Modélisations mathématiques

3. Réalisation d'un système pour reconnaître les auteurs d'une liste de phrases, à l'aide de modèles *n*-grammes

Solen Quiniou

solen.quiniou@univ-nantes.fr

IUT de Nantes

Année 2017-2018 – Info 2

Plan des séances

Plan de des séances

Objectifs du travail

L'objectif de ce second travail sur les modèles de langage est triple :

1. Concevoir et implémenter un premier système pour une tâche de reconnaissance des auteurs des phrases d'un fichier, en utilisant des modèles *n*-grammes.

2. Construire des modèles *n*-grammes sur des corpus d'apprentissage pour les utiliser dans le précédent système de reconnaissance, afin d'améliorer la reconnaissance des auteurs des phrases.

3. Étendre le système précédent pour détecter si les phrases ont été écrites par des auteurs inconnus de ceux du système.

→ **Le travail sera réalisé en binôme (préférablement) ou seul.**

Déroulement d'une tâche de reconnaissance

La tâche de reconnaissance des auteurs des phrases données dans un fichier, se déroule en 2 phases :

1. Phase de développement d'un système de reconnaissance
   - Création d'un modèle de langage pour chaque auteur, à partir des corpus dits d'apprentissage.
   - Réalisation du système de reconnaissance, en utilisant et en combinant les modèles de langage des différents auteurs.

2. Phase de validation du système de reconnaissance
   - Création du fichier hypothèses d'auteurs, en utilisant le système de reconnaissance sur chacune des phrases pour lesquelles il faut identifier l'auteur ; le fichier hypothèses d'auteurs contient le nom de l'auteur reconnu par le système, pour chaque phrase considérée.
   - Calcul des performances du système en comparant le fichier hypothèses d'auteurs et le fichier référence d'auteurs ; le fichier référence d'auteurs contient les noms des auteurs ayant réellement écrit chaque phrase.
   - Modification éventuelle du système si les performances ne sont pas satisfaisantes et nouvelle phase de validation à effectuer.

$\rightarrow$ Les 2 phases précédentes sont répétées, pour chaque nouveau système de reconnaissance créé.

Déroulement du travail

1. Exercice 1 : utilisation de la classe `BaselineAuthorRecognizer` pour prendre en main l'utilisation d'un système simple de reconnaissance des auteurs d'un fichier de phrases.

2. Exercice 2 : implémentation de la classe `AuthorRecognizer1` pour créer et utiliser un premier système de reconnaissance d'auteurs de phrases, en utilisant les modèles *n*-grammes simples fournis.

3. Exercice 3 : implémentation de la classe `CreateLanguageModels` pour construire de nouveaux modèles *n*-grammes plus performants, en utilisant les corpus d'apprentissage, de taille plus conséquente, fournis.

4. Exercice 4 : implémentation de la classe `UnknownAuthorRecognizer1` pour créer un nouveau système de reconnaissance capable de détecter également les phrases d'auteurs inconnus du système de reco.

5. Exercice 5 : création de classes `UnknownAuthorRecognizerXX` pour proposer de nouveaux algorithmes de reconnaissance et de détection des auteurs inconnus.

Projet Eclipse : reco. des auteurs des phrases

- Projet Eclipse : `Etudiant-mm_authorReco`
  - Récupérez le projet sur Madoc puis ajoutez-le dans Eclipse.
- Contenu du projet
  - `data/author_corpus/train` : répertoire contenant les corpus d'apprentissage plus conséquents, à utiliser pour créer de nouveaux ML.
  - `data/author_corpus/validation` : répertoire contenant les fichiers de validation plus conséquents.
  - `data/small_author_corpus/train` : répertoire contenant les petits corpus d'apprentissage, utilisés pour construire les petits ML fournis.
  - `data/small_author_corpus/validation` : répertoire contenant les fichiers à utiliser pour évaluer vos systèmes de reconnaissance.
    - ⋆ `sentences_100sentences.txt` : fichier contenant les phrases pour lesquelles il faut identifier les auteurs.
    - ⋆ `authors.txt` : fichier contenant les nom des auteurs reconnus par les systèmes de reconaissance, pour les phrases du fichier précédent.
    - ⋆ `authors_100sentences_ref.txt` : fichier de référence contenant les auteurs associés aux phrases du premier fichier.
  - `doc` : répertoire contenant la documentation des classes du projet.
  - `lib` : répertoire contenant les bibliothèques utiles au projet.
  - `lm` : répertoire contenant les fichiers de configuration, de vocabulaire et de modèles de langage à utiliser dans les systèmes de reco.
  - `src` : répertoire contenant les classes du projet.

Plan de des séances

Exercice 1 : classe `BaselineAuthorRecognizer`

- Objectif de l'exercice
  - Évaluer les performances du système de reconnaissance de base, en complétant la méthode principale de la classe `BaselineAuthorRecognizer`.

- Classe à compléter : `authorReco.BaselineAuthorRecognizer`
  - La classe contient une *approche de base* qui servira de base de comparaison avec vos différents systèmes.
    Cette approche de base est naïve et simple : pour chaque phrase en entrée du système, le système de reconnaissance choisit aléatoirement un auteur parmi l'ensemble des auteurs connus.

