Rapport final - Traitement des données

Jacob Comeau[†], Gabriel Jeanson[‡], Alexandra Prémont[◇]
[†] Département d'informatique et de génie logiciel, Université Laval, Québec, Canada
[‡] Département d'informatique et de génie logiciel, Université Laval, Québec, Canada
[◇] Département d'informatique et de génie logiciel, Université Laval, Québec, Canada

Abstract

Les enjeux des médias traditionnels sont, de plus en plus, grandissants. Avec le récent blocage des nouvelles par Meta, il est plus difficile pour les individus d'interagir et de partager sur les nouvelles. Dans ce deuxième rapport du projet, on teste différentes variantes d'une forêt aléatoire et d'un *Category boosting* pour développer un système qui prédit si un commentaire sous une publication du journal Le Soleil sur Facebook reçoit au moins une réponse. Le système final sélectionné est un *Category Boosting* où l'entraînement s'est réalisé sur des données où un sous-échantillonnage a été appliqué. Il comprend 30 attributs et des plongements de mots de taille 150. Le taux de succès sans vrais négatifs est de 32,10 sur les données de test.

Mots-clés: Facebook, Réponse, Classification, Forêt aléatoire, *Category Boosting*

1. Introduction

Depuis le 1^{er} août 2023, Meta bloque l'accès au contenu d'information sur Facebook et Instagram. Cela vient limiter, du moins en ligne, l'échange entre les individus sur les différents sujets d'actualité. Pour ce projet, on s'intéresse à prédire quels commentaires des utilisateurs reçoivent des réponses sous les publications du journal Le Soleil sur Facebook. On dispose de données collectées par l'*API Graph* de Facebook.

Le projet est divisé en deux parties et, ainsi, deux rapports. Le premier rapport a permis d'explorer et d'analyser les bases des données et de prendre certaines décisions quant aux attributs initiaux, à la méthodologie et aux algorithmes à utiliser. Le premier rapport sert, en fait, de point de départ pour ce deuxième rapport. Une forêt aléatoire et un *Category Boosting* sont implémentés pour réaliser la présente tâche de classification binaire. On y présente, en fait, un processus de développement itératif pour ces deux algorithmes, des études de cas, la version finale du système et une rétroaction sur le projet.

2. Implémentation des algorithmes

L'objectif est de prédire si un commentaire sous une publication Facebook du journal Le Soleil recevra au moins une réponse. Deux algorithmes sont implémentés, soit une forêt aléatoire et un *Category Boosting*. Deux autres algorithmes ont été testés : un classificateur naïf de Bayes et un réseau de neurones. Les performances étaient plus faibles pour ceux-ci. De plus, l'entraînement du réseau de neurones a été plus long et complexe. Ces deux algorithmes ne sont donc pas présentés en détail dans ce rapport.

2.1. Forêt aléatoire

Le premier algorithme est une forêt aléatoire (*RandomForestClassifier*). Il est reconnu, en pratique, comme l'un des meilleurs algorithmes de classification. Il comporte plusieurs avantages : il permet d'éviter le fléau de dimensionnalité, il est moins sensible que d'autres algorithmes aux valeurs aberrantes et les arbres sont dé-corrélés. Ces avantages sont utiles, dans ce cas-ci, vu le grand nombre de dimensions, la présence de valeurs aberrantes et la forte corrélation entre certains attributs (voir rapport 1 pour plus d'informations).

Plusieurs variantes sont testées. Un des enjeux importants est qu'il y a un débalancement des classes pour l'attribut cible *has_answers*. On essaie trois approches pour traiter ce

[†]jacob.comeau.1@ulaval.ca [‡]gabriel.jeanson.1@ulaval.ca [◇]alexandra.premont.1@ulaval.ca

débalancement des classes. La première approche consiste à utiliser la fonction *BalancedRandomForestClassifier*. À la construction de chacun des arbres de la forêt aléatoire, il y a la pige d'un échantillon *bootstap* de la classe minoritaire ($has\_answers = 1$) et une pige du même nombre d'observations de la classe majoritaire ($has\_answers = 0$). La deuxième approche est d'effectuer un sous-échantillonnage aléatoire, et ce, avec différents ratios (voir prochaine section pour plus de détails). Le sous-échantillonnage permet de rendre le jeu de données plus petit et, alors, de faciliter l'apprentissage. La troisième approche est l'échantillonnage hybride. On commence par doubler le nombre d'instances de la classe $has\_answers = 1$ avec la fonction *SMOTE*. Cette fonction sélectionne aléatoirement une instance de la classe $has\_answers = 1$, recherche ses cinq plus proches voisins et crée une instance synthétique en prenant des combinaisons linéaires des attributs de l'instance initial et de ses cinq plus proches voisins. On supprime, ensuite, des instances de la classe $has\_answers = 0$ jusqu'à atteindre un ratio 2:1 par le sous-échantillonnage aléatoire. Les trois approches sont appliquées aux données d'entraînement et des forêts aléatoires sont entraînées à partir de là. Cela ajoute un nombre considérable de faux positifs. On recherche alors de nouveaux attributs qui permettent de différencier nettement la classe positive ($has\_answers = 1$) et la classe négative ($has\_answers = 0$), et ce, en fonction de différentes analyses et raisonnements logiques (voir prochaine section).

2.2. ***Category Boosting***

Le deuxième algorithme est un *Category Boosting* (*CatBoost*). Il s'agit d'un algorithme de *gradient boosting*, qui est appliqué avec des arbres de décision. Il y a, en fait, la construction itérative et indépendante d'arbres de décision relativement simples (modèles faibles). Ces arbres sont combinés pour former un modèle dit fort. L'ajout d'un nouvel arbre cherche à minimiser la perte obtenue avec les arbres précédents à l'aide de la descente de gradient. De plus, l'algorithme *CatBoost* permet de traiter les variables numériques et catégorielles, et ce, sans prétraitement spécifique. Il comprend des techniques de régularisation pour améliorer les performances et prévenir le surajustement. Cela le rend relativement robuste au bruit et aux valeurs aberrantes et approprié lorsqu'il y a un grand nombre de dimensions. On utilise la bibliothèque *CatBoostClassifier*.

