Distanciel modèles de Markov cachés

Félix Jamet

30 avril 2018

Table des matières

Introduction

L'expérience de "Marvin le martien" (Stamp 2018) montre qu'il est possible d'extraire des caractéristiques linguistiques à partir d'un corpus de petite taille (50000 caractères), et ce sans connaissances préalables sur la langue du corpus. Cette expérience consiste à entrainer des modèles de Markov cachés comprenant deux états sur un corpus en anglais. Les deux états ainsi formés regroupent d'une part les consonnes et d'autre part les voyelles.

Nous allons dans un premier temps reproduire cette expérience sur le corpus évoqué dans (Stamp 2018). Nous allons par la suite mener une expérience similaire sur un corpus en langue française. Finalement cette même expérience sera faite sur un corpus finnois.

Le choix a été fait d'implémenter en python l'algorithme d'entrainement d'un modèle de Markov caché. Cette implémentation est disponible sur github à l'adresse suivante : `https://github.com/mooss/ruskea`.

1 Reproduction de l'expérience sur le *Brown corpus*

Cette section décrit la démarche entreprise pour reproduire l'expérience de "Marvin le martien" sur le *Brown corpus*.

Dans un premier temps la méthode utilisée pour extraire les 50000 premiers caractères du *Brown corpus* va être détaillée. Ensuite, une reproduction de l'expérience va être effectuée en utilisant les matrices proposées dans (Stamp 2018). Finalement, l'expérience va être reproduite en initialisant les matrices procéduralement.

1.1 Extraction des 50 000 premiers caractères du *Brown corpus*

La première étape est d'extraire les caractères nous intéressant depuis le *Brown corpus*. En effet, le *Brown corpus* est distribué avec les annotations intégrées et il faut donc les supprimer.

La bibliothèque NTLK est utilisée pour télécharger le corpus. Les 50000 premiers caractères de ce corpus sont écrits dans le fichier `brown50000.txt`, en utilisant le script `brownextract.py` :

```
import nltk
nltk.download('brown')
nltk.download('nonbreaking_prefixes')
nltk.download('perluniprops')
from nltk.corpus import brown
from nltk.tokenize.moses import MosesDetokenizer

mdetok = MosesDetokenizer()

def remove_brown_annotations(sentence):
    return mdetok.detokenize(
        ' '.join(sent).replace('``', '"')\
        .replace("''", '"')\
        .replace('`', "'").split(),
        return_str=True)


maxnbchar = 50000
currentnbchar = 0
charbuffer = []

alphabet = 'abcdefghijklmnopqrstuvwxyz '

for sent in brown.sents():
    for char in remove_brown_annotations(sent):
        if currentnbchar < maxnbchar and char in alphabet:
            charbuffer.append(char)
            currentnbchar += 1

output = 'brown50000.txt'
with open(output, "w") as text_file:
    text_file.write(''.join(charbuffer))
```

1.2 Matrices prédéfinies

Les matrices de transitions, observations, et des états initiaux sont initialisées selon les valeurs fournies dans l'article de Mark Stamp.

```
from numpy import array
from markov import *

marvin_transition = array([[0.47468, 0.52532],
                           [0.51656, 0.48344]])
marvin_observation = array(
    [[0.03688, 0.03735, 0.03408, 0.03455, 0.03828, 0.03782, 0.03922, 0.03688, 0.03408,
     ↪  0.03875, 0.04062, 0.03735, 0.03968, 0.03548, 0.03735, 0.04062, 0.03595, 0.03641,
     ↪  0.03408, 0.04062, 0.03548, 0.03922, 0.04062, 0.03455, 0.03595, 0.03408, 0.03408],
     [0.03397, 0.03909, 0.03537, 0.03537, 0.03909, 0.03583, 0.03630, 0.04048, 0.03537,
      ↪  0.03816, 0.03909, 0.03490, 0.03723, 0.03537, 0.03909, 0.03397, 0.03397, 0.03816,
      ↪  0.03676, 0.04048, 0.03443, 0.03537, 0.03955, 0.03816, 0.03723, 0.03769, 0.03955]]
)
marvin_initial = array([[0.51316, 0.48684]])

with open('brown50000.txt', 'r') as brownfile:
    corpus = brownfile.read().replace('\n', '')

alphabet = 'abcdefghijklmnopqrstuvwxyz '

model = markovmodel(marvin_transition, marvin_observation, marvin_initial, rel_tol=1e-3)

train_markov_model(model,
                   list(map_el_to_int(corpus, alphabet)),
                   max_iterations=100)
```

Une tolérence relative de $1e^{-3}$ est appliquée à la construction du modèle pour vérifier la stochasticité des matrice. Sans cette tolérance élevé, l'étape de vérification de la stochasticité des matrices échoue.