  → Implémenter la méthode principale de la classe pour
  - ⋆ créer un objet de type `BaselineAuthorRecognizer` ;
  - ⋆ utiliser cet objet pour générer le fichier d'hypothèses `authors_100sentences_hyp-baseline.txt` contenant les noms des auteurs reconnus pour chacune des phrases du fichier `sentences_100sentences.txt`
  - ⋆ utiliser la classe `authorEval.RecognizerPerformance` pour comparer le fichier d'hypothèses généré et le fichier de référence d'auteurs `authors_100sentences_ref.txt`.

Plan de des séances

Exercice 2 : classe `AuthorRecognizer1` (1)

- Objectif de l'exercice
  - Implémenter un premier système de reconnaissance de l'auteur d'une phrase, parmi un ensemble d'auteurs connus, en utilisant de petits modèles de langage (le système sera amélioré par la suite).

- Classe à compléter : `authorReco.AuthorRecognizer1`

  → Implémentez le constructeur `AuthorRecognizer1 (..)` :
  - ⋆ vous devrez charger un fichier contenant les auteurs reconnus par le système, un fichier de vocabulaire pour les modèles de langage et un fichier de configuration contenant les informations pour construire les ML utilisés.
  - ⋆ vous devrez ensuite initialiser l'attribut `authorLangModelsMap` qui contient les modèles de langage utilisés dans le système de reconnaissance.

  → Implémentez la méthode `String recognizeSentenceLanguage (String sentence)` qui retourne le nom de l'auteur reconnu pour la phrase `sentence`.

  → Implémenter la méthode principale `main(..)` de la classe pour créer un système de reco, générer le fichier d'hypothèses d'auteurs `authors_100sentences_hyp1.txt` et comparer ce fichier au fichier de référence d'auteurs (voir exercice 1).

Exercice 2 : classe `AuthorRecognizer1` (2)

- Indications sur la classe `AuthorRecognizer1`
  - La classe `AuthorRecognizer1` hérite de la classe `AuthorRecognizerAbstractClass`.
  - La méthode `loadAuthorConfigurationFile(String configFile)` permet de charger un fichier de configuration de modèles de langage et d'initialiser l'attribut `configLangModels`.

  → Chaque ligne du fichier de configuration contient le nom de l'auteur correspondant au modèle de langage, un nom unique identifiant le ML dans le système ainsi que le chemin du fichier contenant ce ML.

  - Les méthodes `loadVocabularyFile(..)` et `loadAuthorFile(..)` permettent de charger, respectivement, un fichier de vocabulaire et un fichier de noms d'auteurs.

Plan de des séances

Exercice 3 : classe `CreateLanguageModels`

- Objectif de l'exercice
  - Construire des modèles de langage plus performants que les modèles bigrammes fournis, pour améliorer les performances du premier système de reconnaissance et de systèmes de reconnaissance supplémentaires.

- Classe à compléter : `src/langReco.CreateLanguageModels`

  → Implémenter la méthode principale de la classe pour créer de nouveaux modèles de langage, en utilisant les corpus d'apprentissage du répertoire `data/author_corpus/train`.

  - Vous aurez également à modifier la méthode principale de la classe `AuthorRecognizer1` pour utiliser vos nouveaux modèles de langage (en créant un nouveau fichier de configuration).

Plan de des séances

Exercice 4 : classe `UnknownAuthorRecognizer1`

- Objectif de l'exercice
  - Implémenter un nouveau système de reconnaissance pour pouvoir détecter les phrases dont l'auteur est inconnu, c'est-à-dire qu'il ne fait pas partie des auteurs pour lesquels le système possède des modèles de langage.

- Classe à compléter : `src/langReco.UnknownAuthorRecognizer1`
  - Cette classe hérite de la classe `langReco.AuthorRecognizer1` et prend en compte le fait qu'une phrase peut être d'un auteur inconnu.

  → Redéfinissez la méthode `recognizeAuthorSentence(..)` en cherchant la meilleure stratégie pour détecter qu'une phrase est d'un auteur inconnu à l'aide des modèles de langage des auteurs connus du système de reco.
    - ⋆ Le code associé à une phrase d'auteur inconnu est `unknown` : il est à retourner si la phrase `sentence` est d'un auteur inconnu.

  → Implémenter la méthode principale de la classe, comme précédemment.

Plan de des séances

Exercice 5 : classes `UnknownAuthorRecognizerXX`

- Classes à créer : `UnknownAuthorRecognizerXX` (`XX` : nombre supérieur à 1)
  - Chaque classe `UnknownAuthorRecognizerXX` héritera de la classe `UnknownAuthorRecognizer1` et redéfinira les méthodes `recognizeAuthorSentence(..)` et `main(..)`.

  → La redéfinition de `recognizeAuthorSentence(..)` permettra de proposer de nouveaux algorithmes de reconnaissance des auteurs (par exemple, en combinant des modèles de langage d'ordres différents, pour chaque auteur, construits sur des mots ou sur des caractères. . .) et de détection des auteurs inconnus.