Sorbonne Université
Master ANDROIDE

DISCO: Contextually Improving Command Discoverability

Stage de Master 1

Réalisé par :

Alexandre XIA

Encadré par :
Gilles BAILLY, ISIR, Sorbonne Université
Julien GORI, ISIR, Sorbonne Université

Référent :
Thibaut Lust, LIP6, Sorbonne Université

3 juillet 2023

Table des matières

1 Introduction **1**

2 État de l'art **2**
2.1 Les 4 domaines de l'amélioration des performances d'une interface 2
2.2 Approches de systèmes de recommandation de commandes 3

3 Contribution **4**
3.1 Répartition des rôles et contributions mineures 4
3.1.1 Point de départ 4
3.1.2 Répartition des rôles 6
3.1.3 Logiciel de présentation (Powerpoint) 7
3.1.4 Photoshop 7
3.1.5 Vidéo et exemple de scénario 8
3.2 Éditeur de texte 8
3.2.1 Objectifs et Difficultés 8
3.2.2 Les choix des commandes liés à l'éditeur de texte 9
3.2.3 Gestion de la représentation des états et base de données avec player 10
3.2.4 Modèles de classification 12
3.2.5 Choix du modèle final et résultats 13

4 Conclusion **18**

A Cahier des charges **20**
A.1 Cahier des charges 20
A.1.1 Introduction 20
A.1.2 Les 3 Logiciels 21

Chapitre 1

Introduction

Les applications telles que PowerPoint ou Adobe Photoshop qu'on utilise tous les jours fournissent de nombreuses commandes selon nos besoins. Parmi ces commandes, certaines plus efficaces regroupent d'autres commandes plus simples, ce qui permet d'accélérer une tâche. Par exemple, Adobe Photoshop propose la commande "retirer les yeux rouges" qui permet à l'usager de cliquer à partir de l'outil sur les zones correspondantes et de les repeindre en noir. Un certain nombre d'utilisateurs ne connaisse pas l'existence de cette commande et choisisse l'outil "pinceau" et change sa couleur en noir avant de repeindre manuellement les zones rouges.

Plusieurs méthodes pour recommander des commandes ont été proposées mais une partie suggère que l'utilisateur est déjà familier avec les commandes optimales. Les outils permettant à l'utilisateur de découvrir de nouvelles commandes utilisent en majorité des recommandations selon les actions de l'utilisateur.

Dans la continuité du projet de Master I, nous avons durant le stage essayé d'aboutir à une preuve de concept pour la création d'un modèle de recommandation de commandes. Ce modèle se baserait plus sur l'intention de l'utilisateur en utilisant l'état de l'application (une image ou un vecteur de descripteurs avant et après modification par l'utilisateur) plutôt que les commandes utilisées pour atteindre son objectif.

Pour cela, on a cherché dans un premier temps à avoir des commandes pertinentes et pour lesquelles on a des effets assez différents pour couvrir plus généralement les exemples. Puis, on a défini une représentation de l'état ainsi que coder un outil qui nous permet de générer des données. Enfin, nous avons codé ou entraîné des classifieurs selon leur nature ("hard-codé" ou Machine learning) donnant l'occasion à la fin de suggérer potentiellement une bonne commande selon l'état. Chaque point est réalisé pour 3 logiciels (éditeur de texte, logiciel de présentation type Powerpoint et Photoshop) et cela correspond à l'objectif commun. Le système est nommé selon l'article de nos encadrants "DISCO".

Ce projet est fait en collaboration avec mes camarades : Christian ZHUANG et Nassim AHMED-ALI. Nous sommes encadré par Gilles BAILLY et Julien GORI.

Le lien du dépôt git est ici : NOTRE GITHUB

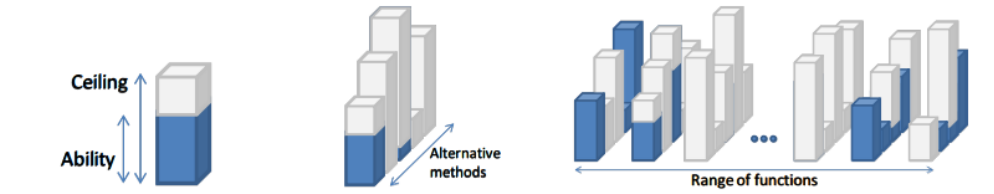
Chapitre 2

État de l'art

2.1 Les 4 domaines de l'amélioration des performances d'une interface

En ce qui concerne le concept d'aide pour les commandes, il existe différentes approches permettant d'améliorer les performances de l'utilisateur avec les interfaces. Ces approches se basent sur 4 points : L'amélioration intramodale (intramodal improvement), l'amélioration intermodale (intermodal improvement), l'extension vocabulaire (vocabulary extension), La planification des tâches (tasks mapping).

Ces notions sont toutes précisées dans l'article Supporting Novice to Expert Transitions in User Interfaces [1]. On définit brièvement l'extension vocabulaire (vocabulary extension) qui correspond aux connaissances de l'utilisateur vis-à-vis des outils du logiciel. C'est le domaine vers lequel notre travail s'oriente car nous souhaitons que l'utilisateur découvre des commandes qui rassemble potentiellement des commandes plus simples. Par exemple, l'utilisateur qui habituellement aligne des objets d'un transparent à la main pourrait utiliser la commande l'alignement en sélectionnant les objets qu'il souhaite aligner. On ne détaillera pas les autres domaines qui ne sont pas vraiment en lien avec notre cas.



(a) Performance characteristics of a method. *Intramodal improvement* concerns the user's performance ability within one method.

(b) Range of methods for a function. *Intermodal improvement* concerns the user's transition to faster methods for activating a function.

(c) Range of functions available. *Vocabulary extension* concerns the user's knowledge and use of the functions available.

FIGURE 2.1 – Représentation des approches amélioration intramodale (a), amélioration intermodale (b) et de l'extension vocabulaire (c) d'après l'article [1]

2.2 Approches de systèmes de recommandation de commandes

L'article "Git command recommendations using crowd-sourced knowledge" [2] présente un système qui recommande à l'utilisateur des commandes git en fonction d'une tâche donnée sous forme d'une requête écrite en langage naturel (son objectif). Ils utilisent une base de données à partir des différents postes sur StackOverflow. Pour la suggestion, il compare l'entrée de l'usager avec les différents mots clés de la base pour donner une liste de commandes candidates à l'utilisateur. Cette approche permet à l'usager de potentiellement découvrir de nouvelles commandes puisqu'il se peut que parmi les commandes candidates, certaines lui sont inconnues. Dans notre approche, on aide l'usager à apprendre plutôt pendant la réalisation d'une tâche alors que dans leur cas, il faut que l'utilisateur fasse une requête avant de commencer la tâche. Md Adnan Alam Khan et Al propose un système similaire qui utilise la documentation web à la place [3]. Cependant, cette méthode de recommandation peut empêcher la mémorisation sur le long terme de ces nouvelles commandes (paradoxe de l'utilisateur guidé) car l'usager peut en devenir dépendant.

Une autre approche assez présente dans la littérature est la recommandation en fonction de commandes de communautés. Le système appelé "CommunityCommands"[4] qui recommande des commandes en comparant l'historique d'un individu à celle de toute une communauté. Emerson Murphy-Hill et Al présente également d'autres façon de choisir les recommandations tel que recommander la commande la plus utilisée par la communauté [5]. Cette méthode dépend fortement de la présence d'une communauté et les suggestions ne seront pas forcément pertinentes pour l'usager en particulier. Le système sur lequel on travaille veut au contraire recommander des commandes pertinentes et ne dépend pas d'une communauté. De plus, il est dépendant du contexte plutôt que de l'historique des commandes.

