Prof. Amal Zouaq – INF8460 (Automne 2019)
Chargé de laboratoire : Félix Martel

École Polytechnique de Montréal

Département Génie Informatique et Génie Logiciel

INF8460 – Traitement automatique de la langue naturelle

Objectifs d'apprentissage

- Explorer les modèles d'espaces vectoriels comme représentations distribuées de la sémantique des mots et des documents
- Comprendre différentes mesures de distance entre vecteurs de documents et de mots
- Utiliser un modèle de langue n-gramme de caractères et l'algorithme Naive Bayes pour l'analyse de sentiments dans des revues de films (positives, négatives)

Ressources

nltk.tokenize package API. http://www.nltk.org/api/nltk.tokenize.html
http://www.nltk.org/book/ch02.html

Logiciels

Ce TP utilise la librairie NLTK, ScikitLearn et Jupyter notebook.

Il est conseillé d'installer Anaconda avec Python 3.7.

Modalités de remise du TP

Vous devez compléter le squelette `inf8460_tp2.ipynb` et le soumettre sous le nom `matricule1_matricule2_matricule3_TP2.ipynb` qui reprend les différentes questions, et implante les fonctionnalités requises avec des commentaires lorsque demandé.

Critères d'évaluation

- L'exécution correcte du code
- Le professionnalisme du notebook
- La clarté des explications qui l'accompagnent

Corpus et code

Le zip du TP contient :

- Un corpus de revues de films : aclImdb_v1.tar.gz. Vous trouverez deux sous-répertoires dans train et test qui contiennent des revues négatives (neg) et positives (pos). Chaque fichier contient une revue.
- Une lecture préparatoire qui vous sera utile pour la classification: E03-1053.pdf
- Le squelette du notebook qui doit être complété

Vous devez retourner un zip qui contient :

- Le notebook complété
- Le fichier vocab.txt

Travail à faire

1. Pré-traitement (10 points)

a) Créez la fonction clean_doc() qui effectue les pré-traitements suivants : segmentation en mots ; suppression des signes de ponctuations ; suppression des mots qui contiennent des caractères autres qu'alphabétiques ; suppression des mots qui sont connus comme des stop words ; suppression des mots qui ont une longueur de 1 caractère. Les stop words peuvent être obtenus avec `from nltk.corpus import stopwords ;`

b) Créez la fonction build_voc qui extrait les unigrammes de l'ensemble d'entrainement et conserve ceux qui ont une fréquence d'occurrence de 5 au moins et imprime le nombre de mots dans le vocabulaire. Sauvegardez-le dans un fichier `vocab.txt ;`

c) Vous devez créer une fonction get_top_unigrams(n) qui retourne les n unigrammes les plus fréquents et les affiche ;

d) Vous devez créer une fonction get_top_unigrams_per_cls(n, cls) qui retourne les n unigrammes les plus fréquents de la classe cls et les affiche ;

e) Affichez les 10 unigrammes les plus fréquents dans la classe positive ;

f) Affichez les 10 unigrammes les plus fréquents dans la classe négative ;

2. Matrices de co-occurrence (20 points)

2.1. Matrices de co-occurrence document × mot M(d,w)

A partir des textes du corpus d'entrainement (neg/pos), vous devez construire une matrice de co-occurrence document × mot qui contient les 5000 unigrammes les plus fréquents et les pondérer avec la mesure TF-IDF.

2.2. Matrices de co-occurrence mot × mot M(w,w)

a) A partir des textes du corpus d'entrainement (neg/pos), vous devez construire une matrice de co-occurrence mot × mot M(w,w) qui contient les 5000 unigrammes les plus fréquents. Le contexte de co-occurrence est une fenêtre de +/-5 mots autour du mot cible.

b) Vous devez créer une fonction calculate_PPMI qui prend la matrice M(w,w) et la transforme en une matrice M'(w,w) avec les valeurs PPMI.

