Documentation pour projet Python

Lara DUNUAN, Chinatsu KUROIWA - M2 TAL INALCO

OBJECTIFS

L'objectif de ce projet est de mettre en pratique les notions que nous avons acquises pendant le cours de python de ce semestre et approfondir nos connaissances sur le langage python avec la classification de textes d'opinions.

L'idée du projet est de classer les opinions collectées en ligne, en utilisant des features extraits du jeu de données et en comparant différents algorithmes de classification.

DONNÉES

- corpus_aVoiraLire.xml

Origine: https://github.com/lpschaub/projets_M1/tree/master/corpus/corpus_aVoiraLire

Format: XML

Opération: Extraction de toutes les données en les transformant en fichier TSV.

Détail de corpus :

Le corpus de critiques de films, livres, spectacles et bandes dessinées (http://www.avoir-alire.com) comporte 2704 critiques et les notes qui leur sont associées. Les critiques ont déjà été classées en 3 niveaux de notes, qui donnent 3 classes bien discriminées avec la répartition suivante : la classe 0 (mauvais) avec 309 échantillons, la classe 1 (moyen) avec 615 échantillons, et la classe 2 (bien) avec 1150 échantillons.

MÉTHODOLOGIES

Répartition de travail:

Tous les 2 membres de notre groupe (Lara DUNUAN et Chinatsu KUROIWA) ont participé dans la partie conception du projet, la programmation et la documentation.

Étapes du projet:

1. Recherche du corpus
2. Validation de sujet et corpus par le prof
3. Transformation de donnée de format XML à TSV
4. Pretraitement
5. Choisir les features, combiner des features
6. Choisir les algorithmes
7. Création des modules

   En même temps, nous avons écrit la documentation petit à petit.

Choix de Prétraitement :

A : Normalisation, tokenization, lemmatisation, suppression des caractères inutiles tel que des signes de ponctuations
B : Prétraitement de A avec suppression de stopwords de NLTK
C : Prétraitement de A avec suppression de stopwords avec notre list_stopwords.txt

Choix de Features :

- TF-IDF, la base
- Bag of Words
- 4 features: Longueur des opinions, Nombres des mots, Longueur moyenne des mots, Nombres d'adjectif dans les opinions
  Au début, nous avons testé les 4 features que nous avons extraits séparément. Cependant, les résultats n'étaient pas satisfaisants. Nous avons ensuite essayé de les combiner, en prenant 2 ou 3 features à la fois. Dans la version finale, nous avons choisi de combiner les 4 features.

Choix d'algorithmes :

- LinearSVC
- GaussianNB
- ComplementNB

Des problèmes rencontrés:

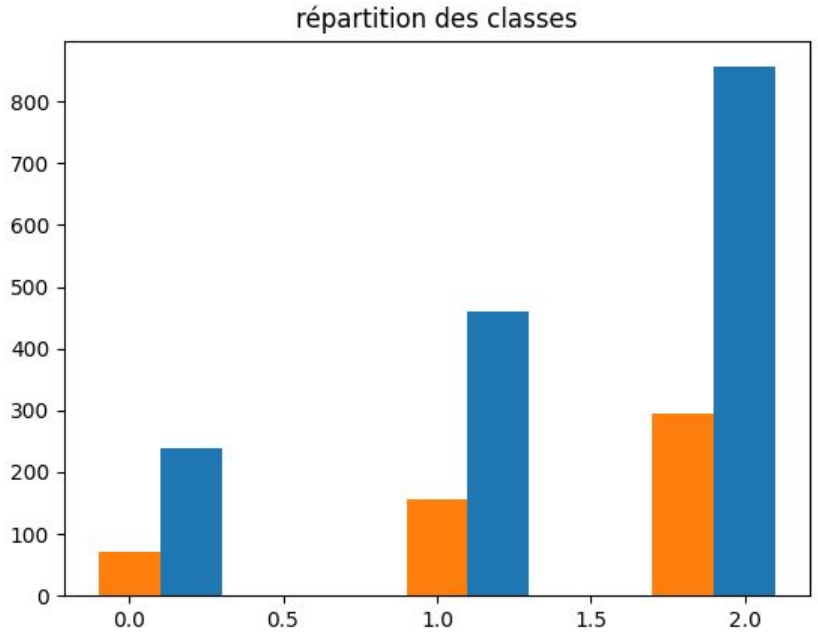
1. Recherche des données

Tout d'abord, nous avons eu la difficulté de la recherche d'un corpus annoté français. Nous avons trouvé quelques corpus intéressants. Cependant ce n'était plus disponible ou c'était payant. Nous avons donc utilisé un corpus proposé par le professeur.

