**Résumé des articles pour le projet de NLP**

*Dans la mesure où on a déjà une dataset labellisée, on s’en fiche de comment ils ont obtenu le jeu de données. C’est plus leur méthode (feature selection, feature extraction, classification model) qui nous intéresse.*

*Peut-être commencer avec des features simples.*

*Globalement, voici comment les chercheurs ont fonctionné :*

* *Ils choisissent plusieurs types d’attributs. Les plus utilisés sont (les lister) :* 
  + *Feature 1*
  + *Feature 2*
* *ils calculent les attributs avec des librairies / API comme AlchemyAPI ou NLTK, Gensim ;*
* *Ils font un feature selection pour conserver les attributs les + discriminants. Typiquement, sklearn.feature\_selection.SelectFromModel avec RandomForest par exemple ;*
* *Ils font une classification binaire (“check-worthy” ou non), et le ranking se fait par la probabilité donnée par le modèle que la phrase soit classée “check-worthy”.*

*Certains ont ajouté une heuristique pour remplacer le score de check-worthiness prédit par le classifieur selon la phrase considérée selon des règles.*

*D’autres se basent uniquement sur les fréquences des mots et la distorsion de textes.*

*D’autres se basent encore sur le learning to rank, pas de classification en soit.*

*Librairies Python utilisées dans les articles :*

* *AlchemyAPI (*[*https://pypi.org/project/AlchemyAPI/*](https://pypi.org/project/AlchemyAPI/)*): calcul de score de sentiment, extraction entity type*
* *NLTK (*[*https://www.nltk.org/*](https://www.nltk.org/)*) : tokenization, POS Tag*
* *Gensim (*[*https://pypi.org/project/gensim/*](https://pypi.org/project/gensim/)*) : détermination de topics*
* *sklearn pour tout ce qui est modèles de classification ou feature selection*

*Voir dans le dépôt Github du projet pour les articles :* [*https://github.com/Arthur8754/NLP-project*](https://github.com/Arthur8754/NLP-project)

**ClaimBuster :**

Problème de classification 3 classes :

* Non factual sentence ;
* Unimportant factual sentence ;
* Check-worthy factual sentence.

5 types de features retenus pour chaque phrase (sentence-level) :

* Sentiment : déterminé par AlchemyAPI ;
* Longueur : tokenization par NLTK puis calcul de la longueur ;
* Word : embedding des mots, avec un tf-idf. 6130 mots retenus dans le document (on enlève les mots apparaissant dans moins de 3 phrases)
* Parts of Speech Tag (POST) : on dit ce qu’est chaque mot grammaticalement (nom, verbe, adverbe, nombre…). Fait avec NLTK. Ils ont 43 POS tags différents avec leur doc.
* Entity Type (ET) : extraction des types d’entités de la phrase (personne, institution…). Ils ont 26 types d’entités.

Au total, ils ont 6201 features (1 sentiment, 1 longueur, 6130 words, 43 POS, 26 ET).

Pour réduire le nombre d’attributs et éviter l’overfitting, ils ont fait une feature selection. Pour ça, ils ont entraîné un random forest pour lequel ils ont utilisé l’impureté de Gini pour mesurer l’importance des features (les + discriminants). Au final ils ont retenu les 100 features les + importants.

Comment ça marche feature selection avec RF : <https://towardsdatascience.com/feature-selection-using-random-forest-26d7b747597f> + l’impureté d’un noeud (d’un feature) est faible, + le feature est important. La construction d’un arbre de décision recherche les attributs les + discriminants (cf IA en 2A, le TD sur les arbres de décision). Utiliser sklearn.feature\_selection.SelectFromModel, en prenant comme estimateur un random forest classifier.

Pour la classification, ils ont d’abord juste tenté en prenant les word, puis word + POS, puis word + POS + ET, puis les 100 features les + discriminants. Pour les modèles, ils ont utilisé Multinomial Naive Bayes Classifier (NBC ), Support Vector Classifier (SVM ) and Random Forest Classifier (RFC ), avec 4-fold CV.

Au niveau des résultats, ils atteignent jusqu’à 85% de précision et 65% de recall pour la détection de check-worthy factual claims.

Limites :

* Pas de contexte. Les phrases sont indépendantes. Après pour des tweets c’est peut-être moins gênant, parce que les tweets contiennent peu de phrases.
* Pas assez de features, on peut peut-être aller un peu plus loin.

**ClaimBusterWithContext :**

Problème de classification binaire, où classe positive = “check-worthy”.

À la différence de ClaimBuster, ajout de features prenant en compte le contexte, les phrases précédentes :

