Rapport du projet CNAM-RCP216 - Analyse de sentiment sur des commentaires de films

Jean Luc LAFENETRE <jl.lafenetre@gmail.com> Fabrice DUNAN <fabrice.dunan@laposte.net>

Table des matières

https://gitlab.com/logrus_fr/CNAM-RCP216

Jean Luc LAFENETRE <jl.lafenetre@gmail.com> Fabrice DUNAN <fabrice.dunan@laposte.net>

v1, 2018-07-06

tuteurs: M.CRUCIANU R.FOURNIER

1. Présentation du problème de données

Le problème de données posé est de donner une note à un film à partir d'un commentaire à son propos.

Intitulé du sujet n°9 : "Analyse de sentiment 2"

```
"l'objectif est de prédire la note (entre 1 et 10) à partir du texte de la revue d'un
film, pour les données issues de la base de données IMDB (50000 revues sur des films,
données accessibles sur http://ai.stanford.edu/~amaas/data/sentiment/) ; découper l
'ensemble de données en données d'apprentissage et données de test."
```

2. Présentation du jeu de données

On dispose pour ce faire de données issues d'un site internet contenant de nombreux commentaires et des notes attribuées correspondantes.
Les données sont directement ou indirectement issues du site internet **IMBD.com** spécialisé dans le cinéma.

Le jeu de données fourni pour cette étude contient :

- des données brutes issues du site : Les commentaires des utilisateurs du site sur des films
- des données dérivées calculées à partir des précédentes par les fournisseurs du jeu de données
- d'un fichier expliquant le contenu du jeu de données

La sources du jeu de données

Le jeu de données peut être trouvé ici : http://ai.stanford.edu/~amaas/data/sentiment/

On verra ci après les informations décrivant le *Dataset* mis à disposition.

2.1. Information sur le contenu de l'archive *"aclImdb"*

La phase de collecte de la donnée est simplifiée par la présence d'un fichier README décrivant données brutes et données dérivées.

Les informations ci-après sont issues de :

- la traduction de ce fichier
- d'investigations rapides à l'aide d'outils simples

2.2. Les données brutes d'apprentissage et de test

Ce sont les commentaires émis par des internautes à propos de films. Certains sont fournis notés, d'autres non.

Informations générales issues du README

- D'après de README, le *Dataset* ne fournit pas plus de 30 commentaires par film car sinon ceux-ci sont très corrélés. Nous n'avons pas vérifié ces assertions.

2.2.1. Forme commune aux fichiers de commentaires

- Motif du nom des fichiers commentaires

```
<indexFilm>_<note>.txt
```

- Les commentaires sont en langue anglaise

2.2.2. Fichiers commentaires notés

Les fichiers d'apprentissage(train) et de test

- dont la note est comprise entre 1 et 4
  ⇒ fournis dans le répertoire *"neg"*
- dont la note est 5 ou 6
  ⇒ non fournis, en découle un problème traité ci-après
- dont la note est comprise entre 7 et 10
  ⇒ fournis dans le répertoire *"pos"*

Nombre total : **50000**

```
22:03:46-user@M40474:~/Bureau/CNAM/git/CNAM-RCP216/donnees/aclImdb/train(feature/FDU)$
find neg/ |wc -l
12501
22:04:04-user@M40474:~/Bureau/CNAM/git/CNAM-RCP216/donnees/aclImdb/train(feature/FDU)$
find pos |wc -l
12501
22:20:58-user@M40474:~/Bureau/CNAM/git/CNAM-RCP216/donnees/aclImdb/test(feature/FDU)$
find pos |wc -l
12501
22:21:01-user@M40474:~/Bureau/CNAM/git/CNAM-RCP216/donnees/aclImdb/test(feature/FDU)$
find neg/ |wc -l
12501
```

On exploitera ces données lors de l'élaboration et de l'évaluation des différents moyens de répondre à la problématique de donnée.

2.2.3. Fichiers commentaires non notés

Ces fichiers suivent le même modèle que les précédents à ceci près que leur note vaut 0.
Ils ne sont présents que dans le répertoire *"train"*.

Nombre total : **50000**

```
21:30:44-user@M40474:~/Bureau/CNAM/git/CNAM-RCP216/donnees/aclImdb/train(feature/FDU)$
find unsup/|wc -l
50001
```

On utilisera ces données pour d'éventuels tests "réels", avec nul autre moyen que le jugement humain de la lecture du commentaire pour comparer le résultat du medium d'analyse de sentiment.

2.3. Les données dérivées communes

2.3.1. Fichier de la "polarité des mots"

Le fichier *imdbEr.txt* contient la "polarité" de chaque fichier vocabulaire. Ce calcul a été effectué par les fournisseurs du jeu de données pour leur étude Potts, 2011.

La "polarité" du mot donne une indication quantitative quant à la tonalité positive ou négative que son emploi va donner dans un commentaire.

2.3.2. Fichier inventaire du vocabulaire

Le lexique des commentaires est fourni sous la forme d'un fichier texte qui contient tous les mots présents dans les commentaires (imdb.vocab).

Il tient lieu d'index pour les fichiers :

- SVM
- polarité des mots

description

1 mot par ligne, le numéro de la ligne donnant la valeur de son index.

nombre de mots distincts **89526**

```
19:00:31-user@M40474:~/Bureau/CNAM/git/CNAM-RCP216/donnees/aclImdb(feature/FDU)$ wc -l
imdb.vocab
89526 imdb.vocab
```

Ce fichier comprend tout le vocabulaire des commentaires dont les termes dits *"stop words"*.

2.4. Les données dérivées d'apprentissage et de test

2.4.1. Fichiers des commentaires tokenisés au format SVM

Des fichiers (*labeledBow.feat, unsupBow.feat*) contenant les commentaires tokenisés, leur fréquence et la note du commentaire en premier champ sont disponibles. Là encore, le travail a été effectué par les fournisseurs du jeu de données pour leur étude.

grammaire du fichier

(num ligne = n°commentaire) mot_id_0:TF mot_id_1:TF…

- relation commentaire / mots et nombres mots / note donnée
- *".feat"* format pseudo libSVM

le mot de la ième ligne dans le fichier .vocab aura un tag i dans chaque ligne du fichier .feat. Chaque ligne représentant un commentaire, le tag peut ne pas apparaitre si ce mot n'est pas dans le vocabulaire utilisé dans le commentaire.

exemple (première ligne donc index = 0)

```
10 0:7 1:4 2:2 3:5 4:5
```

vérification

le fichier contient bien les notes positives et négatives

problème de format de "tag"

la norme SVM (ou l'implémentation Spark ?) impose que le premier indice vale 1 or dans le fichier il est égal à 0 !
Il est nécessaire d'incrémenter les index…sachant qu après import l'index sera décrémenté…

On a donc élaboré des outils écrits ex nihilo et récupérés de scripts de conversion pour pouvoir exploiter ces fichiers.

2.4.2. Fichiers des URL des films

Un fichier regroupe les URL des films correspondants aux commentaires présents pour chaque jeu (train/test) de données (labellisées négatives, positives ou non labellisées).
Le fichier README indique que les liens vers les commentaires sont éphémères dans imdb.com.
Il suffit d'enlever `/usercomments` à la fin de l'URL pour avoir accès au film.

