Catégorisation de mails

Master 2 Data Science 2022

Maxime Le Paumier

24 mai 2022

Sommaire

1 Introduction

1.1 Contexte et motivations

Important acteur de la protection sociale et de la santé, le groupe VYV est issu de l'union de nombreux acteurs mutualistes. Harmonie Mutuelle y tient une place importante tant par ses engagements sociétaux que par son action pour la santé de demain. Au sein d'Harmonie Mutuelle et de sa Direction Assurance, l'équipe Projet et Support Décisionnel a parmi ses nombreuses missions la charge d'assurer le support du décisionnel DECIBEL.

Ce décisionnel est utilisé par de nombreux métiers (souscripteur, actuariat etc.) et est un outil important pour assurer la qualité du service fourni par la mutuelle. Il est par exemple utilisé par les commerciaux pour présenter des comptes de résultats aux clients et il est donc indispensable que ses restitutions soient fiables.

Cette mission de support se traduit le plus souvent par le traitement des demandes émanant des utilisateurs contactant l'équipe (le plus souvent par mail). Ces demandes peuvent conduire au traitement d'une anomalie dans les données, à une assistance d'utilisation sur un outil ou encore à la redirection pour analyse approfondie à une équipe technique. Aujourd'hui, ces demandes sont historisées manuellement afin d'en dégager des indicateurs de performances (qualité de service, nombre d'anomalies, ...) publiés trimestriellement au sein du comité de direction.

Une telle historisation manuelle des demandes reçues est répétitive et conduit souvent à des oublis ou à des erreurs. Il peut donc être intéressant pour l'équipe d'automatiser ce suivi, ce qui est l'objectif premier de ce projet. Le principal problème est donc de reconnaître sans intervention d'un opérateur les mails demandant une assistance. Ainsi, nous chercherons à développer un modèle de classification capable d'identifier de façon fiable les demandes de support concernant le décisionnel DECIBEL, en nous inspirant de différentes méthodes de traitement du langage. Pour plus de détail, le code source est disponible dans les différents notebooks python disponibles sur github : https://github.com/Maxime-LP/NLP-Mail-Classifier.

1.2 Traitement du langage

De la même manière qu'un être humain réclame qu'on lui parle dans un langage qu'il peut comprendre, un ordinateur ne peut comprendre que des langages de programmation précis, structurés et sans ambi-

guïté. Le langage humain est quant à lui imprécis, équivoque et parfois confus. Ainsi, pour permettre à un programme de comprendre le sens de notre langage, il faut employer des algorithmes capables d'analyser le sens et la structure des phrases ou encore de reconnaître certaines références.

Nous verrons d'abord qu'il existe plusieurs manières de représenter des données textuelles afin de les utiliser dans nos modèles de classification. Nous explorerons ensuite des méthodes d'apprentissage statistique adaptées à notre problème et à nos données. Puis nous aborderons un peu plus longuement les méthodes de Deep Learning, notamment les architectures de réseaux de neurones récurrents comme les Long short-term memory (LSTM) et les Bidirectional Long short-term memory (Bi-LSTM) ainsi que des méthodes hybrides alliant ces derniers à des réseaux de neurones à convolution. Nous évoquerons aussi les Gated Recurrent Unit (GRU), et nous verrons que ces modèles se révèlent plus efficaces que les LSTM lorsqu'on les applique sur nos données. Nous parlerons enfin des algorithmes de Deep Learning qui constituent l'état de l'art du NLP aujourd'hui, à savoir les transformers, avec lesquels Google écrasa toute concurrence en 2017. Pour conclure, nous verrons comment ces modèles seront mis en production au sein de l'équipe.

2 Pré-traitement des données

2.1 Données disponibles et pre-processing

Nous disposons d'un ensemble de données très déséquilibré : 4759 mails concernant le décisionnel DECIBEL (label 1) contre 590 ne le concernant pas. La première étape est une étape de nettoyage des textes afin de retirer les mots parasites et de réduire le vocabulaire total. Dans la pratique, il s'agit essentiellement de retirer la ponctuation et les symboles, de normaliser le texte (le mettre en minuscules, ...), de retirer les mots vides (ou "stop words"), c'est-à-dire les mots n'apportant pas de sens aux phrases comme les articles, et enfin de lemmatiser le texte, c'est-à-dire de remplacer chaque mot par sa racine lexicale, aussi appelé lemme (par exemple, mange devient manger et belles devient beau).

Les tâches de pré-traitement basiques comme la suppression de la ponctuation ou des symboles sont simplement réalisées à l'aide d'expressions régulières tandis que les tâches de NLP plus complexes sont effectuées via Stanza[2], une librairie python de NLP développée par l'université de Stanford et prenant en charge plus de 70 langues, ou Spacy[1] pour la suppression des mots vides.

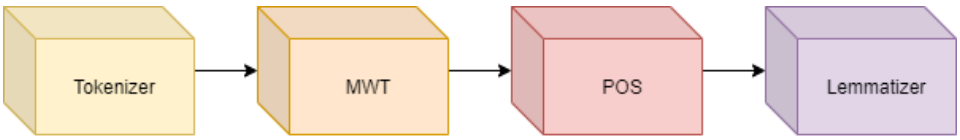

FIGURE 1 – Pipeline Stanza NLP

Plus précisément, la pipeline regroupant les différents traitements de NLP se compose de plusieurs modules : Tokenizer, Multi-Word Token (MWT) expansion, Part-of-Speech (POS) recognition et enfin lemmatisation.

Le premier module a pour fonction de séparer les documents en phrases et en tokens individuels (ici en mots) afin de les préparer pour la suite de la pipeline. Ce processus est indispensable pour le reste des traitements et, à l'issue de ce dernier, chaque document est une liste de phrases et chaque phrase est une liste de tokens.

Le module d'expansion des MWT (c'est-à-dire des tokens constitués de plusieurs mots) a ensuite pour but de repérer les bi-grammes, dont nous parlerons plus en détails dans la section suivante. Les bi-grammes sont des couples de mots apportant plus de sens groupés que séparés (on peut par exemple penser au bi-gramme New York), et

identifier de telles relations permet de faciliter le travail du module de reconnaissance des POS, qui constitue l'ultime étape à réaliser avant la lemmatisation des tokens.

Cette avant-dernière étape de la pipeline consiste à déterminer la fonction des tokens au sein des documents (on parle de part of speech recognition). Ce module se base sur un système international (universal POS (UPOS) tags, cf universaldependencies.org/u/pos/ pour la liste détaillée) et labellise chaque token avec un tag UPOS correspondant à sa fonction dans la phrase, par exemple un adjectif se verra assigner le tag 'ADJ' et un nombre le tag 'NUM'. Cette reconnaissance est nécessaire pour garantir la précision de la lemmatisation des mots.

Une fois ces étapes préliminaires effectuées, on procède à la lemmatisation et, à l'issue de tous ces traitements, on a pour chaque token du document son tag UPOS et son lemme (sa racine).