On implémente cet algorithme avec une complexité algorithmique plus élevée en espérant augmenter les performances. On commence par tester, à nouveau, des approches pour pallier au débalancement des classes de l'attribut cible. On tente une approche supplémentaire à celle présentée pour une forêt aléatoire (en remplacement de *BalancedRandomForestClassifier* en quelque sorte). Il est possible de fournir en argument à *CatBoostClassifier* $auto\_class\_weights =' SqrtBalanced'$. Cet argument vient calculer automatiquement les poids en se basant sur la fréquence dans chacune des classes. Cela permet, en fait, de venir donner plus de poids aux instances de la classe minoritaire. On tente, en parallèle, d'ajouter des plongements de mots comme attribut afin de capter davantage l'informatique syntaxique et sémantique des attributs linguistiques. De plus, on effectue une sélection des hyperparamètres afin d'optimiser les performances. Cela inclut les trois principaux hyperparamètres de *CatBoostClassifier* et le seuil de classification. La sélection de l'hyperparamètre du seuil de classification permet de corriger, en partie, le nombre de faux positif. Plus de détails se retrouvent dans la prochaine section.

3. **Description et analyse des tests**

Les données sont divisées en trois : données d'entraînement (75 %), données de validation (15 %) et données de test (10 %). Dans la prochaine section, les données d'entraînement sont utilisées pour faire apprendre les différents systèmes et les données de validation permettent

de tester le système durant l'entraînement. Les données de test seront utilisées seulement à la fin pour avoir une idée des performances réelles du système final.

3.1. **Modèle de référence**

On commence par créer un modèle de référence à partir des attributs initiaux, c'est-à-dire une forêt aléatoire avec huit attributs qui se retrouvent directement dans les bases de données et qui ont été sélectionnés dans le premier rapport. Les attributs sont : *IDENTITY_ATTACK*, *INSULT*, *PROFANITY*, *SEVERE_TOXICITY*, *THREAT*, *TOXICITY*, *like_count* et *shares*. Le but de ce modèle est de fournir une base à partir de laquelle les améliorations du processus itératif peuvent être évaluées. On utilise les paramètres par défaut de *RandomForestClassifier*. Les résultats obtenus sur les données de validation se retrouvent à la table 1. Le rappel est relativement faible. Il y a, en fait, un grand nombre de faux négatifs (24 479) comparativement aux vrais positifs (4002). Cela s'explique, en partie, par le débalancement des classes pour l'attribut cible *has_answers*.

	Taux de succès	**Précision**	**Rappel**	**Score F1**	**Taux de succès sans vrais négatifs**
Modèle de référence	88,22	61,49	13,70	22,41	12,62

Table 1. Métriques d'évaluation de la forêt aléatoire avec les 8 attributs initiaux et 100 arbres

3.2. **Balancement des classes**

On tente alors de balancer les classes à l'aide des trois approches expliquées précédemment. Pour le sous-échantillonnage, on essaie quatre ratios (négatifs : positifs). Le ratio deux instances négatives pour une instance positive fonctionne généralement bien en pratique. On choisit donc quatre ratios en se basant sur cela. Les résultats obtenus sur les données de validation (non-balancées) se retrouvent à la table 2. Comparativement aux résultats pour le modèle de référence, le rappel, le score F1 et le taux de succès sans vrais négatifs sont plus élevés peu importe l'approche et le ratio utilisé. Or, il y a une diminution du taux de succès et de la précision. Cela est dû au fait que les approches de balancement des classes amènent, en quelque sorte, le système à prédire davantage d'instances positives, ce qui cause, par le fait même, l'augmentation du nombre de faux positifs. Il faut noter que toutes les approches viennent réduire la dimensionnalité lors de l'entraînement, ce qui réduit le temps de calcul des modèles. Les résultats pour le sous-échantillonnage et l'échantillonnage hybride sont similaires. On choisit de poursuivre le processus avec le sous-échantillonnage avec un ratio 1:1 comme l'approche est plus simple, permet d'avoir moins de dimensions et a des performances comparables pour le taux de succès et la précision aux autres approches et ratios et légèrement plus élevées pour le rappel, le score F1 et le taux de succès sans vrais négatifs. Une recherche d'hyperparamètres, notamment le ratio de sous-échantillonnage, sera exécutée une fois que la sélection d'attributs finale sera faite.

	Taux de succès	**Précision**	**Rappel**	**Score F1**	**Taux de succès sans vrais négatifs**
Forêt aléatoire balancé	71,26	24,79	64,69	35,85	21,84
Sous-échantillonnage 1:1	83,61	35,89	40,73	38,16	23,58
Sous-échantillonnage 1.5:1	79,69	30,50	49,75	37,82	23,32
Sous-échantillonnage 2:1	83,42	35,40	40,65	37,84	23,34
Sous-échantillonnage 2.5:1	85,53	40,26	34,31	37,05	22,74
Échantillonnage hybride	84,66	37,70	36,15	36,91	22,63

Table 2. Métriques d'évaluation des forêts aléatoires avec différentes approches et ratios pour traiter le débalancement des classes de *has_answers*

3.3. **Construction de nouveaux attributs**

À partir de ce modèle, on recherche des attributs pour mieux différentier la classe positive de la classe négative et, ainsi, améliorer les performances. On souhaite exploiter les attributs d'horodatage et les attributs linguistiques contenus dans les bases de données. Il faut noter que l'ajout d'attributs nécessite un temps d'extraction et augmente la dimensionnalité et, ainsi, le temps moyen d'exécution des algorithmes.

