UFR LLASIC

UNIVERSITÉ Grenoble Alpes

Projet Python :
Analyse sentimentale de commentaires Web

Réalisé par | Julie COLETTI
Myriam EL HELOU

Dirigé par | Oliver KRAIF

Master 2 Sciences Du Langage parcours Industries De la Langue

Table des matières

Le petit état de l'art

L'analyse de sentiments ou *l'opinion mining* est apparu au début des années 2000 et il a pris en ampleur grâce à l'abondance des données provenant des réseaux sociaux. Le but de ces analyses est de déduire les sentiments exprimés par un large nombre de données.
Il existe différents outils permettant d'effectuer l'identification des sentiments dégagés par un texte. Nous pouvons citer :

- Werfamous : cet outil est un outil en ligne. Il donne un score de sentiment sur une échelle de -100 à 100, ainsi qu'un niveau de confiance lié à score fourni.
- SenticNet : cet outil fournit une analyse avancée de la polarité des mots.
- Wordnet : à l'aide des synset (synonymes), cet outil permet de savoir si un mot est positif ou non.
- SentiWordNet : c'est une extension de Wordnet. A chaque groupe de synonyme dans Wordnet, il attribue 3 scores de sentiment : positif, négatif et neutre.

Il existe deux grandes catégories d'analyse de sentiments :

- L'analyse lexicale : cette analyse consiste à déduire les sentiments dégagés par une phrase à l'aide de l'analyse sémantique des mots qui la composent.
- L'analyse par apprentissage automatique : c'est une méthode de classification de mots. Elle est basée sur différents algorithmes :
    - Classification naïve bayésienne
    - Principe d'entropie maximale
    - Séparateurs à vaste marge

Nous allons nous focaliser un peu sur les séparateurs à vaste marge car nous avons utilisé cet algorithme pour effectuer notre analyse.

Les séparateurs à vaste marge font partis des modèles discriminants en apprentissage machine ce qui veut dire qu'ils séparent les classes par des frontières. C'est une méthode de classification binaire par apprentissage supervisé. L<es SVM permettent une classification de deux classes en faisant appel à un jeu de données d'apprentissage pour apprendre les paramètres du modèle. Cette méthode est basée sur l'utilisation de fonctions dites « noyau » (kernel) qui permettent une séparation optimale des données. Le but du SVM est de trouver un classificateur qui va séparer les données et maximiser la distance entre les deux classes en question.

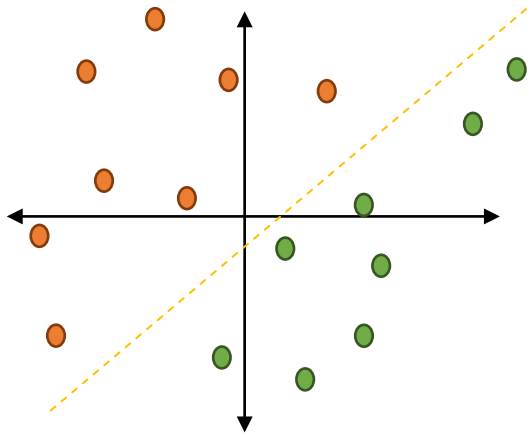


Figure 1 : exemple de frontière de séparation (hyperplan) de 2 classes

Dans la figure 1, nous apercevons la frontière qui sépare les 2 classes (orange et verte). Les points qui sont proches de la frontière sont ceux qui ont aidé à déterminer sa position. On les appelle vecteurs de support.

