TERJANI Reda
BONNETAIN Baptiste

RAPPORT DE PROJET

CHATBOT

Sommaire

I - Présentation du sujet

1.1- Sciences des données

Le sujet de ce projet s'inscrit dans le domaine très vaste des sciences des données. Cette discipline récente s'appuie sur une mélange d'outil mathématiques, statistiques et d'optimisation, mais également sur les méthodes probabilistes et l'apprentissage automatique afin de traiter et d'exploiter au mieux la grande quantitée d'information dont la société moderne est submergée. Tout au long du projet, nous avons été confrontés à des problèmes d'analyse des données et de mise en place des algorithmes d'apprentissage que nous allons détailler dans ce rapport.

1.2- Identification des enjeux du projet

L'objectif principal du projet est de créer un **Chatbot industriel,** c'est à dire un agent conversationnel capable de converser avec un internaute. La volonté de mettre en place un ChatBot vient du besoin d'automatisation d'une tâche particulière, qui est dans notre cas l'**activité d'impression en ligne**, ainsi que l'ordonnancement des tâches d'impression. Le projet contient deux modules : **Chatbot & Gestion d'Agenda**

- **Module Chatbot :** Le module chatbot est constitué de deux parties

  . **Partie "Hors Ligne"** : Chargé de créer et entraîner un modèle de classification des attentes (demande d'impression, autre ..).

  . **Partie "En Ligne" :** Réalise l'analyse d'une requête utilisateur, la classifie et extrait les informations pertinente.

- **Module de Gestion d'Agenda :** Chargé d'ordonnancer les demandes en remplissant la liste des tâches d'impression de la machine et les dates. On suppose que le temps d'impression d'une page est 1 seconde.

1.3- Étape de résolution

Dans un premier temps, nous avons identifié le type de problème auquel nous étions confrontés, qui est un **problème de classification supervisé,** ainsi que le type de format de données que nous devions créer qui est un format texte, ce qui nous amène à ce qu'on appelle le traitement automatique des langues. Le traitement automatique des langues, **TAL,** est une technologie qui permet aux machines de comprendre le langage humain grâce à l'intelligence artificielle. Dans un deuxième temps nous nous sommes documentés sur les différentes approches possible pour résoudre ce type de problème en utilisant de **l'apprentissage automatique** ou de **l'apprentissage profond.** Et enfin, nous avons comparé les différentes approches implémentées en utilisant des métriques adéquates pour évaluer les performances et choisir la meilleure pour notre cas d'études.

1.4- Répartition du travail

Dans un premier temps, nous nous sommes penchés ensembles sur la recherche des méthodes à utiliser puis nous avons décidé de qui partirait sur quelle méthode. Tout au long du projet, nous avons travaillé en parallèle en nous tenant au courant de l'avancé de nos méthodes, et en s'entraidant lors des problèmes rencontrés. Puis nous avons fini par discuter de la meilleure approche et nous avons continué dessus afin de l'améliorer. Enfin, nous avons réfléchi au module de gestion de l'agenda pour l'optimiser le plus possible et nous l'avons implémenté en tenant compte de l'avis de chacun.

II - Travail Effectué

2.1- Environnement de travail & Framework

. Langage de Programmation

En analysant les outils que nous allons utiliser pour la résolution du projet, nous avons décidé d'utiliser **python** en tant que langage de programmation. Python est un langage de plus en plus utilisé en Sciences des données, notamment pour tout ce qui apprentissage automatique et apprentissage profond.

. Framework

Python possède plusieurs librairies pour implémenter des algorithmes d'apprentissage ainsi que pour l'exploration des données. Dans notre cas, nous avons utilisé : **Numpy**, **Pandas**, **Scikit-learn**, **Tensorflow** & **Keras, NTLK, tkinter**

- **Numpy :** manipulation des tableaux
- **Pandas** : librairie permettant de manipuler facilement les données sous forme d'un dataframe.
- **Scikit-learn :** librairie permettant d'implémenter les algorithmes d'apprentissages ainsi que des algorithmes de traitement du texte.
- **Tensorflow & Keras BackEnd :** implémentation des réseaux de neurones et traitement du texte
- **Nltk :** librairie de traitement de texte
- **tkinter :** librairie utilisée pour implémenter l'interface graphique

En ce qui concerne la visualisation des données, nous avons utilisé **matplotlib** et **seaborn.**

2.2- Données

Les données n'étant pas fournies dans le sujet du projet, nous devions nous même les collecter de la meilleure façon possible afin que cela soit représentatif et adapté à nos algorithmes d'apprentissages. Nos données sont constituées de deux colonnes : **Messages reçus** et **Attente Prédite**.

