IUT de Nantes

Année 2015-2016

Modélisation Mathématique

Auteurs :

* Nathan Maraval
* Cédric Berland



# Introduction

Ce projet a été réalisé dans le cadre de notre formation en DUT Informatique, à l’IUT de Nantes. Il fût demandé durant le cours de Modélisation Mathématique.

Le but de ce travail est de pouvoir identifier, à partir de probabilités, le langage d’origine d’une phrase donnée, en ayant par avant utiliser des corpus d’apprentissage dans plusieurs langues.

Le projet se sépare en deux parties, une ayant pour principe de générer les probabilités des phrases, et l’autre cherchant les langages les plus probable, tout en gérant les langages d’origine inconnue.

# Différents problèmes rencontrés

## Les langages inconnues

Durant ce projet, nous avons pu rencontrer quelques problèmes. Le principal de tous ces problèmes fût la gestion des langues inconnues. Dans un modèle de langage Naïf, toutes langues inconnues auront une probabilité de zéro, mais ce genre de modèle n’est pas très fiable – en effet, si un mot est inconnu, c’est toute la phrase qui perdra sa probabilité.

Nous devons donc utiliser un modèle dit de Laplace, qui affectera à chaque tuple de la phrase un poids, ce qui empêchera d’avoir une probabilité de 0 pour un simple mot inconnue dans une phrase.

Cependant, si une phrase est inconnue, cela veut dire que toutes les langues dans le modèle que nous avons créé auront une probabilité non-nul, dans un intervalle pouvant varier. Il faut donc se mettre d’accords sur un seuil à partir duquel la probabilité que la phrase appartienne à telle langue est trop faible, et il devient plus probable que la langue est inconnue.

Pour ce faire, nous avons pris en compte le fait que pour une langue, plus la taille du vocabulaire est grande, plus la probabilité de la phrase est petite. Et le fait que plus la taille de la phrase est grande plus la plus la probabilité de la phrase est grande. Après plusieurs tests, nous nous sommes mis d’accords sur la formule suivante pour le seuil :

1/ (taille du vocabulaire) (taille de la phrase) \* 10 000

## La ponctuation

Un problème rencontré fût que, si un langage utilisait beaucoup de ponctuation, sa probabilité était tronquée. Nous avons rencontré ce problème sur la langue Tchèque, qui apparaissait plus souvent, au lieu de la langue inconnue. Notre idée fût de supprimer toutes les ponctuations des phrases que nous testions, en utilisant des expressions régulières. Les ponctuations étaient donc bien supprimées, mais il fut avéré que les probabilités n’étaient pas plus justes. Il semble donc que la ponctuation soit importante et propre à chaque langue, au même titre que n’importe qu’elle mot. Donc nous n’avons pas gardés cette partie dans notre code.

## Présence de l’anglais

Après avoir réalisé la plupart des tests, nous avons observé que l’anglais avait tendance à être reconnu à la place d’autres langages. Après avoir regardé les fichiers de bigrams que nous avions générés, nous nous sommes rendu compte que l’anglais possédait le moins de bigrammes différents. Ceci est dû au fait que l’anglais réutilise souvent les mêmes mots. Pour éviter ce problème, il faudrait avoir une base de mots plus conséquente, ou alors adapter la probabilité d’une phrase en fonction de la taille du vocabulaire de la langue ciblé.

# Etat d’avancement

## Modèle de langue

L’implémentation de toutes les méthodes sur le projet langModel a été achevée. Chaque classe fonctionne, et ont été testés. Elles implémentent toutes une interface permettant la gestion du vocabulaire, du compte de n-gramme, et du modèle de langage.

## Reconnaissance de langue

Le projet langModel a été mis sous forme de .jar et utilisé comme bibliothèque dans la reconnaissance. On hérite LanguageRecognizer dans différentes classes, en fonction du modèle utilisé. Et la reconnaissance de langage inconnu a été mise en place.

Le constructeur va initialiser un MyLaplaceLanguageModel pour chaque langue. On va appliquer à chacun de ses langages de modèles un MyNgramCounts. Tous ces langages de modèles vont ensuite être placés dans la variable de classe lms, dans le but de pouvoir les réutiliser rapidement, et de ne devoir les créer que lors de la création de la classe. La variable de classe lms est un HashMap avec la langue comme clé car c’est le moyen le plus efficace de récupérer le modèle de language.

Ensuite l’algorithme qui a pour but de reconnaitre la langue de la phrase va récupérer tous les langages utilisés, afin de pouvoir tester la phrase avec tous les modèles de langage différents. On part du principe que la langue est inconnue, et on attend une probabilité supérieure à une valeur minimale calculée en fonction de la taille de la phrase et de la taille du vocabulaire pour considérer que la langue n’est plus inconnue. On compare les probabilités au-dessus du seuil entre elles, et on garde la plus haute.

Une classe a été développée afin de permettre l’écriture de fichier de bigramme, ou n-gramme, à partir d’un corpus d’entrainement. Le programme créé ensuite un fichier de config à utiliser pour pouvoir calculer les probabilités. Ces nouveaux fichiers générés pourront remplacer les fichiers donnés de base, dont les performances étaient médiocres.

# Modèle n-gramme

Pour ce projet, nous avions le choix d’utiliser le modèle que nous trouvions le plus performant. Nous utilisons donc un modèle unigramme et bigramme, en utilisant un modèle de langage de Laplace.

## Modèle Laplace

Nous avons fait le choix d’utiliser un modèle de Laplace, car dans le cas où un mot n’était pas présent dans le vocabulaire, nous souhaitions quand même pouvoir connaitre la langue de l’ensemble de la phrase.

## Unigramme et Bigramme

L’avantage d’un modèle unigramme contre un modèle bigramme est que dans un ensemble de mots plus restreints il aura une plus grande chance de trouver la langue d’une phrase, que pour un modèle bigramme ou trigramme.

Cependant, dans un ensemble plus grand, un modèle bigramme sera considéré comme étant plus fiable que l’unigramme.

# Performance

En utilisant le fichier gold-sent.txt, nous obtenons pour les modèles bigramme et unigramme :

- unigramme : 0,993350

- bigramme : 0,917400

# Utilisation

MyLanguageRecognizer1 utilise le modèle bigramme de base, sans détecter les modèles inconnus.

MyLanguageRecognizer2 utilise notre modèle bigramme, en détectant les modèles inconnus.

MyLanguageRecognizer2 utilise notre modèle unigramme, en détectant les modèles inconnus.

Pour les tester nous avons le package langReco.reco avec un test de performance pour chaque classe.

Attention : pour créer les modèles de langage spécifique à notre projet nous avons la classe exécutable LanguageModelEstimator dans la package langReco.train. Il va créer les fichiers de config dans le fichier lm/unigram et les fichiers de n\_grams dans le fichier lm.

Pour les runs : la classe exécutable est runTest dans le package langReco.reco, et les 2 fichiers de texte sont créés dans le fichier lm/runs.