

# IFT3395 Rapport Kaggle

Gabriel Beaudoin (20268138)  
Tom Sanic (20277286)

Novembre 2024

# 1 Introduction

Notre objectif pour ce projet est de concevoir un modèle de classification binaire pour trier des documents textuels en fonction de l'occurrence d'apparition des différents mots. Pour aborder cette tâche, nous avons expérimenté plusieurs algorithmes d'apprentissage, le modèle Naive Bayes a été notre premier réflexe comme il avait été mentionné dans le devoir 1. Pour la partie 2 nous avons essayé un réseau de neurones simple, et un modèle de Random Forest. Le modèle Naive Bayes s'est avéré être le plus performant, atteignant un F1-score de 73.5%, suivi du Random Forest avec 73% et du réseau de neurones avec 70%. Ce rapport détaille notre pipeline d'apprentissage automatique, incluant le prétraitement des données, le choix et l'évaluation des modèles, ainsi que la procédure de validation et de réglage des hyperparamètres. Nous analysons également les avantages et inconvénients de chaque approche et discutons des pistes d'amélioration possibles pour cette tâche.

## 2 Conception des fonctionnalités

Nous avons commencé par importer et visualiser les données afin de bien comprendre leur structure et les dimensions des matrices représentant la fréquence des mots. Cette étape exploratoire nous a permis d'avoir une meilleure compréhension du problème. Étant donné la structure des données, nous avons décidé d'utiliser directement les matrices fournies pour entraîner nos modèles. Bien que nous ayons essayé des techniques de réduction de dimensionnalité, comme l'analyse en composantes principales (PCA) et la sélection de caractéristiques par gain d'information, les résultats obtenus n'ont pas été satisfaisants. Nous avons donc choisi de conserver l'ensemble complet des données afin d'avoir davantage de caractéristiques sur lesquelles baser notre prédiction. Pour tester nos modèles, nous avons divisé les données en deux ensembles : un ensemble d'entraînement et un ensemble de validation. Cette séparation nous a permis de régler nos hyperparamètres et d'identifier les meilleures configurations pour chaque modèle. Ensuite, nous avons ré-entraîné notre modèle optimal sur l'ensemble complet des données d'entraînement en utilisant les meilleurs hyperparamètres trouvés. L'ensemble de test final pour l'évaluation est celui fourni sur Kaggle.

## 3 Algorithmes

### 3.1 Première étape

Notre premier modèle est un classifieur Naive Bayes, qui s'est révélé particulièrement efficace pour la classification de documents textuels. Nous avons appliqué le lissage de Laplace et tenté d'optimiser les hyperparamètres 'alpha' (pour le lissage) et 'beta' (pour ajuster l'influence des mots fréquents).

### 3.2 Deuxième étape

Nous avons essayé pour l'étape 2 un réseau de neurones artificiels en utilisant le MLP de scikit learn. Nous avons mené une recherche d'hyperparamètres sur le nombre de couches cachées, la régularisation 'alpha', et le taux d'apprentissage 'learning\_rate\_init' afin d'optimiser le modèle.

Nous avons également mis en oeuvre un arbre aléatoire et une random forest en utilisant tree et RandomForestClassifier de scikit learn. Les Paramètres optimisés pour la random forest sont n\_estimators, max\_depth, min\_samples\_split, min\_samples\_leaf, max\_features (sqrt vs log2), class\_weight (balanced, balanced\_subsample) et ccp\_alpha pour l'arbre aléatoire.