Les **messages reçus** sous forme "*string*" est notre variable de prédiction, c'est à dire qu'elle va nous servir à prédire si notre machine doit effectuer une tâche particulière ou non.

L'**attente prédite** sous forme de "string" (**Impression ou autre**) est la variable à prédire.

Nous avons environ **100 échantillons** pour entraîner notre modèle d'apprentissage.

2.3- Pré-traitement des Données

Les données étant sous format texte ne peuvent pas être utilisées directement par les algorithmes d'apprentissages, pour cela nous devons les convertir de données catégoriques en données numériques. Pour le cas de la variable prédictive i.e Message reçu, nous avons utilisé des méthodes de pré-processing de texte tel que **CountVectorizer**, **TfidVectorizer**, **Word Embedding** que nous avons détaillé dans le prochain paragraphe (Cf . 2.4). En ce qui concerne la variable à prédire qui est une variable binaire, nous l'avons converti en des entiers: **1 pour impression et 0 pour autres.**

2.4 - Algorithmes de pré-traitement de textes et d'apprentissages

. Algorithmes de pré-traitement de textes

Pour cette partie de pré-traitement, comme nous étions confrontés à un problème de type **classification,** nous avons trouvé comme approches ce qu'on appelle **Sac de Mots**, *en angais* **Bag of Words.** Cette approche consiste à créer une grande matrice dite sparse, qui stocke tous les mots que nous avons dans notre corpus (corpus = tous les documents = toutes les phrases). Parmi ces algorithmes, il y a:

- ***CountVectorizer :*** Elle permet de tokenizer la phrase dans un premier temps puis assigner une valeur à chaque token, donc chaque token est représenté par un seul index. *Scikit-learn* propose une fonction CountVectorizer facile à utiliser.

TF-IDF : abréviation de term frequencies and inverse document frequency. Les deux termes sont définis par les relations suivantes :
TF(t) = Nombre de fois un terme t apparaît dans un document / Nombre total des termes dans le document
IDF(t) = LOG (Nombre total des documents / Nombre des documents contenant le terme t)

Et finalement la relation finale est : **TF-IDF(t) = TF(t) * IDF(t)**
Scikit-learn propose une fonction TfidfVectorizer facile à implémenter.

Un autre concept en traitement automatique des langues est **N-grams.** **N-grams** est une combinaison de mots en ordre. Ce concept peut améliorer la performance de notre algorithme puisqu'il permet d'élargir encore plus notre corpus.

- **Stemming & Lemmatization :** ces méthodes permettent de réduire les mots à leur forme minimale.

- **StopWords :** Cette méthode nous permet de supprimer les mots qui n'ont aucune influence sur la décision de notre algorithmes d'apprentissages, et qui peuvent au contraire bruiter notre algorithmes. Par exemple : "the" ,"a", "," , ":", "he" etc ..