On procède par l'ajout itératif d'un attribut ou d'une catégorie d'attributs. Il faut noter qu'on garde, à chaque itération, tous les attributs testés précédemment et qu'on ajoute seulement un attribut ou un ensemble d'attributs. De plus, à chaque ajout, on utilise la même forêt aléatoire avec les données sous-échantillonnées avec un ratio 1:1 pour faire l'entraînement. On est alors en mesure d'analyser les gains de performance reliés à l'ajout de nouveaux attributs et de faire une sélection d'attributs. La table 3 regroupe toutes les itérations effectuées. La première ligne reprend les performances du modèle ajusté jusqu'ici, c'est-à-dire une forêt aléatoire avec les attributs initiaux de la table et avec du sous-échantillonnage 1:1. La colonne de gauche correspond à l'attribut ajouté ou à la catégorie d'attributs ajoutée à chaque itération. À première vue, on peut remarquer, avec la table 3, que quelques attributs se distinguent par les gains qu'ils apportent au modèle, notamment *post-comment_ time_ difference* et *has_ pers*.

Les sous-sections suivantes du rapport présentent l'intuition derrière la construction des attributs, des statistiques pertinentes et l'analyse des résultats obtenus par rapport aux résultats attendus. Le taux de succès sans vrais négatifs sera beaucoup utilisé comme il s'agit de la métrique qui servira à l'évaluation du modèle final sur les données du fichier *Test.csv*.

	Taux de succès	**Précision**	**Rappel**	**Score F1**	**Taux de succès sans vrais négatifs**
Attributs initiaux	83,61	35,89	40,73	38,16	23,58
post-comment_ time_ difference	83,45	36,02	42,91	39,16	24,35
comment_ word_ count	83,54	36,40	43,57	39,66	24,74
spelling_ error_ rate	83,84	37,12	43,54	40,08	25,06
has_ emoji	83,88	37,25	43,60	40,17	25,14
comment time attributes	84,25	38,12	43,08	40,45	25,35
post time attributes	84,40	38,54	42,64	40,48	25,38
sentiments	84,41	38,64	43,55	40,95	25,75
opinion	84,43	38,72	43,62	41,03	25,81
has_ pers	85,07	41,79	50,72	45,88	29,20
has_ loc	85,09	41,80	50,95	45,89	29,22
has_ org	85,10	41,82	50,97	45,89	29,23
comment_ post_ sim	85,18	41,88	51,39	46,01	29,74
comment_ title_ sim	84,91	41,29	51,12	45,68	29,60
post_ title_ sim	84,96	41,37	50,77	45,59	29,53
embeddings	85,60	43,69	42,28	43,92	28,14

Table 3. Performance du modèle *RandomForestClassifier* selon l'ajout itératif de différents attributs

3.3.1. **Temps entre la création de la publication et la création du commentaire**

Le premier attribut testé est *post-comment_ time_ difference*, qui correspond au temps entre le commentaire et la publication. À partir de la table 3, il est possible d'observer que l'ajout de cet attribut améliore grandement les performances. Ce résultat était attendu puisqu'on avait déjà analysé cet attribut au premier rapport et on avait obtenu une corrélation de *Spearman* de -0,1472 avec l'attribut cible. L'analyse plus complète de cet attribut se retrouve dans le premier rapport.

3.3.2. **Contenu syntaxique des commentaires**

On se concentre, maintenant, sur la création d'attributs reliés au contenu syntaxique des commentaires. On créé l'attribut *comment_ word_ count*, soit le nombre de mots par commentaire. Intuitivement, on peut penser qu'un commentaire avec peu de mots a une plus faible probabilité d'être répondu. Un commentaire de plusieurs mots a plus de contenu, comprend des opinions ou des explications plus élaborées, laisse place à davantage de débats ou de réactions et, ainsi, a une probabilité plus élevée d'avoir au moins une réponse. La corrélation de *Pearson* de cet attribut avec *has_ answers* est 0,0037 et sa corrélation de *Spearman* est 0,0100. Les corrélations sont positives comme il était attendu, mais elles ne sont pas très élevées. On observe, à la table 3, tout de même une augmentation des performances, dont près de 0,4 % pour le taux de succès sans vrai négatifs.

On analyse le nombre de fautes d'orthographe par commentaire. Par expérience, on peut voir souvent sur Facebook que les commentaires avec un grand nombre de fautes d'orthographe engendrent souvent des réactions. D'un côté, certaines personnes vont, parfois, commenter et argumenter à propos du nombre de fautes d'orthographe. D'un autre côté, on peut présumer que d'autres personnes ignorent un commentaire s'il comporte un grand nombre de fautes d'orthographe. On a, à la base, essayé l'attribut *spelling_ error*. On s'est aperçu qu'un commentaire avec plus de mots avait généralement plus de fautes d'orthographe. On a alors créé l'attribut *spelling_ error_ rate* : le taux de fautes d'orthographe en fonction du nombre de mots du commentaire. La corrélation de *Pearson* avec l'attribut cible est de -0,0107 et celle de *Spearman* de -0,0241. Comme les corrélations sont négatives, cela signifie, en quelque sorte, qu'un commentaire avec un haut taux de fautes d'orthographe a une probabilité plus faible de recevoir au moins une réponse. Cela s'aligne avec l'hypothèse formulée plus haut selon laquelle certaines personnes peuvent ignorer un commentaire contenant beaucoup de fautes d'orthographe. Les performances (sauf pour le rappel) sont légèrement meilleures avec l'ajout de cet attribut, mais ne sont pas marquées.

