ministère éducation nationale

Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

éduscol

Ressources pour le lycée général et technologique

Ressources pour le cycle terminal général et technologique

Informatique et Sciences du Numérique

Le filtrage du « spam »

Juin 2012

Présentation / Le filtrage du « spam »

1 / Thème abordé

1.1 Problématique, situation d'accroche

Les courriels non sollicités, ou *spams*, sont une plaie de la communication électronique actuelle, pouvant représenter jusqu'à 95% du volume de courrier électronique traité sur certains serveurs. C'est donc un enjeu éthique et économique de premier plan que de lutter efficacement contre cette plaie. Bien que la décision « *spam / non-spam* » soit le plus souvent facile à prendre pour un humain, la masse des messages en circulation empêche de traiter manuellement le nécessaire tri entre les courriels acceptables et les autres.

La présente ressource propose de mettre en place quelques principes généraux de la détection automatisée de contenus textuels dans un cadre restreint et simplifié.

À titre introductif, un petit défi peut être proposé, consistant à retrouver un message inséré dans un support textuel plus vaste, soit pour l'extraire soit pour l'éliminer : un devoir de mathématiques glissé dans un texte anodin, une ou plusieurs publicités dans une page web, une ou plusieurs chaînes de bases dans un brin d'ADN.

1.2 Le travail proposé

Il s'agit alors de rechercher certains mot-clés en estimant empiriquement leur probabilité d'apparition dans une phrase à caractère mathématique (ou culinaire). On présente donc le concept d'expression régulière par la pratique[1] sans en faire une étude approfondie, en faisant l'analogie avec les méthodes utilisées par les logiciels anti-spam ou les bloqueurs de publicités. Après cela, l'étude se poursuit dans le cadre d'un TP sur plusieurs séances ou d'un mini-projet.

1.3 Frontières de l'étude et prolongements possibles

On pourra d'ailleurs proposer :

- dans le cadre d'un projet et/ou d'une réalisation plus conséquente, l'installation et la configuration d'un serveur de messagerie (accessible par webmail) et d'un logiciel anti-spam tel que spamassassin) ;
- la réalisation d'exposés et/ou débats autour des bons usages de la messagerie électronique (« nétiquette ») et des précautions à adopter (développer son esprit critique vis-à-vis des spams, chaîne, canular, phishing etc). Ce genre de travail pourra être également proposé à d'autres élèves de l'établissement sous forme d'exposition ou de mini-conférence/débat.

La maîtrise des recherches par expressions régulières ouvre la voie à d'autres importants domaines applicatifs comme la recherche sur les codes génétiques ou certaines démarches typiques de l'informatique décisionnelle.

2 / Objectifs pédagogiques

2.1 Disciplines impliquées

L'analyse des langues[2] est un des domaines de recherche les plus actifs pour l'informatique et les sciences du numérique ; dans le présent contexte, quelques connaissances de base en linguistique française seront mobilisées.

Des connexions étroites avec le programme de mathématiques de terminale S[3] sont à envisager, tout particulièrement en statistiques et en probabilités dont voici un extrait :

> Construire un arbre pondéré en lien avec une situation donnée. Exploiter la lecture d'un arbre pondéré pour déterminer des probabilités. Calculer la probabilité d'un événement connaissant ses probabilités conditionnelles relatives à une partition de l'univers.
>
> Le réinvestissement des notions vues en probabilités conditionnelles dans ce cadre sont l'occasion de leurs donner du sens. On pourra à l'aide d'un arbre de probabilités conditionnelles introduire la formule de Bayes.

Des liens peuvent être aussi établis avec les SVT en illustrant le principe des « faux positifs ».

1 Un premier contact peut se limiter aux caractères spéciaux : `. * + ^ $ [ ]`

2 Cela s'appelle le traitement automatique de la langue naturelle (TALN).

3 http://www.education.gouv.fr/pid25535/bulletin_officiel.html?cid_bo=57529

2.2 Prérequis

Des notions d'algorithmique de base (boucles, test, entrée/sortie) et des connaissances concernant les probabilités conditionnelles (formule des probabilités totales) sont nécessaires.