Exemple d'URL présente dans le fichier (HTTP 404)

```
http://www.imdb.com/title/tt0893406/usercomments
```

URL du film correspondant

```
http://www.imdb.com/title/tt0893406
```

3. Objectif de l’analyse de données massives

Forme de la matrice des données labellisées

On déduit de ce qui précède que la matrice termes / documents, qui va représenter la transformation vectorielle des données brutes, sera approximativement de la forme 90000 x 50000 ce qui est important.

On appliquera donc une méthode appropriée pour traiter cette matrice afin d’atteindre les résultats escomptés.

La démarche de l’étude

On va appliquer deux méthodes au moins pour résoudre le problème de données. Cela nous permettra de comparer leurs résultats.

De plus, les données présentes dans le dataset, commentaires bruts et fichiers SVM, sont redondantes. On tentera d’exploiter l’intégralité du jeu de donnée afin d’en évaluer la qualité.

Les deux méthodes

1. Modélisation
   Cette première approche s’appuie directement sur l’exploitation des données dérivées fournies dans la dataset. On ne remet pas en question les traitements effectués pour aboutir au fichier SVM et au vocabulaire.
   L’objectif est de tenter de trouver un modèle statistique de classification et d’évaluer sa précision.
2. Similarité
   Dans cette partie, on procédera
   a. à l’application des préalables classiques en analyse de texte
   b. à la réduction la dimension des mots en concepts par LSA
   c. à la recherche de la note du film commenté en calculant la *similarité* du commentaire à d’autres commentaires. On déduira la note du commentaire en calculant le barycentre des 9 notes des commentaires les plus *similaires*.

4. Description des démarches mises en oeuvre

Ce chapitre décrit, en détail, les méthodes, outils et développements utilisés pour résoudre le problème de données massives.

4.1. Problème de l'absence des données labellisées "neutre"

On remarque que le jeu de données ne fournit pas de données notées 5 ou 6. Comment donc prédire les notes 5 6 des films ?

On ignorera ce problème même si une brève étude et des pistes d'améliorations sont mentionnées dans le chapitre "Pour aller plus loin".

4.2. Modélisations sur fichier SVM : Description et résultats

Dans cette première approche, l'objectif est d'utiliser les fichiers suivants, fournis dans le jeu de données :

- fichier de commentaires tokenisés (format libSVM)
- fichier d'inventaire du vocabulaire
- fichier de polarité des mots

On cherche à déterminer un ou plusieurs modèles prédictifs de la note d'un film à partir du commentaire, et à comparer leur précision.

4.2.1. Apprentissage du modèle sur toutes les données

Le problème posé est celui d'une prédiction multi-classes, ce qui oriente vers le choix d'un modèle statistique possible, et disponible dans Mlib :

- Régression logistique
- Arbre décisionnel
- Forêt aléatoire
- Bayesien naïf
- Perceptron multicouche

Le premier choix se porte sur la mise en place d'une forêt aléatoire car ce type de modèle fournit habituellement de bons résultats et reste simple à mettre en place.

Le premier test porte sur un échantillon de 200 lignes du fichier libSVM (obtenu par tirage aléatoire sans remise). Cet échantillon est utilisé pour la construction d'un modèle avec 10 arbres, dont on

évalue la précision sur un autre échantillon de 200 lignes (obtenu dans les mêmes conditions). Les différents calculs s'effectuent sans problème et en quelques secondes sur un ordinateur de bureau, et indiquent une précision de 0,214.
Cette valeur modeste n'est sûrement pas représentative compte tenu du très faible échantillon utilisé, mais l'objectif est de valider le code Scala et d'obtenir une valeur de précision de référence pour comparer les modèles.

Les tests suivants consistent à augmenter progressivement la taille de l'échantillon afin d'observer l'évolution du temps de traitement et de la précision du modèle obtenu. Cependant, dès que la taille de l'échantillon augmente au-delà de quelques centaines de lignes (soit moins de 10% des données), le modèle ne peut pas être calculé sur notre poste de travail par manque de ressources mémoire.

Face à ce problème, nous avons testé un autre classifieur pour voir si ce problème de passage à l'échelle se pose.

Le seconde modèle mis en place se base sur une classification bayésienne naïve. De la même façon, le code est validé sur des échantillons de données de taille croissante, et on ne rencontre pas le problème de ressource mémoire précédent, y compris lors de l'utilisation des 50 000 lignes de données, 70% étant utilisé pour l'apprentissage, et le reste pour la validation.
Cependant, la précision du modèle résultant ne dépasse pas 0,068…
Autant dire que le modèle obtenu n'a pas vraiment d'intérêt.

4.2.2. Réduction du vocabulaire

Face à ce problème de passage à l'échelle qui ne permet pas de déterminer un modèle prédictif sur ce volume de données, nous avons envisagé de réduire le nombre de variables.

En effet, comme vu précédemment, l'ensemble des données correspond à une matrice 90000 mots x 50000 documents. Dans les premiers tests, nous avons cherché à réduire le nombre de documents par échantillonnage, mais avec peu de succès.

Une autre approche serait donc de réduire le nombre de mots dans le vocabulaire. Pour ce faire, nous avons choisi d'utiliser le fichier de polarité des mots fourni dans le jeu de données. En effet, une lecture rapide du vocabulaire montre que les stop words n'ont pas été supprimés lors de la tokenisation des commentaires, ce qui conduit à un vocabulaire exhaustif mais sûrement trop complet pour notre étude.

Partant du principe que le fichier de polarité fournit pour chaque mot une évaluation quantitative (notée entre -4.5 et 4.5) de sa tonalité positive ou négative, nous avons imaginé de filtrer les données selon un seuil S variable :
. les mots dont la polarité est dans l'intervalle [-4.5 , -S] sont conservés.
. les mots dont la polarité est dans l'intervalle [+S , +4.5] sont conservés.
. les autres mots sont abandonnés (leur polarité variant entre -S et +S).

L'idée est de ne conserver que les mots les plus "positifs" et "négatifs", et de se débarrasser ainsi des stop words et des mots neutres.

Pour cela, nous utilisons un script qui filtre en premier lieu le vocabulaire en fonction du seuil, puis produit un jeu de données libSVM purgé des mots non retenus.

IMPORTANT	Il faut noter qu'une ligne de commentaire qui n'a plus de mots suite au filtrage du vocabulaire est elle même supprimée du jeu de données.

Il est intéressant d'étudier comment le nombre de mots et de commentaires évoluent en fonction du seuil :

Seuil	Vocabulaire conservé	Commentaires conservés
0.25	49 %	100%
0.5	37 %	99%
1	20 %	99%
1.5	11 %	88%
2	6%	59%
3	2%	8%
4	0%	2%

On constate que le vocabulaire réduit rapidement avec l'augmentation du seuil, alors que le nombre de documents réduit faiblement pour un seuil inférieur à 1, avant de diminuer rapidement.

En première approche, une valeur de seuil de 1 semble intéressante car elle réduit la taille du vocabulaire de 80% (en ne conservant que les mots les moins neutres) sans avoir d'impact sur le nombre de commentaires.

4.2.3. Apprentissage sur des données filtrées

Avec ce filtrage du vocabulaire, nous avons repris les tests de modélisation précédents, le seuil étant fixé à 1.