L'attribut *has_ emoji* est un attribut booléen permettant de définir si un commentaire contient un *emoji*. On peut penser qu'un commentaire contenant un *emoji* véhicule une forte émotion et est, ainsi, plus susceptible d'obtenir une réponse. C'est effectivement ce qui est observé. En effet, les coefficients de corrélation *Pearson* et de *Spearman* sont de 0,0188 avec l'attribut cible. De plus, l'ajout de cet attribut engendre une augmentation du nombre vrais positifs et du taux de succès sans vrais négatifs de près de 0,1 %.

3.3.3. **Horodatage**

Il est possible de construire différents attributs à partir des attributs d'horodatage inclus dans les bases de données. Cela a été exploré, en partie, dans le premier rapport. On y a formulé l'hypothèse suivante : on peut supposer qu'un commentaire publié à une heure où il y a un plus grand nombre de visites sur Facebook a une plus grande probabilité d'avoir au moins une réponse. En suivant ce même raisonnement, on extrait d'autres informations des attributs d'horodatage. Ces informations sont l'heure, le jour de la semaine, le jour du mois, le mois et l'année de création du commentaire et de la publication. Ces attributs sont regroupés et présentés dans la table 3 par les attributs *comment time attributes* et *post time attributes*. On évalue leur corrélation respective avec l'attribut cible. On constate que la corrélation la plus élevée est avec *comment_ hour*, suivi de *post_ hour*, avec des coefficients de corrélation de *Pearson* respectifs de 0,0190 et 0,0114. On décide d'ajouter ces attributs puisque quelques corrélations avec *has_ answers* sont notables. Avec la table 3, on voit que l'ajout de ces attributs permet d'augmenter les résultats globalement sauf le rappel.

3.3.4. **Contenu sémantique des commentaires**

On passe à l'analyse du contenu sémantique des commentaires. Il faut, d'abord, noter qu'il peut être difficile de bien extraire le contenu sémantique. On utilise des modèles pré-entraînés disponibles sur *Hugging Face*. On construit, en fait, trois catégories d'attributs à partir de ces modèles.

La première catégorie est l'analyse des sentiments des commentaires[1]. Elle se subdivise en trois attributs : positif, négatif et neutre où un score entre 0 et 1 est attribué pour chacun. On a vu, dans le cours sur les algorithmes de recommandation, qu'une des raisons pour laquelle les gens passent beaucoup de temps sur internet est la colère. On peut supposer la même chose par rapport aux interactions sur Facebook et, ainsi, s'attendre à ce qu'un commentaire plus négatif incite les gens à répondre, tandis qu'un commentaire plus positif ou neutre aura moins de réponse. C'est exactement ce que l'on observe avec les corrélations entre les attributs de sentiments et l'attribut cible à la table 4. On n'est donc pas surpris de voir un gain de performance avec l'ajout des attributs de sentiment.

Attribut	**Coefficient de Pearson**	**Coefficient de Spearman**
positif	-0,0200	-0,0156
négatif	0,0275	0,0229
neutre	-0,0152	-0,0120

Table 4. Coefficients de Pearson et de Spearman entre l'attribut cible *has_ answers* et les attributs de sentiment

La deuxième catégorie est le sujet des commentaires[2]. Cette nouvelle catégorie d'attributs se base sur le fait que certains sujets peuvent susciter davantage l'intérêt et, ainsi, augmenter la probabilité d'obtenir au moins une réponse. On est en mesure d'extraire huit sujets, qui sont présentés dans la table 5. Or, lors de nos premiers tests, on a remarqué que l'ajout des huit attributs faisait plutôt chuter les performances du modèle, notamment sur la métrique de taux de succès sans vrais négatifs. Cela n'était pas attendu. On a donc analysé les corrélations de ces attributs avec l'attribut cible comme on peut voir dans la table 5. Un des sujets qui se démarque est *opinion*. Il est, bien sûr, attendu qu'un commentaire donnant une opinion a plus de chance de recevoir une réponse. On a donc entraîné un modèle avec seulement cet attribut, ce qui améliore les performances. On a aussi testé plusieurs combinaisons de sujets, notamment avec *culture* et *sport*, qui ont une corrélation non-négligeable, mais seul *opinion* faisait monter les performances. On décide donc de conserver seulement *opinion* comme attribut de sujet lors de l'entraînement du modèle final.

Attribut	**Coefficient de Pearson**
opinion	0,0667
culture	-0.0243
économie	0,0056
environement	0,0108
politique	-0,0041
societé	-0,0021
sport	-0,0190
technologie	-0,0048

Table 5. Coefficients de Pearson entre l'attribut cible *has_ answers* et les attributs de sujet

[1]https://huggingface.co/lxyuan/distilbert-base-multilingual-cased-sentiments-student
[2]https://huggingface.co/lincoln/flaubert-mlsum-topic-classification

La troisième catégorie est les entités nommées[3]. L'idée au départ a été de trouver les commentaires qui contiennent le nom de personnes en supposant que si une personne est identifiée dans un commentaire, elle risque de répondre ou que si le commentaire concerne une personne spécifique, cela suscite plus de réactions. On crée alors l'attribut *has_pers*, qui a la valeur 1 si le commentaire contient un nom propre et 0 sinon. Tel qu'il est possible de constater avec la table 3, cet attribut semble, bel et bien, significatif. La corrélation de *Pearson* de -0,1693 entre l'attribut cible et *has_pers* montre bien ce lien. Dans le même ordre d'idées, on extrait deux autres entités nommées disponibles, soit *has_loc* et *has_org*. Il n'y avait pas de hautes attentes pour ces deux attributs, mais ils permettent d'améliorer légèrement le modèle. Ils vont cibler des cas plus rares et spécifiques et permettent d'augmenter le nombre de vrais positifs.

