Rapport de stage

Master 2 IA

LAURENT Thomas

Années: 2018 - 2019

Contents

Chapter 1

Remerciements

Au vue de ma monté en compétence que ce stage m'as donné, il en vient d'en remercier tout les acteurs de ce projet.
Commençons par Monsieur DM et Monsieur BM pour m'avoir fait confiance et pour m'avoir accueilli au sein de leur entreprise, à S pour le suivis du projet, au multitudes d'acteurs étant présent lors des réunions afin de pouvoir justifier nos choix de conception ou de les réfuter.

Chapter 2

Introduction Générale

Depuis les années 1950, on parle d'intelligence artificielle (IA, ou AI en anglais) un processus ayant les capacités de penser ou de raisonner comme un processus qui serait humainement exécuté à l'aide d'une conscience ou un réseau neuronal. Un processus pensant pouvant prédire une action, un état ou un résultat. Mais l'intelligence artificielle est en réalité plus complexe, un processus pouvant donner des résultats sur des données qu'elle connait et donc que le processus connait à l'avance le résultat n'est pas un processus d'intelligence artificielle car elle ne fait que pour un dictionnaire de données retourner la valeur associée aux datas donnés.
Pour qu'un processus soit dit, doté d'une intelligence, il faut qu'elle soit équipée d'une capacité de raisonnement voir d'apprentissage pour les cas où les données en entrée soit totalement inconnue du processus, dans ce cas la le processus doit savoir prédire une réponse viable et ayant du sens en utilisant les données qu'elle connait déjà.

Le domaine de l'intelligence artificielle est bien plus vaste que la description donnée ci dessus, selon le *Texte de la 236e conférence de l'université de tous les savoirs donné par Jean-paul HALTON* il existerait trois grandes approches à l'intelligence artificielle, l'approche *Symbolique*, l'approche *Connexioniste* puis l'approche *Statistique*, je vais les décrire ci-dessous.

2.1 L'approche symbolique

Un exemple réel d'approche symbolique serait dans le code de la route, chaque panneau (ou inscription qu'elle soit au sol ou peu importe où) ont une signification spécifique sur l'état que l'automobiliste doit adopter sur certain tronçon de route ou dans un état future. Si un automobiliste voit un panneau ¡sens interdit¿ sur un tronçon juste devant lui, une information est envoyé au cerveau (où plutôt dans la base de connaissance) et une réponse direct (sans aucun apprentissage) est retourné par le réseau de connaissance indiquant qu'il ne faudrait pas aller sur ce tronçon sous resserve d'accidents par exemple.
Cette approche est dite *offline* car avant d'effectuer des requêtes à la base de connaissance celle-ci a besoin d'être au préalable remplit d'informations vrais et de touts les cas possible. Admettons qu'un conducteur n'ai pas apprit la signification d'un panneau ¡sens interdit¿ voyer vous le problème que cet individu peut causer. L'hypothèse du monde clos est appliqué par cette approche, pour une donnée que le réseau de connaissance ne connais pas, la réponse retournée sera une réponse négative, prenons encore notre conducteur ci dessus, si celui ci voit un nouveau tronçon devant lui (un tronçon qu'il n'a jamais vue), il n'a aucune données dans sa base de connaissance qui décrit ce nouveau tronçon (bien-sur nous allons admettre que le conducteur n'a pas la possibilité d'ajouter de nouvelles informations dans sa base de connaissance ni d'apprendre sur ce nouveau tronçon), le conducteur va trivialement dire qu'il ne va pas s'aventurer sur ce nouveau tronçon car, sa base de connaissance n'a pas pu fournir aucune informationes sur ce tel tronçon.

Dans l'informatique, nous le savons, les ordinateurs ne savent que faire des opérations logiques simple comme le ET, le OU, et le NOT, ces opérations sont accompagnés de registres (mémoire vive, disque dur, ...) pouvant pour la durée de la vie d'un processus stocker les données et faire des opérations dessus avec d'autres données stocké dans la machine, c'est de la qu'avec quelques registres et une succession d'opérations logiques simple on peut former des circuits pouvant traiter les opérations mathématiques comme l'addition, la soustraction, la multiplication et la division. Ses informations sont possibles car, elles sont ancré dans la base de connaissance de la machine qui lance les processus, par héritage les processus sont capables d'appeler les circuits pouvant faire les opérations cité ci dessus.
Un exemple logiciel d'approche symbolique serait le langage *Prolog* qui pour

une base de connaissance donné en début de fichier, infère les différentes questions posé dans la suite du programme.
Prenons un exemple simple, pour une base de connaissance donné ci dessous:

```
1 – Homme(Jean)
2 – Homme(Pierre)
3 – Femme(Marie)
4 – En_couple(Jean,Marie)
```

Effectuons des requêtes à la base de donnée ci dessus, demandons:

Es ce que Jean est un homme? la base de connaissance va répondre oui.

Es ce que Marie est un homme? non.

Es ce que Marie et Pierre sont en couple? non.

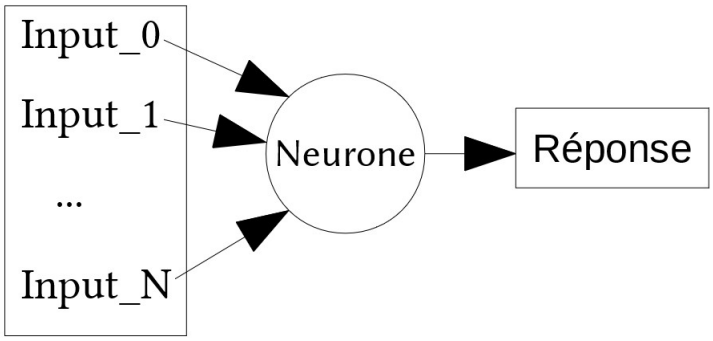
Es ce que Jean et Marie sont en couple? oui.

Pour confirmer la théorie du monde clos, nous pouvons demander si Philippe et un homme, n'ayant aucune informations sur un dénommé Philippe, la base de connaissance va déclencher une exception disant qu'il n'a pas trouvé un tel Philippe, vu qu'il ne connait pas ce Philippe, la base de connaissance va répondre NON.

L'hypothèse du monde clos est une échappatoire fréquemment utilisé dans les bases de connaissances, mais le non envoyé à cause de cette hypothèse n'a pas la même signification que le non Philippe n'est pas un homme. Pour y remédier certains logiciels ont ajouté une valeur à "oui" et "non", nous appelons ceci le *Three valued logic* et ce mode est implémenté dans certaines bases SQL, la troisième valeur est nommé "unknow" ayant la bonne signification demandé par l'hypothèse du monde close.

2.2 L'approche connexionniste

L'approche connexionniste est l'approche la plus proche du schéma du cerveaux au vue de requêtes. Le processus de réflexion du cerveau est représenté par une série de mini processus ne pouvant répondre qu'a un type de problème qui est définit lors de la création de ce mini processus, un mini processus est nommé neurone dans le cerveau, et nous allons préférer l'utilisation du terme neurone pour la suite.
Chaque neurone est définit par plusieurs entrées, et une seul sortie et peu être semblable à un *Perceptron* en machine learning:



En informatique, Chacun de ses neurones sont équipé d'une capacité d'apprentissage qui va influer sur les futures résultats, via un vecteur de variables (dites Poids ou weight en anglais) ce sont ses variables couplé au données d'entré (via un dot) qui va donner le résultat de l'opération, lors de la période de création du neurone, ils se voit attribuer son vecteur de poids à des valeurs aléatoire (plus généralement que des zero), une base de connaissance de référence, une fonction de résultat, et d'une fonction d'erreur pour donner une signification future au vecteur de poids.

Chaque sample (ou ligne) de la base de connaissance de référence est doté d'une variable réponse (noté étiquette) indiquant la réponse que le neurone devrait donner si dans le future on demande au neurone le résultat de ce sample. Avant de rendre le neurone fonctionnel, il se doit de passer les testes d'apprentissages, pour chaque sample dans la base de connaissance de référence, il va le multiplier avec le vecteur de poids et en tirer le résultat via une fonction de résultat et calculer l'indice d'erreur avec le résultat donné et la variable étiquette du sample, via l'indice d'erreur les poids du neurone

vont s'équilibrer jusqu'à converger vers un vecteur dit optimal. Une fois optimal ce neurone donnera les meilleurs résultats qu'il pourra donner.

