nous nous appuyons sur l'analyse de la précision, de l'erreur d'entraînement, de l'erreur de test et de la précision. Le Tableau 7 nous donne les valeurs chiffrées de ces différentes métriques de performance.

Ces résultats nous permettent d'observer qu'avec ce modèle nous obtenons une erreur d'entraînement et une erreur de test très élevées (bien plus qu'avec les précédentes méthodes). Nous constatons donc un sous-

apprentissage important, ce qui conduit logiquement à une mauvaise classification des données de test. De plus, la précision est très faible. Nous en concluons que ce modèle n'est pas approprié pour notre jeu de données.

En sortie du Grid Search, la variance est de 0.01. Ce modèle à priori optimisé présente une précision de 25.65%, ce qui prouve une nouvelle fois qu'il n'est pas approprié pour nos données.

Si on s'intéresse néanmoins au rapport de classification relatif à ce modèle (Figure 20), nous remarquons que c'est toujours la classe 1 qui est la mieux représentée avec un rappel de 92%. Concernant les classes moins bien représentées, il est difficile de se prononcer dans le cas de ce classifieur, puisque les précisions sont mauvaises pour la plupart des classes. Les courbes ROC (Figure 21) confirment largement ces résultats.

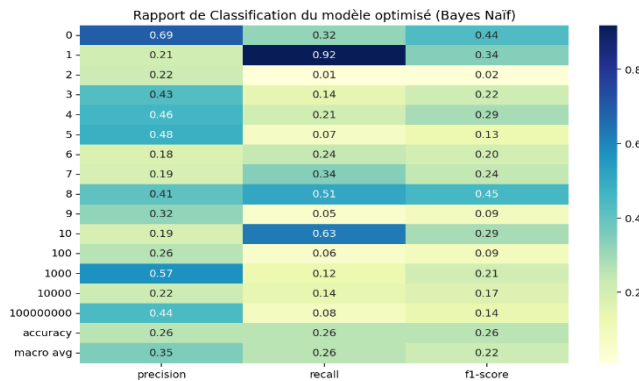


Figure 19 : Bayes Naïf – Rapport de classification

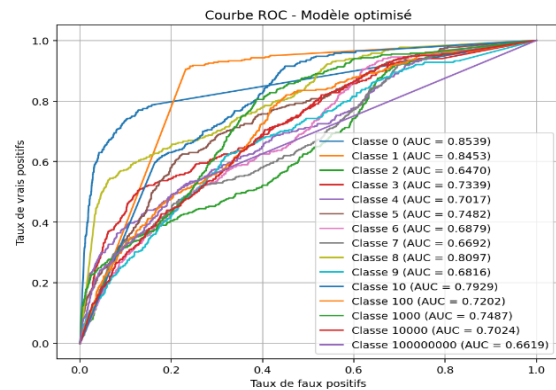


Figure 20 : Bayes Naïf – Courbe ROC

Analyse des classes problématiques

La classe 1 présente un rappel très élevé de 92% traduisant un faible taux de faux négatifs. Cela peut indiquer qu'elle est représentée d'une meilleure manière dans les données d'entraînement ou qu'elle est plus facilement différenciable que les autres. Cependant, beaucoup d'autres exemples sont placés dans la classe 1 à tort, ce qui lui confère une précision très faible de 21%. La classe 10 présente un rappel relativement supérieur aux autres classes mais la précision, et donc le F1-score, restent très faibles. Globalement, la plupart des classes sont mal identifiées.

Limites identifiées pour ce modèle

Dans notre situation, le modèle bayésien naïf n'est pas performant. Ces faiblesses peuvent être dues aux hypothèses qu'il impose. En effet, le jeu de données présentant des images de pixels ne respecte pas l'hypothèse d'indépendance conditionnelle des attributs dû à la corrélation des pixels voisins. Cela entraîne une mauvaise estimation des probabilités et donc des erreurs de classification.

Par ailleurs, le modèle bayésien naïf gaussien suppose que les données suivent une distribution normale, ce qui n'est pas forcément le cas dans le jeu de données considéré ici.

Ces deux hypothèses étant violées, la performance du modèle diminue de manière drastique.

2.2.5. Perceptron

Le Perceptron est un algorithme de classification supervisée de classifieurs linéaires binaires. Il est toutefois possible de l'utiliser pour une classification multiple en appliquant la méthode one-vs-all. Cette approche permet de transformer un problème de classification multi-classe en plusieurs problèmes de classification binaire. Le principe est de créer un classifieur pour chaque classe pour la distinguer de toutes les autres classes. Le perceptron calcule une somme pondérée des données d'entrée et, en fonction de cette somme, le modèle prédit la classification des données. Ensuite, il ajuste ses poids en cas d'erreur afin de mieux classer les données. Il est composé d'une seule couche de nœuds d'entrée connectée à une couche de nœuds de sortie.

Les hyperparamètres sélectionnés pour l'analyse de ce modèle sont donnés dans le Tableau 8.

Hyperparamètres	Valeur(s) testée(s)	Description
eta0	0,01 ; 0,1 ; 0,5 ; 1	Taux d'apprentissage initial utilisé pour la mise à jour des poids. Constante par laquelle les mise à jour sont multipliées
Nombre maximal d'itérations (max_iter)	50, 100, 200, 500, 1000	Limitation du nombre de passages sur l'ensemble des données pour entraîner le modèle.
penalty	None, l1, l2, elasticnet	Type de régularisation à appliquer.
alpha	0,0001 ; 0,001 ; 0,01	Ajustement du poids de la régularisation dans la fonction de coût.
shuffle	True, False	Mélange ou non des données avant chaque itération
fit_intercept	True, False	Ajout ou non d'un terme d'interception au modèle.

Tableau 9 : Récapitulatif des hyperparamètres du Perceptron

Résultats obtenus

eta0	Erreur d'entraînement	Erreur de test	Précision
0.01	0.5319	0.6267	0.3733
0.1	0.5332	0.6275	0.3725
0.5	0.5332	0.6275	0.3725
1	0.5332	0.6275	0.3725

Tableau 10 : Perceptron – Comparaison des erreurs sur les données

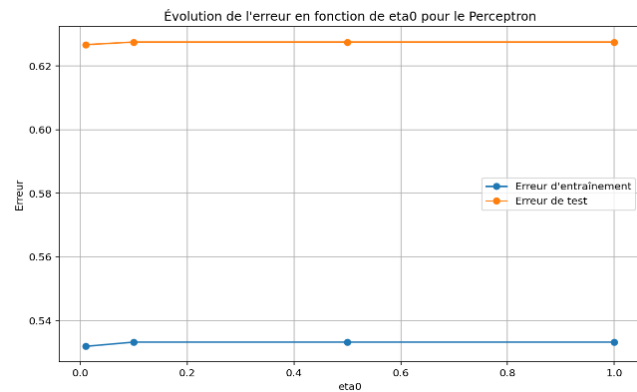


Figure 21 : Evolution de l'erreur en fonction du paramètre eta0

Les performances du modèle sont entre autres évaluées à partir de la précision et des erreurs sur les données d'entraînement et de test en fonction du paramètre eta0, le taux d'apprentissage (Tableau 9, Figure 23).

Les résultats montrent que le paramètre eta0 n'influence pratiquement pas les erreurs d'entraînement et de test. Il n'y a que le modèle utilisant la valeur de paramètre eta0 = 0.01 qui présente une erreur légèrement inférieure aux autres. L'erreur d'entraînement étant également élevée, nous concluons que ce modèle n'apporte pas de bons résultats dans la résolution de ce problème de classification.

A l'issue du Grid Search, nous obtenons les meilleurs paramètres suivants : {'alpha': 0.0001, 'eta0': 0.5, 'fit_intercept': True, 'max_iter': 50, 'penalty': 'elasticnet', 'shuffle': True}. La précision du modèle optimisé est faible (33,95%), confirmant sa difficulté à classer correctement nos données.

