# Rapport de Normalisation de Texte – Nombres Cardinaux (0 à 1000)

## 1. Introduction

La normalisation de texte est une étape essentielle dans plusieurs applications de TALN, comme la synthèse vocale ou la reconnaissance automatique de la parole. Elle permet de transformer une forme numérique ou abrégée en sa forme textuelle standard.

L'objectif du projet était de concevoir un système capable de normaliser automatiquement les nombres cardinaux entre 0 et 1000 en français. Ce travail s'inscrit dans le cadre du test de stage proposé par IndabaX.

Le périmètre du projet se limite volontairement aux nombres cardinaux simples, sans traitement des dates, heures, monnaies ou nombres supérieurs à 1000.

## 2. Règles linguistiques d'écriture des nombres cardinaux en français (0 à 1000)

L'écriture des nombres en toutes lettres en français obéit à un ensemble de règles précises, notamment concernant l'usage du trait d'union, la présence de la conjonction *et*, ainsi que l'accord des numéraux *vingt* et *cent*. Les principes ci-dessous résument les conventions utilisées dans la construction de la grammaire de normalisation.

### 1. Usage du trait d'union (Rectifications orthographiques de 1990)

Les rectifications de 1990 ont introduit une règle de simplification importante : **tous les éléments d'un nombre composé doivent être reliés par un trait d'union**. Cette normalisation facilite la lecture et la cohérence des numéros écrits.

### 2. Usage de la conjonction *et*

La conjonction *et* apparaît dans des cas bien précis, souvent hérités d'usages historiques. Elle remplace un trait d'union lorsque l'unité est *un*, ou dans certains cas *onze*.

- **Cas où *et* est utilisé**

La forme *-et-un* est employée pour les nombres terminés par 1 de 21 à 61, ainsi que pour 71.

Exemple : 21 = vingt-et-un ; 71 = soixante-et-onze

- **Cas où *et* n'est pas utilisé**

Dans les nombres reposant sur la base vicésimale (80, 90), la conjonction disparaît au profit du simple trait d'union.

Exemple : 81 = *quatre-vingt-un* ; 91 = *quatre-vingt-onze*

### 3. Règles d'accord (pluriel)

Parmi les numéros, seuls *vingt* et *cent* prennent un *s* au pluriel, et uniquement dans des situations spécifiques. Les autres termes, tels que *mille*, *deux*, *trente*, restent invariables.

- **Accord de *vingt* et *cent***

Ces mots s'accordent lorsque :

1. Ils sont **multipliés** (ex. : quatre-vingts, cinq-cents).
2. Ils sont **en fin de nombre**, c'est-à-dire non suivis d'un autre numéral.

- **Invariabilité de *mille***

Le numéral *mille* est **toujours invariable**, quelle que soit la quantité.

## 2. Méthodologie

L'approche retenue repose sur l'utilisation de Transducteurs à États Finis (FST), outils particulièrement adaptés pour modéliser des règles linguistiques déterministes.
J'ai adopté une démarche **bottom-up**, en construisant d'abord des blocs simples (unités, dizaines, nombres composés, centaines) que j'ai ensuite combinés pour former un FST complet. Le système repose sur une **architecture modulaire** :

- Construction du FST pour chaque catégorie de nombres ;
- Union finale du FST pour couvrir l'ensemble 0–1000 ;
- Intégration au sein d'un pipeline Python permettant la normalisation de phrases complètes.

Le projet utilise principalement la bibliothèque Pynini, qui fournit un environnement robuste pour la création et l'optimisation de transducteurs.

La grammaire se structure en plusieurs blocs :

- Unités (0 à 20) ;
- Dizaines régulières (20, 30, 40…) ;
- Nombres composés (21–69, 71–79, 81–89, 91–99) ;
- Centaines (100 à 999) avec gestion du "s" dans "cents" ;
- cas particulier du nombre 1000.

### 2.1 Nombres de 0 à 20
Les unités et les formes de 10 à 19 sont définies comme mappings directs via des FST simples. Les expressions irrégulières comme “onze”, “quinze”, “seize” ont été intégrées individuellement.

### 2.2 Dizaines (20–90)
Les dizaines régulières (vingt, trente, quarante…) utilisent un FST basé sur le premier chiffre, suivi éventuellement d’un effacement du zéro final pour les dizaines exactes.

### 2.3 Nombres composés (21–99)
Les constructions particulières du français ont été intégrées :

- “vingt-et-un” uniquement pour les nombres finissant en 1 jusqu’à 61,
- Structure “soixante-dix”, “quatre-vingt-dix”
- gestion du “s” dans “quatre-vingts” à 80 mais pas dans 81–89.

### 2.4 Centaines (100–999)
Les centaines utilisent un FST combinant :

- Unité des centaines ;
- ‘-’ + ‘cent’ ou ‘cents’ selon le contexte ;
- Un FST secondaire gérant les deux derniers chiffres (00–99).

