IFT 615 : Devoir 4
Travail individuel

Remise : 2 août 2012, 11h35 (**au plus tard**)

Ce devoir comporte 2 questions de programmation. Vous trouverez tous les fichiers nécessaires pour ce devoir ici : `http://www.dmi.usherb.ca/~larocheh/cours/ift615_E2012/devoir_4/devoir_4.zip`.

Veuillez soumettre vos solutions à l'aide de l'outil **turnin** :

```
turnin -c ift615 -p devoir_4 solution_reseau_neurones.py solution_pourriels.py
```

1. [**5 points**] Implémentez une classe Python `ReseauDeNeurones` correspondant à un réseau de neurones à une couche cachée, entraîné à l'aide de l'algorithme de rétropropagation des gradients, pour un problème de classification à 2 classes. Le problème correspond à de la reconnaissance de caractères, où l'on doit distinguer les caractères "e" ($y = 0$) des "o" ($y = 1$).

   Pour ce faire, vous devez implémenter une classe `ReseauDeNeurones` ayant les méthodes suivantes :

   - `__init__(self, alpha, T)` : Le constructeur ayant comme arguments le taux d'apprentissage `alpha` et le nombre d'itérations `T` à utiliser pour l'entraînement. Cette méthode est déjà implémentée.
   - `initialisation(self, W, w)` : Initialise la matrice de poids $\mathbf{W}$ entre la couche d'entrée et la couche cachée, puis le vector de connexions $\mathbf{w}$ entre la couche cachée et le neurone de sortie. `W` est donc un tableau Numpy à deux dimensions (matrice) et `w` est un tableau Numpy à une dimension (vector). Plus spécifiquement, la valeur de la connexion entre le i^{e} neurone caché et la j^{e} entrée x_j correspond à `W[i,j]`. De plus, la connexion entre le i^{e} neurone caché et le neurone de sortie correspond à `w[i]`. Cette méthode est déjà implémentée.
   - `parametres(self)` : Retourne la paire `(W,w)` de la matrice de connexions $\mathbf{W}$ (c'est-à-dire une matrice Numpy `W`) et du vector de connexions $\mathbf{w}$ (c'est-à-dire un vector Numpy `w`) du réseau de neurones. Cette méthode est déjà implémentée.
   - `prediction(self, x)` : Retourne la prédiction par le réseau de neurones de la classe d'une entrée, représentée par un vecteur Numpy `x`. Cette prédiction doit donc être 0 ou 1.
   - `mise_a_jour(self, x, y)` : Met à jour les paramètres du réseau de neurones à l'aide de sa règle d'apprentissage, à partir d'une entrée `x` (vecteur Numpy) et de sa classe cible `y` (0 ou 1).
   - `entrainement(self, X, Y)` : Entraîne le réseau de neurones durant T itérations sur l'ensemble d'entraînement formé des entrées `X` (une matrice Numpy, où la t^{e} rangée correspond à l'entrée $\mathbf{x}_t$) et des classes cibles `Y` (un vecteur Numpy où le t^{e} élément correspond à la cible y_t). Il est recommandé d'appeler votre méthode `mise_a_jour(self, x, y)` à l'intérieur de `entrainement(self, X, Y)`.

   Votre implémentation de cette classe doit être placée dans le fichier `solution_reseau_neurones.py`, qui contient déjà une ébauche de la classe. Ce fichier sera importé lors de la procédure automatique de correction. Le fichier `reseau_neurones.py` contient un exemple d'exécution de votre code. Vous pouvez l'appeler comme un script. Ce script nécessite également que les fichiers `train_reseau_neurones.pkl`, `test_reseau_neurones.pkl` et `parametres_attendus_reseau_neurones.pkl` soient présents dans le même répertoire. Une implémentation correcte obtiendra une erreur d'entraînement de 0% et une erreur de test de 0%. `reseau_neurones.py` compare également les valeurs de paramètres trouvées par votre implémentation et celles trouvées par une implémentation correcte.

2. **[5 points]** Implémentez en Python un détecteur de pourriels (*spam*) basé sur un modèle bayésien naïf multinomial. Chaque courriel (document) est représenté sous la forme d'une longue chaîne de charactère (`str`) dans laquelle chaque mot est séparé par un caractère espace `' '`[1]. Associé à chaque document est un indice de classe indiquant si le courriel est un pourriel (classe 0) ou est un bon courriel (classe 1).

   Pour ce faire, vous devez en premier lieu implémenter une classe `Probabilite` permettant de modéliser les distributions $P(C)$ et $P(W|C)$ nécessaires dans un modèle bayésien naïf multinomial. Les objets de cette classe ont comme vairables membres :

   - `nbMotsParClasse` : Dictionnaire[2] (`dict`) contenant le nombre de mots total dans les documents de chaque catégorie. Par exemple, `nbMotsParClasse[0]` doit être le nombre de mots total dans tous les pourriels du corpus d'entraînement.
   - `nbDocsParClasse` : Dictionnaire (`dict`) contenant le nombre de documents de chaque catégorie. Par exemple, `nbDocsParClasse[1]` doit être le nombre de courriels qui ne sont pas des pourriels dans le corpus d'entraînement.
   - `freqWC` : Dictionnaire (`dict`) contenant le nombre de fois que chaque mot apparaît dans chaque catégorie de documents. Par exemple, `freqWC[('allo',0)]` doit être le nombre de fois que le mot `'allo'` apparaît dans les pourriels du corpus d'entraînement.
   - `vocabulaire` : Un (`set` de `str`) déterminant le vocabulaire à utiliser.

