# TP4 : Génération de texte avec des chaînes de Markov

Rapport de TP réalisé par **Viet Nguyen -- 20006303**.

## Exercice 0 : préparer les données

Dans cet exercice, j'ai utilisé le fichier `proust/3000-0.txt` pour couper le texte en phrases et en mots. J'ai également supprimé les caractères de ponctuation et les espaces vides. Le code dans le fichier `exo_0.py` et `exo_0.ipynb` a été exécuté avec succès, et le résultat est le suivant :

```
$ python3 exo_0.py
[['\ufeffMa', 'mère,', 'quand', 'il', 'fut', 'question', "d'avoir", 'pour',
'la', 'première', 'fois', 'M.', 'de'], ['Norpois', 'à', 'dîner,', 'ayant',
'exprimé', 'le', 'regret', 'que', 'le', 'Professeur', 'Cottard', 'fût'],
['en', 'voyage', 'et', "qu'elle-même", 'eût', 'entièrement', 'cessé', 'de',
'fréquenter', 'Swann,'], ['car', "l'un", 'et', "l'autre", 'eussent',
'sans', 'doute', 'intéressé', "l'ancien", 'ambassadeur,'], ['mon', 'père',
'répondit', "qu'un", 'convive', 'éminent,', 'un', 'savant', 'illustre,',
'comme'], ['Cottard,', 'ne', 'pouvait', 'jamais', 'mal', 'faire', 'dans',
'un', 'dîner,', 'mais', 'que', 'Swann,'], ['avec', 'son', 'ostentation,',
'avec', 'sa', 'manière', 'de', 'crier', 'sur', 'les', 'toits', 'ses'],
['moindres', 'relations,', 'était', 'un', 'vulgaire', 'esbrouffeur', 'que',
'le', 'marquis', 'de'], ['Norpois', 'eût', 'sans', 'doute', 'trouvé',
'selon', 'son', 'expression,', '«puant».', 'Or', 'cette'], ['réponse',
'de', 'mon', 'père', 'demande', 'quelques', 'mots', "d'explication,",
'certaines'], ['personnes', 'se', 'souvenant', 'peut-être', "d'un",
'Cottard', 'bien', 'médiocre', 'et', "d'un"], ['Swann', 'poussant',
"jusqu'à", 'la', 'plus', 'extrême', 'délicatesse,', 'en', 'matière'],
['mondaine,', 'la', 'modestie', 'et', 'la', 'discrétion.', 'Mais', 'pour',
'ce', 'qui', 'regarde'], ['celui-ci,', 'il', 'était', 'arrivé', "qu'au",
'«fils', 'Swann»', 'et', 'aussi', 'au', 'Swann', 'du'], ['Jockey,',
"l'ancien", 'ami', 'de', 'mes', 'parents', 'avait', 'ajouté', 'une',
'personnalité'], ['nouvelle', '(et', 'qui', 'ne', 'devait', 'pas', 'être',
'la', 'dernière),', 'celle', 'de', 'mari'], ["d'Odette.", 'Adaptant',
'aux', 'humbles', 'ambitions', 'de', 'cette', 'femme,', "l'instinct,"],
['le', 'désir,', "l'industrie,", "qu'il", 'avait', 'toujours', 'eus,',
'il', "s'était", 'ingénié', 'à'], ['se', 'bâtir,', 'fort', 'au-dessous',
'de', "l'ancienne,", 'une', 'position', 'nouvelle', 'et'], ['appropriée',
'à', 'la', 'compagne', 'qui', "l'occuperait", 'avec', 'lui.', 'Or', 'il',
"s'y", 'montrait']]
```

## Exercice 1 : chaîne de Markov de premier ordre (unigrammes)

Le code est dans le fichier `exo_1.py` et `exo_1.ipynb`.

Suite à l'implémentation de mon programme de génération de texte basé sur un modèle de chaîne de Markov, j'ai effectué des tests pour évaluer sa capacité à produire des textes variés et cohérents. Les

résultats obtenus révèlent plusieurs points clés concernant la performance et les domaines potentiels d'amélioration de mon programme.

**Diversité du Texte Généré**: Mon modèle a démontré une capacité remarquable à générer une variété de textes à partir d'un même token de départ. Cette diversité reflète la compétence du modèle à exploiter la structure et les relations entre les mots apprises du corpus. C'est un indicateur positif de la flexibilité du modèle dans la création de contenu varié.

**Cohérence et Fluidité**: Bien que le modèle soit capable de générer des phrases contenant des idées et des phrases relativement liées, la cohérence globale et la fluidité des textes peuvent être améliorées. La nature du modèle Markov, qui prend en compte uniquement le mot actuel pour prédire le suivant, limite sa capacité à construire des textes hautement cohérents sur des séquences plus longues. L'intégration de modèles plus complexes, tels que les bigrams ou les trigrams, pourrait améliorer cette situation.

**Répétitions**: Certains textes générés présentent des répétitions de phrases ou d'idées, ce qui peut diminuer l'intérêt et la qualité du texte produit. Il s'agit d'un aspect que je pourrais optimiser, soit en affinant le modèle, soit en introduisant des techniques de post-traitement pour minimiser ces répétitions.

**Richesse Linguistique**: Les textes produits témoignent d'une richesse linguistique et d'un vocabulaire varié, illustrant la capacité du modèle à reproduire une gamme étendue d'expressions langagières apprises du corpus. Cette caractéristique est l'un des points forts du modèle, montrant qu'il peut servir de base solide pour la génération de textes créatifs.

