Introduction à l'apprentissage
Master 1
TP filtre anti-spam

F. Lauer

Le but de ce TP est de construire un filtre anti-spam basé sur le classifieur naïf de Bayes.

1 Rendu du TP

— Ce TP est à réaliser en binôme.
— Chaque binôme devra déposer son TP sur ARCHE sous forme d'une archive ZIP (ou TAR.GZ) **avant le 8 mai 2018 à 23h55** (dépôt impossible passé cette date).
— L'archive ZIP devra contenir au minimum :
  — Le rapport de TP décrivant la contribution de chaque membre du binôme, la solution mise en œuvre, les choix effectués et les difficultés rencontrées. Ce rapport devra également contenir des exemples d'exécution du programme et une partie sur la validation de la solution proposée : à combien estimez-vous la performance de votre filtre anti-spam ?
  — Les codes sources (incluant tous les fichiers nécessaires au bon fonctionnement du programme). Ces fichiers devront pouvoir être compilés aisément sur une machine Linux standard. L'idéal est de fournir un TP qui fonctionne suite à ces commandes :
  ```
  unzip TP_NOM1_NOM2.zip
  cd TP_NOM1_NOM2
  javac filtreAntiSpam.java
  ```
  (tout autre mode de création de l'exécutable devra être décrit dans le rapport)
— La note du TP prendra en compte les points suivants :
  — le nombre d'étapes correctement réalisées et fonctionnelles (voir section 4),
  — le respect des consignes de la section 5,
  — la qualité du rapport et la pertinence des commentaires,
  — la performance du classifieur évaluée sur une base de test d'environ 3000 exemples (*non fournie*).
  — l'implémentation éventuelle des améliorations de la section 6.

2 Fichiers à disposition

Vous pouvez récupérer sur ARCHE l'archive `TP.zip` contenant les fichiers suivants.
— `dictionnaire1000en.txt` : un fichier texte contenant 1000 mots anglais parmi les plus couramment utilisés (un mot par ligne).
— `baseapp/` : un répertoire `spam` contenant 500 messages de type spam et un répertoire `ham` contenant 2500 messages de type ham. Chaque message est un fichier texte nommé `X.txt` où `X` représente l'indice du message.
— `basetest/` : un répertoire `spam` contenant 500 messages de type spam et un répertoire `ham` contenant 500 messages de type ham.

3 Modélisation des messages

— Nous considérerons les messages comme des sacs de mots, c'est-à-dire des ensembles de mots dont l'ordre n'a pas d'importance.

— Vous utiliserez le modèle binomial de textes conduisant à une représentation sous forme de vecteurs binaires de taille fixe où seule la présence des mots est prise en compte.

4 Etapes du TP à réaliser

Vous devrez implémenter un filtre anti-spam basé sur le classifieur naïf de Bayes. Les grandes étapes de cette implémentation sont les suivantes :

— **Fonction** `charger_dictionnaire` : cette fonction doit pouvoir charger un dictionnaire (par exemple dans un tableau de mots) à partir d'un fichier texte.

— **Fonction** `lire_message` : cette fonction doit pouvoir lire un message (dans un fichier texte) et le traduire en une représentation sous forme de vecteur binaire x à partir d'un dictionnaire.

— **Apprentissage** : l'apprentissage des paramètres du classifieur naïf de Bayes à partir de $m_{spam} + m_{ham}$ messages. Les nombres m_{spam} et m_{ham} serons donnés par l'utilisateur, mais l'emplacement de la base d'apprentissage pourra être figée dans le code. Vous pourrez tester votre classifieur avec, par exemple, $m_{spam} = 200$ et $m_{ham} = 200$.

— **Test** : le classifieur devra pouvoir être testé sur un ensemble de $m'_{spam} + m'_{ham}$ messages de test stockés dans 2 sous-répertoires `spam` et `ham` d'un répertoire passé en argument de la ligne de commande. La programme doit alors calculer le taux d'erreur sur les spams, sur les hams et le taux d'erreur global sur l'ensemble des exemples.
Exemple d'exécution pour une base de test dans le dossier courant contenant 100 messages dans `basetest/spam/` et 200 messages dans `basetest/ham` :

```
java filtreAntiSpam basetest 100 200
Combien de SPAM dans la base d'apprentissage ? 200
Combien de HAM dans la base d'apprentissage ? 200

Apprentissage...

Test :
SPAM numéro 0 identifié comme un SPAM
SPAM numéro 1 identifié comme un HAM  *** erreur ***
SPAM numéro 2 identifié comme un SPAM
...
HAM numéro 0 identifié comme un HAM
HAM numéro 1 identifié comme un HAM
HAM numéro 2 identifié comme un SPAM *** erreur ***
...

Erreur de test sur les 100 SPAM      : 24 %
Erreur de test sur les 200 HAM       : 12 %
Erreur de test globale sur 300 mails : 16 %
```

