Projet TALN

SemEval Task 8 - 2017

Enzo Poggio

M1 Informatique pour sciences humaines Unige

github.com/EPgg92/SE8-2017

Introduction:

Pour mon projet du cours Traitement Automatique de la Langue Naturelle, Madame Merlo et moi-même trouvons l'idée bonne de participer à une tâche partagée, proposée par SemEval. Nous avons choisi une tâche proposée en 2017 qu48i se nomme "*Task 8: RumourEval: Determining rumour veracity and support for rumours*". Ici nous nous intéresserons seulement à la sous-tâche A à savoir:
L'analyse du discours environnant pour déterminer comment les utilisateurs dans les médias sociaux considèrent une rumeur, à partir d'un tweet source. Pour cette sous-tâche il nous est fourni une conversation structurée par un arbre, formé de tweets répondant aux tweets lançant la rumeurs. Chaque tweet présente son propre type de soutien à l'égard de la rumeur. Ces tweets et ces tweets-réponses sont classés en termes de soutien, de refus, d'interrogation ou de commentaire (SDQC). Donc la sous-tâche a pour objectif de marquer le type d'interaction entre une déclaration donnée (tweet rumeur) et un tweet de réponse (ce dernier peut être une réponse directe ou imbriquée).

Une discussion sur Tweeter se représente ainsi:

Tweet A:

Tweet B: réponse au tweet A

Tweet D: réponse au tweet B

Tweet C: réponse au tweet A
Tweet E: réponse au tweet A

Tweet F: réponse au tweet E

Les données de *SemEval 2017 task 8*:

Pour chaque tweet source nous avons plusieurs informations:
- Un json contenant le tweet source et toutes ses données relatives.
- Les tweets réponses dans un sous dossier avec leurs données relatives.
- L'arborescence des tweets réponses.
- Les urls qui sont cités dans le tweet sources et ses réponses.

Les urls ne nous servent pas. En effet, la sous-tâche A est pensée close. Nous ne pouvons utiliser que les données fournie par SemEval. Donc nous devons uniquement utiliser les textes des tweets (et d'autres méta-données fournies) pour les classifier.

Structures de données:

Afin d'utiliser au mieux les données, nous avons décidé d'utiliser des objets. En effet, les

objets permettent de faire une enveloppe simple, contenant facilement des attributs et des méthodes spécifique aux tweets sources et aux réponses. J'ai donc trois objets : *Data*, *Tweet*, *Reply*.
Tweet et *Reply* sont des sous-classes qui héritent de la super-classe *Data*.
Chaque classe a un getter et un setter pour chacun de ses attributs. Les signatures des différentes classes sont les suivantes:

- *Data(data, subject, categorie)*
- *Tweet(data, subject, categorie, structure)*
- *Reply(data, subject, categorie, source_tweet)*

Le paramètre *data* est un dictionnaire contenant le texte du tweet (*text*), son identifiant (*id*) et les autres méta-données fournies. L'attribut *text* est pour le moment une liste de mot qui a été décapitalisée, tokenisée puis lemmatisée. Les paramètres *subject* et *categorie* sont récupérés pour l'analyse de données et la classification. Le paramètre *structure* est un dictionnaire contenant l'arbre de réponses du tweet-source. Le paramètre *source_tweet* permet de connaître d'où provient la réponses, donc à quel tweet source elle appartient. Elle initialise l'attribut *source*. L'attribut *vector* et un tableau *numpy* instancié vide. Il accueillera lors de la vectorisation un vecteur créé à partir de *text*.

