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 2017-2018 – Info 2

Plan des séances

1. Objectifs, déroulement du travail et projets Eclipse
2. Exercice 1 : prise en main de la bibliothèque de modèles de langage
3. Exercice 2 : implémentation de la classe Vocabulary
4. Exercice 3 : implémentation de la classe NgramUtils
5. Exercice 4 : implémentation de la classe NgramCounts
6. Exercice 5 : implémentation de la classe NaiveLanguageModel
7. Exercice 6 : implémentation de la classe LaplaceLanguageModel
8. Exercice 7 : création de la bibliothèque
9. Exercice 8 (bonus) : pour aller plus loin

Plan des séances

1. Objectifs, déroulement du travail et projets Eclipse
2. Exercice 1 : prise en main de la bibliothèque de modèles de langage
3. Exercice 2 : implémentation de la classe Vocabulary
4. Exercice 3 : implémentation de la classe NgramUtils
5. Exercice 4 : implémentation de la classe NgramCounts
6. Exercice 5 : implémentation de la classe NaiveLanguageModel
7. Exercice 6 : implémentation de la classe LaplaceLanguageModel
8. Exercice 7 : création de la bibliothèque
9. Exercice 8 (bonus) : pour aller plus loin

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 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.**

Déroulement du travail d'implémentation

1. Exercice 1 : prise en main de la bibliothèque de modèles de langage.
2. Exercice 2 : implémentation de la classe `Vocabulary`.
3. Exercice 3 : implémentation de la classe `NgramUtils`.
4. Exercice 4 : implémentation de la classe `NgramCounts`.
5. Exercice 5 : implémentation de la classe `NaiveLanguageModel`.
6. Exercice 6 : implémentation de la classe `LaplaceLanguageModel`.
7. Exercice 7 : création du fichier `jar` correspondant à la bibliothèque.
8. Exercice 8 (bonus) : ajouts pour aller plus loin sur les ML.

Projet Eclipse 1 : prise en main des ML

- Projet Eclipse : `Etudiant-mm_useLangModel`
  - Récupérez le projet sur Madoc puis ajoutez-le dans Eclipse : sélectionnez `Import...` puis `General/Existing Projects into Workspace` et choisissez le fichier correspondant à l'archive récupérée sur Madoc dans `Select archive file`.

- 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.

- Exercices concernés : exercice 1

Projet Eclipse 2 : implémentation des ML

- Projet Eclipse : `Etudiant-mm_langModel`
  - Récupérez le projet sur Madoc puis ajoutez-le dans Eclipse 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 à implémenter, pour manipuler les modèles de langage.
  - `test/langModel` : classes de test à créer, pour tester les classes du répertoire `src/langModel`.

- Exercices concernés : exercices 2 à 8

Plan des séances

1. Objectifs, déroulement du travail et projets Eclipse
2. Exercice 1 : prise en main de la bibliothèque de modèles de langage
3. Exercice 2 : implémentation de la classe Vocabulary
4. Exercice 3 : implémentation de la classe NgramUtils
5. Exercice 4 : implémentation de la classe NgramCounts
6. Exercice 5 : implémentation de la classe NaiveLanguageModel
7. Exercice 6 : implémentation de la classe LaplaceLanguageModel
8. Exercice 7 : création de la bibliothèque
9. Exercice 8 (bonus) : pour aller plus loin

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 Eclipse `Etudiant-mm_useLangModel`, avant d'en implémenter les différentes classes.

- Prise en main des classes de la bibliothèque
  - Lisez la documentation de la bibliothèque et dessinez le diagramme de classes de celle-ci.

- Classe à compléter : `src/Application_LanguageModels`
  - Implémentez la méthode `main` de cette classe, en reprenant les exemples du cours de la semaine 1 et en suivant les indications données en commentaire.

Plan des séances

1. Objectifs, déroulement du travail et projets Eclipse
2. Exercice 1 : prise en main de la bibliothèque de modèles de langage
3. Exercice 2 : implémentation de la classe Vocabulary
4. Exercice 3 : implémentation de la classe NgramUtils
5. Exercice 4 : implémentation de la classe NgramCounts
6. Exercice 5 : implémentation de la classe NaiveLanguageModel
7. Exercice 6 : implémentation de la classe LaplaceLanguageModel
8. Exercice 7 : création de la bibliothèque
9. Exercice 8 (bonus) : pour aller plus loin

Exercice 2 : classe `Vocabulary`

- Objectif de l'exercice
  - Implémenter les méthodes de la classe `Vocabulary` qui permettent de manipuler les mots associés à un modèle de langage.

