William Bourget
111 129 490

Travail 3

Cours IFT-7022

Université Laval
14 décembre 2018

1 Introduction et comment rouler le code

Ce projet porte sur la seconde suggestion de travaux 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) dégagé dans un échange de 3 textos. Un corpus d'entraînement est fourni pour réaliser la tâche.

Le code est constitué de plusieurs fichiers. Le plus important est celui nommé *Main*. C'est le seul que l'on ait besoin de rouler. Différentes options d'exécution sont offertes :

```
########################## Options de roulage de code ########################################################################
bool_faire_analyse_donnees_preliminaire=False #Environ 240 secondes
bool_faire_analyse_rapide_modeles=False #Environ 200 secondes
bool_faire_longue_optimisation=False #Environ 2h30
bool_faire_test_meilleur_model=False #Environ 240 secondes
bool_faire_prediction=False #Environ 260 secondes
bool_print_tous_models_optimisation=True #Rapide


bool_ajouter_autres_features=True #Augmente considérablement les délais 0.55 secondes passe à 240 secondes
```

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

`bool_faire_analyse_rapide_modeles = True` permet de faire l'analyse présentée dans la sous section *Modèles de classification* dans la section *Procédure et méthodologie*.

`bool_faire_longue_optimisation = True` permet de faire l'optimisation présentée dans la sous section *Modèles de classification* dans la section *Procédure et méthodologie*.

`bool_faire_test_meilleur_model = True` permet d'obtenir les résultats présentés dans la sous section *Simulation de prédictions* dans la section *Résultats et conclusions*.

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

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

`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

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, :(, :-)).

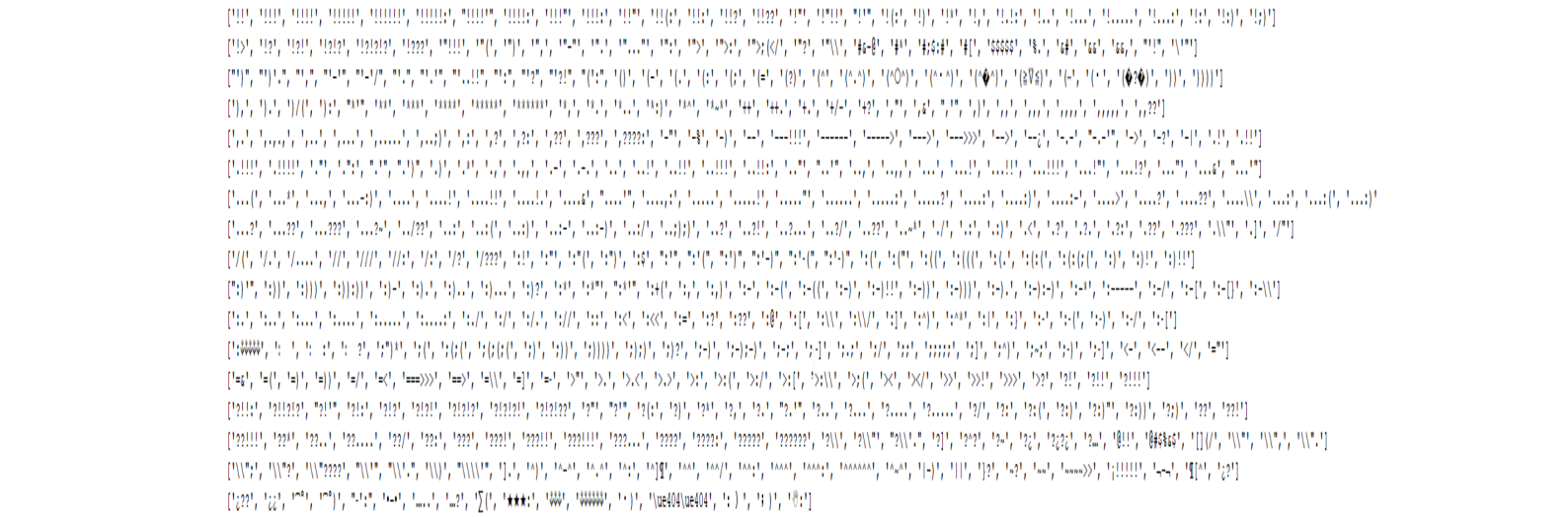
On peut regarder le nombre de textes avec la présence d'au moins un emoji particulier selon chaque classe pour voir si ceux-ci semblent être beaucoup plus présents dans une classe ou une autre :

```
Total de ❤
Label
angry      8
happy     31
others    41
sad       13
Name: Ind ❤, dtype: int32

Total de 💔
Label
angry      2
happy      0
others     3
sad       64
Name: Ind 💔, dtype: int32

Total de 😍
Label
angry     10
happy     88
others    92
sad       10
Name: Ind 😍, dtype: int32

Total de 😒
Label
angry      15
happy       1
others     15
sad       282
Name: Ind 😒, dtype: int32

Total de 😂
Label
angry      59
happy     989
others    197
sad        49
Name: Ind 😂, dtype: int32

Total de 😔
Label
angry     156
happy       6
others     10
sad        18
Name: Ind 😔, dtype: int32
```

