1. Pré-traitement des datasets

- Télécharger les news (fakes et true).
- Effectuer une première analyse des deux *datasets*. Combien y-a-t-il de données dans chaque dans les *datasets*. La répartition est-elle équilibrée.
- Analyser les textes entre les *fake news* et les *true news*. Que constatez-vous ?
- On souhaite supprimer pour les agences de presse qui ont émis les news. Pour cela vous utiliserez la fonction *split* des *string* afin de découper vos ligne en deux au niveau du premier tiret (' – ').
- Cette découpe peut vous retourner zéro, une ou deux sous-chaines. Identifier les index correspondant aux découpes avec zéro ou une seule sous-chaine, et afficher les *rows* du *dataset* correspondant.
- Modifier votre *dataset* en ne conservant que l'information émise sans les agences de presse, ou une valeur vide lorsqu'il n'y a pas d'information.
- Combien y a-t-il de textes vides dans les deux *datasets*, supprimez-les.
- Effectuer la même chose pour les doublons.
- Ajouter une colonne '*fake*' avec la valeur 1 pour les *fake news* et 0 pour les *true news*.
- Le titre de la *news* peut être porteuse d'information. Vous ajouterez alors dans la colonne *text* le champs *title* à la fin de la colonne *text*.
- L'analyse de texte se fait généralement sur des mots en minuscule. Transformer vos données des champs *text* en minuscule.
- Créez un dataset (*Data*) regroupant les colonnes *text* et *fake* des datasets (normalement 44058 news). Vous réindexerez le *dataframe* obtenu.

2. Mise en forme des textes

On souhaite préparer les données de texte en vue de leur traitement. Pour cela on se doit de modifier ou de supprimer certains mots ou référence. De plus nous allons effectuer notre classification sur la présence de certains mots dans

le contexte plus que sur la structure des phrases. On pourra alors supprimer les mots les plus courants du langage (*stopwords*).

Remarque : Avec les instructions sur *re* en fin de fichier il est possible de traiter les questions suivantes.

- On constate la présence dans les textes de mots concaténés de la forme 2017trump ou 2017the … (soit ChiffreChaine). On souhaite des dissocier en utilisant la bibliothèque des expressions régulières *re*. Créer une fonction permettant de dissocier ce pattern : composé d'un blanc, suivi d'un entier et suivi d'une chaine composée de caractère minuscules uniquement.
- Appliquez via la fonction apply sur la colonne text cette fonction. Vérifiez que les formes ont bien été dissociées.
- On souhaite maintenant supprimer les formes de type @chaine et #chaine dans les textes. Pour cela vous devez créer une fonction permettant de remplacer par un blanc les patterns composé d'un blanc suivit d'un @ ou d'un # et suivi d'une chaine alphanumérique.
- Appliquez cette fonction aux textes votre data.
- On souhaite également supprimer les liens correspondant à des référence tels que pic.twitter.com/xtzw5pdu2b c'est-à-dire débutant soit par un blanc suivi éventuellement d'un '(', puis suivi d'un format avec un mot composé de caractères alphanumériques suivit d'un des trois caractères suivants [.\/\\] suivi ensuite d'un mot composé de caractères alphanumériques. Ce dernier format pouvant être répété plusieurs fois (+). Appliquer cette transformation.
- Il nous faut maintenant supprimer les caractères nom alpha numérique soit le pattern "[^\w\s]" par un blanc et ensuite remplacer les séries de blancs par un seul blanc.
- Enfin nous souhaitons supprimer des chaines tous les mots courants du langage comme : 'a', 'about', 'above'….'yourselves'. Il existe plusieurs bibliothèques permettant le traitement naturel du langage (nltk, spacy, grove…). Nous utiliserons *nltk* (mieux adaptée pour l'anglais que par exemple spacy) et en particulier la fonction *nltk.download('stopwords')* et ensuite nous devons télécharger le fichier correspondant à l'anglais *stopwords.words('english')*. Récupérez ces

données en enregistrez les dans un ensemble *set*. Affichez les mots correspondants.