Pour ce qui concerne le Machine learning, il est assez difficile de trouver de la littérature en rapport avec les systèmes de recommandation de commandes. Samarth Aggarwal et Al propose un "framework" (cadre) pour les systèmes de recommandation de commandes qui dépendent de l'objectif (contexte) [6]. Dans leur cas, c'est plutôt un cas d'apprentissage non-supervisé pour prédire alors que dans notre cas, cela serait supervisé lorsque notre classifieur est entraîné car on inclut un label : la commande. Pour l'éditeur de texte qui est la principale contribution de ce rapport, il existe plusieurs méthodes de traitement de texte (Natural language processing) dans la littérature qu'on pourrait appliquer à notre cas, avec GPT (generative pre-trained transformer) utilisé dans l'article de Aggarwal et Al [6] ou un modèle plus simple comme Bag of word présent dans la bio-informatique [7] mais dont on n'a pas réussi à trouver de cas pour les systèmes de recommandation.

Chapitre 3

Contribution

Dans cette partie nous allons présenter les différents points traités avec nos encadrants pour mener à bien notre projet. Il faut noter que le travail a été fait en collaboration avec des camarades du même Master : Nassim Ahmed-Ali et Christian Zhuang

3.1 Répartition des rôles et contributions mineures

Nous allons présenter un peu plus généralement le travail réalisé dans les autres logiciel du système. Nous nous sommes repartis les tâches de sorte à ce que chaque personne traite un logiciel. Enfin, nous terminerons avec une brève présentation de la tâche finale de notre projet de stage qui est la vidéo faisant démonstration du système.

3.1.1 Point de départ

Puisqu'il s'agit de la suite du projet ANDROÏDE de Master 1, nous allons rapidement résumer ce qui a été fait dans le projet. L'objectif était de concevoir un assistant pour un logiciel simplifié. La figure suivante montre une version de ce qui a été fait :

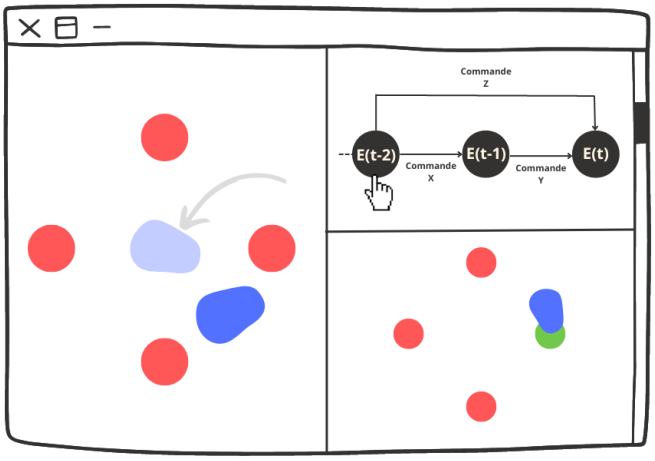


FIGURE 3.1 – Travail du PAndroide

Nous avons codé un programme avec une figure qu'on pouvait déplacer et tourner dont la représentation du vecteur pour l'apprentissage était ses coordonnées (x et y) et sa rotation (à gauche dans la figure). Les mouvements étaient soit un mouvement simple (direction ou rotation d'un sens), soit un mouvement qui fait un plus grand déplacement en une action. Nous avions aussi ajouté une tâche utilisateur (en bas à droite de la figure) qui donne un objectif lors de l'utilisation du logiciel. Le recommandeur (en haut à droite de la figure) suggérait une commande lorsqu'un usager se servait d'une commande qui pouvait être remplacé par une autre qui serait mieux. Un exemple de présentation visible sur l'image est d'afficher les états et transitions vers les états correspondant aux commandes par l'usager et de montrer l'effet si on utilisait la recommandation à la place.

Pour mieux illustrer, lorsque l'utilisateur déplace la figure en appuyant plusieurs fois sur une touche directionnelle, l'assistant interroge le classifieur entraîné (l'utilisation du Machine Learning était une des tâches dans notre cas) et propose la commande qui fait un plus grand mouvement vers la direction correspondante, ce qui permet de mieux accomplir l'objectif. Le logiciel étant très simple ce stage va nous amener à appliquer ce système vers des trois autres cas (Editeur de texte, Logiciel de présentation et Photoshop).

Le fonctionnement du système pour le stage est représenté dans la suite. Cela est similaire au projet mais sans restriction sur le type de classifieur.

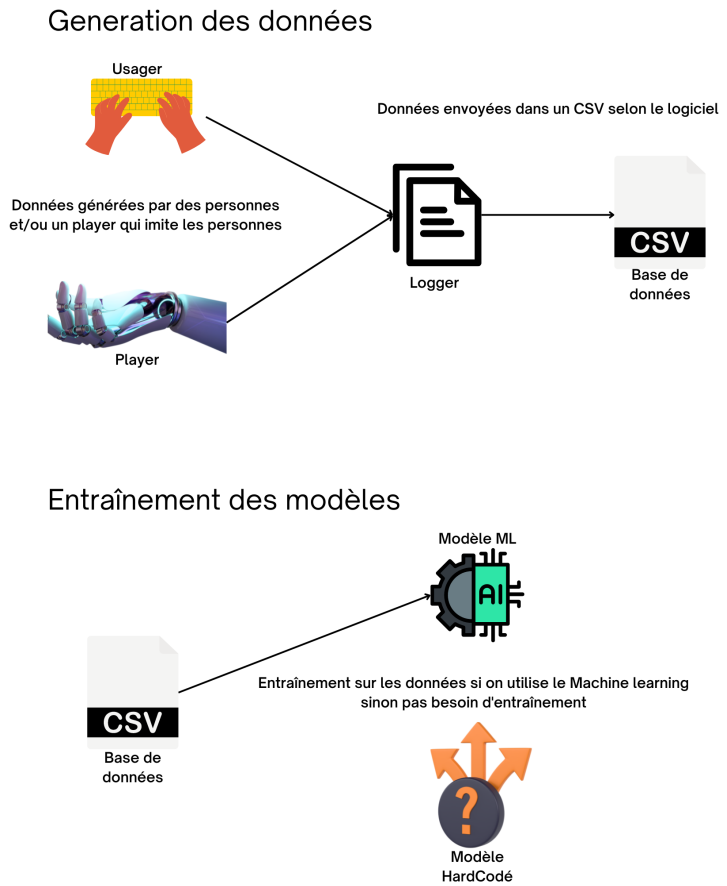


FIGURE 3.2 – Fonctionnement du système hors ligne : Génération des données et entraînement (ou non)

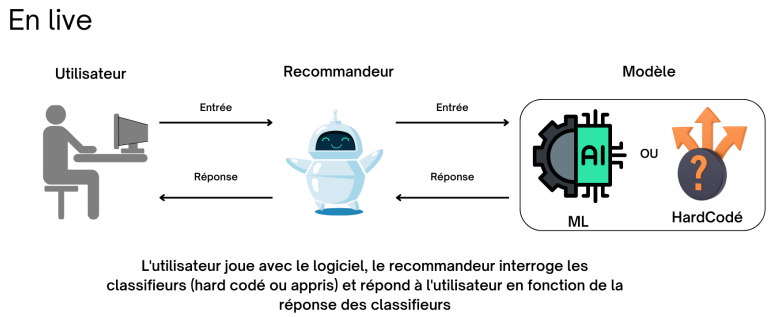


FIGURE 3.3 – Fonctionnement du système en live

Pour résumer les figures (les composantes du système) :

- Le player est un outil qu'on code pour imiter d'une certaine façon l'utilisateur afin de générer des données. C'est moins cher que de demander à d'autres personnes et plus rapide.
- Le recommandeur est l'interface entre l'usager et le système, il prend la décision d'afficher ou non la commande selon la prédiction du classifieur.
- Le logger est la composante qui détecte les commandes utilisées. Lorsqu'on n'est pas en utilisation par un usager, un changement d'état (défini plus tard selon le cas) du logiciel est enregistré et un "label" (classe) est attribué à ce changement : le nom de la commande. Quand le système est employé par un usager, son rôle est d'envoyer les informations de l'état au recommandeur.
- Le classifieur (ou "Mapper") est le modèle qui va permettre la prédiction d'une commande. Il peut être codé avec des règles définies ou entraîné (hors ligne) à reconnaître les "formes" des commandes.

