Portail IE / INF2 2 mai 2019
Contrôle continu TP 30 mn

Ce sujet comporte 2 pages. Répondre de façon concise mais précise aux questions suivantes en utilisant les cadres. Aucun document n'est autorisé, hormis un aide-mémoire personnel, manuscrit, de format A4, recto seul. Remplir le cadre « Identification » dès le début de l'épreuve.

1. **Rappeler l'expression mathématique de l'information portée par une lettre d'un alphabet ?**

$$h(s) = log_2\ 1/p(s)$$

où $p(s)$ est la probabilité d'occurrence de la lettre s .

Attention : vous êtes tous censés avoir implémenté une variante de cette formule dans votre TP.

Attention : Beaucoup trop d'entre vous ont confondu information et entropie. Pourtant, les deux ne s'appliquent pas du tout au même type d'objet ; information s'applique aux lettres, et entropie aux alphabets.

2. **Cette définition fait référence à une probabilité d'occurrence *a priori*. Qu'est-ce qui en tient lieu dans votre TP ?**

Dans le TP la probabilité d'occurrence a priori est estimée en comptant les nombres d'occurrences des différentes lettres sur un échantillon.

En conséquence, le $p(s)$ de la formule qui précède sera lui-même calculé à partir des mesures de fréquence. Plus précisément, $p(s)$ est approchée par $f(s)/total$ ou $f(s)$ est le nombre d'occurrences de s et $total$ est le nombre de lettres de l'échantillon.

Attention : la définition de l'information c'est $log_2\ 1/p(s)$ donc $log_2\ total/f(s)$ et pas $log_2\ f(s)/total$.

Attention : « occurrence » avec 2 'r' ! L'écrire autrement alors que le mot figure dans la question est inexcusable.

3. **Donner l'expression en Python de l'information portée par une lettre dans le cadre de votre TP.**

En supposant la distribution des fréquences disponible dans un dictionnaire `d`, on peut imaginer une expression dans le genre suivant :

```
total = totalFreq(d)
…
h[s] = math.log(total/d[s],2)
```

où `h` est le dictionnaire qui stocke les quantités d'information de chaque lettre.

Beaucoup d'autres variantes sont possibles, du moment qu'elles sont correctes.

4. À quoi sert la structure de dictionnaire de dictionnaires ?

Elle a été introduite pour représenter les n-grammes.

Attention : Expliquer en quoi consiste un dictionnaire de dictionnaires ne répond pas du tout à la question « À quoi sert … ».

5. Quels différents cas doit prévoir la fonction `IncFreq_pfx` ?

La fonction `IncFreq_pfx` incrémente le nombre d'occurrences d'une lettre derrière un préfixe donné dans un dictionnaire de dictionnaires. On doit donc prévoir les 3 cas suivants :

- Le préfixe existe et la lettre aussi ;
- Le préfixe existe mais pas la lettre ;
- Le préfixe n'existe pas dans le dictionnaire.

Attention : imaginer le cas où la lettre existe mais pas le dictionnaire est absurde ! Où serait rangée cette lettre ?

On souhaite développer une nouvelle fonction `string2ngram_entropy` qui calcule l'entropie de l'alphabet constitué des n-grammes présents dans un texte source. Elle devra répondre à la spécification suivante :
`String2ngram_entropy` prend deux paramètres qui sont le texte source et la taille des n-grammes (le n), et elle retourne l'entropie de l'alphabet des n-grammes.

```
;; Computes n-gram entropy
;; string2ngram_entropy : string * int -> float
```

Dans la suite on ne demande pas des programmes détaillés en Python, juste leurs grandes lignes en utilisant le plus possible les fonctions du TP Shannon.

6. On suppose disponible un dictionnaire qui associe à chaque n-gramme le nombre de ces apparitions dans le texte : `(dico ngram int)`. Dire ce qu'il reste à faire.

Avec l'hypothèse fournie, on dispose d'un dictionnaire simple comme dans la première partie du TP. Il suffit donc de reprendre les traitements qui sont faits à la suite de `string2distribution` pour réaliser `string2entropy` ; c-à-d. calculer les quantités d'information portées par les différents n-grammes, puis en faire la moyenne pondérée.

