Devoir 2 : Fouille de données

Simon Pelletier & Antoine St-Laurent
Département d'informatique et recherche opérationelle
Université de Montréal
Montreal, Québec, Canada

1 Introduction

Nous avons effectué une fouille de donnée des revues de jeux de la plateforme *steam* contenu dans le fichier *australian_user_reviews.json*. À partir de ces données, nous avons développé des méthodes pour extraire un maximum d'informations sur les commentaires en automatisant le plus possible le processusde fouille de données. Nous avons d'abord utilisé une approche par expression régulière pour détecter quelques noms de jeux, puis nous avons obtenu le reste des informations grâce à l'apprentissage de plongements de mots ou indirectement pas la classification des revues selon si elles accompagnent une recommandation ou non. Il est important de remarquer qu'il y a beaucoup plus de recommandation que de non-recommandations (tableau 1). Il est aussi intéressant de constater malgré que la plupart des utilisateurs ne font qu'un seul commentaire et la plupart de ceux-ci ne font jamais de commentaires négatifs, alors que la plupart de ceux qui font au moins un commentaire négatif en font au moins deux et font généralement plus de commentaires positifs que négatifs, bien qu'il y ait des utilisateurs qui font beaucoup plus de commentaires négatifs (l'utilisateur ayant fait le plus de commentaires négatifs en a fait 8 et 1 seul positif, pour un total de 9). Notre hypothèse est que plusieurs recommandations positives accompagnent des commentaires plutôt neutres, ce qui rendrait leur détection plus bruité que les commentaires négatifs qui semblent être moins ambigu. Dans ce travail, dont le code source est disponible en ligne

TABLE 1 – Statistique descriptive des commentaires contenus dans le fichier *australian_user_reviews.json*. Il y a au total 52 473 commentaires positifs, soit environ 88.5% du nombre total de commentaires (6832 commentaires négatifs et 59305 au total)

	Tous			**Au moins 1 positif**			**Au moins 1 negatif**		
	Positif	Negatif	Tous	Positif	Negatif	Tous	Positif	Negatif	Tous
Moyenne	2.03	0.26	2.30	2.15	0.22	2.36	2.41	1.34	3.75
Median	1	0	1	1	0	1	2	1	3
Max	10	9	10	10	8	10	9	9	10
#users	25799			24428			5087		

(https ://github.com/spell00/game_reviews_scraping), nous avons trouvé des noms de jeux par expression régulière et trouvé des mots pour les décrire grâce à des méthodes d'apprentissage machine. Nous avons utilisé un modèle pré-entraîné de *word2vec* (GoogleNews-vectors-negative300) que nous avons affiné sur les données du fichier *australian_user_reviews.json* pour trouver des *features* décrivant les jeux, des catégories de jeux ainsi que des adjectifs pour qualifier les jeux identifiés ou les *features*. Les listes de mot pertinents que nous avons extraites ne sont pas exhaustives, le modèle appris pourrait donc fort probablement renseigner sur encore davantage de choses sur le texte que ce que nous présentons dans ce rapport. Nous avons aussi identifié beaucoup de jeux supplémentaires grâce aux modèles de plongements que nous avons utilisés. Nous avons aussi effectué une Analyse Discriminante Linéaire (LDA) pour identifier les termes les plus fortement associés avec des revues positives ou négatives. Nous présentons les résultats que nous jugeons les plus intéressants, mais plus encore sont présentés sous forme de calepins Jupyter (*Jupyter notebooks*) sur le répertoire. Suite au téléchargement du projet sur Github, le modèle de plongements de mots *word2vec* doit être ré-entraîné car les points de sauvegarde sont trop volumineux pour être intégrés au répertoire, mais la

normalisation, qui était la partie la plus longue, n'a pas à être relancée (il y a les points de contrôle). L'un de nos objectifs était de créer une approche qui soit la plus automatisée possible, dans le but de pouvoir réutiliser facilement le script pour effectuer ultérieurement d'autres fouilles de données avec des nouvelles données sur les jeux vidéos, ou même sur des sujets qui ne sont pas nécessairement relié aux jeux vidéos.

2 Méthodes

2.1 Système d'exploitation

Nous avons développé une bonne partie du code sur Mac OS X (version 10.14.6) et aucun problème ne fut rencontré pour entraîner les modèles. Étant donné la mémoire limitée de la machine (128 Go) et la grosseur des fichiers enregistrés suite à l'entraînement d'un modèle à partir du modèle pré-entrainé de word2vec (total d'environ 12 Go en comptant le fichier du modèle pré-entrainé d'environ 3.5 Go), nous avons porté le projet sur une machine qui utilise Ubuntu 18.04 avec beaucoup plus de mémoire (2 To). La machine Ubuntu a plus de mémoire vive (16 Go versus 8 Go sur mac) et utilise une partition *swap* de 32 Go. Nous n'avons pas été en mesure d'entraîner le modèle de plongement sur Ubuntu (version 18.04), alors qu'il n'y ait jamais eu de problème sur Mac OSX. Sur Ubuntu, nous n'avons pas pu affiner le modèle de plongement *word2vec* car il y a une erreur de mémoire. Nous n'avons pu entraîner que des modèles d'Analyse Discriminante Linéaire (section 2.4) basé sur 2000 *features* ou moins. Après ce seuil, nous obtenions des erreurs indiquant un manque de mémoire.