2.3 Éléments du programme

Contenus

- Structuration et organisation de l'information
- Programmation : types de données
- Persistance de l'information (une adresse mail incluse dans une page web est une donnée persistante)
- Supra-nationalité des réseaux (les législations nationales sur le spam et autres pratiques diffèrent)

Compétences et capacités

Concevoir et réaliser une solution informatique en réponse à un problème :

- Observer, analyser un problème et proposer une solution algorithmique ;
- Choisir un type de données en fonction d'un problème à résoudre ;
- Concevoir et programmer un algorithme ;
- Mettre un programme au point en le testant, en l'instrumentant.

Collaborer efficacement au sein d'une équipe dans le cadre d'un projet :

- Travailler en équipe, rechercher une information.

Faire un usage responsable des sciences du numérique :

- Mesurer les limites et les conséquences de la persistance de l'information numérique, des lois régissant les échanges numériques, du caractère supranational des réseaux.

3 / Modalités de mise en œuvre

3.1 Durée prévue pour la partie se déroulant en classe

1 heure pour la réflexion animée par l'enseignant, 4 heures en classe et travail de mise au point hors classe.

3.2 Type de l'animation

Présentation et délimitation du problème, suivies d'une réflexion commune en groupe classe avec l'enseignant puis constitution d'équipes de 2 à 3 élèves pour la réalisation de la tâche attendue.

3.3 Production des élèves

Les différentes équipes constituées de deux à trois élèves doivent réaliser un script capable de « parser » un texte de référence.

3.4 Évaluation

Les élèves sont évalués en partie sur l'efficacité de leur script (benchmark avec le texte de référence) et en partie sur les compétences mentionnées ci-dessus.

4 / Outils

Environnement de développement et de programmation (voir les références à la fin du document).

5 / Auteur

Abdellatif KBIDA, professeur de Mathématiques, académie de Nancy-Metz, avec l'aide de :

Yannick PARMENTIER, maître de conférence de l'université d'Orléans, laboratoire d'informatique fondamentale
Valéry FEBVRE, chef de projet, société Easter-eggs.

Le filtrage du « spam »

1 / Première partie

1.1 Problématique

« Être le destinataire d'un courrier électronique non-sollicité », chacun d'entre nous en a fait la désagréable expérience. On perçoit sans difficulté et à coup sûr si un message constitue une information valide ou si son contenu est un message non-sollicité du type « spam ». Cependant, devant la masse des messages en circulation, on se trouve dans l'incapacité de les traiter manuellement, il est donc indispensable d'imaginer des procédés relativement fiables pour automatiser ce genre de tâche que notre cerveau accomplit si bien. Des logiciels, tant du côté serveur que du côté client, travaillent quotidiennement à filtrer le spam sans même que l'on s'en rende compte.

Le terme *spam* provient d'un sketch des Monthy Python, comiques britanniques à l'humour décalé. Un couple s'installe à la table d'une taverne, demande le menu et se voit harceler par la tenancière qui ne cesse de leur proposer du spam (contraction de *spiced ham*)[4]. L'explication complète peut être lue ici : http://fr.wikipedia.org/wiki/Spam

Le travail proposé ici consiste à illustrer les principes généraux de la détection automatisée de contenus textuels dans un cadre restreint et simplifié.

1.2 Situations d'accroche

On peut présenter le défi de la détection du spam sous une forme amusante :

Retrouver une aiguille dans une botte de foin !

« Voici un texte d'environ 1500 lignes. Ce texte contient l'énoncé de votre prochain devoir de mathématiques. Malheureusement chacune des phrases de ce devoir est disséminée parmi d'autres phrases tout à fait banales. Vous devez mettre en œuvre une méthode algorithmique permettant de retrouver tout ou du moins une grande partie des phrases de ce devoir.»

D'autres domaines (vocabulaires) peuvent être utilisés à la place des mathématiques, le vocabulaire culinaire pourrait tout aussi bien fonctionner du moment que les mots sont nettement identifiables.

Une accroche alternative pourrait s'attacher au web :

Le web aseptisé !

« Les pages du web sont de plus en plus envahies de publicités au point de devenir illisibles. Vous allez imaginer une sorte de moulinette qui prendrait en entrée une page web et produirait en sortie la même page mais débarrassée de certains contenus ou liens publicitaires. Attention, la page doit rester syntaxiquement correcte (du point de vue du langage HTML). Indication : la plupart des liens publicitaires font apparaître des expressions telles que *adserver* ou *adverts* (pour *advertisements*) ou *doubleclick* … »

1.3 Débat autour de la problématique

Notions : Sensibilisation aux problèmes sociétaux induits.

