TRAVAIL PRATIQUE #3 - CLASSIFICATION DE TEXTES ET ANALYSE DE SENTIMENTS DANS LES CONVERSATIONS

 Lien GitHub

William Bourget
111 129 490
Samuel Lévesque
111 127 772

Pour le cours IFT-7022
Techniques et applications du traitement de la langue naturelle

Présenté le 14 décembre 2018 au professeur

Luc Lamontagne

Département d'informatique et de génie logiciel
Faculté des sciences et de génie
Université Laval

Faculté des sciences et de génie
Université Laval
Automne 2018

Table des matières

1 Introduction **2**

2 Analyse préliminaire des données **2**

2.1 Analyse des *emojis* et des binettes 2

2.2 Choix des variables à ajouter 5

3 Procédure et méthodologie **6**

3.1 Corpus et objets de compte 6

3.2 Modèles de classification 7

3.3 Optimisation de paramètres 8

3.4 Test du modèle 9

4 Résultats et conclusions **9**

4.1 Simulation de prédictions 9

4.2 Prédictions sur le jeu de test 12

5 Améliorations et retrospective sur le projet réalisé **13**

5.1 Améliorations possibles 13

5.2 Apprentissage 13

6 Annexe **14**

6.1 Liste de scores d'optimisation 14

1 Introduction

Ce projet porte sur la seconde suggestion de travail proposée sur le site de cours : *Classification de texte et analyse de sentiments dans les conversations*. Le but consiste à prédire le sentiment (happy, angry, sad ou others) dégagé dans un échange de 3 textos. Un corpus d'entraînement est fourni pour réaliser la tâche.

Le code développé pour résoudre cette problématique est constitué de plusieurs fichiers. Le plus important est celui nommé *Main.py* dans le dossier *python*. C'est le seul que l'on ait besoin de rouler. Différentes options d'exécution sont offertes :

```
############################ Options d'exécution du code ######################
bool_faire_analyse_donnees_preliminaire = True  # Environ 240 secondes
bool_faire_analyse_rapide_modeles = True  # Environ 200 secondes
bool_faire_longue_optimisation = True  # Environ 2h30
bool_faire_test_meilleur_model = True  # Environ 240 secondes
bool_faire_prediction = True  # Environ 260 secondes
bool_print_tous_models_optimisation = True  # Rapide
bool_ajouter_autres_features = True  # Augmente considérablement les délais .55 secondes
↪  passe à 240 secondes
```

`bool_faire_analyse_donnees_preliminaire = True` permet de faire l'analyse préliminaire des données présentée dans la section Analyse préliminaire des données.

`bool_faire_analyse_rapide_modeles = True` permet de faire l'analyse présentée dans la sous-section Modèles de classification.

`bool_faire_longue_optimisation = True` permet de compléter l'optimisation présentée dans la sous section Modèles de classification.

`bool_faire_test_meilleur_model = True` permet d'obtenir les résultats présentés dans la sous-section Simulation de prédictions.

`bool_faire_prediction = True` permet de faire les prédictions présentées dans la sous-section Prédictions sur le jeu de test.

`bool_print_tous_models_optimisation = True` permet d'afficher tous les résultats possibles de modèles créés dans la sous-section Optimisation de paramètres.

`bool_ajouter_autres_features = True` permet de transformer les *emojis* en texte et d'ajouter les attributs supplémentaires présentés à la section Analyse préliminaire des données.

2 Analyse préliminaire des données

2.1 Analyse des *emojis* et des binettes

En regardant rapidement les données fournies dans le fichier *train.txt*, on peut voir que les échanges de textos sont très courts. Également, il y a beaucoup d'*emojis* et de binettes créés à partir de caractères spéciaux (ex : :), :D, :(, :-)).

On peut regarder le nombre de textes avec la présence d'au moins un emoji particulier selon chaque classe pour voir si ceux-ci semblent être beaucoup plus présents dans une classe ou une autre. Les résultats obtenus

Table 1 – Comptes préliminaires d'emojis

Emoji testé	Angry	Happy	Sad	Others
❤️	8	31	13	41
💔	2	0	64	3
😍	10	88	10	92
😞	15	1	282	15
😂	59	989	49	197
😡	156	6	18	10

pour quelques emojis testés se trouvent dans la table 1

À première vue, les emojis semblent avoir un impact important. Par exemple, le coeur brisé et le bonhomme triste sont souvent utilisés dans des échanges de textos tristes alors que celui qui fait un sourire est utilisé pour les échanges neutres ou joyeux. Il semble donc pertinent de convertir tous les emojis en texte pour les traiter avec un objet de compte de mots.

On peut également tenter de repérer les binettes créées à partir de caractères spéciaux. Pour parvenir à les identifier, on utilise une expression régulière qui identifie les séries de 2 à 6 caractères spéciaux. On voit ici ladite expression régulière avec la liste des chaînes de caractères ainsi extraites.

```
reg ex : [^\w\s\d]{2,6}
```