2.2 Normalisation

2.2.1 Base

Nous avons effectué une normalisation pour améliorer les résultats des analyses en réduisant le nombre de mots dans le vocabulaire, tout en essayant de ne pas réduire le sens. Avec un corpus d'environ 60 000 revues faites par environ 27 000 utilisateurs différents avec une grande diversité de sujets (plusieurs jeux y sont révisés par les utilisateurs), le texte devrait profiter d'une normalisation pour faire ressortir des mots ou séquences normalement bruitées. Cela pourrait être davantage vrai pour faire des analyses sur des revues négatives, car elles sont beaucoup moins nombreuses. Nous croyons que la normalisation est davantage critique pour des analyses qui utilisent des vecteurs *one-hot* pour représenter les mots, car ceux-ci doivent avoir autant de dimensions qu'il y a dans le vocabulaire retenu pour l'analyse. Nous avons utilisé des vecteurs de comptes de mots (défini avec la classe *CountVectors* de *sklearn*) pour identifier les mots qui maximisent la discrimination entre des revues négatives et positives. La but de la normalisation du texte est d'enlever du bruit dans le texte et ainsi réduire la taille de vocabulaire. Il s'agit d'une opération délicate, car une mauvaise normalisation peut, au contraire de l'objectif, augmenter le bruit ou enlever des informations pertinentes par erreur. Par exemple, nous avons remarqué que la correction des fautes d'orthographe dans le texte avec *SymSpell* réduit effectivement la taille du vocabulaire, mais la correction n'est pas parfaite. Le texte ayant une quantité importante de fautes d'orthographe, nous avons opté pour l'effectuer au moment de la classification pour trouver les *features* les plus importants, mais nous les avons gardés pour apprendre les plongements de mots. Cette décision pourrait toutefois être remise en question, car le mot ayant la plus grande connotation positive est *gud* (figure **??**). Il s'agit d'une erreur de la correction de l'orthographe, car ce mot aurait dû être corrigé en *good*. Ce mot, et potentiellement plusieurs autres, font semble-t-il parti du *slang* dans le milieu du jeu vidéo et apparaissent fréquemment dans les commentaires. Le mot *good*, quant à lui, n'apparaît pas dans la liste (figure **??**). Si la correction avait été parfaite, il se pourrait que *good* soit au sommet de la liste, mais peut-être pas. Même si *good* se retrouvait au sommet de la liste, l'interprétation serait différente qu'en obtenant le mot *gud*.

2.2.1.1 Méthodes de normalisation utilisées Nous avons enlevé les espaces inutiles, les accents, la ponctuation et enlevé les lettres majuscules (certaines majuscules seront toutefois insérés lors des normalisation subséquentes (section 2.3.2 et 2.3.4), par exemple pour identifier les jeux). Nous avons aussi normalisé les *emojis* avec le package *demojize* et normalisé les expressions de rire par [RIRE] (nous avons réutilisé les méthodes que nous avions développé pour le TP2). Nous n'avons retiré les mots outils que pour faire la classification avec la analyse discriminante linéaire(en anglais, *Linear Discriminant Analysis* ou *LDA*) (section 2.4). Pour faire concorder le vocabulaire du corpus à analyser avec celui d'un modèle pré-entraîné utilisé pour apprendre des plongements de mots, nous avons

aussi normalisé les noms de jeux (section 2.3.2) que nous avons identifié ainsi que plusieurs mots et expressions du vocabulaire du modèle pré-entraîné (section 2.3.4).

2.3 Identification des noms de jeux videos

2.3.1 Identification de jeux par expression régulière

	n_mentions
Team Fortress 2	251
Nuclear Throne	232
Now	190
Dont Starve	86
Dota 2	79
The Launcher	73
Terraria	71
Insurgency Really	65
Portal	51
Sonic Generations	49
Counter Strike Global Offensive	43
The Game	40
Your Ship	37
Well There	36
Payday	34
The Culling	27
Garrys Mod	26
The Community	24
Starbound	22
Sanctum 2	20

FIGURE 1 – Top des jeux identifiés par expressions régulières (Liste complète dans word2vec.pdf, une copie du notebook)

Nous avons identifié plusieurs jeux précédent les mots *is a* sont suivis du mot *game* avant la fin de la phrase, ce qui permet d'identifier des jeux dans une séquence telle que *Team_Fortress_2 is a really great game* (figure 1). La séquence qui précède cette expression est identifiée comme un jeu si *is a* est en début de phrase. Si le nombre de mots dans la séquence ainsi identifiée est de 5 ou moins, la séquence est considérée comme étant le nom d'un jeu. Nous avons déterminé, de façon arbitraire, que les séquences de plus de 5 mots ne sont probablement trop longues pour être des noms de jeux. Quelques expressions pouvant désigner un jeu sans le nommer doivent aussi être enlevées (par exemple, nous avons enlevé *this*, *this game* ; nous en avons toutefois manqué certains comme *Now* ou *The Launcher* qui devraient être ajoutés à la liste). Alors, *Team_Fortress_2 is a really great game* détecte *Team_Fortress_2* comme étant un jeu, mais dans *The crazy game with the pyros and stuff is a really great game*, le groupe de mots *The crazy game with the pyros and stuff* est trop long (8 mots). Cela n'enlève pas tout le bruit, mais une partie.

