Modélisations mathématiques

2. Développement d'une bibliothèque de gestion des modèles de langage de type *n*-grammes

Solen Quiniou

solen.quiniou@univ-nantes.fr

IUT de Nantes

Année 2020-2021 – Info 2

(dernière mise à jour : 3 décembre 2020)

Plan des séances

Plan de la séance

Objectifs du travail

L'objectif de ce premier travail sur les modèles de langage (ML) est triple :

1. Comprendre et utiliser une implémentation d'une bibliothèque permettant de créer et d'utiliser des modèles de langage ;
2. Implémenter, en partie, cette bibliothèque pour créer et utiliser des modèles de langage ;
3. Utiliser votre bibliothèque dans les systèmes de reconnaissance que vous implémenterez lors de la semaine 3.

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

Plan de la séance

Projet Java 1 : prise en main des ML

- Projet 1 : `Etudiant-mm_useLangModel`
  - Récupérez le projet sur Madoc puis ajoutez-le dans Eclipse ou dans IntelliJ, en vérifiant que la bibliothèque de modèles de langage et celle de JUnit sont bien reconnues

- Contenu du projet
  - `data` : corpus d'apprentissage en français, à utiliser pour construire des modèles de langage.
  - `doc` : documentation des classes de la bibliothèque de modèles de langage.
  - `lib` : bibliothèques utiles au projet.
  - `lm` : fichiers de vocabulaire et de modèles de langage.

  - `src/Application_LanguageModels` : méthode `main` à compléter, pour utiliser la bibliothèque de modèles de langage.
  - `test` : répertoire qui devra contenir les classes de test de chacune des classes de la bibliothèque de modèles de langage.

- Exercices concernés : exercice 1

Plan de la séance

Exercice 1 : prise en main de la bibliothèque de ML

- Objectif de l'exercice
  - Se familiariser avec la bibliothèque de création et d'utilisation des modèles de langage, grâce au projet `Etudiant-mm_useLangModel`, avant d'en implémenter certaines classes.

- Travail à réaliser
  1. Regardez la documentation de la bibliothèque ainsi que les indications sur les classes qui se trouvent ci-après

  2. Complétez les 3 classes de test du répertoire `test/langModel`, en écrivant au moins 2 tests par méthode : classes `NgramUtilsTest`, `NaiveLanguageModel` et `LaplaceLanguageModel`.

  → Si vous le souhaitez, vous pouvez ajouter des classes de test pour les autres classes de la bibliothèque de modèles de langage.

  3. Implémentez la méthode `main` de la classe `src/Application_LanguageModels`, en reprenant les exemples du cours de la semaine 1 et en suivant les indications données en commentaire.

Plan de la séance

Classe `NgramUtils` (1)

- Description de la classe
  - La classe `NgramUtils` permet de manipuler les *n*-grammes en découpant une phrase en ses *n*-grammes, en générant les *n*-grammes composant une phrase, en calculant l'historique d'un *n*-gramme dans une phrase...
  - Cette classe ne contient que des méthodes statiques.

- Indications générales sur les méthodes de la classe
  - Par simplicité, la structure de données choisie pour représenter un *n*-gramme est la chaîne de caractères.

  $\rightarrow$ Les *n*-grammes sont mis en minuscule.

  $\rightarrow$ L'espace est utilisé pour séparer les mots composant un *n*-gramme.

Classe `NgramUtils` (2)

- Méthode `decomposeIntoNgrams(..)`
  - La méthode `List<String> decomposeIntoNgrams(String sentence, int order)` retourne la liste des *n*-grammes d'ordre `order` composant la phrase `sentence`.

  → Par exemple, `decomposeIntoNgrams("<s> il fait beau </s>", 3)` retourne la liste suivante : `["<s>", "<s> il", "<s> il fait", "il fait beau", "fait beau </s>"]`.

- Méthode `generateNgrams(..)`
  - La méthode `List<String> generateNgrams (String sentence, int minOrder, int maxOrder)` retourne la liste des *n*-grammes d'ordre `minOrder`, d'ordre `minOrder+1`, ..., jusqu'à l'ordre `maxOrder`, composant la phrase `sentence`.

  → Par exemple, `generateNgrams("<s> il fait beau </s>", 1, 3)` retourne la liste suivante : `["<s>", "il", "fait", "beau", "</s>", "<s> il", "il fait", "fait beau", "beau </s>", "<s> il fait", "il fait beau", "fait beau </s>"]`.

Classe `Vocabulary`

- Description de la classe
  - La classe `Vocabulary` permet de manipuler les mots associés à un modèle de langage.
  - Cette classe implémente l'interface `VocabularyInterface`.
  - En pratique, le vocabulaire sera créé lors de l'association d'un objet de type `NgramCounts` à un objet de type `LanguageModel`.

Classe `NgramCounts` (1)

- Description de la classe
  - Les méthodes de la classe `NgramCounts` permettent de stocker et de manipuler les *n*-grammes apparaissant dans un corpus d'apprentissage et leurs nombres d'occurrences dans ce corpus.
  - Cette classe implémente l'interface `NgramCountsInterface`.

- Indications générales sur les méthodes de la classe
  - Les méthodes de la classe `NgramUtils` seront utilisées pour faire les manipulation de base sur les *n*-grammes.

