TP Apprentissage 2017

Alexis Lanoix & Jofrey Luc

Binôme

Pour ce projet, la répartition du travail s'est effectuée à peu près de la manière suivante : Alexis a écrit les focntions de chargement de dictionnaire, de lecture de mail, d'enregistrement et chargement de classifieur, et d'apprentissage en ligne. Jofrey s'est chargé des fonctions d'apprentissage, de lissage, de prédiction, de calcul de probabilités et de test. Nous avons tous deux travaillé sur le rapport.

Solution mise en oeuvre

Le filtre est programmé en python 3, et n'utilise que les libraires standard.

La structure du code est la suivante :

- `moduleFiltreAntiSpam.py` contient toutes les fonctions proprement dites du classifieur (charger_dictionnaire, apprendre_spam, etc.)
- `filtreAntiSpam.py` contient un main lançant l'apprentissage et le test d'ensembles de mails (sur les bases par défaut si rien n'est précisé) en affichant les résultats
- `apprend_filtre.py` contient un main lançant l'apprentissage d'un ensemble de mail et stockant le classifieur obtenu dans un fichier (en Json)
- `filtre_mail.py` contient un main lançant une prédiction d'un classifieur enregistré sur un mail précis
- `apprend_filtre_enligne.py` contient un main permettant d'ajouter un mail dans la base d'apprentissage d'un classifieur et de recalculer ses paramètres sans refaire l'intégralité de l'apprentissage
- `modulUtils.py` contient des fonctions "utiles" pas vraiment liées au classifieur (validation des paramètres, des entrées utilisateur, etc.)

Pour réaliser ce filtre, nous avons suivi les grandes étapes du sujet :

- Pour le dictionnaire de mots, nous avons utilisé une liste python (qui est à la fois un tableau, une liste et un dictionnaire) à deux dimensions, de taille [nb mots dictionnaire] * [2] : la première dimension contient les mots du dictionnaire, qui sont les clefs de la liste, et la deuxième contient un tableau de deux valeurs, qui sont la probabilité de voir ce mot apparaître dans un spam et dans un ham. Dans la pratique, `dico_probas['MONEY'][0]` renvoie la probabilité que le mot "money" apparaisse dans un spam et `dico_probas['MONEY'][1]` la probabilité qu'il apparaisse dans un ham. Le dictionnaire est initialisé à [0,0] pour tous les mots.
- Le vecteur binaire représentant un mail est de la même forme, avec pour valeurs non pas un tableau de deux probabilités mais un booléen indiquant si le mot est présent dans le mail : `dico_probas['MONEY']` renvoie `True` ou `False`.
- Apprendre un message consiste du coup simplement à parcourir les mots du message qui sont à `True` dans son vecteur et à modifier la probabilité correspondante à ce mot dans le dictionnaire (en multipliant l'ancienne probabilité par le nombre de spams/hams, en y ajoutant 1, puis en la redivisant par le nombre de spams/hams).

— Le lissage se fait de la même manière, en parcourant tous les mots et en ajoutant un epsilon dans les probabilités.
— Pour prédire un message, on pacourt le dictionnaire et on somme les logs des probabilités spam/ham si le mot est présent dans le mail ou de 1 - ces probabilités si le mot n'est pas présent. On obtient deux valeurs, pour la somme des log des spam et celle des ham. On multiplie ces probabilités par la probabilité d'être un spam ou un ham à priori, et on compare les deux valeurs obtenues.

Pour les améliorations :

— La sauvegarde d'un classifieur se fait en utilisant le sérialiseur json standard de python. On stocke ainsi dans un fichier le dictionnaire (mots + probabilités spam/ham) + le nombre de spams/hams utilisés pour ce classifieur. Le chargement d'un classifieur se fait aussi de manière standard.
— Pour faire l'apprentissage en ligne, on parcourt tout simplement le dictionnaire en transformant chaque probabilité spam ou ham (selon le type du message ajouté) tel que décrit dans le sujet : en ajoutant 1 au nombre de spam/ham au dénominateur, et en ajoutant éventuellement 1 au nombre 'occurences du mot au numérateur (si le mot est présent dans le mail).