==> La programme basé sur le modèle de chaîne de Markov montre un potentiel significatif pour la génération de texte, avec des forces notables dans la diversité et la richesse linguistique du contenu généré. Néanmoins, pour atteindre une plus grande cohérence et fluidité dans les textes et réduire les répétitions indésirables, je prévois d'explorer des améliorations du modèle actuel ainsi que l'application de techniques de post-traitement. L'intégration de modèles linguistiques plus avancés pourrait également contribuer à améliorer la qualité globale des textes produits.

## Exercice 2 : chaîne de Markov d'ordre 2 (bigrammes)

Le resultat de l'exécution du code dans le fichier `exo_2.ipynb` est le suivant :

```python
def generate_bi(markov_chain, start_token):
    phrase = []
    maximum = NB_MOTS_MAXI
    token = start_token
    while len(phrase) < maximum:
        if token in markov_chain:
            next_tokens = markov_chain[token]
            token = max(next_tokens, key=next_tokens.get)
            phrase.append(token)
            if token in ponctuation:
                break
        else:
            break
    return ' '.join(phrase)
```

```python
print(generate_bi(markov_chain, "Si"))
```

```
Swann et de la Berma dans la Berma dans la Berma dans la Berma dans la Berma dans la Berma dans la Berma dans la Berma dans la Berma dans la
Berma dans la Berma dans la Berma dans la Berma dans la Berma dans la Berma dans la Berma dans la Berma dans la Berma dans la Berma dans la B
erma dans la Berma dans la Berma dans la Berma dans la Berma dans la Berma dans la Berma dans la Berma dans la Berma dans la Berma dans la Be
rma dans la Berma dans la Berma dans la
```

```python
import random

def generate_bi(markov_chain, start_token, n_best=1):
    phrase = []
    maximum = NB_MOTS_MAXI
    token = start_token
    while len(phrase) < maximum:
        if token in markov_chain:
            next_tokens = list(markov_chain[token].items())
            next_tokens.sort(key=lambda x: x[1], reverse=True)
            top_tokens = next_tokens[:n_best] if len(next_tokens) > n_best else next_tokens
            token, _ = random.choice(top_tokens)
            phrase.append(token)
            if token in ponctuation:
                break
        else:
            break
    return ' '.join(phrase)
```

```python
print(generate_bi(markov_chain, "Si", 2))
```

```
j'avais été la vie de la vie et de la Berma les plus que le plus que je ne pas de sa mère qui me semblait que je ne pas de la Berma dans le p
lus de la Berma les autres et de la Berma dans la Berma dans le plus que je ne me dit à la Berma les autres par le plus de la vie et qui me d
it à la vie de la Berma les plus de la Berma dans le même pas à la vie de la Berma les plus que le même que je me
```

```python
print(generate_bi(markov_chain, "Si", 10))
```

```
donc en effet, je suis laissé aller pêcher ce qu'on avait dit ma mémoire n'est que ce qu'elle pût avoir connu jadis, j'avoue que nous ne se s
entait que dans son air découragé qui nous ne peut surprendre puisqu'il était pas de mon mari que vous vous a pas de lui avait à ce fut chang
é, et que j'avais pu aller au lieu du mal sans en effet dès qu'ils se faisait maintenant bénéficier Odette qui avait pour une sorte de Norpoi
s est un moment voulu, au lieu précisément le lui et le plus de la femme à un jour
```

- **Sans utiliser** `n_best` : la sortie de la fonction qui n'utilise pas `n_best` entraîne la répétition de la même chaîne de mots "dans la Berma" plusieurs fois, créant ainsi une boucle qui ne se termine que lorsque la limite maximale de mots est atteinte. Cela se produit parce que le modèle choisit le mot suivant en fonction de la probabilité la plus élevée, sans aucune variété ; Si une certaine paire de mots (dans ce cas "la Berma") a une probabilité très élevée par rapport à d'autres paires, le modèle continuera à la choisir à chaque fois, ce qui entraînera une boucle.

- **Utilisation de** `n_best` **= 2** : Lorsque nous utilise `n_best` avec une valeur de 2, les résultats montrent plus de variété dans le choix du mot suivant, avec moins de répétitions et l'apparition de nouvelles séquences de mots. Cependant, il existe encore un taux de répétition important, ce qui montre que même si limiter le choix aux 2 premiers mots peut contribuer à augmenter la variété, cela ne suffit pas à empêcher complètement la répétition si le modèle privilégie toujours certains mots ou expressions.

- **En utilisant** `n_best` **= 10** : Avec une valeur `n_best` de 10, le résultat est une phrase avec une diversité significativement plus élevée que les deux cas précédents. Cela montre que permettre au modèle de considérer un plus grand nombre d'options potentielles pour choisir le mot suivant réduit considérablement la répétition et augmente la probabilité de générer un texte nouveau et plus riche.

En résumé, augmenter `n_best` améliore considérablement la variété et la fraîcheur du texte généré, minimisant ainsi les répétitions indésirables. Cependant, choisir une valeur `n_best` élevée peut également

conduire à choisir des mots moins pertinents, réduisant ainsi la fluidité et la cohérence du texte. Par conséquent, il est nécessaire de trouver un équilibre entre la création d'un texte diversifié et le maintien d'une grande cohésion dans le texte.

## Exercice 3 : chaîne de Markov d'ordre arbitraire

Le code est dans le fichier `exo_3.py` et `exo_3.ipynb`.