['!!!', '!!!!', '!!!!!', '!!!!!!', '!!!!!!!', '!!!!!!:', '"!!!!!"', '!!!!!:', '!!!!"', '!!!!:', '!!!":', '!!!(:', '!!!:', '!!!?', '!!!??', '!!"', '!!"!!!', '"!"', '!!(:', '!!)', '!!#', '!!,', '!!.!:', '!!...', '!!....', '!!.....', '!!.....', '!!:', '!!:)', '!!:)"']
['!!>', '!!?', '!!?!', '!!?!?', '!!?!?!?', '!!???', '"!!!!', '"(!', '"!)', '"!,', '"-"!', '"!.', '"!...', '"!:', '"!>', '"!>:', '"!>:(</', '"!?', '"!\\', '"#¢-₫', '"#¹', '"#;¢:#', '"#[', '"¢¢¢¢¢', '"$.', '"¢#', '"¢¢', '"¢¢¢', '"!"', '"\'"']
['"!)", '"!)'.", "!,", "!-'", "!-'/", "!.", "!.'", "!...!!", "!:", "!?", "!?!", "'(!:", '(()', '(-', '(.', '(:', '(;', '(=', '(?)', '(^', '(^.^)', '(^0^)', '(^・ω・^)', '(^◆^)', '(ಠ∀ಠ)', '(-', '(・', '(◆:◆)', '))', '))))']
[')),', '),.', ')//(', ');', "):", '):', '):::', '):::::', '):::::::', '):::::::::', '),', '),', '),', '),,', '),,,', '),,,,', '),,,,,', '),,,,,,', '),??']
[',.', ',,,,,', ',,,,,,', ',...', ',,...', ',,,,,', ',,,,,)', ',,:', ',?', ',?:', ',??', ',????', '-"', '-$', '-)', '--', '---!!!', '------', '------>', '--->', '--->>>', '-->', '-->(', '-.-', "-.-'", '->', '->?', '-!', '.!', '.!!']
['.!!!!', '.!!!!!', '."', '."-', '.".', '.")', '.)', '.:', '.,', '.,,', '.,,,', '.-', '.-.', '..', '...', '...!', '...!!!', '.....', '.....!', '.....!!', '.....!!!', '....!', '....!?', '....", '....$', '....']
['....(', '....', '....', '....-:)', '.....', '.....!', '.....!!', '.....!.', '.....', '.....', '.....', '.....!', '.....', '.....', '.....?', '.....', '.....)', '.....:-', '.....>', '.....?', '.....??', '.....\\', '.....', '.....(', '.....)'
['....?', '....??', '....???', '....?»', '../??', '...', '...(', '...)', '...-', '...-)', '...:/', '...:)', '...)', '..<', '..:', '..?', '..?!', '..?....', '..?/', '..??', '..»', '../', '..:', '..:)', '.<', '.:', '.?', '.?.', '.?:', '.??', '.???', '.\\"', '.]', '/"']
['//(', '//.', '/....', '///', '////', '///:', '/:', '/?', '/???', '!:', ':"', ':"(', ':")', ':$', ':'", ':'(", ':')", ':'-)", ':'-(", ':'-)", ':(', ':("', ':(((', ':(.', ':(:(:(', ':(:(', ':(:(;(;(', ':);', ':)!', ':)!!']
[':)"', ':))', ':)))', ':)):)', ':)):)))', ':):-', ':).', ':)..', ':)....', ':)?', ':)"', ':)"', ':)"', ':)*', ':)-', ':)-((', ':)-(!', ':)-(!!!', ':)-)', ':)-)))', ':)-)))', ':)-.', ':)-):-)', ':)-^', ':-----', ':)-/', ':)-[', ':)-[]', ':)-\\']
[':..', ':...', ':....', ':.....', ':......', ':;', ':;/', ':;/.', ':;//', ':;://', ':;<', ':;<(', ':;=', ':;?', ':;??', ':;@', ':;[', ':;\\', ':;\\/', ':;]', ':;^)', ':;^^', ':;!', ':;)', ':;-', ':;-(', ':;-)', ':;-/', ':;-[']
[':;;;;;;;', ':; ', ':; :', ':; ?', ':;")', ':;(', ':;(;(', ':;(;(;(', ':;)', ':;))', ':;)))', ':;):)', ':;):)', ':;)?', ':;-)', ':;-):-)', ':;-:', ':;-]', ':;...', ':;/', ':;;', ':;;;;;', ':;]', ':;^)', ':>:;', ':;-)', ':;-]', '<-', '<--', '</', '=""']
['=:(', '=(', '=)', '=))', '=/', '=<', '===>>>', '==>', '=\\', '=]', '=-', '>"', '>.', '>.<', '>.>', '>:', '>:(', '>:/', '>:[', '>:\\', '>:(', '>\', '>\/', '>>', '>>!', '>>>', '>>>!', '>?', '?!', '?!!', '?!!!', '?!!!!']
['?!!:', '?!!?!?', '?!"', '?!:', '?!?', '?!?!', '?!?!?!', '?!?!?!?!', '?!?!?!?', '?"', '?!"', '?(:', '?)', '?*', '?,', '?.', '?."', '?...', '?....', '?.....', '?/', '?:', '?:(', '?:)', '?:)"', '?:))', '?:)', '??', '???!']
['???!!!', '???!', '??..', '??....', '???/', '??:', '???', '????', '?????', '??????', '???????', '?\\', '?\\"', '?\\".', '?]', '?^?', '?»', '?¿', '?¿?¿', '?■', '¶!!', '¶¢$&¿', '[](/', '\\"', '\\",', '\\".']
['\\"', '\\"?', '\\"????', '\\"', '\\".', '\\\', '\\\\"', ']', '^)', '^-^', '^.^', '^:', '^]¶', '^^^', '^^/', '^^:', '^^^^', '^^^^^^', '^^^^^^^', '^»^', '|-|', '|||', ']?', '»?', '»»', '»»»»»', '¡!!!!!', '¬¬', '¶[^', '¿?']
['¿??', '¡¿', 'ᵔᵔ', 'ᵔᵔ', "'-'", '•-•', '•••', '•••••', '₪', '₩₩₩₩', '•`)', '\\ue404\\ue404', '•`)', '•`)', '•`)']

À partir de cela, on peut tenter d'identifier toutes les chaînes qui semblent intéressantes de façon manuelle. Pour voir rapidement si ces chaînes de caractères pourraient être discriminantes, on fait une analyse des comptes selon nos quatre classes. La table 2 présente les résultats les plus intéressants.

