# Comparaison de deux courbes polygonales dans l'espace des embeddings FastText pour l'étude de la relation de traduction entre deux phrases

Adel Nabli

## Abstract

Pour étudier la relation de traduction entre 2 phrases en anglais et français, nous avons choisit d'adopter un point de vue géométrique. En prenant des *"word embeddings"* français anglais alignés, les mots de chaque langue vivent dans le même espace vectoriel et les mots proches sémantiquement dans les deux langues sont aussi proches géométriquement dans cet espace. Ainsi, nous pouvons voir une phrase comme une séquence de points dans cet espace, ou encore, une phrase comme une courbe polygonal dans l'espace des embeddings. Ainsi, à l'aide d'outils issus de la géométrie sur le sujet des distances entre courbes dans des espaces euclidiens, nous avons étudié la similitude des courbes générées par une phrase en français et une phrase en anglais, réussissant à gagner une intuition sur leur relation de traduction. L'avantage de cette méthode est que les quelques features calculées (les 6 différentes mesures de similitudes que nous avons utilisées) donnent beaucoup d'information sur la tâche considérée, à tel point que nous avons seulement besoin d'**une vingtaine d'exemples** pour entraîner un classificateur qui obtient une *accuracy* de 97%. Notre méthode plafonne par contre rapidement, et passé le cap des 97-98% d'*accuracy*, ajouter plus de données n'améliore pas les résultats.

## 1 Motivations:

La majorité des techniques d'apprentissage actuelles prennant en entrée des vecteurs, vectoriser une séquence de caractères est une tâche cruciale en NLP. Aujourd'hui, la représentation de mots en vecteur est, depuis l'arrivée de *word2vec*, une tâche qui semble gagner en maturation. Beaucoup d'efforts sont par contre encore à faire pour arriver à des résultats similaires à l'échelle de la phrase ou du document. Allant à contre-courant de cette tendance à vouloir vectoriser des séquences de mots, nous avons voulu tirer partie de la qualité des *embeddings* de mots que l'on sait produire actuellement. Ainsi, au lieu de représenter une phrase comme un seul vecteur, nous nous sommes demandé si la voir comme une suite de points dans l'espace vectoriel des *embeddings* serait pertinent pour la tâche fixée par l'exercice. Dans notre approche, nous considérons ainsi qu'une phrase est une courbe polygonale *(obtenue en reliant par un segment deux mots/vecteurs consécutifs)* orientée *(on conserve l'ordre des mots de la phrase)* dans l'espace des embeddings choisis. Nous ne tentons donc pas d'encapsuler le sens de toute une phrase dans un vecteur de dimension fixe, mais pensons plutôt que le sens est encodé dans le "chemin" tracé dans l'espace des embeddings par la suite de mots constitutive d'une phrase.
Ensuite, nous nous sommes demandé si 2 phrases sémantiquement proches ne dessineraient pas 2 courbes semblables dans cet espace, et si nous pouvions utiliser cela pour repérer les traductions.
Cependant, afin de pouvoir comparer directement ces 2 courbes, il faut encore qu'elles "vivent" dans le même espace. En effet, produire des *embeddings* de mots dans **une langue** donnée est une tâche que l'on sait faire, mais notre exercice faisant intervenir 2 langues différentes, nous avons besoin en plus que les 2 espaces soient alignés: pour que notre comparaison ait un sens, il faut que passer du point *"the"* au point *"car"* dans l'espace des embeddings anglais dessine un segment proche de celui créé en passant de *"la"* à *"voiture"* dans l'espace français.
Un autre problème immédiat que soulève notre méthode est celui de la métrique à utiliser: *Comment peut-on comparer deux courbes? Quelles notions de distance existe-t-il dans l'espace des courbes ?*

## 2 Méthode utilisée:

### 2.1 Embeddings:

Afin de pouvoir directement comparer nos courbes "françaises" et "anglaises", nous devons avoir à disposition un jeu d'embeddings alignés. Justement,

Facebook a mis à disposition pour 44 langues *(dont le français et l'anglais)* des embeddings FastText pré-entraînés sur Wikipédia et alignés suivant une méthode présentée par Joulin et al., 2018 [1][2]. Nous avons donc utilisés ces embeddings de dimension 300 pour notre travail.