On propose une réflexion aux élèves en les questionnant : « Avez-vous déjà entendu parler du SPAM ? En avez-vous déjà reçu ? etc.» On suscite un débat qui sera l'occasion de traiter de certains enjeux sociétaux et de développer le sens critique des élèves face à des courriers non sollicités.

On peut soit interroger directement les élèves soit leurs présenter une situation qui va naturellement engendrer le débat. Ouvrir par exemple une boîte mail contenant un grand nombre de messages indésirables ou fournir aux élèves un corpus de courriers suspects ou bien encore visionner le sketch des MonthyPython.

Contrairement au monde physique, un simple autocollant « Stop Spam » ne suffit pas à se prémunir du spam. Les logiciels anti-spam travaillent le plus souvent en amont et de façon transparente. Tous les élèves n'ont pas conscience de ce filtrage anti-spam, il est intéressant de les interroger sur ce point. Susciter la curiosité et montrer des aspects méconnus des sciences du numérique est un objectif essentiel d'ISN. « Comment lutter contre ce fléau numérique ? ???» est une question qui se pose naturellement. On propose alors aux élèves

4 L'image provient de Wikimedia Commons : http://commons.wikimedia.org/wiki/File:SpamInACan.jpg

d'illustrer une des méthodes de lutte contre le spam : « le filtrage par méthode probabiliste » dans un cadre restreint.

On peut alors commencer à faire chercher aux élèves quelques méthodes de traitement sur la situation d'accroche présentée auparavant.
Il ne s'agit pas de demander aux élèves de parvenir à résoudre de but en blanc ce problème mais plutôt de les accompagner en leur proposant des étapes intermédiaires.

2 / Lancement des mini-projets ou du TP

2.1 Première version

On fournit un texte allégé aux élèves et on leur demande de trouver les phrases contenant le mot « fonction »; pour simplifier, on remplace la notion de « phrase » par celle de « ligne »[5], ensemble de caractères s'achevant par un caractère de fin de ligne ou « retour chariot »[6]. Pour accéder au texte, on peut proposer une documentation simplifiée contenant des exemples de code pour ouvrir, lire, fermer un ficher et pour chercher une chaîne de caractères.

Notions abordées : manipuler des fichiers avec un langage de programmation, choisir des types de données.

En Python, on accède aux fichiers via l'objet fichier créé dès l'appel de la fonction *open()*. On peut ouvrir le fichier en écriture (paramètre "w"=*write*), en lecture (paramètre "r"=*read*) ou en ajout (paramètre "a"=*append*) puis utiliser des méthodes spécifiques de cet objet fichier.

Ouverture et fermeture simple d'un fichier texte en lecture seulement :

```
texte = open("monfichier.txt", "r")
texte.close()                          # Appel de la méthode close de l'objet texte (fichier)
```

Dans la suite, il est nécessaire de traiter le contenu du fichier, on va parcourir le contenu ligne par ligne et chercher le mot « fonction » :

Parcours du fichier et recherche du mot « fonction » avec la méthode find() [7]:

```
texte = open("monfichier.txt", "r")
for ligne in texte:                    # Pour chaque ligne (chaîne de caractères) du texte faire.
      if ligne.find("fonction")>0 :    # Appel de la méthode find de l'objet ligne (caractères)
            print "***fonction dans cette ligne**** :",ligne
texte.close()
```

Dans le script ci-dessus on se contente d'afficher les résultats, bien sûr on peut les sauvegarder dans un fichier. Pour cela on peut utiliser la méthode *write()*.