Table 2 – Comptes des chaînes de caractères spéciaux

Chaîne de caractères	Angry	Happy	Sad	Others
:)	140	254	191	610
:D	43	104	44	166
;)	34	66	28	121
=)	1	4	0	10

Table 2 – Comptes des chaînes de caractères spéciaux

Chaîne de caractères	Angry	Happy	Sad	Others
:-)	22	50	27	74
;-)	2	13	6	14
:(	71	18	348	101
;(	3	0	24	3
:'(	7	3	29	6
:/	40	13	33	49
:-/	6	2	1	4
:-(	5	0	21	6
Série de .	469	410	459	1085
Série de !	90	63	68	162
Série de ?	54	22	64	180
Série de ! ou ?	157	89	148	359

On peut voir que les smileys qui ont une apparence plus joyeuse (ex : :), :D) sont souvent utilisés dans des textos joyeux ou autres. Pour ce qui est de ceux avec une apparence triste, on remarque qu'ils sont souvent dans les textos tristes et parfois ceux fâchés.

On peut également analyser la ponctuation, plus particulièrement les séries de 3 signes de ponctuation et plus (ex : ..., ! ! ! ! !, ? ? ? ?, ? ! ? ! ? ? !). Les séries de points ne semblent pas se trouver en plus grande fréquence dans une classe en particulier. Pour ce qui est des séries de ! ou ?, on remarque qu'elles sont plus présentes dans les textos fâchés et triste.

On peut également s'attarder au pourcentage de lettres en majuscule dans un texto, ainsi que le score de sentiment, le nombre de mots positifs et négatifs. Les résultats obtenus pour chacun de ces attributs sont présentés dans la table 3.

Table 3 – Analyse des attributs de sentiments

Attribut	Angry	Happy	Sad	Others
Moyenne de lettres en majuscule	0,078271	0,079926	0,076783	0,082716
Moyenne de score de sentiment	-0,272566	0,417544	-0,201983	0,077257
Moyenne de nombre de mots positifs	0,594987	1,181475	0,786198	0,698689
Moyenne de nombre de mots négatifs	1,054668	0,538770	1,068094	0,523013

Le score de sentiment utilisé est calculé avec *Senti Word Net* en sommant les valeurs de sentiment attribuées à chaque mot de l'échange de textos. Le nombre de mots positifs/négatifs est calculé en comptant le nombre de mots avec des valeurs de sentiment supérieur/inférieur à 0.

On constate que le pourcentage de lettre majuscules dans les textos est similaire pour toutes les classes. On aurait pu penser que cette mesure aurait été plus élevée pour les textos fâchés, mais ça ne semble pas être le cas.

Les 3 variables associées aux sentiments semblent beaucoup parler. En effet, les catégories triste et fâché ont des connotations beaucoup plus négatives, alors que la classe joyeuse a beaucoup plus de mots positifs. La classe *Others* quand à elle a un score presque égal à 0. Ces résultats confirment notre intuition quant à l'utilisation de cet attribut.

2.2 Choix des variables à ajouter

On peut maintenant décider quelles variables supplémentaires nous ajouterons dans notre modèle. La sélection des variables s'est faite de façon assez subjective en analysant les comptes des attributs présentés dans la section Analyse préliminaire des données et en ajoutant les attributs qui semblaient avoir le plus grand pouvoir discriminant. Voici la liste des variables ajoutées :

```
add_sentiment_features(data_frame)

add_presence_of_characters_feature(data_frame, ":\)", "Ind :)")
add_presence_of_characters_feature(data_frame, ":D", "Ind :D")
add_presence_of_characters_feature(data_frame, ";\)", "Ind ;)")
add_presence_of_characters_feature(data_frame, ":-\)", "Ind :-)")
add_presence_of_characters_feature(data_frame, ":\)|:D|;\)|=\)|:-\)|;-\)|:'\)|:^\)|:]", "Ind smiley positif")

add_presence_of_characters_feature(data_frame, ":\(", "Ind :(")
add_presence_of_characters_feature(data_frame, ";\(", "Ind ;(")
add_presence_of_characters_feature(data_frame, ":'\(", "Ind :'(")
add_presence_of_characters_feature(data_frame, ":/", "Ind :/")
add_presence_of_characters_feature(data_frame, ":-\(", "Ind :-(")
add_presence_of_characters_feature(data_frame, ":\(|:\(|=\(|:-\(|:'\(|:^\(|:\[|:/|:-/", "Ind smiley negatif")

add_presence_of_characters_feature(data_frame, "!{3,}", "Ind serie de !")
add_presence_of_characters_feature(data_frame, "\?{3,}", "Ind serie de ?")
add_presence_of_characters_feature(data_frame, "[!\?]{3,}", "Ind serie de ! ou ?")
```

La fonction `add_sentiment_features` ajoute les 3 variables de sentiments (score de sentiment, nombre de mots positifs et nombre de mots négatifs) et la fonction `add_presence_of_characters_feature` ajoute un 1 si l'expression régulière donnée en argument est trouvée dans l'échange de textos, 0 sinon. Ainsi, on ajoute des variables de sentiments, des indicateurs de *smileys* positifs et négatifs et des indicateurs de séries de ponctuation à notre jeu de données bruts. On peut ainsi profiter des données et de l'entraînement de *Senti Word Net* pour ajouter de l'information à nos données avant d'entraîner nos modèles de classification.

En conclusion, lorsqu'on roule le code avec `bool_ajouter_autres_features=True`, on ajoute les variables décrites ci-dessus et on transforme tous les *emojis* en texte qui pourra être capté par les techniques standards de vectorisation des données textuelles.

3 Procédure et méthodologie

Pour parvenir à réaliser la tâche, on utilisera un objet de compte fourni dans la librairie *sklearn* et d'autres variables décrites dans la section Analyse préliminaire des données. Ensuite encore avec *sklearn*, nous utiliserons un modèle de classification pour prédire l'émotion dégagée dans les échanges de textos.

Pour développer et tester notre modèle, le corpus d'entraînement fourni dans *train.txt* a été séparé en corpus d'entraînement (80%) et de test(20%) pour pouvoir évaluer la performance hors-échantillon de notre modèle.