Les résultats donné par le neurone ne sont pas viable à 100%, il se peu qu'il y ai des faux positif qui passe pour de vrai négatif, mais nous en discuterons plus bas dans ce rapport.

2.3 L'approche statistique

L'approche statistique se base sur statistiques, prenons un exemple d'une grande chaine de distribution, celle ci a des besoin plus ou moins conséquent d'innover pour continuer à avoir une trace dans les têtes des consommateurs, pour s'y faire le staff organise des études sur leurs site internet, ses études sont a complété de chiffres et de choix prédéfinit. Une fois équipé de beaucoup de résultats, il s'en vient de la prise de décision sur les futures changements, l'entreprise munit d'un arbre de décision le remplit selon les valeurs généré par l'étude. pour chaque nœud de l'arbre, l'entreprise va devoir entreprendre une action (peindre, commander, offrir ....), sur les feuilles de l'arbre le gain (ou la perte) prévue en hypothèse, et entre deux nœuds la probabilité calculer avec l'étude.

Une autre approche statistique largement utilisé est via les réseau bayésien. Les enfants qui trient leurs bonbons en fessant une pile de bonbon qu'ils aiment, une pile de bonbon qu'ils n'aiment pas puis une pile de bonbon qu'ils se savent pas quoi en penser de se bonbon. Sur l'arrière du paquet il est indiqué le gout du bonbon par rapport à la forme, la couleur et la taille (l'axiome de bijection n'est pas satisfait). Dans un premier temps l'enfant va pour chaque bonbon qu'il prend lire l'arrière du paquet puis classe le bonbon dans une pile, à force de voir les piles de bonbon grossir, il décide de laisser tomber le paquet de bonbon pour classer la suite de lui même, ainsi pour chaque bonbon prit, il calculera une sorte de probabilité avec ses croyances et l'état actuel des piles. Le cerveau est très complexe pour pouvoir introduire ce raisonnement dans un processus, il existe donc des versions mathématiques du calcule de l'appartenance d'un bonbon à une pile, par exemple *l'entropie* ou le *le gain d'information*.

Selon moi, ses trois approches forme une définition simple de l'intelligence artificiel, mais plusieurs approches qui ne sont pas cité ci dessus ont leur place dans la définition, car rien qu'avec les trois approches ci dessus nous pouvons inférer sur une base de donné en posant des questions simple, généraliser un concept en une série de matrice et prédire des événements selon une base de croyance. Enfin, l'une des approches que je voudrais ajouter est cité un peu plus haut dans l'approche statistique, l'autre approche est aussi basé sur des inférences sur une base de donné, non pour donner une réponse seulement boolean, mais pour fournir tout un modèle en guise de réponse.

2.4 L'approche décisionnel

La prise de décision, une brique très utilisé dans beaucoup de processus se disant intelligent, la où un système de recommandation trie les films selon des probabilités allié à un algorithme de *ranking*, il n'y a rien de vraiment décisionnel. Un processus devant prendre des décisions est un processus qui veut maximiser son gain ou minimiser sa perte. Un joueur d'échec cherche a maximiser son gain pour faire perdre son adversaire, c'est un jeu à somme nul, la somme des gain du gagnant et du perdant donne zero. Pour cela, il nous faut une décomposition des possibilités que les deux joueurs peuvent faire puis construire une matrice qui pour chaque lignes numéroté par un choix que le joueur 1 peut faire, pour chaque colonnes par un choix que le joueur 2 peut faire et pour chaque cellule un couple de valeurs représentant respectivement le gain du joueur 1 et le gain du joueur 2. Via une stratégie inférer sur la matrice pour en déduire les meilleurs possibilité de décision des deux joueurs.

La grande chaine de distribution dans l'approche statistique a réussi à construire un arbre de décision qui pour chaque feuille donne le gain (ou perte), pour chaque node une question, et pour chaque arrête la probabilité de la réponse à la question. Ceci est de la planification, remonter des feuilles jusque la racine en sommant le produit de la feuille et de la probabilité posté sur l'arrête.

2.5 L'approche par satisfaction

Un problème est peut être décomposé en un ensemble de sous problème, prenons une usine qui doit fournir entre 2000 et 6000 écrous et entre 1000 et 7000 vis en métal, l'usine ne peut faire qu'une pièce à la fois quelque soit la pièce, les écrous et les vis sont taillé depuis des petits cubes de métal, mais l'entreprise ne dispose que de 8000 cubes de métal, or pour produire un vis il faut un cube en entier et pour produire un écrous il faut une moitié de cube, l'entreprise voudrait produit autant de vis que d'écrous. Ce problème est plutôt simple si on demande à quelqu'un de l'entreprise qualifié pour le résoudre, mais si nous devons automatiser cette tache à l'aide d'un processus.

On dit qu'un processus est capable de faire de la recherche opérationnel si elle est capable de donner une solution satisfaisante pour un problème combinatoire modélisé en instructions mathématique simple. Une modélisation simple serait de décrire le modèle comme suit:

Soit les variables suivantes: vis, écrous

Maximiser vis (ou écrous)

vis == écrous

$2000 <$ vis < 6000

$1000 <$ écrous < 7000

2 x écrous + vis == 8000

On parle aussi de satisfaction de contraintes ou de problème de satisfiabilité, ses deux processus sont capable de donner des réponses à une majorité de problème et même en donnant une affectation à chaque variables, dans le cas que le processus n'est pas capable de donner une réponse, on dira que le problème est insatisfiable.

2.6 L'approche par représentation et raisonnement

Dans le monde d'aujourd'hui l'un des problème pour des touriste est de comprendre leurs environnements, bien sûr avec une très bonne maitrise de la langue local (ou de l'anglais) le boulot est assez simple, mais pas tout les voyageurs ne savent manipuler la langue local ou la langue de Shakespeare, c'est pour sa qu'il existe des processus pouvant pour une entré sous forme textuel, retourner un autre texte traduit dans une autre langue, quelques fois l'algorithme n'arrive pas à bien traduire la demande, quelque fois le résultat est plutôt acceptable.
Ce problème ci dessus est plus un problème de raisonnement sur le langage naturelle que de représentation, je ne rentrerais pas en détail sur les algorithmes de transformation de la langue naturelle. Mais l'un des problème majeur de la traduction texte vers texte c'est qu'il faut écrire du texte en entrée, voyez vous à chaque coin de rue taper une nouvelle requête, non. Un axe orienté représentation et l'axe de transformation image vers texte, plus techniquement appelé OCR (optical character recognition) qui pour une image retrouve des motifs et les isoles tout sa pour pouvoir donner du texte au traducteur.
Mais le travail ne s'arrête pas la pour toutes les polices (machine ou humain) il faut savoir avec précision quelle parcelle de pixel correspond à quelle caractère, une approche connexionniste et statistique (ou plus généralement Machine learning) est requit pour la transformation de vecteur vers caractère.

Chapter 3

Le Machine Learning

Dans le cadre du rapport de stage, je vais me focaliser sur l'approche *représentation et raisonnement*.

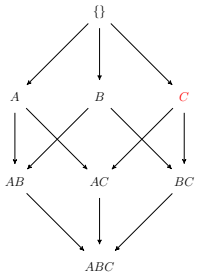
Un *Dataset* est un tableau représentant pour chaque colonne une donnée et pour chaque ligne un *sample*, les samples sont composés d'une et d'une seule valeur par colonne, cette valeur peut être n'importe quoi (chaine de caractère, tableau, entier, boolean, ...). Un des formats les plus utilisés pour illustrer la table est le format *csv*, *SQL* ou *excel*.

3.1 Les premiers pas

Dans les années 1950, on ne parlait pas encore du *Machine Learning* que l'on n'a maintenant, on parlait de méthode de généralisation d'un modèle. Imaginons que vous vouliez prédire un résultat de type boolean à partir de données que vous avez enregistré.