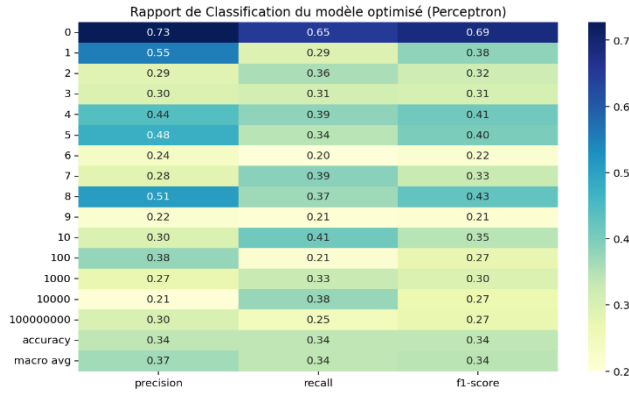


Figure 22: Perceptron – Rapport de classification

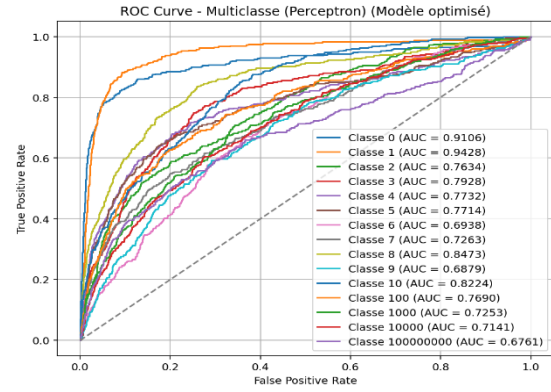


Figure 23 : Perceptron – Courbe ROC

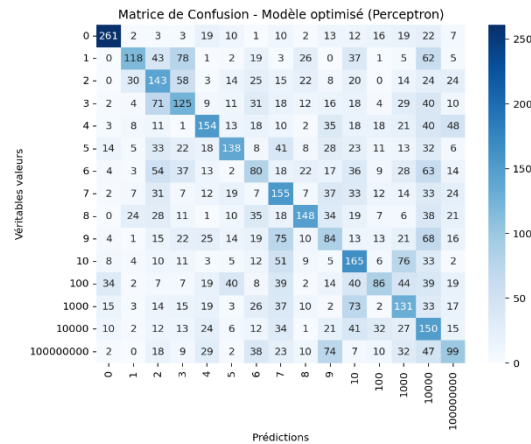


Figure 24 : Perceptron – Matrice de confusion

Le rapport de classification relatif au modèle optimisé (Figure 24) montre que, après optimisation des paramètres, les différentes métriques de performance du modèle sont très faibles. Les précision, rappel et F1-score moyens sont très bas. La seule classe présentant des performances acceptables est la classes 0, avec un F1-score de 69%. Toutes les autres possèdent au moins une métrique inférieure à 50%, démontrant la difficulté du modèle à généraliser. Les courbes ROC (Figure 25) ne présentent pas non plus des résultats satisfaisants : les aires sous-tendues aux courbes sont faibles. La matrice de confusion (Figure 26) montre qu'il y a anormalement beaucoup de faux positifs pour la majorité des classes.

Analyse des classes

Les analyses précédentes montrent que la qualité de classification de la classe 0 est bien supérieure à celle des autres classes mais reste, néanmoins, assez faible. Les autres classes sont globalement toutes très mal identifiées.

Limites identifiées pour ce modèle

Le Perceptron est un modèle assez simpliste. En effet, ce dernier n'effectue qu'une somme pondérée des entrées, suivie d'une fonction seuil. Ceci pose des difficultés dans l'analyse de structures plus complexes.

En outre, il n'est pas capable de classer les données qui ne sont pas linéairement séparables, ce qui limite fortement ses performances pour notre jeu de données.

2.2.6. Multilayer perceptron

Le Multilayer Perceptron (MLP) est un réseau neuronal feedforward composé de neurones entièrement connectés. Il est utilisé pour des tâches complexes lorsque les données ne sont pas linéairement séparables. Il est constitué de trois types de couches : la couche d'entrée qui prend des données en entrée, les couches cachées qui permettent la propagation des informations entre les couches et la couche de sortie qui permet de créer les résultats finaux. La propagation des données est permise grâce à des connexions entre les neurones comportant des poids ajustables en fonction de l'importance des relations. Ces connexions sont reliées par des relations traduites par des fonctions d'activation qui permettent au modèle de traiter les jeux de données non-linéairement séparable. Le MLP possède un système de rétropropagation afin d'apprendre les schémas dans les données.

Les hyperparamètres choisis pour l'analyse de ce modèle sont présentés dans le Tableau 10.

Hyperparamètres	Valeur(s) testée(s)	Description
hidden_layer_sizes	(128,),(256,),(512,),(256,128),(512,256),(300,150,75),(512,256,128),(1024,512,256)	Structure des couches cachées du réseau neuronal, en spécifiant le nombre de neurones par couche.
activation	relu, tanh	Fonction d'activation utilisée dans les neurones.
solver	adam, sgd	Algorithme d'optimisation pour ajuster les poids.
alpha	0,0001 ; 0,001 ; 0,01 ; 0,1	Paramètre de régularisation qui contrôle la pénalisation des poids pour éviter le surapprentissage.
learning_rate	constant, adaptative	Stratégie de mise à jour du taux d'apprentissage

Tableau 11 : Récapitulatif des hyperparamètres du Multilayer Perceptron

Résultats obtenus

Taille des couches cachées	Erreur d'entraînement	Erreur de test	Précision
(128,)	0.0000	0.1947	0.8053
(256,)	0.0000	0.1927	0.8073
(512,)	0.0000	0.1892	0.8108
(256,128)	0.0000	0.1790	0.8210
(512,256)	0.0000	0.1618	0.8382
(300,150,75)	0.0000	0.1765	0.8235
(512,256,128)	0.0020	0.1595	0.8405
(1024,512,256)	0.0232	0.1845	0.8155

Tableau 12 : MLP – Comparaison des erreurs sur les données

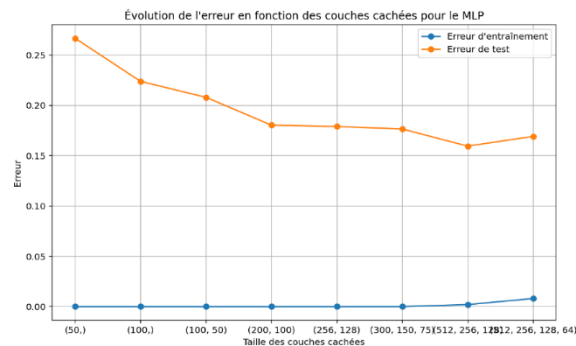


Figure 25 : Evolution de l'erreur en fonction du nombre de couches cachées

L'évaluation des performances du modèle en fonction du nombre de couche cachées est donnée dans le Tableau 1. L'évolution des erreurs (Figure 27) montre que le modèle souffre de surapprentissage car nous observons que l'erreur sur les données d'entraînement reste constante à 0. Néanmoins, nous constatons que l'erreur de test est assez faible et diminue lorsque les nombres de couches cachées et de neurones augmentent. Ce modèle semble d'un premier abord bien fonctionner pour notre jeu de données.

A l'issue du Grid Search, nous obtenons les meilleurs paramètres suivants : {'activation': 'relu', 'alpha': 0.1, 'hidden_layer_sizes': (512, 256, 128), 'learning_rate': 'constant', 'solver': 'adam'}. La précision relative au modèle optimisé atteint les 83,87%, précision la plus élevée par rapport aux modèles précédemment testés.

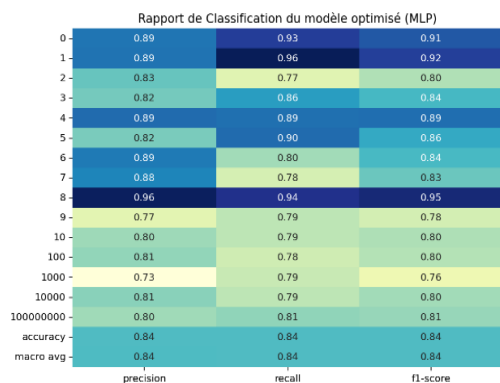


Figure 26 : MLP – Rapport de classification

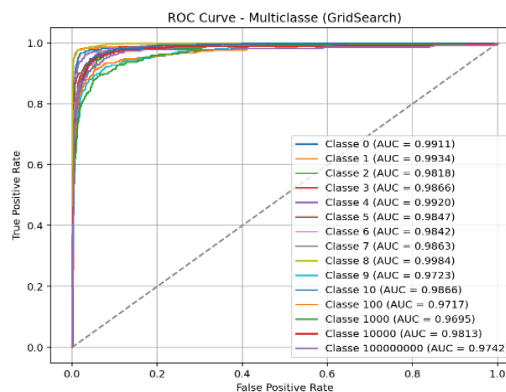


Figure 27 : MLP – Courbe ROC

Le rapport de classification relatif au modèle optimisé (Figure 28) montre que les classes possèdent de bonnes métriques de performance. Les valeurs moyennes de précision, de rappel et de F1-score sont toutes à 84%. Ici encore, on observe que les classes les mieux représentées sont les classes 0, 1 et 8 avec des précisions atteignant les 96%. La plus mauvaise précision est de 77% pour la classe 2, ce qui nous semble plutôt acceptable en comparaison avec les autres méthodes. Les courbes ROC associées (Figure 29) renforcent nos constations : les courbes sont toutes proches du coin supérieur gauche.