### 2.2 Pré-processing des données:

Il nous faut pouvoir transformer les ensembles de phrases en suites de points dans $\mathbb{R}^{300}$ pour continuer. Pour cela, nous avons maintenant à disposition deux ensembles de phrases francais/anglais et deux dictionnaires d'embeddings alignés.
On aimerait donc pouvoir trouver, pour chaque mot de chaque phrase, une correspondance dans le dictionnaire d'embeddings de la langue correspondante. Pour se faire, il faut effectuer quelques traitements à nos données:

- Tokenizer nos phrases en enlevant les tirets *("celle-ci" n'est par exemple pas dans le dictionnaire fourni par FastText)*
- Mettre toutes les lettres en minuscule *(car les mots des dictionnaires FastText ne contiennent pas de majuscules.)*

En plus de cela, nous avons décidé d'enlever les tokens "." et "\n" présents en fin de chaque phrase car nous croyons que cela ajoute des similarités artificielles entre les phrases.
Avec ce pré-processing, nous créons un vocabulaire de taille 96929 pour le corpus français et de taille 68215 pour le corpus anglais *("corpus" désignant les phrases du train et test set)*. Cependant, si l'on essaie de mettre en relation les tokens de ces corpus avec ceux des dictionnaires FastText, on n'arrive pas à assigner d'embeddings à 17132 mots de notre vocabulaire français, et 8367 mots du vocabulaire anglais. Après inspection, on trouve que ce qui coince sont en majorité les valeurs numériques *(il n'y a pas d'embeddings pour les nombres)* et des versions "complexifiées" de certains mots *(conjugaisons exotiques)*. Nous avons donc décidé de lemmatiser tous les mots auxquels on n'arrivait pas à associer de vecteur grâce à `spacy` et d'attribuer au mot le vecteur de son lemme si on en trouvait un (*au vu de la tâche considérée, nous faisons l'hypotèse que la perte d'information qui résulte de cette simplification n'impacte que peu nos résultats*). Par cette méthode, nous avons réussi à réduire le nombre de mots sans embeddings à 2208 pour le français et 154 pour l'anglais.
A l'issue de ce processus, tous les mots sans *embeddings* seront ignorés par la suite. Cela implique notamment d'ignorer toutes les valeures numériques présentes dans les phrases, et ce malgré le fait que l'on puisse s'attendre à ce qu'elles puissent être de bons indicateurs à regarder pour la tâche à faire *(deux phrases traduction l'une de l'autre contenant vraisemblablement les même valeurs numériques)*.

### 2.3 Distances entre 2 courbes d'un espace métrique:

Comparer deux courbes dans un espace métrique est un problème classique en *computational geometry*. C'est cependant encore un domaine de recherche actif et plusieurs notions de distances existent. Nous avons pour notre part implémenté en `python` une méthode approximant la **distance de Fréchet**, utilisé le package `scipy.spatial` pour le calcul des **distance de Hausdorff** et celle donnée par **analyse procustéenne**. Nous avons également calculé le *cosine* entre les moyennes des vecteurs constitutifs des 2 phrases. Enfin, nous avons **créé une mesure** qui nous semblait appropriée, et calculé un **score s'inspirant de Stacc**.