Sans tout ses algorithmes de machines learning actuel, le principe était de généraliser le modèle au maximum,pour ceci avec un trahit puis pour chaque ligne faire l'intersection avec le trahit, tout ceci donne c:

A	B	C	résultat
0	0	1	1
0	1	1	1
1	1	0	0



Cette technique, beaucoup utilisé en fouille de données, a ses limites, quelques années plus tard vient les premiers algorithmes de machine learning qui de nos jours sont encore bien répandu, en voici une liste non exhaustif:

Gradient : Pour un nuage de points, le but est de trouver une droite qui minimise la distance (sur l'axe Y) entre celle ci et chaque points du nuage.

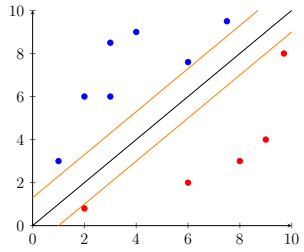
Kmeans : Pour chaque sample du dataset, le but est de former des groupes de samples qui se ressemble le plus.

Ci dessus, deux modèles du machine learning, le Gradient est un modèle dit *Supervisé* et le Kmeans est dit *Non Supervisé*. La différence entre les deux modèles et simple, soit un dataset possédant une colonne nommée *Étiquette* (ou Class en anglais) donne le résultat du sample concerné, il est donc facile pour un sample possédant une colonne Étiquette de connaitre son appartenance, on dit alors que le modèle est supervisé, pour un modèle non supervisé, nous ne connaissons pas les étiquettes des samples.

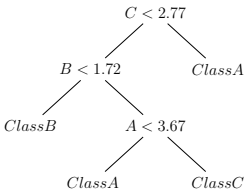
3.2 Algorithmes d'apprentissage

De nos jours il existe une tonne d'algorithme d'apprentissage, tout ont plus ou moins leurs utilités, avantage, faiblesse, d'où le fait que pour le data scientiste doit essayer un peu tous les algorithmes et de juger de leur efficacité via les résultats données. Bien-sûr si un Gradient suffit pourquoi vouloir utiliser un réseau de neurones.
En voici une liste non exhaustive utilisé lors du stage à des fins de teste (via la librairie sci-kit-learn de python):

Support Vector Machine : Le support vector machines cherche un hyperplan (de couleur noir) pouvant départager des classes, Il en existe une infinité d'hyperplan qui peuvent les départager, donc introduisons un autre concept, celui de l'hyperplan qui maximise la séparation entre les deux classes (les droites *Oranges* appelé $Margin$.



Decision Tree : Un arbre de décision est une suite de nœuds relié par au moins une arrête, on emprunte une arrête par rapport à la décision qui est à prendre via le nœuds courant. L'ordre des nœuds est choisis selon des critères (les trois les plus utilisé sont *Entropy*, *Information grain* et *Gini*), ces trois critères de sélection appliqué au même dataset peut donner un arbre diffèrent. Une feuille d'arbre est un nœuds n'ayant aucun fils, les étiquettes des prédictions s'y trouve.



Traiter les samples de type numérique donne beaucoup de possibilités de traiter un problème.
L'évolution a fait que de plus de données sont de type textuel, pour des dates et durées, le problème reste simple à résoudre, pour une suite de mots fini, il suffit de les énumérer, mais pour les données du langage naturel?

3.3 Langage naturel

Un texte peut être long ou court, contenant des liens, des caractères spéciaux, on ne peut pas utiliser lancer un algorithme comme ci dessus sur des textes et en espérer en tirer de bons résultats, plusieurs étapes sont nécessaires pour pouvoir utiliser des algorithmes qui travaillent avec des textes.

3.3.1 Génération de texte

La génération de texte peut être aléatoire ou provenant d'aucune source, mais ce n'est pas le sujet, restons proche des missions abordées en stage. Les deux méthodes de génération de texte ci dessous font appel à beaucoup d'algorithmes de machine learning pour créer un modèle d'interprétation des entrées.

Optical Caracteres Reconization : Le but est de lire les caractères présents sur du support physique, des feuilles de papier, des photos. Après un traitement de l'image, l'algorithme lisant les entrées utilisent des méthodes de *Deep Learning* pour reconnaitre qu'un motif et un caractère. Puis l'algorithme va faire des prédictions vua ce qu'il a apprit, il apprend via des samples représentant des représentations vectorielle des caractères. En pratique *Open CV* et *Sklearn ou tensorflow* sont utilisés.

Speech To Text (STT) : Dans le cadre du stage, nous avons utilisé un de ses algorithmes pour nourrir nos IA, le STT a besoin de beaucoup de données d'entrainement pour pouvoir modéliser le graph du langage souhaité, un sample d'entrainement et tout simplement une piste audio et son transcrit en format textuel, après de longues heures et le graphe généré, nous pouvons entrer de nouvelles pistes audios sans transcrit pour recevoir des prédictions.

3.3.2 Nettoyage des données

Le français est un langage riche en dérivé de lettres notamment pour ses caractères accentués, le premier travail est de réduire l'espace des caractères en éliminent par exemple les majuscules et les remplacer par des minuscules, éliminer la ponctuation, puis normaliser les accents. Nous pouvons obtenir un texte comme suite:

Bonjour, je viens parce-que j'ai une fuite d'eau sur mon plafond, j'ai déjà prit contact avec une entreprise pour ses réparations mais elle n'a pas donné suite.

---

bonjour je viens parce que j ai une fuite d eau sur mon plafond j ai deja prit contact avec une entreprise pour ses reparations mais elle n a pas donne suite

---

Selon la documentation de scikit-learn (librairie python) la méthode la plus optimisée serait le *Bag of words*,pour chaque document i, pour chaque mot du texte w, associer un indice unique j au mot w, et affecter le nombre d'occurrences de w dans $List[i, j]$. Transformer toutes les occurrences en fréquences. Une fois ces étapes terminer, nous pouvons utiliser un classifier pour faire des prédictions sur les textes.
Lors du stage, selon les testes effectué sur presque 4000 corps de mail de longueur descente et nettoyée pour ne laisser que le contenue pertinent (on n'a aussi enlevé les *stopwords* comme les *le,la,d',...*), 2 algorithmes ont eu des résultats correct, les réseaux de bayes naïf (et ses dérivé) et le réseau de neurones.

Concernant des techniques de nettoyage de données, 2 ETL (Extrac Ttransform Load) ont été essayé, Talend et DataIku, j'ai une préférence pour DataIku étant meilleur en python qu'en java, le même travail peut être fait avec ses deux ETL, mais la librairie de DataIku est bien plus remplit et permet de faire la plupart des traitements sur le Data sans écrire une ligne de code, là où Talend le code est quasiment omniprésent. De plus la où Talend s'arrête sur le traitement et la transformation des données, DataIku permet d'exécuter des algorithmes de Machine Learning, tous ses algorithmes sont issu du paquet scikit-learn.

Plus haut j'ai parlé de Machine learning, de traitement des chaines textuels, de génération de chaines textuels, ce qui une fois additionné fait une jolie intelligence artificielle qui pour un besoin peut nous répondre ce qu'il faut. Prenons un exemple présent dans certains films qui se veulent futuriste, tient prenons le film *Her* sortie en 2013.

Pour faire très court et rester dans le sujet du stage, le personnage principal du film nommé Theodore devenais ami avec un système d'exploitation (os) nommé Amy, une os tellement proche de l'être humain, celle ci pouvais apprendre, comprendre et communiquer sans aucun problème avec Theodore. Comme quoi qu'avec un Speech To Text et de bons algorithmes nous pouvons créer un processus qui simule la conscience d'un humain. Prendre par exemple un film n'est pas très pertinent, car tout est réalisable avec de l'argent, revenons dans la vrai vie. Un outil présent dans le monde de L'IOT (internet des objets) qui justement commence à bien se développer, Commencé en 2008 par Google et largement plus connu sous la firme Apple, je parle bien des logiciel ou objets de reconnaissance vocal.