3. Mesures de similarité (20 points)

Vous devez maintenant implanter des mesures de similarité entre vecteurs. Pour implanter ces mesures, vous pouvez utiliser l'api scipy.spatial.distance.

a) Implémentez la fonction get_Euclidean_Distance(v1 ,v2) qui retourne la distance euclidienne entre les vecteurs v1 et v2.

b) Implémentez la fonction get_Cosinus_Distance(v1, v2) qui retourne la distance cosinus entre les vecteurs v1 et v2.
c) Implémentez la fonction get_most_similar_PPMI(word, metric) qui prend un mot en entrée et une mesure de distance et qui retourne les n mots les plus similaires selon la mesure. Les mesures à tester sont : la distance Euclidienne et la distance cosinus implantées ci-dessus. Le vecteur du mot *word* doit être extrait de la matrice M'(w,w).
d) Trouvez les 5 mots les plus similaires au mot « bad » avec les deux métriques et affichez-les. Commentez.
e) Implémentez la fonction get_most_similar_TFIDF(word, metric) qui prend un mot en entrée et une mesure de distance et qui retourne les n mots les plus similaires selon la mesure. Les mesures à tester sont : la distance Euclidienne et la distance cosinus implantées ci-dessus. Le vecteur du mot word doit être extrait de la matrice M(d,w).
f) Trouvez les 5 mots les plus similaires au mot « bad » et affichez-les. Commentez.

4. Classification automatique avec un modèle de langue (20 points)

Vous allez maintenant procéder à l'analyse de sentiments en implantant un modèle de langue Laplace à partir de n-grammes de caractères.

Vous pouvez vous inspirer de https://nbviewer.jupyter.org/gist/yoavg/d76121dfde2618422139 (qui est compatible avec Python 2) pour la création du modèle n-gramme de caractères (n=4).

L'idée est la suivante : Les modèles de langue peuvent être appliqués à la classification de texte. Si on veut classifier un document D dans une catégorie $c \in C = c1,...,cN$, on retourne la catégorie qui obtient la plus grande probabilité postérieure étant donné le texte :

$$c* = \text{argmax}_{c \in C} P(c|D)$$

En utilisant la rège de Bayes:

$$c* = \text{argmax}_{c \in C} P(D|c)P(c)$$

Si nous supposons que toutes les classes ont la même probabilité, nous pouvons simplement supprimer le terme P (c) :

$$c* = \text{argmax}_{c \in C} P(D|c)$$

P(D|c) peut être calculé en entrainant un modèle de langue sur tous les textes associés à la catégorie c. Pour classifier un nouveau document D, on utilise les modèles de langue associés à chaque classe pour calculer la probabilité de D dans ce modèle, et on retourne la classe qui assigne la plus grande probabilité à D.

Pour mieux comprendre cette approche, vous devez lire l'article E03-1053.pdf disponible sur Moodle où cette approche est utilisée pour l'identification d'auteurs.

5. Classification automatique avec un modèle sac de mots (unigrammes) et Naive Bayes (10 points)

En utilisant la librairie scikitLearn et l'algorithme Multinomial Naive Bayes, effectuez la classification des revues avec un modèle sac de mots pondéré avec TF-IDF. Vous devez utiliser la matrice de co-occurrence TF-IDF créée en 2.1.

6. Amélioration de vos modèles (10 points)

Vous devez proposer une méthode qui vous permet d'améliorer les performances du modèle obtenu en 5). Quelques pistes :

1) Réduction/modification et meilleur choix du vocabulaire par exemple en vous basant sur un lexique de sentiments ;
2) Ajouts d'attributs informatifs
3) Gestion des caractères inconnus
4) Réduction de dimensionnalité
5) Autres

Expliquez clairement votre méthode, puis créez le code approprié.

7. Evaluation (10 points)

Vous devez ensuite tester vos algorithmes de la section 4, 5 et 6 sur l'ensemble de test et reporter vos résultats dans une même table avec les métriques suivantes : Accuracy et pour chaque classe, la précision, le rappel et le F1 score.
Affichez les résultats dans votre notebook et discutez-en. Quels sont les meilleurs modèles en général? Pour la classe positive et négative ? Comment se comporte votre classification avec un modèle n-gramme par rapport à Naive Bayes ?