Rapport final
PROJET TAL
Groupe A

Ninoh AGOSTINHO DA SILVA, Ahmed BEJI, Arno WATIEZ

23 Juin 2020

Résumé

Dans le cadre du projet en Traitement Automatique des Langues, nous sommes en train développer un programme répondant au sujet "Saisie prédictive pour téléphone portable". Nous avons choisi d'utiliser pour ce travail le langage de programmation Java. Ce document a pour but de présenter notre sujet, les fonctionnalités attendues, ainsi que de faire le point sur ce que nous avons fait jusqu'à présent et comment nous envisageons la suite.

1 Présentation du sujet

La saisie prédictive a dans premier temps éte conçue comme une technologie de communication avec les sourds [1], jusqu'à l'apparition des téléphones mobiles.
Aujourd'hui, il est difficile de trouver quelqu'un qui n'utilise pas au quotidien d'outil intégrant une forme de saisie prédictive, que ce soit dans un moteur de recherche, un éditeur de texte ou bien une application de messagerie quelconque. La finalité commune à ces outils est d'améliorer l'expérience utilisateur en accélérant la saisie et/ou en évitant sa répetition lorsqu'il s'agit de motifs fréquents. Le programme qui nous a été demandé devra répondre au contrat suivant :

- Réflexion sur le concept de mot et plus généralement sur la façon de diviser les textes sur lesquels nous allons nous appuyer.
- Émulation d'une interface graphique de smartphone avec un clavier virtuel permettant à l'utilisateur de saisir du texte.
- Propositions de mots actualisées en fonction de la saisie :
  1. **Complétion** : Proposition, au fur et à mesure de la saisie, de mots dont cette dernière est un préfixe.
  2. **Prédiction** : Proposition, après chaque mot saisi, de mots fréquemments retrouvés dans cete configuration.

— **Mise à jour utilisateur** : Le programme devra s'adapter pour refléter au mieux le style de saisie de l'utilisateur et éventuellement permettre la création de plusieurs profils personnalisés.
— **Optimisation** : Le programme devra utiliser les structures estimées les moins coûteuses en temps et en espace pour être le plus léger possible et permettre de retrouver la fluidité d'éxécution des systèmes comparables intégrés aux outils du grand public.
— **Evaluation numérique de la performance** : Le programme devra être évalué de façon à avoir des données tangibles et quantitatives de son efficacité.

2 Structures et données

2.1 *Trie* et liste de mots

Le dictionnaire [2] énumère 15612 mots du français, ligne par ligne au format **.txt**. Nous avons commencé par nous servir de cette liste de mots pour tester notre première structure de données, la ***Trie***.

Une ***Trie*** (ou arbre de préfixes) est une variante des arbres de recherche adaptée aux chaînes de caractère. Il s'agit d'une structure descendante de nœuds hiérarchisés ayant des nœuds père et/ou des nœuds fils. Bien que plusieurs choix de modélisation existent pour cette structure, son principe général permet, une fois les données correctement stockées, de les parcourir plus efficacement dans certaines situations que si elles étaient représentées de manière linéaire.
Dans notre cas, une liste de mots est chargée dans la ***Trie*** de cette façon :

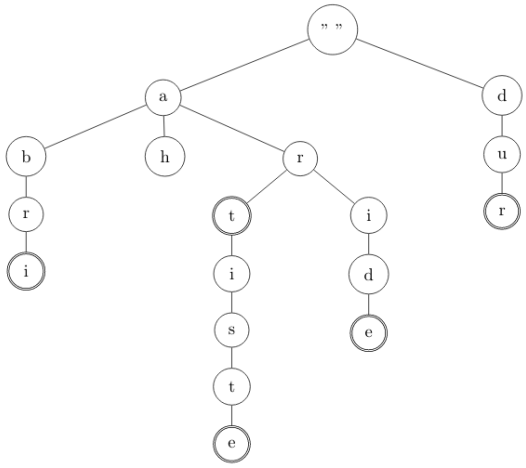

FIGURE 1 – Exemple de ***Trie*** selon le modèle utilisé

Cela permet de rapidement énumérer tous les mots dont la saisie de l'utilisateur est un préfixe. Cependant, une liste de mots n'est pas suffisante pour affiner la sélection du programme. Si les 3 premières suggestions mises en avant respectent simplement l'ordre alphabétique de la liste, il très peu probable que ces mots fassent partie des plus fréquents. Il faut donc ajouter un algorithme de prédiction qui assigne à chaque mot complétant la saisie de l'utilisateur une valeur de probabilité, pour trouver ceux dont cette valeur est la plus élevée.