#### 2.3.1 Discrete Fréchet distance:

Pour comprendre ce que mesure la distance de Fréchet, une analogie est souvent utilisée: celle du maître promenant son chien. Si l'on regarde la trajectoire au sol du parcours du maître lors de la promenade, ainsi que celle de son chien, on a 2 courbes. Le maître et le chien n'ont fait qu'avancer pendant cette promenade *(les retours en arrières ne sont pas permis)*, mais potentiellement à des vitesses différentes *(le chien a pu accélérer parfois, le maître se stopper etc...)*. Ce que va calculer la distance de Fréchet est: étant donné le parcours du maître et du chien (*les deux courbes)*, sachant qu'on n'autorise pas les retours en arrière (*courbes orientées*), mais qu'on autorise le maître et le chien à avancer à des vitesses différentes, quelle est la longueur minimale de la laisse inextensible que le maître peut utiliser pour promener son chien ?
Le problème de calculer cette distance est non trivial en général. Depuis Alt et Godau, 1992 [3], on sait résoudre exactement ce problème dans le cas des

courbes polygonales en $O(pq \log pq)$ où $p$ et $q$ désignent les nombres de segments constitutifs des deux courbes. Cependant, l'algorithme proposé présentant une certaine difficulté à l'implémentation, nous avons décidé de nous rabattre sur un algorithme plus simple et très utilisé permettant de calculer une approximation de cette distance: la *discrete Fréchet distance*. L'algorithme en question, présenté par Eiter et Mannila, 1994 [4] permet de calculer cette distance en $O(pq)$. (*voir notre Notebook pour le pseudo-code, des exemples et plus de détails*).

#### 2.3.2 Distance d'Hausdorff:

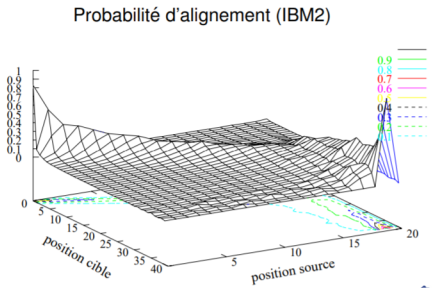
Une autre mesure courante pour la comparaison entre 2 courbes est la distance de Hausdorff. Pour comprendre ce qui est calculé, prenons la courbe *française*. Pour chaque sommet de cette courbe polygonale, on va trouver le sommet de la courbe anglaise qui en est le plus proche au sens d'une distance euclidienne et on stocke cette distance. La "*directed Hausdorff distance*" est alors le maximum de toutes les distances trouvées en parcourant tous les sommets de la courbe française. Cette distance n'est pas symétrique, en parcourant la courbe anglaise au lieu de la française, le résultat aurait pu être différent. Ainsi, la *distance de Hausdorff* se trouve en prenant le maximum des distances trouvées en parcourant la courbe française, puis la courbe anglaise. On remarque qu'en utilisant cette distance, on ne prend pas en compte "l'ordre des mots" dans la phrase comme c'était le cas avec la distance de Fréchet (la phrase est réduite à un ensemble de points).
Taha et Hanbury, 2015 [5] ont proposé un algorithme pour calculer cette distance en $O(pq)$ dans le pire cas. Cet algorithme a été implémenté dans `scipy.spatial`, c'est ce que nous avons utilisé.

#### 2.3.3 Analyse Procustéenne:

L'analyse procustéenne est utilisée pour comparer 2 formes (ayant le même nombre de sommets). Etant donnée 2 formes (dans notre cas, 2 courbes polygonales), on cherche d'abord à faire ressembler nos deux formes le plus possible en appliquant à la courbe française des transformations linéaires (rotations, reflections, dilatations), et on mesure ensuite la somme des distances sommets à sommets entre la courbe française transformée et la courbe anglaise. Cette sommation se fait sur les sommets de même indice (on ne prend que la distance du $sommet_i$ français avec le $sommet_i$ anglais). Ainsi, on a besoin pour cela que nos phrases contiennent le même nombre de mots. Pour cela, nous avons ajouter des vecteurs de 0 à la phrase la plus petite. Nous avons de nouveau utilisé le package `scipy.spatial` pour ce calcul.

#### 2.3.4 Mesure "maison":

Nous avons créé notre propre mesure de similarité pour comparer les deux courbes polygonales. Cette mesure s'inspire du fait que nous avons vu en cours que les alignements entre 2 phrases en relation de traduction pour le français et l'anglais mettent en relation des mots souvent "*aux même endroits dans leurs phrases respectives*".




