Prédiction structurée pour le traitement automatique de la langue

Baptiste Chachura (Leader), Bouger Lisa (Follower)

January 24, 2025

Contents

1 Etiqueteur de super-senses avec transformers pré-entraînés

L'objectif de ce projet est de développer et d'évaluer un système capable de prédire des super-senses, des étiquettes sémantiques de haut niveau, pour des mots dans un corpus annoté (sequoia). Ces étiquettes indiquent la catégorie sémantique gros-grain d'un mot, comme Feeling pour "surprise" ou Body pour "jambe".

Pour cela nous allons utiliser un modèle pré-entraîné pour la prédiction des embeddings des mots des phrases(ici dans notre cas nous utiliserons une architecture BERT et plus précisément les modèles camembert et google-bert), suivis d'un classifieur.

2 Perceptron Multi Couches

Dans un premier temps nous allons utiliser un perceptron multi-couches afin de classifier nos embeddings. L'architecture du modèle est choisie de manière arbitraire et se compose d'une couche d'entrée de la dimension des embeddinds du transformers (ici 768), suivis de deux couches de dimension 200 puis une sortie vers une couche de dimension 25 ce qui correspond à notre nombre de super sense + 1 pour les mots sans classe (annotés par "*"). Les différentes couches sont séparées par des Relu.

Nous observons que le tokenizer associé au modèle Bert nous donne parfois plusieurs tokens pour un mot ce qui est problématique car dans notre cas nous souhaitons avoir un seul et unique embedding par mot afin de pouvoir prédire son étiquette. Pour palier à cela nous définissons un dataloader qui à partir des phrases du jeu de données, calcule les embeddings pour chaque token et retourne des embeddings sur les mots en effectuant la moyenne des embeddings des token qui constitue le mot.

2.1 Camembert VS Google-bert

Les déclinaisons de l'architecture BERT sont nombreuses et leurs performances pas toujours égales ce qui peut avoir une influence non négligeable sur les performances de notre classifieur. Pour étudier cela nous allons comparer deux modèles pour la génération de nos embeddings: Camembert [2] et Google-bert [1],

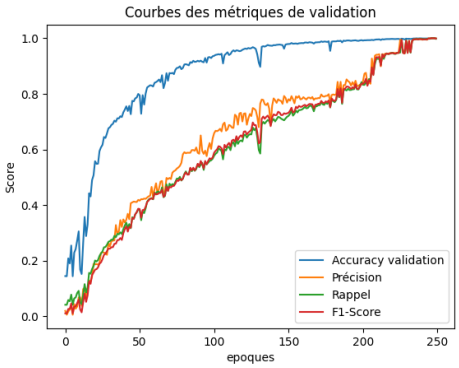


Figure 1: Apprentissage (embeddings : Camembert)

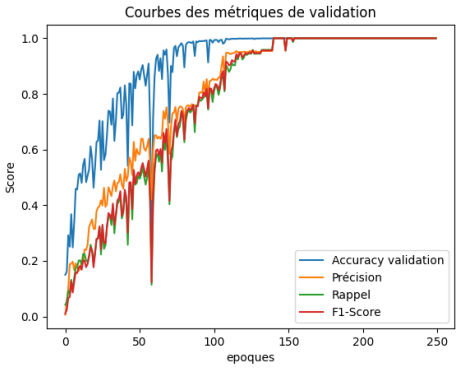


Figure 2: Apprentissage (embeddings : Google Bert)