Les valeurs ont été copiées telles quelles depuis l'article. Il est probables que ces valeurs aient été arrondies par l'auteur de l'article, d'où la nécessité d'assouplir le test de stochasticité.

1.2.1 Résultats

Les résultats sont synthétisés dans les tables 1.1 et 1.2.

La table 1.1 correspond à la transposée de la matrice d'observation du modèle de Markov entrainé, avec comme information supplémentaire en première colonne le caractère auquel correspondent les probabilités d'apparition des colonnes suivantes. Les probabilités d'apparition sont rapportées sous forme de pourcentages, afin d'être plus lisibles.

La table 1.2 regroupe les caractères selon l'état pour lequel il ont la plus grande probabilité d'apparition. Les résultats des autres expériences seront également présentés sous cette forme.

Les résultats obtenus sont similaires à ceux présentés dans (Stamp 2018), à savoir les voyelles (moins y) d'un côté et les consonnes (plus y) de l'autre.

Bien que la tendance générale soit la même que dans l'article original, les valeurs des matrices de probabilité d'observation diffèrent[1]. L'article n'explicite pas quelle démarche a été utilisée pour extraire les premiers caractères du *Brown corpus*. La démarche utilisée ici repose sur la supposition que la méthode `nltk.corpus.brown.sents()` parcours les phrases du *Brown corpus* de la même manière que dans l'article.

1. Les probabilités d'observation de l'article original sont visibles dans la table 1.3.

Il est probable que les méthodes utilisées diffèrent, résultant ainsi en une chaine d'observation et en un modèle différents.

Caractère	État 1 (%)	État 2 (%)
a	0.050	13.487
b	2.231	0.000
c	5.362	0.022
d	7.025	0.000
e	0.267	21.161
f	3.609	0.000
g	2.637	0.204
h	5.415	2.016
i	0.000	12.127
j	0.238	0.000
k	0.518	0.288
l	7.328	0.328
m	3.917	0.000
n	12.077	0.000
o	0.013	13.061
p	3.513	0.122
q	0.162	0.000
r	10.606	0.000
s	11.239	0.033
t	15.192	0.659
u	0.000	4.418
v	1.718	0.000
w	2.255	0.000
x	0.477	0.000
y	2.643	0.115
z	0.121	0.000
␣	1.385	31.959

TABLE 1.1 – Probabilités d'observation - *Brown corpus* - Matrices pré-initialisées

Groupe 1	Groupe 2
b c d f g h j k l m n p q r s t v w x y z	a e i o u ␣

TABLE 1.2 – Groupes formés - *Brown corpus* - Matrices pré-initialisées

1.3 Matrices générées procéduralement

L'article de Mark Stamp ne fournit que peu de détails concernant l'initialisation des matrices de transition, d'observation et de répartition initiale des états. En effet, la seule indication donnée est d'initialiser les éléments de chaque ligne à environ $1/X$, X étant le nombre d'éléments dans la ligne.

Le probème est qu'en utilisant des matrices d'une forme similaire à celle proposée dans l'article, les résultats sont susceptible de différer grandement.

Après beaucoup d'essais infructeux, une solution satisfaisante a été trouvée ; elle consiste à initialiser les matrices à $1/X$ et parcourir chacune des lignes en y ajoutant ou en retranchant un nombre aléatoire entre a et b, dont les valeurs sont dans la table 1.4. Les lignes ainsi crées sont ensuite rendues stochastiques en multipliant chacune de leurs cases par une constante N telle que :

$$N = \frac{1}{\sum\limits_{el \in ligne} el}$$

Caractère	État 1 (%)	État 2 (%)
a	13.845	0.075
b	0.000	2.311
c	0.062	5.614
d	0.000	6.937
e	21.404	0.000
f	0.000	3.559
g	0.081	2.724
h	0.066	7.278
i	12.275	0.000
j	0.000	0.365
k	0.182	0.703
l	0.049	7.231
m	0.000	3.889
n	0.000	11.461
o	13.156	0.000
p	0.040	3.674
q	0.000	0.153
r	0.000	10.225
s	0.000	11.042
t	1.102	14.392
u	4.508	0.000
v	0.000	1.621
w	0.000	2.303
x	0.000	0.447
y	0.019	2.587
z	0.000	0.110
␣	33.211	1.298