3.1.2 Répartition des rôles

Majoritairement, nous avons chacun pris chacun un logiciel avec l'éditeur de texte pour moi, le logiciel de présentation (Powerpoint) pour Christian Zhuang et Photoshop pour Nassim Ahmed-Ali. La figure suivante représente la répartition des tâches ; les pourcentages sont assez arbitraires comme le travail était surtout expérimental :

	Alexandre	Nassim	Christian
Photoshop (code/player/images manuelles et données)	10%	80%	10%
Editeur de texte (code/player/données)	80%	10%	10%
Programme de présentation (code/player/données)	10%	10%	80%
Prise en main de Pytorch (du modèle)	50%	25%	25%
Recommandeur	25%	25%	50%
Vidéo	25%	50%	25%

Table : Répartition arbitraire des pourcentages des tâches

— Photoshop, Editeur de texte et Programme de présentation (Powerpoint) : Toutes les valeurs à 10% représentent surtout l'aide au débogage du code de chacun, c'est difficile de vraiment savoir à quel pourcentage cela correspond mais arbitrairement en prenant le temps on a décidé que ce serait à peu près 10%.

— Pour Pytorch, le Recommandeur et les vidéos, il est plus facile de donner des pourcentages :

  - Les valeurs à 50% correspondent pour Pytorch à la mise en place du modèle de base ainsi que l'application à l'éditeur de texte, pour le recommandeur c'est le codage de celui-ci et l'application à "Powerpoint" et pour la vidéo c'est surtout le montage en plus des scénarios du logiciel correspondant à Photoshop.

  - Les valeurs à 25% sont l'application de Pytorch (classifieur) et recommandeur aux différents logiciels et pour la vidéo aux scénarios (les vidéos sont faites ensemble).

3.1.3 Logiciel de présentation (Powerpoint)

Cette partie du projet est réalisée en majorité par Christian Zhuang. Cela a été codé sous python avec la librairie PyQt. Ce modèle utilise directement une image du slide sur lequel l'utilisateur travail. Pour simplifier la génération des données, le player n'utilisera pas directement les touches du claviers ou la souris mais fait apparaître directement les éléments sur le slide à partir des fonctions PyQt. Le classifieur utilisé dans un premier temps est un réseau de convolution (CNN) car l'entrée étant une image il est plus adéquat de faire cela. Pour ce logiciel nous avons finalement décidé d'utiliser un classifieur entièrement codé par des règles car pas d'entraînement n'est nécessaire et cela est facilement codé.

Pour le logiciel de présentation, ma seule contribution est d'aider au débogage car le classifieur dépendant de règles codées, on ne pouvait que vérifier si les règles correspondent bien aux effets des commandes. Une autre tâche qui dans ce cas a été fait en groupe est la décision des différentes commandes qui sont utilisées pour le système.

3.1.4 Photoshop

Cette partie réalisée surtout par Nassim Ahmed-Ali est comparé aux autres un peu différent. En effet, le logiciel utilisée ici est directement Photoshop. Nous avons décidé qu'il serait intéressant d'utiliser directement un programme existant pour au moins l'un des cas et nous avons décidé de prendre celui-ci car l'API était plus ou moins accessible et que les images sont plus faciles à gérer dans le cadre de Machine learning maintenant. Le modèle est très similaire à celui de "Powerpoint" car on a comme entrée une image qui est la photo sur laquelle l'utilisateur agit, on a également un réseau de convolution (CNN) ici. Cependant, la difficulté ici est qu'il n'est pas possible d'accéder aux commandes correspondant à l'état de la photo modifiée. Nous avons décidé de faire quelque chose qui permet de détecter certaines commandes en externe pour réaliser cela malgré ces limites.

Ma contribution ici était surtout côté Machine learning où nous devions comprendre le fonctionnement de Pytorch pour le cas des images. Cette partie était pour moi surtout des expérimentations de différents réseau de neurones (de convolution) et aide à la prise en main de la librairie pour mes camarades et moi. Les choix de commandes ici ont également été vérifiés en groupe. L'aide au débogage est aussi une des tâches ici.

3.1.5 Vidéo et exemple de scénario

Pour finir, nous avons fait une vidéo faisait une démonstration du système DISCO (Contextually Improving Command Discoverability) pour la preuve de concept. La vidéo contient pour chaque logiciel 2 scénarios qui montre le fonctionnement du système de recommandation lorsque l'utilisateur utilise des commandes basiques de manières répétées alors qu'il existe une commande qui pourrait potentiellement rassembler cela. Cette vidéo marque la fin de notre travail actuel pour la preuve de concept.

Les scénarios ont été fait individuellement avec chacun son logiciel mais pour bien vérifier les scénarios nous les avons passer en revu ensemble pour faire les modifications nécessaires. Pour le tournage des vidéos, nous les avons fait ensembles afin de bien vérifier que les vidéos sont cohérentes. Pour ce qui concerne le montage des vidéos, c'est Nassim qui s'est occupé de cette partie car c'est une partie qu'on ne peut pas vraiment séparer.

Un scénario permettant d'illustrer le but final du système est le suivant (pour l'éditeur de texte) : - Un utilisateur (qu'on appelera A) a écrit un paragraphe de ce qui lui passait pas la tête. Cependant, il se rend compte d'une erreur dans son texte, il a écrit le mot "day" au lieu de "night" et souhaite changer le mot. Il décide de se déplacer vers chaque mot "day" et les remplace un à un en les supprimant et écrivant le mot qu'il voulait. Pendant qu'il remplace les mots, notre système analyse les états (avec le classifieur) et reconnaît une forme qui correspond à la commande "chercher et remplacer" et recommande donc à l'utilisateur que cette commande peut être plus rapide pour accomplir sa tâche. L'utilisateur qui ne connaissait pas cette commande n'a plus à chercher individuellement chaque occurrence de "day". De plus, il aura peut-être retenu cette commande avancée suite à sa découverte et utilisation, ce qui lui permettra potentiellement d'être plus efficace dans ses futures tâches.

Le lien vers la vidéo se trouve ICI

3.2 Éditeur de texte

3.2.1 Objectifs et Difficultés

On va brièvement mettre en avant les objectifs et difficultés pour chaque partie avant de présenter ce qu'on a fait :

- Pour l'implémentation du logiciel, on doit choisir et implémenter une liste de fonctionnalités qui nous semblent pertinentes pour le fonctionnement du système. Nous avons pas vraiment de difficultés majeure dans cette partie puisqu'il suffit de chercher par rapport à un logiciel déjà existant (Microsoft Word) les commandes qui nous intéressent.
- Pour la représentation des données, Il faut définir ce qu'est l'état ici. Contrairement aux autres logiciels, prendre l'image avant et après modification peut être difficile. Le problème et la difficulté du système pour l'éditeur de texte vient majoritairement d'ici car il faut que les descripteurs des états englobent bien les changements liés aux commandes sur le logiciel. De plus, même si une représentation est fonctionnelle, une autre peut être potentiellement mieux (descripteurs mieux interprétés par les classifieurs). Cela rend ce choix subjectif car il n'y a pas de réponse concrète à cette partie surtout qu'on travail sur du texte écrit ou modifié par un utilisateur donc le

texte n'est pas fixe et les classifieurs entraînés ne prennent pas de descripteurs de différentes tailles, c'est alors une autre difficulté à prendre en compte.