Random Forest

Un modèle de random forest sur un échantillon de 200 commentaires donne une précision de 0.207, donc légèrement infèrieure à celle obtenue avec tout le vocabulaire. Cela semble cohérent vu que l'on a réduit le vocabulaire et donc le nombre de variables du classifieur.

La question est de savoir si la réduction du nombre de variables est suffisante pour construire un modèle sur plus de données, et pour augmenter la précision de la prédiction. Malheureusement, on atteint rapidement les mêmes limites matérielles dès que l'on augmente la taille de l'échantillon, et il n'est pas possible de calculer un modèle avec plus d'un millier de commentaires.

Les tests effectués avec des valeurs de seuil plus élevées et le même classifieur n'aboutissent pas non plus au delà d'un millier de commentaires. Pour des échantillons de taille modeste, la précision du classifieur reste voisine des valeurs précédentes.

Classifieur Bayésien Naïf

Si l'on construit un classifieur Bayésien Naïf sur l'ensemble des données mais avec un vocabulaire réduit, on constate que la précision obtenue reste identique et voisine de 0,06. Il n'y a donc aucune

amélioration de la classification avec ce type de modèle.

Régression logistique et autres modèles

Face à ces résultats plutôt décevants, nous avons testé d'autres modèles de classifieurs. Les arbres de décision et perceptron rencontrent les mêmes problèmes de ressources que les random forest, et il ne nous a pas été possible de calculer des modèles de ce type.

Seul un modèle de régression logistique a pu être construit à partir du jeu de données complet. Nous avons procédé à plusieurs essais avec et sans filtrage du vocabulaire, en utilisant 70% des données pour l'apprentissage et 30 % pour la validation. La précision du modèle obtenu est indiquée dans le tableau suivant.

Seuil	Précision
Sans	0,34
0.25	0,32
0.5	0,29
1	0,28
1.5	0,32
2	0,34
2,5	0,39
3	0,24

On constate que la précision, tout en restant très modeste, est tout le temps supérieure aux valeurs précédemment obtenues. Son évolution en fonction du niveau de filtrage du vocabulaire est plus surprenante.
La précision diminue tout d'abord avec l'augmentation du filtrage, avant de remonter jusqu'à un maximum d'une valeur de seuil de 2,5. Pour information, ce seuil ne conserve que 3% du vocabulaire et 32% des commentaires. Le modèle obtenu est donc peu intéressant car il ne permet pas de classifier beaucoup de commentaires.

4.2.4. Conclusion sur la modélisation du fichier SVM

En conclusion de cette première partie, il ne nous a pas été possible de mettre en place un classifieur valable en utilisant directement les données fournies, tout du moins sans disposer d'une puissance de calcul supérieure. En effet, la tokénisation effectuée sur les commentaires fournit un trop grand nombre de termes, et le filtrage envisagé, sur la base de la polarité de ces termes, ne nous a pas permis de nous affranchir de ce problème d'échelle.

Face à ce constat, nous avons envisagé une autre approche, détaillée dans la suite de ce rapport.

4.3. LSA et Similarité : Description et résultats

Pour pouvoir prédire à partir d'un commentaire de film la note de ce film, l'idée est ici de comparer ce commentaire avec une base commentaires de films déjà notés.
Le calcul de sa *similarité* avec d'autres commentaires est une proposition de solution pour déduire la note du film.

On détaille ci-après la méthode qui est mise en place dans le script scala de prédiction de notes.

4.3.1. Utilisation du REPL scala

Pour exécuter le script scala, on a recours au REPL scala

```
18:24:36-user@M40474:~/Bureau/CNAM/git/CNAM-RCP216/code(feature/FDU)$ spark-shell
--driver-memory 1g --jars ./lsa-corrige.jar  --master local[*]
-Dscala.repl.maxprintstring=64000
```

IMPORTANT	Le jar fourni dans le TP est buggué. On a utilisé sa version corrigée. (Tiré d'un rapport CNAM RCP216, merci Denis !)

4.3.2. Constitution de la matrice documents

Dans cette première partie, on constitue la matrice documents. Avec Spark, on "charge" facilement les nombreux fichiers texte "commentaires" qui vont constituer les lignes du RDD.
L'instruction `wholeTextFiles` va permettre de conserver l'information du nom de fichier qui… contient la note !

On verra dans la suite les bénéfices de cette idée pour l'utilisation des bibliothèques déjà utilisées.

Le RDD est constitué des données notées, positives et négatives, qui constituent le corpus de référence. Grâce à la méthode `union` de Spark, le RDD contient également les données à prédire, notées si l'objectif est de tester, non notées dans un cas d'analyse de sentiment réel.

On verra dans la section résultats l'impact sur l'évaluation de la qualité de la similarité.

4.3.3. Tokenisation, suppression des mots vides, lemmatisation

En s'inspirant du TP CNAM sur la fouille de texte, on choisit d'utiliser les bibliothèques illustrées par Sandy Ryza dans son livre et fournies dans son dépot.
Celles-ci exploitent projet Stanford NLP pour la lemmatisation, fonction absente de Spark.

On obtient donc un tableau de couples (chemin fichier, tableau commentaires tokenises) utilisable à l'étape suivante.

4.3.4. Calcul de la matrice TF/ITF

Toujours avec l'interface "O'reilly", on calcule la matrice TF/ITF avec les données précédentes.

IMPORTANT	C’est dans cette phase que l’on réduit le nombre de termes/mots souhaites. On simplifie

Le nombre de termes sélectionné est **1000**. Cet "hyperparamètre" pourrait varier pour obtenir de meilleurs résultats prédictifs cf le chapitre dédié aux variantes

4.3.5. LSA : calcul de la matrice SVD

Pour réduire la dimension des variables "mots" en concepts, on va utiliser LSA (Latent Semantic Analysis). Cela passe par une décomposition de la matrice TF/ITF en SVD.

Spark nous permet de faire cette transformation via son objet `RowMatrix`.

IMPORTANT	Dans cette phase encore on va réduire la dimension aux k concepts les plus importants (les k valeurs propres avec le plus de variance).

Le nombre de concepts sélectionné est **200**. Cet "hyperparamètre" pourrait varier pour obtenir de meilleurs résultats prédictifs cf le chapitre dédié aux variantes

4.3.6. Similarité : calcul de la matrice "U.S" normalisée

Pour obtenir la similarité entre un document et l’ensemble des documents du corpus, la procédure est la suivante :

> La matrice U_k décrit chaque document par les k concepts trouvés par la SVD. Pour trouver des documents similaires à un autre document, on pondère chaque colonne de la matrice U_k par les valeurs singulières correspondantes (trouvées sur la diagonale de la matrice S_k) en faisant le produit U_k . S_k. Ensuite, on normalise chaque ligne du résultat à une norme de 1 et on calcule la similarité entre deux documents comme le cosinus des deux lignes correspondantes de la matrice normalisée.
>
> — TP CNAM fouille de texte

Là encore, "l’interface O’Reilly" nous permet d’effectuer ces calculs.

4.3.7. Similarité : Calcul des documents les plus similaires

IMPORTANT	Le code récupéré dans "l’interface O’Reilly" a une valeur codée en dur du nombre de documents similaires rendus. La ligne en question est : `allDocWeights.filter(!_._1.isNaN).top(10)`.