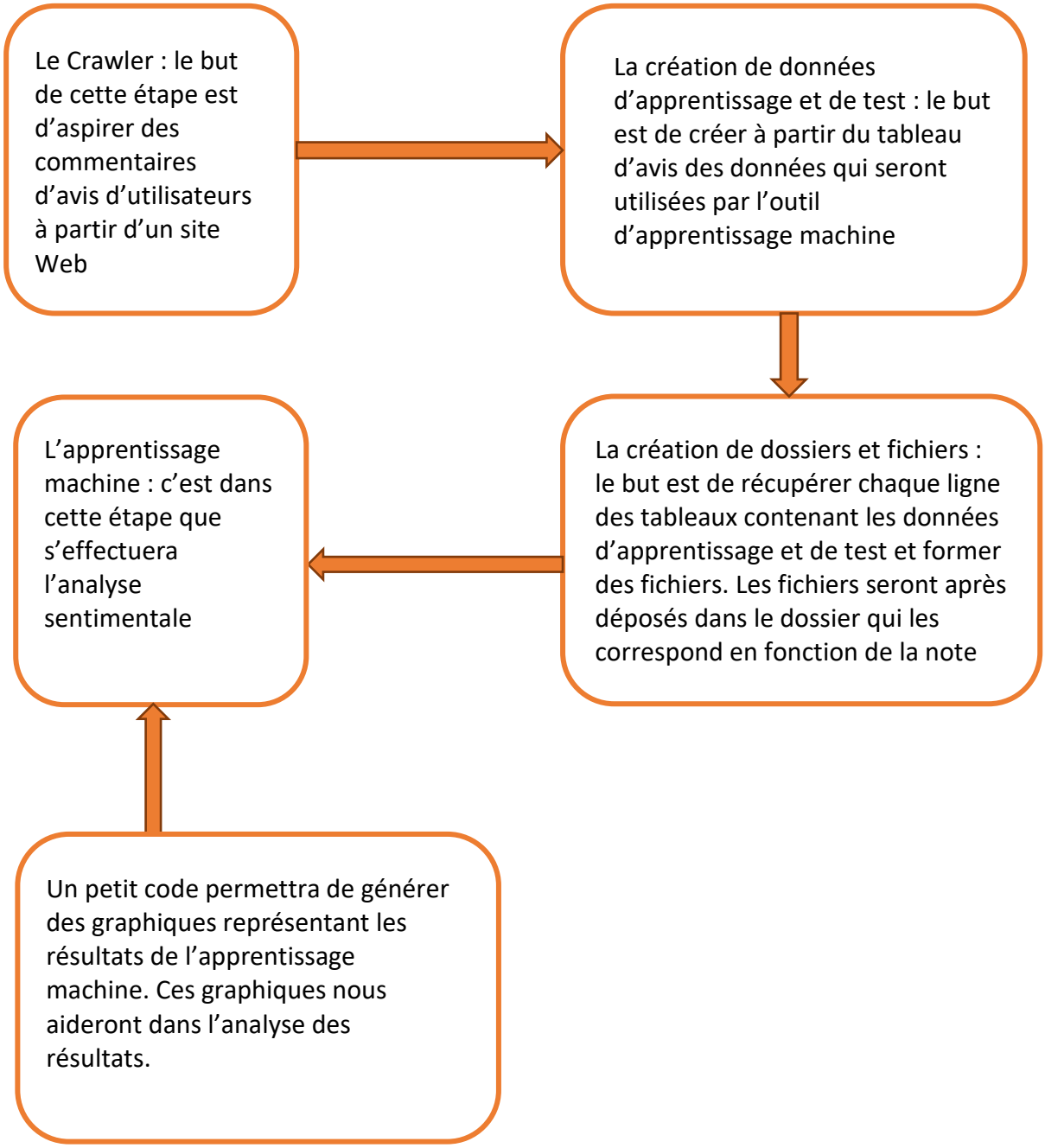
Il existe différents types de noyaux pour le SVM :

- RBF ou Radial Basis Function Kernel
- Noyau linéaire

Dans le travail qui va suivre, nous allons utiliser le modèle SVM avec les deux types de noyaux pour effectuer notre analyse de sentiment et ensuite nous effectuerons une comparaison entre les deux types de SVM.

Le projet

Comme l'indique le titre de ce rapport, le projet a comme but d'effectuer une analyse sentimentale sur une base de données de commentaires Web. Il se découpe en 4 grandes étapes qui seront détaillées au fur et à mesure du rapport.



Le crawler – Cette partie est écrite par Myriam EL HELOU

But de cette étape

L'aspirage de commentaires est la première étape de notre projet. Cette étape permet de fournir une base de données d'avis d'utilisateurs.

Librairies à installer ou à importer dans le code

Pour pouvoir lancer le code de cette étape, il faut installer :

- La librairie *urllib* :
  commande d'installation : *pip install urllib*
- La librairie *csv* :
  commande d'installation : *pip install csv*
- Le module *time* : ce module est un module standard de Python, il n'est pas nécessaire de l'installer
- Le module *re* : ce module est un module standard de Python, il n'est pas nécessaire de l'installer
- Le module *HTML* : ce module est un module standard de Python, il n'est pas nécessaire de l'installer

Ces commandes doivent être effectuées à partir du terminal.