— Ensuite, il faut générer des données qui serviront à l'entraînement dans le cas où on passe par un classifieur entraîné. Pour cela, deux approches sont apparentes : Un générateur automatique qui imite un usager en jouant avec un texte ou alors obtenir directement des données d'utilisateurs mais dans le cadre du stage, il sera très difficile de partir sur la seconde méthode. La difficulté pour la première approche est qu'on doit essayer de faire un générateur qui pourrait encadrer une grande partie du cas pratique pour que le classifieur reconnaisse bien les "formes" du texte.

— L'objectif final est de décider du type de modèle utilisé par un recommandeur. La difficulté ici est liée directement aux étapes précédentes et dépend fortement de la qualité des données générées mais aussi de la représentation de ces données. On a également le fait qu'il y a beaucoup d'hyperparamètres à prendre en compte ce qui rend la tâche difficile globalement. Par exemple, l'architecture du classifieur (architecture du réseau de neurones si on en utilise un), les différents paramètres d'apprentissage ou les performances entre un classifieur appris contre un classifieur qu'on aurait codé avec des règles. Le tout est très expérimental et peut très bien aboutir à un résultat fonctionnel pour le système ou non.

3.2.2 Les choix des commandes liés à l'éditeur de texte

Pour l'implémentation de l'éditeur de texte, nous avons décidé d'utiliser PyQt nous permettant surtout d'avoir un contrôle sur le logiciel crée. Le logiciel étant fait par nous même il est plus simple d'apporter des éléments extérieurs (modèle de classification et recommandeur). Nous avons retenu 2 groupes de commandes arbitraires pour tester notre système.

1. Des commandes qui agissent directement sur le texte :

— Chercher et remplacer par la touche clavier "CTRL+R". Cette commande permet comme le nom l'indique de chercher un élément et de le remplacer. Il faudra noter que dans notre cas, on remplace toutes les occurrences pour permettre de mieux généraliser, car si le cas où pour toutes les occurrences cela fonctionne alors ce sera le cas pour la commande élément par élément ; Par rapport au système, cette commande est une alternative plus rapide en comparaison à chercher chaque élément et les supprimer un à un par le mot que l'on souhaite écrire.

— Copier-Coller (CTRL+C et CTRL+V) des mots sélectionnés que l'on souhaite ajouter à différents endroits. On pourrait généraliser cela pour les morceaux de texte mais il faudrait pouvoir utiliser des données fait par des utilisateurs car reconnaître ce qu'on sélectionne par un programme qui imite l'utilisateur aléatoirement n'est pas très efficace pour cette tâche ; Le système voudrait suggérer cette commande à la place d'écrire le mot à la main.

— Supprimer un mot (CTRL+Backspace). Le mot au niveau du curseur de saisie (curseur clavier) sera supprimé après entrée de la commande. C'est une commande existante dans PyQt il faut donc juste pouvoir la détecter lorsque l'utilisateur supprime un mot en appuyant plusieurs fois sur la touche "Backspace" afin de suggérer la commande.

2. Des commandes qui agissent sur le déplacement ou la sélection :

— Les déplacements mot par mot (CTRL+Left ou Right) qui remplace les déplacements caractère par caractère sur une certaine distance. Par exemple si l'utilisateur appuie plusieurs à droite, il faudra pouvoir lui recommander la commande de déplacement de mot. On a aussi inclut les déplacements vers le début et la fin de ligne qui nous semblait également intéressant pour accélerer le déplacement.

— Le même équivalent pour la sélection des mots, avec "CTRL+Shift+Left ou Right" pour les mots et "Shift+Home ou End" pour les lignes.

— Enfin, nous prenons également la commande de sélection du document entier avec "CTRL+A". Nous donnant ainsi de bonnes différences entre les commandes pour tester en partie la robustesse du système.

On a choisi ces commandes car elles sont relativement facile à voir visuellement et les effets sont assez différents sur l'état de l'application quand on les utilise. Les commandes de sélections sont déjà implémentées, mais il faut tout de même pouvoir détecter leur utilisation ainsi que les touches de claviers simples qui changerait le texte comme les lettres.

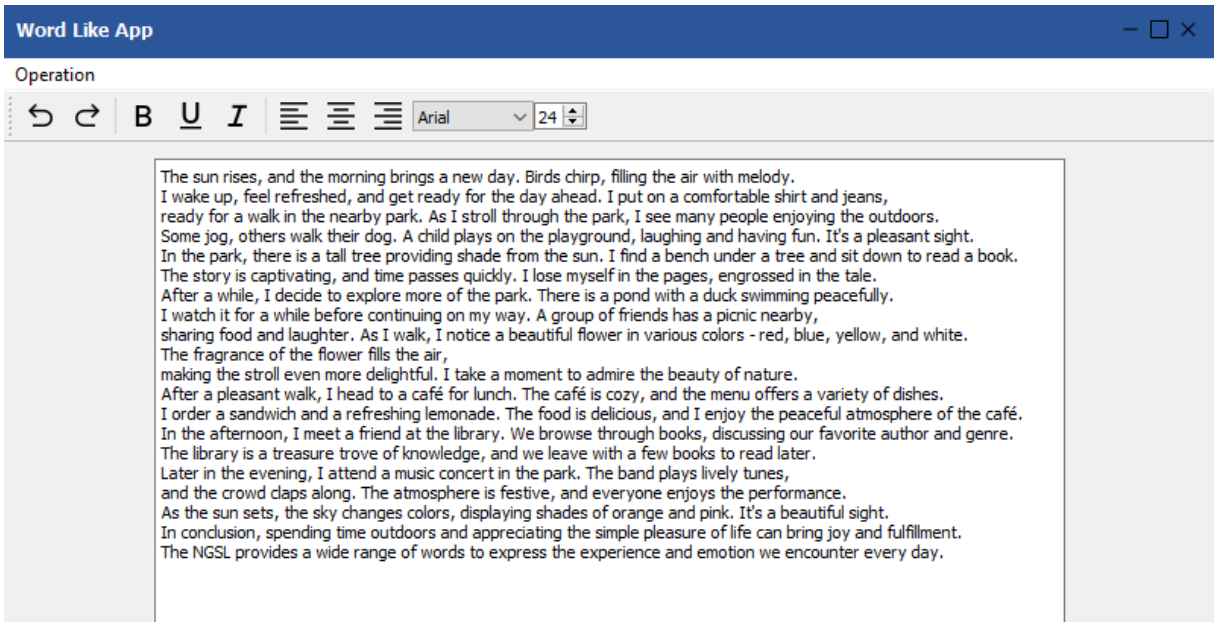


FIGURE 3.4 – Logiciel PyQt pour l'éditeur de texte

3.2.3 Gestion de la représentation des états et base de données avec player

L'étape pour la représentation d'un état dans le cas de l'éditeur de texte est un peu plus difficile par en comparaison aux deux autres qui utilisent directement des images. 3.2.1. Les différentes approches que nous avons testé pour cela sont :