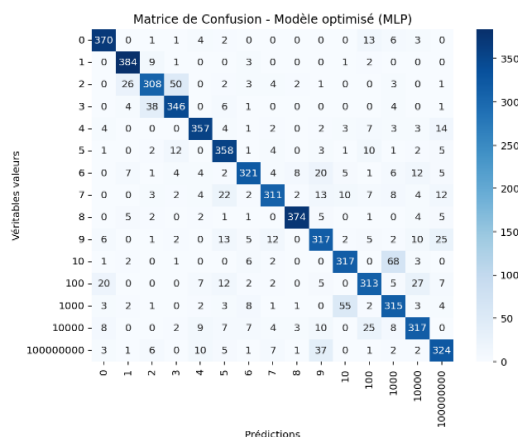


Figure 28 : MLP – Matrice de confusion

La matrice de confusion (Figure 30) confirme ces résultats. Nous remarquons en effet que la majorité des prédictions sont exactes : il n'y a que très peu de faux positifs.

Analyse des classes

Globalement, la performance du classifieur est très satisfaisante : il n'y a aucune classe qui possède de problème majeur. Nous pouvons néanmoins observer que la classe 1000 est celle présentant les métriques les plus faibles, environnant les 75%.

Limites identifiées pour ce modèle

Le MLP est un très bon choix pour la classification de notre jeu de données. Toutefois, il est encore possible d'optimiser ce modèle en testant un nombre plus conséquent de paramètres. Cependant, cette optimisation demande de grandes ressources et beaucoup de temps d'exécution. La configuration de notre ordinateur est donc une limite à l'optimisation de notre modèle.

2.2.7. Régression logistique

La régression logistique est un classifieur basé sur un modèle statistique linéaire permettant de modéliser des relations entre les caractéristiques d'entrée et la probabilité d'appartenance à une classe. Ce modèle utilise une fonction sigmoïde comme fonction de lien et fonctionne en ajustant des poids grâce à une méthode d'optimisation afin de déterminer l'importance de chaque caractéristique dans la prédiction de la classe.

Les hyperparamètres choisis pour l'analyse de ce modèle sont présentés dans le Tableau 12.

Hyperparamètres	Valeur(s) testée(s)	Description
C	0.01 ; 0.01 ; 0.1 ; 1 ; 10 ; 100	Paramètre de régularisation qui contrôle l'importance de la pénalisation dans le modèle.
Penalty	l1, l2, elasticnet	Type de régularisation

Tableau 13 : Récapitulatif des hyperparamètres pour la régression logistique

Résultats obtenus

L'évolution de l'erreur en fonction du paramètre C est donnée à la Figure 31. L'erreur d'entraînement diminue fortement à mesure que C augmente avant de se stabiliser autour d'une erreur de 32,5%, ce qui signifie qu'un C plus élevé permet au modèle de mieux s'ajuster sur les données d'entraînement. L'erreur de test augmente quant à elle avec C, avant de se stabiliser autour de 52,5%. Elle ne suit pas la diminution observée sur les données d'entraînement. Cela indique que même si le modèle s'ajuste très bien aux données d'entraînement avec un C élevé, il généralise mal sur les données de test.

Le modèle souffre de surapprentissage pour des valeurs de C élevées, où l'erreur d'entraînement est très faible mais l'erreur de test ne diminue pas. Le choix d'un C modéré nous semble préférable pour éviter un surapprentissage et obtenir un meilleur compromis entre les deux erreurs.

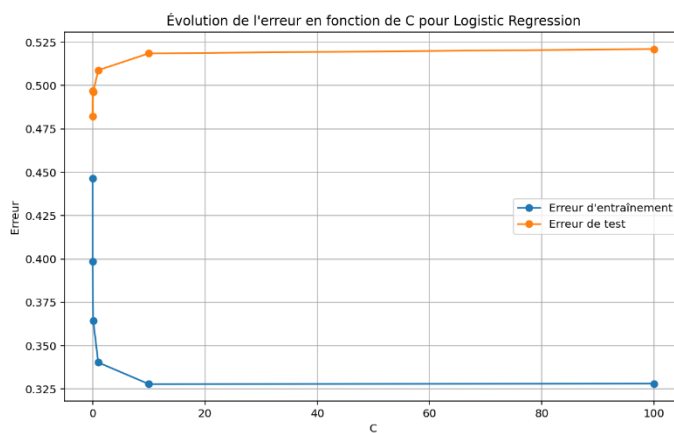


Figure 29 : Evolution de l'erreur en fonction du paramètre C

En sortie du Grid Search, nous obtenons comme valeur de C optimale 0,01 et comme valeur de penalty « l2 » avec une précision assez faible de 51,77%. Cette performance du modèle optimisé nous semble assez mauvaise, nous nous intéressons donc au rapport de classification (Figure 32). Ce dernier montre en effet que la majorité des classes présentent des métriques assez faibles. Encore et toujours, seules les classes 0 et 1 semblent être bien représentées. Les courbes ROC (Figure 33) confirment ces observations, avec des courbes assez éloignées du coin supérieur gauche, hormis pour les classes 0 et 1.

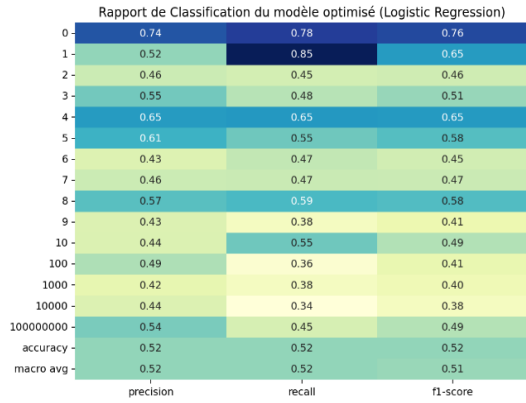


Figure 30 : Régression logistique – Rapport de classification

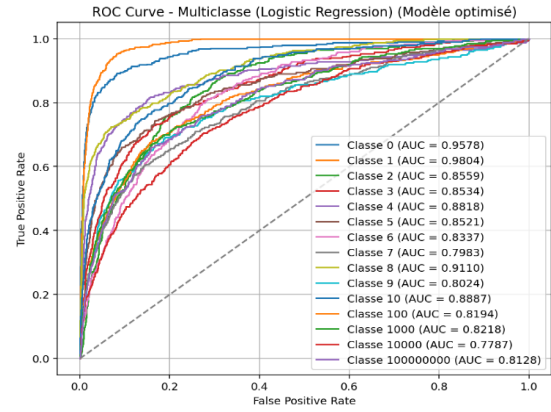


Figure 31 : Régression logistique – Courbe ROC

La matrice de confusion relative au modèle optimisé (Figure 34) confirme également ces analyses, ce qui nous permet de conclure que ce modèle ne présente pas des performances satisfaisantes pour notre jeu de données.

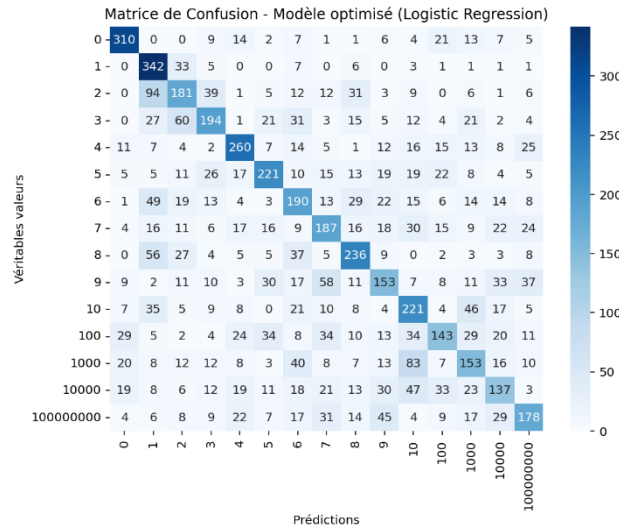


Figure 32 : Régression logistique – Matrice de confusion

Limites identifiées pour ce modèle

Les images MNIST sont des chiffres manuscrits de taille 28x28 pixels, souvent aplaties. La régression logistique traite chaque pixel indépendamment, ce qui fait perdre les relations spatiales importantes présentes dans les images. Cela empêche le modèle de détecter efficacement les chiffres manuscrits de notre jeu de données.

En outre, la régression logistique est un modèle linéaire qui cherche des frontières de décision simples entre les classes. Or, les chiffres manuscrits sont complexes et non linéaires. Une frontière linéaire n'est donc pas suffisante pour bien distinguer les différentes classes.

Par ailleurs, la régression logistique est sensible aux données à haute dimension, ce qui diminue là encore ses performances.