## 4 Méthodologie

### 4.1 Naive Bayes (Première étape)

Comme dit précédemment, pour la première étape, j'ai implémenté un modèle Naive Bayes en m'inspirant de la méthode vue dans un devoir 1. J'ai appliqué un lissage de Laplace pour éviter les probabilités nulles lors de la prédiction, et j'ai utilisé une fonction logarithmique pour éviter les problèmes de valeurs très petites. Comme mentionné précédemment, pour tester nos modèles, nous avons divisé les données en deux ensembles : un ensemble d'entraînement et un ensemble de validation. Ensuite, nous avons ré-entraîné notre modèle optimal sur l'ensemble complet des données d'entraînement en utilisant les meilleurs hyperparamètres trouvés. L'ensemble de test final pour l'évaluation est celui fourni sur Kaggle. Pour notre modèle le plus performant possible, nous avons exploré l'optimisation du paramètre de lissage de Laplace ( $\alpha$ ), initialement fixé à 1. L'optimisation de l'hyperparamètre  $\alpha$  s'est déroulée en deux phases distinctes. Nous avons affiné la plage de recherche en passant de 0.1 à 5 puis avons effectué une recherche sur l'intervalle [0.7, 1], qui a révélé une performance optimale pour  $\alpha = 0.788$ , atteignant un score macro F1 de 0.716. Pour affiner davantage nos résultats, nous avons conduit une seconde phase d'optimisation sur l'intervalle [0.775, 0.81]. Cette analyse plus précise a identifié un plateau, et nous avons pris  $\alpha = 0.775$  comme valeur optimale, maintenant le score macro F1 à 0.7158. Suite à cette première optimisation, nous avons tenté d'améliorer notre modèle en introduisant une combinaison linéaire entre

notre probabilité avec lissage de Laplace personnalisé et la probabilité conditionnelle pure, selon la formule :

$$P_{\text{final}}(w|d) = \beta \cdot P_{\text{laplace\_custom}}(w|d) + (1 - \beta) \cdot P(w|C)$$

Cette approche a nécessité une recherche d'hyperparamètres en deux dimensions, optimisant simultanément  $\alpha$  et  $\beta$ . Après une recherche exhaustive, nous avons retrouvé  $\alpha = 0.775$  comme valeur optimale, et obtenu  $\beta = 1$ . Ce résultat, bien que décevant, confirme que notre approche initiale avec le lissage de Laplace personnalisé était déjà optimale, puisque  $\beta = 1$  signifie que le modèle ignore complètement le terme  $P(w|C)$  dans la combinaison linéaire.

## 4.2 Réseau de neurone (Deuxième étape)

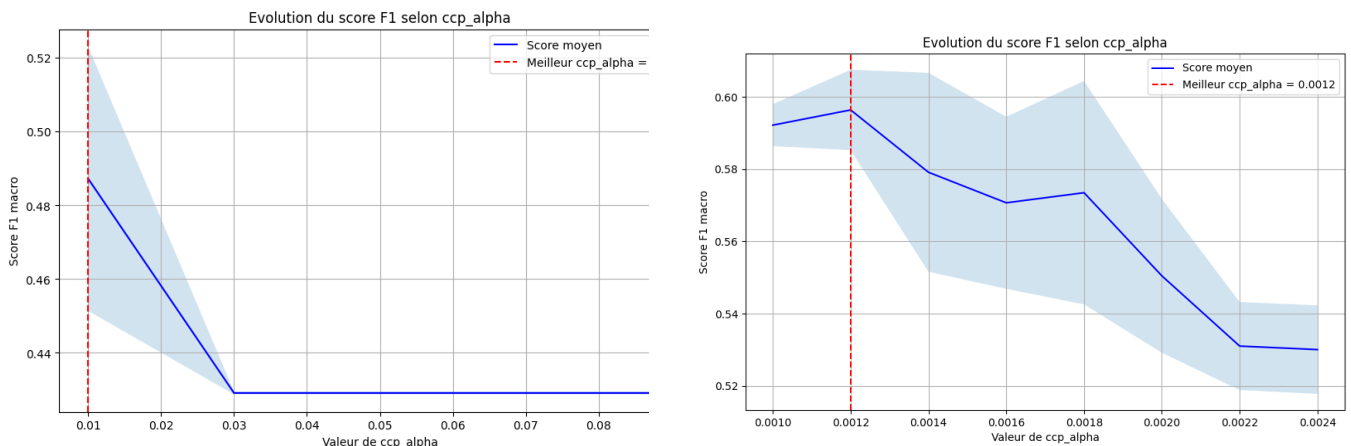
Nous avons implémenté un réseau de neurones en utilisant la bibliothèque MLPClassifier de sklearn. J'ai divisé les données en deux ensembles : un ensemble d'entraînement et un ensemble de validation. Ensuite, j'ai effectué une recherche d'hyperparamètres pour optimiser les performances du modèle. Les hyperparamètres explorés incluent la taille des couches cachées (hidden\_layer\_sizes), le paramètre de régularisation (alpha), et le taux d'apprentissage initial (learning\_rate\_init). Pour tester nos modèles, nous avons utilisé une fonction de séparation des données qui mélange les échantillons et les divise en ensembles d'entraînement et de validation. Ensuite, nous avons ré-entraîné notre modèle optimal sur l'ensemble complet des données d'entraînement en utilisant les meilleurs hyperparamètres trouvés.

