

# Projet d'apprentissage: filtre spam

Baptiste Lesquoy

30 mai 2016

# Table des matières

<b>1</b>	<b>Répartition des tâches</b>	<b>2</b>
<b>2</b>	<b>La solution mise en œuvre</b>	<b>3</b>
<b>3</b>	<b>difficultés/choix effectués</b>	<b>4</b>
3.1	lire message/charger dictionnaire . . . . .	4
3.2	Apprentissage/prédiction . . . . .	4
3.3	enregistrer le classifieur / ajouter un message au classifieur . .	5
3.4	Remarques générales . . . . .	6
3.4.1	prérequis . . . . .	6
3.4.2	exécutions du programme . . . . .	6
3.4.3	performance . . . . .	8

# Chapitre 1

## Répartition des tâches

Je suis tout seul, j'ai donc tout fait :) .

## Chapitre 2

### La solution mise en œuvre

Il s'agit d'un filtre anti-spam, toutes les parties prévues dans le projet ont été mise en place, même les bonus.

# Chapitre 3

## difficultés/choix effectués

### 3.1 lire message/charger dictionnaire

Pour cette partie, il a fallu faire attention à découper les mots le plus précisément possible. En effet, si on se contente de découper les mots à l'aide des espaces, on se retrouve avec des mots tel que : "que :", qui ne seront jamais reconnu par le dictionnaire. Il faut donc bien faire attention à découper en fonction de tous les caractères qu'on pourrait retrouver dans un texte. Au final le programme découpe selon 28 caractères différents. Il ne fallait aussi pas oublier d'enregistrer les mots du dictionnaire et de comparer les mots des mails en majuscule pour éviter tout problème de casse.

### 3.2 Apprentissage/prédiction

Pour implémenter l'apprentissage il était important d'avoir les idée claire sur ce qu'était un classifieur de Bayes, ainsi que plusieurs notions de cours. Aussi, obtenir la probabilité qu'un mot appartienne à un spam ou à un ham nécessite de multiplier beaucoup de probabilités entre eux, et donc d'avoir des probabilité de plus en plus petite. Or une machine a une limite de précision, et avec ce projet on pouvait l'atteindre sur certains exemples. C'est pourquoi j'ai utilisé des logarithmes, puisque, ce qui nous intéresse dans la probabilité c'est de pouvoir la comparer à une autre, on peut les modifier tant qu'on ne change pas l'ordre. Or  $\log(a * b) = \log(a) + \log(b)$ , et si  $a \leq b$ , alors  $\log(a) \leq \log(b)$ . Donc on peut utiliser un logarithme pour ne plus avoir à multiplier les probas entre elles, mais plutôt les additionner, on ne perd ainsi plus en précision tout en conservant l'ordre entre les deux. De même, comme on compare  $P(Y=SPAM|X=x)$  avec  $P(Y=HAM|X=x)$  et que les valeurs ne nous intéressent pas, pas besoin de calculer  $\frac{1}{P(X=x)}$  puisqu'on le retrouve dans

les deux probabilités.

### 3.3 enregistrer le classifieur / ajouter un message au classifieur

Pour le mettre en place, j'ai décidé d'utiliser la sérialisation de java. C'est à ce moment que j'ai trouvé plus pratique de créer une classe regroupant les différents tableaux que j'utilisais et qui représenterait le classifieur à un instant donné. De plus, on ne conserve plus les probabilité, mais directement le nombre de présence de tel ou tel mot dans les spam et dans les ham, ainsi que les nombres de messages de chaque. Ceci permet de continuer l'apprentissage plus tard comme si on s'était arrêté au milieu, ou d'utiliser le classifieur tel quel.

Pour la partie deux du bonus, on peut vérifier que l'ajout un à un donne bien le même classifieur que la création à partir d'une base. Pour ce faire, on commence par créer un classifieur sur toute la base d'apprentissage.

```
[baptiste@Serenity SpamHam]$ java FiltreAntiSpam apprend_filtre mon_classif baseapp/ 2500 500
repertoire: baseapp/ nbHam: 2500 nbSpam: 500
learning ...
la limite: 500 n'a pas été dépassée: 500 fichiers lus
la limite: 2500 n'a pas été dépassée: 2500 fichiers lus
classifieur enregistré dans mon_classif
```