Sauvegarde des résultats dans un fichier à l'aide de la méthode write() :

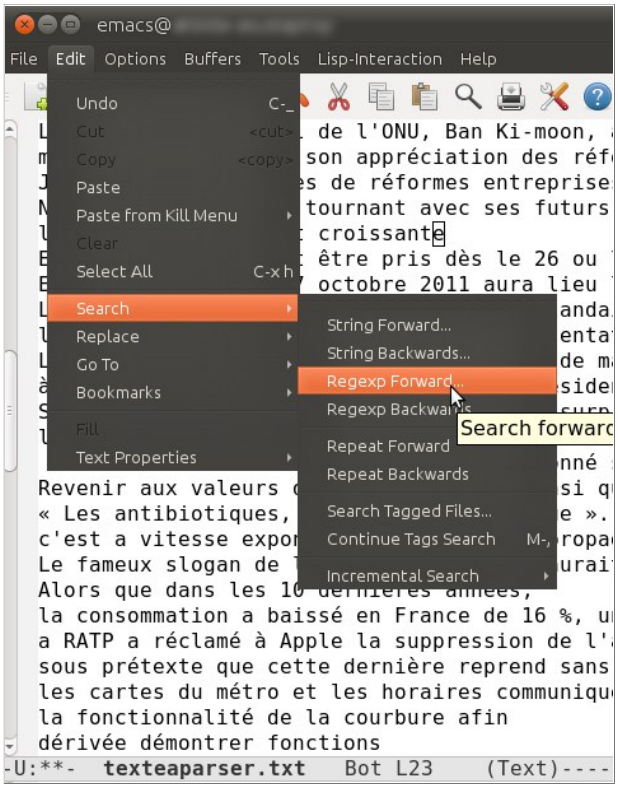
```
texte = open("texteaparser.txt", "r")
spam = open("resultat.txt", "a")
for ligne in texte:
      if ligne.find("fonction")>0 :
            spam.write(ligne)    # Appel de la méthode write de l'objet spam (fichier)
texte.close()                    # Appel de la méthode close de l'objet texte (fichier)
spam.close()                     # Appel de la méthode close de l'objet spam (fichier)
```

***Bilan:** en langage Python on passe par l'objet fichier, l'aspect programmation orientée objet (très limité dans ce cas) peut perturber, il ne faut pas hésiter à multiplier les exemples.*

On constate en conclusion de cette activité que les résultats sont perfectibles. Certains mots « fonction » passent au travers des mailles du filet comme par exemple « Fonction ». D'autres sont effectivement trouvés mais ne sont pas des phrases à caractère mathématique « fonctionnement », « fonctionnalité » ou « La fonction principale d'un téléphone portable est-elle de téléphoner ? ? » .

5 Les lignes du texte seront accessibles par l'objet `ligne` et on y cherchera des mots par la méthode `ligne.find()` .
6 Le caractère exact qui termine les lignes de texte dépend du système d'exploitation.
7 `find()` renvoie la position de la première occurrence si elle est trouvée, 0 sinon.

2.2 Expressions régulières



« Confrontées à un certain problème, certaines personnes se disent "ah je sais, je vais utiliser des expressions régulières". Maintenant, ils ont deux problèmes !» James Zawinski

Pour améliorer le script on peut tout d'abord essayer d'ignorer la casse (majuscule / minuscule) ; l'idée d'utiliser des expressions régulières s'impose assez naturellement.

Avant de se lancer dans l'implémentation de fonctions exploitant la puissance des expressions régulières, il peut être utile de faire un point sur celles-ci sous forme de TD expliquant leurs principes. On pourra proposer de petites activités mettant en œuvre des expressions régulières par l'intermédiaire de logiciels. C'est possible avec certains éditeurs de textes avancés comme Notepad++, Emacs ou encore Nedit, mais on peut aussi faire appel à l'utilitaire Grep en ligne de commande.

Il conviendra en premier lieu de s'interroger sur ce qu'on appelle « texte » puis « mot dans un texte ». Pour simplifier, on peut convenir qu'un « mot » est une suite de caractères alphabétiques terminant sur un caractère non-alphabétique ou une fin de ligne.

Voici quelques exemples de tâches réalisables avec des expressions régulières :

- déterminer le nombre de fois que le mot *schtroumpf* et ses variantes (*schtroumpfette*) apparaissent dans un album de *schtroumpfs*.
- rechercher des champs heure *xx*h*xx* ou *xx*H*xx* ou *xx*:: *xx*, les *x* représentant des valeurs numériques
- ouvrir une page html et retrouver toutes les adresses email qu'elle contient[8], puis pour lutter contre les aspirateurs d'adresse de courrier électronique changer le caractère @ par _at_
- repérer les dates exprimées au format français (31/12/1970) et les convertir au format normalisé ISO 8601 (1970-12-31).