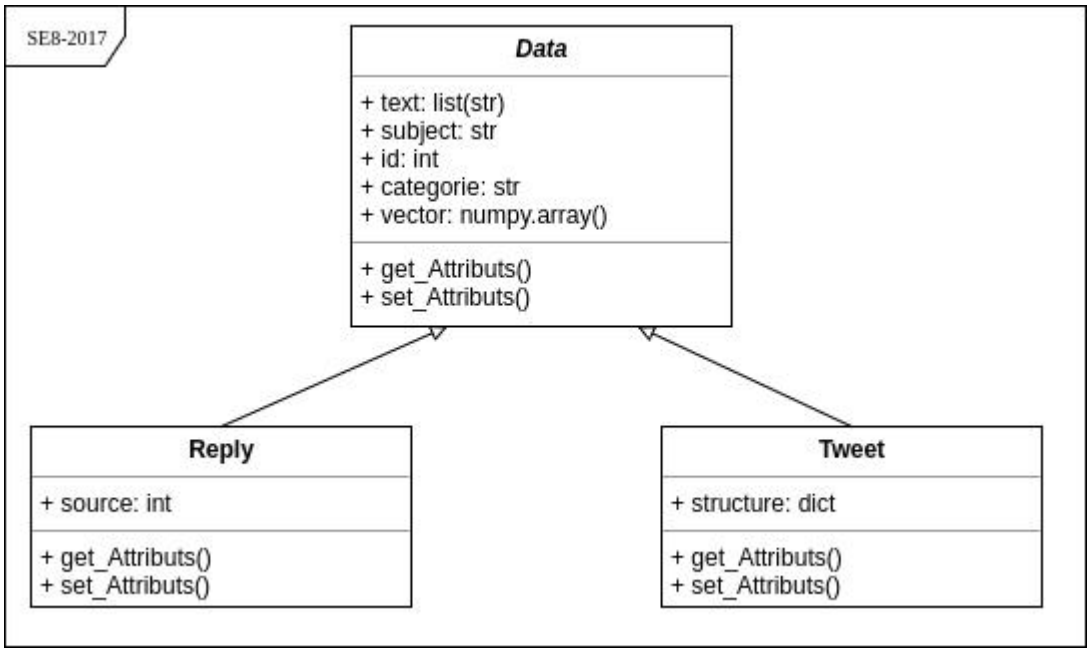


Figure 1 Diagramme des classes

Analyse de données:

Pour effectuer à bien cette classification nous devons avoir une vue exhaustive des différentes dimensions de ce corpus. J'ai fait des analyses à partir du corpus d'entraînement sur trois niveaux:
- La répartition des catégories en général (qui inclut la répartition des catégorie par sujet).

- La répartition des catégories par classe d’objet
- La répartition de la fréquence des mots par catégorie

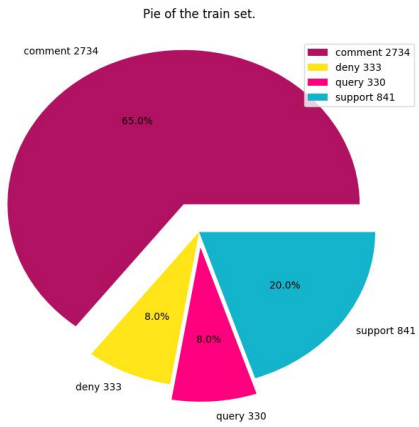


Figure 2 Répartition des catégories dans le corpus d'entraînement

La répartition des catégories en général nous a permis de voir que la classe majoritaire était les commentaires. Ils sont suivis par les supports, correspondant à 1/5 des données. Ils sont eux-même suivis par les refus et les questionnement, tous deux à 8%. Il y a donc un biais important en ce qui concerne la réparation des classes. Il faudra prendre en compte que les commentaires sont majoritaire et peut-être en faire notre catégorie par défaut. À partir de cette analyse nous faisons notre première *baseline* sur cette classe majoritaire. Il y a 65% de commentaires. Dans l’ensemble de test il y a 61,5% de commentaires notre but sera donc de les dépasser. Pour les répartitions par sujet cela suit le tableau ci-dessous :

	Comment	Deny	Query	Support
charliehebdo	67	22	5	5
ebola-essein	62	18	3	18
ferguson	66	16	9	8
ottawashooting	61	21	8	10
prince-toronto	71	22	11	7
putinmissing	53	29	8	10
sydneysiege	63	20	9	8