Ainsi, en prenant $f = f_1...f_F$ et $e = e_1....e_E$ deux phrases française et anglaise, si elles sont en relation de traduction, on peut s'attendre à ce qu'en regardant dans l'entourage du vecteur $f_t$, on trouve un vecteur dont le *cosine* est faible avec le vecteur $e_t$. On construit alors la mesure suivante:

$$d(e,f) = \frac{1}{2\min(E,F)}\Big(\sum_{t\in[\![1,E]\!]} \min_{k\in entourage(t')} \big(d_{cosine}(f_k, e_t)\big) + \sum_{t\in[\![1,F]\!]} \min_{k\in entourage(t'')} \big(d_{cosine}(e_k, f_t)\big)\Big)$$

Avec: $t' = \left\lfloor \frac{t}{E}F \right\rfloor$, $t'' = \left\lfloor \frac{t}{F}E \right\rfloor$ et $entourage(t)$ étant l'ensemble des mots qui sont dans la phrase entre les positions $t - n_{entourage}$ et $t + n_{entourage}$ ($n_{entourage}$ est un paramètre donné par l'utilisateur). La normalisation par $\min(E,F)$ étant là pour pénaliser un trop grand écart de longueur entre les deux phrases.

#### 2.3.5 Mesure inspirée de Stacc:

En cours, nous avons vu que la mesure Stacc donnait de bons résultats. Nous nous sommes donc demandé si on pouvait en construire une variante

géométrique. Ainsi, au lieu de chercher les $n$ premières traductions de chaque mots de la phrase (*mettons française*) et d'agréger cela dans un ensemble pour regarder l'intersection avec les mots de la phrase anglaise, nous avons plutôt considéré, pour la distance *cosine*, des boules $\mathcal{B}$ de rayon $r$ centrées sur les mots de la phrase française et regardé leurs intersections avec les points (*les mots*) de la phrase anglaise (*puis inversement*). Cela donne la mesure suivante:

$$d(e,f) = \frac{1}{2E \times F}\Big( \sum_{t\in[\![1,E]\!]} |\mathcal{B}_{cosine}(e_t, r) \cap \{f_{t'}\}_{t'\in[\![1,F]\!]}| + \sum_{t'\in[\![1,F]\!]} |\mathcal{B}_{cosine}(f_{t'}, r) \cap \{e_t\}_{t\in[\![1,E]\!]}|\Big)$$

#### 2.3.6 Cosine des moyennes:

Comme présenté comme baseline dans le sujet, nous avons également ajouté dans notre liste de $features$ la mesure du *cosine* entre la moyenne des points de la phrase française et anglaise.

### 2.4 Classificateur:

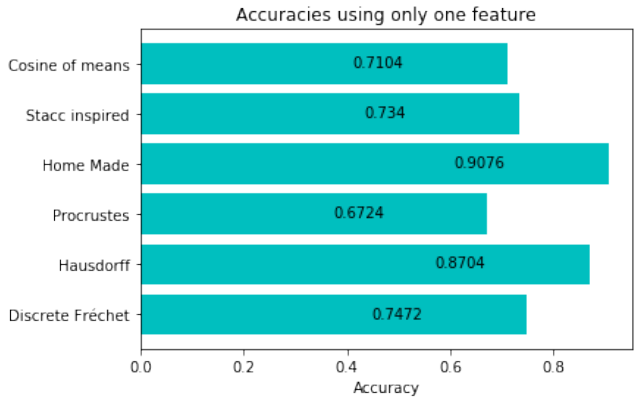
Une fois que nous avons, pour chaque paires de phrases du corpus, calculé les 6 distances présentées, il nous reste à entraîner un classificateur sur ces 6 $features$ pour discriminer les phrases qui ne sont pas en relation de traduction de celles qui le sont. Pour cela, nous avons simplement utilisé un RandomForest.