2.3. Comparaison des résultats

Nous comparons les performances des classifieurs en concevant un tableau récapitulatif (Tableau 14) pour déterminer ceux qui donnent les meilleurs résultats.

Critère	K-NN	Random Forest	SVM	Bayes Naïf	Perceptron	Régression Logistique	Multilayer Perceptron
Précision	67.98	0.7693	0.7840	0.2565	0.3395	0.5177	0.8387
Précision (Validation croisée)	0.6653	0.7628	0.7653	0.2531	0.3691	0.5138	0.8304
Rappel	0.6800	0.7700	0.7800	0.2600	0.3400	0.5200	0.8400
F1-score	0.6800	0.7700	0.7800	0.2200	0.3400	0.5100	0.8400
AUC-ROC	0.9360	0.9739	0.9594	0.7336	0.7744	0.8565	0.9836
Temps	1.85 s	39.59 s	225.54 s	2.66 s	10.43 s	649.18 s	85.81 s
Complexité	$O(nd)$	$O(mn\log(n))$	$O(n^2d)$	$O(nd)$	$O(nd)$	$O(nd)$	$O(ndh)$

Tableau 14 : Comparaison des différentes méthodes

Pour notre jeu de données, nous avons observé des différences notables entre les modèles en termes de précision, rappel et AUC-ROC. Le Multilayer Perceptron (MLP) et le SVM ont montré des performances exceptionnelles, atteignant respectivement les meilleures précisions de 83.87% et 78.40%. Ces modèles ont réussi à capturer efficacement les relations complexes entre les données, ce qui les rend particulièrement adaptés à notre tâche de classification. En revanche, des modèles plus simples comme le Bayes Naïf et le Perceptron ont obtenu des résultats bien plus faibles, avec des précisions de 25.65% et 33.95%, ce qui indique qu'ils peinent à bien généraliser sur notre jeu de données.

Concernant le temps d'exécution, le K-nn s'avère être le plus rapide, avec un temps de calcul de seulement 1.85 secondes. Cela pourrait être avantageux dans des contextes où nous aurions besoin de prédictions rapides, d'autant plus que sa précision n'est pas des plus mauvaises (67.98%). Les modèles plus complexes comme le SVM et le MLP prennent considérablement plus de temps, avec des durées respectives de 225.54 secondes et 649.18 secondes. Bien que ces modèles offrent une meilleure précision, leur temps d'exécution élevé pourrait constituer un inconvénient. Le Random Forest, avec un temps d'exécution de 39.59 secondes, offre un bon compromis entre rapidité et précision.

La complexité algorithmique des modèles varie également et a une influence directe sur leur efficacité. Le K-nn, avec une complexité de $O(nd)$ ⁴, reste rapide mais devient coûteux en temps pour de grandes quantités de données. Les modèles comme le SVM, avec une complexité de $O(n^2d)$, et le MLP, avec $O(ndh)$ ⁵, sont beaucoup plus gourmands en ressources, ce qui les rend moins adaptés à un grand volume de données comme le nôtre. Le Random Forest, avec une complexité de $O(mn\log(n))$ ⁶, se situe entre ces deux extrêmes, et offre un équilibre entre performance et coût de calculs.

⁴ n est le nombre d'échantillons dans l'ensemble d'entraînement et d est la dimension des données.

⁵ h est le nombre de neurones dans les couches cachées.

⁶ m est le nombre d'arbres.

3. Amélioration des Modèles

3.1. Réduction de la dimensionnalité

L'Analyse en Composantes Principales est une technique très utile pour réduire la dimensionnalité des données tout en préservant l'essentiel de l'information. Dans le cadre de notre projet, l'application de la PCA nous permet de simplifier les données d'images en extrayant les principales directions. Cela réduit le nombre de caractéristiques et rend le processus d'entraînement de nos modèles plus rapide et plus efficace. De plus, cette réduction de dimensions nous aide à éliminer le bruit et à éviter le surajustement, tout en maintenant la capacité du modèle à bien généraliser sur de nouvelles données.

Cette approche est particulièrement adaptée à notre jeu de données Chinese MNIST. Les images de ce jeu de données, bien qu'elles soient relativement simples, possèdent en effet une grande quantité de pixels, ce qui peut compliquer l'entraînement d'un modèle. En appliquant PCA, nous pouvons réduire considérablement le nombre de dimensions tout en conservant l'essentiel des informations visuelles. Cela permet non seulement de simplifier la tâche de classification, mais aussi de rendre le modèle plus performant, surtout lorsque les données sont augmentées.

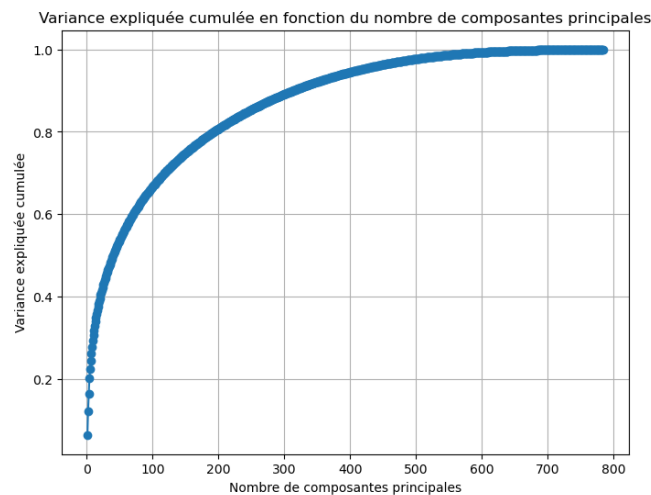


Figure 33 : Variance expliquée cumulée

Dans notre cas, nous avons choisi un nombre de composantes nous permettant de garder 95% de la variance expliquée, ce qui nous donne 413 composantes principales. En effet, au-delà de cette valeur, on peut voir sur la Figure 33 que prendre des composantes principales supplémentaires ne capture pas beaucoup plus d'informations.

3.2. Augmentation des données

L'augmentation de données est une technique utilisée pour améliorer la performance des modèles d'apprentissage automatique, en particulier lorsque le jeu de données est limité. Elle consiste à appliquer diverses transformations sur les images afin de créer de nouvelles images. Ces transformations peuvent inclure des opérations géométriques (rotation, translation), des modifications d'intensité (ajout de bruit ou de flou), et d'autres ajustements (épaississement ou amincissement des lignes). L'objectif est d'augmenter la diversité des données, permettant au modèle de mieux généraliser et de devenir plus robuste face aux variations naturelles dans les images. Jusqu'à présent, les modèles ne sont que peu robustes : pour la plupart, on recense beaucoup de faux positifs pour des classes similaires pourtant bien différentes.

Dans le cadre de ce projet, nous sélectionnons les transformations qui nous semblent les plus adaptées à notre jeu de données :

- Rotation (d'environ 15°) : les chiffres écrits à la main peuvent parfois être légèrement inclinés ;
- Translation (de ± 5 pixels) : léger décalage dans les chiffres écrits à la main ;
- Zoom (Zoom in jusqu'à 0.8 – Zoom out jusqu'à 1.2) : variation dans la taille des chiffres écrits à la main ;
- Distorsion affine (± 0.2) : déformations dues à l'écriture rapide des chiffres ;
- Epaississement/Amincissement des traits ;

Ces techniques augmentent significativement le nombre d'images et permettent au modèle de s'adapter à une plus grande variété de situations. En multipliant le nombre d'images initial par les transformations, on offre au modèle plus d'exemples pour apprendre, réduisant ainsi le risque de surapprentissage et améliorant sa capacité à faire des prédictions sur de nouvelles images. Cette méthode permet également de rendre le modèle plus robuste face aux variations d'écriture, de luminosité, et aux déformations des chiffres écrits à la main. Ici, le nombre total d'images après augmentation s'élève à 165000, contre 15000 au départ. Nous avons séparé en données d'entraînement (151 800) et données de test (13 200).

Méthodes	Précision sur les données initiales	Précision sur les données augmentées
K-nn	0.6798	0.72379
Random Forest	0.7693	0.78439
SVM	0.7840	0.7675
Naïf Bayes	0.2565	0.1178
Perceptron	0.3395	0.18917
Multilayer Perceptron	0.8387	0.90091
Régression Logistique	0.5177	0.37273

Tableau 15 : Comparaison de la précision sur les données initiales et les données augmentées

L'augmentation des données a un effet positif sur certains modèles, mais pas sur tous (Tableau 15). Le Multilayer Perceptron (MLP), qui était déjà performant sur les données initiales, connaît une amélioration notable de sa précision, atteignant 90.09% contre 83.87% sur les données initiales. Le Random Forest également bénéficie de l'augmentation des données, avec une légère amélioration de sa précision, passant de 76.93% à 78.44%. Cependant, le SVM voit sa précision diminuer légèrement, passant de 78.40% à 76.75%, ce qui suggère que l'augmentation des données ne s'est pas traduite par une meilleure généralisation dans ce cas particulier. Les performances du Naïf Bayes et du Perceptron se détériorent davantage, avec des précisions respectives de 11.78% et 18.92% sur les données augmentées, ce qui indique que ces modèles sont sensibles à la variation des données et peuvent souffrir d'un surapprentissage.