Tableau 1 Répartition des catégories par sujets

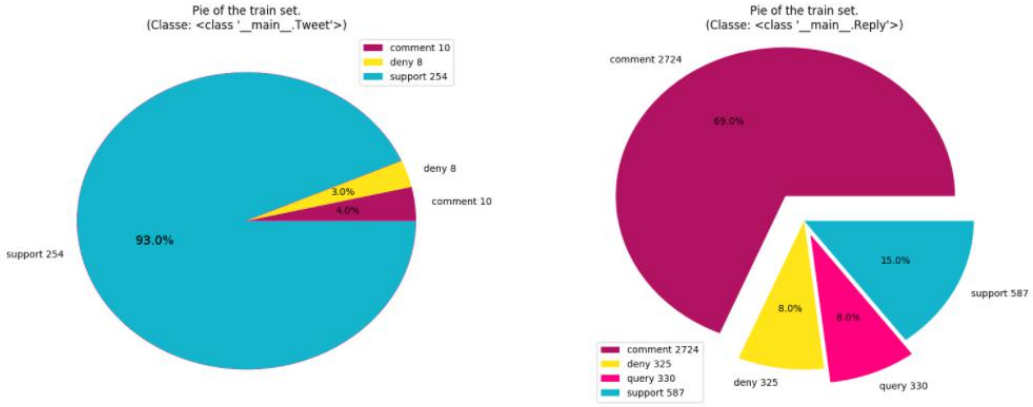


Figure 3 Répartition des catégories par objets

La répartition des catégories en fonction des classes d'objets marque une différence significative. Pour les objets *Reply*, la répartition est assez proche de la répartition générale. Par contre en ce qui concerne les objets *Tweet*, la classe grandement majoritaire est les supports (le contraire aurait été bizarre vu que le tweet lance la discussion sur la rumeur). Cela pourra être une information très utile pour classer les tweets. Les tweets source, qui sont des supports, représentent 30% du corpus.

Ici nous avons représenté les distributions Zipfiennes de la fréquence des mots par catégorie, ordonné du plus au moins fréquent. Sans grande surprise *the* est le mot au premier rang, toutes catégories confondues. *The* n'est pas un trait distinctif. Par contre tous les mots suivants sont différents ou n'ont pas le même rang selon la catégorie.

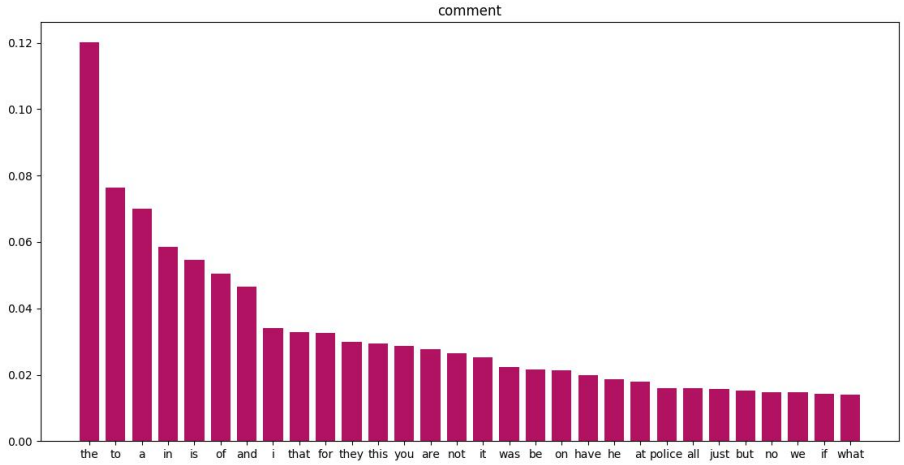


Figure 3 Répartition Zipfienne des fréquences de mots des commentaires

Le vocabulaire qui ressort en majorité des commentaires, est un vocabulaire assez usuel.

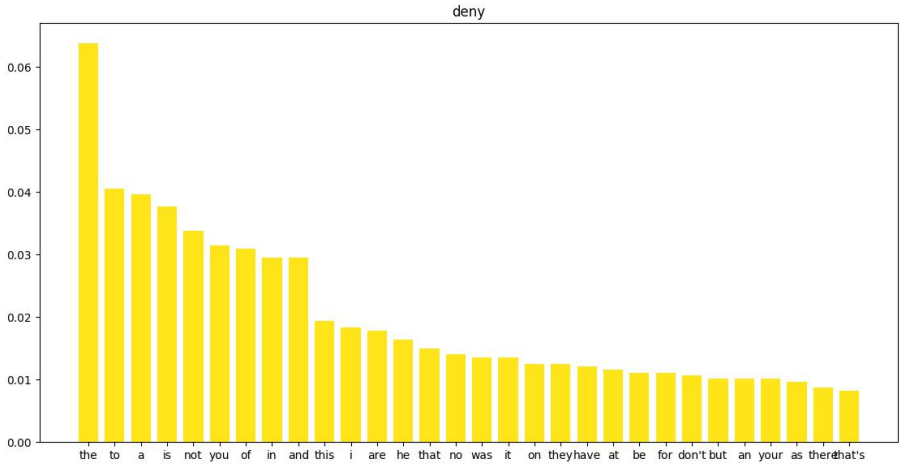


Figure 4 Répartition Zipfienne des fréquences de mots des refus

On voit des mots de négation et un vocabulaire d'interpellation se démarquer.

