Projet NLP ENSAE

Sujet n°2

Prediction du chef de ménage pour le recensement

Rendu final

Keywords: NLP / Text Classification / Recensement / Pipeline Scikit-learn / Bert fine-tuning

Elève (ENSAE) :
Antoine Klein

Professeur (TEKLIA) :
Dr. Christopher Kermovant

April 10, 2024

0 Accès à l'annexe et au code de l'étude

À la fin du document se trouve l'annexe qui contient des informations supplémentaires concernant ce travail. J'ai donc ajouté des **liens hypertextes cliquables** qui permettent au lecteur d'accéder directement à l'objet référencé et de revenir à la section qu'il était en train de lire. Ces liens sont surlignés en jaune, de la manière suivante : *annexe* par exemple.

Les codes utilisés et les fichiers générés au cours de ce projet sont disponibles sur le **dépôt Github** suivant : URL GITHUB.

Le dépôt comprend une liste complète des **scripts utilisés** pour le preprocessing, l'analyse, la visualisation et la modélisation. Les fichiers peuvent être facilement téléchargés et utilisés pour **reproduire les résultats** du projet ou adapter les méthodes à d'autres fins de recherche.

1 Contexte et objectif du projet

Si le recensement est aujourd'hui numérisé et donc au bon format pour y réaliser des traitements statistiques de grande échelle, cela n'a pas toujours été le cas. En particulier, entre 1836 et 1936, les données des ressortissants du territoire étaient renseignées dans des **archives au format papier avec une écriture manuscrite**. Le projet Socface dans lequel s'inscrit ce travail vise à **numériser, océriser**[1] puis **valoriser** ces archives pour y réaliser des études statistiques.

Cependant, ces archives se heurtent à différents enjeux:

- **Normalisation:** La manière de renseigner les informations n'est pas restée intacte sur la plage temporelle étudiée (ordonnancement différent, date de naissance VS age...),
- **Qualité de conservation** variable selon les manuscrits,
- Un grand **nombre de scripteur** avec chacun une écriture différente,
- **Erreurs commises** par le modèle d'océrisation,
- **Informations lacunaires** ou erreurs de retranscription manuelle.

Compte tenu de tous ces défis, nous nous proposons de **reconnaître le chef de ménage** (information parfois renseignée, parfois non). Il s'agit d'une étape cruciale pour reconstruire une **structure généalogique** dans ces archives et donc de mener des études statistiques plus informantes en ce que cela offrirait la possibilté de **suivre la descendance.**

2 Description des données

2.1 Identification et description de la population

Après étape de preprocessing (voir notebook), notre jeu de données océrisé se réduit à:

- **14 features, 25 074 individus**
- 5 640 chefs de ménage; soit **22.49 %** $<<$ 50 % : en cela nous avons à faire à un problème de **classification binaire imbalanced** (classe à prédire de taille non égale)

[1] Etape qui vise à extraire d'une image le texte qu'elle contient

En terme de **valeurs manquantes**, nous voyons que certaines features sont très rarement renseignées; notamment le niveau d'éducation:

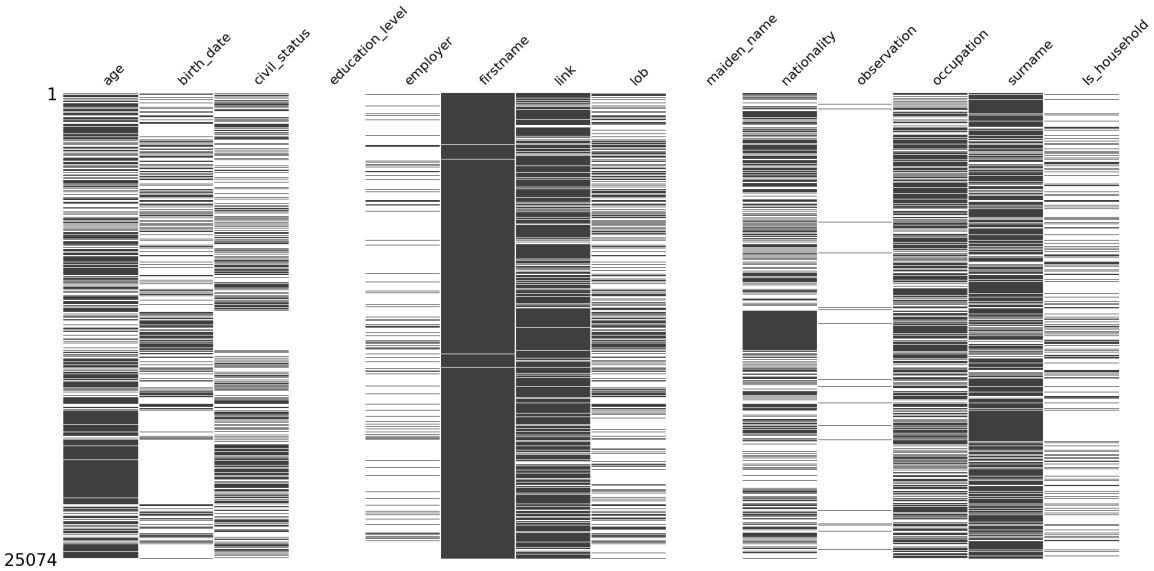

Figure 1: Matrice de sparsité de nos données:
case noire = donnée renseignée, case blanche = donnée manquante

Nous sommes par ailleurs face à une **population jeune** avec une **forte proportion d'enfants** (31 % ont moins de 18 ans). Quant au statut civique renseigné, nous obtenons une **parité pour le sexe** (49.1 % d'hommes pour 50.9 % de femmes) :

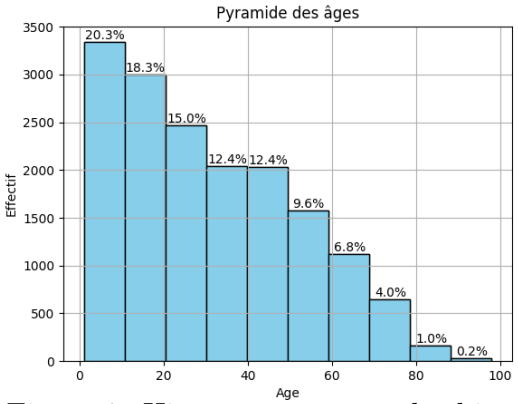

Figure 2: Histogramme avec des bins de largeur égale à 10 ans

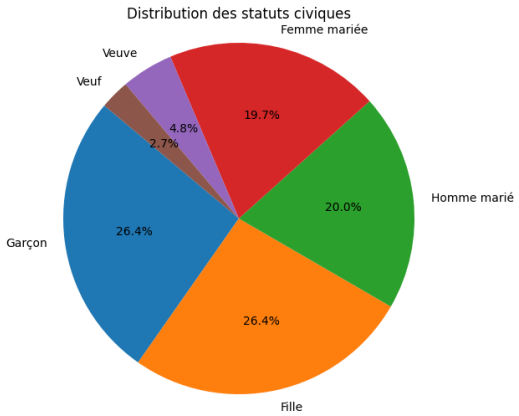

Figure 3: Répartition des statuts civiques

Enfin, en s'interessant à la colonne "link" qui contient, lorsqu'elle est remplie, le lien de l'individu avec le ménage, nous constatons qu'il est **possible de retrouver la grande majorité des chefs de ménage**. En effet, il nous est possible de **retrouver les 21.1 % sur les 22.49% en scannant les principales orthographes de chef de ménage** (voir notebook).

Cette approche (**la plus naïve possible**) montre que la tâche semble **totalement accesible avec une bonne précision** sans avoir à faire appel à des données extérieures en ce que les **features renseignent beaucoup la target** à prédire.

3 Benchmark de modèles

Nous explorerons en détail deux catégories de modèles.