Le nombre de documents sélectionné est **10**. Cet "hyperparamètre" pourrait varier pour obtenir de meilleurs résultats prédictifs cf le chapitre dédié aux variantes

exemple des 10 documents les plus proches d'un document commentaire du set d'apprentissage

```
</home/user/Bureau/CNAM/git/CNAM-RCP216/donnees/aclImdb/train/neg/503_2.txt>
(file:/home/user/Bureau/CNAM/git/CNAM-
RCP216/donnees/aclImdb/train/neg/503_2.txt,1.0000000000000002),
(file:/home/user/Bureau/CNAM/git/CNAM-
RCP216/donnees/aclImdb/train/pos/5219_7.txt,0.5241450797170679),
(file:/home/user/Bureau/CNAM/git/CNAM-
RCP216/donnees/aclImdb/train/pos/5379_7.txt,0.516423382447652),
(file:/home/user/Bureau/CNAM/git/CNAM-
RCP216/donnees/aclImdb/train/neg/135_4.txt,0.5135557997396584),
(file:/home/user/Bureau/CNAM/git/CNAM-
RCP216/donnees/aclImdb/train/neg/8047_2.txt,0.512349848816831),
(file:/home/user/Bureau/CNAM/git/CNAM-
RCP216/donnees/aclImdb/train/neg/8038_4.txt,0.4943711342729597),
(file:/home/user/Bureau/CNAM/git/CNAM-
RCP216/donnees/aclImdb/train/neg/10205_2.txt,0.4890802779660287),
(file:/home/user/Bureau/CNAM/git/CNAM-
RCP216/donnees/aclImdb/train/neg/8736_3.txt,0.48837791337701864),
(file:/home/user/Bureau/CNAM/git/CNAM-
RCP216/donnees/aclImdb/train/neg/293_2.txt,0.48700647828562416),
(file:/home/user/Bureau/CNAM/git/CNAM-
RCP216/donnees/aclImdb/train/pos/5085_7.txt,0.4848006844200967)
```

4.3.8. Similarité : Déduire la note

Pour un document-commentaire dont on veut prédire la note, on propose d'effectuer la moyenne pondérée par la similarité inter-documents des notes des 10 (en réalité 9, puisque le document lui même fait partie du corpus) documents les plus similaires.

note prédite = $\sum_{i=1}^{n}$ **(taux de similarité**$_i$ * **note commentaire**$_i$**)** /$\sum_{i=1}^{n}$ **(taux de similarité)**$_i$

Avec les données de test qui possèdent une note, on peut avoir un aperçu de la précision de l'algorithme :

Extrait de l'exécution du script sur des données de tests aléatoires

```
--------------------------------------------------------------------------------
La note prédite par similarité de </home/user/Bureau/CNAM/git/CNAM-
RCP216/donnees/aclImdb/train/neg/506_2.txt> est ④
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
La note prédite par similarité de </home/user/Bureau/CNAM/git/CNAM-
RCP216/donnees/aclImdb/train/neg/502_1.txt> est ④
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
La note prédite par similarité de </home/user/Bureau/CNAM/git/CNAM-
RCP216/donnees/aclImdb/train/neg/501_2.txt> est ③
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
La note prédite par similarité de </home/user/Bureau/CNAM/git/CNAM-
RCP216/donnees/aclImdb/train/neg/505_2.txt> est ④
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
La note prédite par similarité de </home/user/Bureau/CNAM/git/CNAM-
RCP216/donnees/aclImdb/train/neg/507_4.txt> est ⑤
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
La note prédite par similarité de </home/user/Bureau/CNAM/git/CNAM-
RCP216/donnees/aclImdb/train/neg/503_2.txt> est ④
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
La note prédite par similarité de </home/user/Bureau/CNAM/git/CNAM-
RCP216/donnees/aclImdb/train/neg/504_3.txt> est ⑥
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
La note prédite par similarité de </home/user/Bureau/CNAM/git/CNAM-
RCP216/donnees/aclImdb/train/neg/500_2.txt> est ④
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
La note prédite par similarité de </home/user/Bureau/CNAM/git/CNAM-
RCP216/donnees/aclImdb/train/neg/509_3.txt> est ⑥
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
La note prédite par similarité de </home/user/Bureau/CNAM/git/CNAM-
RCP216/donnees/aclImdb/train/neg/508_3.txt> est ⑥
--------------------------------------------------------------------------------
```

On a:

- le nom du fichier contenant le commentaire
  - dans son nom la note attribuée par l'auteur du commentaire
- la note prédite arrondie à l'entier le plus proche
- la listes des fichiers du corpus dont les commentaires sont les plus proches au sens *LSA* du

commentaire dont on veut prédire la note.

On remarque que les résultats sont approchants mais pas parfaits ce qui nous mène à nous interroger sur la performance de l'algorithme sur tout un ensemble de tests.

4.3.9. Mesure performances de prédiction

L'objectif ici est de mesurer à quel point la prédiction par similarité de la note du commentaire de film approche la note réellement attribuée.

Avec une méthode différente mais pour obtenir les mêmes résultats plus clairement comparables pour 100 fichiers pour lesquels on a calculé la similarité, on obtient le résultat suivant:

comparaison des notes prédites et attribuées à un commentaire du dataset

```
scala> val sqlDF = spark.sql("select fichier,ROUND(note_label),ROUND(note_pred) from
notes where note_label=ROUND(note_pred)").show
+--------------------+--------------------+-------------------+
|             fichier|round(note_label, 0)|round(note_pred, 0)|
+--------------------+--------------------+-------------------+
|/home/user/Bureau...|                 3.0|                3.0|
|/home/user/Bureau...|                 2.0|                2.0|
|/home/user/Bureau...|                 4.0|                4.0|
|/home/user/Bureau...|                 3.0|                3.0|
|/home/user/Bureau...|                 4.0|                4.0|
|/home/user/Bureau...|                 1.0|                1.0|
|/home/user/Bureau...|                 3.0|                3.0|
|/home/user/Bureau...|                 2.0|                2.0|
|/home/user/Bureau...|                 4.0|                4.0|
|/home/user/Bureau...|                 3.0|                3.0|
|/home/user/Bureau...|                 4.0|                4.0|
|/home/user/Bureau...|                 3.0|                3.0|
|/home/user/Bureau...|                 1.0|                1.0|
|/home/user/Bureau...|                 2.0|                2.0|
|/home/user/Bureau...|                 1.0|                1.0|
|/home/user/Bureau...|                 3.0|                3.0|
+--------------------+--------------------+-------------------+

scala> sqlDF.count()
res59: Long = 16
```

Vu le faible nombre de résultats de prédiction exactes, on choisit d'évaluer une proportion de notes prédites dans un intervalle autour de la note réelle.