2.2 Feature engineering

On parle de feature engineering comme du fait de calculer de nouvelles variables significatives à partir des données. En plus du texte brut, il peut en effet être intéressant de calculer de nouvelles variables qui permettront d'améliorer la qualité de nos modèles. On peut penser par exemple à compter le nombre d'adjectifs, d'adverbes, de nombres, de points d'interrogations, etc... En se basant sur les tag UPOS décrits ci-dessus, 18 features sont ainsi calculées : ADJ_count, ADP_count, ADV_count, AUX_count, CCONJ_count, DET_count, INTJ_count, NOUN_count, NUM_count, PRON_count, PROPN_count, PUNCT _count, SCONJ_count, SYM_count, VERB _count, X_count, unique _words_count et _questionmark_count_.

2.3 Déséquilibre des classes

L'important déséquilibre de nos données risque de dégrader les performances des modèles. Ceux-ci auront en effet tendance à sous-estimer l'erreur qu'ils commettent sur la classe sous-représentée. Il existe néanmoins plusieurs solutions à ce problème et nous allons en présenter certaines.

Une méthode intéressante consiste à créer de nouveaux individus appartenant à la classe sous-représentée dans notre ensemble de données (on parle alors de data augmentation ou de sur-échantillonnage). Dans le cas de données textuelles, on peut par exemple remplacer certains mots par un de leurs synonymes. Pour cette approche en particulier, les transformers que nous évoquerons dans la section 6 sont très

performants. A l'inverse, on peut aussi retirer aléatoirement des individus de la classe sur-représentée, on parle alors de sous-échantillonnage ou de subsampling.

Il est également possible d'adapter le score utilisé pour évaluer les modèles. L'idée est de donner plus d'importance aux erreurs commises sur la classe sous-représentée. Par exemple, dans notre cas, il est nécessaire de donner plus de poids aux erreurs commises sur la classe 0 (c'est-à-dire la classe des mails ne concernant pas le décisionnel DECIBEL). Ainsi, si l'on note y la vraie classe d'un mail et $\hat{y}$ la prédiction du modèle, notre fonction de perte l peut s'écrire :

$$l(y, \hat{y}) = \alpha \mathbb{1}_{\{y=0, \hat{y}=1\}} + (1 - \alpha) \mathbb{1}_{\{y=1, \hat{y}=0\}}$$

avec $\alpha \in [0, 1]$ le terme de pénalisation.

Une autre solution peut être d'utiliser le F1-score, particulièrement utile dans les problèmes utilisant des données déséquilibrées. Il s'agit de la moyenne harmonique de la précision et du rappel :

$$F_1 = \frac{2}{\frac{1}{\text{précision}} + \frac{1}{\text{rappel}}}$$

où le rappel (aussi appelé sensibilité) est la proportion des mails pertinents identifiés parmi l'ensemble des items pertinents et la précision est la proportion des mails pertinents parmi l'ensemble des mails identifiés.

De façon équivalente, si l'on note TP, FN et FP les taux de vrais positifs, faux négatifs et faux positif, on peut écrire :

$$F_1 = \frac{\text{TP}}{\text{TP} + \frac{1}{2}(\text{FN} + \text{FP})}$$

Dans le cadre de ce projet, nous n'utiliserons que le F1-score, à la fois comme métrique d'entraînement et comme métrique d'évaluation. Nous verrons dans la section 4 que le sous-échantillonnage conduit aussi à de bons résultats mais que nous ne le retiendrons pas car cette méthode a tendance a diminuer le rappel de la classe 1.

3 Représentation des données textuelles

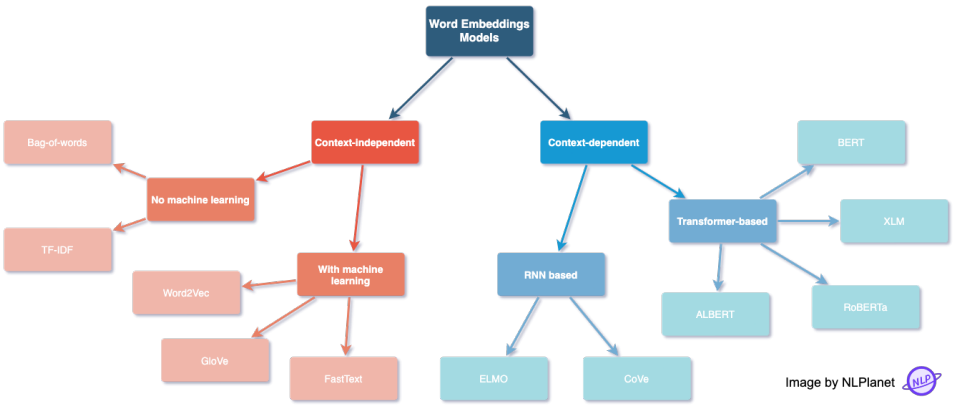

FIGURE 2 – Modèles de word embeddings

La représentation de données textuelles sous la forme de données numériques utilisables par des modèles de machine learning est un sujet vaste. Nous verrons en détails deux paradigmes : la méthode de Bag of Words (sac de mots) et la méthode plus élaborée de Word2Vec.

3.1 Bag of words

La méthode de bag of words est très intuitive : on compte pour chaque mot son nombre d'occurrences dans chaque document.

Mail	referentiel	decibel	direction	assurance
mail1	0	2	1	3
mail2	1	0	2	1
mail3	0	0	0	1
...	...	...	...	...

TABLE 1 – Bag of Words

Comme on l'a évoqué précédemment, certains couples de mots apportent plus de sens regroupés que séparés. Il est donc important de regrouper les bi-grammes comme illustré dans la table 2.

On peut cependant déjà saisir une limite à cette approche qui apporte beaucoup de crédit aux fréquences d'apparition des mots. En effet, les mots apparaissant souvent n'en sont pas pour autant plus importants. La variante TF-IDF (Term Frequency-Inverse Document Frequency) apporte une réponse à ce problème et est une manière de mesurer l'importance d'un mot au sein d'un document par rapport à son importance dans la collection entière des documents.

Mail	referentiel	decibel	direction	assurance	direction_assurance
mail1	0	2	0	2	1
mail2	1	0	1	0	1
mail3	0	0	0	1	0
...	...	...	...	...	...

TABLE 2 – Bag of Words + Bigrams

Le calcul du TF-IDF se fait en multipliant la fréquence apparition du mot dans le document (TF) par l'inverse de sa fréquence d'apparition dans la collection de documents (IDF), elle vise ainsi à donner un poids plus important aux termes les moins fréquents.

L'une des formulations habituelles comprend un logarithme pour atténuer la croissance linéaire des quantités en jeu et s'écrit, pour un corpus de documents D, un document $d \in D$ et un terme t :

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) * \log \frac{|D|}{|\{d \in D | t \in d\}|}$$

où $\text{TF}(t, d)$ est la fréquence d'apparition du terme t dans le document d.