2.3.2 Normalisation des noms de jeux

Les noms des jeux sont souvent plusieurs mots de long. Tous les jeux qui sont trouvés sont renommés pour ne faire qu'un seul mot (par exemple, *e.g. team fortress 2* devient *Team_Fortress_2*).

2.3.3 Plongements de mots

Puisque le corpus pour faire la fouille de données est relativement petit, nous avons initié les poids de notre modèle de plongements avec ceux d'un modèle déjà entrainé (*GoogleNews-vectors-negative300.bin* disponible à partir du lien : *https ://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit*. Nous avons toutefois un petit *script* bash qui télécharge le modèle et le place automatiquement dans le bon sous-dossier). Les plongements ont une longueur de 300, représentent environ 3 million de mots et expressions et ont été entraîné par Google sur environ 100 milliards de mots. Le vocabulaire représenté par les plongements est largement plus grand que celui dans *australian_user_reviews.json* (797514 mots dans les commentaires avant la normalisation). Nous avons utilisé le package *gensim* pour apprendre les plongements.

2.3.4 Normalisation des commentaire avec le vocabulaire de *word2vec*

Le modèle pré-entraîné de *word2vec* contient plusieurs entités nommées. Ces mots sont souvent en fait plusieurs mots collés ensemble par le caractère "_" (*e.g.* President_Barack_Obama). Pour que les mots du vocabulaire de *word2vec* puissent être efficacement utilisés pour analyser les critiques de jeux vidéo à notre disposition, nous avons modifié le texte pour remplacer tous les mots et expressions du texte qui ont une majuscule comme première lettre de chaque mot (dans le texte mis en minuscules lors de la normalisation de base, *portal* devient *Portal*, *team fortress 2* devient *Team_Fortress_2* ainsi que tous les jeux que nous n'avions pas identifié, s'ils sont dans le vocabulaire de *word2vec*). Plusieurs autres expressions que des entités nommées sont présentes dans *word2vec*, comme *Cant_Wait* (*figure* **??**), mais nous croyons que cela sera bénéfique pour que le *fine-tuning* que nous effectuons avec les données du fichier *australian_user_review* soit le plus efficace possible. Utiliser le pré-entrainement permet de trouver des voisins qui semblent pertinents même s'ils n'ont jamais été observé dans l'ensemble d'entraînement (figure 2). Il y a toutefois du bruit, comme *Spectroscobes_Beyond*, mais *Spectroscobes* n'est ni dans le corpus d'entraînement, ni même un vrai mot. Les entités nommés dans

la liste de voisins d'un jeu (*e.g.* Borderlands) étant presque tous des jeux, nous pourrions utiliser ces mots pour faire une nouvelle recherche dans le corpus d'entraînement. Plusieurs jeux pourraient être ainsi identifiés. Il est intéressant de constater que les termes comme *Team_Fortress_2*, que nous n'avons pas trouvé dans la liste de *word2vec*, sont liés à des jeux qui n'étaient pas dans le corpus de revues de jeux suite au *fine-tuning* du modèle. Cela démontre la capacité, avec peu d'entraînement, de faire des liens avec *word2vec* et ainsi augmenter considérablement la capacité d'inférence du modèle obtenu. La fonction que nous avons développé pour effectuer cette normalisation est de loin la plus longue de toutes les étapes effectuée pour ce rapport (il faut rechercher à remplacer environ 1 milliards de mots qui sont souvent composé de plusieurs mots dans les 60 000 commentaires). Ce processus prendrait environ 24 heures, ce qui est très long mais faisable dans le cadre de cette étude. Toutefois, 60 000 documents n'est pas très volumineux et nous n'aurions pas envisagé la faire sur un corpus 10 fois plus volumineux. Pour résoudre ce problème, nous avons parallélisé cette normalisation, ce qui l'a rendu 8 fois plus rapide (car la machine utilisée possède 8 processeurs) et termine en moins de 4 heures. Ce temps pourrait être réduit davantage en utilisant une machine avec beaucoup plus de processeurs disponibles, ce qui rendrait possible l'utilisation de notre fonction sur une grande quantité de documents qui ne devrait être exécutée qu'une seule fois.

2.3.5 Apprentissage par transfère du modèle *word2vec*

2.3.5.1 Objectif Augmenter l'inférence obtenue sur les mots du texte en utilisant le modèle pré-entraîné *word2vec*.

2.3.5.2 *Fine-tuning* Le modèle *word2vec* permet de trouver beaucoup d'informations sur les jeux vidéos, même sans entraînement supplémentaire sur les données de *steam*. Les informations contenues semblent tellement pertinentes que nous n'avions pas remarqué que les informations contenues dans la figure 2 n'avaient aucunement été apprises à partir du corpus sur lequel nous avions affiné (*fine-tuned*) le modèle *word2vec*. C'est en cherchant certains jeux comme *Super_Meat_Boy* (bons résultats à la figure 6), qui n'était pas identifié dans les plongements, que nous avons remarqué et corrigé l'erreur. Pour s'assurer de la validité des plongements de mots appris, nous cherchons les plus proches voisins de mots qui n'étaient pas dans la liste de *word2vec* mais qui devraient s'y trouver après le *fine-tuning*. Pour cela, nous chercherons les noms des jeux que nous avons identifiés par expression régulière (section 2.3)