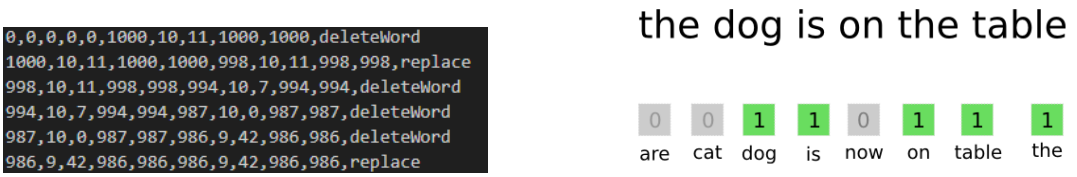
— Représentation avec un vecteur bag of word classique et un vecteur contenant les positions de curseur de saisie et de sélection. Le bag of word ("sac de mots") en lui même dépend du vocabulaire utilisé. Les informations du curseur de saisie sont la position dans le texte, la ligne et colonne ainsi que les positions de sélection (début vers fin). La représentation bag of word transforme un texte en vecteur contenant le

nombre d'occurrences de chaque mot du texte s'ils sont présents dans le vocabulaire utilisé pour le vecteur.

— Deux vecteurs séparés avec en plus dans le vecteur bag of word, les mots non compris dans le vocabulaire et le nombre de caractères du document. Pour l'autre vecteur, nous gardons le même que le précédent

Un problème majeure liée à la première représentation (qui utilise un bag of word classique) est le fait qu'on ne travail pas sur un corpus de texte fixe car un utilisateur écrit librement son texte alors que le vocabulaire du bag of word est fixé. Il semble évident que la seconde représentation est plus adéquate parmi les approches considérées. Nous avons également la possibilité de passer par d'autres représentations utilisées récemment notamment "Word Embedding" utilisé dans des transformers (approche utilisée par chatGPT par exemple). Cela utilise la sémantique, ce qui n'est pas vraiment nécessaire dans notre cas, et nous demande de bien comprendre la notion qui est encore utilisée expérimentalement dans la recherche/littérature.

En revenant à notre cas, on a comme vecteur de sélection : (position du curseur de saisie, ligne du curseur, colonne du curseur saisie, début de sélection, fin de sélection) 2 fois, un avant la commande et un après (avec un échantillon ci-dessous). Le vecteur bag of word étant illisible (utilisant un dictionnaire de mots très dense), nous ne mettons pas d'image mais des exemples peuvent êtres retrouvés sur le github (dans les fichiers de type DataBow.csv) et un exemple simplifié (en-dessous).



(a) Echantillon de vecteur position

(b) Exemple de bag of word

FIGURE 3.5 – La forme des données

A partir de ces représentations, il nous faut une base de données qui nous permettra d'apprendre dans le cas d'un classifieur de type supervisé. Pour mener à bien cela, nous avons fait un player qui imite d'une façon l'utilisateur avec les commandes choisies pour le système. Le classifieur entraîné permet de mieux automatiser les recommandations et potentiellement mieux généraliser dans le sens où contrairement au classifieur codé par règles, nous n'avons pas besoin d'écrire une règle spécifique à chaque nouvelle commande (du même groupe) qu'on souhaite ajouter. Dans le cas où le classifieur entraîné ne donne pas de bons résultats, il est peut-être judicieux d'essayer avec un classifieur codé par règles.

Le player pour ce logiciel en particulier utilise directement PyQt avec la librairie QTest qui permet d'imiter les entrées claviers/souris, dans notre cas on se limitera aux entrées clavier pour simplifier la tâche et également car la souris sera en général plus rapide que le clavier dans la sélection si on laisse la souris. Le player pourra utiliser toutes les commandes de la partie 3.2.2. On a également ajouté une commande d'écriture pour permettre au modèle de reconnaître cela lorsque l'utilisateur écrit un mot qui n'existe pas dans le texte pour éviter de recommander des commandes qui n'ont pas de sens.

Pour son fonctionnement, le player choisira aléatoirement une commande parmi la liste et l'exécutera. Pour avoir des données assez différentes, on place le curseur de saisie à des positions de départ différentes choisi aléatoirement après qu'un certain nombre de commandes soit exécutées. Dans la littérature, notamment ceux de l'état de l'art, la génération des données est pris de journaux (logs) d'utilisateur comparé à notre cas où on génère automatiquement ces données sans l'utilisateur. Cependant, des données venant d'un utilisateur sont généralement mieux pour entraîner un classifieur face à un player qui utilise aléatoirement les commandes car ce sont des données qui apparaissent en pratique. La difficulté pour nous est surtout d'avoir des données cohérentes par rapport au cas pratique même si cette approche nous permet d'avoir des données facilement et ne nécessite pas beaucoup de ressources.

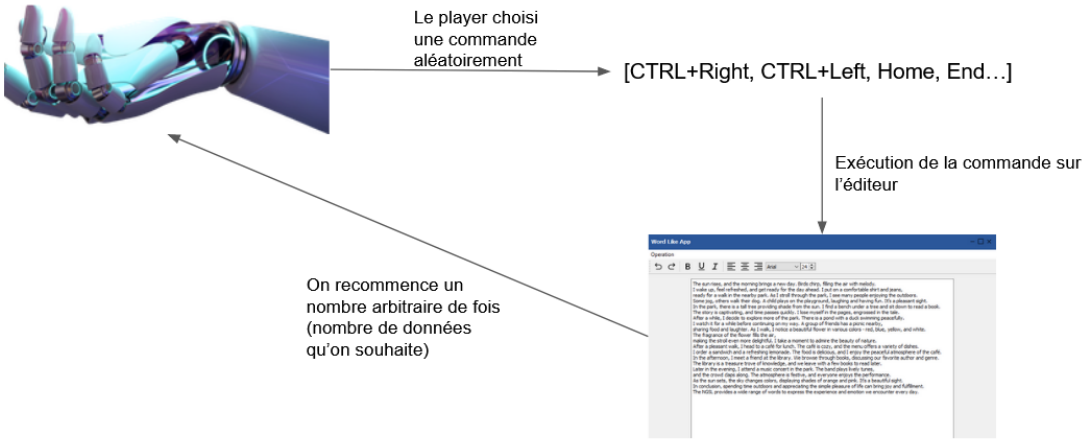


FIGURE 3.6 – Cycle de génération des données

3.2.4 Modèles de classification

Les données générés avec les deux représentations choisies, nous avons testé différents modèles de classification. L'approche étant très expérimentale, nous présentons plutôt les résultats liés aux modèles que nous avons étudiés. Cela nous permet de bien voir les limites de certaines représentations ou leurs points forts.

Les différents classifieurs considérés sont les suivants :

— Un classifieur unique qui utilise la représentation qui concatène le bag of word avec le vecteur des positions et de sélection. C'est le cas de base vers lequel on se tourne car on aurait un classifieur supervisé général pour le logiciel.

— Un classifieur pour chacune des représentations vecteur bag of word et le vecteur des positions. Ce modèle est plutôt pour des vérifications de performances sachant que le bag of word et le vecteur de positions jouent des rôles différents selon le groupe de commandes.

— Enfin, le dernier modèle est la combinaison d'un modèle codé selon des règles pour le vecteur de positions et d'un modèle appris pour le vecteur bag of word. Ce modèle découle plutôt du fait qu'il est simple de choisir des règles pour ce qui concerne la navigation et sélection dans un texte et qu'il est potentiellement mieux de faire

cela pour éviter d'avoir de mauvaises recommandations. Par exemple, lorsqu'un utilisateur se déplacement suffisamment dans une direction, on peut suggérer en comparant les positions du curseur de saisie avant et après déplacement la commande de déplacement mot par mot.

