INF580: Mathematical Programming
Projet : Générateur de poésie

Matthieu Lequesne & Quentin Lisack 24 février 2016

Table des matières

Résumé

Natural Language Processing ... blah

1 Objectif du projet

Notre objectif a été d'obtenir un programme capable d'écrire des poèmes. Pour cela, on a réduit le problème à un programme qui écrit deux vers qui riment. En répétant cette opération, on peut créer des poèmes de taille arbitraire. Nous nous somme placé dans le cadre classique de vers de 12 syllabes, mais cette donnée importe assez peu. Le programme doit donc avoir les spécification suivantes :

- Entrée : un corpus de textes poétiques.
- Sortie : deux vers, tels que :
  - chaque vers comporte 12 syllabes ;
  - les deux vers riment (mais sont différents) ;
  - le texte a un sens.

On pourra ajouter qu'on désire un programme qui comporte une part d'aléatoire puisqu'on souhaite générer des poèmes différents à chaque appel.

2 Implementations

2.1 Gestion des contraintes

2.1.1 Stockage des mots

Pour chaque vers on stocke les mots qui apparaissent. Or, on ne connait pas a priori le nombre de mots utilisés mais on sait que le nombre de syllabes est fixé à 12. En français il ne peut pas y avoir plus d'un mot sur deux qui ne contient pas de voyelle (donc qui compte pour 0 syllabes). Ceci donne une borne supérieure de 24 mots par phrase.

Pour chaque vers, on stocke donc les 24 mots du vers, avec la possibilité que certains mots soient le *mot vide*. Pour plus de simplicité, tous les mots vides sont regroupés au début du vers.

En réalité, on ne travaille pas avec les mots (les chaînes de caractères) mais chaque mot qui apparaît dans le corpus initial est identifié par un numéro. Soit N le nombre de mots différents. On a donc des variables :

$$x_{i,j}^{(k)} = \begin{cases} 1 & \text{si le } i^{\text{ème}} \text{ mot du vers } k \text{ est } j \\ 0 & \text{sinon.} \end{cases}$$

avec $1 \leq i \leq 24, 1 \leq j \leq N$ et $1 \leq k \leq 2$.

On impose qu'il y a un et un unique mot par position :

$$\forall k \in \{1, 2\}, \forall i \leq N, \qquad \sum_{j=1}^{N} x_{i,j} = 1.$$

2.1.2 Le nombre de syllabes

On stocke pour chaque mot i son nombre de syllabes $syl(i)$. On impose donc la contrainte suivante :

$$\forall k \in \{1, 2\}, \qquad \sum_{i=1}^{24} \sum_{j=1}^{N} x_{i,j}^{(k)} syl(j) = 12.$$

2.1.3 Les rimes

A chaque mot i du corpus, on lui assigne sa classe de rime $rim(i)$, qui correspond aux 5 dernières lettres du mot (ou moins s'il est de taille inférieur). On décide donc que deux mots a et b riment si $rim(a) = rim(b)$.

La condition de rime des deux vers devient alors :

$$\sum_{j=1}^{N} x_{24,j}^{(1)} rim(j) = \sum_{j=1}^{N} x_{24,j}^{(2)} rim(j).$$

2.1.4 Le sens du texte

On cherche à ce que le texte du poème ait un sens. Ceci est évidement une contrainte très forte d'autant plus qu'elle est mal définie. Nous avons décidé de poser des conditions sur l'enchaînements des mots. Lors de la lecture de texte, on compte le nombre de fois qu'un mot succède à un autre. On note $suc(a, b)$ le nombre de fois où les mots a et b se suivent.

La fonction objectif du programme est alors :

$$\max \quad \sum_{k=1}^{2} \sum_{j_1=1}^{N} \sum_{j_2=1}^{N} \sum_{i=1}^{23} \left(x_{i,j_1}^{(k)} \; x_{i+1,j_2}^{(k)} \; suc(j_1, j_2) \right).$$

2.1.5 Caractère aléatoire

Comme cela a déjà été précisé dans les objectifs, le générateur de vers doit aussi avoir un caractère aléatoire : en effet, si les vers générés sont toujours les mêmes, cela limite fortement l'intérêt du projet. La première solution que nous avons trouvée est de gérer à part les fins de vers : en les choisissant de manière aléatoire, on force les vers à changer. Cependant, cet aléatoire n'est que partiel : pour chaque mot, il existe un vers optimal terminant par ce mot. Le générateur de poème est donc toujours d'un intérêt assez limité.