Comment ça marche

Ce code prend en entrée un fichier « lien_tripadvisor.txt » qui contiendra des liens vers des pages d'avis sur des restaurants. Il fournira en sortie un fichier « sortie_liens.csv » qui sera de la forme suivante :

Note	**Quote**	**Commentaire**
40	Fin et original	Cuisine raffinée avec une belle présentation . On s'est régalé ! J'aurais bien mangé un peu plus de morceaux de morue en entrée ... La décoration de la salle manque un peu de chaleur mais l'accueil n'en manque pas . Nous reviendrons sûrement

Je me suis basée sur le site « tripadvisor » pour obtenir les multiples commentaires.
Avant de commencer à coder, j'ai accédé aux codes sources du site afin de voir les balises qui entourent les commentaires des utilisateurs. Après avoir identifié les balises qui me permettront de récupérer les 3 types de données qui composeront le fichier .csv j'ai formé des expressions régulières qui permettront ensuite à l'aide de la commande re.findall () de n'aspirer que ce qui m'intéresse.
J'avais alors 3 expressions régulières :

- une pour aspirer les notes
- une pour aspirer les quotes
- une pour aspirer les commentaires.

Après avoir écrit les expressions régulières, j'ai commencé à parcourir le fichier .txt et à accéder à chaque page avec la fonction urllib.request.urlopen ().
De cette page je récupérais les données et je vérifiais que le nombre de notes est identique au nombre de commentaires pour ensuite les écrire dans le fichier .csv.
Entre chaque itération, j'arrête le code 3 secondes pour que le site ne détecte pas mon code comme des activités malveillantes.

Problèmes rencontrés

Un problème que j'ai rencontré est l'incompatibilité de l'expression régulière des commentaires lorsqu'en plus du commentaire il y avait la réponse du restaurateur. J'ai alors regardé de nouveau le code source du site pour détecter les différences de balises entre ces deux types de commentaires.

Défauts de ce code

Ce code contient plusieurs défauts :

- Je n'ai pas su comment récupérer la totalité des commentaires lorsqu' ils ne sont pas affichés en entier sur la page.
- L'utilisation des expressions régulières est très contraignante, car elle rend le code spécifique au site et ce dernier ne pourra pas être utilisé pour aspirer des commentaires d'autres sites.
- Le code a besoin d'un fichier d'entrée avec des liens et ne pourra donc pas parcourir différentes pages du même restaurant sauf si je les lui fournis dans le fichier .txt

Potentielles améliorations

Il serait intéressant d'essayer d'autres librairies Python pour aspirer les commentaires. Par exemple refaire cette étape avec la librairie « scrapy » qui permettra de naviguer entre différentes pages du même site sans avoir à rentrer les liens manuellement.
De plus, il faudra régler le problème des commentaires non complets, car dans l'analyse des sentiments chaque mot dans un commentaire est intéressant et peut apporter des informations significatives pour l'outil d'apprentissage machine.

La création de données d'apprentissage et de test – Cette partie est écrite par Julie COLETTI

But de cette étape

La création de données d'apprentissage et de test est la seconde étape de notre projet. Cette étape permet d'obtenir des données qui seront utilisées par l'outil d'apprentissage machine.

Librairies à installer ou à importer dans le code

Pour pouvoir lancer le code de cette étape, il faut installer :

- Le wrapper *treetaggerwrapper* :
  commande d'installation : *pip install treetaggerwrapper*
- La librairie *nltk* :
  commande d'installation : *pip install nltk*
- La librairie *csv* :
  commande d'installation : *pip install csv*
- Le module *re* : ce module est un module standard de Python, il n'est pas nécessaire de l'installer

Ces commandes doivent être effectuées à partir du terminal.

Comment ça marche

Ce code prend en entrée un fichier « sortie_liens.csv » qui contient des commentaires, leur quote, et leur note sous la forme suivante :

Note	**Quote**	**Commentaire**
40	Excellent	Restaurant excellent , accueil très sympathique . Belle carte , plats originaux et goûteux , une belle cuisine ! !

et produit 6 fichiers :

1. « notes_unigrams.csv » : contient la note + la liste des unigrams du commentaire.

Note	Commentaire
40	Restaurant excellent , accueil très sympathique . Belle carte , plats originaux et goûteux , une belle cuisine ! !

2. « notes_unigrams_SMO.csv » : contient la note + la liste des unigrams sans les mots outils du commentaire.

Note	Commentaire
40	Restaurant excellent accueil très sympathique Belle carte plats originaux goûteux belle cuisine

3. « notes_bigrams.csv » : contient la note + la liste des bigrams du commentaire.

Note	Commentaire
40	Restaurant_excellent excellent_, ,_accueil accueil_très très_sympathique sympathique_. ._Belle Belle_carte carte_, ,_plats plats_originaux originaux_et et_goûteux goûteux_, ,_une une_belle belle_cuisine cuisine_! !_!

4. « notes_bigrams_SMO.csv » : contient la note + la liste des bigrams sans les mots outils du commentaire.