TABLE 1.3 – Probabilités d'observation (Stamp 2018)

Matrice	a	b
Transitions	0.000	0.005
Observations	0.02	0.025
État initial	0.001	0.005

TABLE 1.4 – Bornes aléatoires

Cette solution est implémentée dans la méthode `markovmodel.fromscratch`, visible dans le fichier `markov.py`.

```
from itertools import islice
from markov import *

with open('brown50000.txt', 'r') as brownfile:
    corpus = brownfile.read().replace('\n', '')

alphabet = 'abcdefghijklmnopqrstuvwxyz '

observations = list(islice(
    map_el_to_int(corpus, alphabet),
    0, 50000))

model = markovmodel.fromscratch(2, len(alphabet))
train_markov_model(model, observations, 100)
```

1.3.1 Résultats

Des résultats similaires à ceux de l'expérience originale ont été retrouvés dans 45 des 50 exécutions du script ci-dessus. Les tables 1.5 et 1.6 montrent le résultat d'une de ces exécutions.

Caractère	État 1 (%)	État 2 (%)
a	0.000	14.258
b	2.121	0.000
c	5.097	0.024
d	6.679	0.000
e	0.000	22.566
f	3.432	0.000
g	2.427	0.303
h	7.052	0.059
i	0.000	12.774
j	0.227	0.000
k	0.513	0.282
l	7.283	0.004
m	3.724	0.000
n	11.483	0.000
o	0.000	13.771
p	3.381	0.084
q	0.154	0.000
r	10.084	0.000
s	10.718	0.000
t	13.550	1.665
u	0.000	4.654
v	1.633	0.000
w	2.144	0.000
x	0.453	0.000
y	2.081	0.590
z	0.115	0.000
␣	5.647	28.966

TABLE 1.5 – Probabilités d'observation - *Brown corpus* - Matrices initialisées procéduralement

Groupe 1	Groupe 2
b c d f g h j k l m n p q r s t v w x y z	a e i o u ␣

TABLE 1.6 – Groupes formés - *Brown corpus* - Matrices initialisées procéduralement

2 Expérience sur un corpus français

Cette section s'appuie sur un corpus contenant des articles du journal l'Est Républicain, publiés en 1999. Le corpus est disponible à l'adresse suivante : `http://www.cnrtl.fr/corpus/estrepublicain/`.

2.1 Extraction du texte

Les articles sont contenus dans des fichiers `XML`. Le script suivant est utilisé pour récupérer le texte des articles en ignorant le balisage.

Le texte ainsi extrait est sauvegardé dans le fichier `1999-05-17.txt`.

L'alphabet utilisé correspond à celui décrit dans Wikipédia (`https://fr.wikipedia.org/wiki/Alphabet_fran%C3%A7ais`), soit les 26 lettres fondamentales, les 13 voyelles accentuées (àâéèêëîïôùûüÿ), les deux ligatures (œæ), et le c cédille. L'espace se rajoute à ces 42 lettres.

```
import xml.etree.ElementTree as ET
from itertools import chain

root = ET.parse('1999-05-17.xml').getroot()
articles = root.findall('./tei:text/tei:body/tei:div/tei:div/',
                        {'tei': 'http://www.tei-c.org/ns/1.0'})

alphabet = ' aàâæbcçdeéèêëfghiîïjklmnoôœpqrstuùûüvwxyÿz'
# print(list(root))
# print(articles)

def filterspaces(iterable):
    prevwasspace = True
    for char in iterable:
        if char == ' ':
            if not prevwasspace:
                prevwasspace = True
                yield char
        else:
            yield char
            prevwasspace = False


charbuffer = (char
              for article in articles
              for paragraph in article.itertext()
              for char in paragraph.lower()
              if char in alphabet)

with open('1999-05-17.txt', 'w') as output:
    output.write(''.join(filterspaces(charbuffer)))
```

Cette approche a ses limites, par exemple, il y a beaucoup de `h` isolés à cause de la notation des heures (exemple : de 20h à 20h30). Par ailleurs la suppression de certain caractères spéciaux mène à des juxtapositions indésirables (exemple : saint-mihiel → saintmihiel, l'heure → lheure).

Il serait possible de créer des règles pour traiter ces cas particuliers. Cependant, ils semblent être statistiquement insignifiants, c'est pourquoi le choix a été fait de ne pas s'en soucier.

