Rapport : Projet SDIA SDSC « ChatBot industriel» WEISS Nicolas & SENSENBRENNER Amaury

L’objectif de ce projet est de développer un code de prise en compte automatique de demande d’impression par « ChatBot » et prévoir l’ordonnancement des tâches d’impression.

Approche choisie grâce aux références proposées par la sujet :

Introduction à l'incorporation de mots et à Word2Vec :

L'incorporation de mots est l'une des représentations les plus populaires du vocabulaire des documents. Elle est capable de capturer le contexte d'un mot dans un document, la similarité sémantique et syntaxique, la relation avec d'autres mots et bien plus encore.

Word2Vec est l'une des techniques les plus populaires pour apprendre les incorporations de mots à l'aide d'un réseau neuronal peu profond.

L'objectif de l'incorporation de mots est de faire en sorte que les mots ayant un contexte similaire occupent des positions spatiales proches. On va utiliser le cosinus de l'angle qui, entre de tels vecteurs, devrait être proche de 1, c'est-à-dire un angle proche de 0.

C'est ici qu'intervient l'idée de générer des représentations distribuées. Intuitivement, nous introduisons une certaine dépendance d'un mot par rapport aux autres mots. Les mots dans le contexte de ce mot obtiendront une plus grande part de cette dépendance. Dans les représentations de codage à chaud, tous les mots sont indépendants les uns des autres.

Word2Vec est une méthode permettant de construire un tel encastrement. Elle peut être obtenue à l'aide de deux méthodes (impliquant toutes deux des réseaux neuronaux) : Skip Gram et Common Bag Of Words (CBOW).

Similitudes de textes : Estimer le degré de similarité entre deux textes :

La similarité textuelle consiste à déterminer dans quelle mesure deux morceaux de texte sont "proches", à la fois en termes de surface [similarité lexicale] et de sens [similarité sémantique].

L'idée principale est de représenter les documents comme des vecteurs de caractéristiques et de comparer les documents en mesurant la distance entre ces caractéristiques. Il existe plusieurs façons de calculer les caractéristiques qui capturent la sémantique des documents et plusieurs algorithmes pour capturer la structure de dépendance des documents afin de se concentrer sur le sens des documents.

Voici quelques méthodes afin d'estimer le degré de similarité entre deux vecteurs.

0. Jaccard Similarity :

La similarité de Jaccard ou intersection sur union est définie comme la taille de l'intersection divisée par la taille de l'union de deux ensembles.

Dans la plupart des cas, la similarité de Jaccard n'est pas capable de capturer la similarité sémantique ou la sémantique lexicale de ces deux phrases.

De plus, cette approche a un défaut inhérent. En effet, plus la taille du document augmente, plus le nombre de mots communs a tendance à augmenter, même si les documents traitent de sujets différents.

1. K-means et Dendrogramme de Clustering Hiérarchique :

Avec les algorithmes liés à K-mean, nous devons d'abord convertir les phrases en vecteurs. Plusieurs façons de le faire sont d'utiliser :

- Le sac de mots avec soit la méthode TF (fréquence des termes) appelée Count Vectorizer.
- TF-IDF (fréquence des termes - fréquence inverse des documents).

BoW ou TF-IDF créent un nombre par mot alors que les word embeddings créent généralement un vecteur par mot.
Le BoW ou TF-IDF est bon pour classer les documents dans leur ensemble, mais l'intégration des mots est bonne pour identifier le contenu contextuel.

On applique alors un alogrithme de type K-means sur les vecteurs afin de voir les similarités.

On peut noter que les k-means (et les minibatch k-means) sont très sensibles à la mise à l'échelle des caractéristiques et que dans ce cas la pondération IDF aide à améliorer la qualité du clustering.

2. Cosine Similarity :

La similarité en cosinus calcule la similarité en mesurant le cosinus de l'angle entre deux vecteurs.

Plus l'angle est petit, plus la similarité en cosinus est élevée.

Il existe aussi Smooth Inverse Frequency :

Prendre la moyenne des mots incorporés dans une phrase (comme nous l'avons fait ci-dessus) tend à donner trop de poids à des mots qui ne sont pas pertinents, sémantiquement parlant. Smooth Inverse Frequency tente de résoudre ce problème de deux manières : Pondération + Suppression des composants communs.

3. Latent Semantic Indexing (LSI)

Nous ne comparons pas directement la similarité en cosinus des vecteurs de sacs de mots, mais nous réduisons d'abord la dimension de nos vecteurs de documents en appliquant une analyse sémantique latente.