- **Approche Machine Learning classique:** Ayant à disposition des données tabulaires et une tâche de classification binaire, un bon réflexe voudrait d'implémenter une architecture **Random Forest[3]/ HistogramGradientBoosting[2]**. Ces modèles ont fait leur preuve pour des **données de tout type** (catégorielle, numérique...) ce qui est le cas ici.
- **Approche Deep Learning:** Importer une **architecture BERT[1]** avec son **mécanisme d'attention** et faire un **finetuning** sur sa couche classification avec nos données.

Chaque approche apporte ses avantages et ses inconvénients. En effet, l'approche classique permet de **garder un controle total** sur le modèle en ce qu'il est facile de **tout implémenter soi-même**. En particulier, nous avons utilisé les outils de la **librairie scikit-learn[4]** pour construire un pipeline reproductible (voir notebook) composé d'un **OrdinalEncoder** et d'un **SimpleImputer** pour gérer les données manquantes ou leur typage. Pour autant, une telle démarche ne prend pas en compte **l'ordonnancement et la sémantique** globale de la ligne. Le **mécanisme d'attention** apporté par les modèles comme BERT situe chaque donnée dans son **contexte** et permet de déceler des structures, des pattern. Cette approche, si elle perd dans le controle absolu de son implémentation, tire profit de son pré-entrainement et donc d'un corpus très large. On s'attend donc à ce que ce modèle puisse bien **mieux généraliser**. Nuançons quand même que ce large corpus est **contemporain** (donc non similaire à nos données de recensement). En cela, BERT peut venir avec **des biais.**

Une autre différence majeure vient de la **complexité des modèles.** Là où l'approche classique comporte un nombre limité de paramètres à apprendre, l'approche BERT demande un **temps de training et d'inférence conséquent** et donc des **ressources en GPU.** Cet aspect peut être particulièrement bloquant si l'on veut **garantir la reproductibilité** pour tous les utilisateurs ultérieurs.

Afin d'apporter des **indices empiriques** qui aideraient à la décision du modèle et aux performances que l'on peut en attendre, nous décidons de réaliser un **benchmark:**

Modèle	Accuracy (test set)	Recall	f1-score	Train_Time	Inference_Time
Approche Scikit-Learn (Machine Learning classique)					
RandomForestClassifier	0.915	0.92	0.92	2.549s	0.214s
BalancedRandomForestClassifier	0.917	0.92	0.92	1.798s	0.218s
HistGradientBoostingClassifier	0.928	0.93	0.93	**0.81s**	**0.122s**
BERT-based model (Deep Learning)					
BertForSequenceClassification_fine_tuned	0.859	0.86	0.84	708.442s	21.975s
BertForSequenceClassification_full_retrain	**0.987**	**0.99**	**0.99**	3259.86s	21.967s

Ce benchmark amène plusieurs constats:

- Les modèles de l'approche classique ont des **performances similaires** (et satisfaisantes!) pour des **coûts similaires**; autant pour l'entrainement que pour l'inférence.
- Parmi les modèles classiques, le choix s'orienterait vers **l'HistogramGradientBoosting** en ce qu'il obtient la meilleure performance de sa catégorie pour le moindre coût.
- De manière assez surprenante, le fine-tuning de BERT sur sa couche de classification obtient de **moins bonnes performances que les modèles classiques** tout en étant sensiblement **plus couteux** (même sur GPU) ! L'intuition voudrait que BERT a été entrainé sur un corpus très différent que celui de notre étude. En effet, un tweet ne possède pas du tout la même **structure linguistique** que des donées tabulaires : **biais de modèle significatif !**

- Pour autant, à conserver l'architecture BERT mais réaliser la **backpropagation sur tous ses paramètres**, nous obtenons des résultats remarquables : **0.987 % de test accuracy !** Cela s'approche donc de résultats que l'on pourait attendre d'un humain; surtout que nos données elles-mêmes contiennent des incohérences.