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 créés à partir de caractères spéciaux. Pour parvenir à les identifier, on peut utiliser une expression régulière qui identifie les séries de 2 à 6 caractères spéciaux :

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



À partir de cela, on peut tenter d'identifier toutes les chaînes qui semblent intéressantes de façon manuelle. Voici une liste des plus intéressantes :

```
Total de :)
Label
angry     140
happy     254
others    610
sad       191
Name: Ind :), dtype: int32
```

```
Total de :D
Label
angry      43
happy     104
others    166
sad        44
Name: Ind :D, dtype: int32
```

```
Total de ;)
Label
angry      34
happy      66
others    121
sad        28
Name: Ind ;), dtype: int32
```

```
Total de =)
Label
angry      1
happy      4
others    10
sad        0
Name: Ind =), dtype: int32
```

```
Total de :-)
Label
angry     22
happy     50
others    74
sad       27
Name: Ind :-), dtype: int32
```

```
Total de ;-)
Label
angry      2
happy     13
others    14
sad        6
Name: Ind ;-), dtype: int32
```

```
Total de :(
Label
angry      71
happy      18
others    101
sad       348
Name: Ind :(, dtype: int32
```

```
Total de ;(
Label
angry      3
happy      0
others     3
sad       24
Name: Ind ;(, dtype: int32
```

```
Total de :'(
Label
angry      7
happy      3
others     6
sad       29
Name: Ind :'(, dtype: int32
```

```
Total de :/
Label
angry     40
happy     13
others    49
sad       33
Name: Ind :/, dtype: int32
```

```
Total de :-/
Label
angry     6
happy     2
others    4
sad       1
Name: Ind :-/, dtype: int32
```

```
Total de :-(
Label
angry      5
happy      0
others     6
sad       21
Name: Ind :-(, dtype: int32
```

```
Total de serie de .
Label
angry     469
happy     410
others   1085
sad       459
Name: Ind serie de ., dtype: int32
```

```
Total de serie de !
Label
angry      90
happy      63
others    162
sad        68
Name: Ind serie de !, dtype: int32
```

```
Total de serie de ?
Label
angry      54
happy      22
others    180
sad        64
Name: Ind serie de ?, dtype: int32
```

```
Total de serie de ! ou?
Label
angry     157
happy      89
others    359
sad       148
Name: Ind serie de ! ou ?, dtype: int32
```

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 une série de 3 signes de ponctuation et plus (ex : ...,!!!!!,????,?!?!???!). Les séries de points ne semblent pas particulièrement être plus présent dans une classe particulière. 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. Voici la liste du nombre moyen de ces valeurs pour chaque classe :

```
Moyenne de pourcentage de lettres en majuscule parmi toutes les lettres
Label
angry     0.078271
happy     0.079926
others    0.082716
sad       0.076783
Name: Pourcentage_maj, dtype: float64

Moyenne de score de sentiment
Label
angry    -0.272566
happy     0.417544
others    0.077257
sad      -0.201983
Name: Sentiment, dtype: float64

Moyenne de nombre de mots positifs
Label
angry     0.594987
happy     1.181475
others    0.698689
sad       0.786198
Name: Nombre_positive, dtype: float64

Moyenne de nombre de mots négatifs
Label
angry     1.054668
happy     0.538770
others    0.523013
sad       1.068094
Name: Nombre_negative, dtype: float64
```

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 majuscule dans les textos est similaire pour toutes les classes, on aurait pu penser qu'il aurait été plus haut pour les textos fâchés.