3.3.5. **Similarité publication-commentaire**

On calcule, maintenant, la similarité cosinus entre le contenu du commentaire et le contenu de la publication (*comment_title_sim*), entre le contenu du commentaire et le titre de la publication (*comment_post_sim*) et le contenu de la publication et le titre de la publication (*title_post_sim*). L'intuition derrière la construction de ces attributs est que quelqu'un qui prend le temps de lire une publication et les commentaires sous celle-ci est probablement intéressée par le sujet. Ainsi, si, par exemple, le contenu de la publication et le contenu du commentaire ont une forte similarité, il peut alors être plus enclin à répondre à ce commentaire. La table 6 permet d'observer la corrélation de ces attributs avec *has_answers*. Il semble, bel et bien, y avoir une relation *has_answers* pour les deux premiers attributs. La relation semble correspondre à l'intuition formulée plus haut. Or, la corrélation est négative et faible entre *title_post_sim* et *has_answers*. À la table 3, on remarque un gain de performance avec l'ajout de l'attribut *comment_post_sim* et une perte de performance avec les deux autres attributs de similarité. Le seul résultat qui n'était pas prévisible est pour *comment_title_sim*. En poussant les analyses plus loin, on a découvert que la corrélation de *Pearson* de *comment_title_sim* et de *comment_post_sim* est 0,8264. Il y a donc une certaine forme de redondance entre les deux attributs. On en conclut qu'il est préférable de garder uniquement *comment_post_sim*.

Attribut	**Coefficient de Pearson**	**Coefficient de Spearman**
comment_post_sim	0,0402	0,0584
comment_title_sim	0,0312	0,0536
title_post_sim	-0,0022	-0,0055

Table 6. Coefficients de Pearson et de Spearman entre l'attribut cible *has_answers* et les attributs de similarité

3.3.6. **Plongements de mots (*embeddings*)**

On tente, finalement, d'extraire les plongements de mots avec *Word2Vec* pour représenter de manière vectoriel le contenu des commentaires, le contenu des publications et le titre des publications. On essaie différentes tailles de plongements de mots. Les meilleurs résultats sont avec des plongements de mots de 150 et sont présentés à la table 3. Le nombre de dimensions ajouté est non-négligeable et cela ne permet pas d'améliorer les performances, du moins pour une forêt aléatoire. On n'écarte pas ces attributs pour *CatBoost* pour l'instant.

[3]https://huggingface.co/Jean-Baptiste/camembert-ner

3.4. **Recherche du meilleur modèle**

On a exploré la construction de nouveaux attributs pour mieux distinguer la classe positive et la classe négative. Les nouveaux attributs sélectionnés se retrouvent à la table 11 en annexe. On repart des constats faits et des attributs sélectionnés jusqu'ici. On souhaite implémenter, à partir de là, *CatBoost*. Il s'agit d'un algorithme avec une complexité algorithmique plus élevée vu la construction séquentielle des arbres et l'ajustement par descente de gradient, mais qui offre souvent de bonnes performances. On entraîne plusieurs variantes de l'algorithme *CatBoost* avec *CatBoostClassifier*. On a une version où on applique le modèle *CatBoost* sur les données sous-échantillonnées selon le ratio 1:1. On a une autre version où on prend *CatBoostClassifier* et on ajoute l'argument $auto\_class\_weights =' SqrtBalanced'$. Cette version requiert des ressources computationnelles plus élevées comparativement au sous-échantillonnage puisque toutes les instances sont conservées. On refait les deux mêmes modèles en ajoutant les plongements de mots pour voir l'effet de ceux-ci sur ce type de classificateur. Les résultats obtenus sur les données de validation se retrouvent à la table 7. On y observe que les résultats sont comparables et similaires. Certaines approches dominent selon une certaine métrique et d'autres pour une autre métrique. On remarque que le modèle *CatBoost* avec sous-échantillonnage selon le ratio 1:1 et avec les plongements de mots obtient les meilleures performances pour le rappel, le score F1 et le taux de succès sans vrais négatifs. Il faut noter que plusieurs ratios de sous-échantillonnage ont été testés (un peu comme ce qui a été fait pour la forêt aléatoire), mais seulement le meilleur ratio est présenté dans la table 7.

	Taux de succès	**Précision**	**Rappel**	**Score F1**	**Taux de succès sans vrais négatifs**
Forêt aléatoire + 1:1	85,18	41,88	51,39	46,01	29,74
***CatBoost* + 1:1**	84,34	40,34	54,65	46,41	30,22
CatBoost* + *SqrtBalanced	86,73	46,44	45,13	45,78	29,68
CatBoost* + 1:1 + *embeddings	84,33	40,55	56,35	47,16	30,86
CatBoost* + *SqrtBalanced* + *embeddings	86,55	45,95	47,33	46,63	30,40

Table 7. Métriques d'évaluation des modèles de forêt aléatoire et de *CatBoost* avec les attributs finaux

On passe, maintenant, à la sélection des hyperparamètres afin d'essayer d'optimiser les performances. La recherche d'hyperparamètre est un processus long, alors que pour chaque combinaison d'hyperparamètres testée, un entraînement du classificateur est effectué. On se contente alors de faire la recherche d'hyperparamètres pour le meilleur modèle développé jusqu'ici, soit *Catboost* avec un sous-échantillonnage selon le ratio 1:1 et avec des plongements de mots. Pour ce classificateur (qui est construit avec *CatboostClassifier*), il est possible de faire l'entraînement sur carte graphique contrairement au classificateur avec *RandomForestClassifier*. Cela accélère grandement l'entraînement. Les principaux hyperparamètres de *CatboostClassifier* sont : la profondeur (*depth*), la régularisation l2 ($l2\_leaf\_reg$) et le nombre d'itérations (*iterations*). On a fait une recherche par grille pour ces hyperparamètres et en se basant sur les résultats obtenus sur le jeu de données de validation. Pour la profondeur, la plage de valeurs est de sept à douze avec un pas de 1. On teste des valeurs de deux à six avec un pas de 1 pour la régularisation l2. Le nombre d'itérations est un hyperparamètre où plus la valeur est grande, plus les performances augmentent en général. Toutefois, le bénéfice diminue graduellement et il peut y avoir un risque de surajustement. La fonction comprend un arrêt anticipé par défaut, ce qui vient diminuer ce risque. En considérant tout cela, on fait varier le nombre d'itérations de 500 à 3000 avec des sauts de 500. On obtient les hyperparamètres suivants : profondeur de 11, régularisation de 5 et 2500 itérations. Cela permet d'atteindre un taux de succès sans vrais négatifs de 31,96 %, ce qui correspond à une augmentation de 1,1 %.

