Fouille de données – TD 4 (continuation du TD3)

Ce TD est la continuation du TD 3. Il est a rendre pour **ce soir 23h59**

Tout d'abord, reprenez le td3 ou vous en étiez, et continuez-le, mais en renommant le fichier td4.py.
La première partie de ce TD consiste donc en l'énoncé du TD3.

Vous pouvez télécharger la nouvelle archive test4.tar.xz, qui commence donc par vérifier les questions de l'énoncé du TD3 -- Encore une fois: commencez par finir les Exercices 1 à 7!

Quand vous avez fini les Exercices 1, 2, 3, 4, 5, 6, 7, passez aux questions ci-dessous:

Exercice 8: Évolution, Tuning

On va continuer à améliorer la fonction de classification, que vous allez faire évoluer, toujours dans votre fichier td4.py:

```
def classify_spam(sms):
  """Returns True is the message 'sms' is predicted to be a spam."""
  ...
```

On va maintenant essayer d'**améliorer la qualité** de prédiction!
On va se concentrer sur 2 fonctions:

```
def classify_spam_precision(sms):
  """Like classify_spam(), but guaranteed to have precision > 0.9."""
  ...

def classify_spam_recall(sms):
  """Like classify_spam(), but guaranteed to have recall > 0.9."""
  ...
```

Le but sera que:

- `classify_spam_precision()` ait une précision au moins égale à 0.9 (sur l'ensemble de test); et ait le meilleur recall possible.
- `classify_spam_recall()` ait un recall au moins égal à 0.9 (sur l'ensemble de test); et ait la meilleure precision possible.

Au fur et à mesure des améliorations ci-dessous, vous devriez être capables d'obtenir des valeurs de mieux en mieux pour recall et précision. Matérialisez-les dans ces 2 fonctions.

Votre note sera basé sur la performance de ces 2 fonctions **sur un autre ensemble de test**, caché : **attention à l'over-fitting!**

ATTENTION: Ne touchez plus à vos fonctions des exercices précédents, pour que les tests continuent à passer. De nombreuses suggestions ci-dessous demandent pourtant de les améliorer ou les modifier! Je vous conseille alors de copier ces fonctions en les renommant, avant de modifier la nouvelle copie comme vous le souhaitez, et de les utiliser dans vos 2 fonctions `classify_spam_X()`.

AUTRE CHOSE: une fois n'est pas coutume, votre fichier pourra exécuter des instructions d'initialisation, notamment en lisant un fichier 'SMSSpamCollection'. Cette initialisation (qui a lieu au moment du "import td4" dans test4.py) **ne devra pas prendre trop de temps** -- évitez de prendre plus de 5 secondes.

DERNIERE CHOSE: le score affiché par "python test4.py" n'est pas indicatif de votre score réel sur ce TD, car dans la dernière question (Exo 8) je vous noterai en fonction de la qualité des estimateurs, alors que le test ne vérifie que la syntaxe sur des exemple basiques.
-> ne pensez pas que "SCORE=100" veut dire "impossible de faire mieux", c'est tout à fait faux!

Voilà une liste d'idées à essayer. N'hésitez pas à en essayer d'autres! Tout est valable, a la fin le but est juste d'obtenir les meilleurs estimateurs possibles (donc le meilleur recall pour `classify_spam_precision` et la meilleure précision pour `classify_spam_recall`) :

1. Essayez avec d'autres valeurs pour la probabilité "seuil": au lieu de 0.5, essayez plus.. Essayez moins… Voyez comme le recall / precision changent!

2. Comme mentionné en cours, on a seulement estimé *P(être un spam)*. Or, si on estime la proba inverse: *P(ne pas être un spam)*, avec la même méthode, on ne trouvera pas le complément!
   Par exemple on pourrait obtenir proba_spam = 0.4 et proba_pas_spam = 0.35, ce qui semble étrange. Ou proba_spam = 87.3 et proba_pas_spam = 3.5.
   C'est dû à l'hypothèse d'indépendance, fausse, qui fait qu'on a des valeurs approchées.
   Essayez de changer votre fonction de classification pour, par exemple, implémenter un estimateur de P(pas spam): il suffira simplement de faire une 2eme version de `naive_bayes_train` qui est similaire mais qui se concentre sur les "non spam" (aka "ham").
   Ensuite, votre estimateur pourra utiliser les 2 estimateurs de probabilités, en les comparant par exemple.

3. Les formules vues ci-dessus peuvent être grandement améliorées, car elles ignorent les mots qui ne sont **pas** dans le SMS. Or, c'est une information importante! Les formules sont un peu modifiées, **cf cours.** Si vous changez vos formules pour prendre également en compte l'*absence* des mots (tout en restant en complexité O(taille du SMS) pour l'évaluation d'un SMS, et non pas O(taille du dictionnaire)), votre estimateur sera sans doute (bien) meilleur.

4. On peut grandement tuner la fonction de classification, qui est binaire, pour l'améliorer, notamment en **visualisant** le comportement de p_{spam} et p_{ham} sur un graphe. Si vous tracez tous les points en 2D (abscisse = p_{ham} et ordonnée = p_{spam} par exemple, couleur des points = ham ou spam), vous verrez qu'on peut faire beaucoup mieux que le simple prédicat "$p_{spam} > p_{ham}$".

5. On peut peut-être gagner quelque chose en se focalisant sur les mots les plus importants. Essayez de réduire le dictionnaire de mots `words` pour ne garder que les mots les plus importants.
   Essayez diverses idées! Attention à bien incrorporer vos modifications dans votre rendu: le dictionnaire de mots global utilisé par votre code doit bien incorporer vos modifs, pour que votre fonction `classify_spam()` les prennent en compte.
   Diverses idées peuvent marcher:
   a. Eviter les mots trop rares
   b. Eviter les mots trop universels
   c. Assembler les mots qui se ressemblent, et autres traitements de texte (Typos? Majuscules? Ponctuation?)