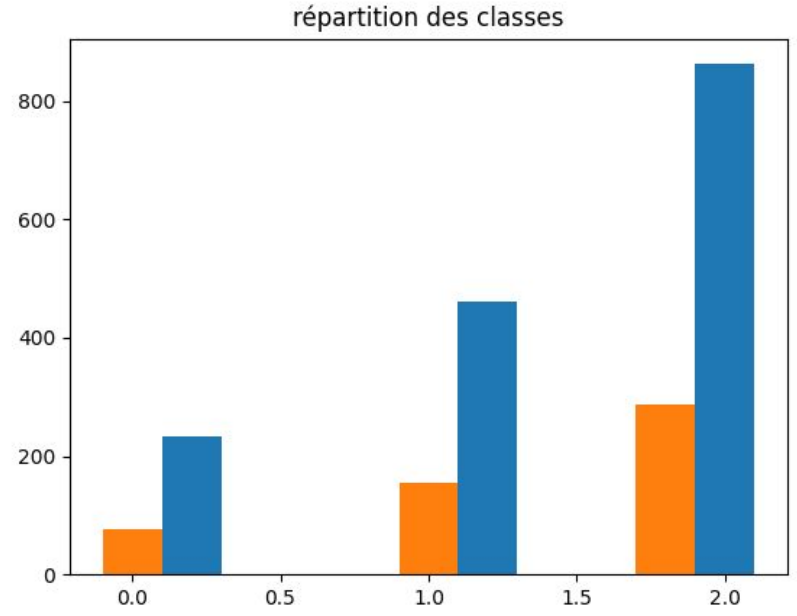
2. Données ne sont pas homogène

Avec matplotlib, nous avons les représentations graphiques des données. Nous avons observé que la répartition des classes à prédire n'est pas homogène pour les données de test. C'est pour cela que nous avons utilisé le paramètre stratify=y pour tenter de résoudre ce problème.

Voici la répartition des classes avant l'utilisation de stratify=y.



Après l'utilisation du paramètre stratify:



L'utilisation du paramètre stratify n'a pas vraiment résolu le problème des données non homogènes. Pour cette raison, le choix de l'algorithme à mettre en œuvre est crucial. Puisque le classificateur Complement Naive Bayes a été conçu pour corriger les «hypothèses sévères» faites par le classificateur standard Multinomial Naive Bayes et qu'il est particulièrement adapté aux ensembles de données déséquilibrés, nous avons décidé d'utiliser cet algorithme.

3. Transformation de donnée

   Lors de transformation de données, nous avons rencontré certaines difficultés.

   Premièrement, nous avons rencontré le mauvais affichage des caractères à cause d'encodage de départ(ISO). Nous avons donc changé l'encodage en ouvrant ce fichier sur Excel en précisant l'encodage en UTF-8 et l'ont sauvegardé.
   Deuxièmement, nous avons eu des difficultés pour la transformation du fichier xml à tsv afin de mettre les éléments dans une colonne pertinente. Tout simplement, nous n'avons pas été à l'aise avec une librairie de python "lxml" qui nous permet de transmettre un fichier xml à la format tsv. Nous avons eu un peu du mal à comprendre sa fonction.
   Dernièrement, nous avons eu des difficultés afin de mettre certains résultats dans un fichier. Par exemple, nous avons voulu stocker les résultats de prétraitement mais nous n'avons malheureusement pas réussi.

4. Choix des hyperparamètres

   Pour nous, il nous fallait plus de temps et d'exemples pour bien comprendre les notions de la plupart des hyperparamètres. Nous les trouvons très techniques. Nous avons donc choisi de jouer avec quelques-uns de ceux que nous avons plus ou moins compris.

   Par exemple, les deux hyperparamètres pour LinearSVC:

   dual : *bool, default=True*
   Select the algorithm to either solve the dual or primal optimization problem. Prefer dual=False when n_samples > n_features.

   C : *float, default=1.0*
   Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive.

IMPLÉMENTATION

Modules python utilisés :

argparse, lxml, csv, re, numpy as np, matplotlib.pyplot as plt, pandas as pd, spacy, nltk, sklearn