Ainsi, tous les paramètres seront optimisés par une validation croisée à K-plis sur notre corpus d'entraînement (80% des données) et on valide la performance du modèle sur le corpus de test.

Après optimisation de nos hyper-paramètres, le modèle est calibré avec l'ensemble du corpus pour mise en production.

3.1 Corpus et objets de compte

Pour créer le corpus, nous avons combiné les 3 échanges de textos en un seul texte séparé par des espaces entre chaque texto. Ainsi, cet échange :

1. Don't worry I'm girl
2. hmm how do I know if you are
3. What's ur name ?

Devient :**Don't worry I'm girl hmm how do I know if you are What's ur name ?** . Ce qui forme le premier texte de notre corpus. Le même processus est appliqué à tous les autres textes.

Pour transformer les mots de notre corpus sous une forme numérique, nous testerons 3 approches différentes : le compte de mots, la présence de mots (0 ou 1) et la valeur TF-IDF (term frequency-inverse document frequency). Pour chacune d'entre elles, nous retirerons les mots outils (*stop words*). Les paramètres qui seront modifiés pour ces méthodes de vectorisation des textes sont la longueur des n-grammes et la fréquence minimale d'un n-gramme pour être conservé dans le modèle.

Il est noté qu'on conservera toutes les classes de mots et aucun stemming ne sera appliqué étant donné la nature des textes avec lesquels on travaille :

```
Bad Bad bad! That's the bad kind of bad.    I have no gf    sad
Ok get it..... I made it an option Ok  others
Money money and lots of money😍😍   I need to get it tailored but I'm in love with it 😍😂😂 happy
My gf left ne   Get over it. Go out with someone else.  Me* sad
get lost    I know you guys want to loose to me always. I don't want to talk u any more angry
You are lying and i know that   I KNOW YOU'RE LYING, AB BYS 😔😔 sad
Ur creator is very bad  you are only the creator of your brain. 😑  sad
ehat is hehe    Haha is more like:Hehe is more of a giggle. ;^) what*   others
```

En effet, on peut voir que certains mots sont inventés, d'autres mal écrits et certains sont des abbréviations. De plus, on voit que certains mots ne sont pas bien séparés par des espaces ce qui fait en sorte que ces algorithmes ne parviendraient pas bien à les séparer. Également, on souhaite limiter le temps de calcul pour l'optimisation de nos hyper-paramètres qui est déjà assez important.

Pour ces raisons, nous avons décidé de ne pas faire de stemming ou de lemmatisation sur nos données textuelles puisque nous jugeons que ces techniques n'apporteraient pas de gains de performance significatifs.

En conclusion, la matrice de données utilisée pour nos modèles de classification sera composée de données d'un des 3 objets de compte décrits dans cette sous-section avec ou sans les attributs supplémentaires décrits dans la section Analyse préliminaire des données (si `bool_ajouter_autres_features=True`).

3.2 Modèles de classification

4 modèles de classification seront testés : SVM, K-PPV, Régression Logistique et MLP. Pour débuter, on peut tester ces modèles avec des paramètres déterminés de façon arbitraire. Pour évaluer le score de performance, on utilise la métrique qui sert à l'évaluation du concours :

> Evaluation will be done by calculating microaveraged F1 score (F1μ) for the three emotion classes i.e. Happy, Sad and Angry on the submissions made with predicted class of each sample in the Test set. To be precise, we define the scoring as following:
>
> Pμ = ΣTPi / Σ(TPi + FPi)∀i {Happy,Sad,Angry}
> Rμ = ΣTPi / Σ(TPi + FNi)∀i {Happy,Sad,Angry}
>
> where TPi is the number of samples of class i which are correctly predicted, FNi and FPi are the counts of Type-I and Type-II errors respectively for the samples of class i.
>
> Our final metric F1μ will be calculated as the harmonic mean of Pμ and Rμ.

Cette valeur basée sur le nombre de vrais positifs, de faux négatifs et de faux positifs de chaque classe sera calculée 3 fois avec une validation 3-plis sur nos données d'entraînement.

$$F1_u = \frac{2}{\frac{1}{P_u} + \frac{1}{R_u}} \tag{3.1}$$

Comme première base de comparison, on teste les modèles suivants avec nos objets de compte 1-gramme avec occurrence minimale de 40 :

SVM(Support Vector Machine) : avec paramètres par défaut.

KPPV (K plus proche voisins) : avec 5 voisins.

Régression Logistique : avec paramètres par défaut. (Un contre tous pour le cas multivarié)

MLP (Multi Layer Perceptron) : avec 2 couches cachées de 100 neurones chacune.

Les résultats ainsi obtenus sont présentés dans la figure 4.

Table 4 – Scores F1 de nos modèles de classification

		Score F1	
Vectorisation des mots	**Modèle**	**Sans ajout de features**	**Avec ajout de features**
Word Count	SVM	0,3457	0,3710
	k-PPV	0,3836	0,4456
	Régression logistique	0,5958	0,6610
	MLP	0,5440	0,6114
Word Count binaire	SVM	0,3223	0,3284
	k-PPV	0,3727	0,4040
	Régression logistique	0,6046	0,6797
	MLP	0,5428	0,6141

TF-IDF	SVM	0,0229	0,0339
	k-PPV	0,4440	0,4482
	Régression logistique	0,5833	0,6552
	MLP	0,5443	0,6115

On remarque que peu importe le modèle, l'ajout de nos attributs supplémentaires améliore assez significativement la performance du modèle. Les 3 objets de compte semblent avoir des scores similaire d'un modèle à l'autre. On ne peut pas facilement en distinguer un qui sort du lot.

Pour ce qui est des modèles de classification, le modèle de régression logistique semble mieux performer que les autres. Cela est probablement dû au fait que les autres modèles ont plus d'hyper-paramètres à optimiser pour arriver à de bons résultats contrairement à la régression logistique qui en a peu (voir pas). Nous pensons que les autres modèles profiteront davantage d'une optimisation plus globale des hyper-paramètres.