FIGURE 3.1 – création d'un classifieur sur toute la base

Puis on crée un classifieur à partir d'une base vide

```
[baptiste@Serenity SpamHam]$ java FiltreAntiSpam apprend_filtre mon_classif2 baseVide/ 0 0
repertoire: baseVide/ nbHam: 0 nbSpam: 0
learning ...
la limite: 0 n'a pas été dépassée: 0 fichiers lus
la limite: 0 n'a pas été dépassée: 0 fichiers lus
classifieur enregistré dans mon_classif2
```

FIGURE 3.2 – création d'un classifieur sur une base vide

Ensuite on ajoute tous les fichiers de la base dans le classifieur vide

```
[baptiste@Serenity SpamHam]$ java FiltreAntiSpam filtre_en_ligne mon_classif2 baseapp/spam/* SPAM
classifieur chargé depuis mon_classif2
classifieur enregistré dans mon_classif2
[baptiste@Serenity SpamHam]$ java FiltreAntiSpam filtre_en_ligne mon_classif2 baseapp/ham/* HAM
classifieur chargé depuis mon_classif2
classifieur enregistré dans mon_classif2
```

FIGURE 3.3 – ajout des spam et des ham

Puis on compare les deux classifieurs sur un même fichier. On devrait obtenir exactement la même chose

```
[baptiste@Serenity SpamHam]$ java FiltreAntiSpam filtre_mail mon_classif basetest/spam/59.txt
classifieur chargé depuis mon_classif
P(Y=SPAM | X=x) = 2.7866565120902627E-61, P(Y=HAM | X=x) = 6.367331726376351E-64
=> identifié comme un spam
d'après le classifieur mon_classif
[baptiste@Serenity SpamHam]$ java FiltreAntiSpam filtre_mail mon_classif2 basetest/spam/59.txt
classifieur chargé depuis mon_classif2
P(Y=SPAM | X=x) = 2.7866565120902627E-61, P(Y=HAM | X=x) = 6.367331726376351E-64
=> identifié comme un spam
d'après le classifieur mon_classif2
```

FIGURE 3.4 – comparaison des classifieurs

## 3.4 Remarques générales

### 3.4.1 prérequis

Il faut que le fichier "dictionnaire1000en.txt" soit à côté du programme.

### 3.4.2 exécutions du programme

#### le manuel

si vous ne savez pas comment utiliser le programme, lancez simplement la commande

```
java FiltreAntiSpam
```

et le manuel du programme s'affichera.

#### Utilisation standard

Pour lancer un apprentissage sur la base "baseapp" et observer le résultat sur une base de test lancez :

```
java FiltreAntiSpam base_test nb_Ham nb_Spam
```

Où *base\_test* est le répertoire de votre base de test, et *nb\_Ham* et *nb\_Spam* les nombres respectif de HAM et de SPAM que vous voulez tester dans cette base.

```
[baptiste@Serenity SpamHam]$ java FiltreAntiSpam basetest/ 100 200
nombre de HAM à apprendre ?
100
nombre de SPAM à apprendre ?
200
repertoire: baseapp nbHam: 100 nbSpam: 200
learning ...

puis après avoir tout testé
HAM numéro: 200: P(Y=SPAM | X=x) = 9.278796265663556E-68, P(Y=HAM | X=x) = 4.688187596912151E-62
=> identifié comme un ham
échec SPAM: 1-(89/101) =11.88%
échec HAM: 1-(190/201) =5.47%
échec totale: 1-(279/302) =7.62%
```

FIGURE 3.5 – exemple d'utilisation standard

### enregistrer un filtre

Pour enregistrer un filtre, il faut taper la commande :

```
java FiltreAntiSpam apprend_filtre fichier_classif base_apprentissage nb_Ham nb_Spam
```

Où fichier\_classif est le fichier dans lequel vous voulez enregistrer le classifieur, base\_apprentissage est le répertoire de votre base d'apprentissage, et nb\_Ham et nb\_Spam les nombres respectif de HAM et de SPAM que vous voulez tester dans cette base.

```
[baptiste@Serenity SpamHam]$ java FiltreAntiSpam apprend_filtre mon_classif baseapp/ 200 200
repertoire: baseapp/ nbHam: 200 nbSpam: 200
learning ...
classifieur enregistré dans mon_classif
```

FIGURE 3.6 – exemple d'utilisation de l'enregistrement de classifieur

### tester un mail sur un filtre enregistré

Pour tester un mail sur un filtre enregistré, il faut taper la commande :

```
java FiltreAntiSpam filtre_mail fichier_classif mail_a_tester
```