   À partir de ces champs membres, vous devez implémenter les méthodes de la classe `Probabilite` suivantes :

   - `probClasse(self, C)` : Calcule la probabilité a priori d'une classe. Par example, soit `P` un object de classe `Probabilite`, `P.probClasses(C=0)` retournera l'estimation de $P(C = 0)$ (i.e. la probabilité a priori qu'un courriel soit un pourriel).
   - `probMotEtantDonneClasse(self, C, W, delta)` : Calcule la probabilité conditionnelle d'un mot sachant la classe d'un document. La probabilité est lissée en utilisant la constante de lissage `delta`. Par example, `P.probMotEtantDonneClasse(W='allo', C=0, delta=1)` estimera la probabilité $P(W = \text{'allo'}|C = 0)$ (i.e. la probabilité du mot `'allo'` en supposant que le courriel est un pourriel) à l'aide d'un lissage utilisant $\delta = 1$.

   La classe `Probabilite` implémente déjà des raccourcis[3] pratiques pour l'appel de ces deux méthodes. Effectivement, l'instruction `P(C=0)` est équivalente à `P.probClasses(C=0)` et l'instruction `P(W='allo',C=0,delta=1)` est équivalente à `P.probMotEtantDonneClasse(W='allo', C=0, delta=1)` (à noter que `P(W='allo',C=0,delta=1)` retourne la probabilité $P(W = \text{'allo'}|C = 0)$ et non pas $P(W = \text{'allo'}, C = 0)$).

   À l'aide de la classe `Probabilite`, vous devez ensuite implémenter les fonctions suivantes :

   - `creerVocabulaire(documents, seuil)` : Fonction qui retourne le vocabulaire à utiliser sous la forme d'une liste (`list`) de mots (`str`). Pour créer le vocabulaire, vous devez retenir seulement les mots ayant une fréquence plus grande ou égale à un certain seuil. Pour calculer ces fréquences, vous devez utiliser `documents`, qui est une liste de `str`. Chaque `str` correspond à un document dans lequel chaque mot est séparé par un espace (`' '`). `seuil` est un entier et correspond au seuil de fréquence à utiliser.
   - `pretraiter(doc, V)` : Fonction qui remplace tous les mots du document `doc` qui ne font pas partie du vocabulaire `V` par le mot `'OOV'`. La fonction doit retourner la liste (`list`) des mots (`str`) contenus dans le document `doc` et prétraités par le vocabulaire.
   - `entrainer(corpus, P)` : Fonction permettant d'entraîner (estimer) les distributions $P(C)$ et $P(W|C)$ à partir d'un corpus d'entraînement. Cette fonction doit donc remplir les dictionnaires `nbMotsParClasse`, `nbDocsParClasse`, `freqWC` de l'objet `P` donné en argument. La fonction ne retourne rien, puisque l'objet `P` est modifié *in-place*. Le corpus d'entraînement (`corpus`) est représenté sous la forme d'une

[1] Voir la documentation de Python pour plus d'information sur la manipulation des chaînes de caractères : `http://docs.python.org/library/string.html`

[2] En fait, les dictionnaires de la classe `Probabilite` sont des `defaultdict`, qui retourne une valeur par défaut (0 dans notre cas) pour les clés qui ne sont pas présentes dans le dictionnaire. Pour en savoir plus : `http://docs.python.org/library/collections.html#collections.defaultdict`.

[3] Ces raccourcis sont implémentés par la méthode `__call__(self, C, W=None, delta=None)` de la classe `Probabilite`

liste de tuples, où chaque tuple est composé d'une liste de mots (document prétraité) et d'un entier indiquant la classe (0 pour pourriel, 1 sinon). Par exemple, `corpus == [..., (["Mon", "courriel", "..."], 1), ...]`.

- `predire(doc, P, C, delta)` : Fonction retournant la classe (0 pour pourriel, 1 pour bon courriel) la plus probable selon le modèle bayésien naïf multinomial, ainsi que le log de la probabilité conjointe de cette classe et des mots du document (log en base naturelle). `doc` est un document prétraité, `P` est l'objet `Probabilite` à utiliser, `C` est la liste des classes possibles (`[0,1]` pour ce problème) et `delta` est la constante de lissage à utiliser. La fonction doit donc retourner une paire `(int,float)` où l'entier désigne la classe la plus probable c et le nombre à virgule est la probabilité conjointe $P(C = c, D = [w_1, \ldots, w_d])$.

Un fichier initial `solution_pourriels.py` vous est fourni, contenant les signatures de la classe `Probabilite` et des fonctions `creerVocabulaire()`, `pretraiter()`, `entrainer()` et `predire()`. Le script Python `pourriels.py` importera votre fichier et l'utilisera afin de calculer les erreurs de classification sur le corpus d'entraînement et de test. Une implémentation correcte devrait obtenir une erreur de 1.2464% et de 7.7533% sur les corpus d'entraînement et de test respectivement. Le script testera également l'exécution individuelle de vos fonctions. Assurez-vous que les fichiers `pourriels_solution_attendue.pkl`, `train_pourriels.pkl` et `test_pourriels.pkl` soient présents dans le même répertoire que le script lors de son exécution.