Apprentissage supervisé
M1 Informatique
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 Fichiers à disposition

Vous pouvez récupérer sur ARCHE l'archive `spam.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.

2 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.

3 Etapes à réaliser

Vous devez implémenter un filtre anti-spam basé sur le classifieur naïf de Bayes. Les grandes étapes de cette implémentation sont les suivantes (voir le fichier `tpspam.py`) :

— **Fonction** `charge_dico` : cette fonction charge un dictionnaire dans un tableau (liste python) de mots à partir d'un fichier texte donné en paramètre.

— **Fonction** `lireMail` : cette fonction lit un message (dans un fichier texte) et le traduit en une représentation sous forme de vecteur de booléens x à partir d'un dictionnaire.
En Python, les booléens valent `True` *ou* `False`*, mais sont évalués à 1 et 0 dans des opérations arithmétiques (par ex.* `4 + True` *renvoie* `5`*).*

— **Fonction** `apprendBinomial` : cette fonction apprend les paramètres d'un modèle binomial de texte à partir d'un ensemble de textes.
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).

— **Apprentissage** : l'apprentissage des paramètres du classifieur naïf de Bayes à partir de $m_{spam} + m_{ham}$ messages correspond à l'apprentissage des deux modèles de textes (un pour les spams et un pour les hams) grâce à la fonction `apprendBinomial` et des probabilités a priori $P(Y = SPAM)$ et $P(Y = HAM)$. Ces étapes sont à réaliser dans le programme principal (à la fin du fichier `tpspam.py`). Vous pourrez tester votre classifieur avec, par exemple, $m_{spam} = 200$ et $m_{ham} = 200$.

— **Prédictions** : la fonction `prediction` retourne la prédiction du classifieur pour un `x` correspondant à l'encodage binaire d'un texte.
Utilisez le fait que `x` *contienne des booléens pour calculer la prédiction du classifieur plutôt que les formes du type* $(b^j_{SPAM})^{x^j}(1-b^j_{SPAM})^{1-x^j}$ *(impliquant des puissances 0 et 1).*
Faites attention aux problèmes numériques. Par exemple, évitez les tests du type if ($10^{-354} < 10^{-342}$*)...*

— **Test** : la fonction `test` permet de tester un classifieur sur un ensemble de fichiers textes de même catégorie. Elle doit retourner l'erreur de test et afficher pour chaque texte une ligne du type
`SPAM basetest/spam/123.txt identifié comme un HAM  *** erreur ***`
ou
`HAM basetest/ham/123.txt identifié comme un HAM`
Dans le programme principal, après l'apprentissage, le classifieur devra être testé sur un ensemble de $m'_{spam} + m'_{ham}$ messages de test stockés dans 2 sous-répertoires `spam` et `ham` du dossier `basetest`. Le programme devra alors calculer le taux d'erreur sur les spams, sur les hams et le taux d'erreur global sur l'ensemble des exemples de test, par exemple :

```
Erreur de test sur 300 SPAM      : 24 %
Erreur de test sur 300 HAM       : 12 %
Erreur de test globale sur 600 mails : 18 %
```

— Modifiez la fonction `apprendBinomial` pour appliquer le lissage des paramètres avec $\epsilon = 1$ et éviter le surapprentissage.

— Le programme doit maintenant 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 ***
```

Pour cela, modifiez la fonction `prediction` pour qu'elle calcule ces probabilités et retourne trois valeurs :
`isSpam, Pspam_x, Pham_x = prediction(...)`
Essayez de limiter l'influence de la précision de la machine sur le calcul de ces probabilités a posteriori.

4 Consignes

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

— 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 `lireMail` 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. Puis, vous pourrez améliorer cette fonction.

5 Améliorations

— Pour faciliter la suite, enregistrez les paramètres du classifieur dans un objet que vous pouvez créer avec, par exemple :

```
classifieur = {}
classifieur['param1'] = ...
classifieur['param2'] = ...
```

— Vous pouvez maintenant créer une fonction `testClassifieur` qui fonctionne exactement comme `test` mais à partir d'un classifieur sous forme de structure plutôt que de toute la liste des paramètres.

— Vous pouvez maintenant enregistrer votre classifieur dans un fichier (en utilisant le module `pickle`).

— Complétez le programme pour permettre de recharger un classifieur existant et de le mettre à jour avec un apprentissage en ligne : à partir d'un seul nouveau message donné les paramètres du classifieur sont améliorés sans devoir réaliser à nouveau un apprentissage complet sur une base étendue. 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 :

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