Ces deux approches conduisent nécessairement à des données éparses et supposent l'utilisation de méthodes fortement régularisées, évitant ainsi le sur-apprentissage, et capables d'appréhender les grandes dimensions comme les SVM, la régression logistique pénalisée ou les algorithmes de k plus proches voisins. Calculatoirement coûteuses, tant au niveau de la transformation des données que de l'apprentissage, elles restent peu adaptées à une utilisation avec des réseaux de neurones.

3.2 Word embedding

Le word embedding (parfois traduit par "plongement lexical") est une méthode de représentation des mots par vectorisation qui apporte une solution au problème des données éparses engendrées par la méthode bag of words. Cette méthode a pour but de représenter les mots d'un corpus par des vecteurs de nombres réels et de dimension fixée afin de faciliter leur analyse. Le word embedding est capable tout en réduisant la dimension de capturer le contexte des phrases ainsi que les similarités sémantique et syntaxique (genre, synonymes, ...). Cette nouvelle représentation a pour propriété remarquable que les mots apparaissant dans des contextes similaires possèdent des vecteurs correspondants relativement proches dans l'espace de représentation. Par exemple, on pourrait s'attendre à ce que les mots « chien » et « chat

» soient représentés par des vecteurs relativement peu distants dans l'espace vectoriel où sont définis ces vecteurs.

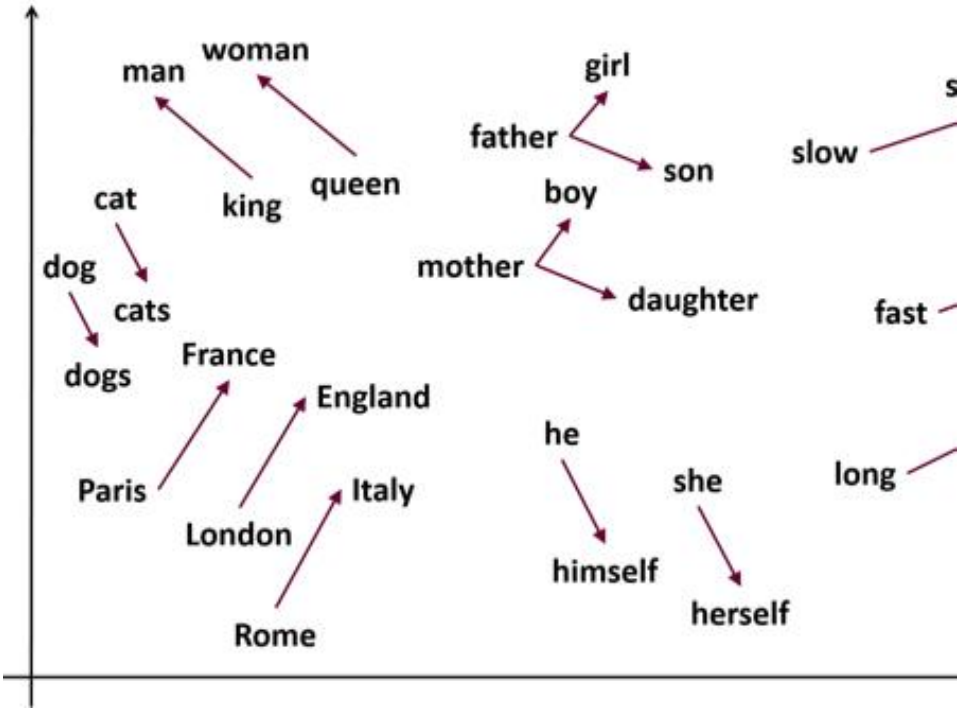

FIGURE 3 – Exemple de word embedding sur un corpus en anglais

3.2.1 Word2Vec

Word2Vec est une famille d'architectures de réseaux de neurones couramment utilisés pour le word embedding. En utilisant d'importants volumes de textes, ils sont capables d'apprendre les relations entre les mots, et de faire correspondre à chacun un unique vecteur lié aux autres par de remarquables relations permettant par exemple d'écrire vec("roi") - vec("homme") + vec("femme") = vec("reine"). Entraînés pour reconstruire le contexte des mots, ces modèles se déclinent en deux versions, Skip-grams (SG) et Continuous-bag-of-words (CBOW), l'une visant à retrouver un mot à partir de son contexte et l'autre visant à retrouver le contexte à partir d'un mot.

Word2Vec utilise une astuce assez répandue en deep learning : on entraîne un réseau de neurones avec une couche cachée à réaliser une tâche, puis, plutôt que d'utiliser le réseau de neurones en lui-même, on va utiliser les poids de la couche cachée appris pendant l'entraînement.

Détaillons le fonctionnement de l'implémentation skip-gram. La tâche pour laquelle on programme notre réseau Word2Vec skip-gram consiste à chercher à reproduire le contexte étant donné un mot d'une phrase. Le modèle est ainsi capable de donner la probabilité pour

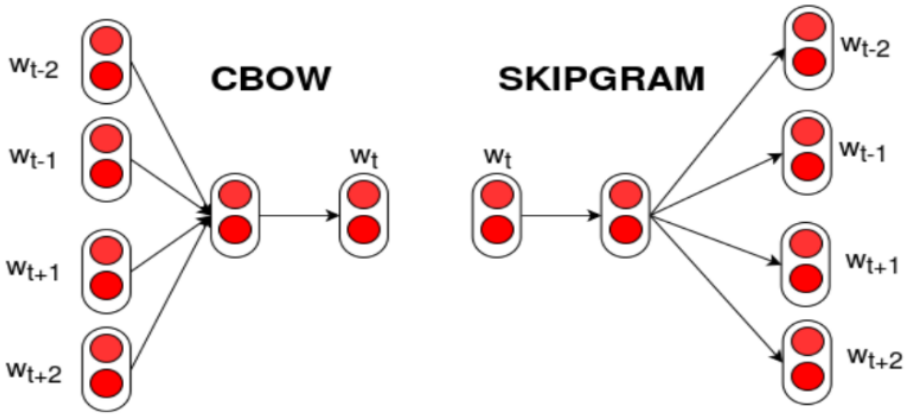

FIGURE 4 – Les modèles CBOW et Skip-gram de word2vec

chaque mot du vocabulaire d'être proche du mot d'entrée (avec une taille de fenêtre paramétrable). On entraîne le réseau en lui fournissant des paires de mots trouvées dans les documents d'entraînement comme illustré dans la figure 5. Il apprend donc, pour chaque mot passé en entrée, à retrouver les mots les plus courants dans le contexte associé.

