Damoun Ayman
Github repository
January 11, 2021

OPT12
Text Mining and Chatbots
Sahar Ghannay

Word embeddings training - Lab 1

Introduction

Word embedding is a learned representation of a word in which each word is represented using a vector in an n-dimensional space. In this approach, Words with the same meaning have the same representation. These representations make it possible to identify synonyms, antonyms and other relationships between words. The idea behind Word embedding can be generalized to develop embeddings for individual sentences, documents, and so on.

word2vec: word2vec is the deep learning Google framework to train word embeddings. This framework use all the words of the corpus to predict the neighboring words. The word2vec algorithm create a vector for all the words present in the data in a way that the context is captured. Word2Vec is an iterative method. Its main idea is as follows [5]:

- take all corpus;
- move through the text in a sliding window, moving one word at a time. At each step, there is a central word and context words ;
- for the central word, compute probabilities of context words;
- adjust the vectors to increase these probabilities.

word2vec is mainly compensated by:

- Skip-Gram
- Continuous Bag of Words (CBOW)

0.1 Skip-Gram

The skip-gram model [1] try to predict a context word when a target word is taken as input. According to the figure (1), input word wi and the output word wj are one-hot encoded into binary vectors x and y of size V where v is the vocabulary size. The embedding vector is obtained by multiplying the input vector x and the word embedding matrix W. To get the output hot encoded vector vector y, the hidden vector is multiplied by the word context matrix W'. The matrix W' encodes the meanings of words as context.

0.2 Continuous Bag-of-Words (CBOW)

Continuous Bag-of-Words model predict the target word using the context words as input. According to [1] the CBOW model is faster to learn, but the skip-gram model generally gives better results.

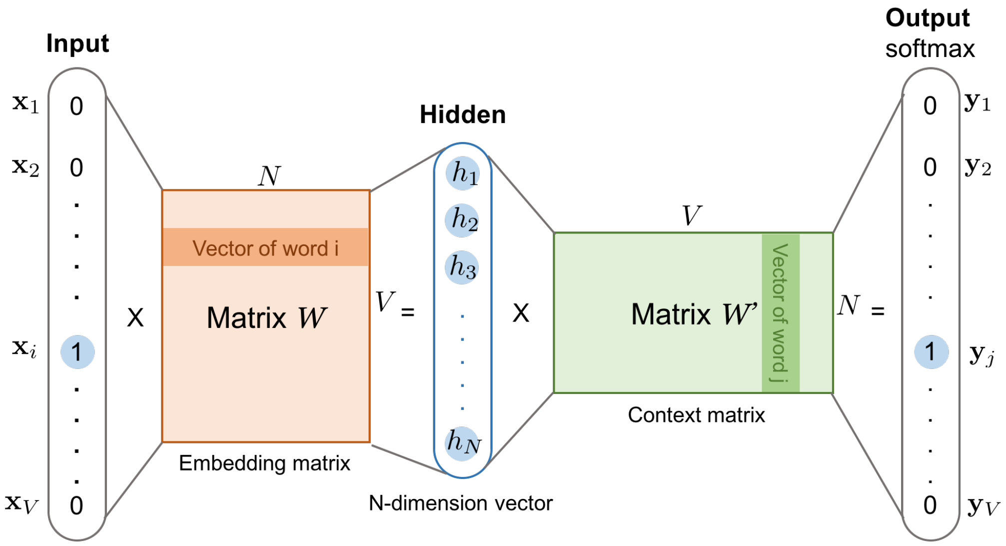

Figure 1: The skip-gram model. Both the input vector x and the output y are one-hot encoded word representations. The hidden layer is the word embedding of size N. [6]