On réalise donc, comme pour les modèles, une évaluation de la performance de la similarité en utilisant les commentaires des jeux d'apprentissage et de tests déjà notés. On ne fait dans ce cas pas de distinction entre les deux puisqu'il n'a pas de réelle phase d'apprentissage de modèle.
En l'absence de réelle méthode pour la mesure de performance de similarité, on se contentera d'évaluer la performance par les éléments ci dessous :

- Par Spark
  Par le code suivant après avoir construit un *Dataframe* ayant pour colonnes la note "label" et la note prédite :

```
val evaluator = new
MulticlassClassificationEvaluator().setLabelCol("note_label").setPredictionCol("note_p
red").setMetricName("accuracy")
```

- Par un calcul élémentaire de taux de bonnes notes sur les i documents similaires :

```
                nombre de notes predites egales ou approchant la valeur label
note prédite = ------------------------------------------------------------
                                nombre total de commentaires
```

Exemple de calcul de taux de prédictions à un point de la note

```
scala> val sqlDF = 1.0 * spark.sql("select * from notes where
ROUND(note_pred)=note_label OR ROUND(note_pred)=note_label+1 OR
ROUND(note_pred)=note_label-1").count/df.count
sqlDF: Double = 0.4666666666666667
```

On notera T_0 le taux de bonnes notes attribuées à un commentaire, T_1 le taux de notes se situant au plus à un point de la note réelle (donc comprenant l'effectif précédent) et T_2 le taux de notes se situant au plus à deux points de la note réelle.
On se cantonne à cet écart en estimant qu'au delà de 2 points d'écart, la similarité n'est plus efficace pour la prédiction.

Autrement écrit :

T_1 = (nombre de commentaires tel que note réelle -1 =< notes prédites =< note réelle +1)/nombre de commentaires"

T_2 = (nombre de commentaires tel que note réelle -2 =< notes prédites =< note réelle +2)/nombre de commentaires

Cardinal ensemble de validation	**précision Spark**	**T0**	**T1**	**T2**
40	0,050	0,050	0,400	0,600
360	0 119	0,119	0,402	0,641
3600	0,141	0,141	0,412	0,644
50000	*	*	*	*

(*) = Résultat non obtenu pour cause d'erreur sur un temps de traitement très long.

4.3.10. Résultats réels

En phase de tests, il suffit de modifier l'expression régulière qui va filtrer les fichiers sur lesquels tester la validité de la similarité pour utiliser les fichiers non notés et ainsi réaliser une prédiction réelle.

Par contre l'appréciation de la prédiction ne peut se faire :

- idéalement par l'auteur du commentaire
- en recherchant la note sur le site
- avec une appréciation humaine a posteriori de la lecture du commentaire

Les optimisations du script qui réalise la prédiction par similarité prévues mais non réalisées sont listées ici.

5. Conclusion

Les différentes tentatives de détermination d'un classifieur n'ont pas abouti à un résultat satisfaisant, quelque soient les modèles statistiques envisagés. La précision reste faible pour les modèles que nous avons réussi à appliquer, et on se heurte à un problème de passage à l'échelle avec de nombreux modèles. Pour ces deux raisons, l'usage direct des données fournies ne nous est pas forcément apparu comme la meilleure solution.

Il était néanmoins intéressant de comparer les techniques dédiées à la fouille de texte à des méthodes classiques.

La méthode de prédiction par similarité a obtenu des résultats quant à la prédiction des notes pour les films. Si la capacité de prédiction des résultats exacts est faible, la prédiction de résultats approximatifs que l'on a jugé "acceptables" (différentiel de 20% sur la note) est supérieure à 50%.

A l'occasion de la tentative de résolution de ce problème de données, on a donc élaboré un système d'evaluation de similarité approximative. Dans ce registre, nous avons "vérifié" les résultats de l'API Spark pour ce qui est de la précision. Au stade de nos recherches, l'API n'a pas été de grande aide pour mesurer la performance approximative par similarité.

Par contre les résultats apar similarité uraient sans doute pu être améliorés si l'on avait pu faire varier les "hyperparamètres" du calcul de similarité, meme s'il aurait fallu rester attentif à l'impact négatif du "bruit".
On a donc a cette occasion, élaboré un système original de mesure de performance

6. Pour aller plus loin

Ci-dessous des pistes d'améliorations qui n'ont pu être traitées faute de temps.

6.1. Création des notes 5 et 6 à partir du *Dataset*

6.1.1. Ignorer l'absence de ces notes

1. Cas Modèle
   Cette solution n'est envisageable que si l'on accepte de n'avoir aucun document prédit avec la note 5 ou 6…
2. Cas similarité
   Si l'on applique une méthode de similarité, on pourra obtenir des commentaires de notes neutres si le document est similaire à la fois de documents négatifs et positifs. Néanmoins, l'absence de similarité à des documents neutres existant pénalisera forcément la précision.

6.1.2. Fusionner les catégories 5 et 6

Dans les cas de la modélisation, cela simplifie la prédiction au prix de la précision.

6.1.3. Par classification

1. Inclure d'éventuels commentaires non labellisés pour compléter la partie de Dataset notée.
2. Effectuer une *CAH* ou *k means avec k=10* et essayer de trouver 2 groupes dans les données non supervisées non présentes dans les données sup.
3. Valider par similarité le fait que ce groupe intermédiaire soit bien le groupe neutre.

6.2. Optimisations

- Rendre parallèle le code de calcul de la performance de la similarité (la méthode `preditNotes`. Une tentative a bien été faite mais elle s'est butée au temps et à l'énergie nécessaires pour rendre parallèle la méthode `RunLSA.topDocsForDoc` de l'interface.
- Faire varier num_terms, k concepts pour tenter d'améliore les performances de prédiction.
- Modifier une constante dans la méthode `RunLSA.topDocsForDoc` pour ramener plus de documents similaires.

6.3. Variantes des méthodes choisies

- Utiliser Spark pour calculer la matrice TF/ITF
  En effet, on s'appuie sur "l'implémentation O'Reilly" et le calcul de la matrice TF/ITF est la part la plus longue de l'exécution du script : ~10 minutes avec 1000 termes.

6.4. Pistes de méthodes alternatives

- Faire une AFD, la note étant la variable qualitative à 10 modalités.
  Le cours précise que les prérequis de l'AFD sont plus difficiles à atteindre que les méthodes que l'on a exposé.
- Etudier si le contexte des mots, donc l'utilisation de word2vec, peut améliorer les prédictions.

7. Annexes

7.1. Algorithme de suppression "stop words" dans les fichiers ".feat" fournis

Ci-dessous une ébauche de solution :
. A partir du fichier *.vocab*, identifier les numéros de lignes inutiles
.. manuellement, c'est impossible de dégager un vocabulaire sans erreur et dans un temps raisonnable. On a quand même près de 90000 mots...
.. Automatiquement cela reviendrait à croiser un fichier de "stop words" avec le fichier *.vocab*. Cela produit un tableau d'index de lignes.
. Charger le fichier *.feat*
. filtrer les indices inutiles de chaque document à l'aide de l'index