On suppose souvent que l'espace sémantique sous-jacent d'un corpus est d'une dimensionnalité inférieure au nombre de tokens uniques. Par conséquent, LSA applique une analyse en composantes principales sur notre espace vectoriel et ne conserve que les directions de notre espace vectoriel qui contiennent le plus de variance (c'est-à-dire les directions de l'espace qui changent le plus rapidement, et qui sont donc supposées contenir plus d'informations). Ceci est influencé par les paramètres num_topics que nous passons au LsiModelconstructor.

4. Word Mover’s Distance :

Une valeur de cosinus de 0 signifie que les deux vecteurs sont à 90 degrés l'un de l'autre (orthogonaux) et ne correspondent pas. Plus la valeur du cosinus est proche de 1, plus l'angle est petit et plus la correspondance entre les vecteurs est grande.

La distance de Word Mover résout ce problème en tenant compte des similitudes des mots dans l'espace d'intégration des mots.

WMD utilise l'encastrement des mots dans deux textes pour mesurer la distance minimale que les mots d'un texte doivent parcourir dans l'espace sémantique pour atteindre les mots de l'autre texte.

5. LDA with Jensen-Shannon distance :

L'allocation de Dirichlet latente (LDA) est un modèle génératif non supervisé qui attribue des distributions thématiques aux documents.

Le modèle génère des variables latentes (cachées) :

- une distribution sur les sujets pour chaque document (1)
- une distribution sur les mots pour chaque sujet (2)

Après l'apprentissage, chaque document aura une distribution discrète sur tous les sujets, et chaque sujet aura une distribution discrète sur tous les mots.

La distance de Jensen-Shannon nous indique quels documents sont statistiquement "plus proches" (et donc plus similaires), en comparant la divergence de leurs distributions.

Jensen-Shannon est une méthode de mesure de la similarité entre deux distributions de probabilité.

6. Variational Auto Encoder :

Le rôle de ces modèles est de prédire l'entrée, à partir de cette même entrée.

Bien que cela puisse sembler trivial au premier abord, il est important de noter que nous voulons apprendre une représentation comprimée des données, et donc trouver une structure. Cela peut être fait en limitant le nombre d'unités cachées dans le modèle. Ce type d'auto-codeurs est appelé sous-complet.

Le problème clé sera d'obtenir la projection de données en une seule dimension sans perdre d'informations.

Dans les auto-codeurs déterministes normaux, le code latent n'apprend pas la distribution de probabilité des données et, par conséquent, il ne convient pas pour générer de nouvelles données.

Le VAE résout ce problème puisqu'il définit explicitement une distribution de probabilité sur le code latent. En fait, il apprend les représentations latentes des entrées non pas comme des points uniques mais comme des régions ellipsoïdales douces dans l'espace latent, forçant les représentations latentes à remplir l'espace latent plutôt que de mémoriser les entrées comme des représentations latentes ponctuelles et isolées.

Notre objectif ici est d'utiliser le VAE pour apprendre les représentations cachées ou latentes de nos données textuelles qui sont une matrice d'intégration de mots. Nous utiliserons le VAE pour mettre en correspondance les données avec les variables cachées ou latentes. Nous visualiserons ensuite ces caractéristiques pour voir si le modèle a appris à différencier les documents de différents sujets. Nous espérons que les documents similaires sont plus proches dans l'espace euclidien, conformément à leurs sujets. Les documents similaires sont les uns à côté des autres.

7. Pre-trained sentence encoders :

L'encodeur universel de phrases encode le texte en vecteurs de haute dimension qui peuvent être utilisés pour la classification de textes, la similarité sémantique, le regroupement et d'autres tâches en langage naturel.

Les encodeurs de phrases pré-entraînés visent à jouer le même rôle que word2vec et GloVe, mais pour les incorporations de phrases : les incorporations qu'ils produisent peuvent être utilisées dans une variété d'applications, telles que la classification de textes, la détection de paraphrases, etc. En général, ils ont été entraînés sur une série de tâches supervisées et non supervisées, afin de capturer autant d'informations sémantiques universelles que possible.

8. Siamese Manhattan LSTM (MaLSTM) :

Les réseaux siamois sont des réseaux qui contiennent deux sous-réseaux identiques ou plus.

Les réseaux siamois semblent donner de bons résultats dans les tâches de similarité et ont été utilisés pour des tâches telles que la similarité sémantique des phrases, la reconnaissance de fausses signatures et bien d'autres encore.