* Sentence-Level Features :
  + Les 5 types de features de ClaimBuster ;
  + Sentiment (version différente de sentiment ClaimBuster) : ils se sont servis d’une dataset NRC Emotion Lexicon (<https://saifmohammad.com/WebPages/NRC-Emotion-Lexicon.htm>) qui à un mot associe son sentiment (positif ou négatif) et ils ont compté, pour chaque mot de la phrase, le nombre de mots positifs et le nombre de mots négatifs. C’est pas le même feature que ClaimBuster car la façon d’obtenir les sentiments est différente.
  + Named Entities (version différente de ET ClaimBuster) : les phrases avec un nom ont + de chances d’être check-worthy. Au lieu de prendre tous les ET, ils en retiennent un : Named Entity (le nombre d’entités noms dans la phrase), avec NLTK.
  + Linguistic features : ils ont compté le nombre de mots dans la phrase qui appartiennent à chacun de ces lexiques : biais, négatifs, positifs, factuels, assertive, implicative, strong, weak (cf datasets dans l’article)
  + Tense (version différente de POS tag dans ClaimBuster) : la plupart des check-worthy sentences font référence à des événements passés. Donc création d’un tag qui dit si c’est passé, présent, ou futur verbe.
  + Longueur (version différente de longueur dans ClaimBuster) : les phrases courtes ont + tendance à être “check-worthy”. Ils ont créé un feature avec la longueur en caractères (et non en tokens comme avec ClaimBuster (CB)).
* Contextual features :
  + Position de la phrase : position de la phrase dans le segment de phrases (par rapport au début et à la fin). 3 phrases dans le segment j’ai l’impression.
  + Tailles du segment : Tailles de chaque phrase dans le segment (taille de la phrase d’avant, de la phrase courante, et de la phrase d’après).
  + Metadata : qui parle, est-ce qu’il s’adresse à son adversaire… Pas trop utile dans notre contexte je pense.
* Mixed features : à la fois au niveau de la phrase et avec le contexte.
  + Topics : certains sujets ont + de chances d’être check-worthy. Pour déterminer le topic de la phrase ou du groupe de phrases, on peut utiliser la librairie gensim. Pour la représentation, ils ont utilisé la distribution des topics pour la représentation des phrases. Ils ont également calculé la similarité cosinus avec cette représentation entre les segments précédents, courants, et suivants.
  + Embeddings : avec word2vec. lls ont également calculé la similarité cosinus avec cette représentation word2vec entre les segments précédents, courants, et suivants (comme au-dessus).
  + Discourse : ils ont utilisé un discourse parser (<https://alt.qcri.org/tools/discourse-parser/>) pour extraire 18 features contextuels.
  + Contradictions : une contradiction entre 2 phrases est souvent le signe d’un check-worthy. ils ont compté le nombre de négations dans la phrase, puis le nombre de négations dans les 2 phrases les + proches.
  + kNN : on compte le nombre de matchs entre les phrases de train et de test, et on multiplie par -1 si pas check-worthy, 0 si interlocuteur différent.

Ils ne parlent pas de feature selection. Peut-être qu’ils ne l’ont pas fait. Pour les modèles de classification, ils ont testé avec SVM (rbf kernel) et avec feed-forward neural network (FNN). Ils ont également fait une 4-fold CV, comme ClaimBuster.

Pour les mesures de perfs, ils ont utilisé des mesures de ranking telles que Precision at k (P@k) et Mean Average Precision (MAP) : <https://stackoverflow.com/questions/55748792/understanding-precisionk-apk-mapk>

* P@k : k est le nombre de tweets évalués, et P@k est la précision sur ces k tweets (c’est comme la précision, mais on se limite à k entrées pour évaluer). Autrement dit : sur les tweets que j’ai considérés comme check-worthy (sachant que j’ai évalué k tweets), combien sont réellement check-worthy ? **Mais dans le contexte du ranking, c’est : parmi les k tweets les + check-worthy dans la classification (les k premiers tweets dans le ranking), combien sont réellement check-worthy ?**
* AP : c’est la moyenne des P@k.
* MAP : c’est la moyenne des AP sur tous les utilisateurs. Pour nous, c’est sur toutes les datasets utilisées un peu (dans leur article ils font la moyenne sur les discours).

Leurs résultats je le comprends pas trop. Ils disent que ça marche bien, et c’est vrai que le P@5 est bon par ex, et + on va vers k grand + ça baisse, mais c’est assez logique en soit car + on va loin + les tweets ont une proba faible d’être check-worthy.

**ClaimRank :**

Rien de nouveau par rapport à l’article précédent. Juste ils utilisent un RN à 2 couches cachées. Leur “nouveauté” est que c’est quelque chose en ligne.

**PriseDeFer :**

Globalement ils ont fait à peu près la même chose que ClaimBusterWithContext vis-à-vis des choix des features. Mais leur contribution réside dans l’ajout d’une heuristique.

Le modèle de classification (basé sur un SVM ou un réseau de neurones), puis ils ont appliqué une heuristique basée sur des règles (spécifiques au fait que ce soit de la politique) pour override (écraser le score initialement mis).

**Copenhagen :**

Ils ont utilisé un RNN (recurrent neural network), mais sur les features ils se sont limités à word2vec, POS tag. Leur contribution est dans le fait qu’ils ont cherché à inclure des infos sémantiques et syntaxiques. Syntaxiques avec syntactic dependency parsing, qui met un POS tag à ses dépendances (voir article pour détail).

**UPV Inaoe :**

Tout est basé sur une technique de distorsion de textes. Leur idée est de mettre en évidence avec moins de fréquences et de réduire les mots avec beaucoup de fréquences. Ils ne font pas de sélection de feature à proprement parler. Ils ont ensuite utilisé des KNN ou des SVM pour effectuer la classification.

**BigIR :**

Ils n’ont pas fait de classification, mais du Learning-To-Rank (L2R) (<https://opensourceconnections.com/blog/2017/02/24/what-is-learning-to-rank/>). Mais pour les features, pas de changement particulier par rapport aux cas précédents.