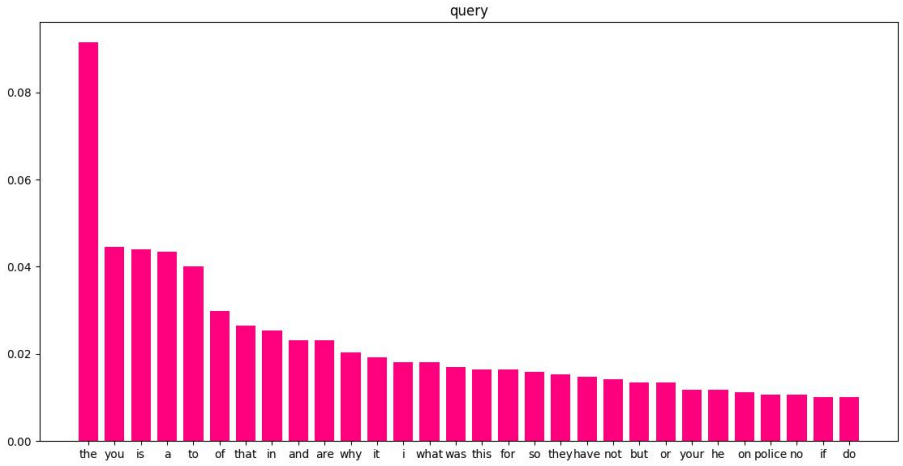


Figure 5 Répartition Zipfienne des fréquences de mots des questionnements

Encore plus que la catégorie refus: on voit clairement des mots interrogatifs et un vocabulaire d'interpellation se démarquer.

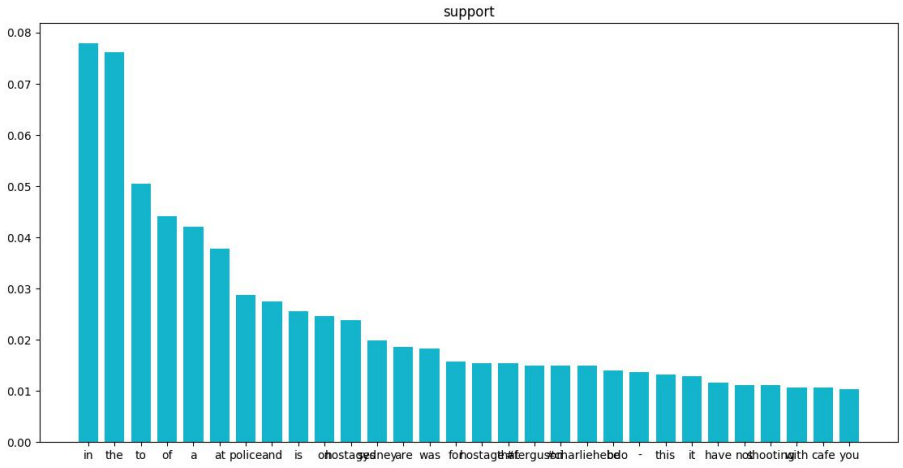


Figure 6 Répartition Zipfienne des fréquences de mots des supports

Les supports n'ont pas l'air d'avoir un vocabulaire clairement propre. Il y a beaucoup de mots usuels (comme pour les commentaires).

Première approche :

Vous retrouverez sous le répertoire "*old*" la première version du code qui avait été faite pour résoudre la tâche. À cette époque j'avais mal compris l'énoncé du problème et je ne pensais pas qu'il fallait uniquement trier les réponses aux tweets, et non les tweets sources eux-mêmes. J'avais donc un manque dans ma classification d'après le *scorer*. Il m'indiquait qu'il manquait 272 classifications; ce qui correspond au nombre de tweets sources. Nous n'allons pas représenter les résultats qui avaient été obtenus avec notre corpus incomplet, car cela ne fait pas de sens de se baser sur des résultats incorrects. Par contre, certaines observations demeurent vraies. En effet, dans cette solution, nous utilisons simplement la fréquence de mots par catégorie, afin de calculer l'*argmax* sur la liste des produit des fréquences de mots par catégorie. On remarque notamment que plus la probabilité des mots inconnus est lissée avec une valeur basse, plus les prédictions de classes sont bonnes. Ce modèle est un algorithme *naive bayes* en soit.