Chaque classifieur entraîné est un réseau de neurones, ce choix est arbitraire mais c'est aussi pour rester uniforme avec les modèles de réseau de convolution (CNN) utilisés par mes camarades. Dans le cas où on passerait des données de type images à la place, cela permettrait de facilement changer le modèle en ajoutant des couches de convolutions.

3.2.5 Choix du modèle final et résultats

Pour les différents modèles qui ont été appris (PyTorch), suite à l'expérimentation, on obtient les résultats suivant sur les données de "Test/Validation" :

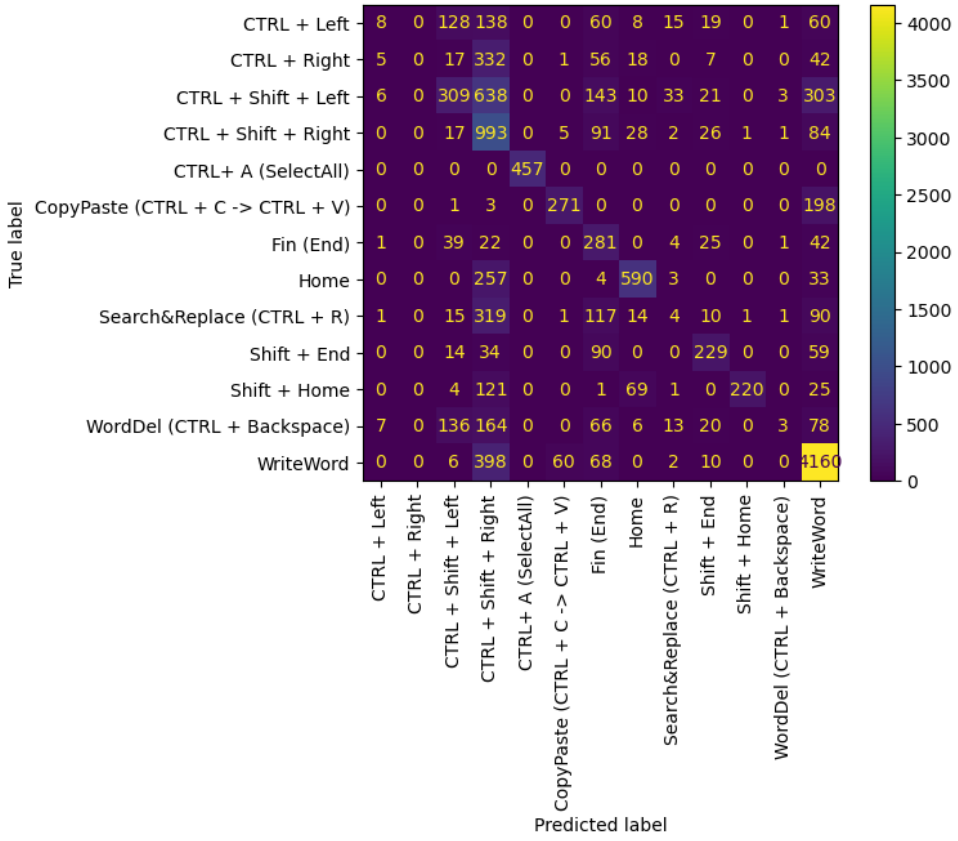


FIGURE 3.7 – Matrice de confusion pour le modèle unique de classifieur

	Précision	False Positive Rate
CTRL+Right (saut de mot droite)	28.6%	0.1%
CTRL+Left (saut de mot gauche)	0%	0%
CTRL+Shift+Left (selection de mot gauche)	45%	3.4%
CTRL+Shift+Right (selection de mot droite)	29%	21.5%
CTRL+A (Tout sélectionner)	100%	0%
CopyPaste (CTRL+C -> CTRL+V)	80.2%	0.5%
End (déplacement vers fin de ligne)	28.8%	5.7%
Home (déplacement vers début de ligne)	79.4%	1.3%
Search and Replace (CTRL + R)	5.2%	0.6%
Shift+End (Sélection jusqu'à la fin de la ligne)	62.4%	1.1%
Shift+Home (Sélection jusqu'au début de la ligne)	99.1%	0%
CTRL+Backspace (suppression d'un mot)	30%	0%
WriteWord (Ecriture d'un caractère)	80.4%	13%

Table 1 : Tableau des précisions et taux de faux positif pour le modèle unique

La précision totale du modèle est de 60%, ce qui n'est pas très bon dans le cadre d'utilisation de notre système. On observe par le tableau qu'il y'a certains problèmes, le saut de mot (CTRL+Droite) vers la droite n'est pas du tout reconnu et la sélection d'un mot vers la droite a un taux de faux positif très élevé. En observant la matrice de confusion liée au-dessus, la plupart des fausses prédictions se font par cette commande (il y a plus de données pour l'écriture de mots mais sachant que ce n'est utilisé que pour permettre au modèle de reconnaître cela, il n'y a pas vraiment de problème). Dans la globalité, on aurait très probablement de grandes difficultés à faire fonctionner ce modèle en pratique.

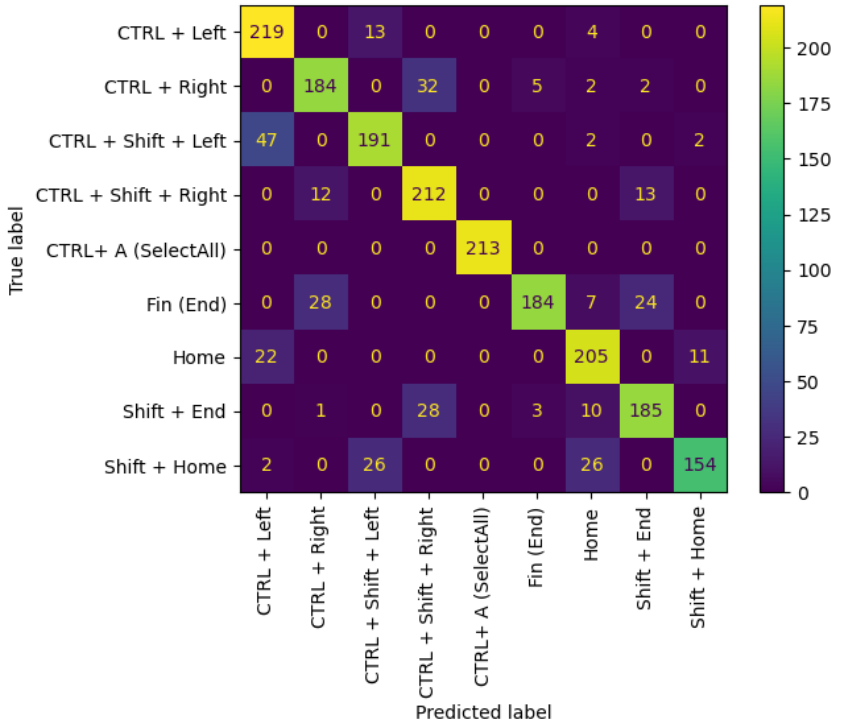


FIGURE 3.8 – Matrice de confusion pour le modèle sur les commandes de sélection

	Précision	False Positive Rate
CTRL+Right (saut de mot droite)	81.8%	2.2%
CTRL+Left (saut de mot gauche)	75.5%	3.8%
CTRL+Shift+Left (selection de mot gauche)	83%	2.1%
CTRL+Shift+Right (selection de mot droite)	77.9%	3.2%
CTRL+A (Tout sélectionner)	100%	0%
End (déplacement vers fin de ligne)	95.8%	0.4%
Home (déplacement vers début de ligne)	80.1%	2.7%
Shift+End (Sélection jusqu'à la fin de la ligne)	82.6%	2.1%
Shift+Home (Sélection jusqu'au début de la ligne)	92.2%	0.7%

Table 2 : Tableau des précision et taux de faux positif pour le modèle de machine learning sur le vecteur de sélection.