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. Cela n'est pas très surprenant.

Choix des variables à ajouter

On peut maintenant décider quelles variables supplémentaires nous ajouterons dans notre modèle. 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, 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, les variables de sentiments, des indicateurs de *smileys* positifs, négatifs et de séries de ponctuations sont ajoutés comme variables.

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.

3 Procédure et méthodologie

Pour parvenir à réaliser la tâche, on utilisera un objet de compte fourni dans la librairie *sklearn* avec 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 simuler un test sur nos données, le corpus d'entraînement fourni dans *train.txt* sera directement séparé en corpus d'entraînement (80%) et en corpus de test(20%) pour pouvoir évaluer la performance de notre modèle. Ainsi, tous les tests seront réalisés avec 80% des données. À la fin complètement, le meilleur modèle sera entraîné avec les données d'entraînement et de test.

Corpus et Objets de compte

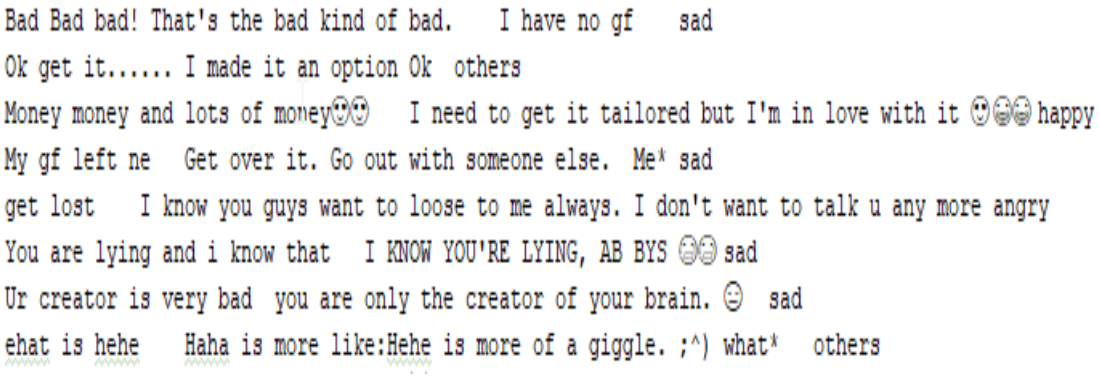
Pour créer le corpus, nous combinerons les 3 échanges de textos en un seul texte séparé par des espaces entre chaque texto. Ainsi, cet échange :

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 modèles différents : le compte de mots, la présence de mots (0 ou 1) et la valeur TF-IDF (term frequency-inverse document frequency). Pour chacun d'entre eux, nous retirerons les mots outils (*stop words*. Les seuls paramètres qui seront modifiés sont : le nombre de n gram utilisés et la fréquence minimale conservée pour chaque n gram.

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 :



On peut voir que certains mots sont inventés, d'autres mal écrits et certains sont des abréviations. Par conséquent, le stemming ou la conservation de classe ouverte seulement ne serait pas une bonne idée puisque beaucoup d'information serait perdue.

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`).

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 peut utiliser 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.

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

On peut tester les modèles suivants avec nos objets de compte 1 gram 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 chaque.

On obtient :

Modèle avec Word Count:
Score SVM: 0.3457078016427561
Score K-PPV: 0.38357444823464243
Score Régression logistique: 0.5957627470552433
Score MLP: 0.5440412380235092

Modèle avec Word Count binaire:
Score SVM: 0.32225464197678466
Score K-PPV: 0.37273463931109396
Score Régression logistique: 0.6046311590311633
Score MLP: 0.542794913738652

Modèle avec TfiDf:
Score SVM: 0.02292437792841466
Score K-PPV: 0.44401396324473247
Score Régression logistique: 0.5833009745500859
Score MLP: 0.5442725471042236