Avec un numérateur de mots inconnus à 10^{-11} on obtient ces résultats:

C:pred	comment	deny	query	support
comment	124	18	18	13
deny	5	1	1	4
query	12	3	9	4
support	45	11	3	10

Tableau 2 Table de confusion pour la première approche

Soit 51.25% d'exactitude.

Deuxième approche :

Traitement des données d'entrée:

Dans cette seconde approche, nous tentons des solutions de *machine learning* après avoir remarqué la complexité de la tâche de la première approche. Nous avons donc vectorisé les informations textuelles de deux manières pour le moment:

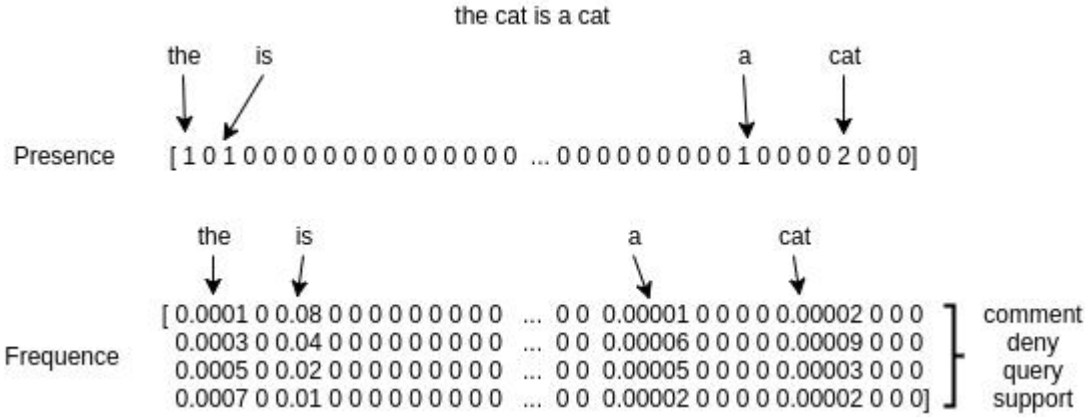


Figure 7 Différentes représentations vectorielles

Les vecteurs de présence contiennent le nombre de fois qu'un mot est vu dans le tweet. Chaque dimension du vecteur représente un mot du vocabulaire total des tweets d'entraînement.
Les vecteurs de fréquence contiennent 4 fois plus d'informations que les vecteurs de présences. Chaque quart de vecteur représente la fréquence relative par catégorie de chaque mot du tweet. Nous ne présenterons pas les résultats des vecteurs de fréquences, en effet, les résultats étant trop mauvais, nous n'avons pas continué d'approfondir cette hypothèse.

Modèles:

La vectorisation de présence a été testée sur deux modèles. Nous allons ici vous les présenter.

Naive Bayes:

En utilisant la librairie *Scikit-Learn*, nous avons créé un modèle naive bayes utilisant une fonction gaussienne pour la classification. Ce modèle prend des vecteur en entrée et ressort une classification en sortie.

$$P(C \mid A) = \frac{P(A \mid C)P(C)}{P(A)}$$

Neural network:

Notre second modèle est un réseau neuronal propulsé dans un environnement *TensorFlow* grâce à *Keras*. Celui-ci prend les mêmes entrées et donne les mêmes classifications que le modèle précédent. Ce modèle neuronal contient trois couches, donc une couche cachée. Les 2 premières couches ont 100 unités qui s'activent avec "relu". La dernière, contient 20 unités qui s'activent avec un "softmax".Résultats:

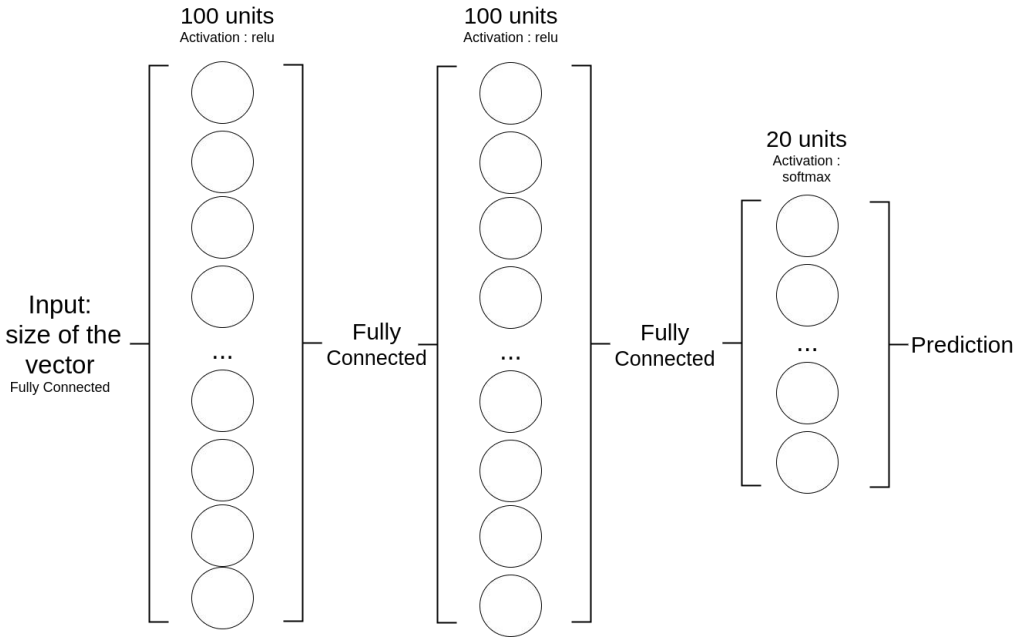


Figure 8 Schéma du réseau neuronal

Résultats:

Comme précédemment, les prédictions pour le modèle Naive Bayes de Scikit-Learn ne sont pas très hautes:

C:pred	comment	deny	query	support
comment	87	12	35	39
deny	5	0	3	3
query	14	1	7	6
support	47	6	4	12

Tableau 3 Table de confusion pour le modèle Naive Bayes

Avec une exactitude de 38%, on voit clairement que le passage aux vecteurs pose un problème. Certaines informations pondérées en tant que fondamentales influencent une erreur de classification. Mais utilisée avec des systèmes plus intelligents, la vectorisation a son importance.

Pour les réseaux neuronaux, nous vous présentons les meilleurs résultats en fonction du nombre de périodes d'entrainement. Ce nombre varie entre 50 et 1000 modulo 50. Comme vous pouvez le voir dans l'image suivante (figure 9), le meilleur nombre de période est 850.

C:pred	comment	deny	query	support
comment	160	0	3	10
deny	8	0	2	1
query	11	0	17	0
support	45	0	2	22

Tableau 4 Table de confusion pour le réseau neuronal

Avec une exactitude de 71%. La classe *deny* n'est clairement pas distinguée des autres. L'efficience de ces résultats réside dans le fait que le système est beaucoup entraîné. Mais cela a un coût non négligeable, puisqu'il a fallu plus d'une heure pour entrainer le modèle.

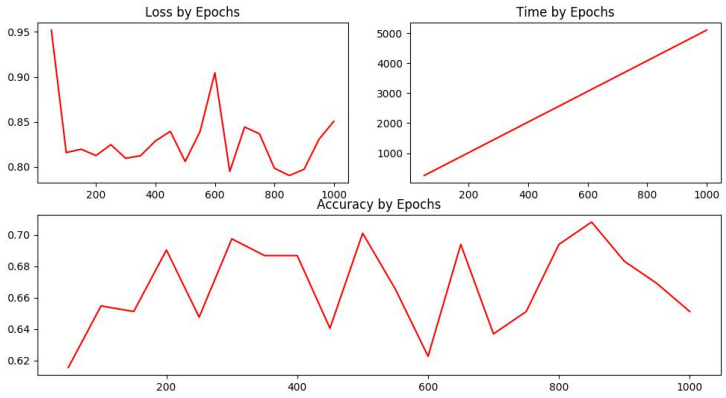


Figure 9 Variables en fonction du nombre de période

Ceci est une mauvaise pratique! En effet, on ne doit pas faire du "cherry picking" sur des données a posteriori pour améliorer son modèle. Par la suite, nous ne retiendrons pas cette constatation: 850 périodes permettent d'avoir des résultats optimaux. Toutefois, cette observation nous permet de comprendre le comportement du modèle sur-entraîné. À savoir qu'il est inutile de le sur-entrainer; car l'exactitude fluctue entre 62% et 70% sur les périodes de 200 à 800 (elle ne dépasse jamais les 71%).