Siamese est le nom de l'architecture générale du modèle où le modèle consiste en deux sous-réseaux identiques qui calculent une sorte de vecteurs de représentation pour deux entrées et une mesure de distance est utilisée pour calculer un score afin d'estimer la similarité ou la différence des entrées.

9. Bidirectional Encoder Representations from Transformers (BERT) with cosine distance :

Les outils de modélisation du langage tels que ELMO, GPT-2 et BERT permettent d'obtenir des vecteurs de mots dont la morphologie connaît leur place et leur environnement.

Les modèles BERT pré-entraînés peuvent être téléchargés et ils ont des scripts pour exécuter BERT et obtenir les vecteurs de mots de toutes les couches.

Notre but ici est de montrer que les vecteurs de mots de BERT se transforment eux-mêmes en fonction du contexte. Les résultats montrent que BERT est capable de comprendre le contexte dans lequel le mot est utilisé.

Lorsque la classification est l'objectif principal, il n'est pas nécessaire de construire un vecteur phrase/document BoW à partir des encastrements BERT.

10. A word about Knowledge-based Measures :

Les mesures basées sur la connaissance quantifient la parenté sémantique des mots en utilisant un réseau sémantique. De nombreuses mesures ont montré qu'elles fonctionnaient bien sur la grande base de données lexicale WordNet pour l'anglais.

Nous considérons ici le problème de l'intégration des entités et des relations des données multi-relationnelles dans des espaces vectoriels de faible dimension, son objectif est de proposer un modèle canonique facile à entraîner, contenant un nombre réduit de paramètres et pouvant s'adapter à de très grandes bases de données.

La métrique de Wu et Palmer mesure la similarité sémantique de deux concepts comme leur profondeur du sous-consommateur le moins commun dans le graphe du wordnet.

Pour la réalisation de ce projet, il nous a fallu développer deux modules : un module ChatBot et un module de gestion d'agenda que nous décrivons ci-dessous.

Module ChatBot

Tout d'abord, le sujet nous a demandé d'implémenter un module ChatBot. Il consistera à recevoir des demandes d'utilisateurs afin de les traiter, de les classer, pour savoir si oui ou non celle-ci concerne une demande d'impression d'un document. Ce module va donc comporter deux composants logiciels distincts : une partie "hors ligne" et une autre "en ligne".

Partie "hors ligne"

Cette partie "hors ligne" permet de créer un modèle de classification puis de l'entraîner afin qu'il réponde à nos attentes : classer les demandes d'utilisateurs, savoir si elles correspondent à des demandes d'impression ou autre chose.

Pour cette partie, nous avons décidé de mettre en place un modèle Bag of Words, premier modèle que nous avions décrit plus tôt.

Pour cela, nous avons mis en place un convertisseur, "TfidfVectorizer" qui nous permettra de convertir un ensemble de textes, de documents, en matrices de features TF-IDF. Il transforme alors les phrases en vecteurs. Pour le rendre encore plus utile, nous l'avons paramétré en lui indiquant les mots vides (stopwords, ce sont des mots communs qui théoriquement n'apportent aucune information au sens de la phrase) existants en français, en les téléchargeant depuis l'import nltk.

Ses autres paramètres ont été sélectionnés afin d'obtenir les meilleurs résultats en terme de classification sans avoir de sur-apprentissage.

Concernant le modèle lui-même, nous avons opté pour un classifieur Random Forest avec 1000 estimateurs. Ce modèle va alors être entraîné en utilisant des données bien précises, décrites ci-dessous.

Les données utilisées

Pour l'entraînement de notre modèle, nous avons utilisé plusieurs jeux de données : un que nous importons en ligne et un autre en local.

Concernant le jeu de données en ligne, il s'agit du jeu de données appelé "Piaf" : un jeu de données de type Questions-Réponses. Ce qui nous a surtout interessé ici était la partie Questions qui est la seule que nous avons gardé. L'ensemble de ces données ont été labellisées 0 qui signifie "autre" dans la classification des requêtes.

Concernant le jeu de données en local, celui-ci est constitué de 2000 lignes et a été à l'origine généré aléatoirement grâce à l'expression régulière suivante :

/(Bonjour, |Bonsoir,)?(je (veux|souhaite))?(impression|imprime|imprimer|print) [a-zA-Z0-9]{1,10} ((qui (contient|a))|a)? [0-9]{1,4} (pages|p) (en [0-9]{1,2} (exemplaires|copies|fois))?/

Ces données là ont donc été labellisées 1 signifiant "demande d'impression".