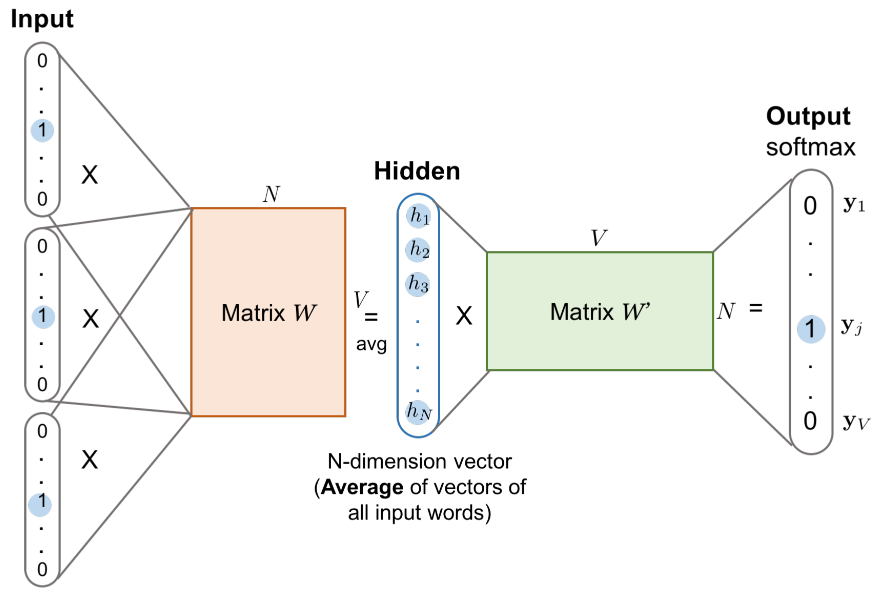

Figure 2: The CBOW model. Word vectors of multiple context words are averaged to get a fixed-length vector as in the hidden layer. [6]

0.3 Objective function

Skip-gram: maximization of the (log-)probability of all words in the context given the target word.

$$J_\theta = \frac{1}{T} \sum_{t=1}^{T} \sum_{j=-c, j\neq 0}^{c} \log p\left(w_{t+j} \mid w_t\right)$$

where T is corpus size and $2c - 1$ context size
CBOW: maximization of the (log-)probability of the target word given its context

$$J_\theta = \frac{1}{T} \sum_{t=1}^{T} \sum_{j=-c, j\neq 0}^{c} \log p\left(w_t \mid w_{t+j}\right)$$

0.4 fastText

fastText is deep learning framework developed by Facebook to capture context and meaning. fastText is the improvised version of word2vec. Word2vec basically considers words to build the representation but fastText takes each character while computing the representation of the word.

0.5 About the dataset

The QUAERO French Medical Corpus: The QUAERO French Medical Corpus has been initially developed as a resource for named entity recognition and normalization [2]. It was then improved with the purpose of creating a gold standard set of normalized entities for French biomedical text, that was used in the CLEF eHealth evaluation lab. It is a complete corpus, tokenized and with one sentence per line.

The QUAERO French Press Corpus: It is a complete corpus, tokenized and with one sentence per line.

0.6 Building CBOW ,Skip-gram, fastText(CBOW) model.

In this work we used Gensim library [3] which is an open-source library for unsupervised topic modeling and natural language processing. Python's gensim library allows us to build Word2vec and fastText model from scratch based on any provided dataset.
let's see how to build Skip-gram model

```
from gensim.models import Word2Vec
model_skipgram1 = Word2Vec(min_count=1,sg=1, size=100, window=10)
# sg=1 means skipgram, else CBOW
model_skipgram1.build_vocab(filtered_sentences1)  # The QUAERO French Medical Corpus
%time model_skipgram1.train(filtered_sentences1, total_examples=model_skipgram1.corpus_count, epochs=100)
```

For the CBOW model we use the same structure as below.
let's see how to build fastText(CBOW) model

```
from gensim.models.fasttext import FastText
%time model_fastText1 = FastText(filtered_sentences1, size=embedding_size, window=window_size, min_count=min_word, sample=down_sampling,sg=0, iter=10)
```

The size of the trained Word2vec vocabulary:

```
len(model_skipgram1.wv.vocab) # The QUAERO French Medical Corpus
8749
len(model_skipgram2.wv.vocab) # The QUAERO French Press Corpus
37770
```

0.7 Results and discussion

Using gensim we can get the most similar word to the target word on the corpus.

```
model_skipgram1.most_similar('douleurs')
# output
[('intenses', 0.7781764268875122), ('chroniques', 0.7395579814910889),
 ('prurit', 0.6572233438491821), ('ventre', 0.6440890431404114),
 ('hypersudation', 0.6434773802757263), ('crampes', 0.6406511664390564),
```

```
('chutes', 0.6405884623527527), ('alopécie', 0.6320321559906006),
('myalgie', 0.6283820867538452), ('variait', 0.6261775493621826)]
```

In order to get semantic similarity between vectors, we need to compare the Word embeddings. **Cosine similarity** is the most used method to compare two vectors.

$$\text{similarity} \; = \cos(\theta) = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

we can get the semantic similarity between two words by:

```
model_skipgram1.similarity('patient','souffre')
# output
0.58506334
model_skipgram1.similarity('patient','contenant')
# output
0.17926422
```