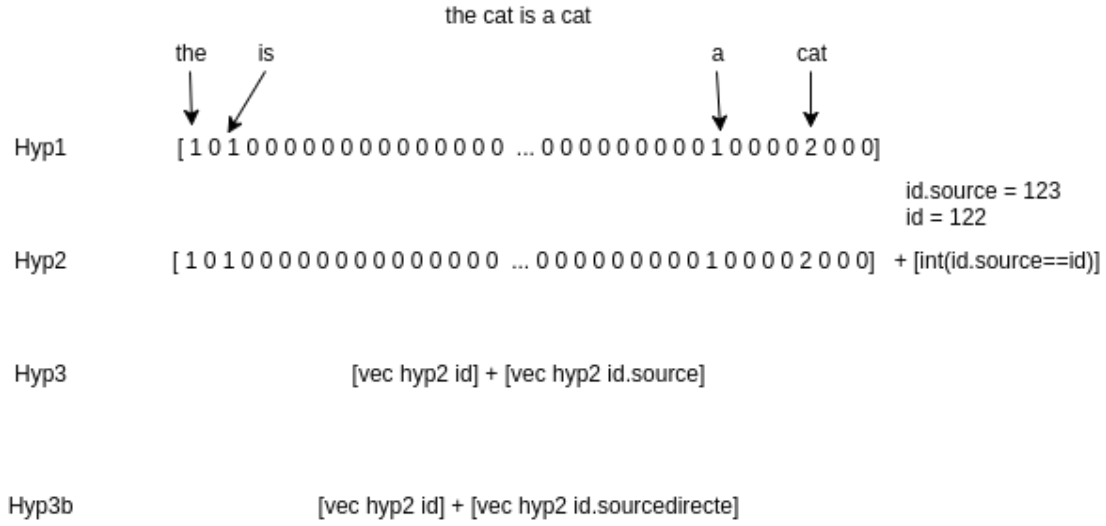
Troisième approche :

Pour la troisième partie, nous avons dû faire beaucoup de tests d'hypothèses. Il a donc fallu régénérer des objets régulièrement, ce qui était long et pénible. Pour remédier à cela nous avons généré différents *json* contenant, pour certains, les données d'entrée nécessaires, pour d'autres, des vecteurs et des labels (ces fichiers sont générés grâce aux sous-script

commençant par *create_*.py*).

Traitement des données d’entrée:

Dans cette section, nous décrivons les différents ensemble de données que nous allons utiliser en expliquant comment nous avons vectorisé les informations.



L’ hypothèse Hyp1 utilise la même vectorisation que les vecteurs de présence décrits précédemment. L’ hypothèse Hyp2 ajoute un 1 ou un 0 à la fin des vecteurs Hyp1 si le tweet est un tweet source ou une réponse. L’ hypothèse Hyp3 concatène le vecteur Hyp2 du tweet traité avec le vecteur Hyp2 de la source de celui-ci (si le tweet traité est lui-même un tweet source alors on concatène deux fois le même vecteur). L’ hypothèse Hyp3b est la même que Hyp3 mais seulement, le tweet source est celui directement au dessus dans l’arborescence des tweets.
Exemple :

Tweet A:

Tweet B: réponse au tweet A

Tweet D: réponse au tweet B

Tweet C: réponse au tweet A

Tweet E: réponse au tweet A

Tweet F: réponse au tweet E

Hyp3 du Tweet F: [hyp2 vecteur F]+[hyp2 vecteur A]
Hyp3b du Tweet F: [hyp2 vecteur F]+[hyp2 vecteur E]

Les hypothèses Hyp4, Hyp5, Hyp6 et Hyp6b sont les mêmes spécifications de vectorisation que Hyp1, Hyp2, Hyp3 et Hyp3b respectivement, mais au lieu d’utiliser des *BOW* on utilise des *bag of 3-grams* sur les caractères.

Modèles:

Ici, nous décrivons les changements que nous avons appliqué au différents modèles entre la deuxième et la troisième approche.