## 4.3 Arbre aléatoire (Deuxième étape)

Recherche d'hyperparamètres pour pruner l'arbre:

Nous avons commencé par utiliser la classe RandomTree de scikit-learn, sans réaliser qu'elle utilisait la méthode Gini par défaut et non l'entropie.

Étant donné que l'entropie est généralement plus efficace que Gini dans le cas de jeux de données déséquilibrés, comme c'est le cas pour cette compétition, nous avons effectué une recherche d'hyperparamètres avec gini puis avec l'entropie car avec GridSearchCV. Voici les graphes pour l'entropie :



Les deux approches gini et entropy ont donné des résultats très similaires avec une variance assez élevée (nous pouvons constater un écart de plus 6% entre les validations croisées ce qui est très important). De plus, une moyenne maximale d'environ 59% est très basse comparée à nos autres modèles.

Cela est sûrement dû au surapprentissage.

Face à ces performances insatisfaisantes, nous avons décidé de passer aux Random Forests, qui devraient mieux résister au surapprentissage et offrir de meilleures performances plutôt que de perdre du temps à améliorer ce modèle plus faible.

## 4.4 Random Forest (Deuxième étape)

### Vue d'ensemble

Nous avons d'abord effectué des recherches avec RandomizedSearchCV. Nous avons ensuite procédé à une recherche hyperparamètre par hyperparamètre (car leur impact était difficilement distinguable avec tout un mix). Nous avons donc pris de petites valeurs de max\_depth et n\_estimators pour que les recherches prennent moins de temps. Finalement, nous avons réalisé une recherche linéaire de tous les hyperparamètres après avoir une bonne idée de comment impactait chaque hyperparamètre.

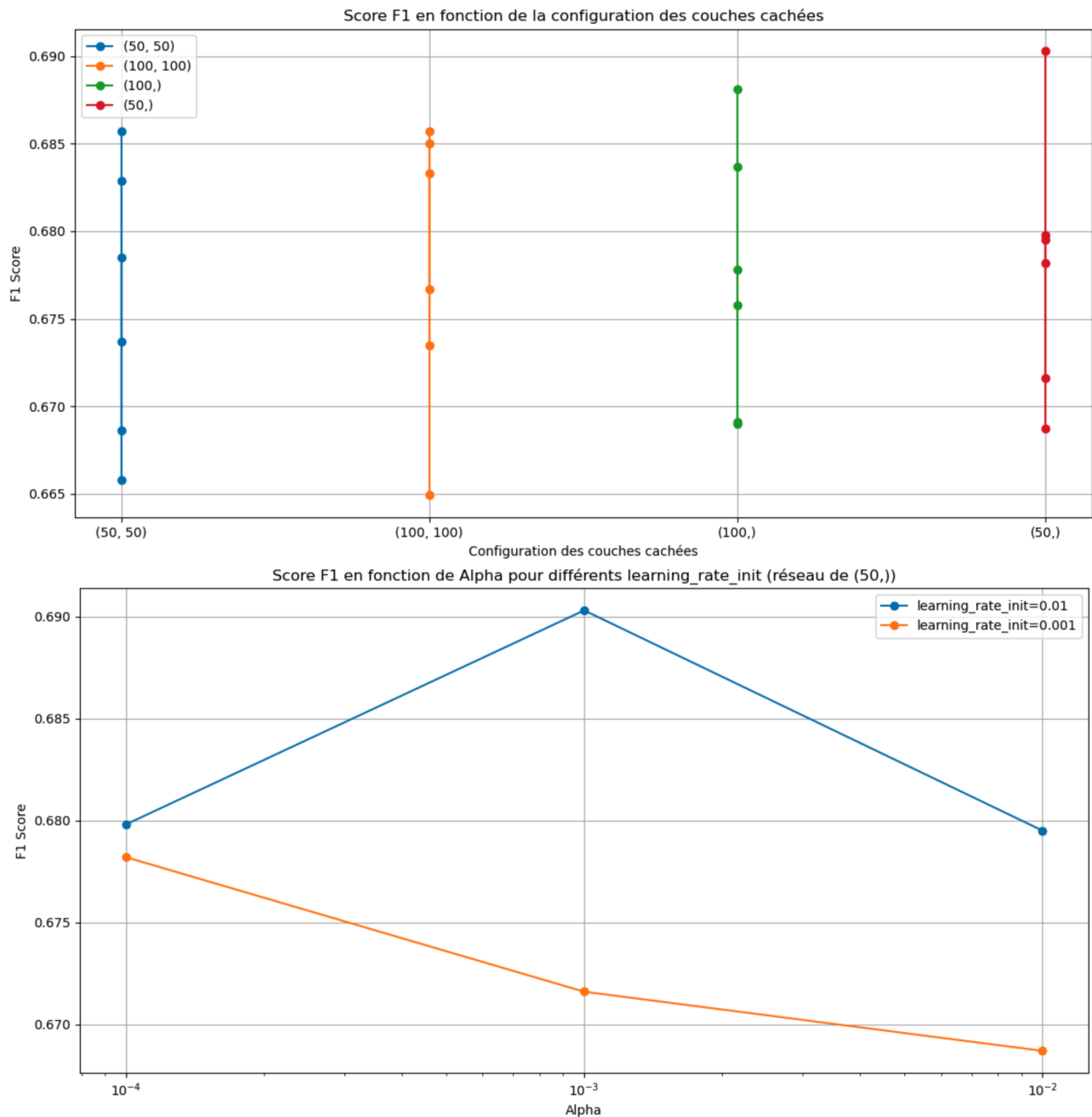
Nous avons utilisé  $cv = 3, 4$  ou  $5$  car la variance de ce modèle est bien plus faible que celle de l'arbre aléatoire.