La reconnaissance vocal passe son temps à écouter son environnement et à traduire ce qu'elle peut prendre comme étant de la parole en chaine de caractère, les intérêts ne sont pas multiple, la plus part du temps c'est pour simplifier la vie des utilisateurs, "Quelle sera la Méteo pour demain.", "Comment préparer des cookies", "Appeler Papa". L'utilisateur fait des requêtes à son logiciel qui lui va transformer l'information en chaine de caractère qui lui va faire une requête ailleurs. On peut comparer la reconnaissance vocal comme des requêtes en base de données, la base de données étant le monde.

3.4 Détection d'anomalies

L'intelligence artificiel qui traduit les données ou celle qui fait des prédictions ont une précision linéaire à la taille des données d'apprentissage qu'il a eu, plus les données d'apprentissage sont importante plus l'ia va gagner en précision dans ses fonctions, moins la taille des données d'apprentissage sont petite plus il va faire d'erreurs. Ce procéder est très robuste si on travail avec des données numérique, des valeurs énumératif d'un ensemble, mais avec des chaînes textuels le travail doit être doublement affiné, la où dans les modèles numériques il y a une valeur par cellule, les textes sont souvent composé de plusieurs mots.

La détection d'anomalies en un cas d'utilisation, L'entreprise Coyote est spécialisé dans tout ce qui est préventions des bouchons, travaux et tout autres événements peuvent impacter un conducteur allant d'un poins A à un point B, celui ci étant équipé de l'application Coyote peut être prévenu à tout moment d'un événement qui risque de perturber sa route. Mais le problème étant que l'entreprise doit être assez réactif que l'envoie des données, si un utilisateur est dans les bouchons et que son apli vient tout juste de le prévenir, l'image que l'utilisateur aura du produit ne sera pas positif.
Donc l'entreprise a décidé de faire appelle au Machine Learning pour optimiser la circulation des informations à travers tout le réseaux qu'il couvrait. Pour s'y faire ils ont découpé tout le réseau en petit segments et ont analysé le trafique segment par segment, via une batterie de tests, ils ont pu trouver des segment qui était instable pour le transfert de données. On peut modéliser ce problème en un graphe qui pour chaque nœuds un poids de stabilité est donné et pour une arrête donnant la distance.

En 4 mois ils ont réussi à augmenter la fiabilité de transfert de 9%.

Dans les chaînes textuels plus haut, on a vue que tout les *stopwords* peuvent être supprimé car ils ne représentent pas des caractères pertinent, reprenons le texte plus en haut et enlevons les *stopwords* de taille 1 et 2:

---

bonjour viens parce que une fuite eau sur mon plafond deja prit contact avec entreprise pour reparations mais elle pas donne suite

---

Le text reste encore viable, nous arrivons encore à comprendre ce que le client veut dire, mais il reste encore du bruit, si on supprime les *stopwords* de taille 3, on obtient:

---

bonjour viens parce une fuite eau plafond deja prit contact avec entreprise pour reparations mais elle donne suite

---

Étonnamment le mot "une" n'est pas prit comme un *stopword*.
Une première erreur mineur apparaît à la fin du texte, on passe du fait qu'une entreprise a était contacté mais elle n'a pas donné suite" à "on n'a contacté une entreprise et elle a donné suite". Dans ce cadre, cette anomalie et mineur, car cette partie de la phrase n'est pas pertinente, seul le début de la phrase l'ai.
En suppriment les *stopwords* de taille 4:

---

bonjour viens parce une fuite eau plafond deja prit contact entreprise reparations donne suite

---

On n'est tenté d'encore retirer des mots, retirer la fin de la phrase et le début, par exemple, ce qui nous donnerai:

---

fuite eau plafond

Nous avons optimisé notre exemple au maximum, on ne peut construire une phrase plus petite sans perdre d'informations.

Pour le viens de cette section nous allons créer une anomalie en suppriment le mot "eau", Notre algorithme d'apprentissage a comme étiquette "Dégât des eau" et "Problème de gaz", le motif doit être le bon, si nous appelons un techniciens spécialisé dans le gaz pour une fuite d'eau, alors la prédiction en plus d'avoir été mauvaise a fait perdre du temps et de l'argent à l'entreprise qui a était appelé.

Le problème et le suivant, après nettoyage des données, nous devons crée un algorithme capable de pour une chaîne textuel de retourner la bonne étiquette nommant le mieux parmi une liste d'étiquettes connus.

Le monde de la fouille de donnée pourrait nous aider à résoudre notre problème, revenons à une liste de chaînes textuels de ce type "fuite eau plafond", L'algorithme Apriori peut nous aider à résoudre ce problème mais vu la diversité des données et le fait que plusieurs même motifs peuvent avoir une étiquettes différent, cela ne nous aidera pas. Nous allons préférer partir sur des technologies plus avancé comme *Tensorflow*, *Keras*, en python bien entendu.

Chapter 4

Contexte du Stage

Dans ce chapitre, nous allons introduire l'entreprise d'accueil, le cadre de travail, une petite description des outils utilisés (seulement les outils qui ne méritent pas une trop grande intention), une description plus approfondit de certains outils dans le chapitre suivant, de la méthodologie de travail utilisé puis le sujet de la mission.

4.1 Présentation de l'entreprise

Crée en 2005 par deux frères, l'entreprise _ maintenant basé à _, Étant une société de service, proposant à ses collaborateurs une solution humaine de personnes maîtrisant leurs sujets le tout pour donner une équipe opérationnelle pour toutes réalisations de projets.
Les domaines de compétences vont du classique Java, passant par des standards comme php, docker vers l'incontournable Python. Aujourd'hui nous comptons environ une quarantaine de prestataires de talent couvrant les grands standards du monde numérique dans le quel nous baignons.

4.2 Présentation de la mission

La demande du client est de pouvoir avoir un service pouvant traiter les demandes des clients dans les heures où le standard d'écoute n'est présent, avant ce projet, toutes les demandes passèrent dans la catégorie des astreintes. Notre intelligence artificielle doit simuler une conversation normal tout en dirigent les questions posées par rapport à la problématique du client puis de décider si oui ou non la résultante de l'appelle sera une astreinte ou non.

4.3 Cadre de travail

Le cadre de travail ce situe au Centre de service, dans un open-space assez spacieux, accompagné d'une petite cuisine, et de quelques salles de repos. Les interactions se faisaient soit par Chat écrit, par mail ou via les réunions presque hebdomadaires qui constituait une partie de la méthodologie agile auquel j'étais impliqué dans les projets. Bien entendu, à ne pas oublier le Git de l'entreprise pour toutes les solutions de travail collaboratif.

4.4 Outils de travail

Je vais énumérer les outils utilisées qui ont soit servi à l'analyse des data, la construction des jeux de données, les technologies vus qui n'ont pas étais implémenté et les algorithmes qui ont été testé.

DataIku : Un logiciel qui permet le traitement de données de masse via de nombreux algorithmes présent par défaut allant de la simple transformation de dates (transformation, extraction, opération, ...) en traitement du langage naturel (lower, normalise, tokenise, ...), quelques algorithmes provenant du module python *scikit-learn* comme la régression logistique, foret d'isolation ou un réseau neuronal permettant d'avoir une pré-visualisation de grands algorithmes, ceci a donné lieu à une simplification du choix des algorithmes pour les prédictions.

DeepSpeech : une solution de transformation de la voix en texte proposé par la fondation Mozilla utilisant la librairie python *Tensorflow*.

Spacy : Une librairie python traitent des motifs du langage naturel, l'extraction de sous chaîne utilise la notion de *POST-tagging* et ainsi découper le texte en une classe d'éléments tels que son genre, son lien avec le reste de la phrase, ...

Apriori : un algorithme de fouille de données qui se base sur la notion d'*itemset*, cet algorithme a été abandonné même si dans un premier temps donnait de meilleurs résultats que les réseaux bayésiens naïf lors de la partie ontologie.

Part I

Un chatbot

Le but initial d'un chatbot est un agent pouvant dialoguer avec un utilisateur dans un certain contexte, un contexte est un ensemble de mots donnant lieu à un langage.
Un modèle très basique de chatbot est un simple programme qui récupère clavier de l'utilisateur puis affiche une réponse formaté, les chatbot basique sont pas aussi intelligents, le contexte de la conversation a besoin d'être conservé pour pouvoir donner un meilleur sens et une interaction bien plus intéressante avec l'utilisateur.