Naive Bayes:

Nous utilisions dans la librairie scikit-learn le Naive bayes gaussien (GaussianNB). Ceci n'était pas une bonne idée. En effet l'objet GaussianNB est plus adapté pour les données qui sont continues. Nous utilisons alors l'objet MultinomialNB, les données étant distribuées multinomialement, cela est plus utile dans la classification de textes.

Neural network:

Pour notre modèle de réseau neuronal, nous avons fait une convergence sur l'exactitude pendant l'entraînement. Ainsi, si le réseaux arrive à 0.94 d'exactitude sur l'ensemble d'entrainement, on estime que le réseaux est suffisamment entrainé. Ceci sert à éviter l'*over-feating* du réseau. Aussi, si le réseau dépasse les 300 périodes d'entraînement, alors on arrête l'expérience car le réseau n'arrive pas à converger.

Résultats:

	Hyp1	Hyp2	Hyp3	Hyp3b	Hyp4	Hyp5	Hyp6	Hyp6b
NB	0.615	0.701	0.644	0.697	0.615	0.661	0.597	0.661
NN	0.615	0.693	0.704	0.690	0.615	0.658	0.619	0.669
Delta	0	0.008	0.06	0.007	0	0.003	0.022	0.008
Epoch	51	54	44	41	27	27	20	26

On remarque que les modèles avec les 3-grams sur les caractères convergent bien plus vite mais n'ont pas de meilleurs résultats. Les tweets étant seulement une chaîne de 180 caractères et avec un si petit corpus d'entrainement, il est certainement difficile de créer une grammaire implicite à partir des 3-grams.

Modèle Naive bayes Hyp2 :

C:pred	comment	deny	query	support
comment	160	2	0	11
deny	10	0	0	1
query	27	0	0	1
support	32	0	0	37

Accuracy: 0.701067615658363
Fscore: 64%
Precision: 61%
Recall: 70%

Les classes deny et query sont complétement négligées. L'ensemble de données d'entrainement les sous-représente déjà, il est donc normale que NB n'arrive pas à les retrouver non plus. Par contre, la classe support a bien été représentée grâce à l'unique dimension mentionnant si le vecteur représente un tweet source ou un tweet réponse.

Modèle Neuronal Network Hyp3:

C:pred	comment	deny	query	support
comment	160	0	5	8
deny	10	0	0	1
query	15	0	13	0
support	44	0	0	25

Accuracy: 0.7046263345195729
Fscore: 67%
Precision: 68%
Recall: 70%

La classe deny n'a vraiment pas assez de traits distinctifs pour ne pas être confondue avec les comment. Le progrès par rapport à NB avec l' Hyp2 est quasi nul. La convergence était peut-être trop rapide. Mais les autres essais qui ont été fait avec une convergence sur une exactitude plus haute ne donne pas de meilleurs résultats.

Classement:

Rank	Team	Score
1	Turing	0.784
2	Uwaterloo	0.780
3	ECNU	0.778
4	Mama Edha	0.749
5	NileTMRG	0.709
6	EPgg92	0.704
7	IKM	0.701
8	IITP	0.641
9	DFKI DKT	0.635

Améliorations:

Nous aurions pu explorer d'autres hypothèses pour améliorer nos résultats:

Utiliser la fréquence des classes du corpus d'entraînement : utiliser les probabilités *a priori* des classes dans l'ensemble d'entrainement pour soumettre une prédiction de test avec la même proportion.

Utiliser les identifiants utilisateurs: pour voir si ceux-ci répondaient à ses propres tweets et donc augmenter ainsi la classe support.

Équilibrer le corpus : multiplier les vecteurs sous-représentés ou travailler sur des combinaisons de vecteurs au sein d'une même classe.

Conclusion:

Ce projet montre bien la difficulté de travailler avec des tweets; des phrases qui sont informelles, mal orthographiées, pleine de symboles, de liens et surtout très courtes. Retirer des traits d'un ensemble aussi disparate est presque impossible. La thématique est compréhensible et le but est louable; on veut pouvoir évaluer la véracité de l'information grâce aux différentes interactions que des utilisateurs ont sur le réseau social Tweeter. Mais il est légitime de se demander si Tweeter est justement la bonne plateforme pour échanger, et donc évaluer des échanges (certes il offre un format normalisé, mais impose beaucoup de contraintes à la sémantique).