Rapport TP

Filtre anti-spam

**Description du travail effectué**

Nous avons implémenté avec succès la majorité de ce qui est demandé.

Avec le programme actuel, nous pouvons :

* Apprendre une base d’apprentissage avec *n* spams et *m* hams, et ainsi construire un classifieur
* Utiliser un classifieur pour classifier un message quelconque
* Utiliser un classifieur sur *n* éléments d’une base de test
* Apprendre un message supplémentaire à un classifieur déjà partiellement appris

Nous n’avons cependant pas réussi à calculer la probabilité effective qu’un message soit un spam ou un ham.

Actuellement, nous pouvons discerner un spam d’un ham sans utiliser P(Y=SPAM | X=x).

En effet, cette probabilité utilise théoriquement un produit (avec beaucoup de facteurs) de probabilités. On se retrouve donc très rapidement avec un nombre très proche de zéro, qui sera potentiellement arrondi à zéro.

Pour pallier à cela, nous ne calculons pas le produit, mais le logarithme de ce produit, c’est-à-dire log(a\*b) = log(a) + log(b). On obtient une somme beaucoup plus facile à calculer (négative puisque les probabilités sont entre 0 et 1).

Nous n’avons pas réussi à déduire P(Y=SPAM | X=x) de ce calcul. Nous nous retrouvons avec un terme ressemblant à :

(exp(Z0) + log(P(Y=SPAM))) / (exp(Z0 + log(P(Y=SPAM))) + exp(Z1 + log(P(Y=HAM))))

Avec Z0 le logarithme précédent, et Z1 le même en remplaçant SPAM par HAM.

Appliquer un logarithme sur ce terme ne nous aide pas, car le dénominateur est une somme.

**Répartition du travail**

Tant que possible, nous avons travaillé en parallèle, sur des classes différentes.

Victor :

* Lecture et transformation d’un message en vecteur de booléens
* Classification d’un message
* Apprentissage d’un message supplémentaire (lissage)
* Sauvegarde et chargement d’un classifieur depuis un fichier
* Interface utilisateur pour l’apprentissage d’un message supplémentaire

Mohammade :

* Chargement d’un dictionnaire
* Apprentissage d’un classifieur
* Filtrer les mails
* Interface utilisateur pour l’apprentissage et le test
* Interface utilisateur pour l’apprentissage d’une base complète
* Interface utilisateur pour la classification d’un unique message

**Détails d’implémentation**

Le programme est séparé en plusieurs classes, dont nous allons décrire les responsabilités :

* ChargerDictionnaire lit un fichier texte pour construire un dictionnaire (tableau de mots)
* LectureMessage lit un fichier texte et à l’aide d’un dictionnaire, construit un tableau de booléens indiquant les mots du dictionnaire présents dans le message
* Classifieur a plusieurs rôles :

1. Apprendre à partir d’une base d’apprentissage et d’un dictionnaire
2. Classifier un message comme SPAM ou HAM
3. Apprendre un exemple supplémentaire (lisser)

Les autres classes sont utiles à l’exécution pour l’interface utilisateur, leurs utilisations sont décrites plus bas.

**Codes d’exécution**

La classe ApprendFiltre permet à un classifieur d’apprendre *n* spams et *m* hams dans *baseapp* puis de s’enregistrer dans le fichier mon\_classifieur.

java ApprendFiltre *mon\_classifieur baseapp n m*

La classe ApprendFiltreEnLigne permet au classifieur *mon\_classifieur* d’apprendre le nouveau message *newMsg.txt* sans devoir tout réapprendre.

java ApprendFiltreEnLigne *mon\_classifieur* *newMsg.txt* SPAM

La classe FiltreAntiSpam permet au classifieur *mon\_classifieur* de tester sur la base de test *basetest* avec *n* spams et *m* hams.

java FiltreAntiSpam *mon\_classifieur basetest n m*

La classe FiltreMail indique, d’après le classifieur *mon\_classifieur*, si le message *msg.txt* est un spam ou non.

java FiltreMail *mon\_classifieur msg.txt*

**Exécutions**

java ApprendFiltre classifieur res/baseapp 300 200

Apprentissage sur 300 spams et 200 hams ...

Classifieur enregistré dans 'classifieur'.

java FiltreAntiSpam classifieur res/basetest 500 500

...

SPAM n°488 détecté comme HAM : \*\*ERREUR\*\*

...

HAM n°499 détecté comme HAM : OK

=====================================================================================

Erreur de test sur les 500 SPAM : 4.8 %

Erreur de test sur les 500 HAM : 0.4 %

Erreur de test globale sur 1000 mails : 2.6 %

java ApprendFiltreEnLigne classifieur res/baseapp/spam/300.txt SPAM

Modification du filtre 'classifieur' par apprentissage sur le SPAM 'res/baseapp/spam/300.txt'.

java FiltreMail classifieur res/basetest/ham/250.txt

D'après 'classifieur', le message 'res/basetest/ham/250.txt' est un HAM

java FiltreMail classifieur res/basetest/spam/250.txt

D'après 'classifieur', le message 'res/basetest/spam/250.txt' est un SPAM

**Performance**

Comme montré ci-dessus, avec 300 spams et 200 hams dans la base d’apprentissage, on obtient 4.8% d’erreurs sur les spams, 0.4% sur les hams, 2.6% globalement.

Nous pouvons encore améliorer ces chiffres, en prenant 400 spams et 300 hams.

*Erreur de test sur les 500 SPAM : 2.4 %*

*Erreur de test sur les 500 SPAM : 2.4 %*

*Erreur de test globale sur 1000 mails : 1.4000000000000001 %*

Il ne faut cependant pas en prendre trop. Avec 500 spams et 500 hams, on détériore les performances :

*Erreur de test sur les 500 SPAM : 3.0 %*

*Erreur de test sur les 500 SPAM : 0.4 %*

*Erreur de test globale sur 1000 mails : 1.7000000000000002 %*