Filtre Anti-SPAM

Introduction à l'apprentissage

Encadré par Fabien Lauer

Rédigé par Valéry LAMBLE et Thomas DENIS

Master 1 Informatique Université de Lorraine

Table des matières

Introduction

Il est aujourd'hui trop courant de voir une boite mail pollué par des e-mails indésirables. La science de l'apprentissage permet de pallier à ce problème, en effet, en connaissant un nombre suffisant d'e-mails indésirable (SPAM) et désirable (HAM), il est possible de prédire avec des probabilités satisfaisante la désirabilité des futurs e-mails reçus et ainsi les catégoriser comme SPAM ou HAM.

Dans ce projet, nous réalisons un filtre anti-SPAM reposant sur un classifieur naïf de Bayes. Nous vous présenterons dans un premier temps, l'aspects théorique, puis nous montrerons les fonctions de notre programme réalisant cette classification et nous termineront par une analyse des performances.

1. Approche théorique du filtre anti-spam

a. Structure du projet

On distingue deux dossiers et un fichier

- Le dossier **baseapp** contient deux sous dossiers, ham et spam, contenant respectivement 2500 hams et 500 spams. Chaque message est un fichier texte nommé X.txt où X représente l'indice du message. Ces messages sont destinés à l'apprentissage.
- Le dossier **basetest** contient également deux sous dossiers, ham et spam, contenant respectivement 500 hams et 500 spams. Ces messages sont destinés aux tests.
- Le fichier **dictionnaire1000en.txt** est un fichier texte contenant 1000 mots anglais parmi les plus couramment

Le programme principale est FiltreAntiSpam.java

b. Explication pas à pas

- Dans un premier temps, il s'agit de charger un dictionnaire de mot, ici nous avons un dictionnaire de 1000 mots usuels anglais. Par la suite on exclura les mots de 3 caractères et moins.

- Pour créer l'apprentissage, nous allons créer des tableaux de probabilités ayant le même nombre de lignes que le nombre de mots dans le dictionnaire. Cette fonction d'apprentissage va prendre n messages en entrées avec un dictionnaire de m mots et calculer la probabilité d'apparitions de chaque mot du dictionnaire dans l'ensemble de ces n messages, la fonction retournera un table d'une colonne et de m lignes. Ce tableau correspond au vecteur de présence.
  La base d'apprentissage sera constituée d'un tableau d'apprentissage pour les HAM et un second pour les SPAM.

- Nous devrons alors calculer

$$P(Y = SPAM|X = x) = \frac{1}{P(X = x)} P(Y = SPAM) \prod_{j=1}^{d} \left(b_{SPAM}^{j}\right)^{x^j} \left(1 - b_{SPAM}^{j}\right)^{1-x^j}$$

$$P(Y = HAM|X = x) = \frac{1}{P(X = x)} P(Y = HAM) \prod_{j=1}^{d} \left(b_{HAM}^{j}\right)^{x^j} \left(1 - b_{HAM}^{j}\right)^{1-x^j}$$

Remarquons que les b_{SPAM}^{j} et b_{HAM}^{j} sont les valeurs des probabilités du j[ième] mot du dictionnaire dans la base d'apprentissage SPAM et HAM.

- Nous déciderons alors de la nature prédite du message en appliquant la formule suivante :

$$\begin{cases} \text{SPAM si } P(Y = SPAM|X = x) > P(Y = HAM|X = x) \\ \text{HAM sinon} \end{cases}$$

- Nous ferons une fonction de test pour essayer le programme sur un grand nombre de message et faire varier la barre d'apprentissage avec le nombre de HAMS et de SPAMS appris.

2. Présentation des fonctions

a. La classe FiltreAntiSpam

La classe FiltreAntiSpam a pour attributs :

ArrayList<String> dico	Correspond au dictionnaire
static int M_SPAM	Nombre de Spams pour la base d'apprentissage appSpam
static int M_HAM	Nombre de Hams pour la base d'apprentissage appHam
static int NB_HAM	Nombre de Hams à tester
static int NB_SPAM	Nombre de Spams à tester
double[] appSpam;	Vecteur de présences contenant les probabilités des mots du dictionnaire pour le mSpam (correspond au b_{SPAM}^{j})
double[] appHam	Vecteur de présences contenant les probabilités des mots du dictionnaire pour le mHam (correspond aux b_{HAM}^{j})

Le constructeur FiltreAntiSpam(int nbSpam,int nbHam,String dic) initialise donc le nombre de hams et de spams appris ainsi que le dictionnaire à utiliser.

b. charger_dictionnaire(String dic)

Cette fonction vas ajouter chaque mot du fichier rentré en paramètre à l'ArrayList<String> dico sous condition que le fichier du dictionnaire sois à la source du projet et au format .txt

c. int[] lire_message(String fichier)

Cette fonction retourne un tableau d'entier de longueur de la taille du dico, avec pour chaque valeur est égal à 1 si le mot correspondant du dictionnaire est présent dans le message entré en paramètre, et 0 sinon.

Pour séparer chaque mot au mieux, il a fallu considéré certains caractères tels que les apostrophes, les caractères d'accentuations collés ou autres. Nous avons utilisé le regex *str.replaceAll("[\\W]", " ").split(" ")* Le fichier entrer en paramètre doit être un .txt

d. void apprentissage(int nbMess, String base)

La fonction apprentissage calcul les vecteurs de présence pour les M_SPAM ou M_HAM rentrés en paramètres et retourne un tableau de probabilités de ses présences.

e. String probDeSachant(String msg)

Calcul $P(Y = SPAM|X = x)$ et $P(Y = HAM|X = x)$ et d'autres probabilités énoncées dans le cours pour décider si le message est un Spam ou un Ham, on fait donc une comparaison entre ces deux probabilités citées précédemment.

f. void test()

La fonction test lance le filtre et prédit les étiquettes des NB_HAM et des NB_SPAM. La fonction affiche les résultats ainsi que les statistiques d'erreurs.

3. Quelques chiffres et autres remarques

Tout d'abord, nous avons réalisé la première partie de l'amélioration (Bonus). Nous pouvons stocker un classifieur dans un fichier texte qui nous lisons après pour analyser un nouveau message grâce à deux nouvelles classes qui sont **ApprendFiltre** et **FiltreMail**.

L'exécution des différents classes se fait de la manière suivante :

Pour la classe **FiltreAntiSpam** :

- javac FiltreAntiSpam.java
- java FiltreAntiSpam basetest 100 200
  100 représente le nombre de spam et 200 le nombre de ham que l'on veut filtrer

Voici les résultats de l'exécution suivante : *java FiltreAntiSpam basetest 500 2500*

```
Erreur de test sur les 500 SPAM : 23.799999999999997%
Erreur de test sur les 500 HAM : 1.2%
Erreur de test globale sur les 1000 mails : 12.499999999999998%
```

Remarque : Nous avons quelques valeurs qui sont en NaN, nous ne savons pas pourquoi, dans ce cas, nous retournons toujours un Ham. Nous avons utilisé les Logs pour essayer d'améliorer nos résultats mais il n'y a aucun changement.

Pour la classe **ApprendFiltre** :

- javac ApprendFiltre.java
- java ApprendFiltre mon_classifieur baseapp 500 1000

Il faut que message.txt existe dans la racine. Et **mon_classifieur** est le nom auquel vous voulez donner au classifieur. Il générera un « mon_classifieur.txt »

Pour la classe **FiltreMail** :

- javac FiltreMail.java
- java FiltreMail mon_classifieur message.txt

Exécution du filtre sur « message.txt » et renvoie si ce message est un SPAM ou un HAM.