2.2 Corpus et modèles de langues de Markov en n-grammes

Nous avons choisi une méthode de prédiction, dont les calculs seront développés dans la partie 3, bâtie sur les modèles de langue de Markov d'ordre $k = n-1$, les n-grammes : Le principe du modèle en n-grammes appliqué aux suites de mots est d'assigner à chaque mot une probabilité qui dépend de sa fréquence d'apparition dans un contexte particulier. Dans un modèle en n-grammes, nous estimons la probabilité d'un mot en comptant ses occurrences après la suite exacte des $n-1$ mots qui le précèdent. Même avec un volume très important de données, la probabilité d'un même mot peut aussi varier selon le texte de référence choisi. Nous avons donc choisi et testé 3 corpus fondamentalement différents :

- Le corpus 88milSMS, corpus de 88 000 SMS théoriquement en français, constitué en 2014. [3]
- Le roman "A la recherche du temps perdu" de Marcel Proust. [4]
- Le corpus "The Chambers-Rostand Corpus of Journalistic French"3 composé d'articles parus dans les quotidiens "Le Monde", "L'Humanité" et "La Dépêche du Midi" parus en 2002-2003.

Ces corpus ont été "nettoyés" et segmentés. pour en extraire tous les n-grammes, et leurs occurrences respectives dans une structure Java. Cela représente "l'entraînement" de notre programme. Le nettoyage consiste à s'assurer qu'aucun caractère présent dans le texte ne pourra poser de problèmes à l'éxécution du programme. Pour la segmentation, nous considérons le caractère "espace" comme le délimiteur absolu entre les mots. De cette façon, des chaînes de caractère comme "peut-être" comptent comme un seul mot, tandis que d'autres comme "pomme de terre" et "disque dur" seront divisées même si elles forment une unité sémantique. Le lien entre ces éléments sera rétabli par la taille, la qualité du corpus et l'évolution du programme au fil des utilisations dans une moindre mesure. Idéalement, nous aurions aimé travailler à partir d'un corpus de mails professionnels et/ou formels mais nous n'en avons pas trouvé en français. Après des essais prolongés sur chacun des 3 corpus mentionnés précédemment, nous avons choisi la collection d'articles comme corpus par défaut pour notre programme, car il garantit une orthographe correcte avec des formulations et des sujets plus proches du français qui s'écrit au quotidien que le roman de Marcel Proust. Dorénavant, lorsque nous parlerons de corpus sans qu'il s'agisse d'une variable, nous ferons référence à ce corpus par défaut. L'entraînement du

programme permet donc de créer un lien entre toute suite de k mots, tout mot m qui apparaît au moins une fois après cette suite, et ce nombre d'apparitions que nous appellerons $P_k(m)$.

Afin que nous puissions savoir si un mot apparaît en début ou en fin de phrase, il a été nécessaire de rajouter k termes correspondant au début et à la fin de phrase de part et d'autres de chaque phrase du corpus. Soit C le corpus constitué des deux phrases :

— **Tu es confiné**
— **Tu es sympathique**

Pour un modèle en trigrammes (3-grammes), ce corpus sera enregistré dans notre structure sous la forme suivante :

[{<DEBUT>,<DEBUT>}={Tu=2},
{<DEBUT>,Tu}={es=2},
{Tu,es}={confiné=1,sympathique=1},
{es,confiné}={<FIN>=1},
{es,sympathique}={<FIN>=1},
{confiné,<FIN>}={<FIN>=1},
{sympathique,<FIN>}={<FIN>=1}]

La segmentation du corpus nous permet ensuite, depuis les n-grammes enregistrés dans notre structure, d'extraire les différents mots du corpus et de les ajouter à la ***Trie***, en assignant dans la ***Trie*** à chaque mot sa fréquence d'apparition parmi tous les mots du corpus.