- Cependant, de telles performances s'obtiennent au prix d'un lourd temps de calcul: **rien n'est gratuit, tout est compromis !**

Avant de conclure, nous tenions à chercher à **comprendre ce qui a amené nos modèles à émettre leurs prédictions.** En terme de features utilsés, nous obtenons :

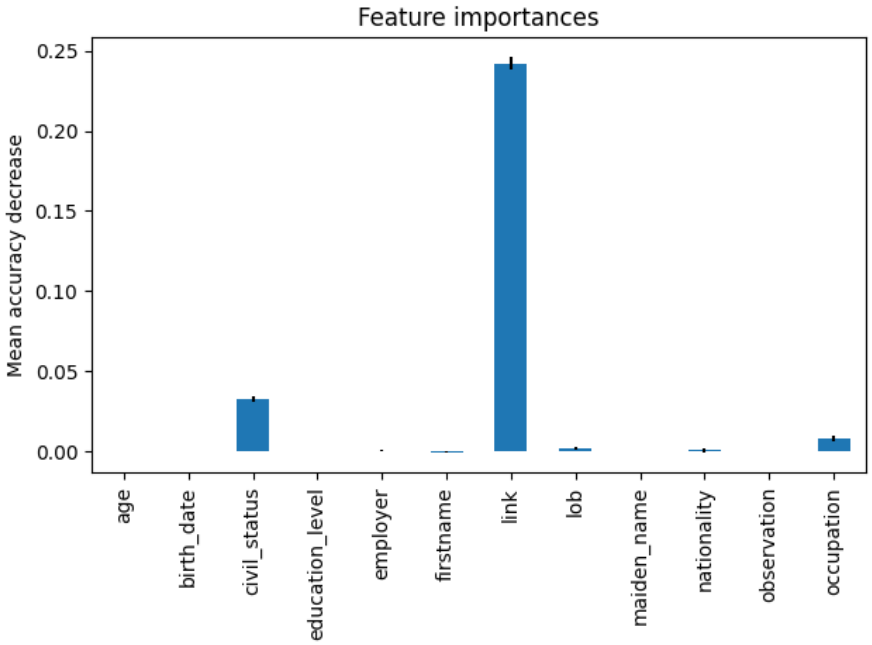

Figure 4: Feature importances de nos modèles classiques:
le renseignement du lien est bien la feature la plus impactante dans la prise de décision

Concernant **l'analyse des erreurs commises** par nos modèles, nous obtenons:

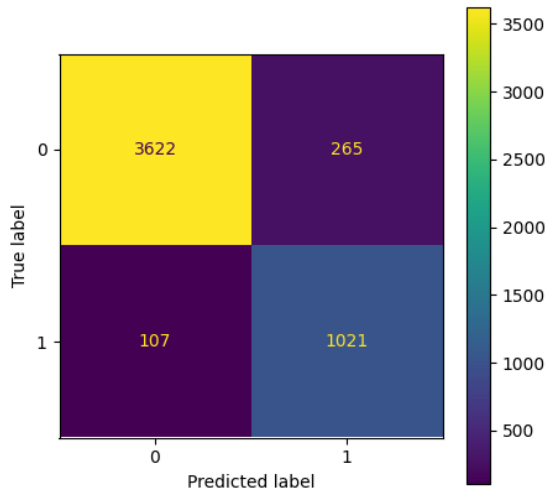

Figure 5: Matrice de Confusion pour HistogramGradientBoosting

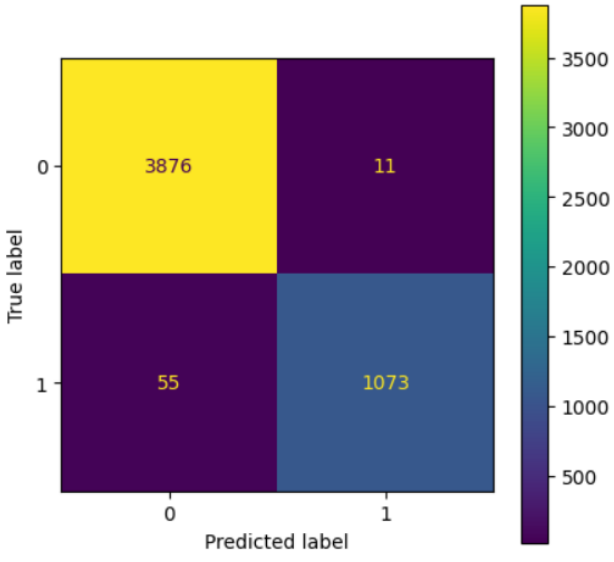

Figure 6: Matrice de Confusion pour BertForSequenceClassification_full_retrain

Nous constatons une tendance de l'approche classique à faire des **faux-positifs** (prédire un chef de ménage alors que non). L'approche BERT réalise quant à elle trop des **faux-négatifs** (ne pas prédire chef de ménage alors que si). Ainsi, l'approche BERT est **plus parcimonieuse** et donc **plus à croire** en ce que ses prédictions favorables sont presque toutes justes; propriété particulièrement **souhaitable** lorsque cette prédiction est elle-même injectée dans d'autres statistiques ce qui est le cas dans ce projet.

Notons que cette tendance reste valide pour tous les modèles implémentés, comme illustré *en annexe*

4 Conclusion

Nous avons réalisé un **benchmark** pour obtenir des indices; lesquels pointent vers les modèles les plus judicieux pour notre **tâche de prédiction de chef de ménage** avec nos données de recensement. Si tous les modèles fournissent des performances correctes, **la démarche à ne surtout pas suivre est celle d'un BERT fine-tuned seulement sur sa couche de classification.** Une telle démarche obtient les moins bonnes performances tout en restant onéreuse en plus de **souffrir d'un biais de modèle lié à son corpus.** L'arbitrage restant est **selon les ressources du projet :** si cela doit être reproductible, maitrisable, explicable et peu couteux, **l'approche HistogramGradientBoosting** couplé à un pipeline Scikit-learn satisfait toutes les attentes. Au contraire, si l'objectif est d'établir un modèle central que l'utilisateur va venir requêter (par API par exemple), alors je conseillerai de **ré-entrainer sur GPU BERT** en ce que cela fournit les **meilleures performances** tout en conservant un temps d'inférence raisonnable. De plus, ce choix est particulièrement **souhaitable** en ce que les prédictions sont **parcimonieuses** et donc **dignes d'être crues** pour de futures études et statistiques. Quoique soit le choix in fine du projet, cette étude apporte les **éléments de cadrages quantitatifs** autour de la tâche pour en établir un cahier des charges concrets.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[2] Aleksei Guryanov. *Histogram-Based Algorithm for Building Gradient Boosting Ensembles of Piecewise Linear Decision Trees*, pages 39–50. 12 2019.