2.4 Prédiction des commentaires de recommandation avec l'Analyse Discriminate Linéaire

. La classification fut effectuée dans l'idée d'obtenir les mots les plus importants pour la classification entre *Recommended* ou *Not Recommended*. La *Linear Discriminant Analysis*(LDA) est utilisée pour révéler automatiquement les mots ou expressions du texte les plus importants pour prédire si le commentaire est positif ou négatif (dans le fichier *json*, respectivement désigné par *'Recommend' : True* et *'Recommend' : False*) La LDA permet d'apprendre une représentation réduite de l'information pour optimiser la classification. Dans ce cas, nous optimisons pour *Not Recommended* (classe 0) et *Recommended* (classe 1). Une fois cette représentation apprise, il est possible d'obtenir les *features* les plus souvent associés avec chacune des classes. Toutefois, ces *features* ne sont intéressant que si une bonne classification est possible. Ce qui qualifie une bonne classification dépend du problème et peut être arbitraire. Dans notre cas, nous allons vérifier la validité de notre approche par validation croisée. Nous avons défini notre propre fonction pour effectuer la validation croisée (dans le notebook *Linear_Discriminant_Analysis_Steam.pdf* ; les résultats finaux sont toutefois dans LDAs.pdf). Le vocabulaire est redéfini pour chaque sous-ensemble de validation. Cela a pour but de vérifier si les prédictions seraient bonnes sur un ensemble de test pour lequel le vocabulaire est inconnu avant la phase d'induction. Selon notre expérience avec le package *scikit-learn*, les données qui sont passées dans la fonction de validation croisée doivent déjà être transformées en *CountVector*, ce qui nécessite d'avoir au préalable obtenu une liste de vocabulaire. Cette liste doit être seulement apprise sur les données d'entraînement, ce qui nécessite de redéfinir le vocabulaire et les *CountVector* à l'intérieur de la fonction de validation croisée.

3 Résultats

3.1 Identification de jeux

Il est difficile d'identifier les jeux en utilisant que des recherches par expressions régulières. Selon notre expérience, les résultats des fouilles de données basés sur des expressions régulières seraient très

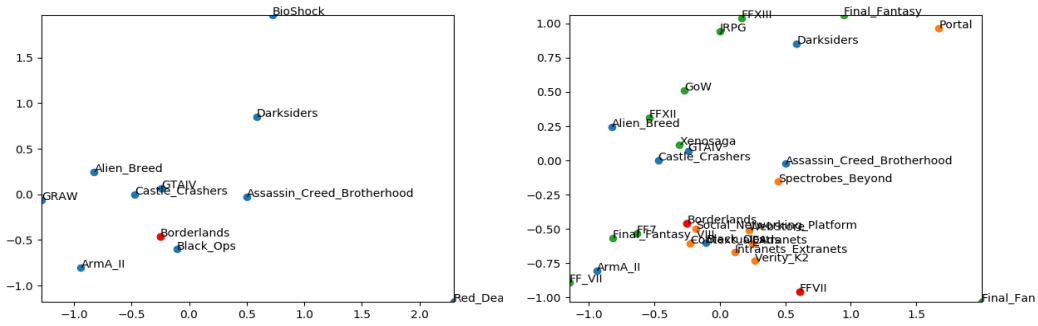


FIGURE 2 – (À gauche) PCA de voisins du jeu Borderlands. (À droite) PCA de voisins du jeu FFVII

bruités si l'on tentait d'extraire une liste complète des noms de jeux dans les données analysés. Il serait nécessaire de filtrer "manuellement" tous les résultats obtenus (encore plus que ce que nous avons fais), ce qui demanderait certaines connaissances du domaine pour identifier tous les jeux. La fouille de données automatique a comme avantage qu'elle permet à un utilisateur d'obtenir des informations juste. Il serait alors préférable de maximiser la justesse (*precision* en anglais) au détriment de perdre de la précision (*accuracy* en anglais). Avec le modèle entraîné à partir de *word2vec* déjà pré-entraîné, nous avons remarqué que plusieurs jeux sont déjà présents dans le vocabulaire (les noms de jeux de *word2vec* que nous avons recherchés semblent avoir une limite autour de 2010, soit quelques années avant *word2vec*). Il est aussi possible de voir plusieurs synonymes en consultant certains noms de jeux (*e.g. Final Fantasy VII, FF7, FF-VII, FFVII*) 2. Après le *fine-tuning*, les voisins des jeux ne sont plus autant majoritairement que des jeux, mais dans la liste de jeux obtenu sans *fine-tuning* (représentée dans la figure 2), tous les mots que nous avons vérifiés qui ont une majuscule pour première lettre étaient des noms de jeux (*Alien_Breed* ou *Assassin_Creed_Brotherhood*). Plusieurs de ces jeux sont probablement dans les commentaires à analyser et ont pu être manqué par notre approche par expression régulière, nous pourrions donc utiliser cette liste pour en détecter davantage (avec comme contrainte les noms de jeux contenu dans *word2vec*, qui ne pourraient toutefois pas identifier les jeux les plus récents). Les jeux ont souvent des suites, nous aurions pu chercher des suites (ou versions précédentes, comme *Team_Fortress_2* qui est identifié, mais pas Team_Fortress).