"patient" and "souffre" have a good amount of similarity, but the similarity between the words "patient" and "contenant" is poor.
In the following we test the different models with both Corpus:

	cbow	**cosine_sim**	**skipgram**	**cosine_sim**	**fastText**	**cosine_sim**
0	hospitalisé	0.472876	carte	0.689585	patientent	0.988254
1	médecin	0.426719	souffre	0.682668	impatient	0.975110
2	survivant	0.391949	encourus	0.679393	contient	0.973933
3	sociologues	0.388757	certitude	0.665628	détient	0.967145
4	panneau	0.387071	montrer	0.661070	maintient	0.966115
5	douar	0.381007	remarquer	0.657120	abstient	0.960881
6	mototaxi	0.377592	établi	0.656300	soutient	0.960649
7	détenu	0.371714	attentif	0.651464	impatientent	0.960281
8	flagrant	0.371424	partenaire	0.651376	initient	0.958548
9	cancéreux	0.369530	alerte	0.649682	réconcilient	0.957152

Table 1: 10 closest words to "**patient**" and the semantic similarity between them (French Medical Corpus)

According to this output 1 the embedding for 'patient' is most similar to 'hospitalisé' using CBOW , carte using skipgram and patientent using fastText (Crow). Intuitively, in general cbow et skipgram of word2vec shows good results however for 10 iteration fastText shows poor results.

	cbow	**cosine_sim**	**skipgram**	**cosine_sim**	**fastText**	**cosine_sim**
0	risque	0.635641	début	0.484610	Traitement	0.998459
1	rapport	0.586497	mois	0.482668	Taaitement	0.997637
2	patients	0.575386	concomitant	0.436435	trait	0.992628
3	pendant	0.553740	semaines	0.433045	Allaitement	0.992032
4	semaines	0.550677	instauration	0.432925	traitment	0.991985
5	SEP	0.545199	suspicion	0.428237	allaitement	0.991451
6	médecin	0.538581	répéter	0.426234	évitement	0.990768
7	contrôle	0.537119	commencer	0.425408	traitements	0.990294
8	maladie	0.534822	réintroduit	0.425116	étroitement	0.986305
9	délai	0.534306	poussées	0.424545	traite	0.984117

Table 2: 10 closest words to "**traitement**" and the semantic similarity between them (French Medical Corpus)

	cbow	cosine_sim	skipgram	cosine_sim	fastText	cosine_sim
0	Parkinson	0.788612	Parkinson	0.751326	Maladie	0.997336
1	liée	0.650960	AINS	0.710730	unique	0.997180
2	avancé	0.633127	Inflammation	0.706206	matière	0.996693
3	infection	0.625664	Hodgkin	0.636584	nombre	0.996459
4	affection	0.617914	Basedow	0.627445	modifier	0.995877
5	Recklinghausen	0.608928	constituée	0.624709	préférence	0.995854
6	SIDA	0.604304	Crohn	0.622536	prazépam	0.995814
7	survenue	0.566109	Cushing	0.619860	raison	0.995807
8	qui	0.553658	mouton	0.619349	malade	0.995751
9	neurophacomatose	0.552713	vraie	0.617380	suspect	0.995612

Table 3: 10 closest words to "**maladie**" and the semantic similarity between them (French Medical Corpus)

	cbow	cosine_sim	skipgram	cosine_sim	fastText	cosine_sim
0	pâle	0.910016	pâle	0.743741	foyer	0.998437
1	flavicollis	0.838099	orange	0.740688	dose	0.995271
2	Fabr	0.835394	anormale	0.704227	Trois	0.994966
3	Calotermes	0.821265	hexagonaux	0.697865	congeler	0.994638
4	dioxyde	0.818143	Fabr	0.692408	soin	0.994324
5	Talc	0.814534	danger	0.687820	conparé	0.993730
6	Ethylcellulose	0.803926	Urines	0.678204	fournir	0.993725
7	éthylcellulose	0.800130	mosaïque	0.677434	confirmé	0.993381
8	fer	0.799840	flavicollis	0.676081	aujourd	0.993358
9	Méthylhydroxypropylcellulose	0.796315	replicase	0.674103	utile	0.993320

Table 4: 10 closest words to "**jaune**" and the semantic similarity between them (French Medical Corpus)

since fastText is building on character level, even for the word that was not there in training, it will provide results.

Mikolov highlighted that the skip-gram approach works well with small corpora and rare terms. With the skip-gram approach, you'll have more examples due to the net- work structure. But the continuous bag-of-words approach shows higher accuracies for frequent words and is much faster to train.
In the figure 3 4 5 we use t-SNE to to visualize similarity between words [4].