Contexte culturel : les expressions régulières sont un exemple d'un langage « autre que les langages de programmation ». L'utilisation d'une bibliothèque telle que « re » (Regular Expressions) facilitant le développement et la lecture de sa documentation sont des activités formatrices et indispensables en informatique.

De retour au script de recherche, en ce qui concerne les expressions régulières on n'aura besoin que d'une recherche insensible à la casse sur le radical d'un mot.

Méthode `search()` *et sauvegarde des résultats dans un fichier :*

```
import sys, os, re
texte = open("texteaparser.txt", "r")
spam = open("resultat.txt", "w")
for ligne in texte:
      m = re.search("fonction",ligne,re.IGNORECASE) # Lance et sauvegarde la recherche
                                                    # m est du type booléen (vrai ou faux)
      if m :
            spam.write(ligne)          # Appel de la méthode write de l'objet spam (fichier)
            print ligne," ",m.start()  # Sauvegarde la position (ligne, colonne)
texte.close()
spam.close()
```

2.3 Les probabilités à la rescousse

« Polysémie : caractère d'un signe qui possède plusieurs contenus, plusieurs sens. » Le petit Robert.

8 Assez difficile …

Les mathématiciens (et ils ne sont pas les seuls), plutôt que de créer de nouveaux mots pour nommer des concepts, ont l'habitude d'employer des mots existants évocateurs, en leur donnant ainsi un nouveau sens dans ce contexte.

Dans les expressions « L'inconnue de la Haute-Marne» et « Résoudre l'équation (E) d'inconnue *x*», il s'agit bien du même mot « inconnue » mais il ne fait aucun doute que ce n'est que dans la deuxième expression qu'il est utilisé dans un sens mathématique. Comment programmer ce qui semble évident à un humain ? Les probabilités conditionnelles et les statistiques constituent une piste sérieuse à explorer. En effet, on peut chercher la probabilité que la phrase soit à caractère mathématique sachant qu'elle contient par exemple le mot « inconnue »[9]. De fait, on cherche à déterminer une probabilité conditionnelle.

Pour calculer ces probabilités conditionnelles on aura besoin de données. On peut dans un premier temps faire des suppositions que l'on pourra affiner par la suite avec des calculs statistiques.

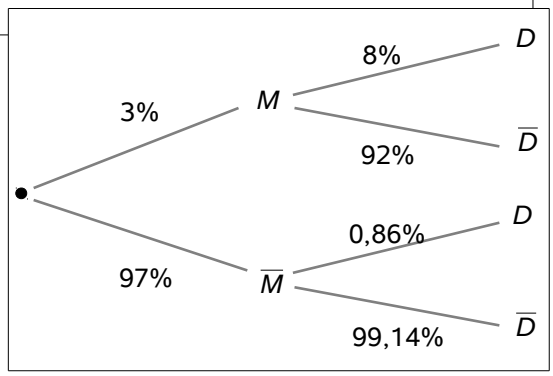
On peut commencer par choisir des fréquences a priori que l'on affinera à l'aide de statistiques issues de « corpus témoins » ; selon l'approche « fréquentiste », on peut prendre comme valeur de la probabilité d'apparition d'un mot particulier sa fréquence d'apparition parmi les phrases (ou lignes) d'un texte de référence.

Pour traiter cette partie il est nécessaire de faire le lien avec les notions vues en classe de mathématiques (arbres de probabilités correspondant à une successions de choix décisionnels).

Considérons par exemple le mot « démontrer » et une ligne quelconque du texte. On désigne :

- M l'événement : « la ligne est à caractère mathématique »
- et D l'événement : « la ligne contient le mot démontrer ».

On peut représenter la situation à l'aide d'un arbre pondéré (les probabilités ci-contre sont données à titre indicatif).



On calcule la probabilité que la ligne soit à caractère mathématique sachant qu'elle contient le mot « démontrer » à l'aide de l'arbre pondéré ; il vient $p_D(M)=\frac{p(M).p_M(D)}{p(\overline{M}).p_{\overline{M}}(D)+p(M).p_M(D)}$ soit ici environ 22%.

Pour chaque ligne on va tester la présence de « mots tabous » contenus dans une liste préalablement établie. On cumule les scores pour chaque mot « tabou » présent dans la ligne pour constituer un score global de spam :

$$Score=\sum_{mot\in ligne} p_{mot}(M).score(mot)$$

Ensuite on comparera ce score à un seuil de tolérance pour prendre une décision (la ligne traitée est du spam, ou non).