— Le programme doit aussi afficher pour chaque exemple de test les probabilités a posteriori des deux catégories, par exemple :

```
SPAM numéro 0 : P(Y=SPAM | X=x) = 9.956780e-01, P(Y=HAM | X=x) = 4.321999e-03
                =>  identifié comme un SPAM
SPAM numéro 1 :  P(Y=SPAM | X=x) = 4.956780e-01, P(Y=HAM | X=x) = 5.043220e-01
                =>  identifié comme un HAM  *** erreur ***
```

5 Consignes et indices

— Les consignes au niveau de l'interface ne sont données qu'à titre indicatif. L'utilisation de votre programme peut être différente si elle est soit évidente soit documentée.

— Le chargement du dictionnaire doit exclure les mots de moins de 3 lettres (typiquement des mots "outils" inutiles pour la classification).

— Le classifieur naïf de Bayes ne nécessite pas de retenir toute la base d'apprentissage en mémoire. Vous pouvez donc parcourir la base d'apprentissage en traitant chaque mail les uns après les autres et indépendamment (en n'ouvrant qu'un fichier à la fois).
— La comparaison des mots du mail avec ceux du dictionnaire doit être insensible à la casse.
— Vous pouvez vous contenter dans un premier temps d'une version simple de la fonction `lire_message` dans laquelle les mots sont supposés tous être séparés par des espaces. Même si cela implique que les mots suivis d'un signe de ponctuation comme "`house,`" ne seront pas reconnus comme faisant partie du dictionnaire et seront ignorés.
— Utilisez des tests "if" plutôt que les formes du type $(b^j_{SPAM})^{x^j}(1-b^j_{SPAM})^{1-x^j}$ (impliquant des puissances 0 et 1) pour calculer la prédiction du classifieur plus efficacement.
— Appliquez le lissage des paramètres avec $\epsilon = 1$.
— Faites attention aux problèmes numériques lors de l'étiquetage d'un nouvel email de test. Par exemple, évitez les tests du type if ($10^{-354} < 10^{-342}$) ...
— Essayez de limiter l'influence de la précision de la machine sur le calcul des probabilités *a posteriori*.

6 Améliorations (bonus)

— Améliorez l'interface : le classifieur doit être enregistré dans un fichier pour permettre de tester un nouveau message rapidement. Par exemple :

```
java apprend_filtre mon_classifieur baseapp 500 1000

Apprentissage sur 500 spams et 1000 hams...
... c'est très long ...
Classifieur enregistré dans 'mon_classifieur'.

java filtre_mail mon_classifieur message.txt

D'après 'mon_classifieur', le message 'message.txt' est un SPAM !
```

— Implémentez un apprentissage en ligne pour permettre d'améliorer l'estimation des paramètres avec un nouveau message sans devoir réaliser à nouveau un apprentissage complet. Par exemple :

```
java apprend_filtre mon_classifieur baseapp 500 1000

Apprentissage sur 500 spams et 1000 hams...
...
Classifieur enregistré dans 'mon_classifieur'.

java apprend_filtre_enligne mon_classifieur newMsg.txt SPAM

Modification du filtre 'mon_classifieur' par apprentissage
sur le SPAM 'newMsg.txt'.

java apprend_filtre_enligne mon_classifieur newMsg2.txt HAM

Modification du filtre 'mon_classifieur' par apprentissage
sur le HAM 'newMsg2.txt'.
```

Attention au lissage :

$$b^j_{SPAM}(m_{spam}) = \frac{n^j_{SPAM} + \epsilon}{m_{spam} + 2\epsilon}$$

où n^j_{SPAM} est le nombre de spams contenant le mot j parmi m_{spam} au cours de l'apprentissage précédent, et, avec un exemple x en plus :

$$b^j_{SPAM}(m_{spam} + 1) = \frac{n^j_{SPAM} + x^j + \epsilon}{m_{spam} + 1 + 2\epsilon}$$