3.3 Optimisation de paramètres

Pour optimiser tous nos paramètres, autant pour les objets de compte que pour les modèles de classification, on calcule le score défini à la section précédente pour plusieurs combinaisons de paramètres possibles. Voici les paramètres choisis pour ces tests :

SVM(Support Vector Machine) : valeur de C, type de noyau (kernel)
KPPV (K plus proche voisins) : nombre de voisins k, type de poids
Régression Logistique : type de pénalité appliquée
MLP (Multi Layer Perceptron) : nombre de neurones pour chaque couche cachée.

Voici les listes des valeurs choisies pour notre recherche en grille :

```
#List de test pour objet de compte
list_n_gram=[1]
list_min_freq=[2,5,10]
list_nom_objet_compte=["Word counts","Binary Word counts","TfiDf"]

#List SVM
list_SVM_c=[0.1,1,10]
list_SVM_kernel=["rbf","linear"]

#List K-PPV
list_KPPV_k=[2,5,8,13,18,23]
list_KPPV_weight=["uniform","distance"]

#List log reg
list_LogReg_penalty=["l1","l2"]

#List MLP
list_MLP_hidden_layer_shape=[(10,),(10,10),(50,50),(100,100),(200,200),(100,100,100),(100,100,100,100),(50,50,50,50,50),(10,10,10,10,10,10,10,10)]
```

Cela fait donc $(1 * 3 * 3) * (3 * 2 + 6 * 2 + 2 + 9) = 261$ combinaisons de paramètres à tester. La combinaison ayant le meilleur score est sauvegardée dans le fichier *Dictionnaire_parametre_meilleur_model* sous la forme d'un dictionnaire contenant ses paramètres. Le dictionnaire prend la forme suivante : {nom du classificateur,

score, dictionnaire de paramètres du classificateur, nom du modèle de compte, dictionnaire de paramètres du modèle de compte}.

3.4 Test du modèle

Pour tester notre modèle, on charge le dictionnaire avec nos meilleurs paramètres contenus sur *Dictionnaire_parametre_meilleur_model*. On entraîne le modèle sur l'ensemble de notre 80% de données séparées au tout début et on prédit le 20% jamais touché lors de notre optimisation ou entraînement. Le meilleur modèle et les résultats sont présentés à la prochaine section (Résultats et conclusions).

4 Résultats et conclusions

4.1 Simulation de prédictions

Après optimisation de nos hyper-paramètres, on conclut que le modèle ayant le mieux performer en validation est un modèle avec un objet de compte binaire (0 ou 1) qui utilise des 1-gramme et qui conserve une fréquence minimale de 2 mots. Le modèle de classification utilisé est une régression logistique avec une pénalité de choix de paramètres *L1* :

```
Paramètres meilleur modèle: {'clf name': 'LogReg', 'score': 0.7616916517620598, 'Dict param clf': {'penalty': 'l1'}, 'Nom compteur': 'Binary Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
```

On peut évaluer notre modèle en calculant le score qu'on obtiendrait sur 80% de notre corpus d'entraînement qui sert d'entraînement, et 20 % qui sert de test. On entraîne donc ce modèle sur l'ensemble de nos données d'entraînement pour faire des prédictions sur notre corpus de test. On procédant ainsi, on obtient la matrice de confusion présentée dans la figure 1.

On peut voir que notre score F1 (utilisé pour l'évaluation du concours) de 0.7669 est quand même assez haut. On remarque que la plupart des erreurs faites sont de prédire *happy*, *sad* et *angry* comme étant *others*. Cette caractéristique serait peut être même avantageuse pour le concours puisque la proportion d'échanges de textos considérés comme étant *Others* est bien plus élevée dans le corpus d'évaluation du concours que dans le jeu d'entraînement fourni avec l'énoncé.

De plus, la confusion entre *sad* et *angry* est un peu plus élevée qu'entre les autres paires d'émotions puisque qu'on a pu voir que les deux sont à caractère négatif et partagent sans doute certains attributs.

On peut également observer la matrice de confusion lorsque l'on n'ajoute pas les variables d'emoji et de binettes décrites dans la section Analyse préliminaire des données :

On voit tout de suite que le score est beaucoup moins bon (0.7095 au lieu de 0.7669). On remarque aussi que beaucoup plus de classes *happy* et *sad* ont été prédites comme étant *others* (768 à 1273 et 1058 à 1503). Pour ce qui est des données de classe *angry*, les nombres changent très peu, on peut supposer qu'il y a très peu d'emojis et de binettes dans notre corpus de test pour la classe *angry*.

Encore une fois, on peut supposer que notre modèle avec les variables ajoutées est meilleur que celui sans, à cause du meilleur score F1. On peut également comparer notre score aux meilleurs scores soumis à la compétition :