On peut dans un premier temps fournir une liste minimale de « mots tabous ».

Une remarque sur les structures de données : le développement de l'idée initiale poussera sans doute à utiliser des structures de données un peu plus adaptées que de simples listes (ou tableaux), et tout particulièrement les tableaux associatifs (également nommés tables de hachage ou dictionnaires) qui font partie de « l'attirail » fourni par la plupart des langages de programmation modernes tout en fournissant une introduction au principe des bases de données (ensembles de couples (clé,valeur)).

Le code-source commenté est fourni en annexe.

2.4 Affiner les filtres

Il est très probable que les résultats obtenus dans un premier temps soient relativement modestes voir décevants. Il est clair que la performance du filtrage probabiliste est directement lié au choix des « mots tabous », à la richesse de cette base de données, à la qualité de la statistique et au seuil de tolérance.

On pourra apporter quelques améliorations techniques en gérant les « mots tabous » dans une base de données

9 la seconde expression va se distinguer de la première par la présence du mot « équation ».

JSON (voir annexe) et en offrant à l'utilisateur d'ajouter ses propres mots.

D'autre part, il est inévitable qu'un certain nombre de faux-positifs apparaissent, cela peut être l'occasion d'une réflexion sur ce phénomène et de chercher des pistes pour le minimiser. Il faut garder à l'esprit qu'un taux de réussite de 100 % est impossible à atteindre. En ce qui concerne la détection du « spam » proprement dit, un autre phénomène apparaît, la *co-évolution* : au fur et à mesure que les détecteurs de spams se font plus efficaces, les concepteurs améliorent leur production afin de la faire ressembler au mieux à des textes « ordinaires ». Ces deux mécanismes (faux-positifs et co-évolution) méritent d'être rapprochés des thèmes abordés en SVT.

3 / Des projets possibles

- Déployer un serveur mail local avec une interface de type webmail, puis installation et configuration de Spamassassin.
- Travailler autour de l'implémentation d'une fonction d'autocomplétion.

4 / Références

4.1 Outils logiciels

Un langage de programmation simple et performant est nécessaire. Le langage Python associé à la bibliothèque RE (Regular Expressions) semble un choix pertinent, de nombreux autres langages peuvent être utilisés. Pour « jouer » avec les expressions régulières, un bon éditeur de texte est recommandé.

Un bon environnement de développement (IDE) rendra de grands services, on peut en signaler deux qui sont libres (licence GPL) et multi-plateformes :

Eric : http://eric-ide.python-projects.org et Geany : http://www.geany.org

Ne fonctionnant que dans Windows, l'éditeur de texte Notepad++ (http://notepad-plus-plus.org/fr) convient tout à fait et inclut un bon support des expressions régulières.

4.2 Documentation scientifique

- La lutte contre le spam grâce au « le filtrage bayésien » :
  http://fr.wikipedia.org/wiki/Filtrage_bayesien_du_spam
- Article sur Interstices de Gilles Richard (université Paul Sabatier de Toulouse):
  http://interstices.info/jcms/c_41867/spams-et-hams-et-comment-les-filtrer
- L'article originel de Paul Graham sur le filtrage bayésien :
  http://www.paulgraham.com/spam.html
  Cet article est toujours la référence en la matière, et sa lecture, bien qu'un peu ardue, est extrêmement enrichissante.

4.3 Documentations sur les logiciels et les expressions régulières

- Documentation du module « re » de Python :

  http://docs.python.org/library/re.html

- L'article sur les expressions régulières dans Wikipedia :

  http://fr.wikipedia.org/wiki/Expression_rationnelle

- Et un article très bien fait sur le site du Zéro :

  http://www.siteduzero.com/tutoriel-3-14608-les-expressions-regulieres-partie-1-2.html