Une fois ces données concaténées, il nous a fallu tout de même les traiter avant de les passer dans le convertisseur. Il nous a donc fallu enlever l'ensemble des caractères spéciaux, les mots à un seul caractère, les espaces en trop (notamment les espaces consécutifs) puis les mettre en minuscule. Une fois ces étapes réalisées, nous les avons réduits à leur forme canonique en utilisant la lemmatisation.

C'est seulement après ce traitement que nous les avons convertis en vecteurs en utilisant "TfidfVectorizer". Le résultat a ensuite été séparé en données d'entraînement et de test avant d'entraîner notre classifieur.

Partie "en ligne"

Une fois la partie "hors ligne" réalisée, nous pouvons utiliser notre classifieur afin de classer toute requête émise sur notre interface par un utilisateur.

L'utilisateur peut alors émettre une requête dans une zone de saisie sur notre interface et celle-ci sera ensuite analysée par notre classifieur avant d'être labellisé 1 ou 0 en fonction de sa nature (1 étant une demande d'impression et 0 étant toute autre requête).

La requête étant alors classée, nous regardons si celle-ci est labellisée 1, si tel est le cas nous allons extraire les informations pertinentes : le nom du document et son nombre de pages. Notre implémentation ne permet malheureusement d'accepter uniquement des noms de documents commençant par "doc" et elle ne prend pas en compte les demandes d'impression n'indiquant pas le nombre de pages à imprimer.

Une fois ces données extraites, nous les envoyons dans le deuxième module du projet : le module de gestion d'agenda.

Module de gestion d'agenda

Le sujet du projet nous demandait aussi d'implémenter un module qui garde la liste des heures d'impression disponibles, et des créneaux occupés. L'objectif est d'ordonnancer les demandes en remplissant la liste des tâches d'impression de la machine et les datés.

Dans un premier temps, nous avons décidé de créer notre interface à l'aide du language de programmation Python et d'une bibliothèque graphique libre d'origine pour ce même language, Tkinter. Nous nous sommes beaucoup inspiré de l'exemple disponible dans le sujet afin de créer notre interface.

Nous avons, dans un premier temps, créé un cadriage de 10 lignes sur 8 colonnes à l'aide de fonction "create_rectangle" qui font partie du module Canvas. Les textes sont affichés à l'aide la fonction "create_text" qui fait aussi partie du module Canvas.

Nous avons aussi décidé que les horaires de 6 heure à 10 heure du matin et les horaires de 20 heure à minuit ne sont pas disponible comme dans l'exemple proposé dans le sujet. Les horaires de minuit à 6 heure du matin ne font pas partie du planning car elles ne font pas partie des horaires de travail dites "classique".

Nous avons par la suite écrite une fonction qui nous permet de réserver un créneau sur le planning afin d'imprimer des documents. La fonction prend en entrée le nom du document et le nombre de pages que ce document contient. Lorsqu'il n'y a plus de place pour imprimer un document en une seule fois dans la journée (c'est-à-dire la fin de l'impression dépasse 20 heure), l'impression est configurée afin qu'elle début le lendemain à 10 heure du matin. Lorsqu'il n'y plus de place pour imprimer un document le samedi avant 20h, l'utilisateur reçoit un message comme quoi il doit réessayer la semaine prochaine car l'imprimante ne tourne pas le dimanche. Pour cette étape, on pourra déjà