The major difficulty in the use of these models is tuning the parameters of each model correctly adjusted to get the best results.

	cbow	cosine_sim	skipgram	cosine_sim	fastText	cosine_sim
0	équipé	0.429447	souffre	0.704553	patientent	0.986054
1	contaminé	0.423183	certitude	0.686332	impatient	0.974955
2	panneau	0.418939	carte	0.684310	contient	0.969230
3	foyer	0.400263	encourus	0.682910	détient	0.968870
4	palliatifs	0.393399	reconstitution	0.675334	maintient	0.960297
5	cancéreux	0.392417	avait	0.675073	abstient	0.959776
6	chirurgie	0.382860	impliqué	0.671785	impatientent	0.958508
7	hospitalisé	0.381749	telle	0.671246	soutient	0.956682
8	peluche	0.379551	conduisant	0.671007	réconcilient	0.954747
9	double	0.370263	montrer	0.668031	initient	0.953342

Table 5: 10 closest words to " **patient**" and the semantic similarity between them (French Press Corpus)

	cbow	cosine_sim	skipgram	cosine_sim	fastText	cosine_sim
0	collectif	0.485459	interrompre	0.688708	promptement	0.972839
1	système	0.477075	décision	0.683222	retraitement	0.966695
2	sida	0.474970	consignes	0.677983	prolongement	0.964781
3	générateurs	0.462132	associé	0.676350	réaménagement	0.962001
4	viol	0.434256	malgré	0.675283	doctement	0.959925
5	fonctionnement	0.425253	symptomatique	0.674607	dépècement	0.958641
6	fondement	0.422760	poussées	0.673262	sagement	0.958573
7	gériatrie	0.419574	contrôlée	0.673186	concrètement	0.958268
8	survivant	0.414400	début	0.671623	dédommagement	0.957592
9	couverture	0.411820	présent	0.670101	rayonnement	0.957189

Table 6: 10 closest words to "**traitement**" and the semantic similarity between them (French Press Corpus)

	cbow	cosine_sim	skipgram	cosine_sim	fastText	cosine_sim
0	épidémie	0.525049	Parkinson	0.857325	malade	0.918827
1	pneumopathie	0.520447	avancé	0.737111	malnutrie	0.901770
2	SIDA	0.505163	avait	0.734355	fantaisie	0.881138
3	atypique	0.473817	expérimentés	0.718634	graphologie	0.880724
4	proportion	0.451687	mouton	0.718522	mélodie	0.880164
5	virus	0.447277	solides	0.716906	trilogie	0.879384
6	pneumonie	0.443140	constituée	0.715785	vitrine	0.872162
7	miroiter	0.439345	liée	0.714865	magie	0.870124
8	grippe	0.439293	localement	0.714551	monture	0.867513
9	maladies	0.427537	épidémiologie	0.714112	pédagogie	0.866968

Table 7: 10 closest words to " **maladie**" and the semantic similarity between them (French Press Corpus)

	cbow	cosine_sim	skipgram	cosine_sim	fastText	cosine_sim
0	maillot	0.787785	pâle	0.876089	Neptune	0.938042
1	Pena	0.641003	orange	0.857020	brune	0.934810
2	Baden	0.628184	hexagonaux	0.856115	lune	0.919055
3	Saâdoune	0.619320	mosaïque	0.855006	Saâdoune	0.918677
4	Bradley	0.618020	oxyde	0.838076	Saadoune	0.912954
5	Armstrong	0.600443	soldats	0.829481	Jeune	0.901259
6	Lance	0.589145	exempte	0.824406	Abdoune	0.886888
7	McGee	0.568337	transparente	0.821675	Pampelune	0.877274
8	emparé	0.562755	fer	0.819734	lagune	0.870731
9	décaleront	0.558092	visibles	0.817816	dune	0.823846

Table 8: 10 closest words to " **jaune**" and the semantic similarity between them (French Press Corpus)