On tente de faire varier un dernier hyperparamètre, soit le seuil de classification. Avec *CatboostClassifier*, on a accès à la probabilité de prédiction pour chaque instance. Par défaut, si cette probabilité est supérieure à 0,5, on prédit que *has_answers* = 1 et 0 sinon. On obtient un seuil de 0,52. Cela fait passer le taux de succès sans vrais négatifs à 32,05 %. Ce nouveau seuil permet surtout de diminuer le nombre de faux positifs (et en plus grande proportion que l'augmentation du nombre de faux négatifs).

4. **Présentation d'études de cas**

On présente dans cette section quelques études de cas typiques qui nous aident à mieux comprendre le fonctionnement de l'algorithme. On étudie quatre exemples : un vrai négatif, un vrai positif, un faux négatif et un faux positif. Les exemples sont tirés du jeu de données de test. Ils n'ont donc pas contribué à l'entraînement du système.

Le commentaire 5268761896482590_5272690736089706 est un vrai négatif. Il n'a donc pas reçu de réponse et a été bien classifié. Certains de ces attributs sont distinctifs et arrivent, ainsi, à expliquer pourquoi le commentaire n'a pas eu de réponse. Le commentaire contient deux noms de personne (*has_pers* est donc égal à 1). Une de ses personnes est l'auteur d'un commentaire précédent. On peut donc penser que le commentaire est, en fait, en réponse à un autre commentaire. Le commentaire n'a pas de mention j'aime. La publication a un très faible nombre de partages, soit deux. On peut donc supposer que le commentaire n'a pas eu une grande visibilité. Le commentaire a des scores de toxicité assez faibles et n'est pas considéré comme une opinion, ce qui, selon les différentes analyses effectuées, semble moins susciter de réaction et, ainsi, de réponse de la part d'autres utilisateurs. De plus, le temps entre la publication et le commentaire est assez élevé (67 455 secondes), ce qui est environ le double du temps médian. Il faut noter que plusieurs de ces attributs dont *has_pers*, *like_count* et *response_time*, sont des attributs très importants pour l'algorithme tel que présenté dans la prochaine section.

Le commentaire 677630971034958_1282066419051275 est un exemple de vrai positif. Ce commentaire est un cas typique de commentaire qui reçoit une réponse. En effet, c'est un message qui ne contient aucune référence à un autre message ou à une personne (*has_pers* = 0). On s'aperçoit qu'il s'agit d'un commentaire d'opinion, avec des scores de toxicité élevés et avec un *emoji* qui accentue l'émotion véhiculée dans le commentaire. Le temps de réponse à la publication est extrêmement rapide avec 206 secondes. En plus, le commentaire est écrit en soirée à 21h, c'est-à-dire une heure où les gens sont souvent très actifs sur Facebook (selon les analyses réalisées pour le rapport 1). Le commentaire contient également quatre mentions j'aime. La probabilité reliée à cette prédiction est assez forte, soit de 86,62 %. C'est ce qui est attendu et ce qu'on espère dans un cas aussi évident!

Le commentaire 6695871240438308_6696867847005314 est un faux négatif. Il a reçu une réponse, mais a été classifié comme n'ayant pas reçu de réponse. On peut facilement voir pourquoi le classificateur s'est trompé. Pour l'analyse de sentiments, le score positif domine. Il n'est pas considéré comme un commentaire d'opinion. Il comprend le nom d'une personne. Le temps de réponse par rapport à la publication est relativement bas et en dessous de la médiane. Par ailleurs, le message est assez court et n'a aucune mention *j'aime*. En lisant le commentaire, on constate que le nom propre extrait est « Molson » qui réfère probablement à « Geoff Molson » et il propose le congédiement de ce dernier. Le commentaire ne répond donc pas à un autre commentaire. Le modèle d'analyse de sentiments n'est pas parvenu à détecter que proposer le congédiement de quelqu'un est plutôt négatif. Il faudrait donc un modèle plus précis au niveau de l'extraction des sentiments.

Le commentaire 4394844307207691_4395020253856763 est un exemple de faux positif. Ce commentaire a été classifié comme recevant une réponse alors qu'il n'en a pas reçu en réalité. On peut déceler pourquoi l'algorithme a mal classifié cet exemple. En effet, le temps

de réponse était très rapide avec 3749 secondes, le commentaire a une mention *j'aime*, il n'y a pas de personne mentionnée, le commentaire est assez négatif avec un score de 0,49 et la majorité des scores de toxicité sont autour de 0,25. Ce sont tous des attributs très importants pour la classification et ils ont, dans ce cas-ci, trompé le classificateur. Bien que le message ne mentionne pas personne, il fait quand même référence à un commentaire précédent avec les mots « votre affirmation ». Cela est assez difficile à capter par des méthodes de traitement de la langue naturelle plus traditionnelles. Il serait potentiellement possible d'utiliser ou de développer des méthodes plus complexes pour extraire des éléments qui suggèrent qu'un commentaire répond à un autre. Cela aiderait probablement à mieux classifier cet exemple, car, selon les analyses faites, un commentaire qui répond déjà à un autre semble avoir une probabilité plus faible d'obtenir au moins une réponse.