Note	Commentaire
40	Restaurant_excellent excellent_accueil accueil_très très_sympathique sympathique_Belle Belle_carte carte_plats plats_originaux originaux_goûteux goûteux_belle belle_cuisine

5. « notes_trigrams.csv » : contient la note + la liste des trigrams du commentaire.

Note	Commentaire
40	Restaurant_excellent_, excellent_,_accueil ,_accueil_très accueil_très_sympathique très_sympathique_. sympathique_._Belle ._Belle_carte Belle_carte_, carte_,_plats ,_plats_originaux plats_originaux_et originaux_et_goûteux et_goûteux_, goûteux_,_une ,_une_belle une_belle_cuisine belle_cuisine_! cuisine_!_!

6. « notes_trigrams_SMO.csv » : contient la note + la liste des trigrams sans les mots outils du commentaire.

Note	Commentaire
40	Restaurant_excellent_accueil excellent_accueil_très accueil_très_sympathique très_sympathique_Belle sympathique_Belle_carte Belle_carte_plats carte_plats_originaux plats_originaux_goûteux originaux_goûteux_belle goûteux_belle_cuisine

Dans ce projet, nous considérons comme mots outils : les abréviations, les déterminants, les interjections, les conjonctions, les noms propres, les pronoms, les prépositions, les ponctions et les symboles. La liste des étiquettes correspondantes à ces catégories est disponible à cette adresse : http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/data/french-tagset.html

Problèmes rencontrés

La plus grande difficulté rencontrée lors de ce projet fut l'installation de Treetagger et du wrapper *treetaggerwrapper.*
Un autre problème rencontré fut le placement dans le code de la ligne :
tagger = treetaggerwrapper.TreeTagger(TAGLANG='fr').

Correspondant aux paramètres français, elle était initialement écrite dans la boucle for destinée à étiqueter chaque commentaire. De ce fait, pour chaque commentaire, le lexique se chargeait entièrement, causant en étiquetage extrêmement long (environ une dizaine d'heures pour 200 commentaires)
Pour permettre un étiquetage bien plus rapide et efficace, il suffisait de charger le lexique une seule fois pour tout le traitement en plaçant la ligne de code en dehors de la boucle.

Défauts de ce code

Le principal défaut de ce code est le fait qu'il ne soit adapté qu'à une seule langue, le français. En effet, si le fichier d'entrée contenait des commentaires non francophones, les fichiers de sorties ne contiendraient pas les résultats attendus notamment pour deux raisons :

- Seuls les paramètres français sont installés et utilisés
  (tagger = treetaggerwrapper.TreeTagger(TAGLANG='fr'))
- Les étiquettes des catégories considérées comme mots outils sont spécifiques au français.

Potentielles améliorations

L'une des potentielles améliorations de ce code pourrait être une version plus compacte et moins redondante de la partie du code permettant l'écriture dans les fichiers.

Ce code pourrait également être amélioré significativement en permettant des analyses multilingues. Effectivement, à l'heure actuelle, les mots non francophones pouvant éventuellement être présents dans un commentaire français ne sont pas identifiés, donc non étiquetés, et par conséquent perdus, causant potentiellement une perte sémantique.

La création de dossiers – Cette partie est écrite par Myriam EL HELOU

But de cette étape

Le but de ce code est de récupérer chaque ligne des tableaux contenant les données d'apprentissage et de test et l'écrire dans un fichier. Les fichiers seront après déposés dans le dossier qui les correspond en fonction de la note. Les commentaires positifs seront dans un dossier « pos », les commentaires négatifs dans un dossier « neg ».

Librairies à installer ou à importer dans le code

Pour pouvoir lancer ce code, il faut importer les modules suivants :

- Le module *os* : ce module est un module standard de Python, il n'est pas nécessaire de l'installer
- La librairie *csv* : cette librairie est normalement déjà installée lors de l'étape 1 (crawler)

Comment ça marche

Ce code prend en entrée un fichier .csv contenant les données traitées dans l'étape 2. Pour rappel, les fichiers d'entrées peuvent être :

- Les commentaires sous forme d'unigram
- Les commentaires sous forme d'unigram sans mots outils
- Les commentaires sous forme de bigram
- Les commentaires sous forme de bigram sans mots outils
- Les commentaires sous forme de trigram
- Les commentaires sous forme de trigram sans mots outils

Ce code va d'abord créer 2 dossiers « neg » et « pos ». C'est dans ces fichiers que nous trouverons les données d'apprentissage et de test.
Ensuite, il va créer les fichiers en question. Si la note du commentaire est strictement supérieure à 30, l'algorithme placera le commentaire dans un fichier qui sera dans le dossier « pos » sinon le commentaire sera placé dans le dossier « neg ».
Une nomenclature spécifique a été attribuée aux fichiers :

cv[000-999].txt

J'ai décidé de limiter la création de fichiers à 1000 fichiers par dossier.