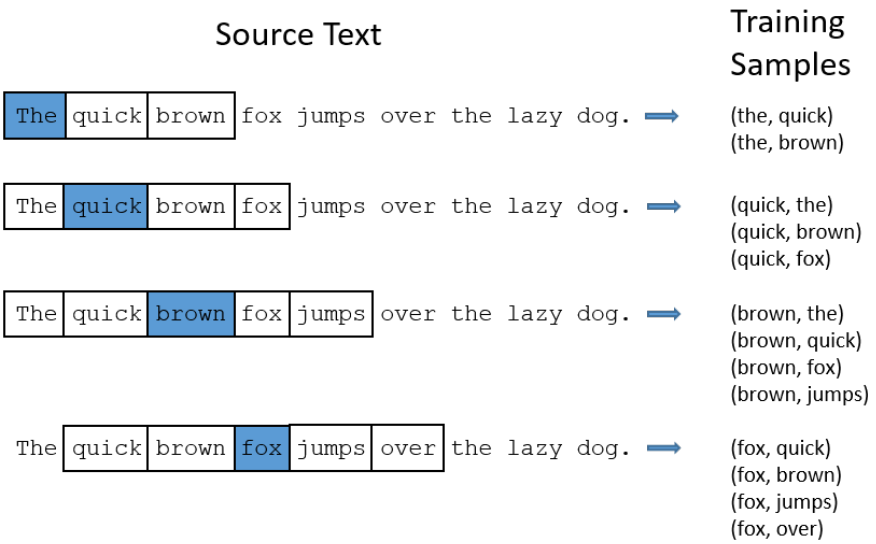

FIGURE 5 – Entraînement Word2Vec (tiré de [7])

Les mots en entrée sont représentés par des vecteurs one-hot avec autant de composantes que de mots dans le vocabulaire du modèle et la sortie du réseau est un vecteur possédant autant de composantes regroupant les probabilités de retrouver chaque mot à proximité du mot d'entrée. La figure 6 illustre bien cette architecture dans un contexte de word embedding à 300 dimensions sur un vocabulaire de 10 000 mots. Dans un tel contexte, on apprend donc 10 000 vecteurs à 300

dimensions, c'est-à-dire une matrice de poids avec 10 000 lignes et 300 colonnes.

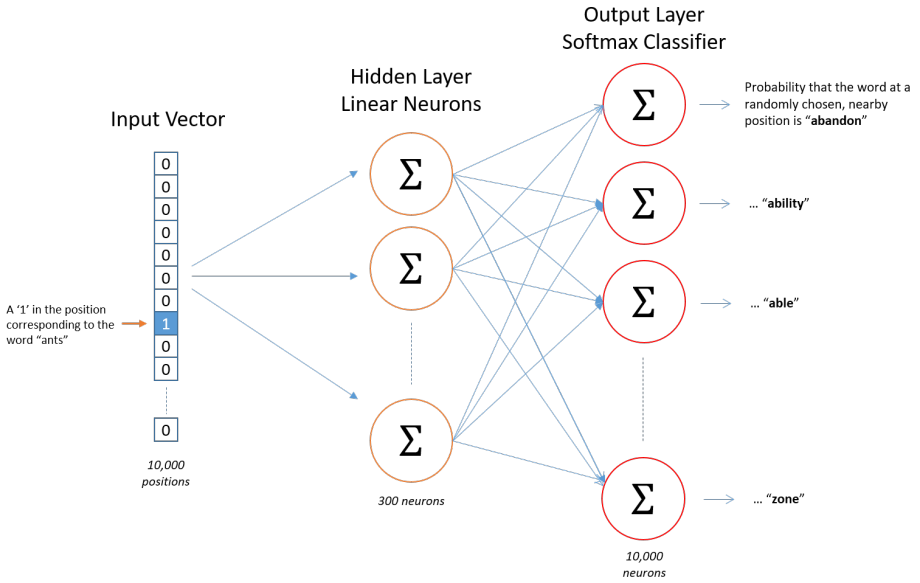

FIGURE 6 – Architecture Word2Vec (tiré de [7])

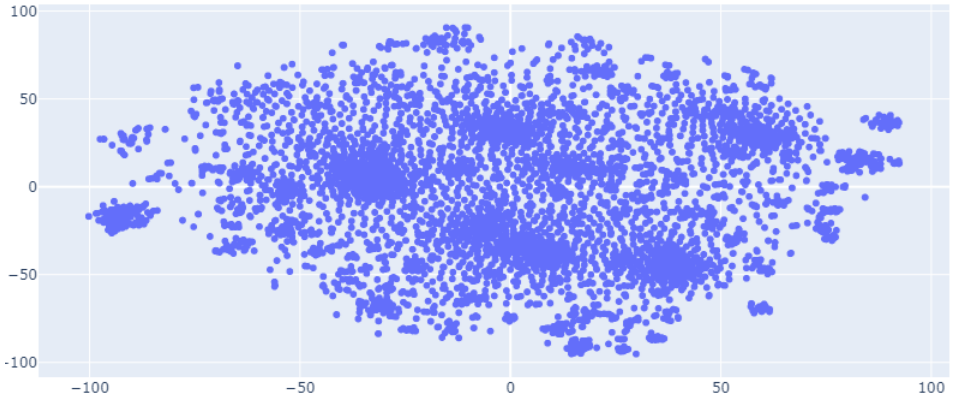

FIGURE 7 – Word2Vec + TSNE

Appliquée à notre corpus, cette méthode a pour avantage de réduire fortement la dimension : de 12 624 variables avec le bag of words, on passe à 519 (500 dimensions d'embedding + 18 meta-features + 1 label).

4 Évaluation des premiers modèles

Pour évaluer les performances des différents modèles, nous utilisons le rappel et la précision de chaque classe. Le F1-score est donné comme la moyenne des F1-score des deux classes. Notre problème initial consistant à identifier les mails demandant une assistance, il est important de garder à l'esprit que, dans le meilleur des cas, aucun ne doit être ignoré. Nous cherchons donc à maximiser le rappel de la classe regroupant les mails DECIBEL, c'est-à-dire que l'on cherche à identifier tous les mails pertinents, quitte à en proposer plus de non pertinents.

4.1 Résultats Bag of Words

Évaluons dans un premier temps les performances des modèles entraînés sur les données Bag of Words avec bi-grammes et features calculées. Pour comparaison, nous donnons aussi les résultats des modèles entraînées sur les données sous-échantillonnées.

Modèle	Métriques				
	Précision 0	Précision 1	Rappel 0	Rappel 1	F1-score
SVC	0,89	0,97	0,77	0,99	0.9
RL	0,89	0,97	0,77	0,99	0.9
KNN	0,85	0,93	0,42	0,99	0.76
KNN_cosine	0,87	0,97	0,73	0,99	0.89

TABLE 3 – Scores de validation sans sous-échantillonnage

Modèle	Métriques				
	Précision 0	Précision 1	Rappel 0	Rappel 1	F1-score
SVC	0,96	0,91	0,91	0,95	0.93
RL	0,95	0,9	0,91	0,95	0.92
KNN	0,71	0,67	0,7	0,68	0.69
KNN_cosine	0,75	0,81	0,85	0,69	0.77

TABLE 4 – Scores de validation avec sous-échantillonnage

Dans le cas de l'approche bag of words, on remarque que le sous-échantillonnage a tendance à augmenter les performances générales des modèles, au contraire du rappel de la classe 1 qui a tendance à diminuer. Ainsi, bien que légèrement plus performants sur la globalité

des données, ces modèles ignoreront plus de demandes d'assistance que les modèles entraînés sans sous-échantillonnage, ce qui n'est pas souhaitable.