Le tableau pour la sélection donne des résultats prometteurs à première vue. La précision est plutôt élevée et le taux de faux positif est assez bas. De plus, la matrice de confusion montre de bons résultats globalement. En théorie, cela indique que ce serait potentiellement fonctionnelle pour la preuve de concept, mais en utilisant le modèle en pratique pour la sélection, on s'est rendu compte que cela ne fonctionne pas du tout. Par exemple, les déplacements sont confondues entre eux (droite et gauche). Une explication possible viendrait du fait que les données ont été générées aléatoirement et par conséquent, on a des données qui ne représentent pas les cas réels. Il nous faudrait des données générées grâce a des actions utilisateurs pour avoir des résultats plus concluants.

On notera que c'est surtout sur ce point qu'on se différencie des méthodes employées dans la littérature ; Ils utilisent des données d'historique utilisateur pour entraîner leur modèle alors qu'on essaye de générer ces données automatiquement qui est moins coûteux mais également difficile à utiliser. La précision globale est de 84.4% ce qui pour des données tests plutôt bons.

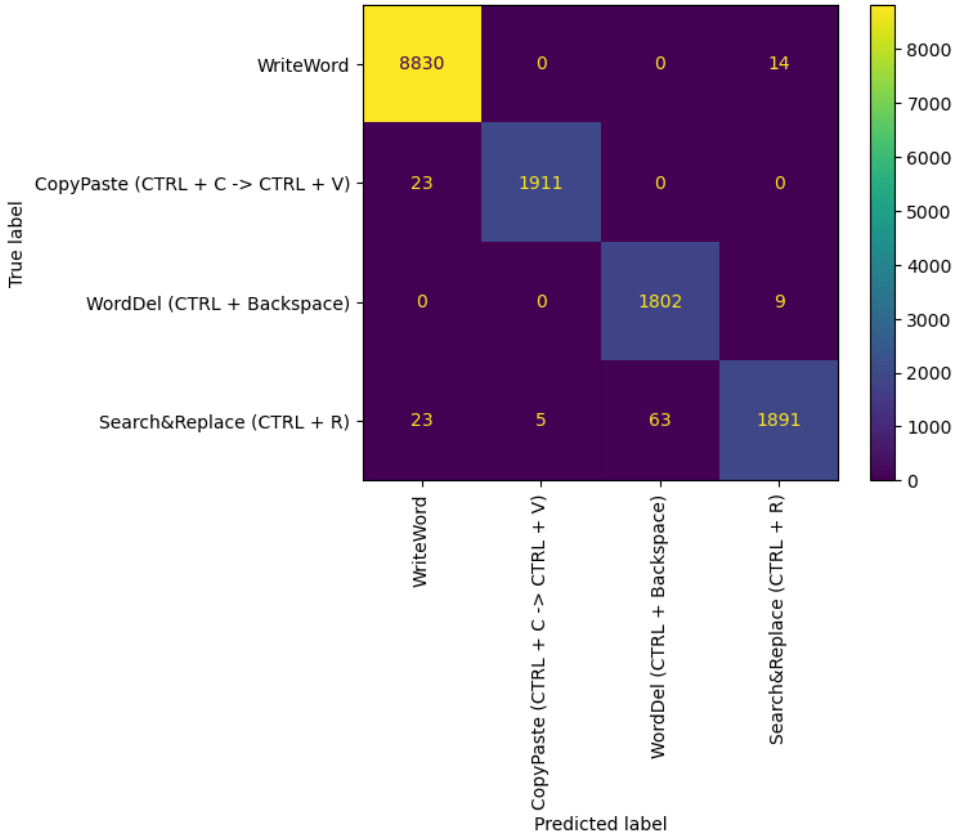


FIGURE 3.9 – Matrice de confusion pour le modèle sur les commandes du vecteur Bag of word

	Précision	False Positive Rate
CopyPaste (CTRL+C -> CTRL+V)	99.7%	0%
Search and Replace (CTRL + R)	98.8%	0.1%
CTRL+Backspace (suppression d'un mot)	96.6%	0.4%
WriteWord (Ecriture d'un caractère)	99.5%	0%

Table 3 : Tableau des précisions et taux de faux positif pour le modèle de machine learning sur le vecteur bag of word.

Pour le vecteur bag of word, le modèle donne des résultats très élevés pour la précision et un taux de faux positif presque nul. En utilisant cela en pratique, les commandes agissant sur le texte directement donne bien ce à quoi on s'attend en majorité et ce résultat nous permet de déduire que ce modèle est bien utilisable pour la preuve de concept. On se retrouve donc à la fin avec seulement le vecteur bag of word utilisable et plutôt efficace en pratique avec une précision de 99.1% qui est le meilleur résultat obtenu par rapport au classifieur de sélection qui ne reconnaît pas toutes les commandes ou celui combinant les deux (sélection et bag of word) qui fonctionne pas très bien en pratique.

Pour le modèle codé avec des règles qui correspond à la partie du vecteur sélection, on n'a pas d'évaluation de type précision à faire, la décision donnée par ce classifieur dépend juste de la façon dont celui-ci est codé et donc de nos règles. L'évaluation qu'il

serait possible de faire est de vérifier que nos règles correspondent aux bons critères (si cela correspond bien au comportement de la commande sur l'état).

Pour notre choix final, l'approche retenue est celui qui combine le classifieur appris sur le vecteur bag of word et un classifieur qui suit des règles codées et ne nécessite pas d'entraînement au préalable et donc pas de données. Cependant, le format du vecteur utilisé sera le même. Dans notre contexte, nous avons fait en sorte de vérifier si un changement a été appliqué au texte, si c'est le cas, nous interrogeons le classifieur supervisé en ne gardant que des décisions où le taux de confiance est supérieur à 95% car en pratique nous remarquons qu'au dessus de ce pourcentage, le recommandeur fait beaucoup moins de faux positif ce qui nous semblent pertinent, sinon on interroge directement le classifieur codé par règles qui garantie une décision (sorte d'arbre de décision).

Chapitre 4

Conclusion

Pour conclure, nous avons donc conçu un système qui sert de preuve de concept pour un recommandeur de commandes contextuel. Pour cette partie concernant donc l'éditeur de texte, nous avons réussi à créer un player qui permet l'apprentissage pour quelques commandes qui agissent directement sur le texte avec une forte précision que ce soit à l'entraînement, test ou en pratique. Cependant, il semblerait comme vu précédemment que cela soit un peu difficile pour ce qui concerne la sélection, car la qualité des données ne reflètent pas assez bien la réalité et montre la difficulté d'un modèle entraîné en pratique.

Le système fonctionnant plutôt bien avec le classifieur entraîné sur le vecteur bag of word (commandes sur le texte) et le classifieur codé avec des règles (commandes de sélection), nous avons donc pu mener à bien ce travail pour la preuve de concept en créant la vidéo de démonstration.

Toutefois, il faut noter qu'avoir des données venant de comportements humain permettrait potentiellement de passer à un modèle entièrement entraîné qui est plus facile à employer dans la cas où il faut généraliser à une majorité des commandes d'un logiciel.