Problèmes rencontrés

Je n'ai pas eu de problèmes spécifiques dans cette partie.

Défauts de ce code

- Dans ce code, nous retrouvons une méthode pour créer les fichiers positifs et une autre pour créer les fichiers négatifs. Ce qui n'est pas optimal

Potentielles améliorations

Fusionner les deux méthodes de création de fichiers pourrait rendre le code plus robuste et plus élégant.

Fichier .log

Dans ce code, en plus de la création de fichiers, un fichier .log par dossier a été créé afin d'enregistrer les informations suivantes :

- Le nombre de fichiers dans le dossier
- Le nombre de mots par fichier

L'apprentissage machine

But de cette étape

Cette étape est l'étape la plus importante de notre projet. C'est ici que toute la magie se fait. Le but de cette étape est d'effectuer l'analyse de sentiments sur notre base de données. En premier temps l'algorithme va apprendre grâce aux données d'apprentissage, en second temps, cet apprentissage sera testé sur les données de test et finalement, nous obtiendrons des résultats sur les données de test sous forme de trois mesures de performance :

- Le rappel : il permet de répondre à la question suivante « Quelle proportion de résultats positifs réels a été identifiée correctement ? ». La formule de calcul est la suivante :

  $R = \frac{VP}{VP+FN}$ où VP : vrai positif et FN : faux négatif
- La précision : elle permet de répondre à la question suivante : « *Quelle proportion d'identifications positives était effectivement correcte ?* ». Sa formule est la suivante :

  $P = \frac{VP}{VP+FP}$ où VP : vrai positif et FP = faux positif

Petit bonus :

	Classe prédite OUI	Classe prédite NON
Classe présentée OUI	Vrai Positif Le système trouve **à raison** le message comme **appartenant** à la classe	Faux Négatif Le système trouve **à tort** le message comme **n'appartenant pas** à la classe
Classe présentée NON	Faux Positif Le système trouve **à tort** le message comme **appartenant** à la classe	Vrai Négatif Le système trouve **à raison** le message comme **n'appartenant pas** à la classe

Tableau 1 : métrique d'évaluation des résultats de l'outil d'apprentissage machine

- La F-mesure : c'est la moyenne harmonique de la précision et du rappel. Elle permet de mesurer la capacité du système à donner toutes les solutions pertinentes et à refuser les autres. Sa formule est la suivante :

  $F = \frac{2\,P\,.\,R}{P+R}$ où P : précision et R : rappel

Librairies à installer ou à importer dans le code

Afin de pouvoir lancer le code d'apprentissage machine il faut avoir les librairies et modules suivants :

- Scikit Learn : c'est une librairie pour l'apprentissage machine sur Python.
  commande d'installation : *pip install -U scikit-learn*
  Toutefois, il faut noter que cette libraire a besoin de :
  - Python (>= 3.4)
  - NumPy (> 1.8.2)
    - Commande d'installation : *pip install numpy*
  - SciPy (>= 0.13.3)
    - Commande d'installation : *pip install scipy*

  Quelques informations sur cette librairie :
  Cette bibliothèque comprend des fonctions pour effectuer des tâches de classification avec différents types d'algorithmes, des tâches de régression, des tâches d'extraction de paramètres et beaucoup d'autres.
  La documentation se trouve sur le site suivant : https://scikit-learn.org/stable/index.html
- Le module sys : ce module est un module standard de Python, il n'est pas nécessaire de l'installer
- Le module time : ce module est un module standard de Python, il n'est pas nécessaire de l'installer
- Le module os : ce module est un module standard de Python, il n'est pas nécessaire de l'installer

Comment ça marche

Avant de commencer à expliquer comment fonctionne l'algorithme, il faut bien évidemment créditer la personne qui nous a permis d'avoir le code. Nous avons récupérer l'algorithme d'analyse de sentiments de *Marco Bonzanini* utilisant la librairie scikit Learn. Nous l'avons ensuite adapté pour qu'il réponde à nos besoins.

