Projet 1 : Détection de langue

Gauvain Jodie et Bielle Benjamin

17 octobre 2013

Sommaire

1 **Détection de langue par la méthode naive de Bayes** **2**
1.1 Introduction 2
1.2 Construction d'un modèle de langage 2
1.3 Amélioration du modèle 3
1.4 Visualisation graphique 4
1.5 Conclusion 4

Chapitre 1

Détection de langue par la méthode naive de Bayes

1.1 Introduction

Dans ce projet nous avions pour but de coder, de manière simple, un programme permettant à partir d'un mot de deviner dans laquelle langue il est écrit.
Notre programme s'appuie sur les contraintes suivantes :

- il doit être écrit dans le langage de programmation JAVA.
- il doit s'appuyer sur un corpus de textes dont la langue est connnue : anglais, français, néerlandais et italien.

Exemple :

- devinelangue(pomme) $\rightarrow francais$
- devinelangue(apple) $\rightarrow anglais$

1.2 Construction d'un modèle de langage

La méthode d'identification que nous avons mis en œuvre est fondée sur l'observation que la fréquence d'apparition d'une lettre dépend de la langue dans laquelle on écrit. Par exemple un texte français comportera beaucoup plus de e qu'un texte écrit en anglais ou en italien. Dans la suite de ce document, nous notons w un mot composé de n lettres.

Nous avons tout d'abord supposé que, pour une langue donnée, la probabilité d'observer un mot ne dépend que de la probabilité d'apparition de ses lettres :

$$P(w \mid l) = \prod P(w_i \mid l)$$

La relation précédente est fondée sur l'hypothèse simpliste que les lettres d'un mot sont mutuellement indépendantes. Si l'on dispose d'un corpus écrit dans la langue l, il est possible de déterminer automatiquement les probabilités $P(wi \mid l)$ en les confondant avec les fréquences d'apparition des lettres wi dans ce corpus :

$$P(w_i \mid l) = \frac{nombred'occurencedewi}{nombredecaracteres}$$

Différentes fonctions de bases du programme :

- Une fonction permettant de compter le nombre d'occurences de chaque lettre d'un corpus donné (parcours de fichier, incrémentation dans un tableau de la case correspondante à chaque caractère lu). Nous avons fait les choix de stocker ces informations dans des fichiers de comptes afin d'éviter de reparcourir le corpus à chaque calcul de probabilité.

- Une fonction qui calcule la probabilité d'un mot w sachant une langue l, basée sur les deux formules ci-dessus.

- Une fonction qui calcule la probabilité d'un mot w pour l'ensemble des langues.

Premiers tests :

- $P\left(statistics \mid english\right) = 3.855279615034976E-12$
- $P\left(probability\right) = 1.4449566675812572E-16$

Un mot w a une probabilité max $P\left(l \mid w\right)$ d'être écrit dans une langue l.
Nous pouvons donc écrire la probabilité de la langue l sachant le mot w : $P\left(l \mid w\right) = P\left(w \mid l\right) * \frac{P(l)}{P(w)}$.
Nous avons dans un premier temps ignoré la valeur $P\left(l\right)$ en supposant que nous ne disposions pas d'information a priori permettant d'estimer sa valeur. En procédant à des tests avec la liste de mots donnés dans l'énoncé du projet, nous avons obtenu un taux de réussite de 53% .
Dans un deuxième temps, nous avons évalué la valeur $P\left(l\right)$ comme étant le nombre de caractères écrits dans la langue l, divisé par le nombre de caractères total, et nous avons ainsi obtenu un taux de réussite de 59%.

1.3 Amélioration du modèle

Liste des améliorations du programme.

- Dans un premier temps nous avons augmenté la taille des données de test.

- Nous avons ensuite augmenté la taille du corpus.
  En effet, plus la taille du corpus est grande, plus les probabilités d'apparition des lettres tendent vers leur probabilité effective dans l'espace des mots que l'on cherche à évaluer.
  Nous avons donc crée plusieurs corpus de tailles différentes avec des textes tirés du web dont nous avions connaisance de la langue.
  Seul inconvénient, plus le corpus est grand, plus le temps d'extraction est grand (linéaire selon la taille) mais plus on gagne en pertinence des informations (logarithmique selon la taille des données du corpus).

- Une des améliorations les plus efficaces est de ne plus considérer que les lettres ont la même probabilité d'apparition quelle que soit leur lettre précedente.
  Il s'agit d'utiliser un modele moins simpliste qui suppose que connaissant la langue, la probabilité d'apparition d'une lettre dépend de la lettre précédente du mot (mais est indépendante des autres connaissant cette lettre).
  $P\left(w\right) = P\left(w1\right) * P\left(w2 \mid w1\right) * P\left(w3 \mid w2\right) * ... * P\left(w_i \mid w_{i-1}\right) * ... * P\left(w_n \mid w_{n-1}\right) * P\left(\mid w_n\right)$.
  avec $P\left(w_i \mid w_{i-1}\right) = \frac{P(w_i - l * w_i)}{P(w_{i-1})}$.

1.4 Visualisation graphique

- Test avec différentes tailles de corpus, avec et sans approximation bigramme → *Graphe.eps*

1.5 Conclusion

Au final, on note une augmentation du taux de réussite grâce aux améliorations que nous avons apportées mais nous les performances sont vite limitées.
C'est pourquoi on pourrait imaginer plusieurs hypothèses heuristiques sur les probabilités d'appartenance d'un mot à une langue. Toutes ces hypothèses heuristiques implémentées, affectées d'une pondération pourraient faire tendre la valeur de la performance du système vers 1. C'est-à-dire un système optimal qui donne 100% de bonnes réponses quand on lui passe un mot en paramètre.