Université de Strasbourg

Intelligence artificielle

Licence 3 - UFR Mathématique - Informatique

TP : Théorie de l'information

Langage de programmation à utiliser : Python (documentation : `https://www.python.org/doc/`). Un ensemble de ressources est listé sur moodle.

1 Entropie d'un texte

On rappelle que le calcul de l'entropie dans un système d'information est :

$$H(f) = -\sum_{v=1}^{n} p(v)\ log_2\ p(v) \tag{1}$$

où v est un octet du fichier lu et $p(v)$ est sa probabilité d'apparaître dans le fichier.

1. Écrire une fonction permettant de calculer et stocker les occurences de chaque caractère d'un texte (passé en paramètre) (vous pouvez par exemple utiliser la méthode Python `count()`).
   Votre fonction devra retourner un dictionnaire associant à chaque caractère du texte son occurence dans le texte (l'alphabet utilisé sera constitué uniquement des symboles du texte)
2. Écrire une méthode calculant l'entropie d'un texte (passé en paramètre). Testez votre fonction avec la phrase "si ton tonton tond mon tonton". Quelle valeur d'entropie obtenez vous ?

2 Codage de Huffman

1. Proposer une classe `Noeud` représantant un noeud dans un arbre binaire. Un nœud est caractérisé par une valeur, un nœud fils gauche et un nœud fils droit. Si c'est une feuille, alors les deux nœuds enfants valent `None`.
2. Écrire une méthode pour calculer un arbre optimal pour coder une chaîne de caractères donnée. Cet arbre se construit en partant de ses feuilles (les symboles du texte et leurs occurences). Ces feuilles sont obtenues par le dictionnaire retourné avec la fonction `occurences` de l'exercice 1. La procédure est la suivante:

```
Avec la structure de noeud définie comme :
Noeud:
|__ Donnée # couple (symbole, valeur)
|__ Branche gauche
|__ Branche droite
Constructeur: Noeud(donnee, noeud gauche, noeud droit)

----- Procédure :
liste_noeuds <- éléments du dictionnaire
trier liste_noeuds
Tant que liste_noeuds a plus de 1 élément faire:
  n1 <- liste_noeuds.pop(0) # retourne le premier elt et le supprime de la liste
  n2 <- liste_noeuds.pop(0)
  n  <- Noeud(n1.donnee[1] + n2.donnee[1], n1, n2)
  ajouter n à liste_noeuds
```

```
  trier liste_noeuds
ftq
retourner liste_noeuds # ne contient que la racine de l'arbre
```

   (a) Tester cette méthode avec les phrases "si ton tonton tond mon tonton" et "un chasseur sachant chasser".

   (b) Pour chacun des deux tests, quel est le nombre moyen de bits nécessaire pour coder chaque caractère ? Que peut-on dire de ce nombre par rapport à l'entropie du texte ?

3. Utiliser l'arbre ainsi créé dans une méthode pour écrire une version compressée du texte.
   Pour se faire, une idée est de construire ce qu'on peut appeler une "table de symboles", qui sera un dictionnaire dans lequel chaque symbole de l'arbre sera associé à son code (nécessite donc un seul parcours –exhaustif– de l'arbre, pour la construction du dictionnaire. Ensuite, l'encodage se fera uniquement à partir du dictionnaire). En parcourant l'arbre, pour construire le dictionnaire, vous encoderez `0` lorsque vous irez à gauche et `1` lorsque vous irez à droite. Ensuite, il reste à utiliser ce dictionnaire pour récupérer le code des symboles constituant le texte.

4. Toujours grâce à l'arbre optimal, écrire une méthode pour décompresser un code de Huffman.
   Dans ce cas, il s'agit de parcourir le code du texte pour déterminer le symbole dans l'arbre. Si le bit courant du code est un `0`, alors vous allez à gauche dans l'arbre; sinon à droite. Lorsque vous tombez sur une feuille, vous avez trouvé votre symbole !