Ce code se lance à partir du terminal avec la commande suivante :

python classif.py [nom du dossier contenant les deux autres dossiers « pos » et « neg »]

Cet algorithme prend en entrée le contenu des dossiers « pos » et « neg » généré grâce au code de l'étape 3 (la création de dossiers).
Il va ensuite créer 2 listes, une liste qui contiendra les données d'apprentissage et une autre qui contiendra les données pour la phase de test.
Le contenu des fichiers cv000 à cv899 servira à l'apprentissage tandis que le contenu des fichiers cv900 à cv999 servira au test.
Ensuite, l'algorithme va effectuer la tâche d'extraction des paramètres en utilisant la vectorisation tf.idf (abréviation de **term frequency–inverse document frequency**). Cette mesure permet de refléter l'importance d'un mot à un document dans une collection de documents ou un corpus. Il est souvent utilisé comme facteur de pondération dans les recherches sur la récupération d'informations, l'exploration de texte et la modélisation d'utilisateurs. La valeur tf.idf augmente proportionnellement au nombre d'apparitions d'un mot dans le document et est compensée par le nombre de documents du corpus contenant le mot, ce qui permet de prendre en compte le fait que certains mots apparaissent plus souvent en général.
Nous allons ensuite créer le vocabulaire et calculer le poids des paramètres pour les données d'apprentissage et calculLé uniquement les poids des paramètres pour les données de test en utilisant le même vocabulaire que celui des données d'apprentissage.
Après avoir effectué l'extraction des paramètres, nous commençons la tâche de classification. Comme expliqué dans l'introduction, notre algorithme utilise deux variations des séparateurs à vaste marge :

- SVM à noyau RBF
- SVM à noyau linéaire

Nous calculons également le temps nécessaire pour effectuer l'apprentissage et le test.
En sortie, nous obtiendrons les valeurs de trois mesures de performance (par sentiments ~pos/neg~et moyenne des deux) :

- La précision
- Le rappel
- La F-mesure

Problèmes rencontrés

Avant d'utiliser cet algorithme pour effectuer la tâche d'analyse de sentiments, nous avions développé un algorithme utilisant la bibliothèque *NLTK*. Nous l'avons fait tourner sur nos données, mais ce dernier s'est avéré inefficace pour différentes raisons :

- *NLTK* est une librairie qui fonctionne parfaitement avec l'anglais et est très mauvaise avec des données en français.
- L'algorithme n'était pas performant, même en testant sur des données anglaises, nous obtenons des résultats incorrects (des phrases classifiées comme positives alors qu'elle ne contenait aucun avis positif)

Suite à tous ces échecs, nous avons décidé d'abandonner cet algorithme et d'utiliser la librairie *scikit-Learn* pour effectuer la dernière étape de notre projet.

Nous avons rencontré un autre problème, beaucoup plus mineur. Le projet a été effectué sur Mac et à chaque lancement de code, le logiciel créait des fichiers .DS STORE.
Lors du lancement du code d'apprentissage machine, un fichier .DS STORE n'était pas encodé en UTF-8, ce qui faisait planté l'algorithme.
Afin de résoudre ce problème, nous avons ajouté une condition au code qui permet de ne prendre que les fichiers ayant l'extension .txt

Fichier .log

Un fichier log a été créé par type de données d'entrée (unigram, bigram, trigram…) afin d'enregistrer les valeurs des mesures de performances ainsi que le temps de calcul.

Hypothèse

Avant d'obtenir les résultats, nous supposons que les valeurs des mesures de performances résultant de l'application de l'algorithme sur les fichiers bigram seront les meilleures. Nous considérons qu'un modèle bigram est un des modèles les plus performants, car ce dernier permet de prendre en compte l'ordre des mots dans la phrase.
Afin de savoir si notre hypothèse est correcte, nous allons exploiter les résultats dans la partie suivante.

Les résultats

Afin d'observer les résultats de notre programme d'analyse de sentiments, nous avons traité les données obtenues dans les fichiers log qui se trouvent dans le dossier « fichiers_log_apprentissage ».

Avant de présenter les résultats, voici une brève comparaison des différents modèles SVM utilisés :