## 5.2 Deuxième étape

### Réseau de neurones

Pour le réseau de neurones, nous avons testé différents hyperparamètres et avons remarqué que la taille du réseau n'affecte pas beaucoup la performance. Cependant, les réseaux avec une seule couche cachée montrent une légère amélioration des performances. Il est donc préférable d'opter pour un petit réseau avec une couche cachée de 50 neurones. En testant différents taux d'apprentissage (learning rate) et valeurs d'alpha, nous avons constaté que les meilleurs paramètres sont un alpha de 0,001 et un taux d'apprentissage initial de 0,01. Au final, nous obtenons un score de 70% sur Kaggle.



### Random forest

Dans notre dernière étapes nous avons affiné notre recherche (en passant de randomGridCV à GridCV) avec les paramètres suivants : **min\_samples\_leaf** : entre 3 et 9, **max\_depth** : entre 40 et 60 (car pas d'amélioration au dessus apart pour la variance mais très légère), **min\_samples\_split** : 5 et 20 (car pas de différence significative observée), **n\_estimators** : 150, 500, 1000

mean_test_score	std_test_score	params	rank_test_score
0.7248076402356666	0.004079373999909413	{'max_depth': np.int64(40), 'min_samples_leaf': np.int64(5), 'min_samples_split': np.int64(20), 'n_estimators': np.int64(575)}	1
0.724552951765201	0.006874653736156651	{'max_depth': np.int64(40), 'min_samples_leaf': np.int64(5), 'min_samples_split': np.int64(5), 'n_estimators': np.int64(1000)}	2
0.7237765727632338	0.004086874124467256	{'max_depth': np.int64(65), 'min_samples_leaf': np.int64(7), 'min_samples_split': np.int64(5), 'n_estimators': np.int64(1000)}	3
0.7226885659807657	0.006186569293734672	{'max_depth': np.int64(40), 'min_samples_leaf': np.int64(7), 'min_samples_split': np.int64(5), 'n_estimators': np.int64(1000)}	4
0.7225661987305472	0.00967948459740562	{'max_depth': np.int64(65), 'min_samples_leaf': np.int64(5), 'min_samples_split': np.int64(20), 'n_estimators': np.int64(575)}	5
0.7220547137636081	0.0034643615739522555	{'max_depth': np.int64(40), 'min_samples_leaf': np.int64(5), 'min_samples_split': np.int64(20), 'n_estimators': np.int64(1000)}	6
0.721897402702884	0.007903547831753684	{'max_depth': np.int64(65), 'min_samples_leaf': np.int64(5), 'min_samples_split': np.int64(20), 'n_estimators': np.int64(1000)}	7
0.7218178492789602	0.005123657807216601	{'max_depth': np.int64(40), 'min_samples_leaf': np.int64(9), 'min_samples_split': np.int64(5), 'n_estimators': np.int64(1000)}	8
0.7217983213022094	0.004831230714418824	{'max_depth': np.int64(90), 'min_samples_leaf': np.int64(7), 'min_samples_split': np.int64(5), 'n_estimators': np.int64(1000)}	9
0.7217204423998025	0.005577173209008841	{'max_depth': np.int64(40), 'min_samples_leaf': np.int64(7), 'min_samples_split': np.int64(5), 'n_estimators': np.int64(575)}	10