[3] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

A Annexes

Retour à la section 0

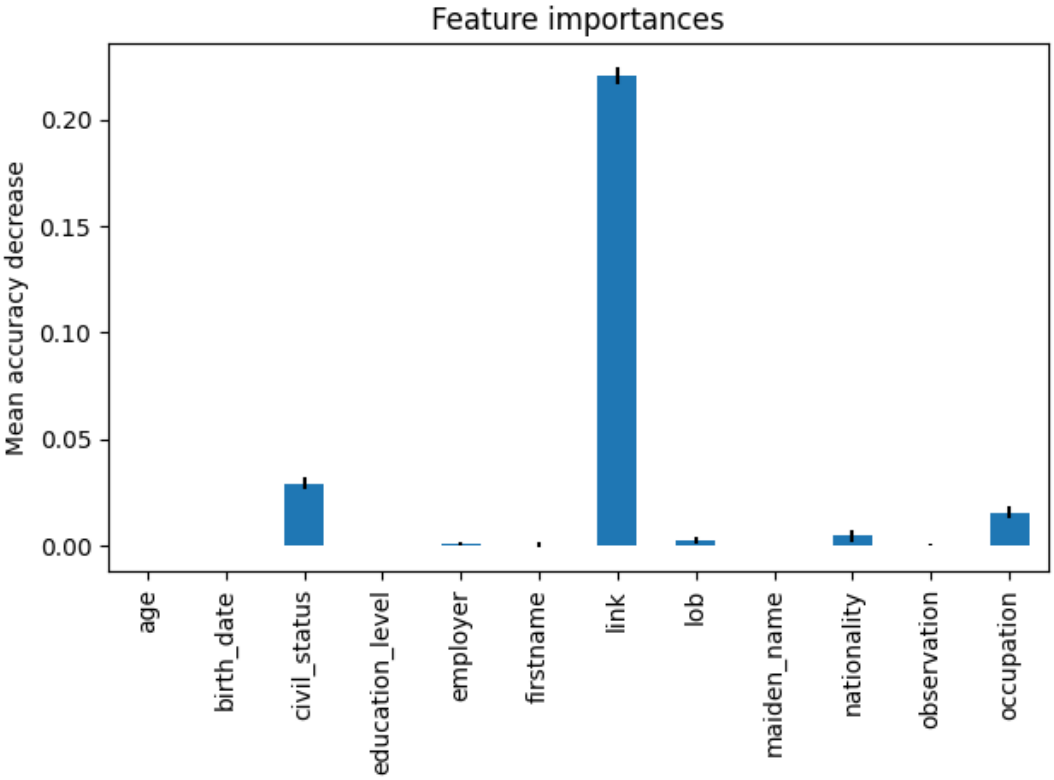

Figure 7: Feature importances pour la Random Forest

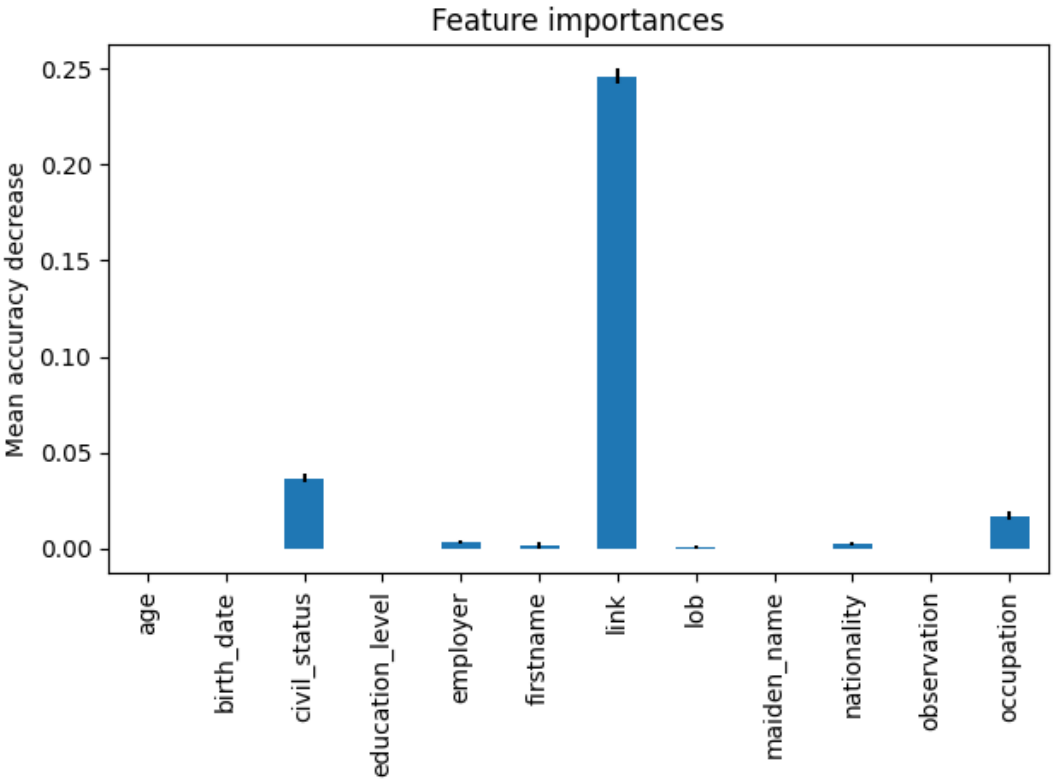

Figure 8: Feature importances pour la Balanced Random Forest