2.2 Analyse du texte

Que ce soit à cause de la taille accrue de l'alphabet, ou des particularités du corpus, les résultats sont moins cohérents au fil des exécutions que lors de l'expérience sur le *Brown corpus*.

Une nouvelle approche s'est donc imposée. En plus du système d'initialisation de matrices précédemment décrit, N modèles sont entrainés M fois et le meilleur d'entre eux est selectionné pour continuer l'entrainement jusquà 100. On a ici $N = 3$ et $M = 8$.

Cette approche n'est cependant pas suffisante, les résultats de plusieurs exécutions restent grandement différents. Pour être sûr d'obtenir de bons résultats, le script est exécuté 50 fois et le meilleur modèle est conservé, c'est à dire celui maximisant la probabilité de la chaine d'observation.

```
from itertools import islice
from markov import *

with open('1999-05-17.txt', 'r') as repfile:
    corpus = repfile.read().replace('\n', '')

alphabet = 'aàâæbcçdeéèêëfghiîïjklmnoôœpqrstuùûüvwxyÿz '

observations = list(islice(
    map_el_to_int(corpus, alphabet),
    0, 50000))

model = train_best_markov_model(
    2, len(alphabet),
    observations,
    nb_candidates=3,
    train_iter=8,
    max_iter=100)
```

2.2.1 Résultats

La table 2.2 indique les groupes formés par le modèle. Les caractères æïüÿ n'étaient pas présents dans le corpus. Les deux groupes formés sont clairement les voyelles et les consonnes, à l'exception des caractères âëù qui ont été classés dans le même groupe que les consonnes.

Un autre résultat remarquable est que comme pour l'anglais, le y est classifié avec les consonnes. Cependant, les probabilités d'observations dans l'état consonne et dans l'état voyelle pour le y sont plus proches en français qu'en anglais [1].

1. Comparer les tables 2.1 et 1.5.

Caractère	État 1 (%)	État 2 (%)
a	12.501	0.000
à	0.000	1.261
â	0.088	0.000
æ	0.000	0.000
b	0.118	1.825
c	0.340	6.146
ç	0.000	0.083
d	0.000	7.753
e	21.872	0.000
é	4.179	0.000
è	0.560	0.000
ê	0.214	0.000
ë	0.000	0.013
f	0.006	2.195
g	0.000	2.018
h	0.000	2.347
i	10.730	0.000
î	0.044	0.000
ï	0.000	0.000
j	0.000	0.687
k	0.000	0.091
l	0.000	10.805
m	0.000	4.943
n	0.000	12.994
o	8.097	0.000
ô	0.085	0.000
œ	0.042	0.020
p	0.371	4.843
q	0.006	1.184
r	0.000	12.525
s	0.000	13.594
t	1.434	10.615
u	8.126	0.053
ù	0.000	0.048
û	0.033	0.000
ü	0.000	0.000
v	0.000	2.474
w	0.000	0.044
x	0.084	1.020
y	0.271	0.304
ÿ	0.000	0.000
z	0.000	0.114
␣	30.801	0.000

TABLE 2.1 – Probabilités d'observation - Est républicain - Alphabet complet

Groupe 1	Groupe 2	Hors groupes
a â e é è ê i î o ô œ u û ␣	à b c ç d ë f g h j k l m n p q r s t ù v w x y z	æ ï ü ÿ

TABLE 2.2 – Groupes formés - Est républicain - Alphabet complet

3 Expérience sur un corpus finnois

Le corpus va être extrait du dernier *dump* du wikipédia finnois (`https://dumps.wikimedia.org/fiwiki/latest/fiwiki-latest-pages-articles.xml.bz2`), à l'aide de la bibliothèque gensim.

3.1 Extraction du texte

Le script suivant (`make_wiki_corpus.py`), partiellement recopié depuis `https://www.kdnuggets.com/2017/11/building-wikipedia-text-corpus-nlp.html`, est utilisé pour transformer le *dump* wikipédia en un fichier texte.