- Classe à compléter : `src/langModel.Vocabulary`
  - Cette classe implémente l'interface `VocabularyInterface`.
    objet de type `NgramCounts` à un objet de type `LanguageModel` (voir exercices 4, 5 et 6).

  → Vous aurez ensuite à créer une classe de test `VocabularyTest` (à placer ensuite dans le répertoire `test/langModel`) pour tester les méthodes de la classe `Vocabulary`, en utilisant des tests `JUnit`.

Plan des séances

1. Objectifs, déroulement du travail et projets Eclipse
2. Exercice 1 : prise en main de la bibliothèque de modèles de langage
3. Exercice 2 : implémentation de la classe Vocabulary
4. Exercice 3 : implémentation de la classe NgramUtils
5. Exercice 4 : implémentation de la classe NgramCounts
6. Exercice 5 : implémentation de la classe NaiveLanguageModel
7. Exercice 6 : implémentation de la classe LaplaceLanguageModel
8. Exercice 7 : création de la bibliothèque
9. Exercice 8 (bonus) : pour aller plus loin

Exercice 3 : classe `NgramUtils` (1)

- Objectif de l'exercice
  - Implémenter les méthodes de la classe `NgramUtils` pour 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...

- Classe à compléter : `src/langModel.NgramUtils`
  - Implémenter les méthodes de cette classe, en vous aidant des commentaires présents dans les en-têtes des méthodes.

  → Vous compléterez également la classe `test/langModel.NgramUtilsTest`, pour tester les méthodes de la classe `NgramUtils` en utilisant des tests `JUnit`.

Exercice 3 : classe `NgramUtils` (2)

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

  → Les *n*-grammes sont mis en minuscule.

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

- Quelques méthodes utiles de la classe `String`
  - La méthode `split()` permet de découper une chaîne de caractères en un tableau de chaînes de caractères, selon un séparateur :
    `String[] wArray = sentence.split("\\s+");`
  - La méthode `trim()` retourne la chaîne sans les caractères espaces/blancs de début et fin de chaîne : `ngram = ngram.trim();`
  - La méthode `parseInt()` permet de convertir une chaîne de caractères en l'entier qu'elle contient : `int nInt = Integer.parseInt("123");`
  - La méthode `asList()` permet de transformer un tableau en une liste :
    `List<String> wordList = Arrays.asList(wArray);`

Exercice 3 : classe `NgramUtils` (3)

- 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>"]`.

  → Attention aux doublons obtenus si on utilise `decomposeIntoNgrams(..)` aux ordres 1, 2 et 3 puis que l'on concatène les 3 listes ainsi obtenues.

Plan des séances

1. Objectifs, déroulement du travail et projets Eclipse
2. Exercice 1 : prise en main de la bibliothèque de modèles de langage
3. Exercice 2 : implémentation de la classe Vocabulary
4. Exercice 3 : implémentation de la classe NgramUtils
5. Exercice 4 : implémentation de la classe NgramCounts
6. Exercice 5 : implémentation de la classe NaiveLanguageModel
7. Exercice 6 : implémentation de la classe LaplaceLanguageModel
8. Exercice 7 : création de la bibliothèque
9. Exercice 8 (bonus) : pour aller plus loin

Exercice 4 : classe `NgramCount` (1)

- Objectif de l'exercice
  - Implémenter les méthodes de la classe `NgramCounts` qui permettent de stocker et de manipuler les *n*-grammes apparaissant dans un corpus d'apprentissage et leurs nombres d'occurrences dans ce corpus.

- Classe à compléter : `src/langModel.NgramCounts`
  - Cette classe implémente l'interface `NgramCountsInterface`.

  → Vous aurez ensuite à créer une classe de test `NgramCountsTest` (à placer ensuite dans le répertoire `test/langModel`) pour tester les méthodes de la classe `NgramCounts`, en utilisant des tests `JUnit`.

Exercice 4 : classe `NgramCount` (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(...)`).

Exercice 4 : classe `NgramCount` (3)

- Indications sur les méthodes à implémenter
  - 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>`.

Plan des séances

1. Objectifs, déroulement du travail et projets Eclipse
2. Exercice 1 : prise en main de la bibliothèque de modèles de langage
3. Exercice 2 : implémentation de la classe Vocabulary
4. Exercice 3 : implémentation de la classe NgramUtils
5. Exercice 4 : implémentation de la classe NgramCounts
6. Exercice 5 : implémentation de la classe NaiveLanguageModel
7. Exercice 6 : implémentation de la classe LaplaceLanguageModel
8. Exercice 7 : création de la bibliothèque
9. Exercice 8 (bonus) : pour aller plus loin

Exercice 5 : classe `NaiveLanguageModel` (1)

- Objectif de l'exercice
  - Implémenter les méthodes de la classe `NaiveLanguageModel` qui permet de représenter des modèles de langage de type *n*-gramme sans lissage.