Attention : on ne peut pas utiliser `string2ngram_entropy` puisque c'est justement la fonction qu'on veut réaliser.

7. Comment le faire ?

On peut suivre la démarche de la première partie du TP :

- Développer `string2ngram_information` sur le modèle de `string2information` ;
- Puis développer `string2ngram_entropy` sur le modèle de `string2entropy`.

Se rappeler qu'on a besoin de l'information et de la fréquence pour calculer l'entropie. Il serait inopérant de partir directement sur `string2information_pfx`.

Attention : « l'entropie de chaque n-gramme » ça n'existe pas. L'entropie c'est une propriété d'un alphabet. C'est comme si on écrivait « La taille moyenne de chaque étudiant ».

8. Proposez une façon de construire le dictionnaire des nombres d'apparitions de ces n-grammes en utilisant le plus possible les fonctions existantes.

La fonction `string2distribution_pfx` fournit un dictionnaire de dictionnaires. Il suffit de lui faire suivre un traitement qui met à plat le dictionnaire de dictionnaires en un dictionnaire simple. Par exemple,

```
d_ngram = {}
for pfx in d_pfx:
   for s in d_pfx[pfx]:
      d_ngram[pfx+s] = d_pfx[pfx][s]
```

où `d_pfx` est la distribution sous-forme de dictionnaire de dictionnaire, et `d_ngram` est la distribution sous forme de dictionnaire à un niveau (de type `(dico ngram int)`).

Certains ont eu l'idée de calculer le dictionnaire de dictionnaires pour les (n+1)-grammes, et ensuite de cumuler les fréquences des lettres qui suivent :

```
d_ngram = {}
for pfx in d_pfx:
   d_ngram[pfx] = 0
   for s in d_pfx[pfx]:
      d_ngram[pfx] += d_pfx[pfx][s]
```

Pourquoi pas ?

On pouvait aussi projeter d'utiliser `IncFreq` et `AnalFreq`, mais à condition de prévoir de les modifier pour qu'elles traitent des n-grammes plutôt que des lettres.

Par contre, il n'y a rien à espérer de `CumulDistrib`. Cette fonction n'avait d'utilité que pour engendrer des textes aléatoires.

9. Dans l'optique de tester la fonction `string2ngram_entropy`, donner des propriétés attendues de cette fonction.

Un test doit toujours prévoir le résultat attendu, sinon on ne pourrait pas savoir si le test réussit ou échoue. Cette fonction calcule une entropie. Un bon début pour le test est donc de construire des cas de test pour lesquels il est facile de prévoir le résultat. Par exemple, avec des n-grammes de taille 2 :

- "aba" : 2 n-grammes de même fréquence, "ab" et "ba", donc information de 1 bit pour chacun, et entropie de 1 pour l'ensemble des n-grammes ;
- "abababa" : pareil, et ainsi de suite ;
- "abcda' : 4 n-grammes de même fréquence, "ab", "bc", "cd" et "da", donc information de 2 bits pour chacun, et entropie de 2 pour l'ensemble ;
- "abcdabcda" : pareil, et ainsi de suite ;
- "aaaab" : 3 occurrences de "aa" et une de "ab", c'est le cas de la distribution ¼ - ¾ vue en cours ;
- etc.

On peut aussi faire appel à des propriétés plus générales, comme « l'entropie est maximale quand la distribution est uniforme » ou « l'entropie décroît quand les inégalités de distribution croissent », mais ça ne donne pas directement des cas de test.

En tout cas, on ne pas juste dire qu'il faut vérifier que le résultat est l'entropie attendue, car justement c'est ce qu'on veut calculer et qu'on ne sait pas la calculer de tête sauf en quelques cas particuliers. Il faut donc au moins tester ces cas particuliers.

Attention : si un texte est constitué d'une seule sorte de n-grammes, ex "aaaaaaaa", la probabilité d'occurrence de ce n-gramme sera de 1, donc son information sera 0, donc l'entropie 0 aussi. Ça peut aussi constituer un cas de test, mais seulement si on sait ce que ça doit donner. Beaucoup qui ont essayé cette voie ont cru que dans ce cas l'entropie serait 1.