Sans ajout d'autres features

Modèle avec Word Count:
Score SVM: 0.3709688473539565
Score K-PPV: 0.4455513720931486
Score Régression logistique: 0.6610472158888931
Score MLP: 0.6114207100923366

Modèle avec Word Count binaire:
Score SVM: 0.3283604008126426
Score K-PPV: 0.40396816456799556
Score Régression logistique: 0.6797250501604125
Score MLP: 0.6141151659383984

Modèle avec TfiDf:
Score SVM: 0.03387059684120452
Score K-PPV: 0.44820677378042
Score Régression logistique: 0.6552280148829729
Score MLP: 0.6115223263349369

Avec ajout d'autres features

On remarque que peu importe le modèle, l'ajout d'autres features que celles de comptes augmente le score 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 de paramètres à optimiser pour arriver à de bons résultats contrairement à la régression logistique qui en a peu (voir pas).

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 :

```
#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 possibles à tester. La combinaison ayant le meilleur score sera sauvegardée sur 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}.

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.

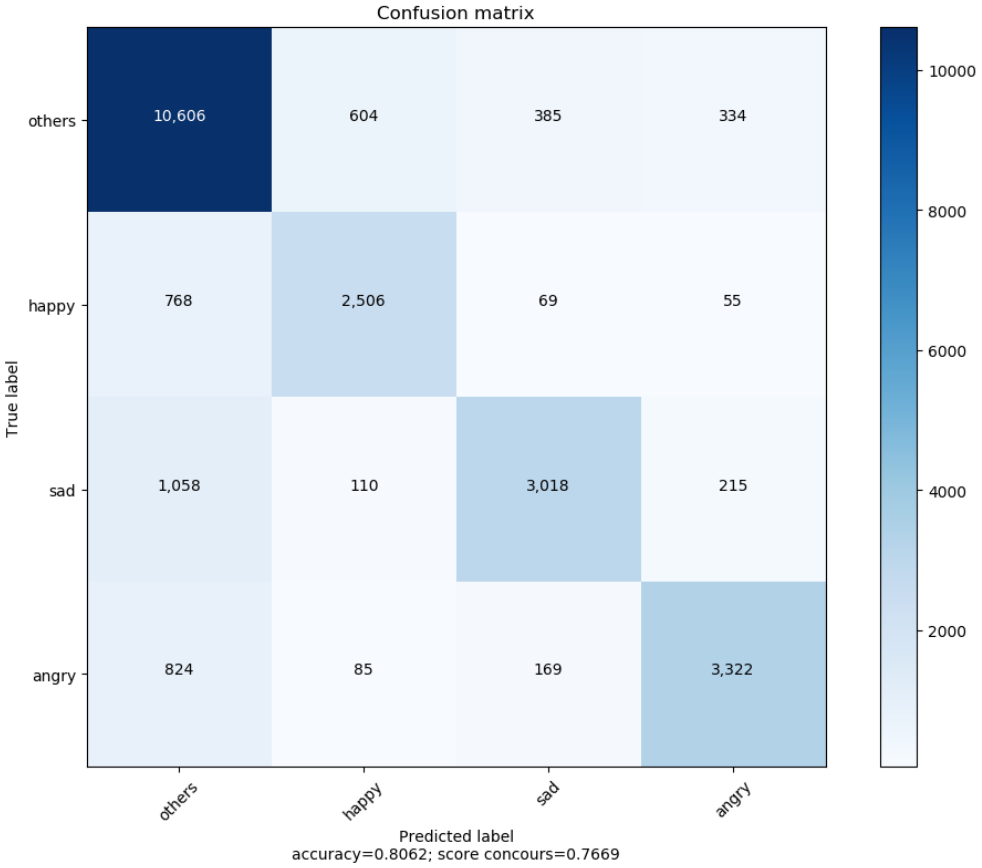
4 Résultats et conclusions

Simulation de prédictions

Après optimisation, on obtient que le modèle ayant le mieux performer en validation est un celui 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* :

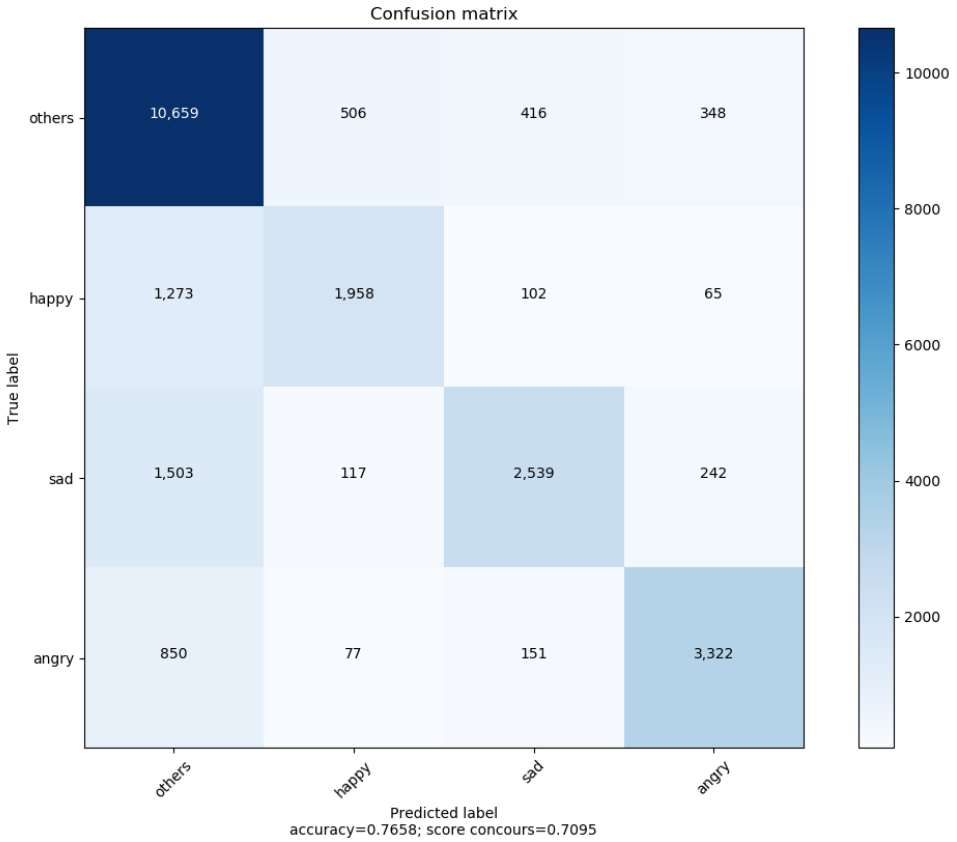
```
Paramètres meilleur modèle: {'clf name': 'LogReg', 'score': 0.7616916517620598, 'Dict param clf': {'penalty': 'l1'}, 'Nom compteur': 'Binary Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
```

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. Avec notre meilleur modèle, on obtient les résultats suivants :



On peut voir que le score utilisé par le concours de 0.7669 est quand même assez haut. On remarque que la plupart des erreurs faites sont majoritairement de prédire happy, sad et angry comme étant others. De plus, la confusion entre sad et angy est un peu plus élevée qu'entre les autres paires d'émotions puisque qu'on peut supposer que les deux sont à caractère négatif.