4.2 Résultats word embedding

Évaluons maintenant les performances des modèles entraînés sur les données word embedding. Nous donnons ici aussi les résultats des modèles entraînées sur les données sous-échantillonnées.

Modèle	Métriques				
	Précision 0	Précision 1	Rappel 0	Rappel 1	F1-score
SVC	0,61	0,97	0,76	0,94	0.81
RL	0,83	0,98	0,87	0,98	0.91
KNN	0,91	0,96	0,67	0,99	0.87
KNN_cosine	0,9	0,97	0,71	0,99	0.89

TABLE 5 – Scores de validation sans sous-échantillonnage

Modèle	Métriques				
	Précision 0	Précision 1	Rappel 0	Rappel 1	F1-score
SVC	0,82	0,7	0,63	0,86	0.74
RL	0,86	0,87	0,87	0,85	0.86
KNN	0,84	0,81	0,8	0,85	0.83
KNN_cosine	0,85	0,87	0,87	0,84	0.86

TABLE 6 – Scores de validation avec sous-échantillonnage

On voit que le sous-échantillonnage a tendance à dégrader à la fois les performances générales et les performances relatives à la classe 1 de nos modèles.

On voit de plus que la différence entre Bag of Words et Word Embedding est faible et que la première approche semble devancer légèrement la seconde sur certains aspects. Mais la principale utilité du Word Embedding réside dans la réduction du nombre de variables. Cette approche ouvre ainsi la voie à des algorithmes plus complexes et bien plus performants, comme les réseaux de neurones récurrents.

5 Réseaux de neurones récurrents

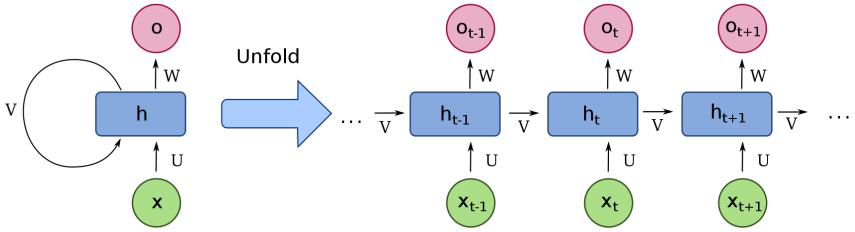

FIGURE 8 – Réseau de neurones récurrents à une unité

Source : Wikipedia

Les réseaux de neurones récurrents (RNN) sont une classe de réseaux de neurones constitués d'unités inter-connectés pouvant s'échanger de l'information et présentant des cycles dans leur structure. En d'autres termes, un RNN est constitué de plusieurs unités s'exécutant un certain nombre de fois et se transmettant, d'une étape à l'autre, non seulement le résultat calculé comme dans un réseau classique, mais aussi le contexte ou l'état du réseau de neurones. On peut penser à une unité comme à une couche cachée d'un réseau classique, un RNN est alors similaire à une succession de réseaux de neurones renvoyant deux sorties : le résultat et l'état. En théorie, les RNN peuvent tenir compte des relations de long terme dans les séquences. Ces algorithmes se révèlent efficaces dans l'analyse de séries temporelles ou dans la plupart des tâches de NLP (traduction automatique, reconnaissance vocale, ...). Cependant, un problème émerge rapidement lorsque les séquences sont longues : ils sont extrêmement difficiles à entraîner pour gérer les dépendances de long terme. En effet, pendant l'entraînement d'un RNN par rétro-propagation, les gradients ont tendance à tendre vers zéro ("vanishing gradient") ou à exploser. De nouvelles variantes de RNN ont été introduites en réponse à ce problème, comme les long short-term memory (LSTM) ou les Gated Recurrent Unit (GRU), dont nous allons détailler le fonctionnement.

5.1 Long short-term memory

Comme illustré dans la figure 9, une cellule LSTM est composée de 4 éléments : forget gate, input gate, output gate et un vecteur de contexte. Pour détailler ce fonctionnement, notons respectivement x_t, c_t, et h_t le vecteur d'entrée, le contexte de la cellule et l'état du réseau

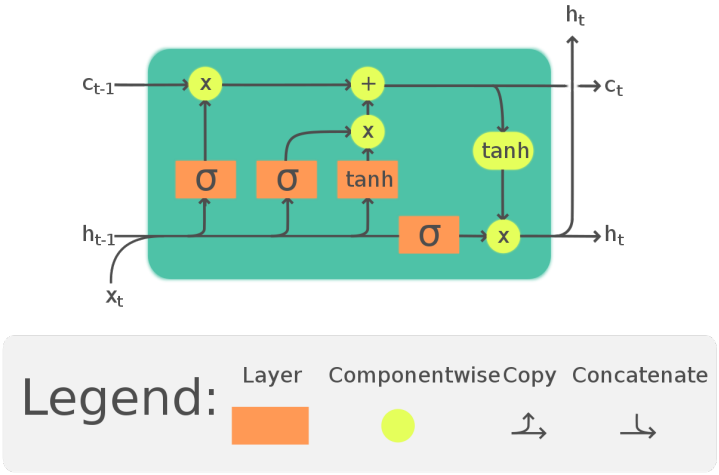

FIGURE 9 – Cellule LSTM

Source : [4]

de neurones à l'itération t. Notons aussi W et U les matrices des poids du vecteur d'entrée x_t et des connections récurrentes h_t. Le contexte c_t représente en quelque sorte la mémoire que la cellule conserve d'un cycle à l'autre.

Dans un premier temps, la forget gate décide quelle information conserver depuis l'état précédent. En se basant sur l'état précédent h_{t-1} et le vecteur d'entrée x_t, elle calcule pour chaque composante une valeur entre 0 et 1 (première couche sigmoïde) correspondant à la quantité d'information conservée pour chaque composante :

$$f = \sigma((W_f x_t + U_f h_{t-1}) + b_f).$$

L'input gate détermine ensuite quelle nouvelle information ajouter au contexte. Cette deuxième étape se fait en deux temps. D'abord, la deuxième couche sigmoïde détermine quelles valeurs du contexte mettre à jour puis la couche tanh crée un vecteur des nouvelles valeurs à ajouter au contexte :

$$\begin{aligned} i &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\ j &= \tanh(W_j x_t + U_j h_{t-1} + b_j) \end{aligned}$$

Le vecteur de contexte sert de mémoire au LSTM, ce qui permet à ce type de RNN d'être plus performants que les RNN classiques pour le traitement de longues séquences. En notant $\circ$ le produit élément à élément, la mise à jour du contexte peut s'écrire :

$$c_t = f \circ c_{t-1} + i \circ j$$

Enfin, l'ouput gate retourne le nouvel état h_t du réseau de neurones :

$$o = \sigma(W_o x_t + U_o h_{t-1} + b_o)$$
$$h_t = o \circ \tanh(c_t)$$

Ce fonctionnement a l'avantage de tenir compte des relations entre les éléments des séquences (dans notre cas des mots d'une phrase) mais seulement dans un seul sens de lecture. Or, lorsque l'on écrit un texte, les mots peuvent avoir des relations avec d'autres mots situés après dans la phrase, par exemple un adjectif précédant le nom qu'il qualifie. L'architecture Bidirectional long short-term memory (Bi-LSTM) a justement pour objectif de pallier ce défaut du LSTM et d'identifier les dépendances de long terme autant dans le passé que dans le futur. Il s'agit simplement de deux cellules LSTM indépendants évoluant dans les deux sens de lecture comme illustré dans la figure 10 et dont les sorties sont combinées par addition, concaténation, etc ...