2.3 Personnalisation du programme

Un autre critère important dans notre sujet était l'adaptabilité du programme à son/ses utilisateur(s). Autrement dit, nous devions trouver un moyen d'enregistrer les saisies des utilisateurs lors de chaque utilisation pour améliorer la pertinence des suggestions. Nous y sommes parvenus en plusieurs étapes.

Dans un premier temps nous avons ajouté une fonctionnalité qui permet de créer des profils utilisateurs et de les stocker dans une structure Java. Lorsqu'un profil utilisateur est créé, un fichier au format **..txt** dont le nom permet d'identifier l'utilisateur correspondant est créé, et le contenu du corpus initial est copié dans ce fichier. C'est ce fichier qui sera ensuite utilisé à chaque fois que l'utilisateur se connectera à son profil.

Puis nous avons ajouté un bouton "Send" (Envoyer) qui simule une confirmation de saisie de texte, correspondant par exemple dans le cas d'une application de messagerie à l'envoi d'un message. Lorsque l'utilisateur appuie/clique sur le bouton, le texte entré est "nettoyé" et segmenté de la même manière que le corpus qui sert de base au programme, et les n-grammes et les mots extraits sont

ajoutés respectivement à la liste de n-grammes utilisée par le programme et à la ***Trie***. De plus, si l'utilisateur s'est connecté, le texte entré est ajouté dans son corpus personnel, et conservé.

Finalement, nous avons utilisé la fonctionnalité de "sérialisation", incluse dans le langage Java, pour sauvegarder la liste des utilisateurs entre les sessions. Si cette liste est non vide au lancement du programme, les différents profils s'associent à leur corpus respectif qui pourra à nouveau être enrichi.

Pour plus de détails concernant cette fonctionnalité au niveau de l'expérience utilisateur, se référer au fichier **README**.

3 Mécanisme de complétion/prédiction

La pertinence des prédictions de notre programme est l'enjeu principal de notre sujet. En effet, au delà de sa fluidité, un clavier dit "intelligent" se doit de suggérer des mots que l'utilisateur. Pour cela, nous avons fait le choix d'utiliser les fréquences absolues des mots ainsi que leur nombre d'occurrences dans les différents n-grammes d'un corpus. Comme nous l'avons dit dans la partie 2.2, notre modèle de prédiction s'appuie sur les modèles de Markov, mais le calcul de probabilités a ici été repensé autour de variables et d'opérations différentes qui s'intègrent dans notre conception du sujet.
Posons E l'ensemble des mots du corpus, $n(E)$ le nombre de mots dans cet ensemble (appelé cardinal de E), $f_A(m)$ la fréquence absolue de m dans E et $P_k(m)$ le nombre d'occurrences de m après les $k = n - 1$ mots de la saisie courante.
Dans notre version d'un modèle n-gramme, à chaque actualisation de la saisie de l'utilisateur, tout mot $m \in E$ pouvant être suggéré se voit attribuer un score déterminé de la façon suivante :

$$score(mot) = \frac{f_A(m)}{n(E)} + P_k(m)$$

Ceci est permis par des fonctionnalités de détection automatique de la saisie de l'utilisateur, ainsi que des k mots précédents. Nous mentionnerons aussi ciette saisie sous l'appellation "préfixe". Lorsque l'information est récupérée, on parcourt la ***Trie*** afin de rechercher tous les mots possibles dont le préfixe saisi par l'utilisateur est un préfixe, et on associe chaque mot à sa fréquence d'apparition dans le corpus. Si nous sommes dans un cas où un mot n'est pas en cours de saisie, par exemple après un espace, la ***Trie*** nous indiquera les mots les plus fréquents. Or le parcours intégrale de la ***Trie*** sans partir d'un préfixe quel qu'il soit est une opération coûteuse en temps dont il faut minimiser les répétitions. Pour cette raison, à chaque "envoi" de la part de l'utilisateur, une nouvelle ***Trie*** est créée à partir des n-grammes actualisés, et une liste de tous les mots de cette ***Trie*** avec leur fréquence respective, est crée. Ce procédé nous a permis d'éliminer complètement le délai perceptible jusqu'alors lorsque l'utilisateur voulait écrire un nouveau mot.