3.2 Extraction d'information sur les jeux à partir de plongements

Nous avons identifié plusieurs facettes et types de jeux en explorant le modèle de plongement *word2vec* appris avec les mots les plus proches soit de noms de jeux, soit en explorant les mots les plus proches de d'autres facettes ou types de jeux en ne gardant que les mots que sont des noms (avec les tags *NN* ou *NNP* identifiés par le *parser* du package *pattern*). Pour trouver des adjectifs pour décrire des jeux ou facettes de jeux, nous avons utilisé une approche similaire, mais en ne gardant que les adjectifs de la liste (tags JJ, JJS ou JJR). Plusieurs exemples de ces listes sont dans le fichier *word2vec.pdf*.

3.3 Mots les plus discriminants de pour les recommandations (obtenus par LDA)

Selon le nombre maximum de *features* acceptés pour le vocabulaire, il peut y avoir un important sur-apprentissage des données d'entraînement. Il est possible d'obtenir une précision d'environ 95% en acceptant tous les *features* possibles dans le corpus d'entraînement (voir figure 3), mais la précision sur les ensemble de validation est mauvaise comparativement à une limite de 1000 *features* pour définir le vocabulaire 3. Nous souhaitons obtenir un modèle qui généraliserais le mieux possible, donc qui optimise la performance de validation. En sur-apprentissage, les *features* les plus importants seraient moins significatifs.

3.3.1 SMOTE : Synthetic Minority Over-sampling Technique

Sans la normalisation, la LDA ne prédisait aucun commentaire négatif (négatif étant non-recommandé) et les résultats semblaient nettement moins bons (analyse qualitative), c'est-à-dire que les mots les

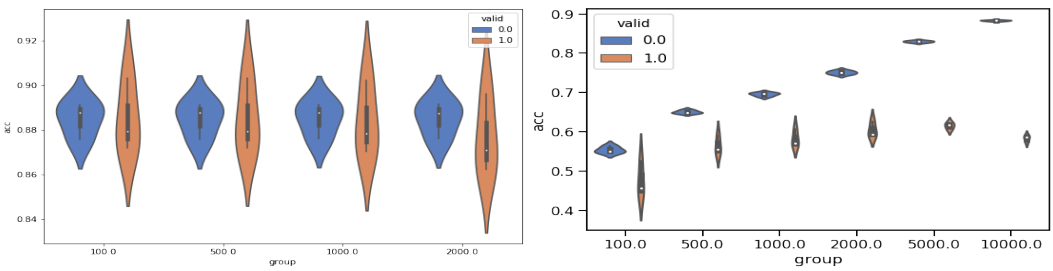


FIGURE 3 – (À gauche) Analyse Discriminante Linéaire. (À droite) Analyse Discriminante Linéaire avec SMOTE (Technique de suréchantillonnage des minorités synthétiques (en anglais *Synthetic Minority Over-sampling Technique*. *acc* signifie *accuracy* et les groupes sont les différentes valeurs de *max_features* utilisés comme paramètre de la classe *CountVectorizer* de sklearn. Les *violinplots* en bleus représentesnt les groupent d'entraînements divisés en 3 validations croisées et en orange les groupes de validation. Un nombre trop grand de *max_features* amène un sur-apprentissage des données d'entraînement.

plus associés à des commentaires négatifs ou positifs avaient moins de sens qu'en effectuant un balancement artificiel des données avec la méthode SMOTE (5), car le balancement des données "forcent" le modèle à prédire des commentaires négatifs pour optimiser la précision. Il semble toutefois que les résultats obtenus suite à la normalisation des commentaires aient rendu le balancement des classes par la méthode SMOTE moins nécessaire ; les résultats sont très similaires. Pour vérifier quelle approche est la meilleure, il faudrait mettre en place un ensemble de test qui ne soit pas ré-échantillonné avec SMOTE.

	coef	feature_name
783	2.437921	gud
764	1.766490	gr
1785	1.673691	timo
710	1.671006	funniest
852	1.663489	higra
38	1.645060	addicting
145	1.615116	bang
898	1.576700	ign
39	1.562537	addictive
286	1.555831	childhood
1194	1.542669	mt
172	1.515876	best
225	1.492479	bueno
37	1.461773	addicted
737	1.435622	geoty
568	1.429919	epic
142	1.412941	balls
1283	1.398303	pants
263	1.398209	cent
597	1.397673	excelente

	coef	feature_name
498	1.593756	drill
319	1.551919	common
263	1.546208	cent
327	1.495556	complaining
801	1.486227	hao
1591	1.484589	shower
1734	1.464284	target
1739	1.298526	teammates
200	1.295208	bonus
1778	1.283279	throw
335	1.166314	computers
747	1.152820	glitch
1780	1.142382	thumbs_up
162	1.134583	beast
447	1.132306	details
783	1.113419	gud
662	1.105886	finally
1238	1.103206	odd
142	1.097926	balls
1286	1.096381	parlour