FIGURE 1 – Matrice de confusion pour paramètres complets

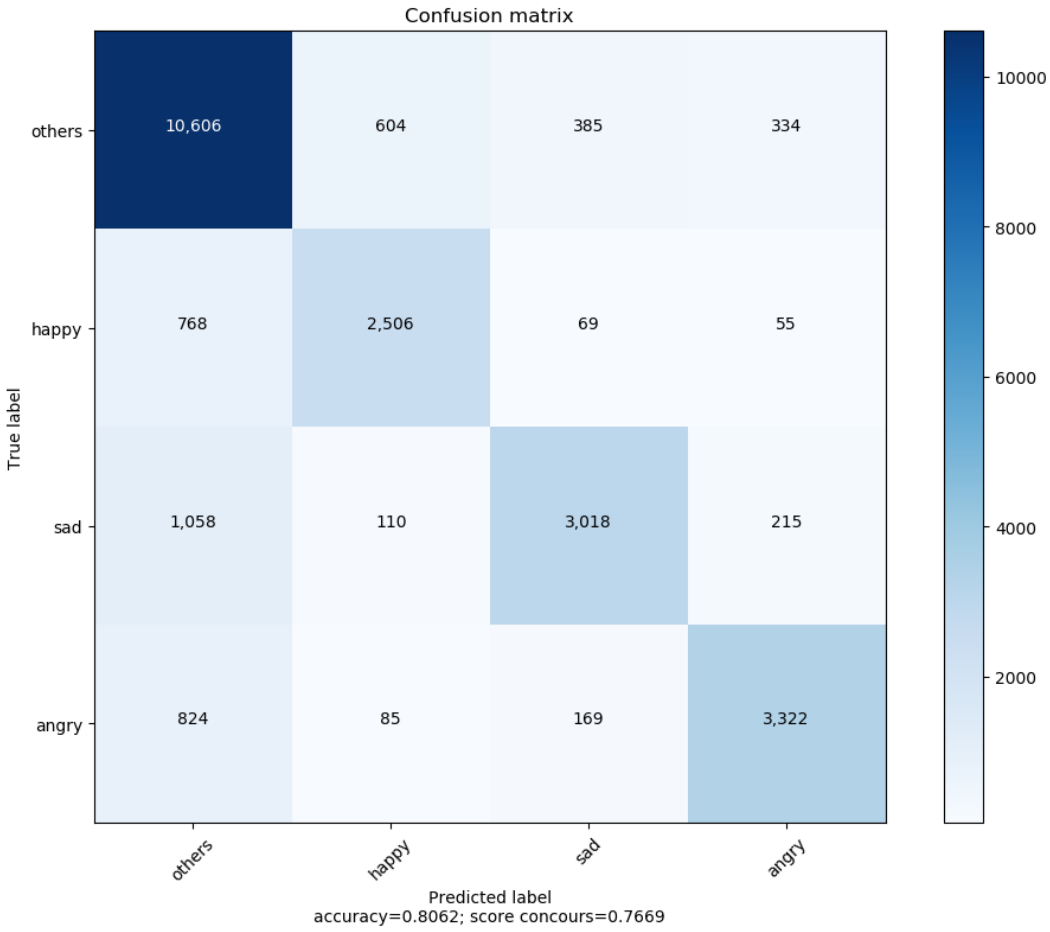
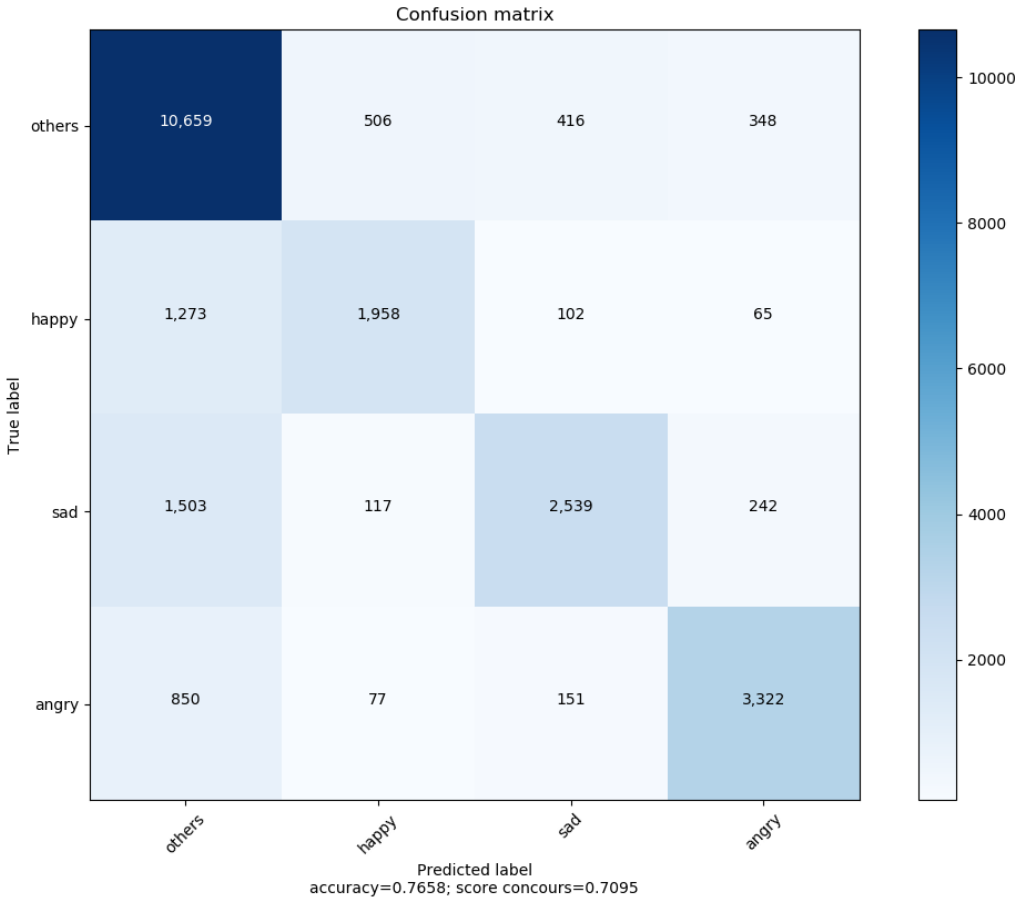


FIGURE 2 – Matrice de confusion sans paramètres additionnels



#	Username	Score
1	parag11	0.7482
2	zhaopku	0.7475
3	soujanyaporia	0.7429

On peut conclure qu'un score de 0.7669 en simulation de test est donc très bon. Il est toutefois difficile de bien comparer notre modèle à ceux qui ont été soumis puisque la répartition des classes n'est pas la même dans le corpus à prédire. Le fichier de test que nous avons utilisé avait environ 16.6% de *happy*, *sad*, *angry* et 50% de *others*, alors que le fichier à prédire a des proportions de 4% pour chacune des trois classes minoritaires et 88% de *other*.