5. **Présentation de la version finale du système**

On a commencé par fusionner les bases de données de *Posts.csv* et de *Comments.csv* à l'aide des identifiants. On a traité les valeurs manquantes. Parmi les attributs sélectionnés au premier rapport, *message* et *title* de *Posts.csv* avaient des valeurs manquantes. On a réussi à ajouter une grande quantité de l'information manquante à l'aide du lien de l'article (*unshimmed_ url*). Après ce traitement, les instances avec au moins une valeur manquante ont été ignorées. Cela représente seulement 0,95 % des commentaires. Cela a été fait puisque *message* et *title* sont des attributs utilisés pour construire de nouveaux attributs qui sont significatifs pour l'algorithme développé. Par ailleurs, certains attributs contiennent du bruit ou des aberrations. Quelques attributs ont été retirés dès le premier rapport pour ces raisons, dont *mainTopic* et *secondTopic*. Sachant que les deux algorithmes implémentés sont relativement robustes à cela, il n'y a pas eu d'autres prétraitements spécifiques appliqués à ce niveau pour les attributs sélectionnés. On a fait la construction de nouveaux attributs à partir des attributs d'horodatage et linguistiques. À la base, les attributs extraits des attributs d'horodatage sont numériques. Ils ont été convertis en attributs catégoriels afin qu'un commentaire, par exemple, créé à en décembre (12) ne soit pas considéré plus proche d'un commentaire créé en novembre (11) que d'un créé en janvier (01). Il y a, d'abord, eu une tokenisation pour les attributs linguistiques. Les attributs suivants ont, ensuite, été créés : *comment_ word_ count*, *spelling_ error_ rate*, *has_ emoji*, *positive*, *négative*, *neutre*, *has_ pers*, *has_ loc* et *has_ org*. Avant de construire les attributs reliés aux sujets, à la similarité et aux plongements de mots, on a effectué un prétraitement, c'est-à-dire qu'on a uniformisé la casse, corrigé les erreurs, supprimé les mots vides et retiré les signes de ponctuation. Par ailleurs, on a utilisé plusieurs fonctions et modèles pour arriver à créer les différents attributs tel que présenté dans les autres sections.

L'algorithme final est un *CatBoost*. Un sous-échantillonnage a été appliqué au jeu de données d'entraînement pour atteindre le ratio d'une instance positive pour une instance négative. Cela vient corriger, lors de l'entraînement, le débalancement des classes. Le débalancement des classes a été un enjeu important dans ce projet, car un modèle qui ne prédit tout simplement pas de positifs peut atteindre un taux de succès supérieur à 80 %. On a fait une sélection d'attributs. Les attributs finaux sont à la table 11 en annexe. Il y a, en fait, 30 attributs et des plongements de mots de taille 150. On a, finalement, fait une sélection d'hyperparamètres. On a retenu une profondeur de 11, une régularisation l2 de 5, 2500 itérations et un taux de classification de 0,52.

La table 8 comprend l'importance relative des dix attributs les plus significatifs au modèle final. On observe, sans surprise, la dominance des attributs *has_ pers*, *like_ count* et *response_ time*. Ce résultat était, en effet, attendu en raison de la nature de ces attributs et de la haute corrélation avec l'attribut cible. Cela a été expliqué plus en détails dans différentes sections du rapport. Il faut noter, par ailleurs, que les plongements de mots ne sont

pas inclus dans cette analyse puisqu'ils correspondent à plusieurs centaines d'attributs et qu'ils sont moins interprétables dans le cadre de cette analyse. Cela explique aussi pourquoi les pourcentages sont relativement faibles dans la table 8.

Attribut	**Importance relative (%)**
has_pers	5,37
like_count	3,48
post-comment_time_difference	3,01
comment_word_count	1,80
shares	1,03
comment_post_sim	0,92
THREAT	0,85
PROFANITY	0,84
IDENTITY_ATTACK	0,83
neutre	0,66

Table 8. Importance relative des dix attributs les plus significatifs dans le modèle final

À la table 9, on retrouve les métriques de la version finale du système appliquée sur les données de test. On note le taux de succès sans vrais négatifs qui est de 32,10 %. Cela peut paraître relativement faible, mais il s'agit d'une amélioration majeure comparativement au modèle de référence au départ (forêt aléatoire avec huit attributs initiaux) où le taux de succès sans vrais négatifs est de 12,62 %. De plus, on remarque que même si le taux de succès est légèrement plus faible, il y a un meilleur équilibre, en quelque sorte, entre la précision et le rappel, ce qui se reflète dans le score F1 également. On peut présumer que les performances sur les données du fichier *Test.csv* seront similaires.

	Taux de succès	**Précision**	**Rappel**	**Score F1**	**Taux de succès sans vrais négatifs**
Version finale sur les données de test	84,97	42,20	57,24	48,60	32,10

Table 9. Métriques de la version finale du système et avec les données de test

La table 10 correspond à la matrice de confusion du système final et avec les données de test. On peut y constater un nombre relativement grand de faux positifs. Cela est dû au balancement des données lors de l'entraînement et où on vient encourager, en quelque sorte, le système à prédire des instances positives (*has_answers = 1*). On a essayé de contrebalancer cela par l'ajout d'attributs significatifs et permettant de bien distinguer les instances positives des instances négatives et par l'ajustement du seuil de classification.