On peut également observer la matrice de confusion lorsque l'on n'ajoute pas les variables d'emoji et de binettes :



On voit tout de suite que le score est beaucoup moins bon (0.7095 au lieu de 0.7669). On peut voir que beaucoup plus de classe happy et sad ont été prédites comme étant others (768 à 1273 et 1058 à 1503). Pour ce qui est de 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. On peut également comparer notre score aux meilleurs scores soumis à la compétition :

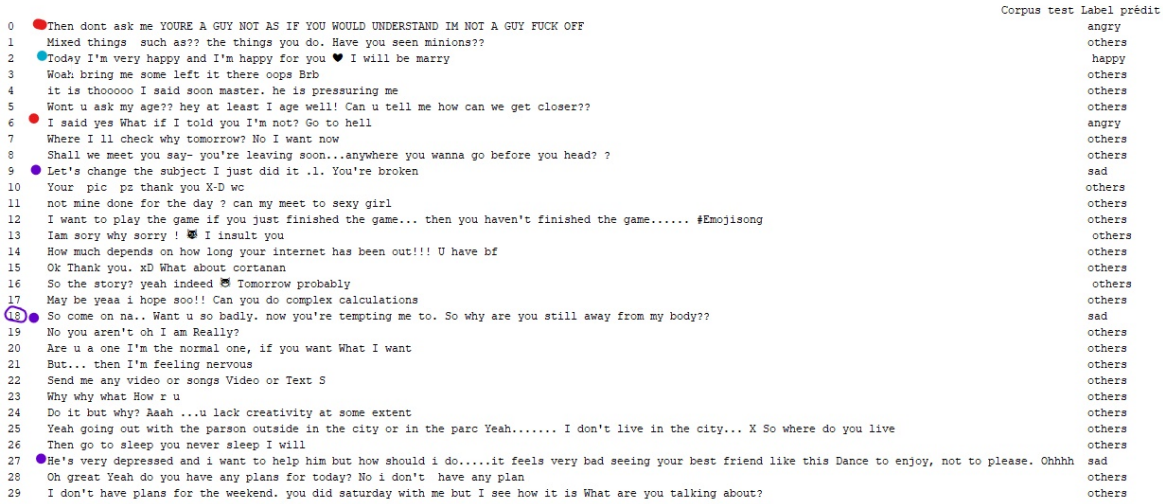
#	Username	Score
1	parag11	0.7482
2	zhaopku	0.7475
3	soujanyaporia	0.7429

On peut conclure qu'un score de 0.7669 en simulation de test est donc très bon. Il est toutefois difficile de bien comparer puisque la répartition des classes n'est pas la même dans le corpus à prédire. Le fichier de test avait environ 16.6% de happy, sad, angry et 50% de others, alors que le fichier à prédire a des proportions de 4% et 88%.

On peut quand même être réellement satisfait du résultat puisque cela ne devrait pas trop affecter nos prédictions.

Prédictions sur le jeu de test

On peut observer nos prédictions sur le jeu de test en entraînant notre modèle sur le jeu d'entraînement au complet. Voici les 30 premières prédictions :



	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 facilement nos prédictions puisqu'on n'a pas les classes à notre disposition.

On peut toutefois voir qu'à tous ceux pour lesquels on prédit soit happy/angry/sad, les prédictions semblent 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.

5 Améliorations et rétrospective sur le projet réalisé

Amélioration 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.
3. Un travail plus exhaustif et robuste pourrait être fait sur les binettes ajoutées comme variables. On pourrait faire un modèle qui analyse toutes les chaînes de caractères spéciaux et qui détermine le degré de signification de chacune pour toutes nos classes.