3.3. Ensembles

Dans le cadre de ce projet, nous avons choisi d'exploiter deux méthodes d'ensembles avancées pour augmenter davantage les performances du classifieur concerné : le Voting et le Stacking. Ces approches nous permettent de combiner les modèles initiaux, de réduire les erreurs de prédiction et ainsi d'améliorer la capacité de généralisation sur des données de test.

3.3.1. Le Voting

Le Voting Classifier est une méthode qui combine les prédictions de plusieurs modèles pour améliorer la performance globale. Il existe deux types principaux de Voting :

- Hard Voting : La classe prédite est celle qui obtient le plus de votes parmi les modèles ;
- Soft Voting : La classe prédite est celle avec la probabilité la plus élevée, basée sur les prédictions probabilistes des modèles.

Cette approche nous permet de tirer parti des forces de chaque modèle, réduisant ainsi le risque de surapprentissage et améliorant la généralisation.

Dans notre cas, nous combinons les modèles qui ont donné les meilleurs résultats, à savoir K-nn, SVM, Random Forest et MLP, en utilisant un voting soft. Ce dernier permet de combiner les probabilités de chaque modèle plutôt que de simplement faire un vote majoritaire, ce qui améliore la précision en tenant compte de la certitude des prédictions. Les résultats sont les suivants :

- En combinant les 4 modèles, la précision est de 93,14% ;
- En combinant uniquement Random Forest et MLP, la précision est de 93,92%.

Nous avons essayé l'ensemble des combinaisons possibles, et il s'avère que la combinaison de Random Forest et de MLP a donné la meilleure précision, soit de 93,92%.

3.3.2. Le Stacking

Le Stacking est une méthode plus sophistiquée qui combine les performances de plusieurs modèles en utilisant un « métamodèle » pour combiner les prédictions des modèles de base (dans notre cas, il s'agit de Régression Logistique). Contrairement au Voting, qui effectue une combinaison simple, le Stacking permet au métamodèle d'apprendre la meilleure manière de combiner les prédictions des différents classifieurs, améliorant ainsi la performance globale. Par exemple, si un modèle excelle dans la détection de certaines classes mais échoue sur d'autres, un autre modèle peut compenser ces faiblesses.

- En combinant les 4 modèles les plus performants, la précision est de 94,54% ;
- En combinant MLP et Random Forest, la précision est de 94,07% ;

3.3.3. Comparaison

Nous remarquons que le Stacking est plus performant que le Voting. Ce résultat était attendu puisque le Stacking utilise un métamodèle qui permet de pondérer et de compenser les faiblesses des modèles de base, améliorant ainsi la performance globale. En outre, le Stacking gère mieux les erreurs des différents modèles. Ainsi, le Stacking bénéficie d'une combinaison plus flexible des modèles, ce qui lui permet d'exploiter la complémentarité des classifieurs pour une meilleure généralisation sur les données de test.

3.4. Transfer Learning

Jusqu'à présent, nous avons réussi à obtenir une précision de 94,54 % en utilisant des méthodes traditionnelles comme k-NN, Random Forest, SVM et MLP, combinées dans une approche de Stacking. Pour essayer d'améliorer davantage la performance, nous avons testé le Transfer Learning avec des CNN pré-entraînés, comme ResNet. Ces modèles sont très puissants, car ils ont été entraînés sur des millions d'images pour apprendre des caractéristiques complexes. Contrairement à k-NN, SVM ou Random Forest, les CNN sont spécialement conçus pour analyser les relations spatiales dans les images grâce aux convolutions, ce qui pourrait être un avantage pour notre jeu de données.

Cependant, les résultats ne sont pas à la hauteur des attentes : nous n'avons pas dépassé une précision de 73 %, quelle que soit la répartition des données d'entraînement et de test, avec une perte qui reste élevée (2678.2653). Après recherche, nous pensons que cela vient principalement de l'écart entre le modèle et le jeu de données. En effet, des modèles comme ResNet sont faits pour des images complexes et de haute résolution (comme celles d'ImageNet), alors que notre jeu de données Chinese MNIST contient des images beaucoup plus simples, en niveaux de gris et en basse résolution (28x28). Les caractéristiques apprises par ResNet, comme les textures et formes complexes, ne correspondent pas bien aux chiffres manuscrits, ce qui limite leur efficacité.

En outre, les CNN pré-entraînés sont souvent sur-paramétrés, avec des millions de paramètres. Ces modèles sont donc trop grands pour notre jeu de données. Même avec des données augmentées, les chiffres manuscrits n'offrent pas assez de variété pour éviter le sur-apprentissage. À l'inverse, l'approche Stacking, en combinant plusieurs modèles comme k-NN, Random Forest, SVM et MLP, est mieux adaptée aux

spécificités simples de notre jeu de données et parvient à exploiter efficacement les relations des chiffres, ce qui explique pourquoi elle donne de bien meilleurs résultats.

3.5. Suppression classes mal représentées

Au fil de cette analyse et des différents essais que nous avons effectués, nous avons remarqué que certaines classes diminuaient les performances des différents modèles, en particulier les classes 9 et 1000. Nous souhaitons donc effectuer l'analyse en supprimant les classes problématiques, pour vérifier les performances de notre classifieur final.

3.5.1. Suppression de la classe 1000

Le Tableau 16 permet de comparer les résultats obtenus sur les données augmentées avant et après la suppression de la classe 1000.

Méthodes	Précision sur les données augmentées avant suppression	Précision sur les données augmentées après suppression
K-nn	0.72379	0.7472
Random Forest	0.78439	0.8068
SVM	0.7675	0.7803
Naïf Bayes	0.1178	0.1229
Perceptron	0.18917	0.1997
Multilayer Perceptron	0.90091	0.9071
Régression Logistique	0.37273	0.3937

Tableau 16 : Comparaison des précisions sur les données augmentées avant et après suppression de la classe 1000

En observant les résultats, nous remarquons que la suppression d'une classe a globalement conduit à une amélioration de la précision pour la plupart des modèles. Le Multilayer Perceptron (MLP), qui était déjà le meilleur modèle avec une précision de 90.09% avant suppression, a une légère augmentation, atteignant 90.71% après suppression. De même, le Random Forest a montré une amélioration de 78.44% à 80.68%, tandis que K-NN est passé de 72.38% à 74.72%. Cela suggère que la suppression de la classe a permis à ces modèles de mieux se concentrer sur les classes restantes, améliorant ainsi leur capacité de classification.

Nous avons ensuite effectué des méthodes d'ensemble sur les données augmentées ne contenant pas la classe 1000, mais les résultats ne sont pas à la hauteur de nos attentes. Nous obtenons en effet une précision de seulement 90.51% avec un temps de calcul très long pour le Voting – ce qui est inférieur à ce qui a été obtenu précédemment. Pour le Stacking, la précision est toujours légèrement supérieure au Voting (91.38%) mais toujours inférieure à ce qui a été obtenu précédemment. L'un des facteurs pouvant expliquer cette baisse de performance est que ces méthodes d'ensemble nécessitent une gestion plus complexe des modèles concernés, ce qui peut augmenter le risque de surapprentissage ou de sous-apprentissage, surtout lorsqu'il s'agit de données réduites après suppression.

4. Conclusion

4.1. Résumé des résultats

Dans le cadre de notre projet, nous avons d'abord exploré différentes méthodes de classification pour notre jeu de données Chinese MNIST, en nous concentrant sur des modèles de Machine Learning classiques tels que K-nn, Random Forest, SVM, Bayes Naïf, Perceptron, Multilayer Perceptron (MLP), et Régression Logistique. Nous avons évalué la performance de ces modèles sur les données initiales (**étape 1**) en appliquant Grid Search pour obtenir les meilleurs paramètres, puis nous avons appliqué des techniques d'augmentation des données pour augmenter les performances (**étape 2**). Les résultats ont montré que Random Forest, SVM et MLP offraient les meilleures performances.

Nous avons ensuite appliqué des méthodes d'ensemble (**étape 3**) telles que le Voting et le Stacking pour combiner les performances des différents modèles, ce qui nous a permis d'obtenir une précision élevée de 94.54%.

Finalement, nous avons essayé de retirer une classe (**étape 4**) qui nous semblait problématique (souvent confondue avec la classe 1, avec beaucoup de faux positifs) mais les résultats n'ont pas été aussi concluants que lorsqu'elle était incluse dans notre jeu de données initial (91%).