Où fichier\_classif est le fichier dans lequel est enregistré le classifieur et mail\_a\_tester le fichier du mail à tester.

```
[baptiste@Serenity SpamHam]$ java FiltreAntiSpam filtre_mail mon_classif basetest/spam/89.txt
classifieur chargé depuis mon_classif
P(Y=SPAM | X=x) = 3.9188591503457355E-131, P(Y=HAM | X=x) = 3.365597052434455E-144
=> identifié comme un spam
d'après le classifieur mon_classif
```

FIGURE 3.7 – exemple de test sur un classifieur enregistré

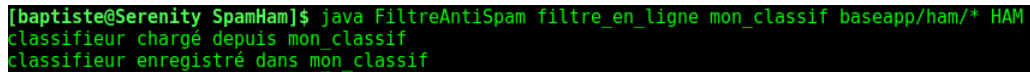


## Apprendre à un filtre enregistré

Pour apprendre un mail à un filtre enregistré, il faut taper la commande :

```
java FiltreAntiSpam filtre_en_ligne fichier_classif  
liste_fichier_a_ajouter HAM/SPAM
```

Où `fichier_classif` est le fichier dans lequel est enregistré le classifieur, `liste_fichier_a_ajouter` correspond à une liste de fichier (qui peut être vide, ou n'être qu'un fichier) que vous voulez apprendre à votre classifieur, et `HAM/SPAM` correspond au type de fichiers que vous ajoutez (donc soit HAM, soit SPAM). Le classifieur doit déjà exister pour qu'on puisse lui apprendre de nouveaux fichiers.



```
[baptiste@Serenity SpamHam]$ java FiltreAntiSpam filtre_en_ligne mon_classif baseapp/ham/* HAM  
classifieur chargé depuis mon_classif  
classifieur enregistré dans mon_classif
```

FIGURE 3.8 – exemple d'apprentissage sur un filtre enregistré

### 3.4.3 performance

Je suppose que le filtre est plutôt performant, avec les meilleurs paramètres on arrive en dessous de 5 pourcents d'erreur. À ce sujet on constate qu'il peut y avoir du surapprentissage comme le montre le graphique représentant le taux d'erreur en fonction de l'apprentissage ( $x=500 \Rightarrow$  toute la base a été apprise) sur le même jeu de test. On remarque que passé un certain moment le taux d'erreur ne fait qu'augmenter, faisant penser à du surapprentissage. Néanmoins il faudrait une base bien plus grande pour voir si on constate une vraie tendance à l'augmentation ou s'il s'agit d'un maximum local.

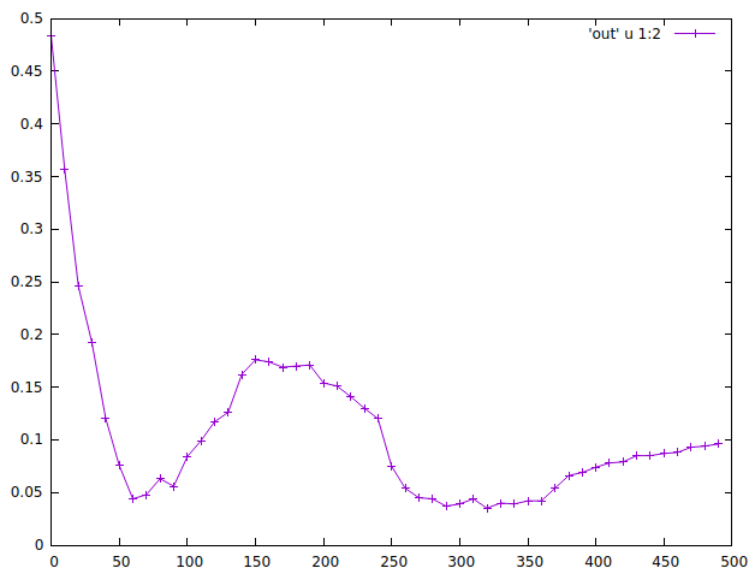


FIGURE 3.9 – Évolution du taux d'erreur en fonction du nombre de message appris

Pour ce qui est du temps que le filtre met à apprendre, il semble raisonnable pour un programme java. On peut espérer augmenter ce dernier en parallélisant/distribuant la lecture des fichiers.

### 3.5 compilation

Vous pouvez compiler exactement comme décrit dans le sujet, à la différence près que ma classe principale s'appelle "FiltreAntiSpam" avec un f majuscule.