4. Dans l'optimisation de paramètres, on pourrait tester un nombre de n gramme supérieur à 1.

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.

L'utilisation d'emojis et de binettes pour classer les échanges de textos était également très intéressant.

Le projet fut également très formateur pour l'auteur du code Python qui contrairement à son habitude, a tenté de faire un code plus lisible et réutilisable pour d'autres applications.

Annexes

Liste de score 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': 0.2747888748514791, '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': 0.4151715955232142l6, 'Dict param clf': {'k': 5, 'Weight': 'uniform'}, 'Nom compteur': 'Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'KPPV', 'score': 0.4391683308091976l5, 'Dict param clf': {'k': 5, 'Weight': 'distance'}, 'Nom compteur': 'Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'KPPV', 'score': 0.3918600958939883l5, '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': 0.3647766909211998l5, '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': 0.3301191512922897l5, 'Dict param clf': {'k': 23, 'Weight': 'uniform'}, 'Nom compteur': 'Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'KPPV', 'score': 0.3497741446880941l5, 'Dict param clf': {'k': 23, 'Weight': 'distance'}, 'Nom compteur': 'Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'LogReg', 'score': 0.7440158216604l5, 'Dict param clf': {'penalty': 'l1'}, 'Nom compteur': 'Word counts', 'Dict param compteur': {'n gram': 1, 'freq min': 2}}
{'clf name': 'LogReg', 'score': 0.7205836550007l24, '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, 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': 0.7005337689606l1, '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': 0.3055257453904481l3, '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': 0.3980233223174363l7, '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': 0.3487236447989539l6, '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`