Le script original a été modifié afin de n'extraire que les N premiers caractères.

```
"""
Creates a corpus from Wikipedia dump file.
Inspired by:
https://github.com/panyang/Wikipedia_Word2vec/blob/master/v1/process_wiki.py
"""

import sys
from itertools import chain
from gensim.corpora import WikiCorpus

def get_n_chars(wiki, n):
    nbchar = 0
    charbuffer = []
    for text in wiki.get_texts():
        for word in text:
            for char in chain(word, ' '):
                if nbchar < n:
                    nbchar += 1
                    charbuffer.append(char)
                else:
                    return charbuffer
    return charbuffer


def make_corpus(in_f, out_f, maxchar):
    wiki = WikiCorpus(in_f)
    charbuffer = get_n_chars(wiki, maxchar)
    with open(out_f, 'w') as output:
        output.write(''.join(charbuffer))


if __name__ == '__main__':

    if len(sys.argv) != 4:
        print('Usage: python make_wiki_corpus.py <wikipedia_dump_file>
        ↪  <processed_text_file> <number_of_characters>')
        sys.exit(1)
    in_f = sys.argv[1]
    out_f = sys.argv[2]
```

```
    maxchar = int(sys.argv[3])
    make_corpus(in_f, out_f, maxchar)
```

Les 50000 premiers caractères du texte sont extraits dans le fichier `wiki_fi_50000.txt`.

```
./make_wiki_corpus.py fiwiki-latest-pages-articles.xml.bz2 wiki_fi_50000.txt 50000
```

3.2 Analyse du texte

L'approche adoptée est la même que celle utilisée pour le corpus français ; N modèles sont entrainés M fois, le meilleur est par la suite entrainé $100 - N$ fois. Ce processus est répété 50 fois, et les résultats du meilleur modèle sont conservés.

```
from itertools import islice
from markov import *

with open('wiki_fi_50000.txt', 'r') as suofile:
    corpus = suofile.read().replace('\n', '')

alphabet = 'aääbcdefghijklmnoöpqrstuvwxyz '

observations = list(islice(
    map_el_to_int(corpus, alphabet),
    0, 50000))

model = train_best_markov_model(
    2, len(alphabet),
    observations,
    nb_candidates=3,
    train_iter=8,
    max_iter=100)
```

3.2.1 Résultats

L'alphabet finnois est composé des 26 caractères de l'alphabet latin, et de trois voyelles supplémentaires (äåö). Parmi les 26 caractères latins, certains sont majoritairement utilisés dans des mots d'emprunt (`bcfgqwxz`). De plus, `å` est un emprunt de l'alphabet suédois et n'est présent que dans des noms propres.

Les groupes formés dans la table 3.2 sont les consonnes et les voyelles. Une analyse de la table 3.1 suggère que les distinctions entre les consonnes et les voyelles sont fortement prononcées, hormis pour les lettres `c`, `g` et `s`.

Caractère	État 1 (%)	État 2 (%)
a	21.880	0.050
ä	4.969	0.007
å	0.000	0.000
b	0.000	0.737
c	0.165	0.200
d	0.000	2.595
e	13.655	0.000
f	0.003	0.355
g	0.189	0.694
h	0.000	2.929
i	17.054	0.000
j	0.000	4.033
k	0.000	9.685
l	0.000	10.978
m	0.000	6.286
n	0.000	15.539
o	9.861	0.000
ö	0.800	0.000
p	0.000	3.287
q	0.000	0.004
r	0.000	5.565
s	0.289	12.416
t	0.000	15.069
u	10.072	0.000
v	0.000	3.740
w	0.000	0.173
x	0.000	0.045
y	2.697	0.000
z	0.000	0.115
␣	18.365	5.495

TABLE 3.1 – Probabilités d'observation - Wikipédia finnois

Groupe 1	Groupe 2	Hors groupes
a ä e i o ö u y ␣	b c d f g h j k l m n p q r s t v w x z	å

TABLE 3.2 – Groupes formés - Wikipédia finnois

4 Discussion

Il semble y avoir en anglais, comme en français et en finnois, une distinction statistique entre consonnes et voyelles, distinction pouvant être mise en évidence à l'aide de modèles de Markov.

Ainsi, ces trois langages montrent une relation entre les groupes formés par des modèles de Markov et des caractéristiques linguistiques. En faisant l'expérience avec un vaste échantillon de langues écrites, il serait possible de constater empiriquement si cette relation est une propriété fondamentale des languages naturels.

Il serait également intéressant de répéter cette expérience en utilisant des modèles de Markov à trois, quatre - ou plus - états. L'expérience présentée ne considérait que des caractères individuels ; un autre paramètre à faire varier est le nombre de caractères constituant une observation. Ces variantes nécessiteraient une plus grande puissance de calcul. Ainsi, le temps d'apprentissage pourrait devenir un problème. Il serait mieux pour cela d'utiliser une bibliothèque externe plutôt que l'implémentation naïve utilisée ici.

Sources

Stamp, Mark. (2018). A Revealing Introduction to Hidden Markov Models. https://www.cs.sjsu.edu/~stamp/RUA/HMM.pdf.