- Créez maintenant une fonction qui reçoit un chaine de caractère la découpe via la fonction *split()* et ne retient que les mots qui ne sont pas des *stopwords*. Attention en découpant la ligne vous allez obtenir une liste L de mots que vous devez reconstituer en chaine via un *' '.join(L).*

3. Mise en forme des textes

Pour pouvoir être traitées les données textuelles doivent être mise en forme. Dans un premier temps les phrases doivent être composée de chiffres et non de mots, pour cela on peut utiliser des fonctions de *tokenisation*. Il faut ensuite que toutes les informations (news) fourni au model soient de la même taille (même nombre de mots). Les fonctions de *padding* permettent de réaliser cette tâche. Enfin, pour donner un sens aux mots en fonction du contexte on doit effectuer de l'*embedding* soit au cours de l'apprentissage dans le modèle soit avant.

- Affichez via la fonction *histplot* de *seaborn* la taille moyenne des mots et le nombre de mots par ligne pour les textes des *fake* et de *true-*. Que constatez-vous. Les news peuvent-elles être traitées en l'état.
- Combien y-a-t-il de news avec plus de 1000 mots. Quel pourcentage cela représente-t-il.
- Afin de préparer les données textuelles nous devons créer une liste X de toutes les news (44058) et chaque élément une liste des mots de la news. X sera donc une liste de liste.

Pour créer l'*embedding* des mots nous allons utiliser un de modèle les plus courant : *Word2vec* développé par Google. Ce modèle de construire un vecteur *Dense* porteur de sens, associé à chaque mot. La représentation des mots dépend du contexte dans lequel il se trouve, c'est-à-dire des mots qui sont autour de lui (fenêtre).

- Calculer l'*embedding* des mots de notre corpus (X), a partir du *models.Word2Vec* de *gensim*. Vous donnerez une taille de 100 pour les vecteurs d'*embedding* calculés à partir d'une fenêtre de taille 10. Il est également possible de ne pas prendre en compte lors de l'embedding les mots d'ont la fréquence d'apparitions est trop faible. On choisira ici les mots avec une fréquence de 1 (paramètre *min_count*). Cette

commande retourne un objet de type *Word2Vec* (que vous enregistrerez dans une variable *w2v*) qui contient l'*embedding* des mots du corpus.

- Afficher pour un des mots sa représentation. Ainsi que les mots les plus rapprochant de certains mots (exemple : 'trump', 'france', 'money' …) via la commande *wv.most_similar()* de l'objet retourné.

La tokenisation des mots consiste à les remplacer par des nombres. On utilise pour cela la classe *Tokenizer* de la bibliothèque *keras_preprocessing.text*.

- Créez un modèle tokenizer puis effectuer l'apprentissage sur les mots (liste X) grâce à la fonction *fit_on_texts*. L'apprentissage réalisé il est alors possible de transformer les mots en *token* via la commande *texts_to_sequence( )* appliquée à cette même liste
- Affichez quelques *token* avec leurs indices à partir de votre modèle *tokenizer* (méthode *word_index*) Puis afficher la première news X[0] afin de vérifier que vos textes ont bien été *tokenisés*.
- Il nous faut maintenant normaliser la taille des news. Le plus simple consiste à couper les textes à une taille fixée et à compléter par des 0 (index non fourni par la *tokenisation*) les textes plus courts. Utilisez la fonction *pad_sequences()* de *keras_preprocessing.sequence* pour limiter à 1000 le nombre de *token* par news.
- Vérifiez que la liste X est bien une liste (44058,1000). Affichez une des news afin de constater que l'on a bien que rempli par des 0 pour obtenir les 1000 *tokens* par news.
- Nous allons créer l'*embedding* des mots afin de le fournir à notre modèle qui n'aura pas besoin de recalculer les poids liés à nos mots. Le modèle *Word2vec* développé par Google, la bibliothèque *gensin* dispose d'un modèle pré-entrainé, permettant de créer l'embedding des mots d'un corpus.

On se doit maintenant de créer un lien entre les *tokens* (numéros des mots) et leur représentation *dense (embedding)*.

