Analyse statistique et empirique des modèles de Word Embedding sur Twitter

Kim Antunez, Romain Lesauvage, Alain Quartier-la-Tente
sous l'encadrement de Benjamin Muller (Inria)

1 Évaluation du modèle implémenté

1.1 Comment évaluer évaluer le modèle ?

1.1.1 Similarité cosinus

Afin de pouvoir évaluer le modèle, nous devons fixer un certain nombre critères sur lesquels s'appuyer. L'un des enjeux principaux est de pouvoir estimer la proximité entre deux mots. En effet, le modèle doit capter grâce dans les vecteurs ces notions de proximité entre mots. Il faut donc se fixer une distance afin de mesurer ce phénomène.

Il existe différents types de distances que nous pouvons classiquement utiliser pour mesurer la proximité entre deux vecteurs, parmi elles :

- la distance euclidienne $d(\vec{u}, \vec{v}) = \|\vec{u} - \vec{v}\|_2$;
- la distance de Manhattan $d(\vec{u}, \vec{v}) = |x_{\vec{u}} - x_{\vec{v}}| + |y_{\vec{u}} - y_{\vec{v}}|$;
- la similarité cosinus $d(\vec{u}, \vec{v}) = \frac{\vec{u}.\vec{v}}{\|\vec{u}\|_2 \|\vec{v}\|_2}$.

Chacune de ces distances possèdent des propriétés intéressantes mais l'utilisation d'une d'entre elles en particulier dépend du problème sur lequel on est en train de travailler. Dans le cadre de *Word2Vec*, nous préférerons ici la similarité cosinus car elle s'intéresse plutôt à l'angle formé entre deux vecteurs, tandis que les distances euclidiennes et de Manhattan s'intéressent aux valeurs du vecteur directement, or l'angle est un paramètre plus robuste dans notre cas.

Nous pouvons donc interpréter ici la similarité cosinus comme une mesure de l'angle formé entre deux vecteurs représentant deux mots différents. Aussi, une similarité proche de +1 signifiera que les mots sont très proches, une valeur proche de -1 que les mots sont corrélés négativement et une valeur proche de 0 une quasi-indépendance.

1.1.2 Analyse en Composantes Principales

Une fois que le modèle *Word2Vec* est entraîné, nous obtenons des *word-embeddings* pour chacun de nos mots, représentés par des vecteurs de dimension pouvant être 20, 50 ou même 100. Dès lors, il est très compliqué d'avoir un outil de visualisation permettant de repérer les mots qui semblent proches. Pour parer à cela, nous pouvons alors utiliser une analyse en composantes principales. L'objectif premier de cette analyse est d'arriver à projeter un nuage de points sur un espace de dimension inférieure, tout en étant le plus proche de la réalité. D'un point de vue plus mathématique, on peut également

voir l'ACP comme l'approximation d'une matrice (n, p) par une matrice de même dimensions mais de rang q < p.

La construction des axes de l'ACP est définie de manière séquentielle : on commence par déterminer le premier axe factoriel, sur lequel le nuage de points se déforme le moins possible, puis on cherche alors un deuxième axe orthogonal au premier tel que le nuage de points se déforme le moins possible après projection. On réitère cela jusqu'à obtention du nombre d'axes souhaités.

A CONTINUER AVEC FORMULES ETC.

1.1.3 Algorithme t-distributed Stochastic Neighbor Embedding

Bien que l'ACP soit une première manière de résumer l'information contenue dans nos vecteurs, elle présente des limites, notamment dans les vecteurs aux trop grandes dimensions, pour lesquels l'inertie des premiers axes de l'ACP peut se révéler faible. Pour combler ces lacunes, un autre algorithme peut être utilisé, celui dit du t-distributed Stochastic Neighbor Embedding. Contraitement à l'ACP, cet algorithme est stochastique et favorise l'apparition de groupes de mots proches. L'idée reste cependant la même, c'est-à-dire pouvoir représenter dans un espace l'ensemble de notre nuage de points de manière à repérer les mots proches.

L'algorithme commence par transformer les distances euclidiennes entre nos vecteurs $(X_1, ..., X_n)$ en probabilités conditionnelles qui représenteront également les similarités entre nos vecteurs. Pour cela, on note :

$$p_{j|i} = \frac{\exp - \frac{\left\|\vec{X_i} - \vec{X_j}\right\|^2}{2\sigma_i^2}}{\sum_{k \neq i} \exp - \frac{\left\|\vec{X_i} - \vec{X_k}\right\|^2}{2\sigma_i^2}}$$

La similarité entre les points X_i et X_j est alors la probabilité conditionnelle que X_i choisirait X_j comme voisin si ces derniers étaient choisis selon une loi gaussienne centrée en X_i. Ici, σ_i est la variance de cette gaussienne centrée en X_i, calculer de manière itérative de sorte à maximiser la perplexité du modèle (mesure du modèle de probabilités).

Dans une seconde étape, il faut réaliser les mêmes calculs mais dans l'espace de projection, où nous aurons nos points $(Y_1, ..., Y_n)$. Cependant, puisqu'en dimension réduite, la gaussienne à tendance à concentrer les points, l'algorithme propose d'utilise une distribution de *Student*, d'où le nom de l'algorithme, et ainsi de calculer :

$$q_{j|i} = \frac{(1 + \left\|\vec{Y_i} - \vec{Y_j}\right\|^2)^{-1}}{\sum_{k \neq i} (1 + \left\|\vec{Y_i} - \vec{Y_k}\right\|^2)^{-1}}$$

Il ne reste plus désormais qu'à déterminer les vecteurs $(Y_1, ..., Y_n)$ qui minimise l'écart entre les deux distributions de probabilité. Pour cela, l'algorithme se base sur la divergence de Kullback–Leibler entre les distributions P et Q :

$$KL(P, Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

Avec

$$p_{ij} = \frac{p_{i|j} + p_{j|i}}{2n}$$

où n est le nombre de vecteurs.

Ensuite, on minimise grâce à une descente de gradient. On obtient alors les résultats de l'algorithme t-SNE, qu'il faut cependant analyser avec précaution : il ne s'agît pas d'un algorithme linéaire équivalent à l'ACP, on ne peut donc pas interpréter directement des éléments tels que la taille des clusters obtenus ou leur distance relative.