TP « Communications Numériques ». Codage source. v1.3-py

N. Lebedev

1 Objectifs de ce TP

- **Étudier la compression de la source par un code de Huffman à partir d'un message texte.**
- **Simuler la transmission des mots de code à travers un canal à erreurs.**
- **Détecter le signal transmis.**
- **Mettre en évidence les erreurs**

2 Consignes et outils

Python, PyCharm en mode venv, libraries numpy, duhaffman, scikit-dsp-comm, scikit-commpy :
`>>> pip install dahuffman scikit-dsp-comm scikit-commpy`

(En option : Matlab® / Communications Toolbox)
Sources complémentaires à étudier pour l'exemple de réalisation :
`https://fr.mathworks.com/help/comm/source-coding.html`

3 Etapes de réalisation

1. On voudrait pouvoir générer un message d'information (mot, phrase, texte,...) soit dans un premier temps au plus simple codé en dur dans le fichier, sous forme de chaine de caractères, soit à partir de l'entrée standard clavier—moins intéressant car manuel, mais surtout,dans la version finale de votre programme, à partir du fichier texte,—livre libre de droits (GUTenberg), fourni ou choisi librement. Nom de variable phrase `'phrase'`
2. A partir du message généré, construire un alphabet, calculer la fréquence relative que l'on supposera proche de la probabilité d'apparition de chaque symbole (caractère). Tracez l'histogramme de cet alphabet en bar-plot.
3. Calculer l'entropie de cet alphabet.

Question	Recherchez et comparez le résultat obtenu avec les résultats théoriques et expérimentaux trouvés dans la littérature scientifique et/ou projets sur `https://stackoverflow.com` ou `https://github.com` . Un bon point de départ est `https://cs.stanford.edu/people/eroberts/courses/soco/projects/1999-00/information-theory/entropy_of_english_9.html` Concluez sur votre méthode.

4. Appliquer un algorithme de codage de source Huffman.
   Les fonctions utiles sont :
   — `from_data(phrase)` construit et retourne l'objet de la classe HuffmanCoder
   — `print_code_table()` , méthode de la classe HuffmanCoder permettant d'afficher le tableau d'encodage. Analysez-le.
   — `get_code_table()` , méthode de la classe HuffmanCoder qui retourne le dictionnaire qui est une type 'dictionnaire' Python avec les paires $"key" : value$
5. Encodez la source par un code Huffman avec `encode(phrase)`
6. Décodez le code Huffman—mettre en correspondance les symboles, revenir à la séquence émise par la source.
7. Effectuez les deux mêmes opérations en passant par le format binaire `'np.array()'` pour préparer le codage canal.
8. Convertissez le vecteur binaire des données issues de compression source Huffman en matrice de (k) colonnes, où (k) correspondra à la taille des messages en entrée du CCE—Code détecteur et Correcteur des Erreur, propres à chaque binôme pour la séance TPn2. Pour le moment vous pouvez prendre $k = 4$, mais votre programme doit être suffisamment générique.
   Pensez à gérer les *bits de bourrage*. En effet, la longueur de la séquence codée $Nbits\_tot$ peut ne pas être multiple de (k), par conséquents, les bits de bourrage à "0" doivent être insérés avant le codage et retirés avant la reconstruction du messsage par le décodeur Huffman.
   Soyez attentifs à la notion de bits de poids fort / faible (MSB/LSB) à gauche ou à droite, mais consistant dans votre programme.
9. Rajouter 1 erreur par ligne (de k bits). Cela consiste à alterner, de manière aléatoire selon la loi uniforme, un des bits par mot-message de taille (k). Revenez au format vecteur, donc opération inverse du point précédent. NB : n'oubliez pas la suppression des bits de bourrage.
10. Restituer le message initial à partir du message binaire reçu par décodage Huffman. Comparez les cas avec et sans erreurs.

4 Avancé (en option)

— Appliquer l'algorithme de compression LZ au même message généré par la source, comparer le taux de compression avec Huffman en utilisant les séquences de source de plus en plus longues tirées à partir du même alphabet.
— Etudiez, implémentez et comparez les deux codes précédents au codage algébrique (réf : livre ITILA de D. MacKay, et ses cours vidéos en ligne). Un bon point de départ serait de prendre en main, étudier, et vous en inspirer dans l'implémentation est l'outil DASHER (voir la référence dans le poly), opensource, disponible en application Linux, Android.