### 2.5 Le cas particulier de 1000
Le nombre 1000 est directement mappé à “mille”, sans unité supplémentaire.

### 2.6 Gestion du contexte et intégration dans les phrases
**Préservation du contexte :**

Le système doit normaliser les nombres tout en préservant la structure de la phrase :

- Les espaces avant et après les nombres
- La ponctuation adjacente
- La casse (majuscules en début de phrase)

**Stratégie d'implémentation :**

L’intégration du FST dans des phrases complètes repose sur un mécanisme simple et robuste articulé autour de trois étapes : détection, validation et substitution.

- **Détection des nombres dans le texte**

Une expression régulière (\b\d+\b) permet d’identifier uniquement les séquences numériques autonomes.
Ce choix évite d’interpréter à tort des chiffres intégrés dans des mots ou des

identifiants, et garantit que seules les valeurs réellement perçues comme des nombres cardinaux sont traitées.

- **Vérification du périmètre (0–1000)**

Chaque nombre détecté est converti en entier pour vérifier qu'il appartient bien au domaine géré par le FST.
Les valeurs hors plage, non valides ou non convertibles sont laissées inchangées, ce qui empêche les erreurs de normalisation et préserve l'intégrité du texte.

- **Application contrôlée du FST**

La normalisation d'un nombre isolé est effectuée via apply_fst(), protégée dans un bloc try/except.
Ainsi, si le FST échoue ou rencontre un cas non couvert, le système renvoie simplement le nombre original, assurant une stabilité totale.

- **Substitution locale et non intrusive**

La fonction re.sub() remplace uniquement les occurrences validées, sans altérer le reste de la phrase.
Cette approche garantit une normalisation ciblée, cohérente et respectueuse du contexte grammatical.

## 3. Implémentation

Le développement Python se structure autour de quatre scripts :

- **Text_Normalisation_Cardinaux_0_a_1000.py** : construction de la grammaire FST.
- **script_sauvegarde.py** : compilation du FST dans un fichier FAR.
- **script.py** : normalisation d'une phrase ou session interactive.
- **script_wer.py** : calcul du Word Error Rate (WER).

La construction de la grammaire utilise plusieurs unions successives de FST optimisés. Le fichier FAR est ensuite généré pour permettre un chargement rapide.

Les cas limites (zéros non significatifs, nombres >1000, erreurs de parsing) ont été gérés via conditions et expressions régulières.

## 4. Résultats et performances

4.1 Tests unitaires

Le système a été rigoureusement testé à l'aide des tests unitaires fournis sur Hugging Face. Ces tests couvrent l'ensemble du périmètre des nombres cardinaux de 0 à 1000.

| | input | reference | hyp | wer |
|---|---|---|---|---|
| 0 | 0 | zéro | zéro | 0.000000 |
| 1 | 6 | six | six | 0.000000 |
| 2 | 11 | onze | onze | 0.000000 |
| 3 | 20 | vingt | vingt | 0.000000 |
| 4 | 71 | soixante et onze | soixante-et-onze | 0.666667 |
| 5 | 83 | quatre-vingt-trois | quatre-vingt-trois | 0.000000 |
| 6 | 94 | quatre-vingt-quatorze | quatre-vingt-quatorze | 0.000000 |
| 7 | 100 | cent | cent | 0.000000 |
| 8 | 181 | cent quatre-vingt-un | cent-quatre-vingt-un | 0.166667 |

Soit Un score WER sur l'ensemble de test de **9% qui s'explique simplement par le fait que sur certaines références, la règles d'écritures (*manque de – entre les noms en lettre*) n'ai pas respecté**

La compilation du FST dans un fichier FAR est **rapide** (**moins d'une seconde sur machine standard**).

La vitesse d'exécution en normalisation est très élevée grâce à **shortestpath** et l'optimisation du FST.

Le WER estimé sur un dataset de 133 phrases présente dans le dossier data/ est de :

- **WER moyen: 0.71%**
- **WER min: 0%**
- **WER max: 25%**

**Ce score valide la fiabilité générale du système.**

### 5. Instructions d'utilisation

Après installation des dépendances via requirements.txt :

- Générer le fichier FAR :

  *python script_sauvegarde.py*

- Normaliser un texte :

  *python script.py "J'ai 25 ans"*

- Normaliser en mode interactif :

  *python script.py*

- Calculer le WER :

  *python script_wer.py data/dataset.csv*

NB : La génération du fichier Far est effectuer si le fichier Far est absent.

## 6. Conclusion

Ce projet a permis de développer un système de normalisation précis et performant couvrant les nombres cardinaux entre 0 et 1000.

Les principales limitations concernent :

- La non-prise en charge des nombres supérieurs à 1000 ;
- L'absence de traitement des dates, heures et montants.

Les perspectives d'amélioration incluent :

- L'extension à d'autres catégories (ordinaux, dates, monnaies) ;
- L'intégration dans un pipeline plus large de normalisation de texte.

## 7. Auteur

**Foka Maghen Yann Brondon**

**Etudiant en Master 2 recherche en Informatique option Intelligence Artificielle à l'Université de Douala.**

Ce projet de normalisation de nombres cardinaux en français (0 à 1000) a été développé dans le cadre d'un test pratique pour un stage pratique chez **IndabaX**.

Github du projet : yann214/Test-stage_indabax