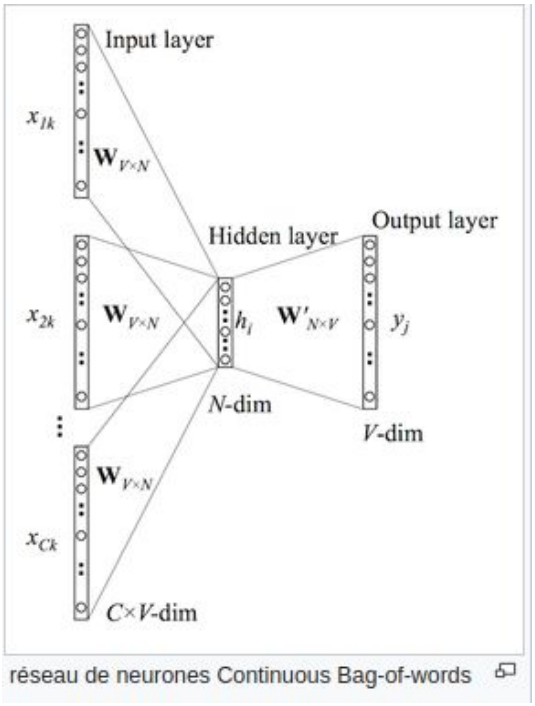
Une autre approche de pré-traitement est ce qu'on appelle **Word Embeddings,** cette méthode permet de convertir les mots sous forme de vecteurs numériques. Parmi les algorithmes les plus connus on trouve :

- **Word2Vec** (Google)**, Fasttext** (Facebook)**, GloVe** (Stanford)**.** Ces approches sont un peu différentes les unes des autres.

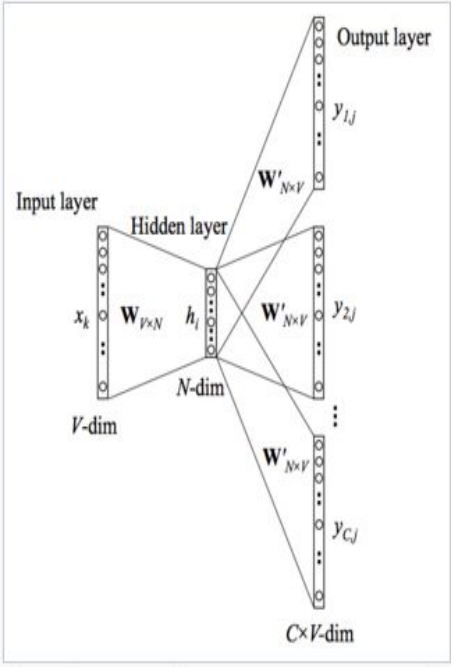
L'idée c'est de construire un réseaux de neurones qui apprend les **embeddings** des mots en reconstruisant la phrase donnée en entrée. En effet, on peut d'entraîner un réseau de neurones qui permet de prédire le mot manquant en utilisant tous les mots autour, et durant le processus d'apprentissage le réseau de neurones apprend et modifie **embeddings** pour tous les mots impliqués. Cette approche est appelé **Continous Bag of Words** ou **CBoW** modèle. On peut aussi utiliser un mot et prédire le context du mot, c'est ce qu'on appelle **skip-gram modèle.**

- **Fasttext :** Cette méthode apprend les embeddings pour des caractères n-grams

- **GloVe :** Cette méthode apprend les embeddings en utilisant des matrices de co-occurence



réseau de neurones Continuous Bag-of-words



réseau de neurones skip-gram

. Algorithmes d'Apprentissage

Nous sommes dans le cadre de résolution d'un problème d'**apprentissage supervisé**, c'est à dire que nos données sont étiquetées, nous connaissons la prédiction (la vraie valeur) pour chaque échantillon. Précisément, nous sommes face à un **problème de classification**, pour cela nous avons plusieurs algorithmes d'apprentissage automatique pour la classification, parmi eux : **k-NN (k plus proche voisins) , Régression Logistique, SVM (Support Vector Machine), Xgboost, Forêt Aléatoire, Arbre de Décision etc …**

2.5- Présentation des approches utilisées

Durant notre travail tout au long du projet, nous avons essayé plusieurs méthodes d'apprentissage : **WordEmbedding, Bag of Words, Réseaux de neurones récurrent (LSTM).**

WordEmbedding : nous avons utilisé des word embeddings pré-entraîné **GloVe** (Global Vectors for Word Representation) mis à disposition par stanford (Cf. Annexe) contenant des vecteurs de 50 dimensions, 6B tokens, 400K vocab.
Nous avons essayé **Fasttext** mis à disposition par facebook (Cf. Annexe) contenant des vecteurs de 300 dimensions, 600B tokens.
Après avoir bien préparé nos données d'entraînement, que nous a stocké en mémoire sous forme de tableau numpy, nous avons appliqué par la suite différents algorithmes d'apprentissage (Cf. Algorithme apprentissage) afin de prédire la phrase d'entrée était bien une tâche d'impression ou une autre.