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer

from sklearn.model_selection import train_test_split

from sklearn.svm import LinearSVC

from sklearn.naive_bayes import GaussianNB

from sklearn.naive_bayes import ComplementNB

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

from sklearn.model_selection import cross_validate, cross_val_score

from sklearn.model_selection import GridSearchCV

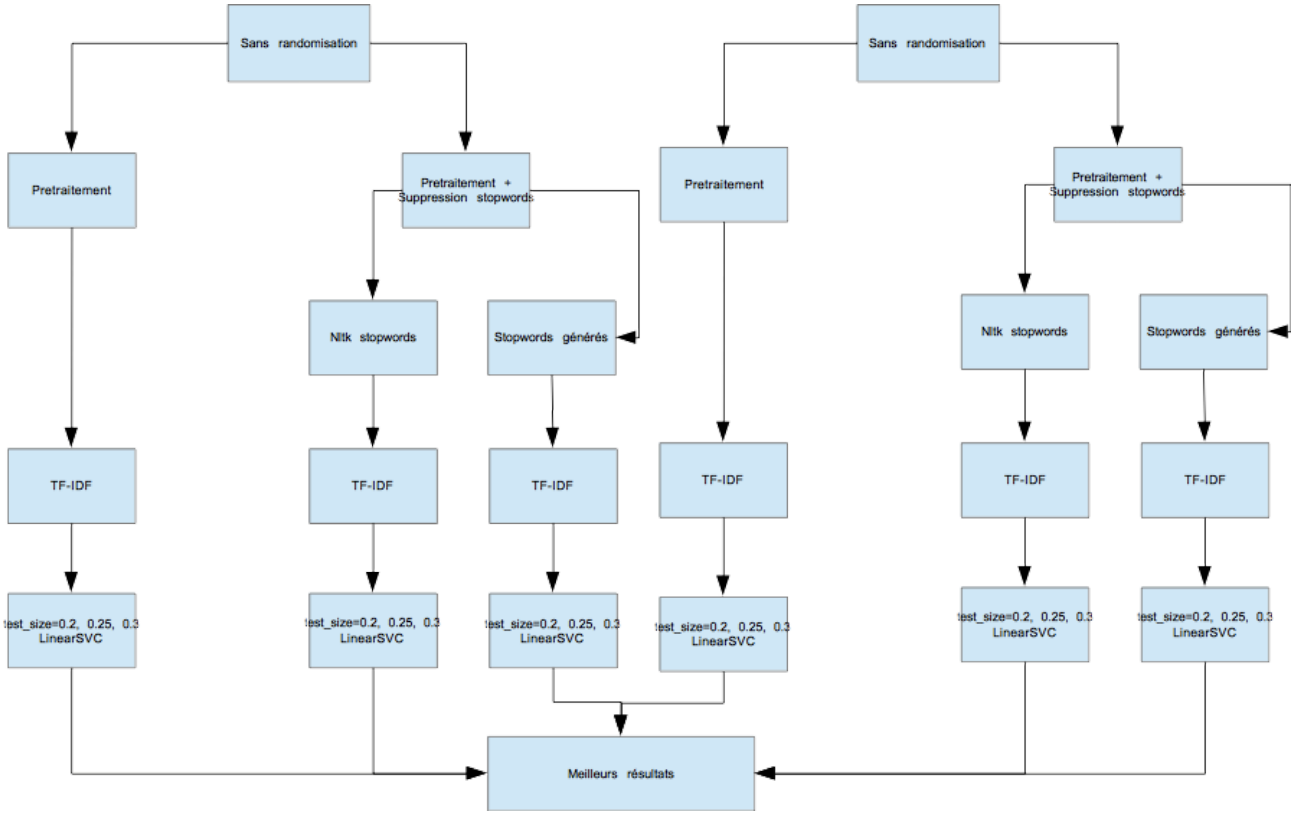
Puis, nous avons créé trois modules pour les différentes étapes de la classification:

- pretraitement
- features
- algorithmes

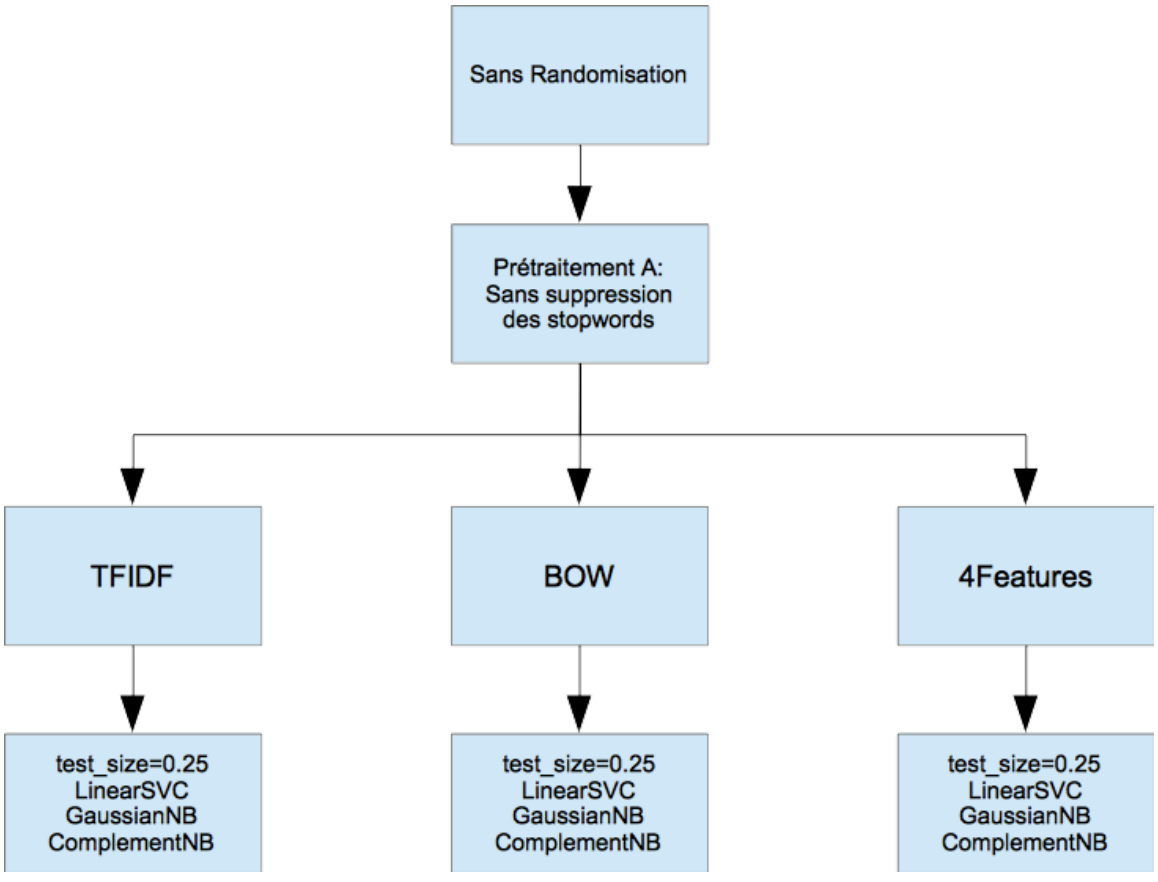
Modélisation:

Nous avons divisé la tâche de classification en deux étapes principales. Tout d'abord, nous avons testé plusieurs configurations dans la première étape. Les configurations varient en fonction de randomisation (oui/non), prétraitement (trois façons) et test_size (0.2, 0.25, 0.3). Pour cette étape, nous avons choisi TF-IDF comme feature et l'algorithme LinearSVC. Après avoir testé ces configurations, nous avons choisi celle qui a rendu les meilleurs résultats. C'est cette configuration que nous avons utilisée dans la deuxième étape.