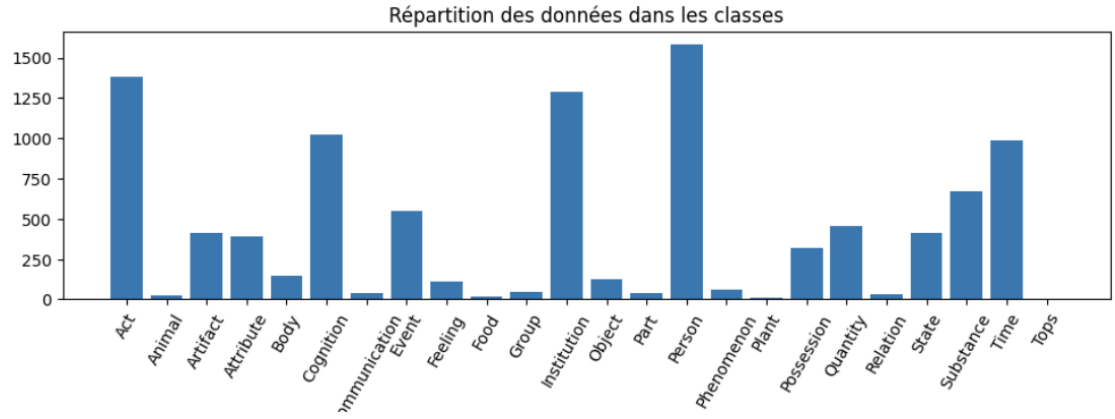
Nous constatons que la courbe d'évaluation du mlp sur les embeddings de google bert bien qu'augmentant plus rapidement, est beaucoup moins régulière que la courbe d'apprentissage du mlp sur les embeddings Camembert. Cela provient probablement du fait que le modèle base de Google est un modèle entraîné pour du multi-lingue ce qui n'est pas le cas du modèle Camembert spéfiquement entraîné sur des données françaises. C'est différences de performances lors de l'apprentissage suggèrent que le modèle français génère des embeddings plus réguliers et traduisant mieux le context, que le modèle de

google qui lui nécessite un mlp ajusté plus finement afin de lisser ces imprécisions sur les embeddings.

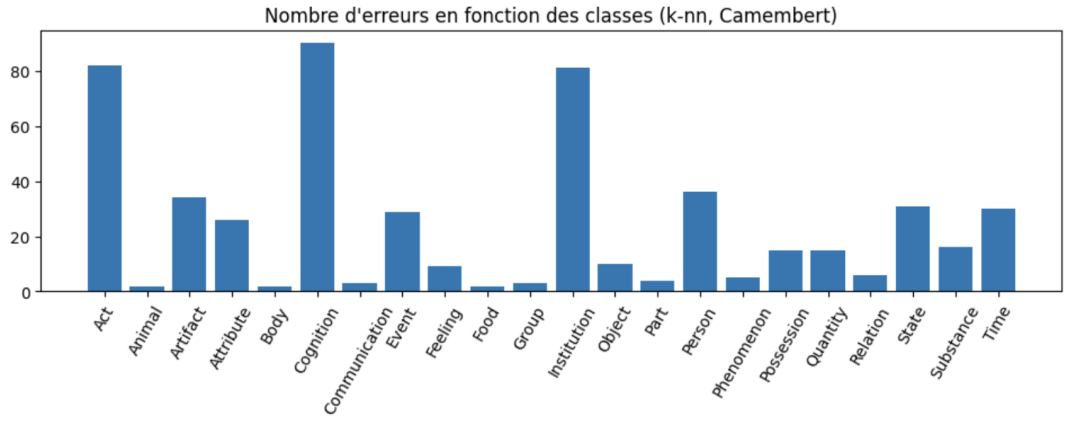
3 K plus proches voisins

Dans cette partie nous nous penchons sur une approximation de notre modèle précédent, en effet les modèles Bert sont connu pour la pertinence des embeddings qu'ils produisent dans un context donné. Comme ces embeddings agglomèrent l'information sur le mot et son context il se pourrait qu'ils suffisent à eux seuls à classifier nos mots. Pour cela nous utilisons l'algorithme des k plus proches voisins qui vise à attribuer la classe majoritaire parmis les voisins du mots que l'on cherche à classifier. Une des premières étapes a été de trouver le nombre

3.1 Répartition des données et impact



Une analyse de nos données d'entraînement nous donne le graphique ci-dessus, nous avons masqué la classe '*' étant donné que nous n'évaluons pas les performances sur la prédiction cette classe étant donné qu'elle est grandement majoritaire (80% des exemples d'entraînement) ce qui fausserait la prédiction. Malgré cela nous constatons tout de même de forte disparités au sein des classes. Nous savons que cela influence négativement les performances des différents classifieurs, un dataset plus équilibré serait donc souhaitable si l'on souhaite améliorer nos performances.



