Apprentissage Supervisé

Filtre anti-spam

AMBROZIK Hugo
KLEINHENTZ Nicolas

Introduction

Le projet de filtre anti-spam a été réalisé a l'aide du classifieur naïf de Bayes, qui permet, après apprentissage, de déterminer si un message est un SPAM ou non. Notre classifieur obtient une erreur moyenne d'environ 10% (voir résultats d'exécution) étant composée de 20% d'erreur sur les SPAM et moins de 1% d'erreur sur les messages ne l'étant pas.
Le projet est constitué de 4 programmes rédigés en Python3 et ayant tous une fonction différente.
Pour commencer, *ApprentissageFiltre.py* servira à créer un classifieur à partir d'une base d'apprentissage et d'un nombre de messages déterminé par l'utilisateur.
Pour un classifieur existant, *ApprentissageFiltreEnLigne.py* permettra de le mettre à jour en exerçant un apprentissage supplémentaire sur un message donné par l'utilisateur et pour une catégorie choisie : SPAM ou non.
Pour tester le classifieur on pourra utiliser *TestClassifieur.py* avec un classifieur ainsi qu'une base de tests choisis par l'utilisateur, il affichera le détail de l'exécution ainsi que l'erreur de chaque catégorie et global en pourcentage.
Pour finir, le programme *FiltreAntiSpam.py* permettra de déterminer si le message choisi par l'utilisateur est un SPAM ou non.

Fonctionnement du programme

L'apprentissage ainsi que la lecture des messages seront fait à l'aide d'un dictionnaire de mots anglais contenant des mots de trois lettres ou plus. Le dictionnaire est représenté sous forme d'une liste des mots qu'il contient, alors qu'un message sera lui décrit à l'aide d'un vecteur d'occurrences des mots dans le message. Le vecteur est ordonné de la même façon que le dictionnaire et nous indiquera si oui (*True*) ou non (*False*) chaque mot est contenu dans le message.
Pour l'apprentissage, il est tout d'abord calculé un vecteur d'occurrences global des messages sous forme de chaque indice contenant le nombre d'apparences uniques du mot dans chaque message. Ce vecteur sera ensuite écrit dans un fichier comme étant le classifieur.
Dans ce classifieur sauvegardé se trouveront aussi les probabilités de $P(Y = SPAM)$ et $P(Y = HAM)$ permettant le calcul des probabilités à posteriori, la taille de la base d'apprentissage pour l'apprentissage en ligne, le epsilon permettant de lisser les paramètres, la longueur des vecteurs d'occurrences global et pour finir ces vecteurs en question.
Nous avons choisi de ne pas stocker le résultat du calcul des probabilités à posteriori dans le classifieur car celui-ci n'est pas utilisable lors de l'apprentissage en ligne, on a donc stocké les deux vecteurs d'occurrences et ainsi l'apprentissage en ligne est donc faisable à souhait, et le calcul des probabilités à posteriori est donc fait au moment du test du classifieur ou du test d'un message.

Concernant le calcul des probabilités à posteriori il a fallu chercher un moyen de transformer le calcul en gardant l'ordre des résultats des uns envers les autres, car une succession de produits de probabilités demande une trop grande précision à la machine. Nous avons aussi remplacé les puissance de x^j et $1 - x^j$ par des conditions, car cela revient à faire le produit sur b^j_{SPAM} ou b^j_{HAM}. Nous sommes donc arrivés à ce calcul des probabilités à posteriori :

$$P(X = SPAM) = ln(P(Y = SPAM)) + \sum_{j=1}^{d} ln(b^j_{SPAM})^{x^j} + ln(1 - b^j_{SPAM})^{1-x^j}$$

$$P(X = HAM) = ln(P(Y = HAM)) + \sum_{j=1}^{d} ln(b^j_{HAM})^{x^j} + ln(1 - b^j_{HAM})^{1-x^j}$$

Note : $log(x)$ est le logarithme naturel, c'est à dire e^x.
Comme recherché, l'ordre entre deux probabilités est respecté, la réponse du classifieur sera donc exacte malgré le changement de valeur des probabilités.
Maintenant, pour l'affichage de ces probabilités, il faut calculer depuis le résultat précédent son résultat en probabilité, c'est à dire compris entre 0 et 1. On les retrouvera donc de la sorte :

$$P(Y = SPAM \mid X = x) = \frac{1}{exp(P(X = SPAM)) + exp(P(X = HAM))} P(X = SPAM)$$

$$P(Y = HAM \mid X = x) = \frac{1}{exp(P(X = HAM)) + exp(P(X = HAM))} P(X = HAM)$$

Pour le test du classifieur, un parcours de la base de test fournie sur un certain nombre de message permet de calculer l'erreur en comptant le nombre de fois où le classifieur fait une erreur et diviser ce résultat par le nombre de message de test.
Enfin pour l'apprentissage en ligne, de part notre manière de sauvegarder le classifieur, il nous suffit uniquement d'ajouter le vecteur d'occurrences du nouveau message au vecteur existant pour la catégorie choisie : SPAM ou non, mettre à jour ne nombre de messages de cette catégorie et écraser l'ancien classifieur par ce nouveau.