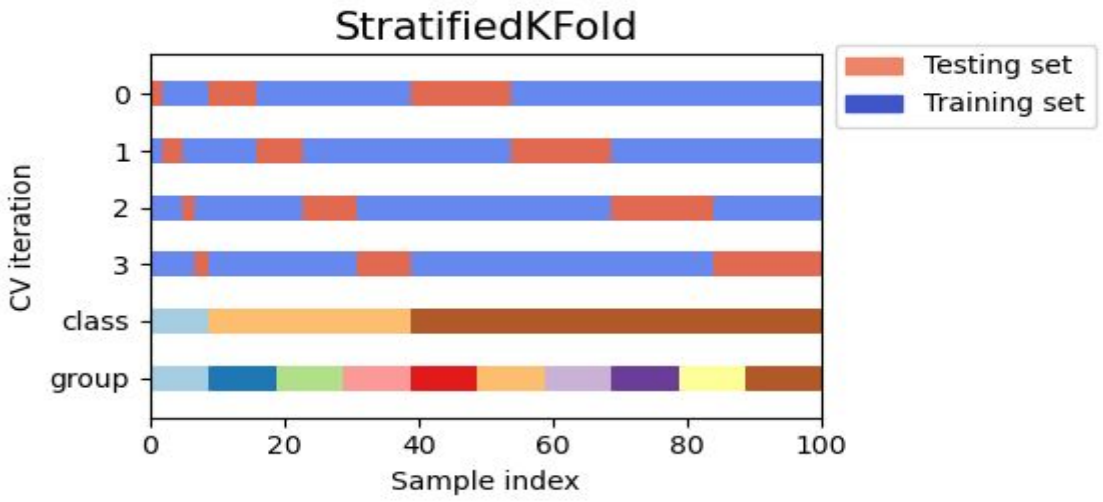
Bag of Words : Nous avons utilisé des méthodes simples de pré-traitement de texte comme CountVectorizer et TfidVectorizer, par la suite nous avons entraîné des algorithmes d'apprentissages.

RNN (LSTM) : C'est un réseau de neurones récurrents adapté aux données séquentielle (Séries temporelles, Texte) qui permet d'obtenir des résultats satisfaisante en terme de performance. Cependant ce type de modèle nécessite beaucoup de données afin de décortiquer en profondeur les spécificités des données d'entraînement pour atteindre une bonne performance tout en évitant **l'overfitting** ou **l'underfitting.** Nous avons utilisé un réseaux de neurones profond à plusieurs couche (Couche embedding, Couche LSTM, Couche Dense ...) et des fonctions d'activation telles que "ReLu" et **Sigmoid en sortie puisqu'on est dans le cadre d'une classification binaire.**

2.6 - Validation Croisée

La validation croisée est une étape importante dans la construction d'un modèle d'apprentissage automatique puisqu'elle nous permet d'être sûr que notre modèle s'adapte à nos données avec bonne précision et aussi de s'assurer que l'ont n'ait pas de problème de sur-apprentissage.

Il existe plusieurs méthode de validation croisée, on trouve parmi elles: **K-Fold**, **Stratified K-Fold**, **Hold-out Kfold**, **Leave-one-out K-Fold**, **Group K-Fold.** Nous n'allons pas détailler chaque méthode, mais nous avons choisi **Stratified K-Fold** qui est la plus adapté à notre cas puisque nous avons des données **non équilibrées.** Cela nous permet dans chaque **Fold** d'avoir la même portion initiale des deux classes (**Impression / Autres**), afin d'éviter d'obtenir des résultats erronés.