Dans un second temps, après avoir accédé aux mots stockés dans la ***Trie*** ainsi que leurs fréquences, le programme regarde dans la liste de n-grammes obtenus avec la segmentation du corpus si les k derniers mots entrés par l'utilisateur ont déjà été utilisés. Si c'est le cas, le programme repère, dans la liste, les mots qui apparaissent dans le corpus à la suite de ces derniers n-1 mots entrés, et ajoute aux fréquences absolues des mots concernés le nombre d'occurrences de ceux-ci au sein après cette suite de mots exacte. C'est ce qui permet au programme, avec l'ajout des k termes en début de phrase expliqué dans la partie 2.2, de donner nativement un meilleur score aux mots les plus utilisés en début de phrase lorsqu'un symbole de ponctuation est détecté dans la zone de saisie. Dans n'importe quelle situation, une liste de mots possibles associés à leur score est créée puis triée dans l'ordre décroissant, afin de suggérer à l'utilisateur les trois mots qui ont le meilleur score dans ce contexte.

La fréquence absolue d'un mot étant représentée par une fraction dont le dénominateur sera toujours supérieur ou égal au numérateur, notre définition du score accordera toujours la priorité à la fréquence contextuelle qui est une valeur entière.

Les 3 mots ayant les fréquences absolues les plus importantes du corpus seront donc proposés lorsque la suite des$k$ derniers mots saisis par l'utilisateur n'est jamais apparue dans tout le corpus.

4 Interface Graphique

En plus de la fiabilité des algorithmes et de la fluidité d'éxécution du programme de manière générale, une interface graphique convaincante était nécessaire pour que l'utilisateur du programme se sente en terrain familier. Nous nous sommes fortement inspirés des claviers des applications de messagerie les plus utilisées, en combinant certaines de leufs contionnalités et en adaptant le desing à notre problème. Le résultat est un clavier réactif, compact et complet pour la saisie du français, dont le fonctionnement est détaillé plus longuement dans notre fichier **README**.