Nous sélectionnons donc comme modèle final le MLP, puisqu'il s'agit du classifieur qui a apporté individuellement la précision la plus élevée (avec plus de 90% sur les données augmentées). Nous suggérons néanmoins de combiner ce modèle à Random Forest pour faire monter la précision jusqu'à 94.5%.

4.2. Perspectives d'amélioration

Des modèles plus complexes tels que les réseaux de neurones profonds ou les réseaux convolutifs (CNN) pourraient être envisagés pour améliorer davantage les performances, surtout pour des tâches liées à des données complexes comme les images.

Il serait également pertinent de tester la suppression d'autres classes de manière itérative afin d'identifier la combinaison optimale, ce qui n'a pas été possible par manque de temps.

En outre, une exploration plus approfondie des hyperparamètres de chaque modèle pourrait affiner les résultats, bien que cela nécessite un investissement de temps considérable.

Enfin, il serait idéal de tester tous les modèles sur différentes tailles de données d'entraînement et de test pour identifier la séparation optimale, ce qui a été partiellement effectué pour K-nn et Random Forest, mais pas pour tous les essais.

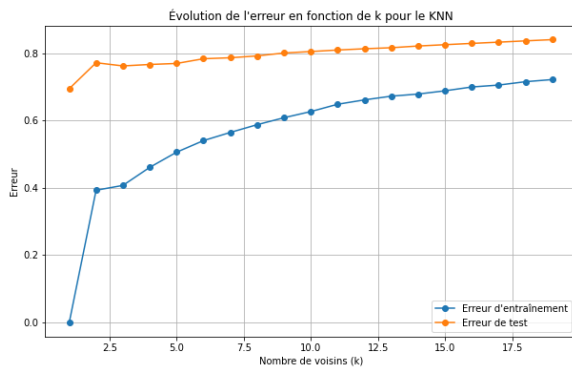
Annexes

Annexe 1 – résultats relatifs à KNN

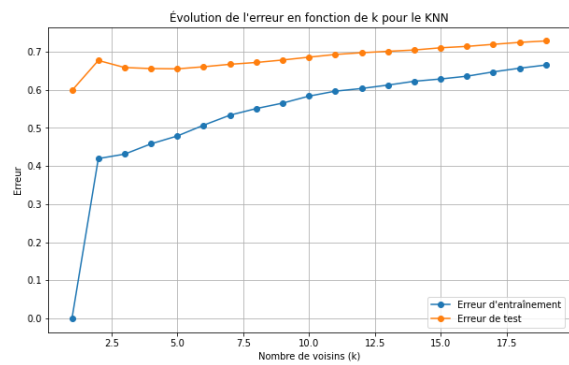
Remarque préliminaire

Il nous semble important de noter que ces résultats ont été obtenus lorsque les images étaient redimensionnées en 64x64 pixels. Les valeurs chiffrées varient donc par rapport à ce qui est indiqué dans le corps du rapport puisque, après analyse et réflexion, nous avons choisi de redimensionner les images en 28x28. Néanmoins, la réflexion en termes de choix est identique, raison pour laquelle nous laissons cette Annexe.

Résultats



(1) Graphique de l'évolution de l'erreur pour les 20 premières personnes sélectionnées pour KNN



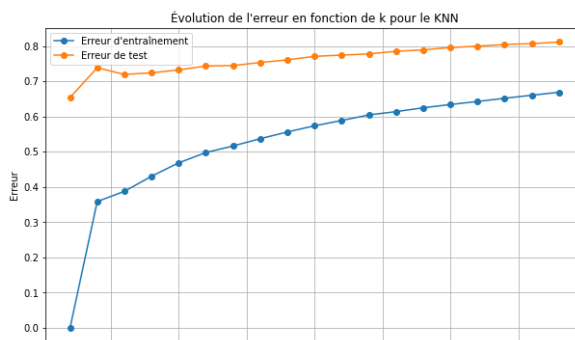
(2) Graphique de l'évolution de l'erreur quand l'ensemble d'entraînement contient 20% des données initiales pour KNN

Nous remarquons, pour les résultats repris en (1) et (2), que

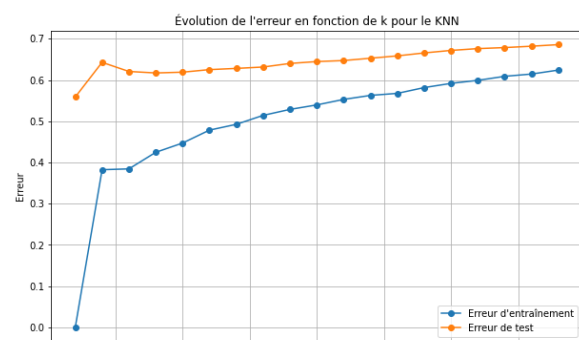
- Le meilleur k pour les données ordonnées : 1 avec une précision en validation croisée de 0.3058 ;
- Le meilleur k pour les données aléatoires : 1 avec une précision en validation croisée de 0.3940.

Ici, il y a un grand risque d'underfitting puisque la valeur idéale est de 1. Une précision de 0.5091 indique que le modèle fait mieux avec les données aléatoires que les données ordonnées, mais la performance reste encore relativement faible, étant proche du hasard pour un problème de classification avec plus de deux classes.

Comme nous suspectons de l'underfitting, nous essayons d'augmenter la taille de l'ensemble d'entraînement. Un ensemble d'entraînement plus grand nous permettra, on l'espère, de saisir plus efficacement les structures sous-jacentes dans les données.



(3) Graphique de l'évolution de l'erreur pour les 30 premières personnes sélectionnées pour KNN

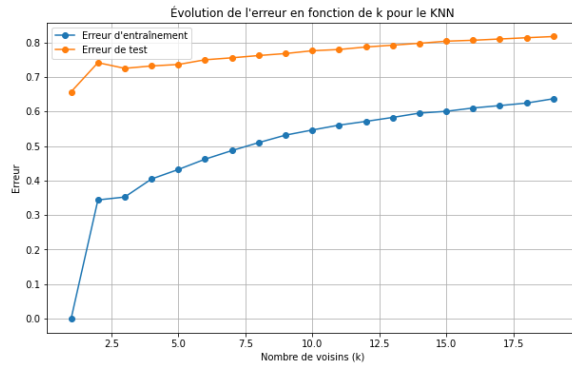


(4) Graphique de l'évolution de l'erreur quand l'ensemble d'entraînement contient 30% des données initiales pour KNN

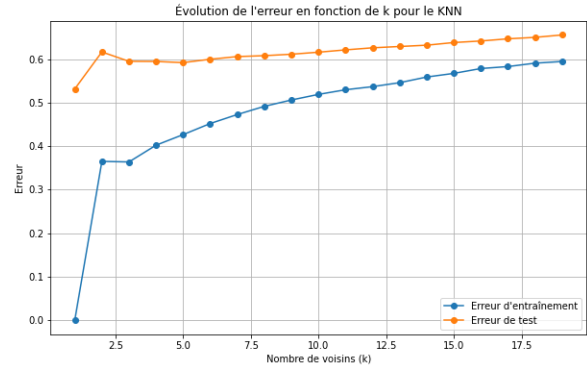
Nous remarquons, pour les résultats repris en (3) et (4), que

- Le meilleur k pour les données ordonnées : 1 avec une précision en validation croisée de 0.3558 ;
- Le meilleur k pour les données aléatoires : 1 avec une précision en validation croisée de 0.4224.

Les résultats s'améliorent mais semblent toujours très mauvais. En effet, le k idéal reste 1 – ce qui montre un risque de sous-apprentissage. La précision est en effet très faible sur l'ensemble des données d'entraînements et sur l'ensemble des données test.



(5) Graphique de l'évolution de l'erreur pour les 40 premières personnes sélectionnées pour KNN

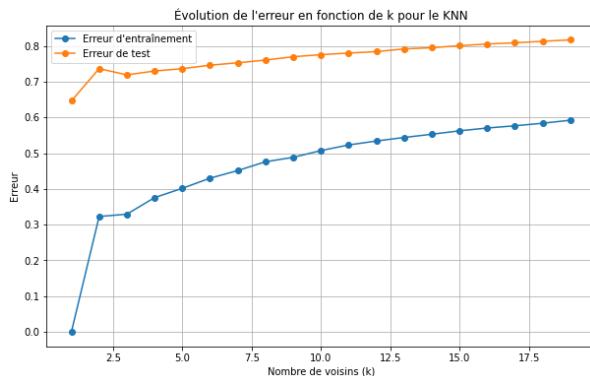


(6) Graphique de l'évolution de l'erreur quand l'ensemble d'entraînement contient 40% des données initiales pour KNN

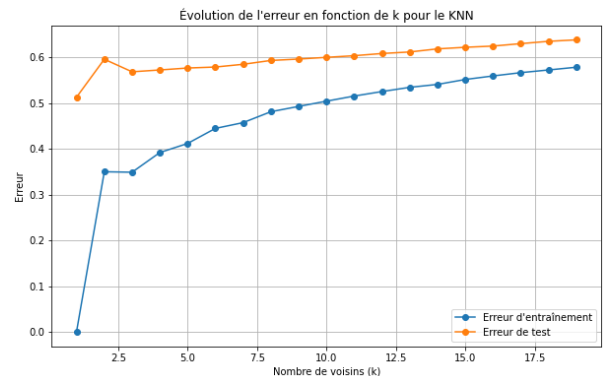
Nous remarquons, pour les résultats repris en (5) et (6), que

- Le meilleur k pour les données ordonnées : 1 avec une précision en validation croisée de 0.3888 ;
- Le meilleur k pour les données aléatoires : 1 avec une précision en validation croisée de 0.4390.