Par ailleurs, une futur perspective qu'on pourrait explorer dont on a surtout discuté avec nos encadrants Gilles Bailly et Julien Gori mais qui n'est pas abordé dans les exemples de ce projet : le code. Par exemple, l'utilisation de la bibliothèque pandas (avec des dataframes) où celui qui code utilise beaucoup de commandes basiques pour transformer un dataframe mais qu'on aurait pu directement utiliser une seule commande qui regrouperait les actions simples. Dans ce genre de système, on aurait sûrement besoin de méthodes de traitement du langage naturel (NLP) comme les transformers, modèle utilisé par ChatGPT.

Bibliographie

[1] Andy Cockburn - Carl Gutwin - Joey Scarr - Sylvain Malacria. « Supporting Novice to Expert Transitions in User Interfaces ». In : *ACM Comput. Surv.* (2014). URL : https://inria.hal.science/hal-02874746/document (page 2).

[2] Jia Haitao - Yang Wenhua - Shen Chaochao - Pan Minxue - Zhou Yu. « Git command recommendations using crowd-sourced knowledge ». In : *Information and software technology, Vol.159, p.107199, Article 107199* (2023). URL : https://doi.org/10.1016/j.infsof.2023.107199 (page 3).

[3] Md Adnan Alam Khan - Volodymyr Dziubak - Andrea Bunt. « Exploring Personalized Command Recommendations based on Information Found in Web Documentation ». In : *IUI '15 : Proceedings of the 20th International Conference on Intelligent User Interfaces, Pages 225–235* (2015). URL : https://doi-org.accesdistant.sorbonne-universite.fr/10.1145/2678025.2701387 (page 3).

[4] Justin Matejka - Wei Li - Tovi Grossman - George Fitzmaurice. « CommunityCommands : command recommendations for software applications ». In : *UIST '09 : Proceedings of the 22nd annual ACM symposium on User interface software and technology, Pages 193–202* (2009). URL : https://doi-org.accesdistant.sorbonne-universite.fr/10.1145/1622176.1622214 (page 3).

[5] Emerson Murphy-Hill - Rahul Jiresal - Gail C. Murphy. « Improving software developers' fluency by recommending development environment commands ». In : *FSE '12 : Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, Pages 1–11 , Article No. : 42* (2012). URL : https://doi-org.accesdistant.sorbonne-universite.fr/10.1145/2393596.2393645 (page 3).

[6] Samarth Aggarwal - Rohin Garg - Abhilasha Sancheti - Bhanu Prakash Reddy Guda - Iftikhar Ahamath Burhanuddin. « Goal-driven Command Recommendations for Analysts ». In : *RecSys '20 : Proceedings of the 14th ACM Conference on Recommender Systems, Pages 160–169* (2020). URL : https://doi-org.accesdistant.sorbonne-universite.fr/10.1145/3383313.3412255 (page 3).

[7] Ofer Dan - Brandes Nadav - Linial Michal. « The language of proteins : NLP, machine learning & protein sequences ». In : *Computational and structural biotechnology journal, Vol.19, p.1750-1758* (2021). URL : https://doi.org/10.1016/j.csbj.2021.03.022 (page 3).

Annexe A

Cahier des charges

A.1 Cahier des charges

A.1.1 Introduction

Ce projet a pour objectif de créer un assistant qui incite l'utilisateur à découvrir certaines commandes qui sont potentiellement meilleures que d'autres. Dans la suite du projet de Master I, certains points restent identiques : On aura un assistant qui analysera l'état du logiciel (image ou vecteur représentant cela) et proposera par un classifieur, une meilleure commande s'il en existe une (qui combinerait d'autres commandes basiques par exemple).

Pour mener à bien la preuve de concept, on implémentera cette assistant dans 3 logiciels différents : Un éditeur de texte (type word), Un programme de présentation (Google slide, Powerpoint) et enfin un logiciel de traitement d'images/dessins (photoshop). En prenant un exemple dans un éditeur de texte, l'utilisateur pourrait faire supprimer un mot en appuyant plusieurs fois sur "backspace" au lieu de juste appuyer sur "CTRL + backspace" qui supprime le mot directement et qui serait plus efficace au niveau du temps et effort.

Par rapport à d'autres travaux, il faut ici qu'on utilise l'état de l'application (le contexte en d'autres termes) pour comprendre l'intention de l'utilisateur plutôt que de suggérer une commande en fonction de ceux qui ont été utilisées par l'usager.

Le travail à réaliser sera représenté dans un premier temps par le schéma suivant (cela permettra de faire la base du projet) :

Le travail consiste à :

généraliser à différents contextes (Éditeurs de forme, éditeur photo, éditeur de texte).

- Réaliser un composant qui capture l'ensemble des commandes (pertinentes) de l'application
- Réaliser un modèle qui, à partir de deux états de l'application retourne la commande la plus appropriée.
- Réaliser un assistant qui repose sur les composants précédents pour présenter les raccourcis les plus pertinents aux bons moments.

— Réaliser des évaluations techniques pour comprendre les performances du modèle
— Réaliser des études utilisateurs pour évaluer la pertinence de l'approche.
— Réaliser des démonstrateurs pour étudier dans quelle mesure cette approche peut être

A.1.2 Les 3 Logiciels

Pour ce qui concerne les logiciels, l'éditeur de texte et le programme de présentation seront fait par nous même à l'aide de python (librairie PyQt) avec des fonctionnalités qui nous semblent pertinentes. On a grâce à cela, un meilleur contrôle sur ce qu'il passe dans les commandes. L'idée étant d'explorer les différentes commandes et de choisir ceux qui seraient les plus pertinents et reconnaissables par notre système.

Pour le logiciel de traitement d'images, nous avons décidé qu'il serait intéressant d'utiliser directement un des logiciels déjà entièrement fait entre krita (qui est open source) et photoshop. Le choix sera fait en fonction des fonctionnalités du logiciel.

Les décisions prisent sur quelles commandes nous semblaient intéressantes à utiliser sur l'assistant sont :

Pour le logiciel de type Word :

— Chercher et remplacer (CTRL + R dans notre logiciel)
— Les commandes de sauts entre les mots (CTRL + -> ou <-)
— Les commandes de sauts vers le début et vers la fin d'une ligne (home ou end sur un clavier qwerty)
— Les commandes de selections de mots (CTRL + SHIFT + <- ou ->) et du document (CTRL + A)
— La commande d'indentation (TAB)

Chacune de ces commandes peuvent être effectuées de façon un peu moins efficace (niveau temps et nombre de commandes)

Pour le Logiciel de type Powerpoint

— Commandes de groupage (CTRL + G) et dégroupage (CTRL + SHIFT + G) + Commandes alignements

Pour photoshop

— blur (gaussien, surface et iris)

Une grande partie du projet de stage a pour objectif d'implémenter les éléments de l'article suivant :

On devra également étudier différentes approches en ce qui concerne le modèle et ses données :

— Des modèles d'apprentissages et des modèles codés en dur.
— Les données seront générées par un outil qui imitera d'une façon l'utilisateur (utilisation aléatoire ou précise des commandes)

— Différents traitements sur les données ou même une façon particulière de les générer (cropping sur les images, normalisation, introduction de données fait à la main)

A partir de tous ces logiciels, le produit final qu'on doit fournir est une vidéo de scénarios pour chaque logiciel qui présente le concept de l'article (DISCO) donné par nos encadrants.

Au fur et à mesure du stage, d'autres points seront développés avec nos encadrants. Les tâches n'étant pas concises dû au fait qu'il faille expérimenter avec les différents logicielles et décider des commandes ainsi que des classifieurs les plus performants pour notre système.