2.7 - Présentation des métriques

Il existe 3 types de problèmes en machine learning : Classification, Regression et Clustering. Chaque type de problème possède ses propres métriques qui nous permettent d'évaluer notre modèle. Ainsi on trouve pour la **Régression** (R-squared, Mean absolute error, Mean squared Error ..) , **Classification** (Accuracy, F1-score, AUC, Precision, Recall ..),
Clustering (Adjusted mutual info score, Adjusted Rand score …)

Dans notre cas on va utiliser les métriques de classification et plus précisément F1-score qui est une combinaison de la précision et du rappel.

F1 = 2 * Précision * Rappel / (Précision + Rappel)

avec :

Précision = TP / (TP + FP) et Rappel = TP / (TP + FN)

TP = True Positive : La valeur réelle correspond à celle prédite, on a prédit que c'était une impression alors que réellement c'est bien une impression.
FN = False Negative : Si on prédit que ce n'est pas une impression alors que réellement c'est une demande d'impression

2.8 - Optimisation des paramètres des modèles

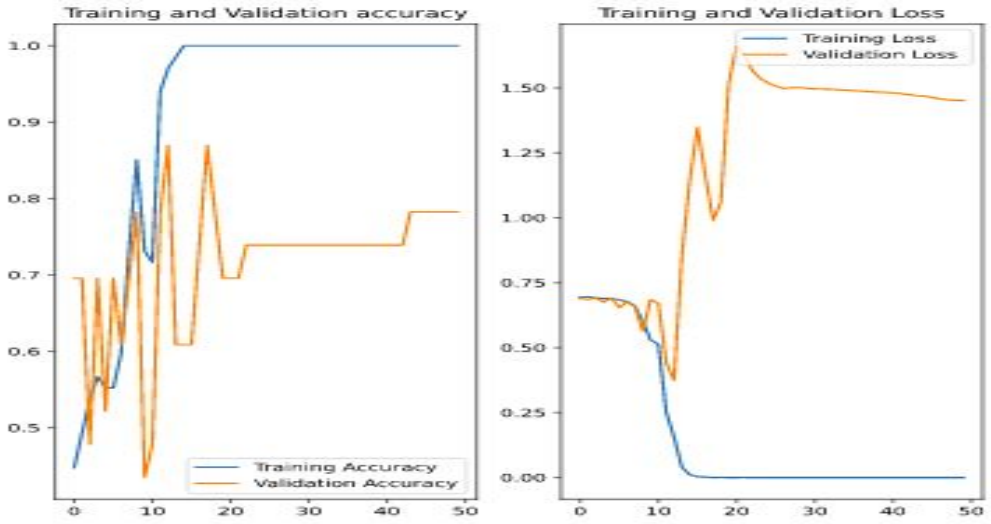
- Pour ce qui est l'optimisation des algorithmes d'apprentissage automatique, on a utilisé un GridSearch qui permet d'essayer toute une combinaison de paramètres et d'obtenir à la fin les valeurs paramétriques permettant d'avoir le meilleure résultat tout en évitant **l'overfitting**.
Parmis ces paramètres on trouve : **Regularization, number of estimators, max_depth, min_samples_split, normalization,class_weight ..**

- En ce qui concerne le modèle de réseaux de neurones récurrents, on a essayé de **changer le nombre de couches, le nombre de neurones pour chaque couches, la fonction d'optimisation, Dropout** ainsi que quelques hyper-paramètres comme **Learning rate**, **Batch size, Earlystopping**

2.9 - Comparaison des différentes approches

Pour choisir le meilleur modèle on s'est basé sur la métrique F1-score, tout en vérifiant s'il y'a un éventuel sur-apprentissage ou sous-apprentissage.

Le modèle de réseau de neurones récurrents overfit les données puisque nous n'avons pas assez de données ou bien que les données ne sont pas représentative. Ce qui se traduit par un comportement assez étrange du **Loss et un écart remarquable entre l'accuracy des données d'entraînement et celles de validation (Cf courbes en dessous)**



Concernant les modèles avec Word Embedding pré-entraîné, nous avons obtenus de bon résultats avec pas d'overfitting ni de underfitting. Par contre, l'importation du fichier contenant le **WordEmbedding** est volumineux ce qui prenait un peu de temps pour le charger en mémoire.