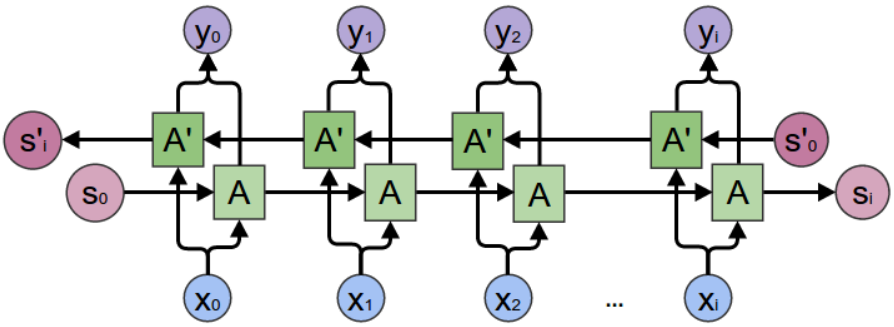

FIGURE 10 – Réseau Bi-LSTM

Source : [3]

5.2 Gated Recurrent Unit

Les réseaux Gated Recurrent Unit (GRU) sont une variante des LSTM introduite en 2014 et ayant des performances comparables à ces derniers. Leur principal avantage réside dans le fait qu'une unité GRU possède moins de paramètres à entraîner qu'une unité LSTM. Cela s'explique par une légère modification de la structure d'une cellule : un neurone n'est associé qu'à un état caché et les input et forget gates sont fusionnées en une update gate. La porte de sortie est quant à elle remplacée par une porte de réinitialisation (reset gate). Ce type de réseau nous intéresse fortement pour ce projet car nous disposons d'un volume limité de données et entraîner une réseau GRU est moins coûteux en données qu'entraîner un réseau LSTM.

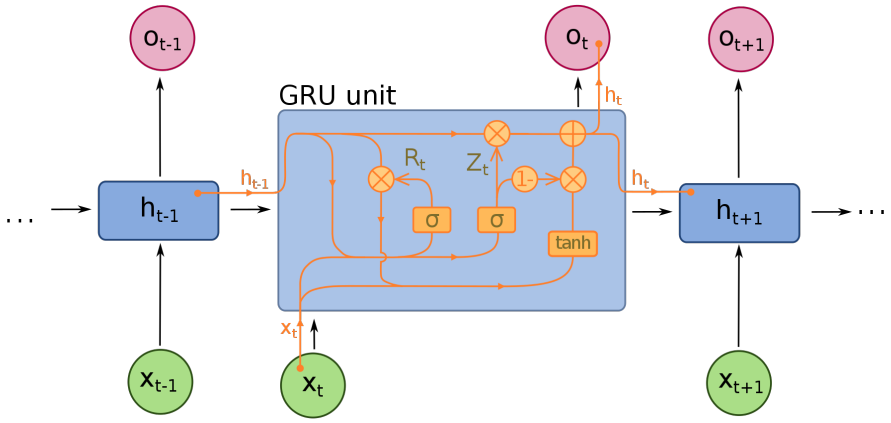

FIGURE 11 – Réseau GRU à une unité

Source : Wikipedia

Modulo l'absence du terme de contexte c_t, le calcul du nouvel état du réseau h_t est assez similaire aux calculs développés précédemment. La structure de la cellule est néanmoins légèrement plus complexe, comme l'illustre la figure 11. Les calculs correspondant peuvent s'écrire :

$$Z_t = \sigma(W_Z x_t + U_Z h_{t-1} + b_Z) \quad \text{(update gate)}$$
$$R_t = \sigma(W_R x_t + U_R h_{t-1} + b_R) \quad \text{(reset gate)}$$
$$h_t = Z_t \circ h_{t-1} + (1 - Z_t) \circ \tanh(W_h x_t + U_h(R_t \circ h_{t-1}) + b_h)$$

5.3 Évaluation des modèles de RNN

Les modèles mis en oeuvre sont des modèles de type LSTM et Bi-LSTM dont certains sont précédés d'une couche de convolution (modèles CNN_LSTM et CNN_Bi-LSTM) afin de réduire la complexité des vecteurs d'embedding passés en entrée. Nous allons aussi regarder les performances d'un modèle de Bi-GRU issu du framework Mélusine [6] et dont l'architecture est détaillée dans la figure 12.

On voit dans la table 8 que le modèle Bi-GRU issu de Mélusine écrase de façon flagrante les modèles utilisés jusqu'à présent. Ce résultat est cohérent avec notre observation sur les GRU étant plus performants que les LSTM sur les petits ensembles de données.