Figure 2: Top 10 score de la recherche

`max_depth = 40` apparaît dans 5 des 10 meilleures configurations

Les deux meilleures configurations ont : `min_samples_leaf = 5`, `max_depth = 40` On retrouve bien 8 fois parmi 10 1000 estimators ce qui montre qu'il est préférable mais seulement pour une très légère amélioration.

Les écarts-types sont relativement faibles (entre 0.003 et 0.009)

Nous allons donc tester (avec `max_depth = 40`):

```
{'min_samples_leaf': (5), 'min_samples_split': (5), 'n_estimators': (1000)}
```

```
{'min_samples_leaf': (5), 'min_samples_split': (5), 'n_estimators': (575)}
```

Résultat kaggle : 0.73034 et 0.72949 respectivement ce qui est un assez bon score même très légèrement meilleur que par la cross-validation !

## 6 Discussion

Pour le modèle de Naive Bayes, nous avons obtenu un modèle performant avec un score macro F1 de 73,5% sur l'ensemble de test. Ce résultat n'est pas particulièrement surprenant, car l'approche Naive Bayes avec bag of words, semble bien adaptée à ce type de problème de classification de textes. Ce modèle présente plusieurs avantages. Il est simple à mettre en oeuvre, rapide à entraîner. Cependant, cette méthode montre aussi ses limites lorsqu'elle est appliquée à des textes plus complexes. Pour améliorer la technique, nous pourrions tenter de capturer la relation entre les caractères du texte pour avoir une prédiction plus réaliste.

Pour notre réseau de neurones, bien que les performances obtenues soient légèrement inférieures au modèle de Naive Bayes (score macro F1 de 70%), cette approche reste prometteuse pour des tâches impliquant des relations complexes entre les termes. Cependant, un inconvénient majeur est le coût en termes de puissance de calcul, qui peut rendre l'entraînement et la recherche d'hyperparamètres longs et trop chère en ressources. En raison des contraintes de calcul, nous avons testé une gamme limitée d'hyperparamètres. Avec davantage de temps et un accès à des ressources de calcul accrues, une recherche d'hyperparamètres plus poussée aurait pu nous permettre d'améliorer les performances du réseau de neurones.

Pour la random forest, la stratégie de recherche progressive a permis d'explorer efficacement l'espace des hyperparamètres tout en maintenant des temps de calcul raisonnables. L'isolation des paramètres a fourni une compréhension approfondie de leur impact individuel. La stabilité des résultats, avec des écarts-types entre 0.003 et 0.009, confirme la robustesse du modèle.

Cependant, cette méthodologie comporte certaines limitations. La recherche séquentielle, bien qu'efficace en temps de calcul, peut masquer des interactions importantes entre les hyperparamètres.

Nous aurions aussi pu envisager l'implémentation d'une optimisation bayésienne permettant une exploration plus intelligente de l'espace des hyperparamètres

## 7 Conclusion

Les différentes techniques donnent sensiblement des résultats semblables, mais notre hypothèse de départ semble être bonne, car effectivement le Naïve Bayes obtient le meilleur score pour ce problème. Bien entendu, avec plus de temps et de ressources, nous serions en mesure d'obtenir des résultats sensiblement meilleurs.

## 8 Références

<https://medium.com/@dancerworld60/demystifying-naïve-bayes-log-probability-and-laplace-smoothing-d6da61b0e70b>  
[https://scikit-learn.org/dev/modules/generated/sklearn.neural\\_network.MLPClassifier.html](https://scikit-learn.org/dev/modules/generated/sklearn.neural_network.MLPClassifier.html)