Nous remarquons un avantage sur le k-nn : la répartition des classes ne semble pas avoir un fort impact sur les prédictions du modèles contrairement à ce qui est connu pour les réseaux profond (qui commettent beaucoup d'erreurs sur les classes très minoritaires).

4 Comparaison des performances en fonction de la quantité de données d'apprentissage

Dans cette partie nous allons nous intéresser à l'influence de la quantité de données lors de l'apprentissage de nos modèles sur leurs performances. En effet les tâches de labelisation de données pouvant être complexes et coûteuses il est intéressant d'étudier la quantité de données nécessaire à un certain type de modèle pour atteindre un certain niveau de performances.

4.1 Comparaison

Pour cela nous partons toujours du dataset d'entraînement et nous créons des sous échantillons de proportions différentes (entre 20% et 100% du dataset). Nous réalisons 5 expériences pour chacune de ces proportions et comparons les résultats en généralisation. Pour des raisons de temps de calcul, les entraînement seront plafonnés à 20 epochs et non 250 comme dans la partie précédente.

4.1.1 Perceptron Multi Couches

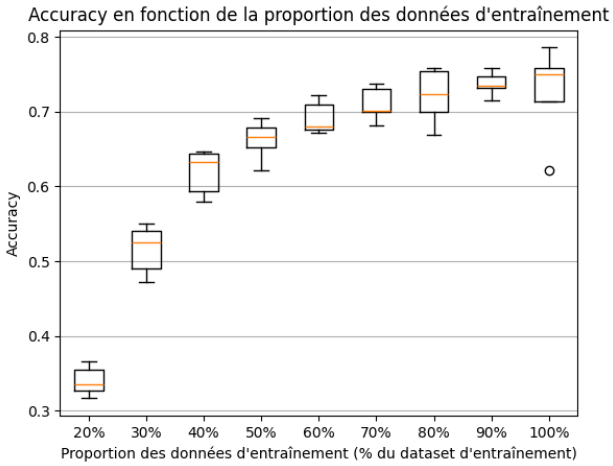


Figure 3: Apprentissage (embeddings : Camembert)

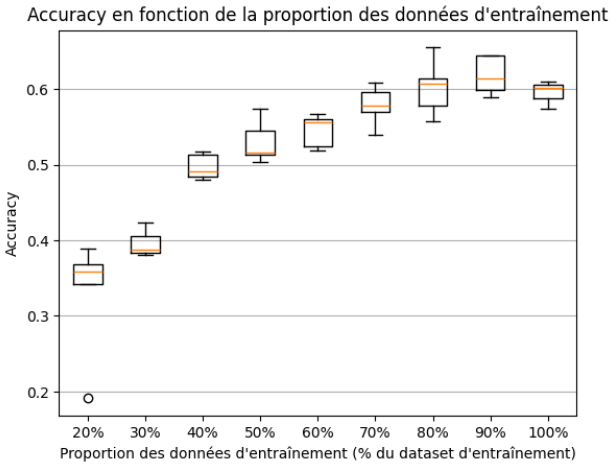


Figure 4: Apprentissage (embeddings : Google Bert)

La figure ci-dessus représentant les diagrammes en boîtes de nos expériences révèlent quelques points importants. Comme nous pouvions nous y attendre la performance augmente avec le nombre de données utilisées lors de l'entraînement, pour 20% des données les résultats oscillent entre 19% d'accuracy (pour un outlier) et 40%. Nous observons une augmentation jusqu'à 75% en moyenne pour un entraînement sur toutes les données disponibles pour les embeddings Camembert et 60% pour les embeddings google. Nous remarquons aussi une courbe moins régulière pour le modèle de google probablement pour les mêmes raisons que mentionnées dans la partie 1.1.1.

Comme vu dans les figures 1 et 2, augmenter le nombre de d'itérations lors de l'entraînement augmenterait l'accuracy de nos modèles, cependant comme nous n'utilisons que peu de données lors de l'entraînement le risque d'overfitting serait augmenté.