7.2. Script de filtrage du vocabulaire

```
#!/usr/bin/perl
# Filtrage des données
# Usage : filter_data.pl score
# Filtre le fichier libSVM pour supprimer les mots dont la note est comprise entre
-score et +score
# ATTENTION !!!
# Le filtrage doit être fait sur les fichiers libSVM avec le décalage d'index pour
commencer à 1 !!!
# JLL - Juin 2019

$vocabulaire='../donnees/aclImdb/imdbEr.txt';
$feat='../donnees/fixed_full_data.feat';
#$feat='../donnees/aclImdb/train/labeledBow.feat';
$outfile='../donnees/filtered_data.feat';

$note=$ARGV[0];
die "Usage : filter_data.pl score\n" unless ($note);

# Chargement du vocabulaire
open(VOCAB,"$vocabulaire")||die "Impossible d'ouvrir le fichier $vocabulaire\n";
for $score (<VOCAB>) {
    $nb ++; # On commence à indexer les mots à 1
    chomp($score);
    # On garde un tableau avec le numéro des mots valables
    $vocab{$nb}='1' if ($score <= -$note || $score >= $note);
}
close(VOCAB);
$retenu=scalar(keys %vocab);
$percent=int(100*$retenu/$nb);
print "Nombre de mots retenus : $retenu ($percent % du vocabulaire)\n";
```

```
# Filtrage du fichier LIBSVM
open(FEAT,"$feat")||die "Impossible d'ouvrir le fichier $feat\n";
open(OUT,">$outfile")||die "Impossible d'ouvrir le fichier $feat\n";
for (<FEAT>) {
    chomp;
    $nbfeat ++;
    @line=split /\s+/; # Découpage de la ligne du LIBSVM
    $label=shift @line; # Récupération du label en début de ligne
    $output='';
    for (@line) {
        ($mot,$count)=split /:/;
        $output.="$mot:$count " if (exists $vocab{$mot});
    }
    if ($output) { # S'il reste des mots, on écrit dans le fichier de sortie
        $output=~s/\s+$//;
        print OUT "$label $output\n";
        $nbout ++;
    }
}
close(FEAT);
close(OUT);

$percent=int(100*$nbout/$nbfeat);
print "Données de $feat filtrées dans $outfile ($nbout lignes soit $percent %)\n";
```

7.3. Script de création d'un classifieur random forest

```
import org.apache.spark.ml.classification.{RandomForestClassificationModel,
RandomForestClassifier}
import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator

val data = spark.read.format("libsvm").load("donnees/filtered_data.feat")

// Echantillonnage simple
val train = data.sample(false,0.005)
val test = data.sample(false,0.005)

// Détermination d'un modèle Random Forest
val rf = new RandomForestClassifier().setLabelCol("label").setFeaturesCol("features"
).setNumTrees(10)
val model = rf.fit(train)
val predictions = model.transform(test)

// Calul de la précision
val evaluator = new MulticlassClassificationEvaluator().
            setLabelCol("label").
            setPredictionCol("prediction").
            setMetricName("accuracy")
val accuracy = evaluator.evaluate(predictions)
```

7.4. Script de création d'un classifieur bayésien naïf

```
import org.apache.spark.ml.classification.NaiveBayes
import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator

// Chargement des données
val data = spark.read.format("libsvm").load("donnees/filtered_data.feat")

// Création des échantillons d'apprentissage et de validation
val Array(training, test) = data.randomSplit(Array(0.7, 0.3))

// Création d'un modèle de classifieur Bayésien Naïf
val model = new NaiveBayes().fit(training)
val predictions = model.transform(test)

// Calcul de la précision de prédiction par la méthode d'évaluation multi-classes
val evaluator = new MulticlassClassificationEvaluator().
        setLabelCol("label").
        setPredictionCol("prediction").
        setMetricName("accuracy")
val accuracy = evaluator.evaluate(predictions)
```