- Créez une fonction permettant de lier les tokens à leur représentation. Pour cela vous devez créer un tenseur (*wofw*) de dimension 2 (*numpy array* de zeros). La première dimension correspond au nombre de token (dans le modèle *tokenizer* crée) + 1 et 100. En effet nous avons un token de plus que ceux créés, permettant d'enregistrer le mots vide 0, et pour chaque *token* un vecteur *dense* de 100. Ensuite en parcourant le *tokenizer*

via la commande word_index.items() il est possible d'accéder pour chaque index à sa représentation et l'enregistrer dans *wofw.*

- Vérifier que vous avez bien un tenseur et que pour chaque token i de nos news nous avons bien accès son vecteur *dense* par *wofw[i]*.

4. Création du modèle

- Créez un modèle séquentiel. La première couche doit être une couche d'entrée, justifiez que cette taille est (1000,).
- Ajoutez une couche d'*embedding* qui doit recevoir en entrée la taille de notre corpus (size[0] de *wofw*), la taille des vecteurs d'*embedding*, les poids correspondants (puisque notre modèle est pré-entrainé) c'est-adire la *matrice* d'*embedding wofw*. De plus comme les poids sont déjà calculé la couche ne doit pas les mettre à jour lors de rétropropagation du gradient (*trainable=False*).
- Ajouter maintenant une couche de 128 *lstm* avec l'activation par défaut des *lstm* c'est-à-dire *tanh*.
- Pour la sortie ajouté une couche dense composé d'un seul neurone avec une activation de type sigmoid, permettant de classer les news. Si les valeurs sont <0.5 la sortie sera un 0 (true) et 1 (fake) sinon.
- Afficher via la fonction *summary* le structure du réseau. Comment sont calculé les paramètres de chaque couche.

5. Apprentissage et test du modèle

- Effectuer une découpe de vos données avec une taille de test=25%.
- Créez le modèle d'apprentissage avec une fonction d'optimisation de type *adam*, une fonction de perte de type *crossentropy* et une *accuracy* pour la métrique.
- Réalisez l'apprentissage sur moins de 10 *epoch* (le temps risque d'être assez long), avec des *batch* de taille 32 et un taux de validation de 25%.
- Affichez les résultats obtenus.
- Effectuer une prédiction sur des données de tests. Les résultats sont-ils conformes à ceux de la validation.

Bibliothèque re :

Les expressions régulières sont utilisées pour le traitement du langage. Elles permettent de repérer des structures (suites de caractères) dans les phrases et de les modifier ou de les supprimer.

Dans cet exercice vous pourrez utiliser deux des fonctions de re :

- re.split(pattern, str) : qui permet de découper une chaîne str selon un pattern ;
- re.sub(pattern, ch, str) : qui permet remplacer des sous-chaines identifiées par le pattern par la chaine ch dans str ;

Un pattern est une chaîne de caractères correspondant à un motif qui se présent sous la forme r'(motif-1)(motif-2)…' ou r'[car1,car2…]'

- (motif-1) : définit un motif suite de un ou plusieurs caractères ;
- [car1,car2…] : identifie une série de caractères ;
- le signe + placé après un série permet de répéter un ou plusieurs fois les caractères :
    - ([a-z]+) identifie des mots composé de minuscules (ms) ;
    - ([a-z0-9]+) identifie des mots composé de minuscules (ms) et de chiffre (mc) ;
    - [a-z0-9.@]+ identifie les formes : mc.ms@ms.mc ; mc@ms …
- le signe ? placé après un série correspond à 0 ou une occurrence :
- l'info {n,m} placé après une série correspond à au moins n occurrences de la série et jusqu'à m occurrences. m peut-être omis {n, }, donc pas de limite pour les occurrences.

Le signe \ devant un caractère permet de particulariser les caractères :

- \w : un caractère alphanumérique (lettre, chiffre, ou underscore) ;
- ^\w : tous les caractères non alpha numériques ;
- \s : un espace, un tab, ou un retour à la ligne ;
- \d : un chiffre équivalent à [0-9] ex : (\d+) ou ([0-9]+) un nombre ([\d.]+] ou ([0-9.]+) un nombre à virgule …

Certains caractères ont une signification pour re, comme (, [,\,/… pour les identifier en tant que caractères non spécifique il faut les faire précéder de \. [\(\\\/] permet par exemple d'identifier les caractère (,\,/)