- Classe à compléter : `src/langModel.NaiveLanguageModel`
  - Cette classe implémente l'interface `LanguageModelInterface`.

  → Vous aurez ensuite à créer une classe de test `NaiveLanguageModelTest` (à placer ensuite dans le répertoire `test/langModel`) pour tester les méthodes de la classe `NaiveLanguageModel`, en utilisant des tests `JUnit`.

Exercice 5 : classe `NaiveLanguageModel` (2)

- Indications sur les méthodes à implémenter
  - 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 un attribut de type `NgramCounts`.

  → Pour réaliser ce calcul, utilisez plutôt la formule donnée dans le transparent 12 du cours de la semaine 1.

  - L'attribut `vocabulary` est de type `Vocabulary` : il est initialisé à partir du vocabulaire stocké dans un fichier.

Plan des séances

1. Objectifs, déroulement du travail et projets Eclipse
2. Exercice 1 : prise en main de la bibliothèque de modèles de langage
3. Exercice 2 : implémentation de la classe Vocabulary
4. Exercice 3 : implémentation de la classe NgramUtils
5. Exercice 4 : implémentation de la classe NgramCounts
6. Exercice 5 : implémentation de la classe NaiveLanguageModel
7. Exercice 6 : implémentation de la classe LaplaceLanguageModel
8. Exercice 7 : création de la bibliothèque
9. Exercice 8 (bonus) : pour aller plus loin

Exercice 6 : classe `LaplaceLanguageModel`

- Objectif de l'exercice
  - Implémenter les méthodes de la classe `LaplaceLanguageModel` qui permet de représenter des modèles de langage de type *n*-gramme avec lissage de Laplace.

- Classe à compléter : `src/langModel.LaplaceLanguageModel`
  - Cette classe hérite de la classe `NaiveLanguageModel`.

  → Vous aurez ensuite à créer une classe de test `LaplaceLanguageModelTest` (à placer ensuite dans le répertoire `test/langModel`) pour tester les méthodes de la classe `LaplaceLanguageModel`, en utilisant des tests `JUnit`.

Plan des séances

1. Objectifs, déroulement du travail et projets Eclipse
2. Exercice 1 : prise en main de la bibliothèque de modèles de langage
3. Exercice 2 : implémentation de la classe Vocabulary
4. Exercice 3 : implémentation de la classe NgramUtils
5. Exercice 4 : implémentation de la classe NgramCounts
6. Exercice 5 : implémentation de la classe NaiveLanguageModel
7. Exercice 6 : implémentation de la classe LaplaceLanguageModel
8. Exercice 7 : création de la bibliothèque
9. Exercice 8 (bonus) : pour aller plus loin

Exercice 7 : création de la bibliothèque

- Objectif de l'exercice
  - 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 aurons fourni.

- Création et utilisation du `jar`
  - Faites un clic droit sur votre projet Eclipse et choisissez `Export...` puis `Java/JAR file` et utilisez le fichier `lib_output/create-ml-jar.jardesc` fourni pour créer votre fichier `JAR` final.

  - Remplacez le fichier `jar`, se trouvant dans le répertoire `lib` du projet Eclipse de la semaine 3, par le fichier que vous aurez généré et modifiez le nom de votre fichier `jar` pour que ce soit le même que celui du fichier que nous vous aurons fourni.

Plan des séances

1. Objectifs, déroulement du travail et projets Eclipse
2. Exercice 1 : prise en main de la bibliothèque de modèles de langage
3. Exercice 2 : implémentation de la classe Vocabulary
4. Exercice 3 : implémentation de la classe NgramUtils
5. Exercice 4 : implémentation de la classe NgramCounts
6. Exercice 5 : implémentation de la classe NaiveLanguageModel
7. Exercice 6 : implémentation de la classe LaplaceLanguageModel
8. Exercice 7 : création de la bibliothèque
9. Exercice 8 (bonus) : pour aller plus loin

Exercice 8 (bonus) : pour aller plus loin

- Bonus 1 : 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.

- Bonus 2 : interpolation de modèles de langage
  - L'interpolation de modèles de langage permet de construire un modèle de langage à partir de plusieurs ML.

  → Ajoutez une nouvelle classe `InterpolatedLanguageModel` pour construire et utiliser des modèles de langage interpolés.