4.1.2 K plus proches voisins

A la suite de diverses expériences pour déterminer le k optimal (cf annexes), nous avons observé que prendre en compte un nombre élevé de voisins dans la prédiction conduisait à de moins bonnes performances. Dans la suite de cette partie nous utiliserons donc k=2. Dans le graph suivant : les échantillons de données sont tirés aléatoirement dans le jeu de données d'entraînement ce qui induit

de légères variations de performances.

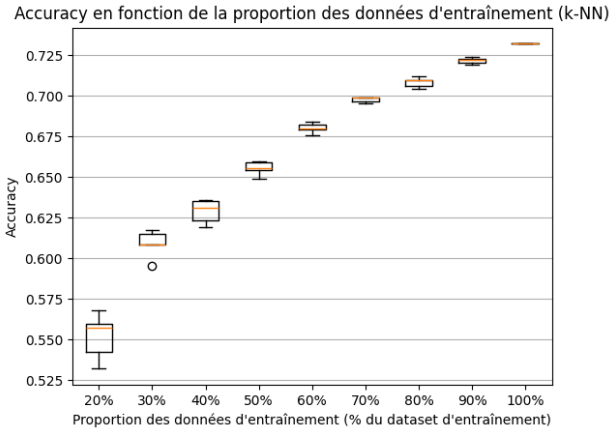


Figure 5: Apprentissage (embeddings : Camembert)

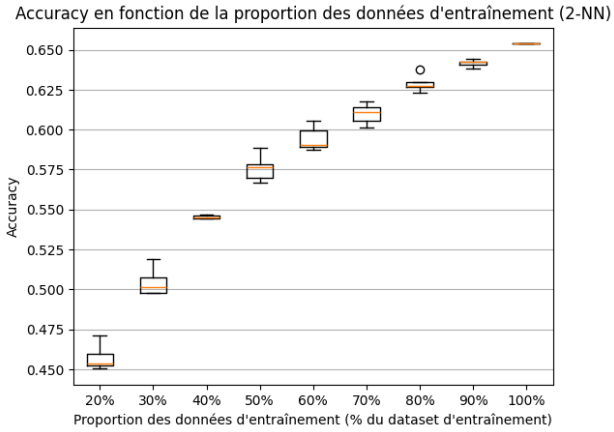


Figure 6: Apprentissage (embeddings : Google Bert)

Nous constatons que ce modèle est bien plus performant pour de très faible quantités de données d'entraînement, et les performances sur l'ensemble du dataset se rapprochent très fortement de celles obtenues avec le perceptron multi-couche (entraîné en 20 epochs). Les résultats montrent aussi que l'apprentissage d'un k-nn est plus régulier sur le mlp sur les embeddings générés par google bert.

5 Conclusion

Toutes ces expériences mettent divers points intéressant en lumière : - le mlp se montrent bien plus performant que le k-nn si on l'entraîne avec de bons hyper-paramètres et que nous avons une quantité de données suffisante et si possible présentant une répartition des classes équilibrée. - le k-nn performe de manière acceptable lorsque l'on a très peu de données et semble moins sensible à la répartition des classes Ainsi nous avons montré que dans la tâche de prédiction de super-senses les deux approches sont possibles et fournissent des résultats aux avantages variés. Nous pourrons aussi remarquer que le k-nn à l'avantage d'être un algorithme plus facile à interpréter qu'un mlp néanmoins il se montre moins rapide lors de l'inférence lorsque la quantité de vecteurs est importante.

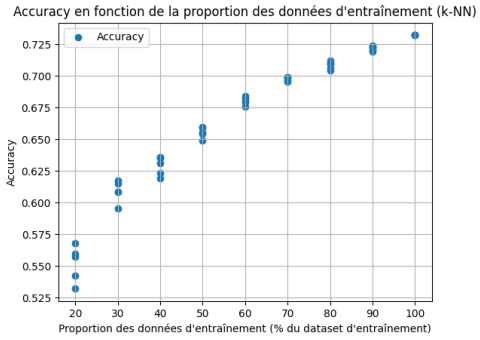
Bibliographie

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

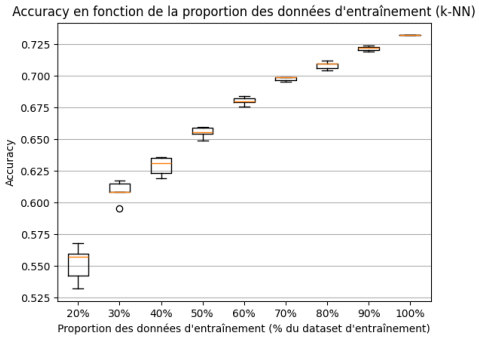
[2] Louis Martin, Benjamin Muller, Pedro Javier Ortiz Suárez, Yoann Dupont, Laurent Romary, Éric Villemonte de la Clergerie, Djamé Seddah, and Benoît Sagot. Camembert: a tasty french language model. *CoRR*, abs/1911.03894, 2019.