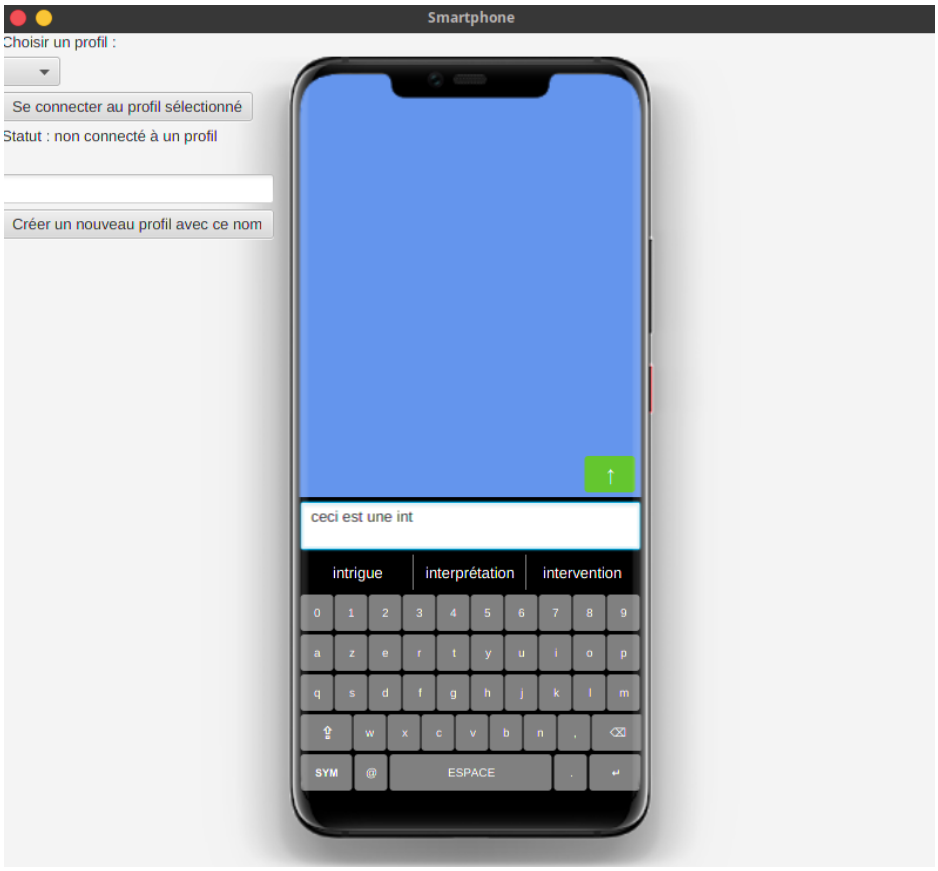

FIGURE 2 – Programme en cours d'utilisation

5 Évaluation numérique de la performance

5.1 Protocole

L'évaluation numérique d'un programme dont la fonction principale est de prédire n'est pas un concept aisé au premier abord. [5] L'objectif général est de confronter le programme à des situations les plus ressemblantes possibles mais à des phases différentes de son évolution afin de comparer les résultats. Une façon de faire est d'entraîner le programme sur une grande partie d'un corpus E, puis d'essayer de lui faire prédire la partie restante de ce même corpus, que nous appelerons référence, en évaluant sa réussite. Si nous considérons naïvement un succès du programme comme une égalité entre un mot prédit et le mot à la même position dans la référence, nous nous apercevons que la nature même du programme pose problème. En effet, en cherchant à maximiser le score des mots,

dans un contexte identique, le même ensemble de 3 mots sera toujours suggéré. Par conséquent, tout premier mot d'une phrase dans le corpus de référence qui ne fait pas partie de l'ensemble des 3 mots les plus fréquents en début de phrase provoquera un échec sur ce mot. Nous choisissons donc de ne pas compter ces comparaisons dans le total. De plus, tout échec sur un mot peut être dû à des facteurs non contrôlables comme des mots non présents dans le corpus d'entraînement. Or, le système de prédiction d'un mot, basé sur la saisie précédente, fait qu'un échec sur un mot conduira probablement à un échec sur le mot suivant. Pour chaque mot à prédire dans le test, nous fournissons donc à notre programme les k mots précédents dans le corpus de référence. Nous pourrons donc évaluer si, dans un contexte rigoureusement identique, il choisit le même mot que la personne ayant produit le corpus de référence, et ce sur l'ensemble des mots de ce corpus. Nous supposons que ce choix permettra d'obtenir un pourcentage de ressemblance avec le corpus de référence plus élevé et plus représentatif de la réalité.
Nous avons tout de même inclus dans les résultats un essai ou nous laissions le programme prédire sans référence.

5.2 Données

Nous avons séparé chacun des 3 corpus mentionnés dans la sous-section 2.2 en 2 parties avec une distribution 80%/20%. Pour simplifier l'affichage dans les résultats, nous donnerons des noms simplifiés à chacun des fichiers. Ces codes seront formés de la lettre S, P ou J (respectivement pour SMS, Proust et Journaux) immédiatement suivie du nombre 80 ou 20 indiquant à quelle partie du corpus le fichier correspond.