  - Le format des corpus d'apprentissage est le suivant : chaque ligne contient une phrase qui commence par `<s>`, dont les mots sont séparés par des espaces et qui se termine par `</s>`.

Classe `NgramCounts` (2)

- Représentation des modèles de langage
  - Par simplicité, un modèle de langage est stocké dans un fichier.

  → Chaque ligne contient un *n*-gramme suivi de son nombre d'occurrences (séparés d'une tabulation), le tout calculé sur un corpus d'apprentissage.
  → Les nombres d'occurrences des *n*-grammes seront ensuite utilisés pour calculer les probabilités dans les modèles de langagee.

- Implémentation dans la classe `NgramCounts`
  - L'attribut `ngramCounts` est une table de hachage : chaque clé est un *n*-gramme et la valeur associée correspond à son nombre d'occurrences.

  → La table de hachage peut être initialisée de 2 manières :
    - ⋆ soit en analysant un corpus d'apprentissage (méthode `scanTextFile(...)`)
    - ⋆ soit en lisant une sauvegarde de nombres d'occurrences de *n*-grammes (méthode `readNgramCountsFile(...)`) ; cette sauvegarde aura été préalablement calculée sur un corpus d'apprentissage et enregistrée dans un fichier (méthode `writeNgramCountsFile(...)`).

Plan de la séance

Projet Java 2 : implémentation des ML

- Projet 2 : `Etudiant-mm_langModel`
  - Récupérez le projet sur Madoc puis ajoutez-le dans Eclipse ou IntelliJ, comme précédemment.

- Contenu du projet
  - `data` : corpus d'apprentissage en français, à utiliser pour construire des modèles de langage.
  - `doc` : documentation des classes de la bibliothèque de modèles de langage.
  - `lib` : bibliothèques utiles au projet.
  - `lib_output` : bibliothèques générées dans le projet.
  - `lm` : fichiers de vocabulaire et de modèles de langage.

  - `src/langModel` : classes fournies et classes à implémenter, pour manipuler les modèles de langage.
  - `test/langModel` : classes de test à copier à partir du premier projet, pour tester les classes du répertoire `src/langModel`.

- Exercices concernés : exercices 2 et 3

Plan de la séance

Exercice 2 : classe `NaiveLanguageModel`

- Description de la classe
  - Les méthodes de la classe `NaiveLanguageModel` permettent de représenter des modèles de langage de type *n*-gramme sans lissage.
  - Cette classe implémente l'interface `LanguageModelInterface`.

- Travail à réaliser
  1. Implémenter toutes les méthodes de la classe `NaiveLanguageModel`.

- Indications sur les méthodes à implémenter
  - Méthode `String getNgramProb(String ngram)`
    - ⋆ Vous devez tout d'abord remplacer les mots du *n*-gramme `ngram` donné, qui sont absents du vocabulaire donné par l'attribut `vocab`, en utilisant la méthode `getStringOOV(ngram, vocab)`.
    - ⋆ Pour réaliser le calcul de la probabilité du *n*-gramme `ngram`, utilisez la formule donnée dans le transparent 12 du cours de la semaine 1.

    → Par simplicté, les probabilités données par le modèle de langage sont calculées « à la volée », c'est-à-dire à partir des nombres d'occurrences des *n*-grammes stockés dans l'attribut `ngramCounts` (qui est de type `NgramCounts`).

  - Méthode `Double getSentenceProb(String sentence)`
    - ⋆ Pour réaliser le calcul de la probabilité de la phrase, utilisez la formule donnée dans le transparent 10 du cours de la semaine 1.

Plan de la séance

Exercice 3 : classe `LaplaceLanguageModel`

- Description de la classe
  - Les méthodes de la classe `LaplaceLanguageModel` permettent de représenter des modèles de langage de type *n*-gramme avec lissage de Laplace.
  - Cette classe hérite de la classe `NaiveLanguageModel`.

- Travail à réaliser
  1. Implémenter l'unique méthode redéfinie dans la classe `LaplaceLanguageModel`.

- Indications sur les méthodes à implémenter
  - Méthode `String getNgramProb(String ngram)`
    - ⋆ Pour redéfinir le calcul de la probabilité du *n*-gramme `ngram`, avec le lissage de Laplace, utilisez la formule donnée dans le transparent 16 du cours de la semaine 1.

Plan de la séance

Partie 3 (bonus) : pour aller plus loin (1)

- Bonus 1 : création de la bibliothèque
  - Créer un `jar` correspondant à votre bibliothèque pour l'utiliser dans votre projet de reconnaissance (semaine 3), à la place du `jar` que nous vous fournissons.
  - Sous Eclipse, faites un clic droit sur votre projet Java, choisissez `Export...` puis `Java/JAR file` et utilisez le fichier `lib_output/create-ml-jar.jardesc` fourni pour créer votre jar final.

- Bonus 2 : log-probabilités à la place des probabilités
  - L'utilisation des log-probabilités (logarithme décimal des probabilités) au lieu des probabilités permet d'éviter les débordements lors de la multiplication des probabilités très faibles, en utilisant des additions au lieu des multiplications.

  → Modifiez les classes concernées pour utiliser des log-probabilités au lieu des probabilités.