	coef	feature_name
1846	-4.164489	uninstalled
1458	-3.774274	refund
1704	-3.684061	sucks
1968	-3.619872	worst
871	-3.484409	horrible
471	-3.414755	disappointing
1811	-3.399493	trash
208	-3.206663	bother
1855	-3.139005	unplayable
1356	-2.985664	poorly
725	-2.894566	garbage
1923	-2.863935	waste
248	-2.816813	cancer
205	-2.748652	boring
1754	-2.697983	terrible
1271	-2.655122	overkill
297	-2.531202	claggy
147	-2.438483	banned
374	-2.409864	crap
1512	-2.327323	ruined

	coef	feature_name
1846	-1.797779	uninstalled
1704	-1.501187	sucks
1811	-1.365430	trash
1968	-1.362284	worst
1458	-1.323910	refund
248	-1.300328	cancer
1754	-1.298848	terrible
205	-1.285051	boring
436	-1.250546	depending
1658	-1.223268	spot
1744	-1.186827	tedious
471	-1.185184	disappointing
871	-1.172877	horrible
1923	-1.171745	waste
127	-1.093633	avoid
1981	-1.086932	xp
728	-1.037402	gay
1513	-0.935125	ruins
1070	-0.907725	looked
1976	-0.902504	wtf

FIGURE 4 – Dans l'ordre, de gauche à droite : 1) Mots les plus associés à des commentaires positifs sans balancement des échantillons. 2) Mots les plus associés à des commentaires positifs avec balancement SMOTE des échantillons. 3) Mots les plus associés à des commentaires négatifs sans balancement des échantillons 2) mots les plus associés à des commentaires négatifs avec balancement SMOTE des échantillons. Les listes sont similaires, mais les coefficients sont plus équilibrés avec SMOTE. Sans SMOTE, les mots associés à des revues négatives sont plus fortement à la classification négatives que pour la classification de revues positives.

4 Discussion

En regardant une liste de potentiels nom de jeux, il nous est relativement facile de prédire ce qui est un jeu vidéo et ce qui n'en est pas un. Par exemple, *Dota 2*, *Left 4 Dead* ou *Team Fortress 2* sont instinctivement des noms qui pourraient être des jeux vidéo (il y a des exceptions, comme

100% Orange Juice). Comme suggéré par les résultats présentés dans la figure 2 et dans le fichier *word2vec.pdf*, il serait possible de fouiller dans le texte et trouver les mots qui ont les plus grandes possibilités d'être un nom de jeu. Certains jeux ont des noms ambiguë, mais dans le contexte il devient évident qu'il s'agit d'un jeu. Cela nécessite d'obtenir d'abord une liste de jeu fiable. Selon notre objectif, nous n'avons pas accès à une telle liste d'avance, il s'agirait alors d'une étape ultérieure à développer. Cependant, il faut noter qu'il devient évident qu'un terme comme *100% Orange Juice* est le nom d'un jeu dans son contexte car nous savons que tous les commentaires sont reliés aux jeux videos de la plateforme *steam*. Le modèle que nous apprendrions pourrait bien généraliser sur des nouvelles données, tant que les documents sont reliés aux jeux vidéos. Pour obtenir des prédictions qui soient précises dans n'importe quel contexte, l'ajout de documents de divers autres contextes que les jeux vidéos devrait être bénéfique. Il serait possible de faire un modèle pour classifier le document dans la catégorie jeu vidéo (previent de australian_user_reviews.json) ou autre (provenant d'un autre ensemble de texte sur des sujets divers). Pour les données sur les sujets divers, il serait important de savoir s'il y a des documents sur les jeux vidéos pour ne pas entraîner le modèle sur un nombre inconnu de faux-négatifs (les documents d'entraînement catégorisés comme d'un autre sujet mais qui sont en réalité à propos de jeux). Malgré qu'il y ait beaucoup moins de commentaires négatifs, il semble qu'il soit toujours plus facile de détecter des mots qui prédisent à eux seuls si un commentaire est pour une recommandation ou non. Ceci est peut-être dû à un langage plus pauvre ou plus prévisible contenu dans les commentaires négatifs.