Difficultés rencontrées

La principale difficulté que nous avons rencontrée (en dehors de quelques problèmes techniques) a été d'afficher correctement les probabilités de spam ou ham *a posteriori* lors de la prédiction. Le calcul direct en mettant à l'exponentielle les valeurs en log renvoyées par le calcul des sommes des probabilités du dictionnaire n'étant pas précis du tout (1 ou 0 quoi qu'il arrive), nous avons fini par mettre en oeuvre la solution conseillée, c'est à dire de diviser le numérateur et le diviseur dans le calcul par l'exponentielle de la somme des log spam/ham, ce qui nous permet d'avoir à calculer une seule exponentielle. Malheureusement ce n'est pas toujours parfait (la précision peut laisser à désirer quand les différences sont trop grandes, comme des calculs de e^{-25} par exemple).

Exécution du code

Les fichiers exécutables sont `filtreAntiSpam.py`, `apprend_filtre.py`, `filtre_mail.py` et `apprend_filtre_enligne.py`. Pour les exécuter, il suffit de se placer dans le répertoire `scripts/` et de taper, selon le fichier (les descriptions des paramètres peuvent être obtenues en tapant `python3 fichier.py -h`) :

— `python3 filtreAntiSpam.py [-a repertoireApprentissage] [-d dictionnaire] repertoireTest [nbSpam] [nbHam]`, pour apprendre et tester
— `python3 apprend\_filtre.py [-d dictionnaire] monClassifieur repertoireApprentissage [nbSpam] [nbHam]`, pour apprendre et sauvegarder
— `python3 filtre\_mail.py monClassifieur mail`, pour prédire un mail avec un classifieur sauvegardé
— `python3 apprend\_filtre\_enligne.py monClassifieur mail type`, pour ajouter un mail dans un classifieur (type = SPAM ou HAM)

Exemples d'exécution et évaluation des performances

D'après les tests que nous avons effectués au cours de notre réalisation du filtre, les performances de notre classifieur sont assez satisfaisantes : selon le nombre d'exemples fournis à l'apprentissage, le taux d'erreur se situe en général entre 10 et 20% ; pour un classifieur naïf, c'est assez bon. En revanche, nous avons également remarqué que le taux d'ereur était très souvent beaucoup plus élevé sur les spams que sur les hams de test ; celà tient sans doute à la nature des mails qu'on lui fournit en entrée (si ils viennent tous du même corpus, etc.).

```
# python3 filtreAntiSpam.py ../basetest/ 100 100

Spams dans la base d'apprentissage ? (max 500) 200
Hams dans la base d'apprentissage ? (max 2500) 200
Apprentissage...
Lissage...
Tests :
Spam ../basetest/spam/7.txt, P(SPAM) = 1.0, P(HAM) = 0.0
-> identifié comme spam
Spam ../basetest/spam/80.txt, P(SPAM) = 1.0, P(HAM) = 0.0
-> identifié comme spam
Spam ../basetest/spam/430.txt, P(SPAM) = 0.999991318927151, P(HAM) = 8.681072849014981e-06
-> identifié comme spam
Spam ../basetest/spam/439.txt, P(SPAM) = 3.4244860494544087e-06, P(HAM) = 0.9999965755139505
-> identifié comme ham **erreur**
Spam ../basetest/spam/102.txt, P(SPAM) = 0.9999998242062487, P(HAM) = 1.757937513335861e-07
-> identifié comme spam
[......]
Ham ../basetest/ham/80.txt, P(SPAM) = 7.818983215159863e-12, P(HAM) = 0.999999999992181
-> identifié comme ham
Ham ../basetest/ham/430.txt, P(SPAM) = 1.4852503260524142e-05, P(HAM) = 0.9999851474967395
-> identifié comme ham
Ham ../basetest/ham/439.txt, P(SPAM) = 7.555832292932339e-08, P(HAM) = 0.9999999244416771
-> identifié comme ham
Ham ../basetest/ham/102.txt, P(SPAM) = 1.0590800912317048e-05, P(HAM) = 0.9999894091990876
-> identifié comme ham
[......]

16.00% d'erreurs sur les spams
3.00% d'erreurs sur les hams
9.50% d'erreurs sur l'ensemble
```

```
# python3 apprend_filtre.py monClassifieur ../baseapp/

Apprentissage sur 500 spams et 2500 hams...
Lissage...
Classifieur enregistré dans 'monClassifieur'.
```

```
# python3 filtre_mail.py monClassifieur ../basetest/spam/1.txt

D'après 'monClassifieur', le message '../basetest/spam/1.txt' est un SPAM à 1.0 !


# python3 apprend_filtre_enligne.py monClassifieur ../basetest/spam/50.txt SPAM

Modification du filtre 'monClassifieur' par apprentissage sur le SPAM '../basetest/spam/50.txt'.
```