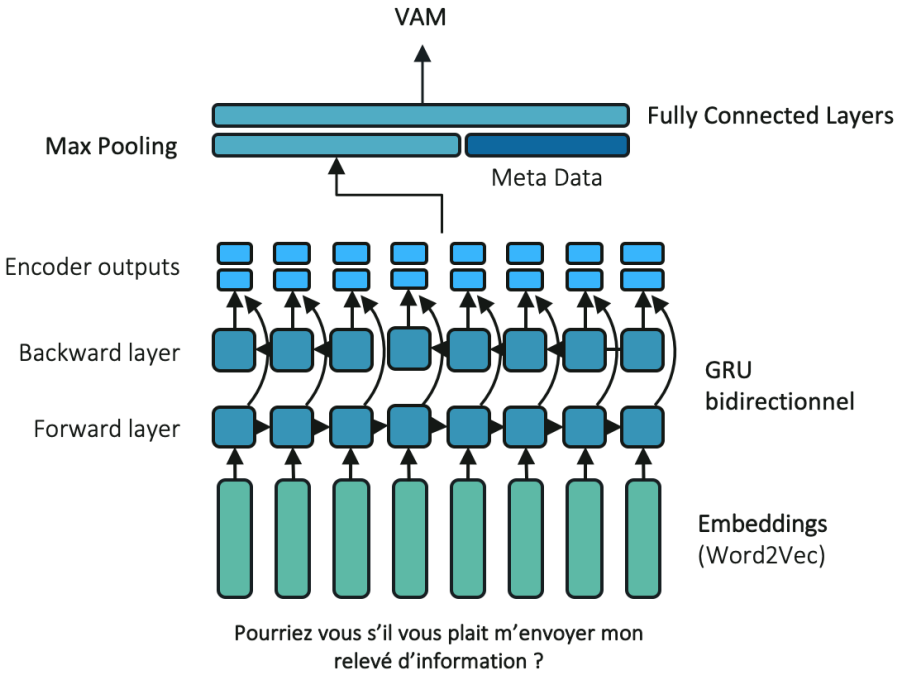

FIGURE 12 – Architecture RNN issue de Mélusine

Modèle	Métriques				
	Précision 0	Précision 1	Rappel 0	Rappel 1	F1-score
LSTM	0.87	0.98	0.82	0.99	0.91
Bi-LSTM	0.86	0.99	0.88	0.98	0.93
CNN_LSTM	0.87	0.98	0.87	0.98	0.93
CNN_Bi-LSTM	0.83	0.98	0.84	0.98	0.91
Melusine_Bi-GRU	0.93	0.99	0.94	0.99	0.97

TABLE 7 – Scores de validation

6 Transformers

En 2017, Google présente une architecture de réseau de neurones révolutionnaire appelée transformer[5] et qui présente de nombreux avantages sur les réseaux récurrents. Cette dernière section constitue une introduction à ces algorithmes complexes, ainsi qu'une mise en oeuvre sur nos données, bien que limitée par la taille de notre corpus.

Les transformers sont rapidement devenus l'état de l'art pour les problèmes de NLP, occultant complètement les architectures de RNN que nous avons développées précédemment. A l'instar de ces dernières, les transformers traitent des données séquentielles. Mais contrairement aux RNN, les transformers n'exigent pas que les données séquentielles soient traitées dans l'ordre. Un transformer n'a par exemple pas besoin de traiter le début d'une phrase avant la fin. Grâce à cette fonctionnalité, les tâches exécutées par un transformer peuvent être parallélisées de façon beaucoup plus importante que pour les RNN et les temps d'entraînement s'en trouvent réduits. On peut ainsi entraîner les transformers sur des ensembles de données beaucoup plus volumineux et gagner en performances.

Là où les RNN n'accédaient qu'à l'état précédent du réseau, les transformers se basent sur un système encodeur-décodeur qui permet au décodeur d'accéder à tous les états précédents plutôt qu'à un seul état construit par l'encodeur comme c'est le cas pour les LSTM et les GRU. Le fonctionnement de ce mécanisme, appelé attention[5], est illustré dans la figure 14.

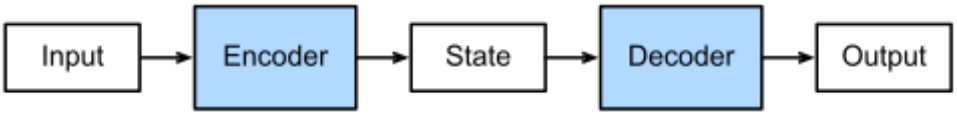

FIGURE 13 – Encodeur-décodeur

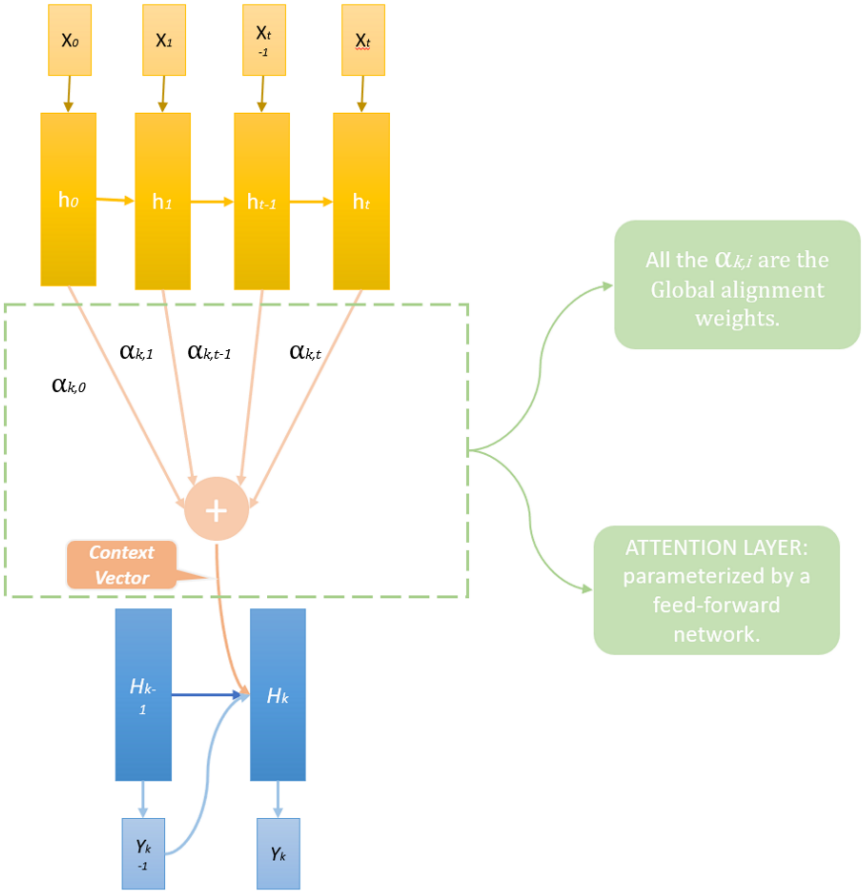

FIGURE 14 – Vecteur de contexte construit à partir de tous les états précédents

Pour comparer les performances des transformers avec celles des modèles mis en oeuvre jusqu'à présent, nous avons utilisé l'implémentation de Mélusine dont l'architecture est détaillée dans la figure 15. Relativement simple, cette architecture donne néanmoins des résultats satisfaisants (cf table 8), bien qu'en dessous de ceux du Bi-GRU. Cette différence s'explique par le faible volume de données dont nous disposons qui compromet la bonne qualité de l'entraînement de ce modèle coûteux à entraîner.