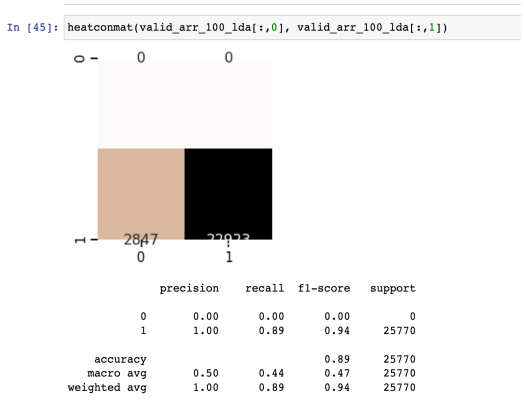
5 Conclusion

Les informations obtenues avec nos méthodes d'extraction d'information ne sont pas parfaites. Il est par exemple difficile d'obtenir une liste ne contenant que des noms de jeux vidéos, mais dans le calepin *Jupyter* intitulé *word2vec.pdf* (disponible sur le répertoire *git*) et dans la figure **??**, nous démontrons que les listes obtenus avec le plongements de mots peuvent être triées pour ne contenir presque que des noms de jeux vidéos qui vont au-delà de ceux contenus dans le corpus d'entraînement. Nous sommes impressionné par la quantité d'information que nous avons réussi à apprendre grâce à notre approche. Par exemple, le mot le plus "positif" est souvent *john*, ce qui paraît à prime abord étrange comparativement aux autres mots de la liste (figure 7), mais une lecteur des commentaires ont rapidement mis en lumière que les commentaires étaient (*spoiler alert!*) à propos de la fin d'un jeu de *Batman* où il était dévoilé que le meurtrier des parents de Bruce Wayne s'appelle John Cena, qui est le nom d'un lutteur professionnel qui se serait déjà comparé à Batman, ce qui a créé une hystérie (très positive) chez plusieurs utilisateurs. Cet exemple mets en lumière le succès auprès des joueurs d'utiliser des références à la culture populaire qui sont humoristiques dans les jeux vidéos. Ceci n'est qu'un seul exemple d'une quantité d'informations fiables qui peuvent être rapidement obtenus à propos d'un sujet vaste sans même avoir de connaissances préalables sur le sujet une fois qu'une fouille de données efficace automatique est mise en place. Notre code source pourrait, selon nous, être facilement utilisé pour analyser d'autres documents avec quelques modifications seulement (principalement pour remplacer la partie où nous avons trouvés quelques noms de jeux par expression régulière et trouver le reste des informations de façon automatisée).

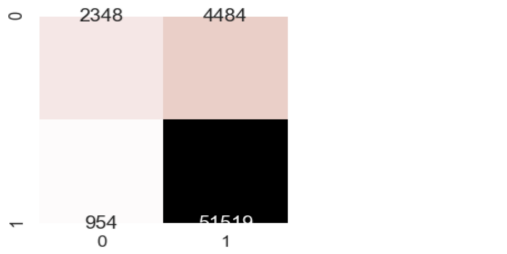
6 Annexe

6.1 Influence du débalancement des données sur la précision de commentaires positifs d'un jeu

La figure 5 démontre qu'il ne faut pas que s'intéresser à obtenir une bonne précision sur la prédiction des classes, car la classification peut être bonne sans apprendre aucun paramètre intéressant en classifiant tous les jeux comme positifs, ce qui obtiendrais une mauvaise validation si l'ensemble de validation est bien balancée (serait 50% au lieu de 88.5% si le modèle appris est celui en *a*).



	precision	recall	f1-score	support
0	0.00	0.00	0.00	0
1	1.00	0.89	0.94	25770
accuracy			0.89	25770
macro avg	0.50	0.44	0.47	25770
weighted avg	1.00	0.89	0.94	25770



	precision	recall	f1-score	support
0	0.71	0.34	0.46	6832
1	0.92	0.98	0.95	52473
accuracy			0.91	59305
macro avg	0.82	0.66	0.71	59305
weighted avg	0.90	0.91	0.89	59305

FIGURE 5 – Matrices de confusion des Analyses Discriminantes Linéaires. (À gauche) Résultats sans SMOTE avec seulement 100 features. La précision finale est bonne, mais le modèle ne fait que classifier dans la classe la plus représentée dans le données. (À droite) Résultats en utilisant un modèle appris suite à une rectification de type SMOTE (les résultats dans la matrice ne sont pas augmentées pour pouvoir comparer avec les résultats sans SMOTE). Plusieurs commentaires négatifs sont identifiés, bien que la précision de classification de ce groupe n'est pas très bonne

```
# Strangely, Dong_Dong is king
tops_games['Super_Meat_Boy'][:
```

	adjectives	similarity
0	Ben_Hur_proportions	0.469
1	Mortal_Combat	0.468
2	BY_GAIL_WOOD	0.449
3	Milli_Vanilli_Orlowski	0.446
4	Cold_Hearted	0.444
5	Mario_Kart	0.441
6	Dong_Dong	0.441
7	Quack_Quack_Quack	0.44
8	grit_Laimbeer	0.438
9	BY_TRACEY_PRISK	0.437

```
tops["Super_Meat_Boy"][:20]
```

	adjectives	similarity
0	noemon	0.486
1	owena	0.464
2	uncharted	0.449
3	stochmal	0.447
4	dareful	0.44
5	virous	0.439
6	appassionate	0.438
7	goustrous	0.437
8	frousty	0.435
9	meinong	0.435
10	cachot	0.434
11	leal	0.43
12	barkan	0.429
13	dejeune	0.428
14	cohune	0.427
15	lavic	0.427
16	libeler	0.426
17	geodic	0.425
18	cereous	0.424
19	myelonic	0.423