7.5. Script de prédiction de note par similarité

```
/**
  * F.DUNAN noteParSimilarite.scala
  *
  * usage:
  * lancer scala REPL : spark-shell --driver-memory 3g --jars ./lsa-corrige.jar
--master local[*] -Dscala.repl.maxprintstring=64000
  * taper : :load /home/user/Bureau/CNAM/git/CNAM-RCP216/code/noteParSimilarite.scala
  */

import java.io.File

/*
Pour utilisation bibliotheque "LSA" : stopwords + lemmatisation
*/
import com.cloudera.datascience.lsa._
import com.cloudera.datascience.lsa.ParseWikipedia._
import com.cloudera.datascience.lsa.RunLSA._

/*
?
 */
import org.apache.spark.mllib.linalg._
import org.apache.spark.mllib.linalg.distributed.RowMatrix

/*
algebre lineaire
 */
import breeze.linalg.{DenseMatrix => BDenseMatrix, DenseVector => BDenseVector,
SparseVector => BSparseVector}

/*
evaluer l'erreur du "modèle"
*/
import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator


/**
  * CONSTANTES
  */
val DISPLAY_NOTE = false

//nombre de termes plus importants à considerer
val numTerms = 1000

// nombre de valeurs singulières de la matrice SVD à garder
val k = 200

var STOP_WORDS_PATH = "/home/user/Bureau/CNAM/git/CNAM-
```

```
RCP216/code/src/main/resources/stopwords.txt"

var TRAIN_NEGATIVE_COMMENT_PATTERN = "/home/user/Bureau/CNAM/git/CNAM-
RCP216/donnees/aclImdb/train/neg/*.txt"
var TRAIN_POSITIVE_COMMENT_PATTERN = "/home/user/Bureau/CNAM/git/CNAM-
RCP216/donnees/aclImdb/train/pos/*.txt"
var TEST_NEGATIVE_COMMENT_PATTERN = "/home/user/Bureau/CNAM/git/CNAM-
RCP216/donnees/aclImdb/test/neg/*.txt"
var TEST_POSITIVE_COMMENT_PATTERN = "/home/user/Bureau/CNAM/git/CNAM-
RCP216/donnees/aclImdb/test/pos/*.txt"

var UNLABELED_COMMENT_PATTERN = "/home/user/Bureau/CNAM/git/CNAM-
RCP216/donnees/aclImdb/train/unsup/*.txt"

/**
  * Methode utilitaire permettant de recuperer des fichiers d'un repertoire selon un
pattern
  * @param dirPath
  * @param pattern
  * @return
  */
def getListOfFilepathByPattern(pattern: String, dirPaths: String* ): Seq[String] = {
    var fileListList = dirPaths.map( x => new File(x).listFiles.filter(_.isFile
).filter(_.getName.matches(pattern) ).map(_.getAbsolutePath).toList.asInstanceOf[Seq
[String]])
    var fileList : Seq[String] = fileListList.flatten
    println("*" * 80 )
    println("nombre de fichiers à prédire <" + fileList.size + ">")
    println("*" * 80 )
    return fileList
    }

//fichiers notes de test selon pattern
var FILES_TO_PREDICT=getListOfFilepathByPattern("^[0-9]*_.*"
,"/home/user/Bureau/CNAM/git/CNAM-RCP216/donnees/aclImdb/train/neg"
,"/home/user/Bureau/CNAM/git/CNAM-RCP216/donnees/aclImdb/train/pos"
,"/home/user/Bureau/CNAM/git/CNAM-RCP216/donnees/aclImdb/test/neg"
,"/home/user/Bureau/CNAM/git/CNAM-RCP216/donnees/aclImdb/test/pos")
/*
Fichiers non notés
var FILES_TO_PREDICT=List("/home/user/Bureau/CNAM/git/CNAM-
RCP216/donnees/aclImdb/train/unsup/10_0.txt",
                          "/home/user/Bureau/CNAM/git/CNAM-
RCP216/donnees/aclImdb/train/unsup/50_0.txt",
                          "/home/user/Bureau/CNAM/git/CNAM-
RCP216/donnees/aclImdb/train/unsup/100_0.txt",
                          "/home/user/Bureau/CNAM/git/CNAM-
RCP216/donnees/aclImdb/train/unsup/150_0.txt",
                          "/home/user/Bureau/CNAM/git/CNAM-
RCP216/donnees/aclImdb/train/unsup/200_0.txt",
                          "/home/user/Bureau/CNAM/git/CNAM-
```

```
RCP216/donnees/aclImdb/train/unsup/250_0.txt",
                          "/home/user/Bureau/CNAM/git/CNAM-
RCP216/donnees/aclImdb/train/unsup/300_0.txt",
                          "/home/user/Bureau/CNAM/git/CNAM-
RCP216/donnees/aclImdb/train/unsup/350_0.txt",
                          "/home/user/Bureau/CNAM/git/CNAM-
RCP216/donnees/aclImdb/train/unsup/400_0.txt",
                          "/home/user/Bureau/CNAM/git/CNAM-
RCP216/donnees/aclImdb/train/unsup/450_0.txt")
*/

/**
  * Methode pour prédire les notes
  * en REPL s'appelle avec : preditNotes(FILES_TO_PREDICT,normalizedUS,docIds)
  *
  * VERSION NON OPTIMISEE SEQUENTIELLE
  *
  * @param filesToPredict
  * @param normalizedMatUS
  * @param doc_id
  */
def preditNotes(filesToPredict : Seq[String], normalizedMatUS : RowMatrix, doc_id :
scala.collection.Map[Long, String]) : Seq[(String,Double,Double)]  = {

  //creation et remplissage de la map idDocs : nomFichier -> idDoc à partir de docIds
tableau de couples (id,nomFichier)
  var id_doc = scala.collection.mutable.Map[String,Long]()
  for(pair <- doc_id) id_doc(pair._2) = pair._1

  var notesFichiers  : Seq[(String,Double,Double)] = Seq()

  for (file <- filesToPredict) {
    //recuperation du label note
    var note_label:Double = file.split('/').last.split('_').last.split('.')(0).toLong
    //calcul des documents similaires
    val tdfd = RunLSA.topDocsForDoc(normalizedMatUS, id_doc("file:" + file))
    val file_weight = tdfd.map { case (score, id) => (doc_id(id), score) }
    //on se ramene au couple (note du commentaire similaire, %similarité)
    var note_weight = file_weight.map { case (filename, weight) => (filename.split(
'/').last.split('_').last.split('.')(0).toLong, weight) }
    //suppression du document lui  meme
    note_weight = note_weight.drop(1)
    //calcul de la note ponderee par la similarité
    var sum_weight = note_weight.foldLeft(0.0) { case (a, (k, v)) => a + v }
    var note_pred:Double = note_weight.map { case (note, weight) => note * weight
}.reduce((a, b) => (a + b)) / sum_weight
    //ajout cle valeur dans la map !
    notesFichiers = notesFichiers :+ (( file,note_label,Math.round(note_pred
).asInstanceOf[Double]))
    if( DISPLAY_NOTE ) {
    println("-" * 80)
```

```
    println("La note prédite par similarité de <" + file + "> est <" + Math.round
(note_pred) + ">")
    //donner les documents similaires à un document à note inconnue
    RunLSA.printTopDocsForDoc(normalizedMatUS, "file:" + file, id_doc, doc_id)
    println("-" * 80)
    }
  }
  return notesFichiers
}

/**
  * Methode pour prédire les notes
  * en REPL s'appelle avec : preditNotesParallele("/home/user/Bureau/CNAM/git/CNAM-
RCP216/donnees/aclImdb/train/unsup/100_0.txt",normalizedUS,docIds)
  * impossible cf SPARK-5063
  *
  *l appel de topDocForDocuments doit se faire sequentiellement
  *
  * @param file
  * @param normalizedMatUS
  * @param doc_id
  */
def preditNotesParallele(file : String, normalizedMatUS : RowMatrix, doc_id :
scala.collection.Map[Long, String]) : (String,Double,Double)  = {
  //creation et remplissage de la map idDocs : nomFichier -> idDoc à partir de docIds
tableau de couples (id,nomFichier)
  var id_doc = scala.collection.mutable.Map[String,Long]()
  for(pair <- doc_id) id_doc(pair._2) = pair._1
  //recuperation du label note
  var note_label:Double = file.split('/').last.split('_').last.split('.')(0).toLong
  //calcul des documents similaires
  val tdfd = RunLSA.topDocsForDoc(normalizedMatUS, id_doc("file:" + file))
  val file_weight = tdfd.map { case (score, id) => (doc_id(id), score) }
  //on se ramene au couple (note du commentaire similaire, %similarité)
  var note_weight = file_weight.map { case (filename, weight) => (filename.split('/'
).last.split('_').last.split('.')(0).toLong, weight) }
  //suppression du document lui  meme
  note_weight = note_weight.drop(1)
  //calcul de la note ponderee par la similarité
  var sum_weight = note_weight.foldLeft(0.0) { case (a, (k, v)) => a + v }
  var note_pred:Double = note_weight.map { case (note, weight) => note * weight
}.reduce((a, b) => (a + b)) / sum_weight
  //ajout cle valeur dans la map !
  var notesFichiers = ( file,note_label.asInstanceOf[Double],Math.round(note_pred
).asInstanceOf[Double])
  if( DISPLAY_NOTE ) {
    println("-" * 80)
    println("La note prédite par similarité de <" + file + "> est <" + Math.round
(note_pred) + ">")
    //donner les documents similaires à un document à note inconnue
    RunLSA.printTopDocsForDoc(normalizedMatUS, "file:" + file, id_doc, doc_id)
```

```
    println("-" * 80)
    }
return notesFichiers
}

/**
  * MAIN
  */

// inutile en REPL echo on par defaut
// println(FILES_TO_PREDICT)

println("*" * 80)
println("* CHARGEMENT DONNEES BRUTES *")
println("*" * 80)

//Negative
val rawTrainTextNeg=spark.sparkContext.wholeTextFiles(TRAIN_NEGATIVE_COMMENT_PATTERN)
val rawTestTextNeg=spark.sparkContext.wholeTextFiles(TEST_NEGATIVE_COMMENT_PATTERN)
//Positives
val rawTrainTextPos=spark.sparkContext.wholeTextFiles(TRAIN_POSITIVE_COMMENT_PATTERN)
val rawTestTextPos=spark.sparkContext.wholeTextFiles(TEST_POSITIVE_COMMENT_PATTERN)

//non notes
//val rawTextUnknown=spark.sparkContext.wholeTextFiles(UNLABELED_COMMENT_PATTERN)
//val rawTextUnknown=spark.sparkContext.wholeTextFiles(FILES_TO_PREDICT mkString(","))

//union
val rawText=rawTrainTextPos.union(rawTrainTextNeg).union(rawTestTextNeg).union
(rawTestTextPos)
//.union(rawTextUnknown) si donneee non presentes jeu train/test non labellisees

println("*" * 80)
println("* * CHARGEMENT STOP WORDS, LEMMATISATION + SUPPRESSION STOP WORDS MATRICE
DOC/TERM*")
println("*" * 80)
//Chargement des stops words via API "cloudera"
val stopWords = sc.broadcast(ParseWikipedia.loadStopWords(STOP_WORDS_PATH)).value
//Lemmatisation + suppression stop words de la matrice documents
val lemmatized = rawText.mapPartitions(iter => {val ppl = ParseWikipedia
.createNLPPipeline();iter.map{ case(filepath, tokenized) => (filepath, ParseWikipedia
.plainTextToLemmas(tokenized, stopWords, ppl))};}).cache()

println("*" * 80)
println("* CALCUL MATRICE TF/ITF *")
println("*" * 80)

//Calcul matrice TF/ITF, mapping document-Iddoc, mapping terme-termId
val (termDocMatrix, termIds, docIds, idfs) = ParseWikipedia.termDocumentMatrix
(lemmatized, stopWords, numTerms, sc)
```

```
println("*" * 80)
println("* Serialisation 'cache' matrice TF/ITF *")
println("*" * 80)
termDocMatrix.cache

println("*" * 80)
println("* CALCUL MATRICE SVD *")
println("*" * 80)
//transformation en rowmatrix
val mat = new RowMatrix(termDocMatrix)

//calcul SVD
val svd = mat.computeSVD(k, computeU = true)

//(opt) val topConceptTerms = RunLSA.topTermsInTopConcepts(svd, 10, 10, termIds)
//(opt)  val topConceptDocs = RunLSA.topDocsInTopConcepts(svd, 10, 10, docIds)

println("*" * 80)
println("* CALCUL MATRICE U.S Normalisée*")
println("*" * 80)
val US = RunLSA.multiplyByDiagonalMatrix(svd.U, svd.s)
//Normalisation U.S
val normalizedUS = RunLSA.rowsNormalized(US)

println("*" * 80)
println("* CALCUL DES SIMILARITES ET DEDUCTION DES NOTES *")
println("*" * 80)

//calculer la somme ponderee des notes des documents
//veille version LENTE
var notesFichiers:Seq[(String,Double,Double)]=preditNotes(FILES_TO_PREDICT
,normalizedUS,docIds)

//version paralelisee ne fonctionne pas
//conversion en dataset
//var filesDs = FILES_TO_PREDICT.toDS
//var notesFichiers = filesDs.map( row =>
preditNotesParallele(row,normalizedUS,docIds))

println("*" * 80)
println("* CALCUL DE LA PERFORMANCE DE LA PREDICTION PAR SIMILARITES *")
println("*" * 80)

//conversion en dataframe
val notesFichiersDf = notesFichiers.toDF("fichier","note_label","note_pred")
//calcul de la precision par Spark
val evaluator = new MulticlassClassificationEvaluator().setLabelCol("note_label"
).setPredictionCol("note_pred").setMetricName("accuracy")
val accuracy_spark = evaluator.evaluate(notesFichiersDf)
//calcul de la precision par calcul
//val accuracy_calcul =  df.filter($"note_label" == Math.Round($"note_pred")).count /
```

```
df.count

//creation vue
notesFichiersDf.createOrReplaceTempView("notes")

//liste notes equivalentes aux predictions
val notes_exactes_DF = spark.sql("select fichier,ROUND(note_label),ROUND(note_pred)
from notes where note_label=ROUND(note_pred)")
notes_exactes_DF.show
val nb_notes_exactes = notes_exactes_DF.count
val accuracy_calcul = 1.0 * nb_notes_exactes/notesFichiersDf.count

//calcul du % de notes predites à 1 point de la note reelle
val notes_moins_un = 1.0 * spark.sql("select * from notes where
ROUND(note_pred)=note_label OR ROUND(note_pred)=note_label+1 OR
ROUND(note_pred)=note_label-1").count/notesFichiersDf.count

//calcul du % de notes predites à 2 points de la note reelle
val notes_moins_deux = 1.0 * spark.sql("select * from notes where
ROUND(note_pred)=note_label OR ROUND(note_pred)=note_label+1 OR
ROUND(note_pred)=note_label-1 OR ROUND(note_pred)=note_label+2 OR
ROUND(note_pred)=note_label-2").count/notesFichiersDf.count
```