Les résultats s'améliorent mais semblent toujours très mauvais. En effet, le k idéal reste 1 – ce qui montre un risque de sous-apprentissage. La précision est toujours très faible sur l'ensemble des données d'entraînements et sur l'ensemble des données test.



(7) Graphique de l'évolution de l'erreur pour les 50 premières personnes sélectionnées pour KNN

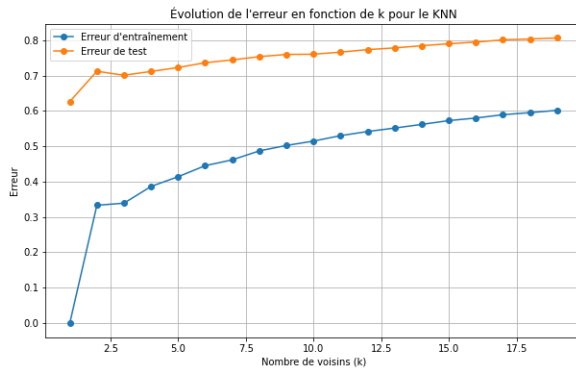


(8) Graphique de l'évolution de l'erreur quand l'ensemble d'entraînement contient 50% des données initiales pour KNN

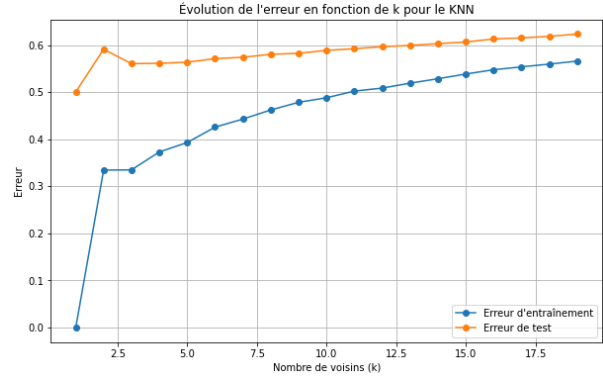
Nous remarquons, pour les résultats repris en (7) et (8), que

- Le meilleur k pour les données ordonnées : 1 avec une précision en validation croisée de 0.4217 ;
- Le meilleur k pour les données aléatoires : 1 avec une précision en validation croisée de 0.4619.

Le constat reste le même que précédemment, bien que les résultats s'améliorent.



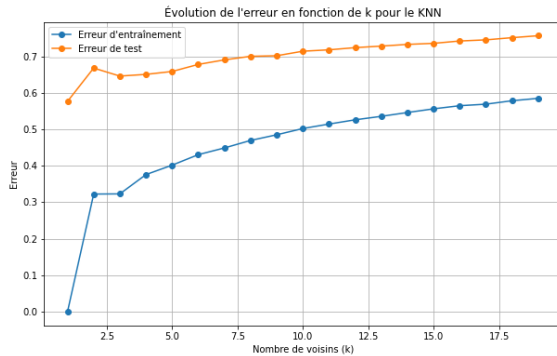
(9) Graphique de l'évolution de l'erreur pour les 60 premières personnes sélectionnées pour KNN



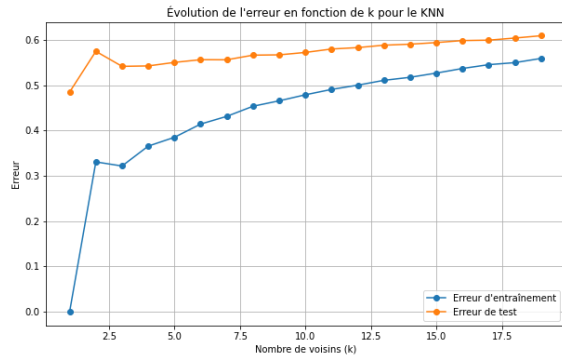
(10) Graphique de l'évolution de l'erreur quand l'ensemble d'entraînement contient 60% des données initiales pour KNN

Nous remarquons, pour les résultats repris en (9) et (10), que

- Le meilleur k pour les données ordonnées : 1 avec une précision en validation croisée de 0.4208 ;
- Le meilleur k pour les données aléatoires : 1 avec une précision en validation croisée de 0.4838.



(11) Graphique de l'évolution de l'erreur pour les 70 premières personnes sélectionnées pour KNN



(12) Graphique de l'évolution de l'erreur quand l'ensemble d'entraînement contient 70% des données initiales pour KNN

Nous remarquons, pour les résultats repris en (11) et (12), que

- Le meilleur k pour les données ordonnées : 1 avec une précision en validation croisée de 0.4369 ;
- Le meilleur k pour les données aléatoires : 1 avec une précision en validation croisée de 0.4922.

A l'aune de tous ces résultats, nous remarquons que la précision augmente à mesure que la dimension de l'ensemble d'entraînement augmente. Si l'on ne tient pas compte de la valeur 1 pour le paramètre k, on remarque que la meilleure valeur semble être de 3. En effet, c'est à ce moment-là que l'erreur a diminué le plus avant de réaugmenter sans cesse pour un nombre de voisins supérieur à 3. Cela signifie qu'au-delà de $k=3$, le modèle commence à surapprendre aux données d'entraînement (puisque l'erreur de test commence à réaugmenter à partir de cette valeur), ce qui réduit sa capacité à généraliser sur les données de test. En d'autres termes, avec $k = 3$, le modèle trouve un bon équilibre entre biais et variance, offrant ainsi une meilleure performance globale.

Il nous semble néanmoins curieux que la valeur idéale reste un nombre de voisins équivalent à 1.

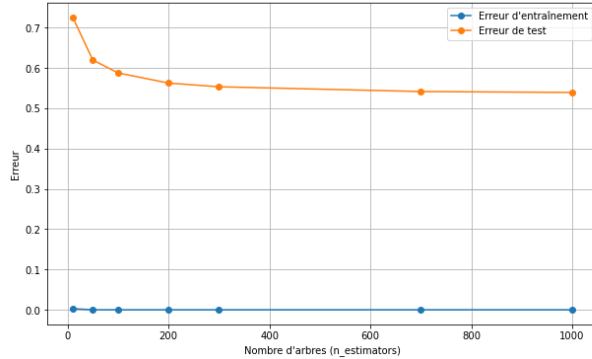
Annexe 2 – résultats relatifs à Random Forest

Remarque préliminaire identique à l'Annexe 1

Résultats

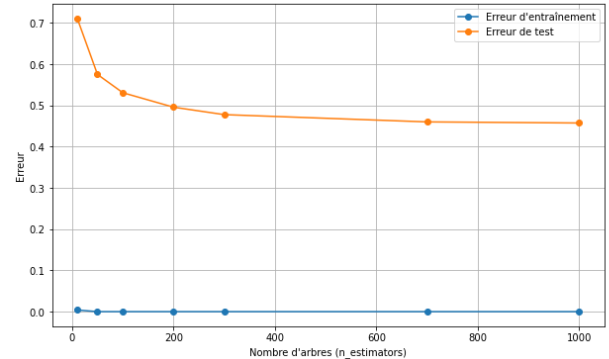
Dans un premier temps, nous testons l'algorithme de Random Forest sur différentes valeurs de l'ensemble d'entraînement et de l'ensemble test.