Voici la modélisation de la première étape:



Et la modélisation de la deuxième étape:



RÉSULTATS

Voici le tableau des résultats avec des configurations différentes.

Etape 1						
	Randomisation	Algorithme	Feature	PreT:A	PreT:B	PreT:C
1	**non**	**LinaireSVC** test_size=0.2 **test_size=0.25** test_size=0.3	**TF-IDF**	 0.62 **0.66** 0.62	-	-
Optimisation des hyperparamètres: rank_test_score=1 (C=0.5, dual=True/False) pour les 3 test_size						
2	non	LinaireSVC test_size=0.2 test_size=0.25 test_size=0.3	TF-IDF	-	 0.63 0.63 0.62	-
Optimisation des hyperparamètres: rank_test_score=1 (C=0.1, dual=True/False) pour test_size=0.2 et 0.25 rank_test_score=1 (C=0.5, dual=True/False) pour test_size=0.3						
3	non	LinaireSVC test_size=0.2 test_size=0.25 test_size=0.3	TF-IDF	-	-	 0.61 0.62 0.63
Optimisation des hyperparamètres: rank_test_score=1 (C=0.5, dual=True/False) pour les 3 test_size						
4	oui	LinaireSVC test_size=0.2 test_size=0.25 test_size=0.3	TF-IDF	 0.53 0.56 0.53	-	-
Optimisation des hyperparamètres: rank_test_score=1 (C=0.5, dual=True) pour test_size=0.2 rank_test_score=1 (C=0.1, dual=True/False) pour test_size=0.2.5 rank_test_score=1 (C=0.5, dual=True/False) pour test_size=0.3						
5	oui	LinaireSVC test_size=0.2 test_size=0.25 test_size=0.3	TF-IDF	-	 0.54 0.61 0.52	-
Optimisation des hyperparamètres: rank_test_score=1 (C=0.1, dual=True/False) pour les 3 test_size						
6	oui	LinaireSVC test_size=0.2 test_size=0.25 test_size=0.3	TF-IDF 0.1 0.5 0.1	-	-	 0.59 0.62 0.55
Optimisation des hyperparamètres: rank_test_score=1 (C=0.1, dual=True/False) pour test_size=0.2 et 0.3 rank_test_score=1 (C=0.5, dual=True/False) pour test_size=0.25						

Etape 2						
	Randomisation	Algorithme	Feature	A	B	C
7	non	GaussianNB ComplementNB LinearSVC test_size=0.25	TF-IDF	0.53 0.60 0.66	-	-
Optimisation des hyperparamètres: GaussianNB: rank_test_score=1 ('var_smoothing': 1e-05) ComplementNB : rank_test_score=1('alpha': 2.0) LinearSVC : rank_test_score=1 (C=0.5, dual=True/False)						
8	**non**	GaussianNB **ComplementNB** LinearSVC **test_size=0.25**	**Bag of Words**	0.54 **0.60** 0.52	-	-
Optimisation des hyperparamètres: GaussianNB: rank_test_score=1 ('var_smoothing': 1e-05) ComplementNB : rank_test_score=1('alpha': 0.5) LinearSVC : rank_test_score=1 (C=0.1, dual=False)						
9	**non**	**GaussianNB** ComplementNB LinearSVC **test_size=0.25**	**4Features**	**0.54** 0.49 0.31	-	-
Optimisation des hyperparamètres: GaussianNB: rank_test_score=1 ('var_smoothing': 1e-05) ComplementNB : rank_test_score=1('alpha': 0.5, 2.0) LinearSVC : rank_test_score=1 (C=[0.1,0.5,1,10], dual=False)						

Discussion:

1. Analyse

Lors de la première étape, nous avons testé 3 prétraitements différents avec un feature TF-IDF et un algorithme LinearSVC en changeant la valeur de séparation de données en test et en train (test_size). De plus, nous avons testé s'il y a un impact si on équilibre le nombre de texte de chaque classe de notre corpus.