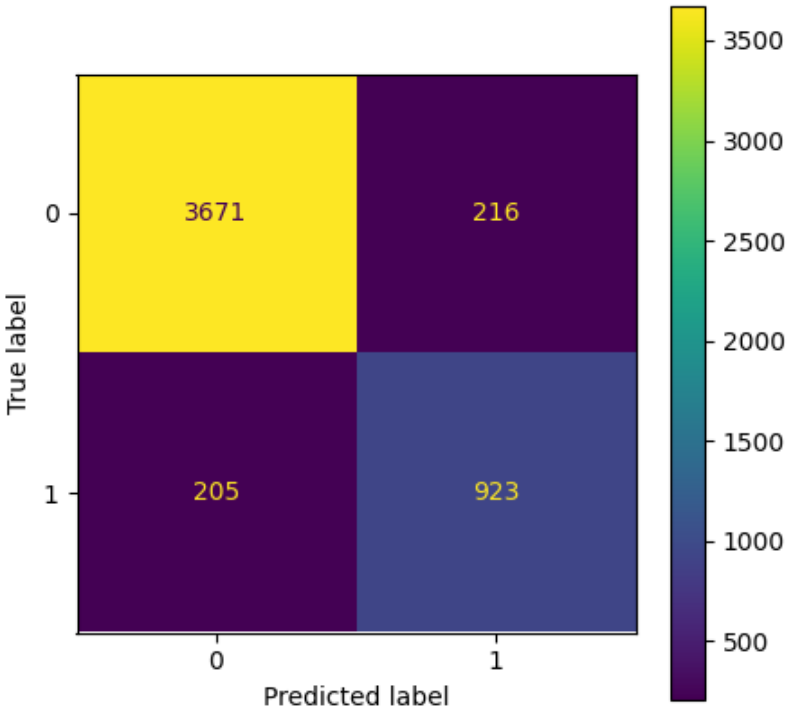

Figure 9: Confusion Matrix pour la Random Forest

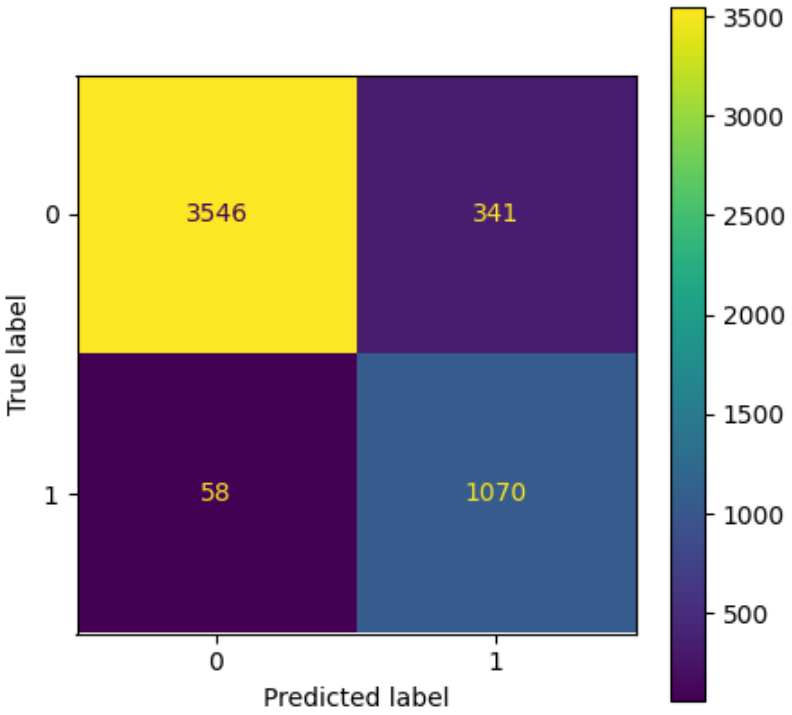

Figure 10: Confusion Matrix pour la Balanced Random Forest

Retour aux illustrations empiriques.