6 Annexes

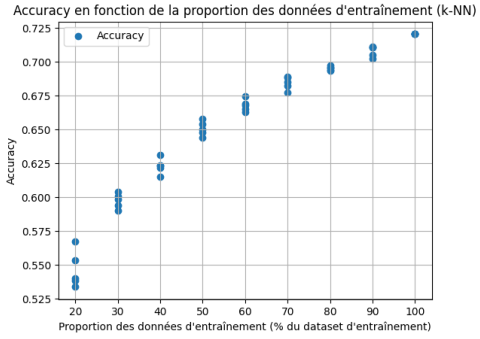
6.1 K-NN, embeddings Camembert



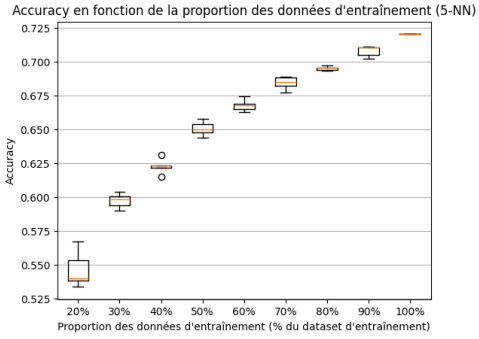
(a) Résultat brut k=2



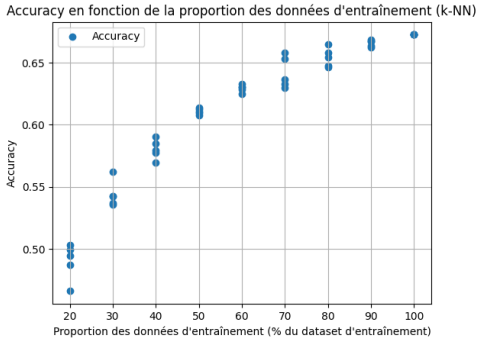
(b) box plot k=2



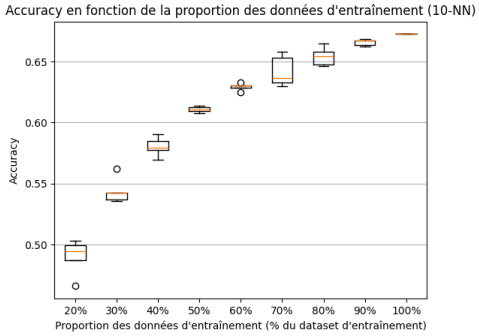
(c) Résultat brut k=5



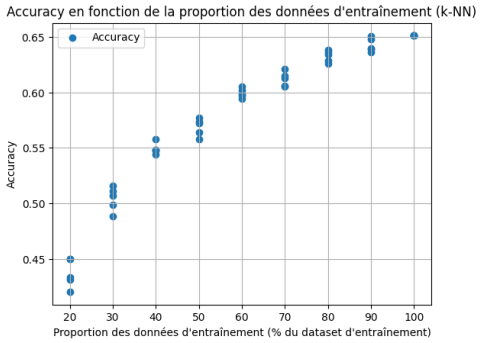
(d) box plot k=5



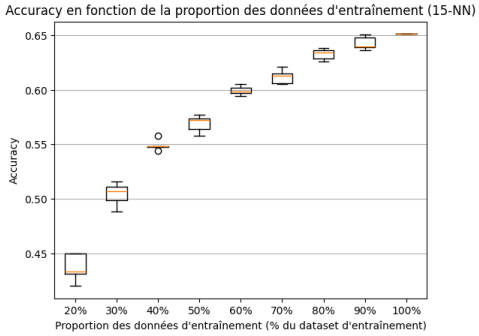
(e) Résultat brut k=10



(f) box plot k=10



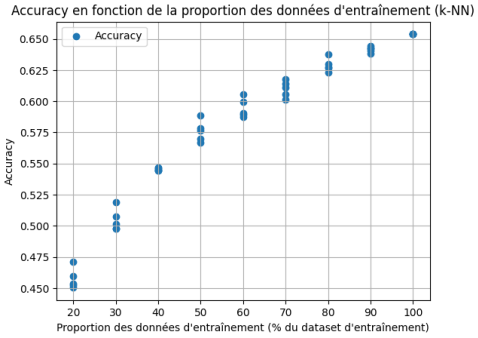
(g) Résultat brut k=15