Nous avons donc cherché à introduire un caractère aléatoire dans la construction même des vers. Pour faire ceci, nous générons pour chaque vers une liste de *mots interdits*. En pratique, nous générons un tableau *alea* de taille N, binaire, initialisé de manière alétoire (en arrondissant la fonction *Uniform* de AMPL). $alea(i) = 0$ signifie que le mot i ne pourra pas

être utilisé. Ce tableau est utilisé pour mettre à zéro la valeur des enchainements concernant les mots interdits : on multiplie dans la fonction objectif la bonification $suc(i,j)$ par $alea(i)$ lorsque j est le dernier mot :

$$\max \quad \sum_{k=1}^{2} \sum_{j_1=1}^{N} \sum_{j_2=1}^{N} \left(\sum_{i=1}^{22} x_{i,j_1}^{(k)} \, x_{i+1,j_2}^{(k)} \, suc(j_1,j_2) \; + \; x_{23,j_1}^{(k)} \, x_{24,j_2}^{(k)} \, alea(j_1) \, suc(j_1,j_2) \right).$$

2.2 Génération des données

Le programme nécessite un prétraitement des données : il faut prendre un corpus de textes, les lire et générer les fichiers `.dat` nécessaires. Ce travail a été effectué en Java. Les textes sont d'abord mis dans un format standart (toutes les lettres en majuscule, suppression des accents et de la ponctuation) puis lus mot à mot. On créé plusieurs tables de hachage qui assignent à nouveau mot un nombre (l'index), une classe de rime, un nombre de syllabes, et de même on compte les occurences de successions.

Afin de compter les syllabes, on compte le nombre d'occurence d'une succession consonne-voyelle en supposant que le mot commence par une consonne fictive. Plus précisément, soit $m = m_1 m_2 m_3 \dots m_{k-1} m_k$ un mot, on compte (avec m_0 une consonne) :

$$syl(m) = \sum_{i=1}^{k} \mathbb{1}_{(m_{i-1} \text{ est une consonne})} \mathbb{1}_{(m_i \text{ est une voyelle})}.$$

Les rimes sont déterminées par les 5 dernières lettres d'un mot. Afin de déterminer les mots à la fin des vers, on choisit aléatoirement une classe de rimes parmi celles qui contiennent au moins deux éléments (et telle que la rime commune fait au moins 3 caractères), puis on tire aléatoirement deux éléments différents dans cette classe.

3 Résultats et autres approches

3.1 Résultat

Le programme tel que défini ci-dessus fonctionne bien. Sur un texte d'une longueur de 5000 vers, il lui faut environ 2 secondes de calcul par couple de vers. Le résultat a été légèrement adapté pour générer des sonnets (poèmes de 14 vers). On remarque néanmoins une tendance à favoriser les mots courts (monosyllabes). On pourrait ajouter un bonus à la longueur en multipliant la fonction objectif par le nombre de syllabes.

3.2 Autres approches

Nous avons également essayé d'autres approches plus ambitieuses mais qui n'ont pas donné de résultat satisfaisant.

3.2.1 Approche globale

3.2.2 Syllabe par syllabe

Au lieu de construire les vers mot à mot, nous avons imaginé le faire syllabe par syllabe. L'avantage est qu'on n'a plus la contrainte sur la somme des syllabes des mots puisqu'il suffit de choisir 12 syllabes par vers. Néanmoins si l'on procède ainsi le texte n'a pas de sens.

3.2.3 Par groupes de mots

3.3 Exemple de sonnet généré

SI BIEN RACE D UN SOMBRE ET DU DIFFORME
CE CHAT I RACE D OU SUR LE VIN INFORME
VIE ET SOMBRE ET DE MA VIE ET D UN ORDRE
LE COEUR ET BRULE JUSQU AU FOND DE PERDRE

PLEIN DE TOI BIZARRE DEITE DEITE BRUNE
DE LA VIE ET SOMBRE ET DE LA VIE ET BRUNS
COEUR PLEIN DE MA VIE ET QUE JE TE COMPARER
NATURE AINSI QU A LA VIE ET PAREE

YEUX DE LA VIE ET DE L ON MEPRISE SALUE
ET D AUSSI PETITS RACE D UN DECOR SALUT
ET VOTRE PURE LUMIERE ET QUI S EFFRAYA

ET BRULE PRENDS PITIE DE LA VIE ET LES FRAYE
LA VIE ET DE TA VIE ET DANS LES NATIVE
LA VIE ET COMME UNE NUIT FUGITIVE