On peut quand même être réellement satisfait du résultat obtenu par notre approche, surtout en considérant que la majorité de nos erreurs consistaient à prédire *others* alors que nous faisions face à un échange avec un autre sentiment.

4.2 Prédictions sur le jeu de test

Dans cette section, on utilise notre modèle entraîné sur la totalité des données d'entraînement fournies avec l'énoncé et on l'applique sur les données de test non-étiquetées fournies avec l'énoncé. On présente ici les 30 premières prédictions ainsi obtenues.

	Corpus test	Label prédit
0	Then dont ask me YOURE A GUY NOT AS IF YOU WOULD UNDERSTAND IM NOT A GUY FUCK OFF	angry
1	Mixed things such as?? the things you do. Have you seen minions??	others
2	Today I'm very happy and I'm happy for you ♥ I will be marry	happy
3	Woah bring me some left it there oops Brb	others
4	it is thooooo I said soon master. he is pressuring me	others
5	Wont u ask my age?? hey at least I age well! Can u tell me how can we get closer??	others
6	I said yes What if I told you I'm not? Go to hell	angry
7	Where I ll check why tomorrow? No I want now	others
8	Shall we meet you say- you're leaving soon...anywhere you wanna go before you head? ?	others
9	Let's change the subject I just did it .l. You're broken	sad
10	Your pic pz thank you X-D wc	others
11	not mine done for the day ? can my meet to sexy girl	others
12	I want to play the game if you just finished the game... then you haven't finished the game..... #Emojisong	others
13	Iam sory why sorry ! 😹 I insult you	others
14	How much depends on how long your internet has been out!!! U have bf	others
15	Ok Thank you. xD What about cortanan	others
16	So the story? yeah indeed 😹 Tomorrow probably	others
17	May be yeaa i hope soo!! Can you do complex calculations	others
18	So come on na.. Want u so badly. now you're tempting me to. So why are you still away from my body??	sad
19	No you aren't oh I am Really?	others
20	Are u a one I'm the normal one, if you want What I want	others
21	But... then I'm feeling nervous	others
22	Send me any video or songs Video or Text S	others
23	Why why what How r u	others
24	Do it but why? Aaah ...u lack creativity at some extent	others
25	Yeah going out with the parson outside in the city or in the parc Yeah..... I don't live in the city... X So where do you live	others
26	Then go to sleep you never sleep I will	others
27	He's very depressed and i want to help him but how should i do.....it feels very bad seeing your best friend like this Dance to enjoy, not to please. Ohhhh	sad
28	Oh great Yeah do you have any plans for today? No i don't have any plan	others
29	I don't have plans for the weekend. you did saturday with me but I see how it is What are you talking about?	others

Il est difficile d'analyser quantitativement nos prédictions puisque nous n'avons pas les vraies classes associées à chacune des observations de ce corpus.

On peut toutefois voir en regardant l'échantillon de prédictions ci-dessus qu'à chaque fois qu'on prédit happy, angry ou sad, la prédiction semble bonnes sauf pour le numéro 18 qui n'a clairement pas l'air triste. On peut supposer que cet échange cocasse est mal classé à cause de "Want u so badly". En effet, le mot *"badly"* a une forte connotation négative, ce qui induit notre modèle à classer ce texte en tant que triste.

En regardant rapidement, on peut voir que ceux classés comme *others* ont l'air particulièrement neutre, sauf peut-être le 13, mais cela est assez subjectif.

Ce petit échantillon nous laisse avec une très bonne impression de notre modèle en prédiction hors-échantillon.

5 Améliorations et retrospective sur le projet réalisé

5.1 Améliorations possibles

Plusieurs améliorations ou changements pourraient être effectués pour réaliser ce projet, en voici quelques uns :

1. Lors de la création du corpus, on considère l'échange de 3 textos comme un gros bloc de texte. Il pourrait être intéressant de tenter de trouver une relation entre les échanges qui pourrait aider à faire une meilleure classification.
2. Un gros problème du projet réalisé est le problème de *"runtime"* du code. En effet, l'ajout de variables de sentiment dans les attributs augmente terriblement le temps d'exécution du code. Il faudrait revoir en entier les fonctions qui y sont associées pour pouvoir tester plus de paramètres lors de notre recherche en grille.
3. Un travail plus exhaustif et robuste pourrait être fait sur les binettes ajoutées comme variables. On pourrait faire un modèle statistique qui analyse toutes les chaînes de caractères spéciaux et qui détermine le degré de signification de chacune pour toutes nos classes au lieu d'utiliser une méthode plus ad hoc et manuelle comme nous avons utilisée.
4. Dans l'optimisation de paramètres, on pourrait tester un nombre de n-gramme supérieur à 1. Encore une fois, cet item pourrait être adressé en optimisant le temps d'exécution du code.

5.2 Apprentissage

Ce fut un projet très intéressant qui permet d'appliquer plusieurs connaissances en traitement de langue naturelle. Il permet d'appliquer les notions de classification de textes, d'analyse de sentiment et les expressions régulières. Également, contrairement aux autres projets réalisés au cours de la session, ce projet laissait libre cours à notre imagination et nous avions la liberté d'expérimenter avec plusieurs approches pour résoudre un même problème.

L'utilisation d'emojis et de binettes pour classer les échanges de textos était également très intéressante puisqu'elle n'a pas été vue en classe, mais apporte tout de même une quantité non-négligeable d'information à notre modèle.

Le projet fut également très formateur pour les auteurs du code Python qui contrairement à leur habitude, ont tenté de faire un code plus lisible et réutilisable pour d'autres applications.