Exécution des programmes

Pour l'apprentissage d'un nouveau filtre :

python3 ApprentissageFiltre.py mon_classifieur dossier_base_app nb_spam nb_ham
mon_classifieur : nom du classifieur à créer
dossier_base_app : dossier contenant la base d'apprentissage
nb_spam : nombre de message SPAM sur lesquels apprendre
nb_ham : nombre de message HAM sur lesquels apprendre

Pour l'apprentissage en ligne sur un classifieur existant :

python3 ApprentissageFiltreEnLigne.py mon_classifieur.cla message.txt SPAM|HAM
mon_classifieur.cla : nom du classifieur à mettre à jour
message.txt : fichier texte représentant le message à utiliser pour la mise à jour
SPAM|HAM : choix de la catégorie d'apprentissage, choisir entre SPAM et HAM

Pour le test du classifieur :

python3 TestClassifieur.py mon_classifieur dossier_base_test nb_spam nb_ham
mon_classifieur.cla : nom du classifieur à tester
dossier_base_test : dossier contenant la base de test
nb_spam : nombre de messages SPAM sur lesquels faire le test
nb_ham : nombre de messages HAM sur lesquels faire le test

Pour exécuter le classifieur sur un message :

python3 FiltreAntiSpam.py mon_classifieur.cla message.txt
mon_classifieur.cla : nom de classifieur à utiliser
message.txt : message à tester

Notes :
Les classifieurs générés et sauvegardés par nos programme sont d'extension *.cla* par soucis de compréhension des fichiers.
Le dictionnaire est supposé être dans le dossier courant des programmes et nommé « *dictionnaire1000en.txt* ».
Quand les extensions de fichiers sont précisés ci-dessus, il faudra les ajouter lors de l'exécution.

Librairies utilisées

Aucune librairie tierce à Python3 n'a été utilisée. La librairie *sys* permet de récupérer les arguments de l'exécution d'un programme. Pour la lecture des messages nous avons utilisé la librairie *re* permettant de modifier des chaînes de caractères grâce à des expressions régulières, notamment pour retirer les caractères spéciaux des messages. Enfin la librairie *math* pour les opérations telles que *log( )* pour le logarithme népérien et *exp( )* pour l'exponentiel.

Exemples d'exécution

Création d'un classifieur :

```
> python3 ApprentissageFiltre.py classifieur baseapp 500 2500
Apprentissage sur 500 SPAM et 2500 HAM...
Classifieur enregistré dans ‘classifieur.cla'.
```

Mise à jour de ce dernier avec un nouveau message :

```
> python3 ApprentissageFiltreEnLigne.py classifieur.cla baseapp/ham/600.txt HAM
Modification du filtre 'classifieur.cla' par apprentissage sur le HAM 'baseapp/
ham/600.txt'...
Classifieur 'classifieur.cla' mis à jour.
```

Test de ce classifieur :

```
> python3 TestClassifieur.py classifieur.cla basetest 500 500
Test du classifieur sur 500 SPAM et 500 HAM...
[…]*
Erreur de test sur les 500 SPAM           : 22.0 %
Erreur de test sur les 500 HAM            : 0.8 %
Erreur de test global sur 1000 mails      : 11.4 %
```

Utilisation du classifieur pour filtrer un message :

```
> python3 FiltreAntiSpam.py classifieur.cla basetest/spam/42.txt
Test du message basetest/spam/42.txt...
basetest/spam/42.txt : P(Y = SPAM | X = x) = 0.9864237108569618 , P(Y = HAM | X
= x) = 0.013576289143038114
D'après classifieur.cla, le message basetest/spam/42.txt est un SPAM !

> python3 FiltreAntiSpam.py classifieur.cla basetest/ham/128.txt
Test du message basetest/ham/128.txt...
basetest/ham/128.txt : P(Y = SPAM | X = x) = 2.0219495942120664e-08 , P(Y = HAM
| X = x) = 0.9999999797805039
D'après classifieur.cla, le message basetest/ham/128.txt est un HAM !
```

*[…] : ici se trouveront le détail du test de chaque message de la forme :

```
HAM numéro 308  : P(Y = SPAM | X = x) = 9.619877328126027e-13 , P(Y = HAM | X =
x) = 0.999999999999038
      => identifié comme un HAM
```

Difficultés rencontrées

Le plus long a été la bonne compréhension du classifieur naïf de Bayes, une fois compris la traduction en algorithme est plutôt rapide.
La principale difficulté du sujet a été la précision des calculs de probabilité à priori, qui nous a pris plus de temps que prévu. En essayant de passer par des librairie comme NumPy pour utiliser leur type *float64* ou encore la classe *BigDecimal* de Java. Finalement une fois la solution trouvée, la mettre en place n'a pas été très compliqué, et permet au projet de ne nécessiter aucune librairie autre que celle fournie avec Python3.
Pour le reste, analyser et traiter des fichiers textes est une activité qu'on réalise souvent, la traduction des formules est aisément faisable grâce aux boucles pour les sommes et la librairie math pour certaines opérations.

Contribution des membres

AMBROZIK Hugo : Analyse et traitement des fichiers texte, test du classifieur, travail en commun sur le calcul des probabilités à priori, amélioration en ligne d'un classifieur existant.

KLEINHENTZ Nicolas : Apprentissage du classifieur, travail en commun sur le calcul des probabilités à priori, enregistrement du classifieur et lecture de celui ci, test d'un message unique.