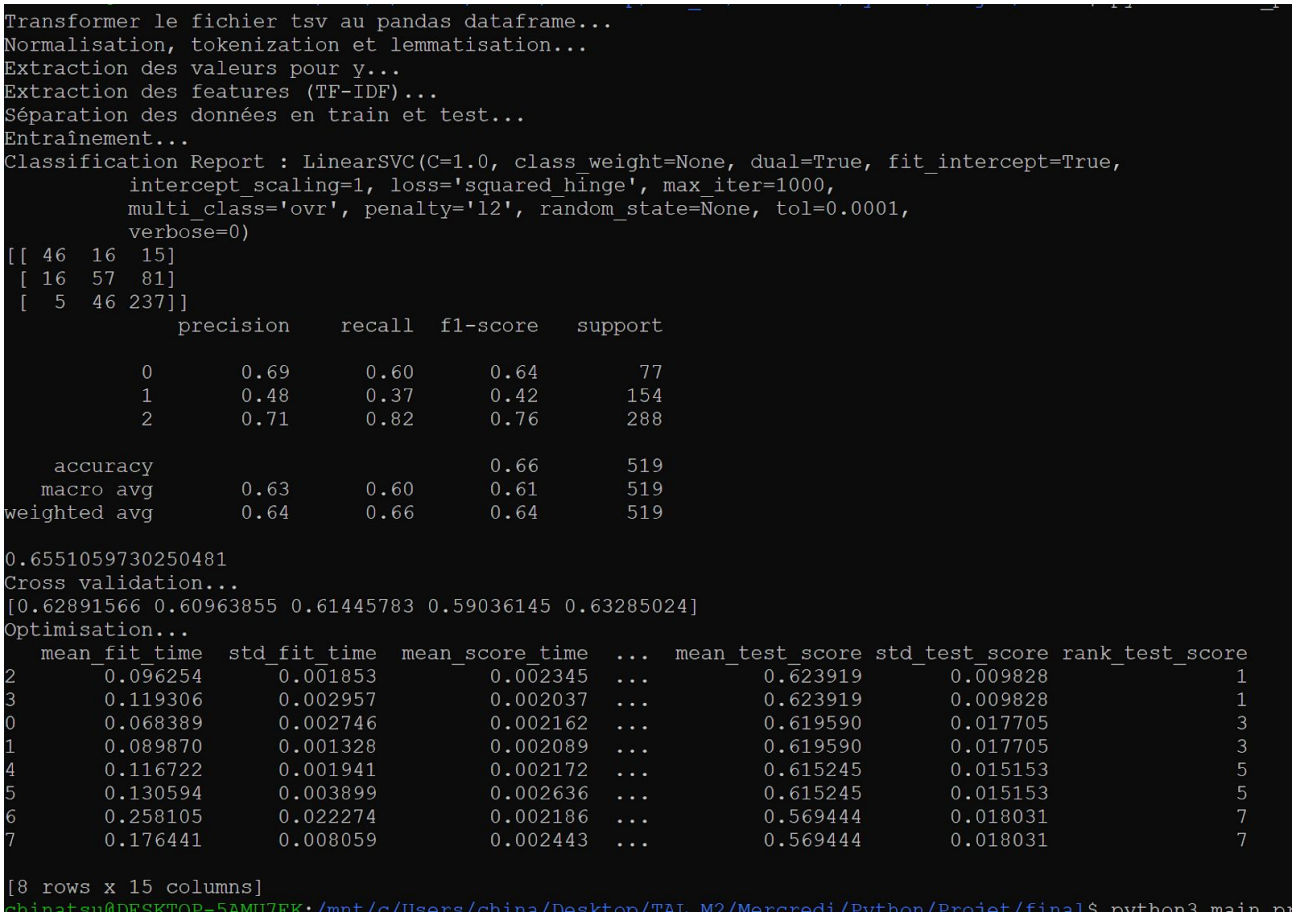
Nous avons obtenu le meilleur résultat avec Prétraitement: A (Normalisation, tokenization, lemmatisation, suppression des caractères inutiles tel que des signes de ponctuation, sans suppression des stopwords), test_size = 0.25 et sans Randomisation. En effet nous avions attendu qu'avec randomisation aura le meilleur résultat car l'équilibre du nombre de données entre les classes pourraient être un biais mais ce n'était pas le cas. Avec ce corpus, lorsque nous équilibrons le nombre d'échantillons, il ne reste que 309 échantillons pour chaque classe et nous avons remarqué que ce n'était pas la taille suffisante pour la classification de donné.

Par ailleurs, les résultats de 3 prétraitements différents étaient quasi identiques. Nous avons donc testé en deuxième étape, des différents algorithmes avec des features différents sur les conditions qui ont donné le meilleurs résultat en premier étape.

Lors de la deuxième étape, nous avons testé trois features (TF-IDF, Bag of Words et 4Features : la combinaison des 4 features que nous avons extrait) et trois algorithmes (LinearSVC, GaussianNB et ComplementNB) . Comme ce guide (https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html) recommande d'utiliser les Naive Bayes comme algorithme pour la classification supervisée pour notre jeu de données, nous avons choisi Gaussian Naive Bayes et Complement Naive Bayes. Ce dernier algorithme est particulièrement adapté aux ensembles de données déséquilibrés comme les données sur lesquelles nous travaillons.