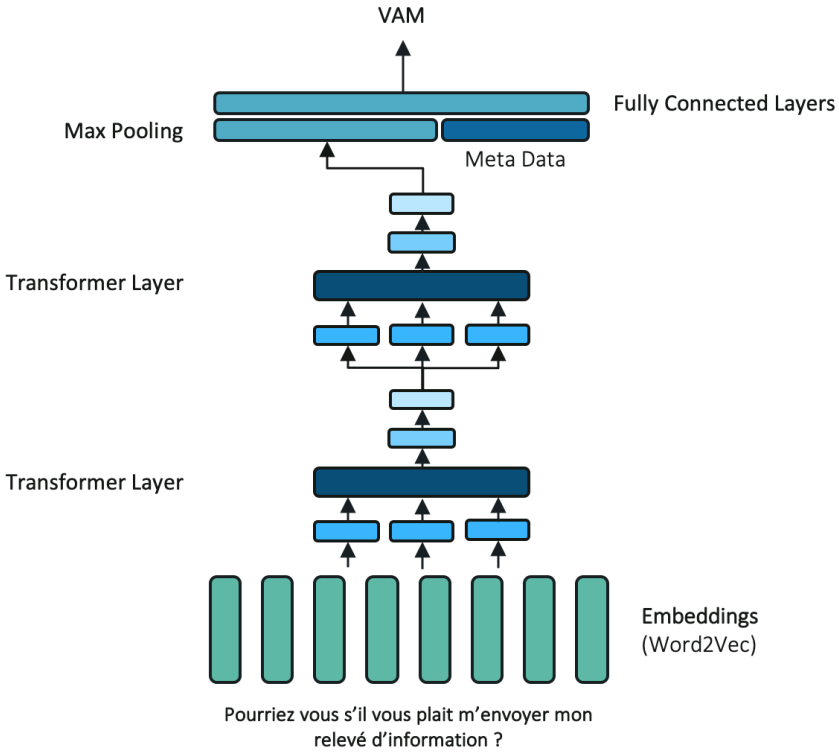

FIGURE 15 – Architecture transformer issue de Melusine

Modèle	Métriques				
	Précision 0	Précision 1	Rappel 0	Rappel 1	F1-score
Melusine_Transformer	0.91	0.98	0.82	0.99	0.93

TABLE 8 – Scores de validation

7 Conclusion

Nos nombreux essais ont montré que l'architecture Bi-GRU était de loin la plus performante sur l'ensemble de mails dont nous disposons. De plus, le rappel de 0.99 pour la classe 1 en validation garantit que ce modèle est calibré pour repérer la quasi-totalité des mails demandant une assistance. La figure 16 représente l'architecture de la pipeline qui sera mise en oeuvre. Précisons aussi que nous n'avons pas utilisé de modèles pré-entraînés pour ce projet, tant pour Word2Vec que pour les transformers. En effet, notre corpus de mails présente un vocabulaire très spécifique et les modèles génériques ne sont pas pertinents pour saisir ce contexte particulier.

Évoquons pour conclure ce rapport la mise en production d'une telle pipeline et les travaux futurs en lien avec ce projet. L'objectif est de produire une solution opérationnelle qui, lorsqu'un mail arrivera dans l'équipe de support, détectera si celui-ci est une demande d'assistance ou non. Le cas échéant, les informations associées au mail (utilisateur, nom de la demande, etc ...) seront extraites et injectées dans une base de données regroupant l'historique des demandes. Cette base pourra être complétée manuellement à posteriori après analyse par l'équipe (date de résolution, redirection de la demande, ...). Cette extraction des informations des mails pourra se faire au moyen de l'outil Power Automate développée par Microsoft.

La finalité de cette historisation est d'avoir à disposition des données afin de calculer des indicateurs reflétant la qualité du service proposé aux utilisateurs et de développer des rapports à partir de ces indicateurs dans PowerBI. Le modèle de prédiction développé dans le cadre de ce projet et servant de base à la solution cible a montré de bonnes performances et permettra une mise en oeuvre efficace de l'outil cible.

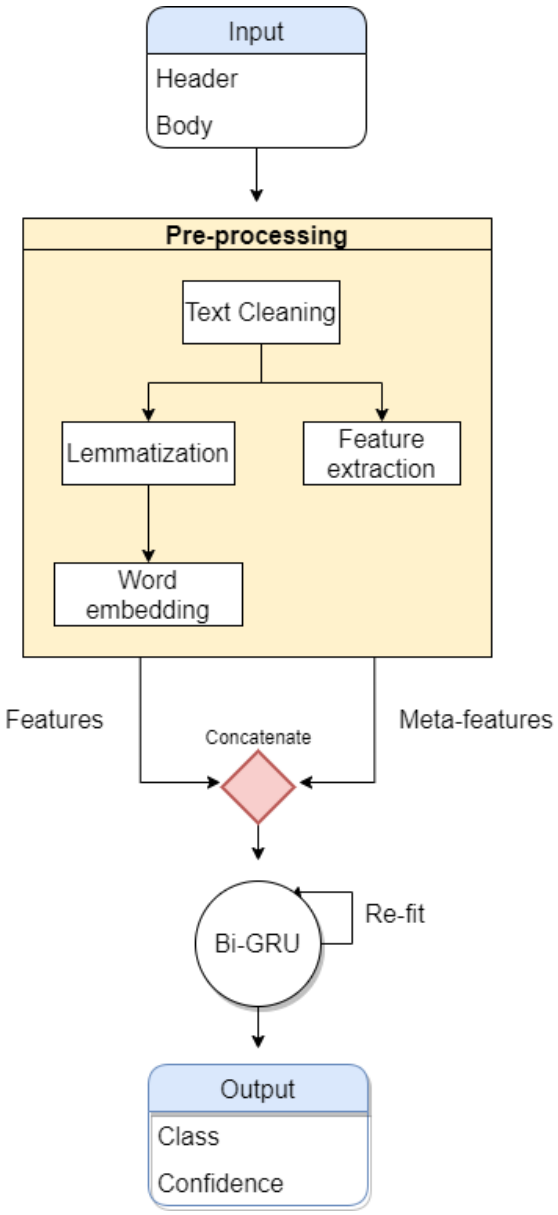

FIGURE 16 – Modèle final

```
INPUT_MAIL_DF = pd.DataFrame({
    'header':[
        'ami',
        'ano cr',
        'ano synergie'
        ],
    'body':[
        "bonjour n'oubliez pas de renseigner votre mot de passe",
        "bonjour j'ai un souci au moment de générer le cr",
        "bonjour j'ai un souci au moment de générer la table"
        ]
})
y_pred = model.predict(INPUT_MAIL_DF)
y_pred
```

	header	body	classe	confiance
0	ami	bonjour n'oubliez pas de renseigner votre mot ...	0	1.0
1	ano cr	bonjour j'ai un souci au moment de générer le cr	1	1.0
2	ano synergie	bonjour j'ai un souci au moment de générer la ...	0	1.0

FIGURE 17 – Exemple d'utilisation du prédicteur

Table des figures

Références

[1] Spacy documentation : spacy.io. 5

[2] Stanza nlp documentation : stanfordnlp.github.io/stanza. 5

[3] Raghav Aggarwal. Bi-lstm. 2019. 17

[4] Wikipedia contributors. Long short-term memory. 2022. 16

[5] A. Vaswani N. Shazeer N. Parmar J. Uszkoreit L. Jones A.Gomez L. Kaiser and I. Polosukhin. Attention is all you need. 2017. 20

[6] MAIFl. Melusine documentation. 18

[7] Chris McCormick. Word2vec tutorial - the skip-gram model, 2016. 11, 12, 26