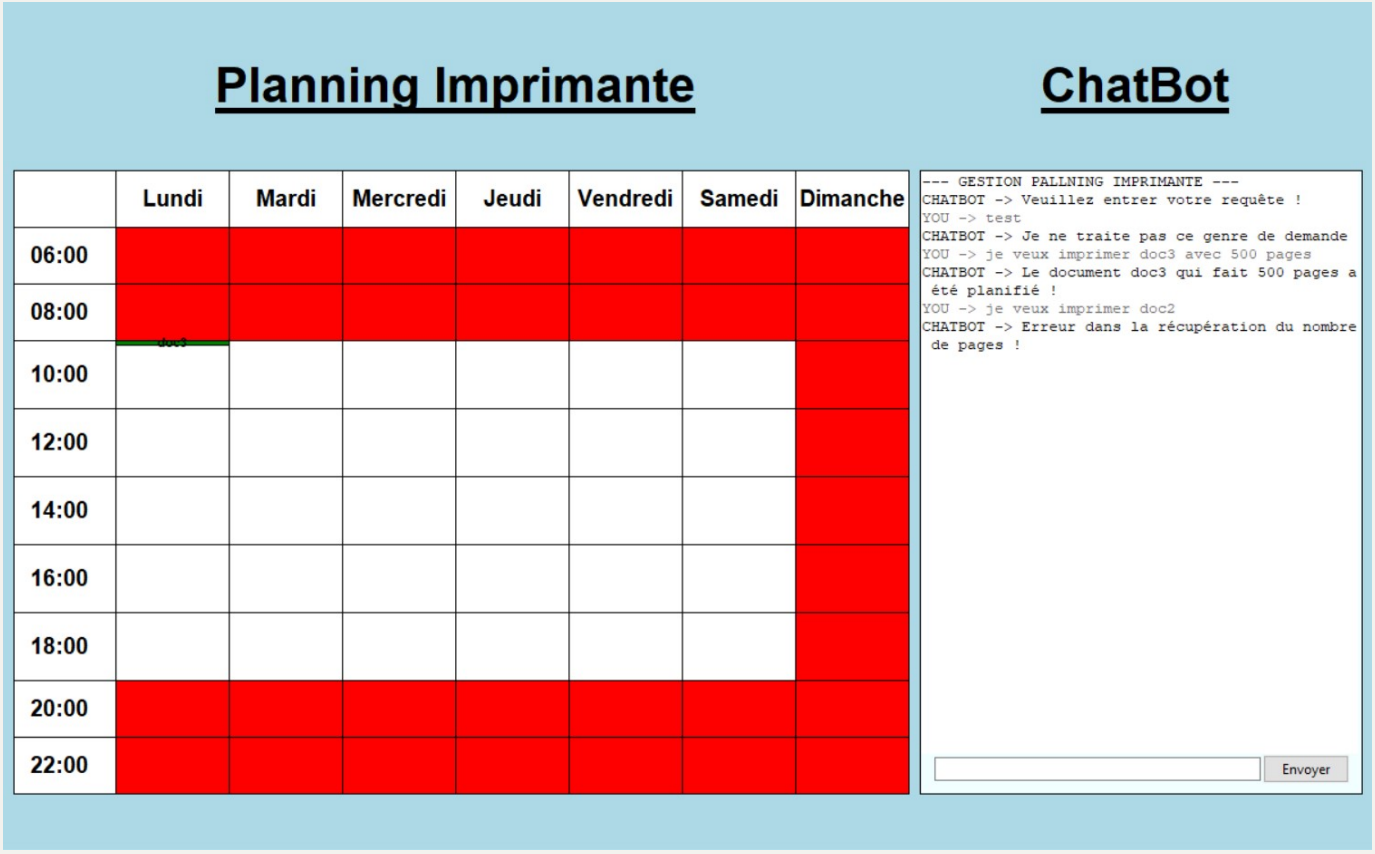
enregistrer les futures impression pour la semaine suivante sur un autre calendrier mais ceci serait trop gourmand en ressource. Notre exemple porte uniquement sur une semaine complète.

Chaque plage horaire de 2 heures est décomposé en 120 pixel de hauteur donc chaque pixel correspond à 2 minutes en terme de temps sur notre planning. En effet, nous ne pouvons pas faire plus grand par soucis de résolution. J'ai décidé de laisser 2 minutes de pause entre chaque impression afin que puisse voir les différents bloc dans le planning qui correspond aux document à imprimer. Cela veut dire que même si on fait une impression d'une page, nous pourrons le voir s'afficher dans le planning même si cela restera très petit.

Notre fonction ne prend pas en compte l'heure où la demande d'impression se fait. Cette fonctionnalité aurait pu être mise en place par la suite.

L'interface a été développée avant le module du chatbot, nous avons décidé de rajouter un espace pour le ChatBot dans la même interface que le planning mais on aurait bien pu faire deux interfaces différentes. Il existe beaucoup de moyen pour transmettre l'information du ChatBot au planning. Nous avons donc rajouter un petit module de chat afin que l'utilisateur puisse entrer ces requêtes et obtenir une réponse du bot en cas de succès et en cas d'échec.

Capture d'écran de l'interface finale :



Répartition du travail au sein du groupe :

Comme nous sommes que deux pour la réalisation de ce projet, le partage des tâches à été beaucoup plus simple. Nicolas WEISS s'est occupé principalement du ChatBot alors que Amaury SENSENBRENNER s'est occupé principalement de l'interface graphique. Puis dans un second temps, chacun à apporté sa contribution afin d'améliorer l'interface graphique, le ChatBot et les connexions qui fallait faire entre les deux. Le rapport a aussi été écrit par les deux membres du groupe.

Nicolas WEISS / Amaury SENSENBRENNER