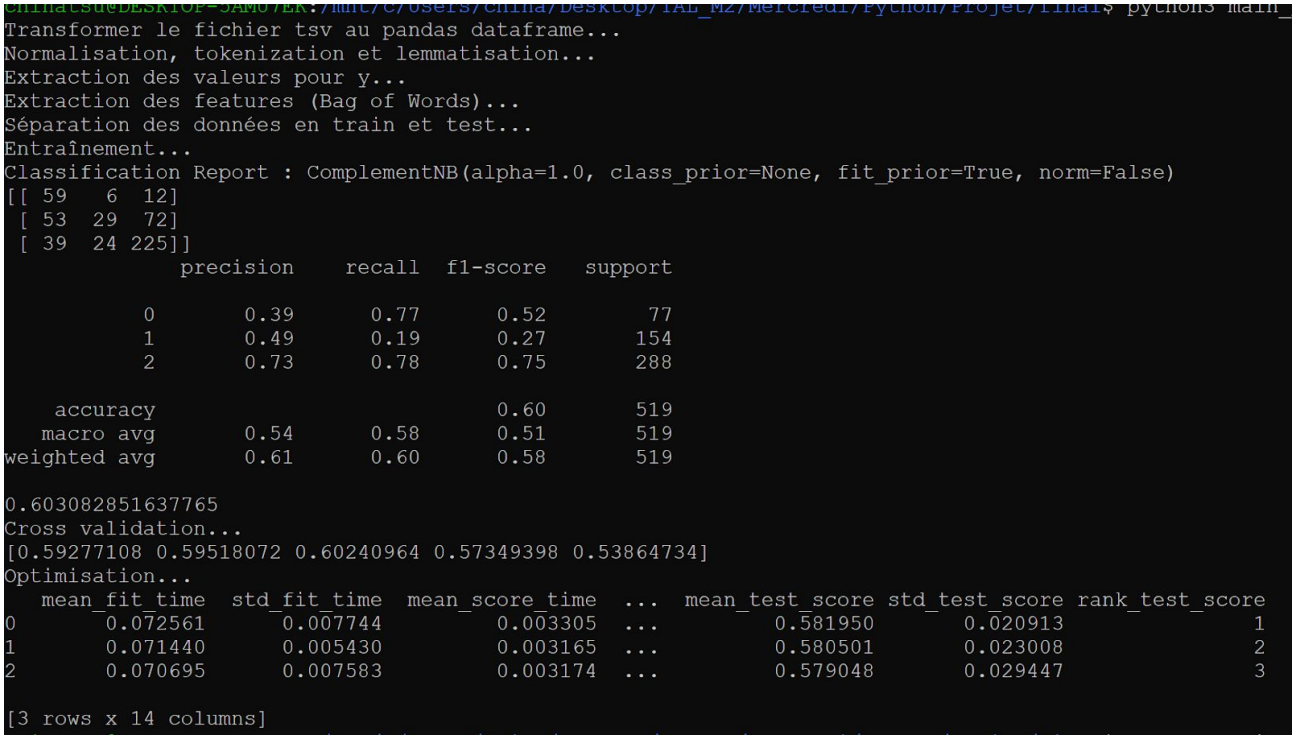
Le meilleur résultat était identique à celui de la première étape. Les résultats n'ont pas été à la hauteur de nos espérances. Cependant nous avons remarqué que chaque algorithme a donné les meilleurs résultats en fonction de features.

Pour le feature TF-IDF, LinearSVC donne le meilleur résultat:



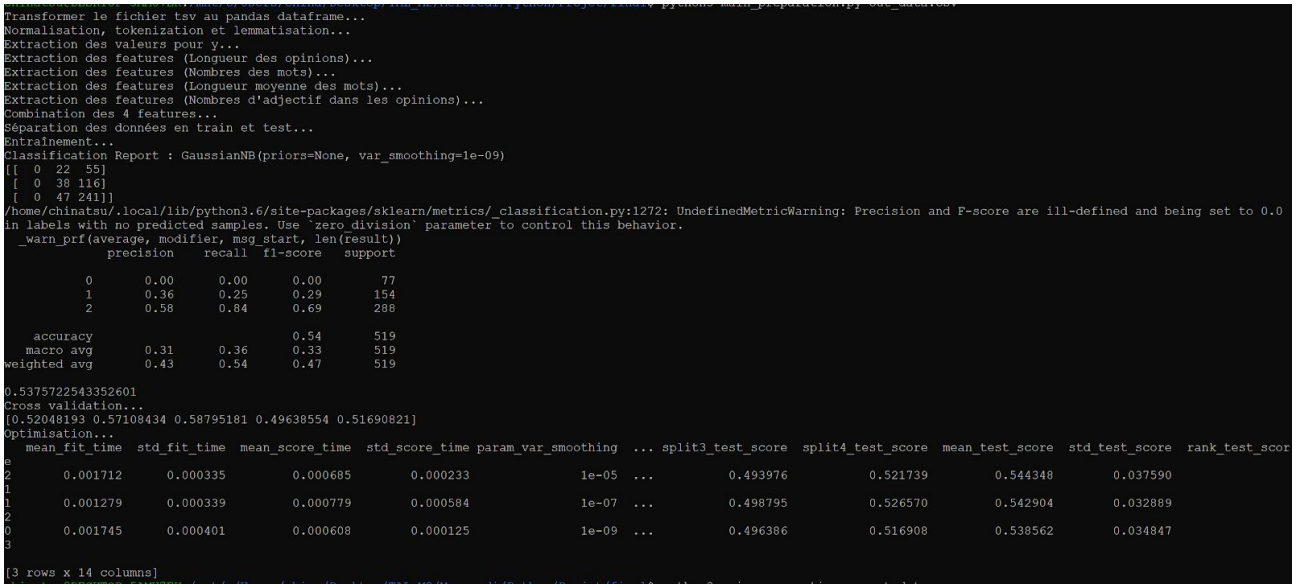
```
Transformer le fichier tsv au pandas dataframe...
Normalisation, tokenization et lemmatisation...
Extraction des valeurs pour y...
Extraction des features (TF-IDF)...
Séparation des données en train et test...
Entraînement...
Classification Report : LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
          verbose=0)
[[ 46  16  15]
 [ 16  57  81]
 [  5  46 237]]
              precision    recall  f1-score   support

           0       0.69      0.60      0.64        77
           1       0.48      0.37      0.42       154
           2       0.71      0.82      0.76       288

    accuracy                           0.66       519
   macro avg       0.63      0.60      0.61       519
weighted avg       0.64      0.66      0.64       519

0.6551059730250481
Cross validation...
[0.62891566 0.60963855 0.61445783 0.59036145 0.63285024]
Optimisation...
   mean_fit_time  std_fit_time  mean_score_time  ...  mean_test_score std_test_score rank_test_score
2       0.096254      0.001853         0.002345  ...         0.623919       0.009828               1
3       0.119306      0.002957         0.002037  ...         0.623919       0.009828               1
0       0.068389      0.002746         0.002162  ...         0.619590       0.017705               3
1       0.089870      0.001328         0.002089  ...         0.619590       0.017705               3
4       0.116722      0.001941         0.002172  ...         0.615245       0.015153               5
5       0.130594      0.003899         0.002636  ...         0.615245       0.015153               5
6       0.258105      0.022274         0.002186  ...         0.569444       0.018031               7
7       0.176441      0.008059         0.002443  ...         0.569444       0.018031               7

[8 rows x 15 columns]
chinatsu@DESKTOP-5AMU7EK:/mnt/c/Users/china/Desktop/TAL_M2/Mercredi/Python/Projet/final$ python3 main.py
```