Pour qu'un chatbot soit intéressant, celui-ci doit savoir de quoi on parle et savoir donner des réponses pertinentes, sans compter sur la qualité des réponses qui doivent être humainement compréhensible.

Dans cette partie, nous allons voir les technologies utilisées, les algorithmes utilisés, des papiers théoriques et pratiques des recherches sur certains algorithmes utilisés et l'assemblage de tous ses composant pour aboutir à un prototype du projet final.

Chapter 5

Le langage naturel

Le langage naturel n'est pas aussi facile à interpréter qu'une série de valeurs pour les algorithmes de machine learning, c'est pour cela que pour gérer le langage naturel, nous avons d'autres algorithmes que ceux cité ci dessus. Nous parlerons de *Sequence To Sequence*, *Long Short Term Memory*, *Conditional Random Field*, *Named Entities Recognition* ou de réseaux bayésiens naïf.

5.1 Les réseaux bayésiens naif

C'est la première référence que nous avons quand on cherche dans le registre du travail avec les chaînes textuelles.
L'idée du réseau bayésiens naïf est de représenter un ensemble de documents en une liste de fréquences de pairs $(w, |w|)$, w était un mot dans le langage des documents concernés, pour chaque labels, nous allons construire un modèle de probabilité via la formule suivante:

$$P(X|Y = y)$$

Pour classifier un document, nous allons utiliser la formule ci-dessous et retirer le meilleur résultat en tant que prédiction:

$$pred = P(X|y) * P(y)$$

D'après le module *Scikit-Learn*, la variante Multinomial se démarque des autres variantes pour des raisons de fonctionnement, la où les autres variantes demandent des cardinaux des mots, le multinomial fonctionne avec l'algorithme nommé *TF-IDF*, cette algorithme sera introduit sous peu.

Prenons ses deux corpus suivants, extrait de wikipedia. (le label de chaque corpus est représenté par un mot en gras):

Pomme : La pomme est un fruit comestible à pépins d'un goût sucré et acidulé et à la propriété la plus ou moins astringente, selon les variétés. D'un point de vue botanique, il s'agit d'un faux-fruit. Elle est produite par les pommiers.

Automobile : Le terme populaire automobile désigne un véhicule à roues mû par un moteur et destiné au transport terrestre de personnes et de biens.

La procédure d'apprentissage par Multinomial demande deux traitements sur la donnée, la première est nommée $Tokenization$ et la seconde $Frequencies$:

$Tokenization$: Pour une chaine textuelle, nous allons supprimer tous les mots *stopwords* comme $'le', 'la', 'ces'$, ... des mots qui n'ont aucun impacte sur le sens général du texte, si on applique à **Automobile** ceci donnerai:

terme populaire automobile désigne véhicule roues moteur destiné transport terrestre personnes.

$Frequencies$: Pour tous les mots présent dans le texte courant, nous allons lui associer un dans son vecteur, la valeur vide étant zero. Pour que le tableau ne soit pas très large, nous allons prendre ces deux textes suivant:

1: gare train gauche gauche magasin

2: chaussures rangé haut magasin

Ce qui donne:

–	chaussures	gare	gauche	haut	magasin	rangé	train
1:	0	1	2	0	1	0	1
2:	1	0	0	1	1	1	0

Pour avoir une implémentation complète venant de $scikit-learn$, la $tokenization$ sera décrit par l'algorithme $CountVectorizer$ et la $Frequencies$ par $TF-idf$.

5.2 Conditional Random Field

Un autre modèle statistique, mais qui cette fois ne s'arrête pas à un simple encodage des variables, mais à de l'extraction de sous chaînes, mais prend aussi en compte les variables voisines, pour correctement illustrer, prenons trois mots d'une phrase, comme "*pour correctement illustrer*", et prenons le mot "*correctement*", ce mot sera inséré dans une structure de données ayant comme champs:

lower : le mot en minuscule (donc "*correctement*")

digit : un boolean disant si le mot est un nombre ou pas

title : si ce mot est un titre (sous la forme capitalisée)

trois champs -1 : qui lui contient les trois champs du dessus, mais avec le mot $n - 1$

trois champs +1 : qui lui contient les trois champs du dessus, mais avec le mot $n + 1$

Ce modèle largement utilisé lorsque qu'on veut traiter le langage naturel donne d'assez bon résultats.

Pour appuyer la qualité des résultats, en 2018, l'entreprise Google a développé un algorithme nommé **BERT** un outil de pré traitement du langage naturel qui a significativement amélioré les algorithmes de traitement du langage naturel. Dans le cadre du CRF nous utilisons le mot $n + / - 1$ pour n, la où **BERT** prend en compte aussi les mots $n + / - 2$, mais ce n'est pas tout, quand l'algorithme devait être testé, celui-ci devait deviner un mot en index n de la phrase en ayant que ses quatre mots, les résultats étaient correct sans plus, donc en amélioration ils ont finalement laissé le pouvoir à l'algorithme de pouvoir lire la phrase en entier du sens normal (de gauche à droite) et aussi de lire la phrase de droits à gauche, ainsi de considérablement augmenter la précision des réponses.

5.3 Named Entities Recognition

Dans la catégorie de l'extraction de données, l'extraction des entités nommées consistent à extraire un groupe de mots pouvant décrire des mots clef de type Entreprise, Localisation comme des villes ou des noms...
Un exemple de base serait:

la firme Mafirme était présent le mois dernier à Paris pour tenter de gagner 100 000 euros.

Mot	index début	index fin	catégorie
Mafirme	9	15	Entreprise
Paris	49	54	Localisation
100 000 euros	78	91	Argent

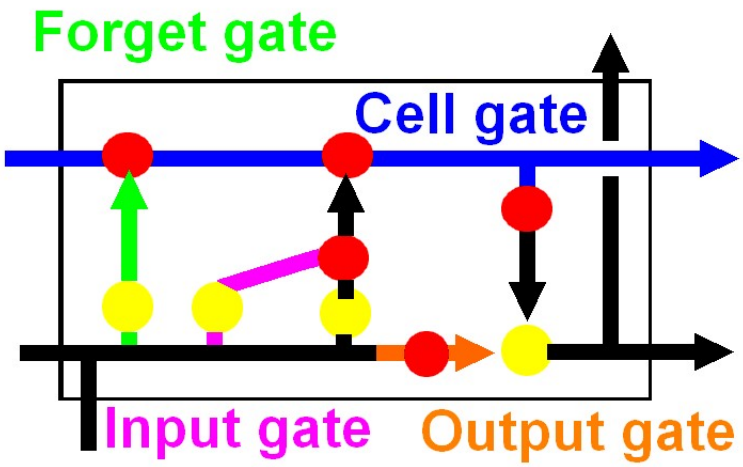
En important un modèle personnalisé notre NER pourrait avoir d'autres utilités, comme par exemple, nous voulons que notre bot sache répondre dès qu'il a compris le problème, nous utilisons un NER personnalisé avec une cinquantaine d'exemples afin que notre modèle sache répondre correctement, il bot sait extraire le sujet d'une phrase, par exemple:

Client : Depuis cinq jours le tuyau de mon évier de salle de bain fuit à grosse goutte

Bot : tuyau de mon évier de salle de bain fuit

5.4 Long Short Terme memory

Commençons les gros algorithmes avec le LSTM, il fait partie de la famille des *Recurrent Neural Network (RNN)*. Les réseaux de neurones récurrents peut être vu comme des combinaisons de circuits imprimés (comme des nos ordinateurs) relié ensemble par de petits ponts de transferts de données. Un circuit est représenté par cette illustration suivante:



Forget Gate : Prenant en entrée le mot à l'indice N et l'information du circuit $N-1$, la fonction sigmoid va décider si le circuit va oublier ou garder le mot courant.

Input Gate : Ce bout de circuit fait appel aux fonctions sigmoid et tangente, le produit des résultats va déterminer l'importance du mot courant.

Cell State : Cette partie du circuit va calculer la variable de sortie du circuit.

Output Gate : Cette dernière partie du circuit va calculer pour le circuit suivant si le prochain mot doit être conservé ou oublier par rapport au résultat donné tout au long du circuit courant.