6 Annexe

6.1 Liste de scores d'optimisation

{'clf name': 'SVM', 'score': 0.0, 'Dict param clf': {'C': 0.1, 'Kernel': 'rbf'}, 'Nom compteur': 'Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'SVM', 'score': 0.6985357583653151, 'Dict param clf': {'C': 0.1, 'Kernel': 'linear'}, 'Nom compteur': 'Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'SVM', 'score': 0.015125407049193235, 'Dict param clf': {'C': 1, 'Kernel': 'rbf'}, 'Nom compteur': 'Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'SVM', 'score': 0.7203334307634582, 'Dict param clf': {'C': 1, 'Kernel': 'linear'}, 'Nom compteur': 'Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'SVM', 'score': 0.27478887848514791, 'Dict param clf': {'C': 10, 'Kernel': 'rbf'}, 'Nom compteur': 'Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'SVM', 'score': 0.6768962802374062, 'Dict param clf': {'C': 10, 'Kernel': 'linear'}, 'Nom compteur': 'Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'KPPV', 'score': 0.3614611802777718, 'Dict param clf': {'k': 2, 'Weight': 'uniform'}, 'Nom compteur': 'Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'KPPV', 'score': 0.4347727247707239, 'Dict param clf': {'k': 2, 'Weight': 'distance'}, 'Nom compteur': 'Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'KPPV', 'score': 0.41517155952314216, 'Dict param clf': {'k': 5, 'Weight': 'uniform'}, 'Nom compteur': 'Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'KPPV', 'score': 0.43916833080919765, 'Dict param clf': {'k': 5, 'Weight': 'distance'}, 'Nom compteur': 'Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'KPPV', 'score': 0.39186009589398835, 'Dict param clf': {'k': 8, 'Weight': 'uniform'}, 'Nom compteur': 'Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'KPPV', 'score': 0.4280308019194216, 'Dict param clf': {'k': 8, 'Weight': 'distance'}, 'Nom compteur': 'Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'KPPV', 'score': 0.36477669092119985, 'Dict param clf': {'k': 13, 'Weight': 'uniform'}, 'Nom compteur': 'Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'KPPV', 'score': 0.3952308730118086, 'Dict param clf': {'k': 13, 'Weight': 'distance'}, 'Nom compteur': 'Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'KPPV', 'score': 0.3485283390717491, 'Dict param clf': {'k': 18, 'Weight': 'uniform'}, 'Nom compteur': 'Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'KPPV', 'score': 0.3728407461495076, 'Dict param clf': {'k': 18, 'Weight': 'distance'}, 'Nom compteur': 'Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'KPPV', 'score': 0.33011915129228975, 'Dict param clf': {'k': 23, 'Weight': 'uniform'}, 'Nom compteur': 'Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'KPPV', 'score': 0.3497741446809415, 'Dict param clf': {'k': 23, 'Weight': 'distance'}, 'Nom compteur': 'Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'LogReg', 'score': 0.744015821660455, 'Dict param clf': {'penalty': 'l1'}, 'Nom compteur': 'Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'LogReg', 'score': 0.720583655000724, 'Dict param clf': {'penalty': 'l2'}, 'Nom compteur': 'Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'MLP', 'score': 0.6537570223803751, 'Dict param clf': {'hidden layers sizes': (10,)}, 'Nom compteur': 'Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'MLP', 'score': 0.6463201470454588, 'Dict param clf': {'hidden layers sizes': (10, 10)}, 'Nom compteur': 'Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'MLP', 'score': 0.6607520268306001, 'Dict param clf': {'hidden layers sizes': (50, 50)}, 'Nom compteur': 'Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'MLP', 'score': 0.6737236145085261, 'Dict param clf': {'hidden layers sizes': (100, 100)}, 'Nom compteur': 'Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'MLP', 'score': 0.6837936850286841, 'Dict param clf': {'hidden layers sizes': (200, 200)}, 'Nom compteur': 'Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'MLP', 'score': 0.6755009147725252, 'Dict param clf': {'hidden layers sizes': (100, 100, 100)}, 'Nom compteur': 'Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'MLP', 'score': 0.6634481402051691, 'Dict param clf': {'hidden layers sizes': (100, 100, 100, 100)}, 'Nom compteur': 'Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'MLP', 'score': 0.6654684054979634, 'Dict param clf': {'hidden layers sizes': (50, 50, 50, 50, 50)}, 'Nom compteur': 'Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'MLP', 'score': 0.5192589850460636, 'Dict param clf': {'hidden layers sizes': (10, 10, 10, 10, 10, 10, 10, 10)}, 'Nom compteur': 'Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}
{'clf name': 'SVM', 'score': 0.0, 'Dict param clf': {'C': 0.1, 'Kernel': 'rbf'}, 'Nom compteur': 'Binary Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'SVM', 'score': 0.700533768960601, 'Dict param clf': {'C': 0.1, 'Kernel': 'linear'}, 'Nom compteur': 'Binary Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'SVM', 'score': 0.0, 'Dict param clf': {'C': 1, 'Kernel': 'rbf'}, 'Nom compteur': 'Binary Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'SVM', 'score': 0.7283107011786764, 'Dict param clf': {'C': 1, 'Kernel': 'linear'}, 'Nom compteur': 'Binary Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'SVM', 'score': 0.12410864427508339, 'Dict param clf': {'C': 10, 'Kernel': 'rbf'}, 'Nom compteur': 'Binary Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'SVM', 'score': 0.6875398124345851, 'Dict param clf': {'C': 10, 'Kernel': 'linear'}, 'Nom compteur': 'Binary Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'KPPV', 'score': 0.3055257453904481, 'Dict param clf': {'k': 2, 'Weight': 'uniform'}, 'Nom compteur': 'Binary Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'KPPV', 'score': 0.39802332231743637, 'Dict param clf': {'k': 2, 'Weight': 'distance'}, 'Nom compteur': 'Binary Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'KPPV', 'score': 0.34872364479895396, 'Dict param clf': {'k': 5, 'Weight': 'uniform'}, 'Nom compteur': 'Binary Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}

Le reste des pages peuvent être "*print*" avec le code, avec `bool_print_tous_models_optimisation`