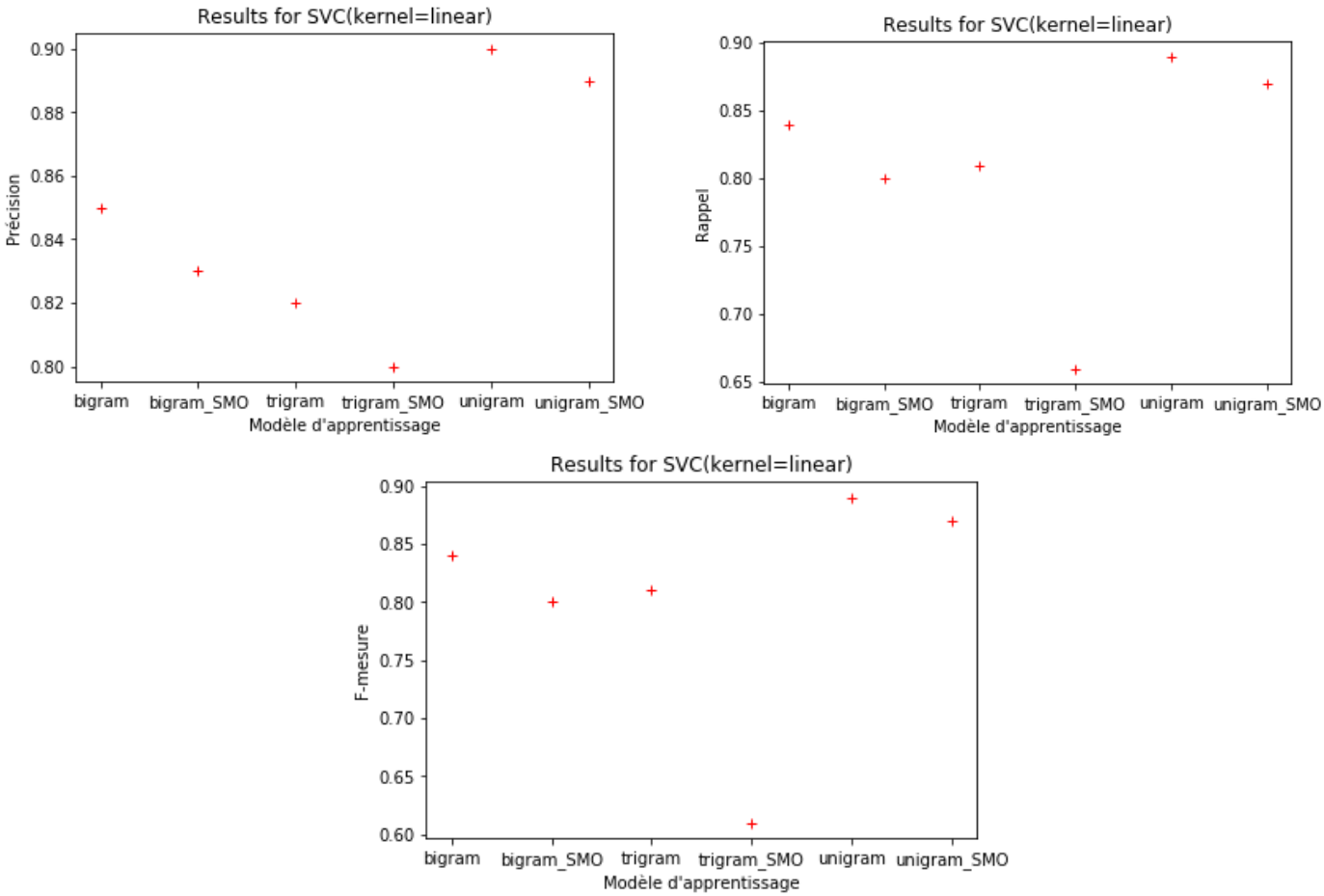


Figure 1 : les résultats des 3 mesures pour le modèle SVM au noyau linéaire

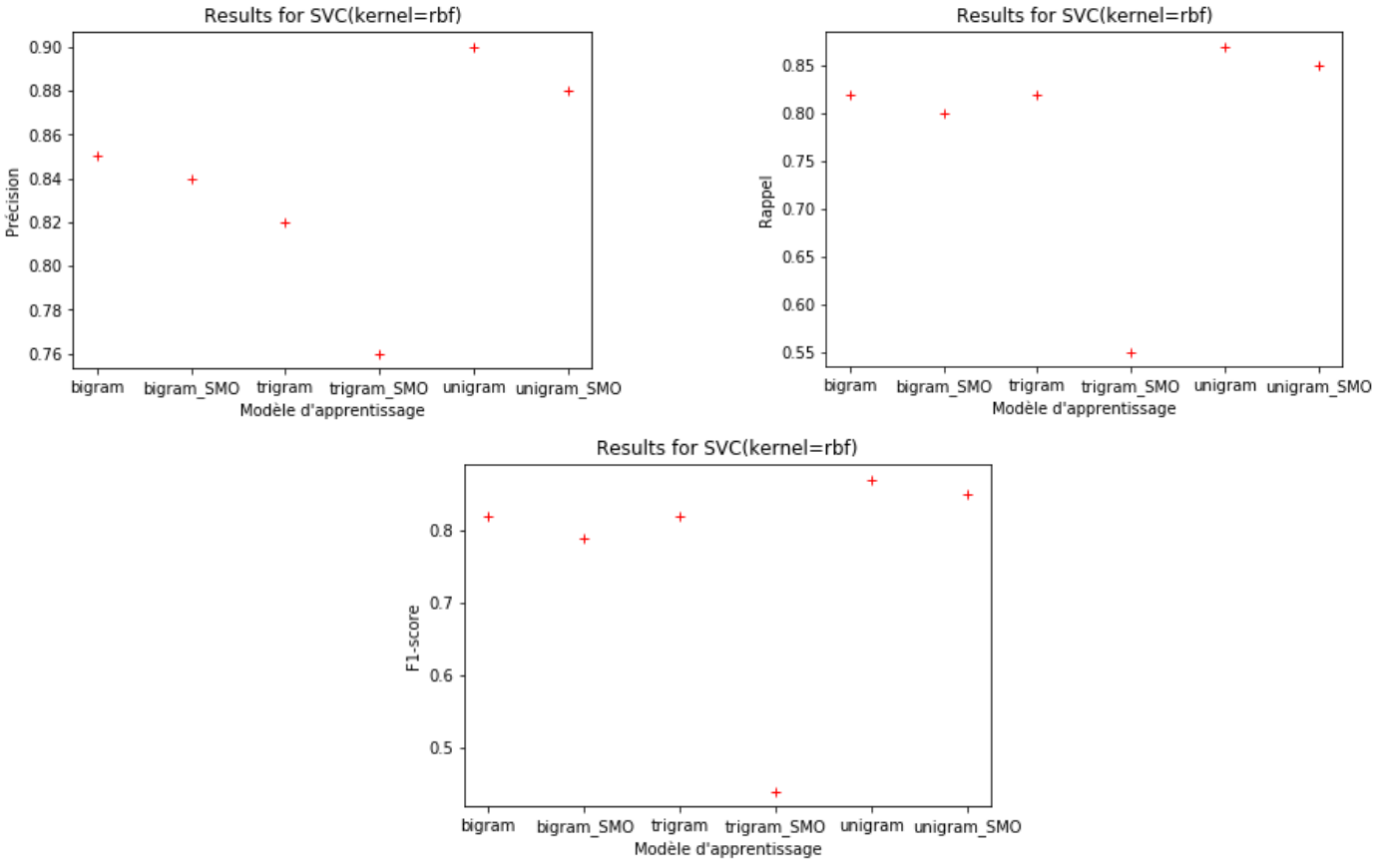


Figure 2 : les résultats des 3 mesures pour le modèle SVM au noyau RBF

Comme nous pouvons le constater en observant les Figures 1 et 2, les valeurs des 3 mesures de performance sont similaires dans les 2 modèles SVM et pour n'importe quel type de données.
Nous pouvons conclure que l'utilisation de SVM à noyau RBF ou à noyau linéaire n'a à priori aucun impact sur notre analyse.

Peu importe le modèle SVM, le modèle unigram a obtenu les meilleurs résultats pour les trois mesures de performance (précision, rappel et f-mesure). Les résultats sont les suivants :

- Précision : ~ 0.90
- Rappel : ~ 0.87
- F-mesure : ~ 0.87

Le modèle trigram sans mots outils quant à lui, a obtenu les moins bons résultats :

- Précision : ~ 0.76
- Rappel : ~ 0.55
- F-mesure : ~ 0.44

Nous n'avons pas remarqué une différence significative entre les résultats du modèle unigram et ceux du modèle bigram. Cependant, une différence notable existe entre les modèles unigram et trigram, ainsi que les modèles bigram et trigram.

De plus, la suppression de mots outils ne semblait pas impacter les résultats, comme nous pouvons le constater avec les résultats suivants :

	Modèle unigram	Modèle unigram sans mots outils
Précision	0.90	0.84
Rappel	0.87	0.80
F-mesure	0.87	0.79

Finalement, contrairement à l'hypothèse émise initialement, le modèle unigram était plus performant que le modèle bigram.

Les potentielles améliorations

Concernant les améliorations possibles à ce projet, nous proposons :

- Ajouter une classe « neutre » dans l'analyse
- Contextualisation des commentaires : quels sont les sujets qui sont le plus souvent mal notés et quels sont les sujets fréquents très bien notés.
- Lemmatisation des données afin de réduire les redondances dans les données à traiter.
- Étendre le traitement à un modèle multilingue.

Toutes ces améliorations peuvent faire l'objet d'un travail à part entière.

Les ressources

- SVM : Machines à Vecteurs de Support ou Séparateurs à Vastes Marges, Mohamadally Hasan (2006)
- Cours de Machine Learning, Ringeval F. (2018)
- https://marcobonzanini.com/2015/01/19/sentiment-analysis-with-python-and-scikit-learn/