Erreur d'entraînement et de test en fonction du nombre d'arbres dans Random Forest (Données ordonnées)



(13) Graphique de l'évolution de l'erreur pour les 10 premières personnes sélectionnées pour Random Forest

Erreur d'entraînement et de test en fonction du nombre d'arbres dans Random Forest (Données aléatoires)

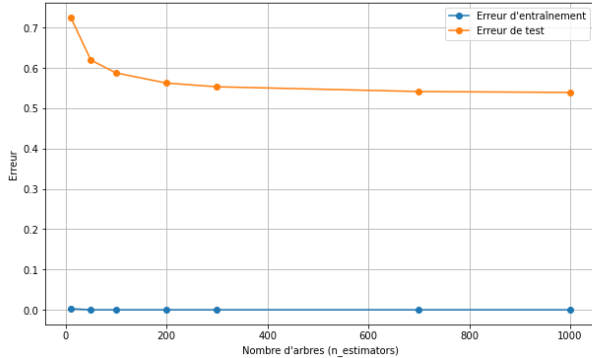


(14) Graphique de l'évolution de l'erreur quand l'ensemble d'entraînement contient 10% des données initiales pour Random Forest

Pour les résultats repris en (13) et (14), on retrouve :

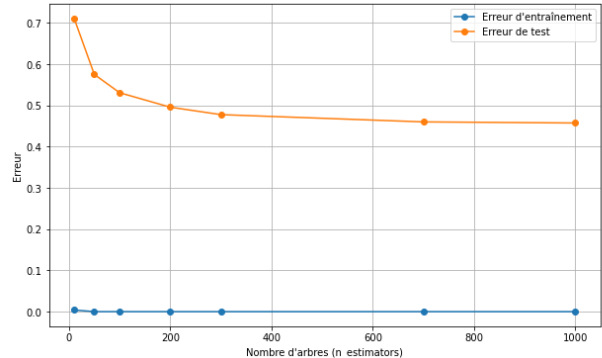
- Une meilleure précision en validation croisée pour 200 arbres de 0.4073 pour les données ordonnées ;
- Une meilleure précision en validation croisée pour 200 arbres de 0.4760 pour les données aléatoires.

Erreur d'entraînement et de test en fonction du nombre d'arbres dans Random Forest (Données ordonnées)



(15) Graphique de l'évolution de l'erreur pour les 30 premières personnes sélectionnées pour Random Forest

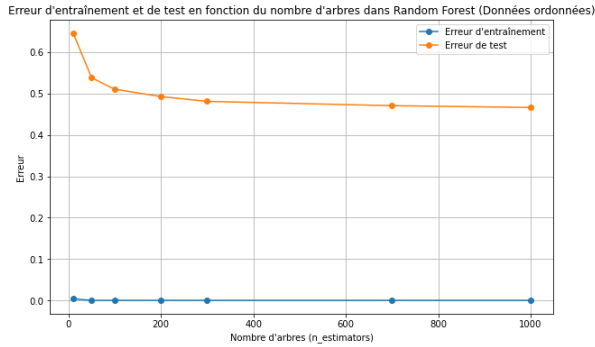
Erreur d'entraînement et de test en fonction du nombre d'arbres dans Random Forest (Données aléatoires)



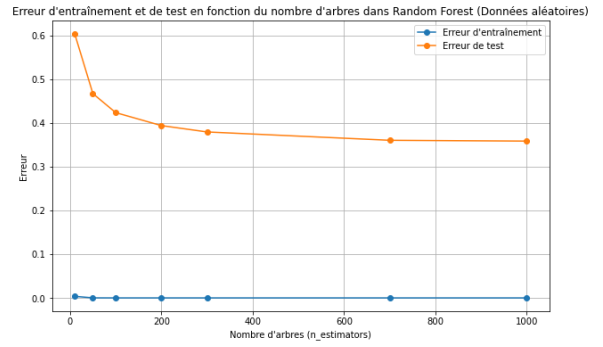
(16) Graphique de l'évolution de l'erreur quand l'ensemble d'entraînement contient 30% des données initiales pour Random Forest

Pour les résultats repris en (15) et (16), on retrouve :

- Une meilleure précision en validation croisée pour 200 arbres de 0.5562 pour les données ordonnées ;
- Une meilleure précision en validation croisée pour 200 arbres de 0.5573 pour les données aléatoires.



(17) Graphique de l'évolution de l'erreur pour les 50 premières personnes sélectionnées pour Random Forest

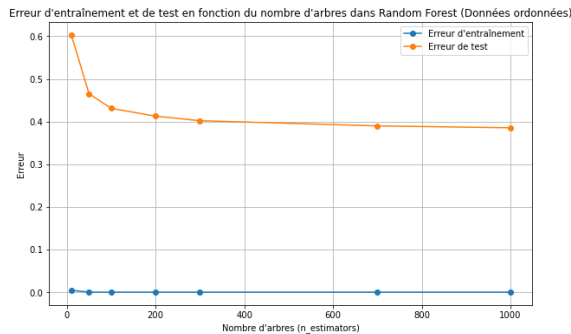


(18) Graphique de l'évolution de l'erreur quand l'ensemble d'entraînement contient 50% des données initiales pour Random Forest

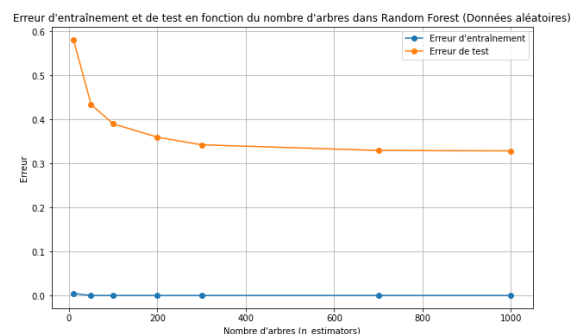
Pour les résultats repris en (17) et (18), on retrouve :

- Une meilleure précision en validation croisée pour 200 arbres de 0.6184 pour les données ordonnées ;
- Une meilleure précision en validation croisée pour 200 arbres de 0.5979 pour les données aléatoires.

On remarque que la séparation aléatoire devient moins précise que la séparation ordonnée en termes de validation croisée. Cependant, la précision (accuracy) est toujours plus élevée lorsque les données sont séparées aléatoirement. On interprète cette incohérence comme une conséquence de la manière dont les données sont présentées au modèle. Le modèle semble mieux gérer l'ordonnancement des données pendant l'entraînement dans le cadre de la validation croisée, mais il performe mieux avec des données aléatoires dans le test. Lorsque les données sont organisées de manière ordonnée, l'ensemble de test peut ne pas bien représenter la diversité des cas de données réelles.



(19) Graphique de l'évolution de l'erreur pour les 70 premières personnes sélectionnées pour Random Forest



(20) Graphique de l'évolution de l'erreur quand l'ensemble d'entraînement contient 70% des données initiales pour Random Forest

Pour les résultats repris en (19) et (20), on retrouve :

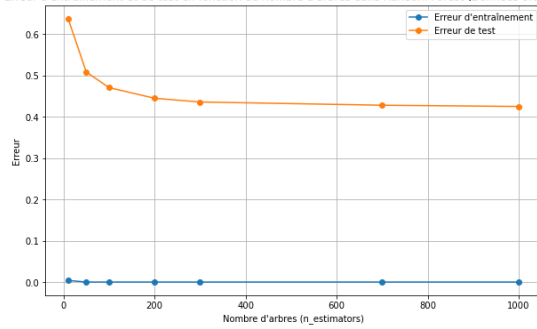
- Une meilleure précision en validation croisée pour 200 arbres de 0.6049 pour les données ordonnées ;
- Une meilleure précision en validation croisée pour 200 arbres de 0.6290 pour les données aléatoires.

Si on prend des valeurs intermédiaires (Ensemble d'entraînement 60%), on obtient les résultats suivants :

- Meilleurs paramètres pour les données ordonnées : {'max_depth': None, 'max_features': 'sqrt', 'n_estimators': 200} avec une précision en validation croisée de 0,6132 ;

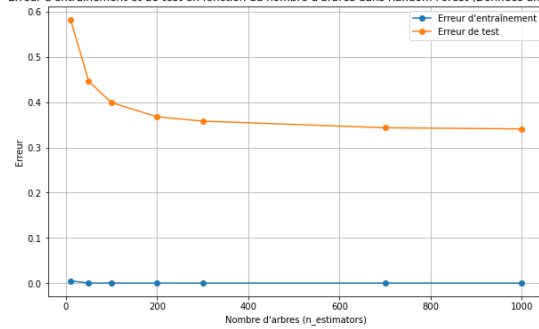
- Meilleurs paramètres pour les données aléatoires : {'max_depth': None, 'max_features': 'sqrt', 'n_estimators': 200} avec une précision en validation croisée de 0,6136

Erreur d'entraînement et de test en fonction du nombre d'arbres dans Random Forest (Données ordonnées)



(21) Graphique de l'évolution de l'erreur pour les 60 premières personnes sélectionnées pour Random Forest

Erreur d'entraînement et de test en fonction du nombre d'arbres dans Random Forest (Données aléatoires)



(22) Graphique de l'évolution de l'erreur quand l'ensemble d'entraînement contient 70% des données initiales pour Random Forest