LSTM est comme Seq2Seq, dans son fonctionnement général ils sont égaux, mais LSTM peut couvrir plus de choses que Seq2Seq qui lui a plus une utilité FAQ (pour les Foires Aux Questions il est très adapté) tandis que LSTM peut vraiment tout couvrir, le seul défaut c'est qu'il demande énormément de données, il a un temps d'apprentissage énorme et demande des cœurs GPU (carte graphiques) pour un gain de vitesse considérable.

5.5 Sequence To Sequence

Le modèle Seq2Seq est très puissant, sa structure est donnée par deux colonnes, la première étant n'importe quoi de type texte et la sortie étant aussi n'importe quoi de type texte. Les exemples simples donnés sur le net sont un traducteur anglais vers français ou un générateur de réponses à des questions.

Ce processus demande de connaitre à l'avance tous les caractères utilisées pour ainsi transformer chacune des lettres en indice numérique unique, tous les mots seront ensuite remplit de caractère dit "vide" pour avoir l'unicité de la longueur des mots.

j	0
e	1
s	2
u	3
i	4
l	5
a	6

je: 01

suis: 2342

la: 56

-	j	e	s	u	i	l	a
0	0	0	1	0	0	0	0
1	0	0	0	1	0	0	0
2	0	0	0	0	1	0	0
3	0	0	1	0	0	0	0

Le *Seq2Seq* fait partie de la famille des réseaux neuronales, celui-ci fonctionnant comme le `LSTM` donc ayant une couche d'encodage et de décodage. Dans les travaux théoriques, cet algorithme peut aussi servir à raccourcir les longs textes comme des histoires, en un résumé sans trop perdre le sens de celle-ci.
D'autres travaux, plus dans l'ordre de la recherche a permis le fait que *Seq2Seq* pourrait probablement servir comme traducteur *Text To Speech*.

Pour résumer, les réseaux naïves sont des algorithmes qui pour des entrées de types chaînes textuelles retourne une catégorie, Seq2Seq et LSTM sont des algorithmes de générations de chaînes textuel, CRF est utilisé pour la prédiction de mots, selon une chaînes de mots passé en entrée puis NER va extraire des motifs depuis une chaîne textuel et lui attribuer une étiquette.

Ces cinq algorithmes seront utilisés dans les futures api utilisées lors de l'élaboration du chatbot.

Chapter 6

Rasa le chatbot

Dans ce chapitre nous allons aborder le moteur même du chatbot, l'api RASA qui met à disposition un bot vierge configurable et bien réalisé. En description générale, RASA se décompose en deux parties, l'unité NLU et l'unité CORE, l'unité NLU peut être utilisé sans unité CORE, mais le CORE a besoin du NLU pour fonctionner (Ces deux termes seront expliqué plus bas). La mise en place de cette brique est cruciale, car c'est cette api qui devra répondre au mieux au attentes du client.

6.1 le NLU

NLU où *Natural Language understanding* est la brique de compréhension du langage naturel, cette brique fonctionne sous une pipeline. Une pipeline est un enchainement de composants qui va traiter la donnée textuelle qui a été reçu de l'utilisateur.

Je vais énumérer toutes les composants que j'ai utilités afin de réaliser la pipeline:

WhitespaceTokenizer : Dans un premier temps, notre texte doit être formaté en une liste de mot, une liste de mot sans caractères spéciaux ni url.

CRFEntityExtractor : Ajouter aux explications sur le CRF, le *CRFEntityExtractor* en ai une dérivée à la différence que le CRFEE va nous générer un modèle donnant une liste de mot intéressant. La liste des mots intéressants sont crée lors du processus d'apprentissage du NLU, qui pour une série de phrases et une série d'index de sous phrases dit intéressant.

EntitySynonymMapper : Le corpus de mot généré via le CRFEE peut être factorisé en moins d'entrées, c'est le travail de ce composant.

CountVectorsFeaturizer : le *CountVector* va comme indiqué plus haut dans la section parlant des réseaux bayésiens naïf, va représenter une chaîne textuelle en un vecteur d'entiers.

EmbeddingIntentClassifier : ce processus fait appel à l'apprentissage supervisé, pour une série de catégories déterminé par le fichier *nlu.md* (qui sera présenté plus bas) et un vecteur précédaient crée par *CountVector*.

Un exemple serait mieux histoire d'illustrer ces cinq composants, leurs effets sur les entrées et les structures de données demandés.
La structure de données pour l'apprentissage sont représentés au format *Markdown*, toutes les étiquettes sont représentés par des titres de niveau 1, et tous les entrées sont représentés par une liste à puce.

```
#intent:ClassA
- Input1
- Input2

#intent:ClassB
- Input3
- Input4
- Input5
```

Ceci représente la structure de base utilisée pour la pipeline, les *InputX* sont tous les textes d'entrées, ils peuvent être de type entier, chaine de caractères, des phrases, les *ClassX* représente l'étiquette du groupe d'entrée.
Prenons, deux étiquettes pour deux phrases à classifier:

```
#intent:Voiture
- Avec ces quatre roues motrice et son nouveau moteur, ce bolide va probablement
    passer les 100 kilometres heurs dans les virages.

#intent:Alimentaire
- Ce pommier va bientot nous donner de belles pommes.
```

WhitespaceTokenizer

Pour tout les mots, nous allons lui associer une structure stockant, la position de la première et de la dernière lettre.

Input: Ce pommier va bientot nous donner de belles pommes.

Output: $[Ce, 0{-}2], [pommier, 3{-}10][va, 11{-}13][bientot, 14{-}21][nous, 22{-}26][donner, 27-33][de, 34-36][belles, 37-43][pommes, 44-50]$

CRFEntityExtractor

Pour un mot donné, remplir tous les champs demandé, le champ -1 est le mot précédant (si il existe), et le champ $+1$ est le champ suivant (si il existe). Le champ $prefixX$ est représenté par les X premières lettres d'un mot, le champ $suffixX$ est représenté par les X derniers caractères.

Input: Pour le mot *belles*

Parsed Input: $\{'-1 : low' :' de','-1 : title' : False,'-1 : upper' : False,'0 : bias' :' bias','0 : low' :' belles','0 : prefix5' :' belle','0 : prefix2' :' be','0 : suffix5' :' elles','0 : suffix3' :' les','0 : suffix2' :' es','0 : upper' : False,'0 : title' : False,'0 : digit' : False,'1 : low' :' pommes','1 : title' : False,'1 : upper' : False\}$

L'algorithme utilisé pour le *CRFEE* est la chaîne de Markov où chaque arrêtes est étiqueté par un mot et les nœuds par des étiquettes, les autres paramètres vu plus haut donnent des probabilités d'appartenance sur certaines étiquettes.

EntitySynonymMapper

Fondamentalement pour la classification de textes, cette brique n'est pas utile, mais elle sera utilisée plus tard lors de la section sur les Slots.

CountVectorsFeaturizer

Précisé au-dessus dans la section *CountVectorsFeaturizer*, notre phrase se retrouve convertit en un très grand vecteur de bit.