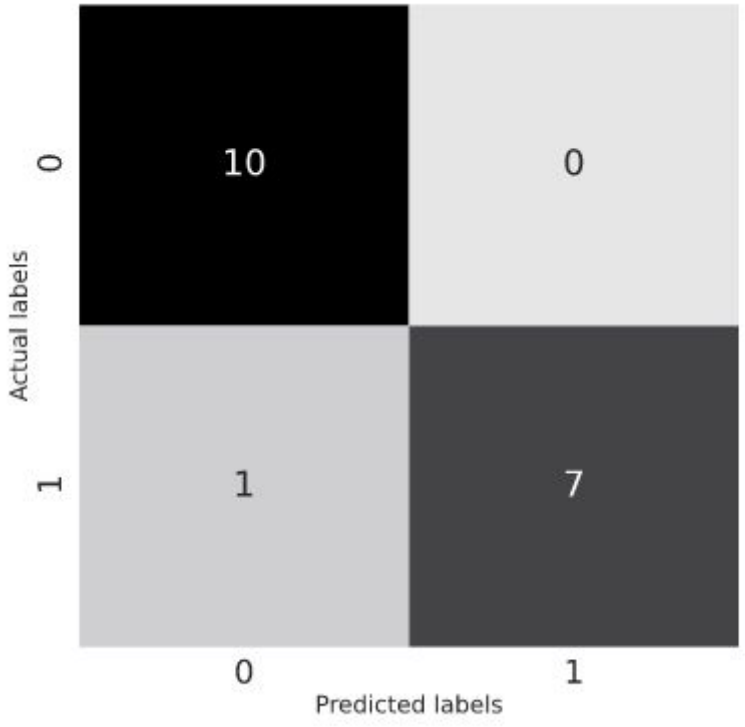
Et finalement la dernière approche dans laquelle nous avons utilisé des algorithmes basiques de pré-traitement de texte (CountVectorizer, TfidVectorizer ..) et que par la suite on appliquait des algorithmes d'apprentissages. L'algorithme de Régression logistique était le plus performant puisqu'on avait que peu de données et qui était le mieux généralisable.

2.10 - Modèle final et résultats

Le modèle retenu est celui utilisé avec CountVectorizer en pré-traitement et l'algorithmes d'apprentissage de Régression logistique pour la classification. Ce modèle ne présentait pas d'overfitting ni de underfitting avec un bon score (Cf image ci-dessous)

```
Fold = 1 , F1-Score _train = 0.9841269841269841
Fold = 1 , F1-Score = 0.9333333333333333
```

Ainsi, on a évalué notre modèle à l'aide d'une matrice de confusion :