	Prédictions négatives	Prédictions positives
Vrais négatifs	71463	8925
Vrais positifs	4871	6521

Table 10. Matrice de confusion du système final avec les données de test

6. **Rétrospective sur le projet**

À la base, cette tâche de classification binaire paraît plutôt simple. Au début de la session, on pensait qu'il allait être possible d'obtenir de bonnes performances. Avec l'élaboration du modèle de référence, on s'est rendu compte rapidement que ce n'est pas si simple. On a donc été surpris de la difficulté de la tâche à réaliser. Le processus de développement itératif a été assez long. De nombreux tests ont été réalisés. Dès le départ, on a considéré le débalancement de l'attribut cible comme un enjeu important de cette tâche de classification,

ce qui a, bel et bien, été le cas. Avec le balancement des classes, la construction de nouveaux attributs, la sélection d'un algorithme plus complexe et la recherche d'hyperparamètres, on a réussi à augmenter le taux de succès sans vrais négatifs de 19,43 % sur le jeu de données de validation en comparaison avec le modèle de référence. C'est un gain non-négligeable dans ce cas-ci!

Au premier rapport, on a indiqué vouloir tester une forêt aléatoire et un classificateur naïf de Bayes. On n'est pas arrivé à de bonnes performances avec ce dernier. On s'attendait à des performances similaires à celles obtenues avec les forêts aléatoires, mais ce ne fut pas le cas. On s'est tourné vers des systèmes avec une complexité algorithme plus élevée, soit un réseau de neurones et un *CatBoost*. On n'a pas atteint des performances satisfaisantes ou comparables à celles de la forêt aléatoire et de *CatBoost* avec le réseau de neurones, et ce, malgré que de nombreuses variantes ont été testées. Les algorithmes par ensemble semblent alors plus appropriés et performants pour cette tâche de classification.

Si le projet était à refaire, on mettrait une plus grande emphase sur la construction de nouveaux attributs à partir des attributs linguistiques. Avec les analyses de cas présentées, on voit qu'il y aurait potentiellement un intérêt à distinguer les personnalités publiques des personnes identifiées dans un commentaire et à extraire des indicateurs qui montrent qu'un commentaire répond à un autre (et n'est donc pas un commentaire racine) comme « votre affirmation ». On pense que ce type d'attributs peut améliorer le modèle en se basant sur l'hypothèse qu'un commentaire racine a probablement une plus forte probabilité d'avoir au moins une réponse. Or, les solutions pour extraire ces types d'attributs sont plus complexes, peuvent être moins flexibles et peuvent contenir une quantité notable de bruit. Par ailleurs, on pense que des attributs reliés à l'utilisateur seraient utiles pour améliorer le système (bien que ces informations ne sont pas rendues publiquement par l'*API Graph* de Facebook). Par exemple, on peut supposer que si un utilisateur avec un grand nombre d'amis ou d'abonnements sur Facebook commente une publication, cela apparaît dans le fil d'actualité d'un plus grand nombre de personnes (toutes choses égales par ailleurs) et que la probabilité qu'il reçoive au moins une réponse est plus élevée.

7. Conclusion

Rappelons que le but du projet est de prédire si un commentaire sous une publication du journal Le Soleil va recevoir au moins une réponse. Ce présent rapport comprend le processus de développement itératif, la description du système final, quatre études de cas sur celui-ci et une rétrospective sur le projet. Avec le système final et sur les données de test, on atteint un taux de succès sans vrais négatifs de 32,10 %. Ce résultat peut paraître relativement faible, mais il y a une amélioration notable avec le modèle de référence établi au départ et est convenable vu la complexité de la tâche.

Remerciements

Ce rapport reprend plusieurs notions et passages du cours « Analyse et traitement de données massives » (GLO-7027). Ils ont également guidé plusieurs analyses et choix.

Annexe

Attribut	Définition
IDENTITY_ ATTACK	Score de toxicité assigné par l'outil Perspective
INSULT	Score de toxicité assigné par l'outil Perspective
PROFANITY	Score de toxicité assigné par l'outil Perspective
SEVERE_ TOXICITY	Score de toxicité assigné par l'outil Perspective
THREAT	Score de toxicité assigné par l'outil Perspective
TOXICITY	Score de toxicité assigné par l'outil Perspective
like_ count	Nombre de *likes* du commentaire
shares	Nombre de partages de la publication associée au commentaire
post-comment_ time_ difference	Temps entre la création de la publication et la création du commentaire (en secondes)
comment_ word_ count	Nombre de mots du commentaire
spelling_ error_ rate	Taux du nombre de fautes en fonction du nombre de mots dans le commentaire
has_ emoji	Présence d'émoji dans le commentaire (1 si oui, 0 sinon)
comment_ year	Année de création du commentaire
comment_ month	Mois de création du commentaire
comment_ day	Jour de création du commentaire
comment_ hour	Heure de création du commentaire
comment_ day_ of_ week	Jour de la semaine de la création du commentaire
post_ year	Année de création de la publication
post_ month	Mois de création de la publication
post_ day	Jour de création de la publication
post_ hour	Heure de création de la publication
post_ day_ of_ week	Jour de la semaine de la création de la publication
positif	Score de positivité du commentaire
négatif	Score de négativité du commentaire
neutre	Score de neutralité du commentaire
opinion	Présence d'opinion dans le commentaire (1 si oui, 0 sinon)
has_ pers	Présence d'un nom propre dans le commentaire (1 si oui, 0 sinon)
has_ loc	Présence d'un nom de lieu dans le commentaire (1 si oui, 0 sinon)
has_ org	Présence d'un nom d'organisation dans le commentaire (1 si oui, 0 sinon)
comment_ post_ sim	Score de similarité entre le commentaire et la publication
comment_ embeddings	Plongements de mots du contenu du commentaire (150; seulement pour *CatBoost*)
post_ embeddings	Plongements de mots du contenu de la publication (150; seulement pour *CatBoost*)
title_ embeddings	Plongements de mots du titre de la publication (150; seulement pour *CatBoost*)
has_ answers	Commentaire reçoit au moins une réponse (1 si oui, 0 sinon). Attribut cible.

Table 11. Attributs finaux pour l'entraînement des modèles