7.6. Scripts correction fichier SVM

7.6.1. Le script code "à la main"

```
## auteur : F.DUNAN
## objet :
## entree :
## sortie :
## infos:
## commande de lancement: python3 ./feat2SVM.py

#nombre de lignes en entree
# 14:45:15-user@M40474:~/Bureau/CNAM/git/CNAM-
RCP216/donnees/aclImdb/train(feature/FDU)$ wc -l ./labeledBow.feat
# 25000 ./labeledBow.feat

import time

DEBUG=False
featFile = open('../../../../donnees/aclImdb/train/labeledBow.feat', "r")
featFile_out = open('../../../../donnees/aclImdb/train/labeledBow_FDU.feat', "w")
try:
    i=1
    for line in featFile:
        index=[]
        tf=[]
```

```
        if( i % 1000 == 0):
            print("*** ligne ",i)
        splitted_line=line.split()
        id_nums = splitted_line[1:]
        if(DEBUG):
            print("id_nums =", id_nums)
        for id_num in id_nums:
            #if(DEBUG):
            #    print('id_num = ',id_num)
            id_num_list=id_num.split(':')
            #if(DEBUG):
            #    print('id_num_list = ',id_num_list)
            index.append(id_num_list[0])
            tf.append(id_num_list[1])
        index_modifie=[str(int(x)+1) for x in index]
        idx_tfs=list(zip(index_modifie,tf))
        if(DEBUG):
            print('-'*80)
            print('INDEX = ',index)
            print('-'*80)
            print('-'*80)
            print('INDEX_MODIFIE = ',index_modifie)
            print('-'*80)
            print('TF = ',tf)
            print('-'*80)
            print("idx_tf =",idx_tfs)
        #ecriture note
        featFile_out.write(splitted_line[0])
        #ecriture idx et tf
        for idx_tf in idx_tfs:
            featFile_out.write(" " + idx_tf[0] + ':' +idx_tf[1])
        featFile_out.write('\n')
        i=i+1
        if(DEBUG):
            time.sleep(50)
finally:
    featFile.close()
    featFile_out.close()
```

7.6.2. Le script récupéré sur Internet

```
from sklearn.datasets import load_svmlight_file, dump_svmlight_file
X,y = load_svmlight_file('full_data.feat')
X.indices=(X.indices+1)
dump_svmlight_file(X,y,'fixed_full_data.feat')
```

Références

- Il est nécessaire de citer l'étude d'où provient le dataset :
  Potts, Christopher. 2011. On the negativity of negation. In Nan Li and David Lutz, eds., Proceedings of Semantics and Linguistic Theory 20, 636-659.
- Le TP du cours de fouille de données textuelles du CNAM-RCP216
  http://cedric.cnam.fr/vertigo/Cours/RCP216/tpFouilleTexte.html
- La cross validation dans les cours et TP CNAM-RCP216
  http://cedric.cnam.fr/vertigo/Cours/RCP216/coursApprentissageLargeEchelle.html#apprentissage-supervise-avec-spark
  et surtout
  http://cedric.cnam.fr/vertigo/Cours/RCP216/tpSVMlineaires.html
- Le grid search en Spark pour la recherche des hyperparamètres
  http://cedric.cnam.fr/vertigo/Cours/RCP216/tpSVMlineaires.html#recherche-des-valeurs-optimales-des-hyperparametres
- Le livre dont est inspiré le TP précédent
  Ryza, S., U. Laserson, S. Owen and J. Wills, Advanced Analytics with Spark, O'Reilly, 2010.