5.3 Résultats

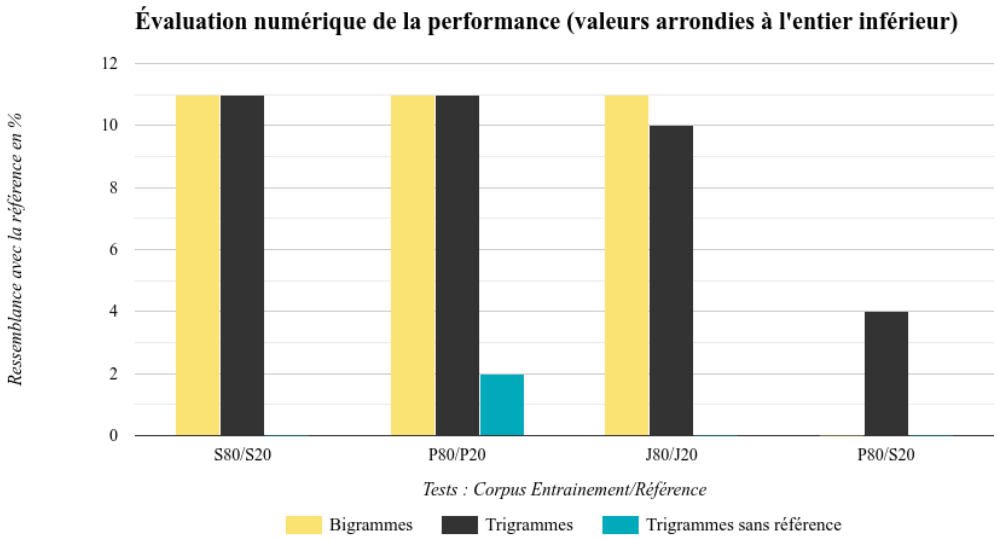

FIGURE 3 – Représentation des résultats

5.4 Interprétation des résultats et conclusion

Nous observons des résultats quasiment identiques entre les différents corpus, avec les modèles en bigrammes qui sont toujours au moins aussi performants que ceux en trigrammes. Bien que les 3 corpus soient de taille importante, ils ne sont pas assez volumineux pour permettre des évaluations pertinentes d'ordre plus élevé plus grands, car en analysant les textes préd. L'essai unique sans référence semble confirmer notre hypothèse de départ.

Nous pouvons dire que la régularité des performances du programme sur 3 corpus très différent semble indiquer une certaine solidité des algorithmes de prédiction. Bien qu'une ressemblance à hauteur de 11% puisse paraître peu élevée, il s'agit quasiment du triple de la valeur obtenue lors que le corpus de référence n'est pas issu du même fichier que le corpus d'entraînement, comme lors du test P80/S20.

Nous pensons que tout programme peut-être amélioré, et dans notre cas, les pistes d'évolution se situent certainement au niveau de la quantité, et le traitement des données plus qu'au niveau du choix. Cependant, nous avons accompli tous nos objectifs. Nous avons proposé une émulation de saisie prédictive comparable à celle qui peut-être trouvée sur n'importe quel smartphone. Notre programme est fluide et agréable à l'utilisation. Il est personnalisable. Enfin, il

remplit les critères de complétion et prédiction et a réussi les tests d'évaluation de la performance que nous avons imaginés.

Références

[1] USPTO. *United States Patent 4,754,474, Feinson June 28, 1988*. URL : http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO2&Sect2=HITOFF&u=%5C%2Fnetahtml%5C%2FPTO%5C%2Fsearch-adv.htm&r=1&p=1&f=G&l=50&d=PTXT&S1=4754474.PN.&OS=PN/4754474&RS=PN/4754474.

[2] FREELANG. *Liste de mots français*. URL : https://www.freelang.com/dictionnaire/dic-francais.php.

[3] Panckhurst R. et al. *88milSMS. A corpus of authentic text messages in French*. 2014. URL : http://88milsms.huma-num.fr/corpus.html.

[4] Marcel PROUST. *À la recherche du temps perdu*. URL : https://alarecherchedutempsperdu.org/.

[5] Chip HUYEN. *Evaluation Metrics for Language Modeling*. URL : https://thegradient.pub/understanding-evaluation-metrics-for-language-models/.