EmbeddingIntentClassifier

Cet algorithme va classifier notre phrase précédemment vectorisé, la base de cet algorithme est *StarSpace* qui d'après l'université Cornell à New York peut résoudre tout les problèmes de classification de textes.
Grâce à cet algorithme, nous nous retrouvons avec la liste des probabilités associées à chaque étiquettes:

```
1 {'name':'Voiture', 'confidence': '0.10'}
2 {'name':'Alimentaire', 'confidence': '0.90'}
```

6.2 Les slots

Reprenons notre fichier *Markdown*, à nos entrées, nous pouvons introduire des variables, les variables sont emplacements dans le coeur du chatbot pour retenir des informations, ainsi pour aider à la compréhension de la conversation, les variables sont introduit dans les Inputs.

```
#intent:Voiture
- Avec ces quatre [roues motrice](automobile) et son nouveau [moteur](automobile), ce
    [bolide](automobile) va probablement passer les 100 [kilometres heurs](automobile)
    dans les [virages](automobile).

#intent:Alimentaire
- Ce [pommier](alimentaire) va bientot nous donner de belles [pommes](alimentaire).
```

Lors de la phase de classification (*EntitySynonymMapper*) c'est au tour du NER d'extraire ces termes et ainsi automatiquement stocker des informations ciblé sur des axes clef de la conversation dans le cœur du chatbot. Plusieurs variables peuvent être utilisé dans une phrase:

```
- Avec ces quatre [roues motrice](equipement) et son nouveau [moteur](equipement), ce
    [bolide](appellation) va probablement passer les [100 kilometres heurs](vitesse)
    dans les [virages](typeroute).
```

Ce qui donne après exécution de *EntitySynonymMapper*, une liste de mots avec leurs appartenance:

```
[{'start': 46, 'end': 51 'value': 'moteur', 'entity': 'equipement', 'confidence':
    0.8474657889627953}]
```

La validation des slots peut se faire dans les **Actions** (qui seront expliqués dans la section Actions), ce-ci permet de contrôler les affectations dans les slots, par exemple si on veut une identifiant numérique, alors nous pouvons faire une condition sur le slot en question et lui dire de ne pas accepter la valeur si elle n'est pas numérique. Dans un autre sens on peut ré affecter la valeur comme par exemple dans notre cas, utiliser la phrase pour faire une prédiction puis insérer la prédiction à la place de la phrase initiale.

Chapter 7

Le Core

Comme indiquée plus haut, le chatbot se décompose en deux grande brique, l'une étant le *NLU* et l'autre le *Core*. Le core est le cœur de chatbot, c'est cette partie qui s'occupera de réfléchir à la réponse qu'il faudra donner à l'utilisateur, c'est aussi là où le concept de *Slots* sera utilisé, le Concept *D'actions*, les *stories*, *domaine* puis les *formulaires*.

7.1 Le Domaine et les stories

Si le fichier *nlu.md* est l'entrée de l'algorithme, alors le fichier *domains.yml* en ait la sortie, ce fichier répertorie toutes:

- les slots utilisés
- les intents utilisés
- des réponses près définit

Les réponses près définit sont des phrase que le bot va envoyer à l'utilisateur lorsque le nom associé à la phrases près définit est déclenché soit via la *Storie* ou soit via les *Actions*, (les *utter_* et *utter_ask_* sont des conventions:

```
templates:
  utter_oui:
  - text: "Vous avez dit Oui!"

  utter_non:
  - text: "Vous avez dit Non"

  utter_ask_demander:
  - text: "Voulez vous une glace?"
```

Les *utter_* décrit la phrase comme étant une réponse alors que les *utter_ask_* décrivent plutôt une question. Les stories font le lien entre le fichier *nlu.md* et le fichier *domains.yml*, celui-ci, au format *Markdown* donne une ligne directeur concernant la conversation, par exemple pour notre marchant de glace il ressemblerait à:

```
## story_je_veux_une_glace
* intent_bonjour
  - utter_bonjour
  - utter_ask_glace
* intent_oui
  - utter_oui

## story_pas_de_glace
* intent_bonjour
  - utter_bonjour
  - utter_ask_glace
* intent_non
  - utter_non
```

Deux scénarios sont possibles, soit vous dites *Oui* à la glace soit *Non*, mais ce qui est sûr c'est que vous devez dire bonjour et que le vendeur doit vous demander si vous voulez une glace. Le changement de *story* se fait via deux facteurs qui sont l' *EmbeddingIntentClassifier* et les *Slots*.

7.2 Les Actions et les Formulaires

Pour l'instant, notre Chatbot permet de réponde simplement à des phrases via des réponses près définit, le concept d'action et de personnaliser l'action que le bot va réaliser. Plus haut nous avons vu que les slots sont des variables qui sont extraits des entrées de l'utilisateur, nous pouvons utiliser ces fameux slots pour diriger le chatbot dans une direction précise.

```
Stories.md:

## story_je_veux_une_glace
* intent_bonjour
 - utter_bonjour
 - action_conseil_glace_du_moment
 - utter_conseil_glace_du_moment
```

```
Actions.md:

class Action_conseil_glace_du_moment:

  def run(self, domain, tracker, dispatcher):
    prediction = Predire_la_glace_du_moment()
    set_slot("glace_du_moment", prediction")
    return []
```

```
domain.yml:

templates:
 utter_conseil_glace_du_moment:
 - text: "Nous vous conseillons la {glace_du_moment}"
```

Avec ce processus nous pouvons exécuter une tâche externe comme appeler des algorithmes d'apprentissage sur les données de l'utilisateur. Les formulaires sont un moyen d'automatiser le remplissage des slots, tous les slots qui seront insérés dans un formulaire seront obligatoirement à remplir. Le Core va avec les slots présents dans sa base va générer une liste de slot à remplir, tant que cette liste n'est pas vide, le chatbot va poser des questions afin que l'utilisateur puise donner des informations requit.

Chapter 8

l'api de prédiction

La brique du chatbot étant fini, reste la partie Machine Learning, j'ai décidé de séparer le chatbot des programmes d'intelligence artificielle et de placer tous les programmes dans une API qui est obtenable via requêtes *REST*, ainsi, l'implémentation du projet se fait beaucoup plus facilement, d'un coté tout les programmes de prédictions et de l'autre le chatbot.
Les requêtes à l'api de Machine Learning se fait dans les Actions du chatbot.

Pour aboutir à un chatbot qui répond plutôt bien aux attentes des utilisateurs, nous devons d'extraire la nature du problème (*nature_probleme*) les équipements concernés (comme les murs, fenêtres, grillage) (*equipement*) et les demandes techniques (une réclamation, renseignement, incident) (*demande_technique*).

8.1 Les ontologies

Une ontologie est un ensemble de concepts qui donnent un sens à un groupe de mots, dans le chapitre précédant lors de l'introduction des slots, il y a eu une petite notion d'ontologie, dans le fait que pour une phrase on extrait des mots pour les associer à des slots.
Mais l'implémentation de la déduction des variables demandées (*nature_probleme*, *equipement* et *demande_technique*) est plus complexe. Pour un corpus de mots construits à partir de huit Gigabit de données clients et de quelques programmes d'automatisation de l'extraction et d'analyse fait par nous-même, nous avons pu construire les corpus comme suit:

```
1 Mots_Clef_Demande_technique.csv
2
3 Clef,Synonyme
4 adaptation,"adaptation,adaptations,adapter,adapte,ajustement,ajustements,ajuster,..."
5 effraction,"effraction,pillage,piller,pille,vol ,voller,voleur,vol avec effraction, ..."
6 epave,"epave,epave,epaves,delabre,delabrer,ruine,debris, ..."
7 deratisation,"deratisation,deratisation,presence de rat,des rats"
8 ...
```

Pour chaque mots dans les synonymes, on lui associe sa clef, ce qui donne par un exemple:

```
1 str = "Il y a une epave de voiture qui est devant une maison abandonne depuis
      longtemps, ce matin des rats sont venue m'attaquer."
2 print(ontologies(demande_technique, str)
3 > ["epave", "deratisation"]
```

Nous avons résumé en ontologie la phrase donnée pour en obtenir une série de mot clefs, ainsi coupler avec un algorithme d'apprentissage travaillant sur les chaînes de caractères, comme un réseau bayésiens naïf nous obtenons le type de *demande_technique* associé à la phrase posé ci-dessus:

```
Demande_technique_train.csv

Class,Input
adaptation,"[adaptation]"
amenagement,"[voirie]"
maintenance,"[maintenance,reclamation]"
incident,"[degradation,garantie]"
reclamation,"[change,degradation,reclamation]"
occupation_vide,"[epave]"
```