Pour le feature Bag of Words, ComplementNB donne le meilleur résultat:



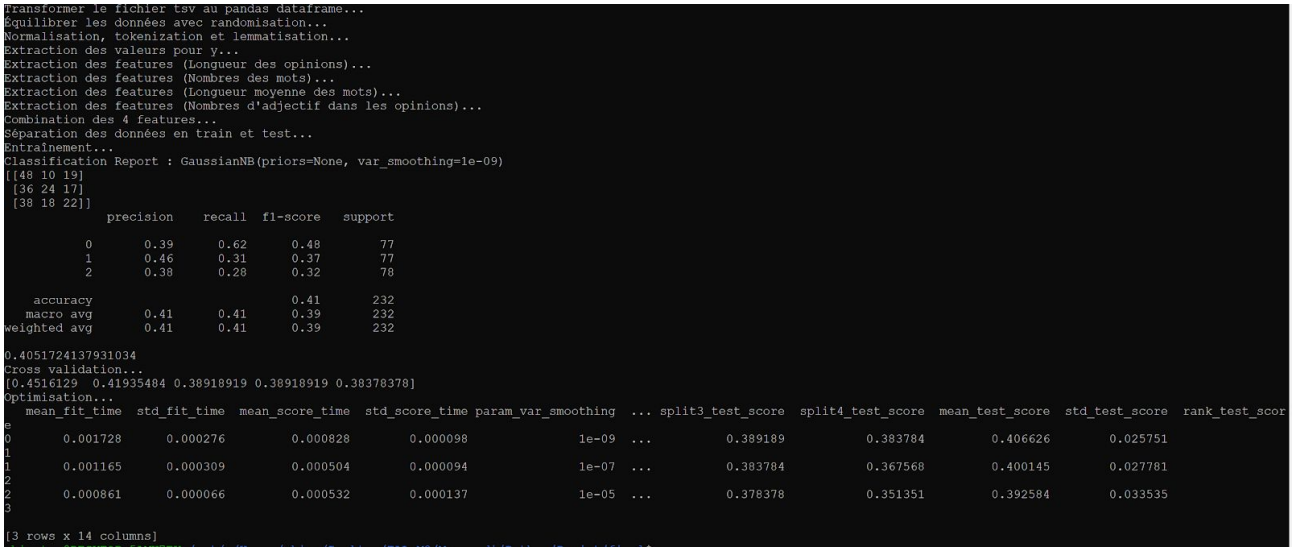
```
chinatsu@DESKTOP-5AM07EK:/mnt/c/Users/China/Desktop/TAL_M2/Mercredi/Python/Projet/Final$ python3 main_
Transformer le fichier tsv au pandas dataframe...
Normalisation, tokenization et lemmatisation...
Extraction des valeurs pour y...
Extraction des features (Bag of Words)...
Séparation des données en train et test...
Entraînement...
Classification Report : ComplementNB(alpha=1.0, class_prior=None, fit_prior=True, norm=False)
[[ 59   6  12]
 [ 53  29  72]
 [ 39  24 225]]
              precision    recall  f1-score   support

           0       0.39      0.77      0.52        77
           1       0.49      0.19      0.27       154
           2       0.73      0.78      0.75       288

    accuracy                           0.60       519
   macro avg       0.54      0.58      0.51       519
weighted avg       0.61      0.60      0.58       519

0.6030828516377649
Cross validation...
[0.59277108 0.59518072 0.60240964 0.57349398 0.53864734]
Optimisation...
   mean_fit_time  std_fit_time  mean_score_time  ...  mean_test_score std_test_score rank_test_score
0       0.072561      0.007744         0.003305  ...         0.581950       0.020913               1
1       0.071440      0.005430         0.003165  ...         0.580501       0.023008               2
2       0.070695      0.007583         0.003174  ...         0.579048       0.029447               3

[3 rows x 14 columns]
```