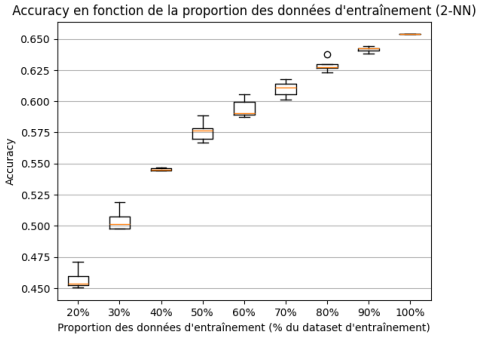
(h) box plot k=15

Figure 7: Expériences pour déterminer le nombre de voisins à utiliser

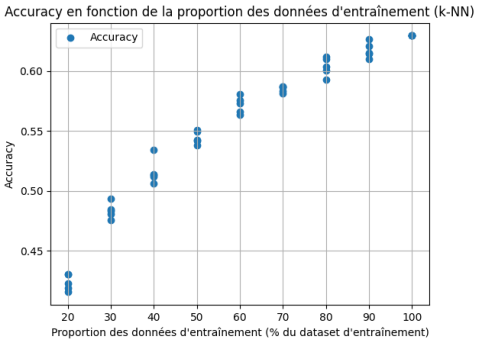
6.2 K-NN, embeddings Google Bert



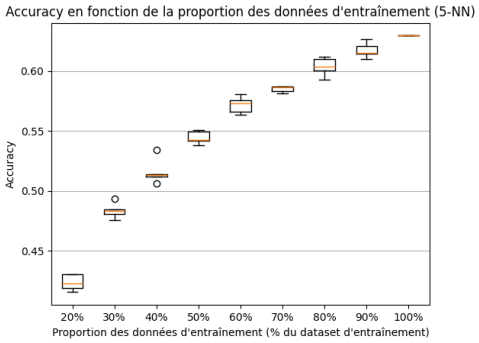
(a) Résultat brut k=2



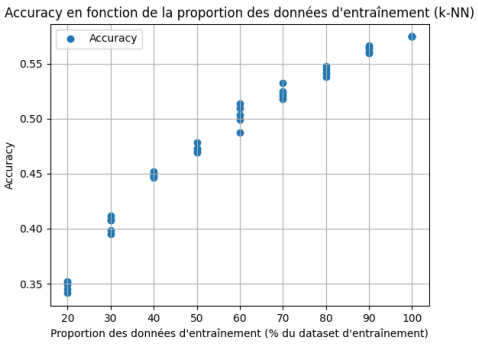
(b) box plot k=2



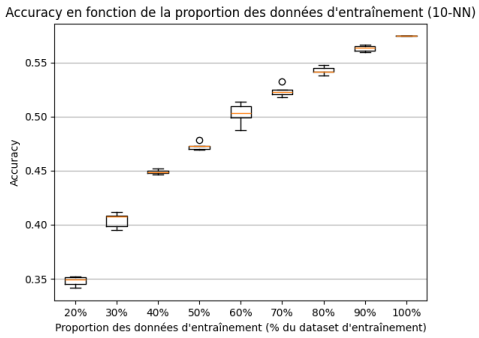
(c) Résultat brut k=5



(d) box plot k=5



(e) Résultat brut k=10



(f) box plot k=10

Figure 8: Expériences pour déterminer le nombre de voisins à utiliser