FIGURE 6 – (À gauche) Noms associés au jeu Super Meat Boy (tentative moins bien réussi qu'avec Borderlands dans la figure 2 de détecter des noms de jeux). (À droite) Adjectifs qui décrivent le jeu Super Meat Boy

```
tops_nn['soundtrack'][:10]
```

	word	similarity
0	music	0.577
1	vocals	0.515
2	artwork	0.488
3	storyline	0.475
4	atmosphere	0.467
5	falsetto_vocals	0.467
6	cinematography	0.465
7	inventive_choreography	0.461
8	lush_harmonies	0.46
9	ambient_textures	0.459
10	moody_atmospherics	0.459
11	sountrack	0.456
12	acoustic_guitar_riff	0.454
13	layered_vocals	0.454
14	crashing_cymbals	0.454
15	acoustic_instrumentation	0.453
16	falsetto_harmonies	0.453
17	pixelovej	0.45
18	lush_orchestration	0.449
19	composer	0.449

```
tops_nn['gametypes'][:30]
```

	word	similarity
0	gametype	0.771
1	deathmatch_mode	0.689
2	loadouts	0.678
3	Deathmatch_Team_Deathmatch	0.676
4	mutators	0.669
5	mulitplayer	0.668
6	deathmatches	0.661
7	killstreaks	0.656
8	playstyle	0.655
9	playstyles	0.654
10	Singleplayer	0.654
11	multiplayer_modes	0.652
12	gameplay	0.644
13	gameplay_tweaks	0.644
14	multiplayer	0.642
15	Gameplay_wise	0.642
16	multiplayer_mode	0.639
17	PvP_arenas	0.637
18	Achievements_Trophies	0.637
19	powerups	0.635

```
tops["gameplay"][:30]
```

	adjectives	similarity
0	gameplay_mechanic	0.746
1	replayable	0.654
2	minigame	0.653
3	overworld_map	0.645
4	sidescroller	0.631
5	side_scroller	0.629
6	fast_paced_arcade	0.624
7	metagame	0.608
8	side_scrolling_platformer	0.608
9	highly_replayable	0.607
10	frag_fest	0.598
11	microgame	0.592
12	infinitely_replayable	0.592
13	ingame	0.591
14	videogame	0.587
15	fragfest	0.569

```
tops_games['Borderlands'][:50]
```

	adjectives	similarity
0	Alien_Breed	0.663
1	Darksiders	0.658
2	GTAIV	0.655
3	BioShock	0.645
4	GRAW	0.637
5	Assassin_Creed_Brotherhood	0.636
6	ArmA_II	0.636
7	XBLA	0.635
8	Zeno_Clash	0.635
9	Bulletstorm	0.634
10	General_Knoxx	0.633
11	Killzone	0.633
12	Red_Faction_Armageddon	0.629
13	Undead_Nightmare	0.628
14	Geometry_Wars_Galaxies	0.626
15	Assault_Heroes	0.625

FIGURE 7 – Dans l'ordre, de gauche à droite : 1) Identification de facettes des jeux qui sont proches de la facette *soundtrack*. 2) Identification de plusieurs types de jeux qui sont proches de la facette *mode*. 3) Identification d'adjectifs les plus proches de la facette *gameplay*. 4) Tentative d'identifiés les jeux les plus près de Borderlands_2. Plusieurs autres listes sont retrouvées dans le document *word2vec.pdf*

6.2 Clustering de mots

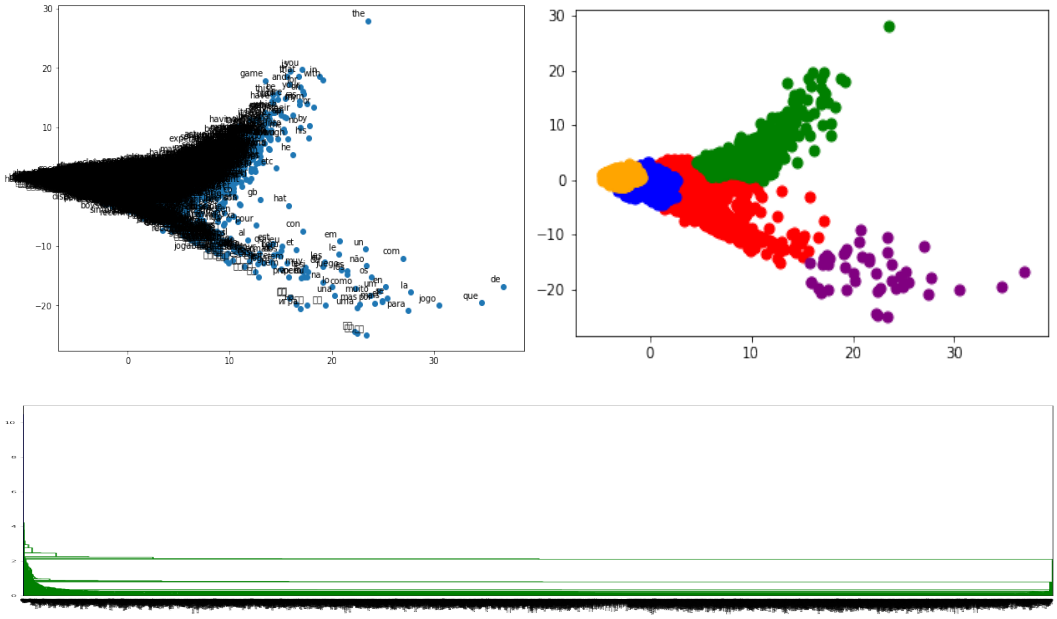


FIGURE 8 – (En haut à gauche) PCA d'une sous-ensemble de mots du corpus d'entraînement. (À droite) Clustering basé sur la similarité des mots. (En bas) Clustering hierarchique du même sous-ensemble de mots