Nous comptons respectivement pour les trois ontologies (*nature_probleme*, *equipement* et *demande_technique*), au moins deux cents synonymes (par fichier) pour respectivement 30, 52 et 23 clefs pour les fichiers des mots clefs et respectivement 975, 1242 et 156 lignes d'apprentissage pour l'algorithme de réseau bayésiens naïf.
A l'aide d'un de nos programmes, nous avons pu faire un estimateur sur le pouvoir de notre ontologie, nous avons semi automatisé la classification des huit Gigabit de transcrits et nous avons appelé l'entièreté des phrases utilises (donc des phrases autre que "oui","bonjour", "au revoir"...) sur notre api, les résultats pour *demande_technique* ne sont pas mauvais

```
comportement_invervenant: OK:    0 | NOK:    0 | TOTAL:    0 | RATIO:0.00000
                securiter: OK:    1 | NOK:    1 | TOTAL:    2 | RATIO:0.50000
            contestation: OK:  127 | NOK:   85 | TOTAL:  212 | RATIO:0.59906
           amenagement: OK:  373 | NOK:  154 | TOTAL:  527 | RATIO:0.70778
             abandonne: OK:   68 | NOK:   27 | TOTAL:   95 | RATIO:0.71579
           maintenance: OK:  130 | NOK:   42 | TOTAL:  172 | RATIO:0.75581
           reclamation: OK: 2126 | NOK:  678 | TOTAL: 2804 | RATIO:0.75820
             adaptation: OK:   70 | NOK:   12 | TOTAL:   82 | RATIO:0.85366
      eclairage_collectif: OK:   51 | NOK:    8 | TOTAL:   59 | RATIO:0.86441
                 epave: OK:   14 | NOK:    2 | TOTAL:   16 | RATIO:0.87500
              effraction: OK:   27 | NOK:    3 | TOTAL:   30 | RATIO:0.90000
                 voirie: OK:  411 | NOK:   35 | TOTAL:  446 | RATIO:0.92152
           intervention: OK:  743 | NOK:   55 | TOTAL:  798 | RATIO:0.93108
            degradation: OK:  282 | NOK:    8 | TOTAL:  290 | RATIO:0.97241
           information: OK: 4999 | NOK:  107 | TOTAL: 5106 | RATIO:0.97904
                change: OK:  691 | NOK:   10 | TOTAL:  701 | RATIO:0.98573
              garantie: OK:  530 | NOK:    7 | TOTAL:  537 | RATIO:0.98696
              incident: OK:  403 | NOK:    3 | TOTAL:  406 | RATIO:0.99261
         deinsectisation: OK:    1 | NOK:    0 | TOTAL:    1 | RATIO:1.00000
             non_resolu: OK:    2 | NOK:    0 | TOTAL:    2 | RATIO:1.00000
           amelioration: OK:   21 | NOK:    0 | TOTAL:   21 | RATIO:1.00000
                 squat: OK:    5 | NOK:    0 | TOTAL:    5 | RATIO:1.00000
           deratisation: OK:   16 | NOK:    0 | TOTAL:   16 | RATIO:1.00000
```

8.2 Les algorithmes annexes

Dans notre API, je viens d'expliquer la brique la plus importante, mais je voudrais aussi évoquer les autres *end-points* qui ont leurs intérêts dans la construction du projet:

Normalizer : traitent toutes chaînes avant tokenisation en leurs enlevant les url, les caractères spéciaux, les accents et les majuscules.

Tokenizer : Qui pour un corpus de mots clefs, (voir un exemple ci-dessus) transforme la phrase précédemment normalisé en un ensemble de mots clefs

un_une : qui pour un mot, donne sont genre ("un" ou "une"), utilisant un algorithme de réseau bayésiens naïf.

un_des : qui pour un mot, prédit son article (du,de la,des,l') afin de construire des phrases le plus humainement compréhensible.

astreinte : qui pour un motif, un équipement et une nature problème précédemment prédit va prédire si une astreinte est requit pour le problème.

ner : Qui pour une phrase du client retourne que la partie intéressante (voir dans la section NER au-dessus)

"J'ai une fuite dans ma salle de bain et mes enfants jouent dans le jardin"

"une fuite dans ma salle de bain"

ner_conjugate : qui pour une phrase extraite depuis le NER va changer les mots donnant L'impression que le bot répond normalement à une phrase:

"une fuite dans ma salle de bain"

"votre fuite dans votre salle de bain"

pos_neg : qui pour une réponse donnée sous forme d'une phrase, prédit si la réponse est positif ou négatif.

Chapter 9

Fonctionnement de l'application

Toutes les pièces du puzzle ont été introduit, il ne manque plus qu'a les assembler pour construire le chatbot. Comme expliqué ci-dessus, les algorithmes de prédictions seront tous dans l'api via le protocole REST, le chatbot sera équipé d'une brique NLU et CORE, un formulaire d'action qui va s'occuper de remplir les slots et appeler l'api de prédiction. Dans ce chapitre nous allons voir le bot en action.

Voici une interaction avec le chatbot:

user : bonjour

bot : Bonjour je suis marie, quelle est votre soucie

user : voilà je vous appel car ma chaudière est en panne

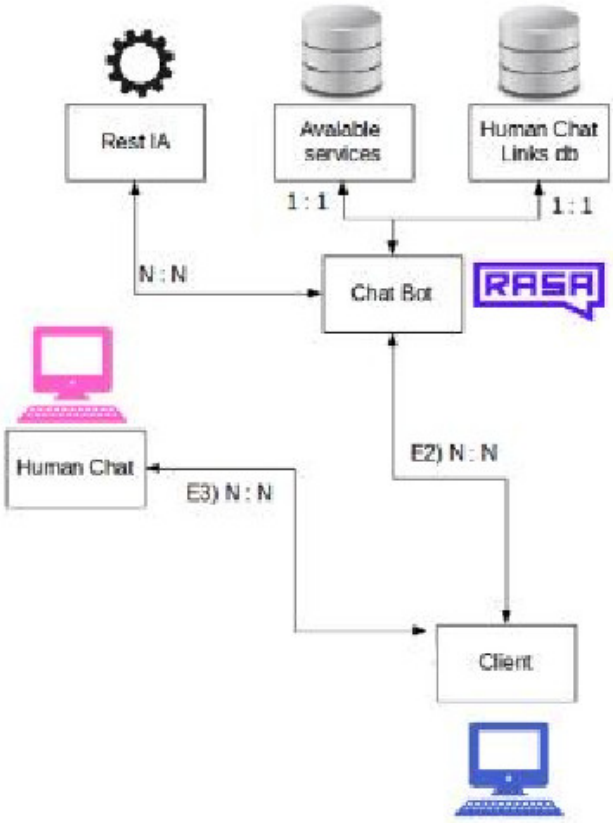
bot : Parler moi plus en détail de votre chaudière qui est en panne, si vous en avait

user : elle ne fonctionne plus quand je l'allume

bot : Je vais faire monter l'information au service concerné qui vous rappellera

bot : au revoir.

Le chatbot possède aussi un port serveur, les appelles se font via le protocole REST entièrement en *POST*.
Le premier souci du chatbot c'est que nous ne pouvons pas faire passer à travers le chatbot une interaction entre deux clients sans que celui-ci ne consomme les données, donc il a fallu un programme fessant office de routeur pour les conversations de l'utilisateur, soit le schéma suivant:



Lors de la phase de développement du projet, le programme routeur était un programme python, qui attendait un certain paquet provenant du bot pour pouvoir changer l'url d'envois des données vers un service client. Une fonctionnalité en plus est que le bot à un annuaire (noté par la base de données *Human Chat Links db* qui redirige le client vers tel endpoint de telle adresse de tel service.

9.1 Implémentations futures

Dans ce qu'il reste du projet, nous devons brancher le chatbot sur une interface pouvant être compatible avec un téléphone, donc nous devons avoir une brique transformant la voix en texte et inversement, nous allons utiliser *DeepSpeech* la librairie de Mozilla qui va nous permettre via des algorithmes Tensorflow et Seq2Seq de générer un modèle répondant à nos besoins.

Chapter 10

Conclusion

Ce fut un stage riche en découverte, une bonne continuation des cours de machines learning proposé à la faculté, un branchement plus que parfait. Lorsque qu'à la fac nous apprenons les bases du machine learning, de la fouille de données, et quelques implémentations avec des datasets plus orienté numérique, le stage s'est directement enchaîné avec du travail sur le langage naturelle.
la méthodologie agile m'as permis de me donner des objectifs à réaliser dans des délai autre que la deadline du projet.
J'ai gagné en compétence dans l'art de la fouille de données et dans l'analyse de données en ayant travaillé avec des donnée clients car qui a donné un vrai terrain d'apprentissage du métier de *Data Analyste* dans la vie professionnel.