- En anglais, le site suivant est incontournable :

  http://www.regular-expressions.info

Annexe : codes-source

1 / La mise au point d'un filtre probabiliste

```
import sys, os, re, simplejson, errno      # charger les bibliothèques
# re pour les expressions régulières, simplejson pour l'accès à un fichier structuré
try:
# ouvre le fichier à analyser et lève une exception si le fichier est absent.
  fichier = open("fichier_a_analyser.txt", "r")
  lignes = fichier.readlines()             # retourne une liste des lignes du fichier.
  fichier.close()
# On crée les deux fichiers pour stocker les résultats (bons, mauvais)
  resultat_spam = open("resultat_spam.txt", "w+")
  resultat_ham = open("resultat_ham.txt", "w+")
# On ouvre le fichier des mots tabous et on remplit un dictionnaire avec
  fichier_mots_tabous = open("mots_tabous.json")
  prb_mots_tabous= simplejson.load(fichier_mots_tabous)
  fichier_mots_tabous.close()
# On crée la liste des mots tabous à partir des clés du dictionnaire.
  mots_tabous = prb_mots_tabous.keys()
  prb_spam = 0.1                           # constantes à adapter : 10% de spam maxi
  seuil_tolerance = 2                      # et seuil de sensibilité 2.
# Pour chaque ligne du fichier on teste la présence des mots tabous et on cumule les scores pour constituer
un score global de spam
  for ligne in lignes:
      score = 0
      motpresent =""
      for mot in mots_tabous:
           motregex=mot+"_?"
           if re.search(motregex,ligne,re.IGNORECASE) :
                test = (prb_mots_tabous[mot][0]*prb_spam)/
                 (prb_mots_tabous[mot][0]*prb_spam+prb_mots_tabous[mot][1]*
                 (1-prb_spam))
                score=score+test*prb_mots_tabous[mot][2]
                motpresent = motpresent+" "+mot
      if (score>seuil_tolerance):          # Si le score de la ligne est supérieur au seuil choisi
           print ligne                     # on ajoute la ligne au fichier resultat_spam.txt
           resultat_spam.write(" "+str(score) + " ***" + motpresent
                +"***\n"+ligne)
           else :                          # Si le score est faible ...
                resultat_ham.write(ligne) # on ajoute au fichier resultat_ham.txt

  print "**** Le texte a été analysé et les résultats sauvegardés dans
  resultat_spam.txt et resultat_ham.txt ****"

  resultat_ham.close()
  resultat_spam.close()

# Gestion de l'exception
except IOError, e:
        print e
        print "Vérifiez que le fichier fichier_a_analyser.txt est bien présent."
```

2 / Un dictionnaire de mots « tabous »

On donne ci-dessous l'amorce d'un dictionnaire (destiné à détecter la présence de phrases ayant trait aux mathématiques dans un texte), structuré de la manière suivante :

"mot": [fréquence d'apparition dans un texte à caractère mathématique, fréquence d'apparition dans un texte non mathématique, score]

```
{
"absciss": [0.02,0.0086,1.5],
"affix": [0.02,0.0001,1],
"angle": [0.05,0.006,0.5],
"arrondi": [0.05,0.006,0.5],
"asympt": [0.01,0.0001,2],
"axe ": [0.05,0.006,1.5],
"barycentr": [0.02,0.0001,1.5],
"born": [0.05,0.002,0.5],
"calcul": [0.1,0.008,1],
"carré": [0.08,0.008,0.5],
"centr": [0.05,0.005,1],
"cercle": [0.1,0.005,1],
"complexe": [0.07,0.005,1.5],
"conjecture": [0.03,0.0008,2],
"converg": [0.05,0.004,1],
"coordonn": [0.05,0.0086,1],
"cos": [0.03,0.0086,1],
"courbe": [0.08,0.0086,1],
"croissant": [0.05,0.0086,0.5],
"cub": [0.03,0.0086,0.5],
"diametr": [0.02,0.0086,0.5],
"distanc": [0.08,0.0086,0.5],
"divisibl": [0.02,0.0086,1.5],
"direct":[0.05,0.0086,0.5],
"deriv":[0.08,0.0086,1.5],
"definie":[0.04,0.0086,1],
"détail":[0.05,0.0086,0.5],
"droite": [0.05,0.0086,0.5],
"égalit": [0.06,0.0086,1],
"élement": [0.05,0.0086,1],
"ensembl": [0.05,0.0086,1],
"entier":[0.04,0.0086,0.5],
"équatio": [0.04,0.0086,2],
"évènemen": [0.04,0.0086,1],
"événemen": [0.04,0.0086,0.5],
"étudier": [0.08,0.0086,2]
}
```