Pour les 4Features, nous observons que GaussianNB donne la plus grande accuracy. Cependant, en y regardant de plus près, nous constatons qu'aucune classification n'a été effectuée pour la classe 0.



```
Transformer le fichier tsv au pandas dataframe...
Normalisation, tokenization et lemmatisation...
Extraction des valeurs pour y...
Extraction des features (Longueur des opinions)...
Extraction des features (Nombres des mots)...
Extraction des features (Longueur moyenne des mots)...
Extraction des features (Nombres d'adjectif dans les opinions)...
Combination des 4 features...
Séparation des données en train et test...
Entraînement...
Classification Report : GaussianNB(priors=None, var_smoothing=1e-09)
[[  0  22  55]
 [  0  38 116]
 [  0  47 241]]
/home/chinatsu/.local/lib/python3.6/site-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0
in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        77
           1       0.36      0.25      0.29       154
           2       0.58      0.84      0.69       288

    accuracy                           0.54       519
   macro avg       0.31      0.36      0.33       519
weighted avg       0.43      0.54      0.47       519

0.5375722543352601
Cross validation...
[0.52048193 0.57108434 0.58795181 0.49638554 0.51690821]
Optimisation...
   mean_fit_time  std_fit_time  mean_score_time  std_score_time param_var_smoothing  ... split3_test_score  split4_test_score  mean_test_score  std_test_score  rank_test_scor
e
2       0.001712      0.000335         0.000685        0.000233              1e-05  ...          0.493976           0.521739         0.544348        0.037590
1
1       0.001279      0.000339         0.000779        0.000584              1e-07  ...          0.498795           0.526570         0.542904        0.032889
2
0       0.001745      0.000401         0.000608        0.000125              1e-09  ...          0.496386           0.516908         0.538562        0.034847
3

[3 rows x 14 columns]
```

Nous avons ensuite tenté de refaire le test. Mais cette fois, nous avons utilisé la randomisation pour équilibrer les données. Nous observons une baisse de l'accuracy mais nous avons maintenant des données correctement classées pour la classe 0.



```
Transformer le fichier tsv au pandas dataframe...
Équilibrer les données avec randomisation...
Normalisation, tokenization et lemmatisation...
Extraction des valeurs pour y...
Extraction des features (Longueur des opinions)...
Extraction des features (Nombres des mots)...
Extraction des features (Longueur moyenne des mots)...
Extraction des features (Nombres d'adjectif dans les opinions)...
Combination des 4 features...
Séparation des données en train et test...
Entraînement...
Classification Report : GaussianNB(priors=None, var_smoothing=1e-09)
[[48 10 19]
 [36 24 17]
 [38 18 22]]
              precision    recall  f1-score   support

           0       0.39      0.62      0.48        77
           1       0.46      0.31      0.37        77
           2       0.38      0.28      0.32        78

    accuracy                           0.41       232
   macro avg       0.41      0.41      0.39       232
weighted avg       0.41      0.41      0.39       232

0.4051724137931034
Cross validation...
[0.4516129  0.41935484 0.38918919 0.38918919 0.38378378]
Optimisation...
   mean_fit_time  std_fit_time  mean_score_time  std_score_time param_var_smoothing  ... split3_test_score  split4_test_score  mean_test_score  std_test_score  rank_test_scor
e
0       0.001728      0.000276         0.000828        0.000098              1e-09  ...          0.389189           0.383784         0.406626        0.025751
1
1       0.001165      0.000309         0.000504        0.000094              1e-07  ...          0.383784           0.367568         0.400145        0.027781
2
2       0.000861      0.000066         0.000532        0.000137              1e-05  ...          0.378378           0.351351         0.392584        0.033535
3

[3 rows x 14 columns]
```

Avec ces expériences, nous pouvons conclure que TF-IDF donne toujours la meilleure accuracy parmi les features. Là encore, le volume et la qualité du corpus jouent un rôle crucial.

2. Amélioration

En conclusion, vu que c'est toujours TF-IDF qui donne le meilleur résultat dans tous les cas, nous constatons que le premier point d'amélioration serait la qualité de prétraitement pour la prochaine fois. Nous n'avons pas considéré l'existence de fautes orthographes dans les textes cette fois-ci. Ce serait un travail délicat mais cela pourrait être un traitement important pour d'autres prétraitements tel que lemmatisation, normalisation et création des liste de stopword.

Pour le deuxième point d'amélioration, nous pouvons observer que la répartition des classes était très déséquilibrée. Nous avons remarqué que pour la classe '0' qui contient la taille de donnée la plus petite est toujours mal classée. Nous aimerions donc utiliser les données plus importantes et plus équilibrées entre les classes et de tester les algorithmes différents.

Enfin, pour la prochaine fois, nous aimerions améliorer quelques points au niveau de la technique. Premièrement, nous n'avons pas réussi à stocker des résultats de prétraitement dans un fichier. Nous avons donc perdu du temps en répétant à chaque fois le même prétraitement qui prend du temps et qui est lourd pour l'ordinateur. Ensuite, nous voulons améliorer la visualisation des résultats. Nous aurions voulu explorer davantage les fonctionnalités de matplotlib afin d'avoir une représentation plus visuelle des résultats.

List des fichiers:
fichier input : corpus_aVoiraLire.xml, out_data.tsv
scriptes : xml_to_tsv.py (transformer le fichier xml en tsv), classify.py (programme main), modules (pretraitement.py, features.py, algorithmes.py), get_wordfreq.py (pour construire list_stopwords.txt)
list_stopwords.txt : utilisé pour la suppression des stopwords
fichier output : rank_test_score_ComplementNB_BOW.txt, rank_test_score_GaussianNB_4Features.txt, rank_test_score_LinearSVC_TFIDF.txt