## 3 Résultats:

Sur notre laptop, en 1 seconde nous calculons les 6 $features$ pour seulement 9 paires de phrases, ce qui n'est pas immédiat. Nous avons donc extrait un jeu de donnée de 10 000 phrases du corpus présenté (5000 phrases en relation de traduction, et 5000 qui ne le sont pas). Nous avons mis 20 min à calculer nos $features$ sur ce dataset.
Nous utiliserons l'*accuracy* comme métrique principale pour évaluer notre modèle.

### 3.1 Utilisation d'une seule feature à la fois:

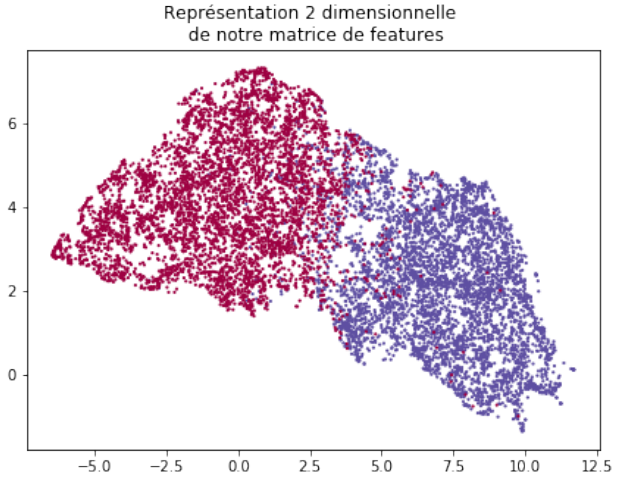
Nous nous sommes demandé quel serait notre résultat si nous utilisions qu'une seule des $features$ calculé pour la classification. Voilà ce que l'on obtient en utilisant 5000 données de train et 5000 de test:




Ainsi, on remarque que sur le *test set*, c'est notre "mesure-maison" qui performe le mieux avec une *accuracy* de plus de 90%, suivit de la distance de Hausdorff. La question est maintenant de savoir comment notre classificateur performerait en utilisant toutes les $features$ à disposition.

### 3.2 Utilisation de toutes les features:

En voyant les résultats en utilisant qu'une seule $feature$ à la fois, on suppose que les distances que nous avons calculé arrivent très bien à séparer notre *dataset* en deux régions. Nous voulions vérifier cela, et avons donc utiliser un algorithme de réduction de dimension (*UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction*, package du `scikit-tda` toolkit) pour réduire nos vecteurs de dimension 6 en vecteurs de dimension 2 et pouvoir représenter le nuage de points dans le plan.



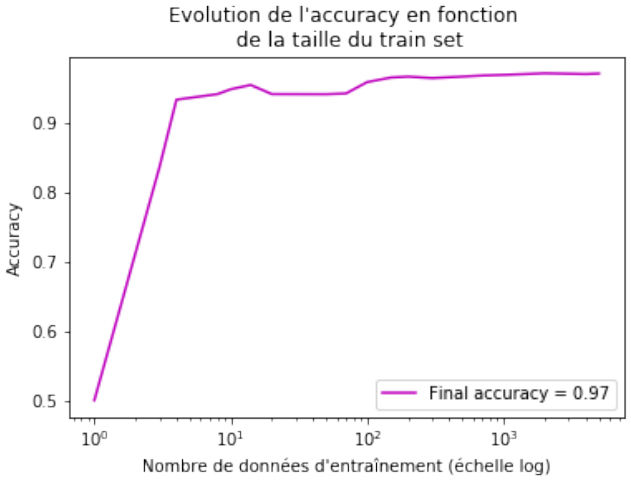
Il semblerait donc bien que nos 6 features arrivent à séparer proprement les phrases en relation de traductions de celles qui ne le sont pas. Cela se vérifie en utilisant notre classificateur, on obtient une ***accuracy* de 97.5%** sur le test set.

| | Relative Importance of the features |
|---|---|
| Home Made | 43.27 % |
| Hausdorff | 29.37% |
| Discrete Fréchet | 11.4% |
| Stacc inspired | 6.53 % |
| Cosine of means | 5.85 % |
| Procrustes | 3.56 % |

En demandant l'importance de chaque $feature$ dans la construction du classificateur au classificateur entraîné, les résultats sont en accord avec ceux obtenus en utilisant qu'une seule $feature$: la distance la plus discriminante est la plus utilisée, la moins bonne la moins utilisée.