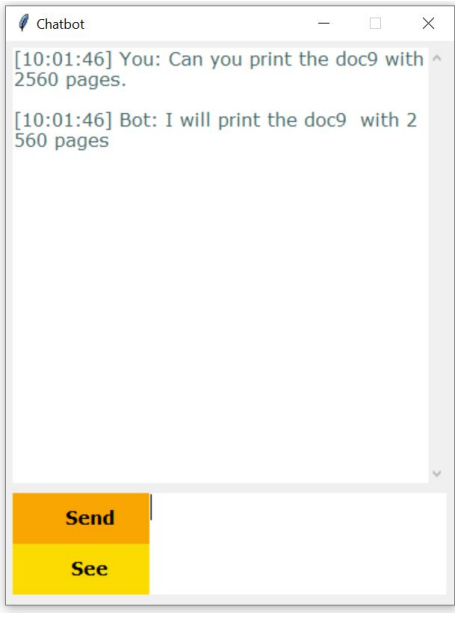
III - Module de gestion de l’agenda

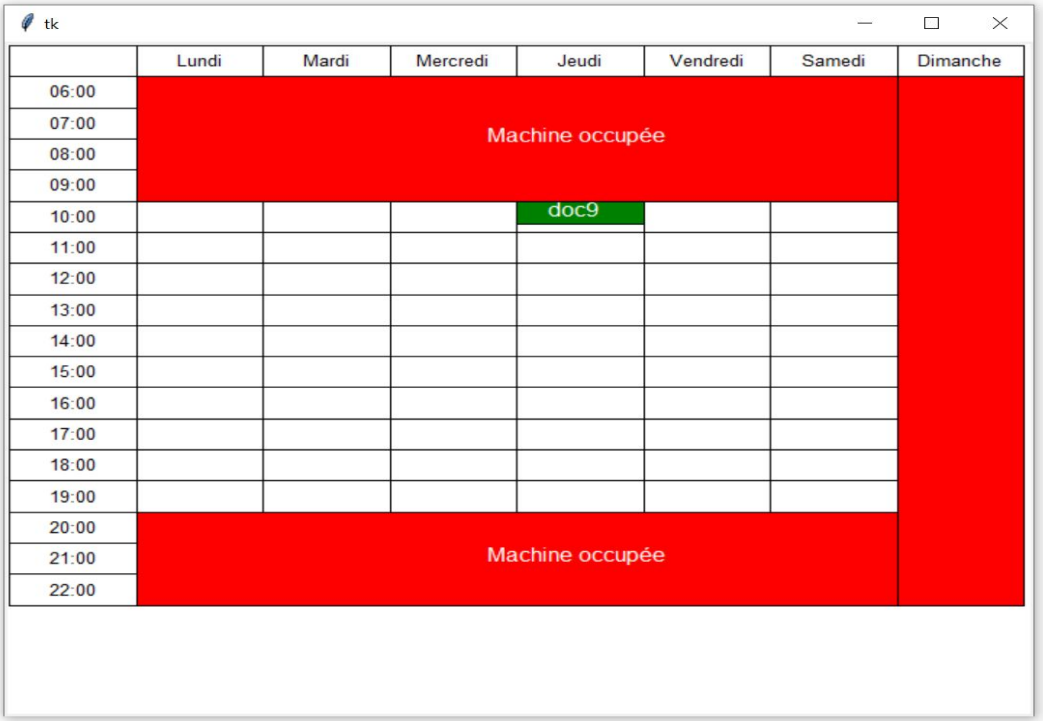
Afin d’ordonnancer correctement la liste des tâches d’impression de la machine, nous avons utilisé les listes chaînées. Pour ce faire, à chaque demande d’impression validée par notre modèle, nous prenions le jour et l’heure de la demande, puis nous cherchions le premier créneau disponible à partir de cette heure là. L’intérêt des listes chaînées était d’optimiser la recherche en parcourant facilement les créneaux, et également d’ajouter facilement les nouveaux créneaux qui ne sont plus disponibles. Nous avons utilisé cette méthode en nous inspirant de la fonction malloc en C/C++.

Nous avons aussi laissé la possibilité de choisir les jours avec l’heure et la durée pendant laquelle la machine n’est pas fonctionnelle.

IV - Présentation de l’interface

Pour ce qui est de l’interface, nous avons réalisé une interface simpliste, permettant d’ouvrir la fenêtre du chatbot et d’interagir avec, à l’aide d’une fenêtre d’écriture permettant à l'utilisateur d’écrire et de deux boutons: l’un pour envoyer la requête, et l’autre pour voir l’agenda des tâches déjà attribuées





V - Conclusion

Pour conclure ce rapport, nous rappelons que le modèle retenu est celui utilisé avec CountVectorizer en pré-traitement et l'algorithmes d'apprentissage SVM pour la classification. Les résultats étaient plutôt corrects au vu du faible nombre de données que nous avions. Pour ce qui est de la recherche d'informations nous n'avons pas pu utiliser NER car nous devions créer un grand nombre de données, nous avons donc utilisé des expressions régulières. Enfin, l'interface est simpliste mais fonctionne correctement.

VI - Annexe

https://nlp.stanford.edu/projects/glove/
https://towardsdatascience.com/different-techniques-to-represent-words-as-vectors-word-embeddings-3e4b9ab7ceb4
https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa
https://fasttext.cc/docs/en/english-vectors.html
https://medium.com/@adriensieg/text-similarities-da019229c894
https://scikit-learn.org/stable/modules/model_evaluation.html
https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html
https://www.nltk.org/