### 3.3 De l'inutilité d'un gros train set:

Nous avons utilisé jusqu'ici utilisé 5000 phrases pour entraîner notre classificateur. Cependant, vu que le classificateur agit sur un espace de relativement faible dimension (*de dimension 6*) et qu'il semblerait que nos $features$ arrivent très bien à séparer notre dataset en deux régions, on s'est posé la question du nombre minimal de données qu'il nous faudrait pour construire un classificateur "*décent*".




Ainsi, on se rend compte qu'**avec notre méthode, il nous suffit de 10 exemples pour obtenir une accuracy dépassant les 90%**. On remarque également qu'une fois la zone des 97% atteinte (avec aux alentours de 100 exemples), notre accuracy n'augmente plus en ajoutant de nouvelles données.

## 4 Discussions:

On peut essayer de trouver des solutions pour "percer" ce plafond de plusieurs manières:

- Ré-introduire et utiliser l'information donnée par les valeurs numériques présentes dans les phrases (Nous avons totalement ignoré tous les chiffres/acronymes/noms qui n'étaient pas dans le dictionnaire d'embeddings FastText)
- Raffiner nos distances (par exemple, intuitivement, on se dit que dans l'analyse procustéenne, au lieu de combler par des vecteurs de 0 la phrase la plus courte, il serait plus sensé de la combler par sa valeur moyenne)

En inspectant les phrases "mal catégorisées" par notre classificateur, on s'est en effet rendu compte que certaines phrases auraient tout à fait pu être bien classifiée si les valeurs numériques présentent dans les deux phrases avaient été consultées.

On s'est aussi rendu compte que notre classificateur avait tendance à dire que deux phrases étaient en relation de traduction (même si c'est faux) si elles étaient toutes deux très longues. Nous pensons que cela vient d'un biais d'apprentissage (La $feature$ avec le plus de poids étant notre mesure-maison, cette-dernière pénalisant une trop grande dissymétrie de longueur, et le "mélange" des phrases qui ne sont pas en relation de traduction ayant été fait de manière aléatoire, notre classificateur a donc surement appris que 2 phrases longues étaient surement en relation de traduction)

Enfin, notre méthode étant entièrement géométrique et aucunement statistique, nous n'arrivons pas à apprendre le parallélisme entre certaines expressions idiomatiques (avec notre méthode, nous n'arriverons jamais à déceler la relation de traduction entre *"nous avons du pain sur la planche"* et *"a lot has to be done"* ). Pour améliorer cela, nous pourrions introduire des features statistiques.

## References

[1] Joulin, Armand and Bojanowski, Piotr and Mikolov, Tomas and Jégou, Hervé and Grave, Edouard *Loss in Translation: Learning Bilingual Word Mapping with a Retrieval Criterion*. Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, 2018.

[2] Bojanowski, Piotr and Grave, Edouard and Joulin, Armand and Mikolov, Tomas *Enriching Word Vectors with Subword Information* Transactions of the Association for Computational Linguistics, vol.5, 2017

[3] H. Alt and M. Godau *Measuring the resemblance of polygonal curves* In Proc. 8th Annu.ACM Sympos. Comput. Geom., pages 102109, 1992

[4] Thomas Eiter and Heikki Mannila *Computing Discrete Fréchet Distance*

[5] Abdel Aziz Taha and Allan Hanbury *An Efficient Algorithm for Calculating the Exact Hausdorff Distance* ieee transactions on pattern analylis and machine intelligence, vol. 37, no.11, november 2015