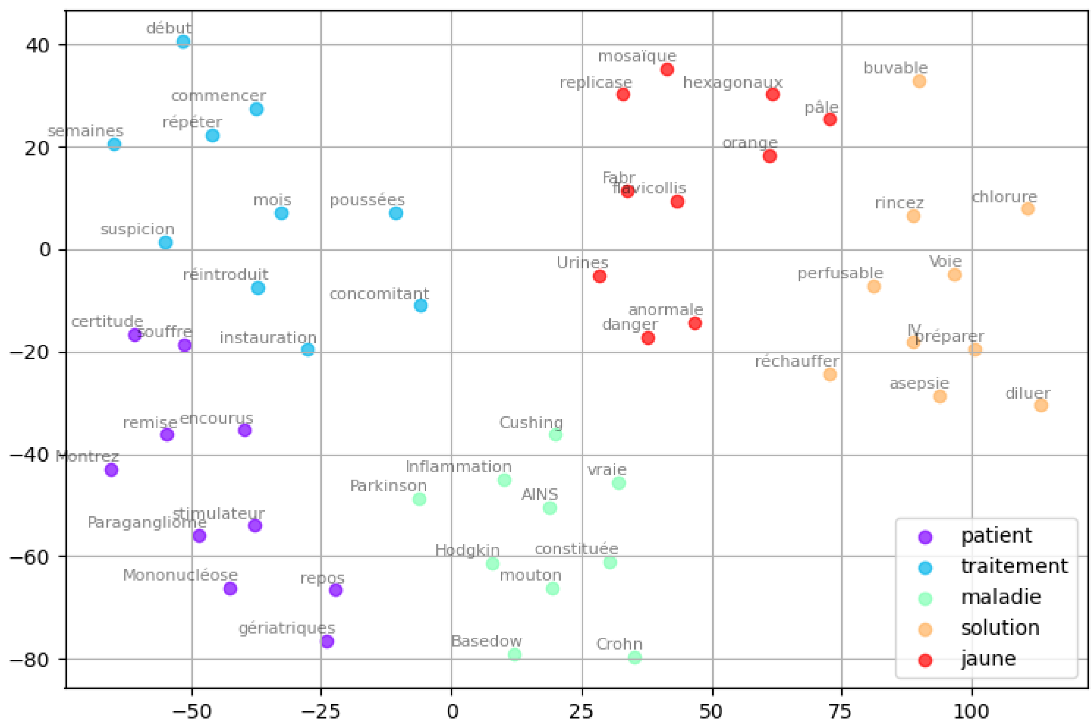

Figure 3: Visualizing Word Embeddings for skipgram (French Medical Corpus)

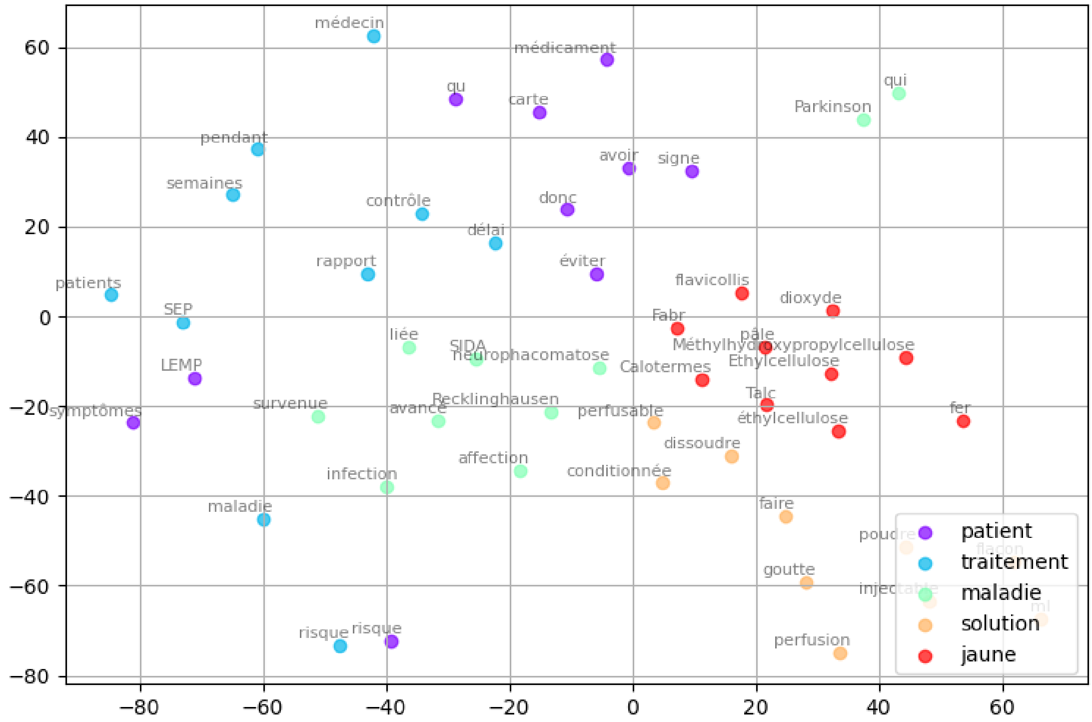

Figure 4: Visualizing Word Embeddings for cbow (French Medical Corpus)

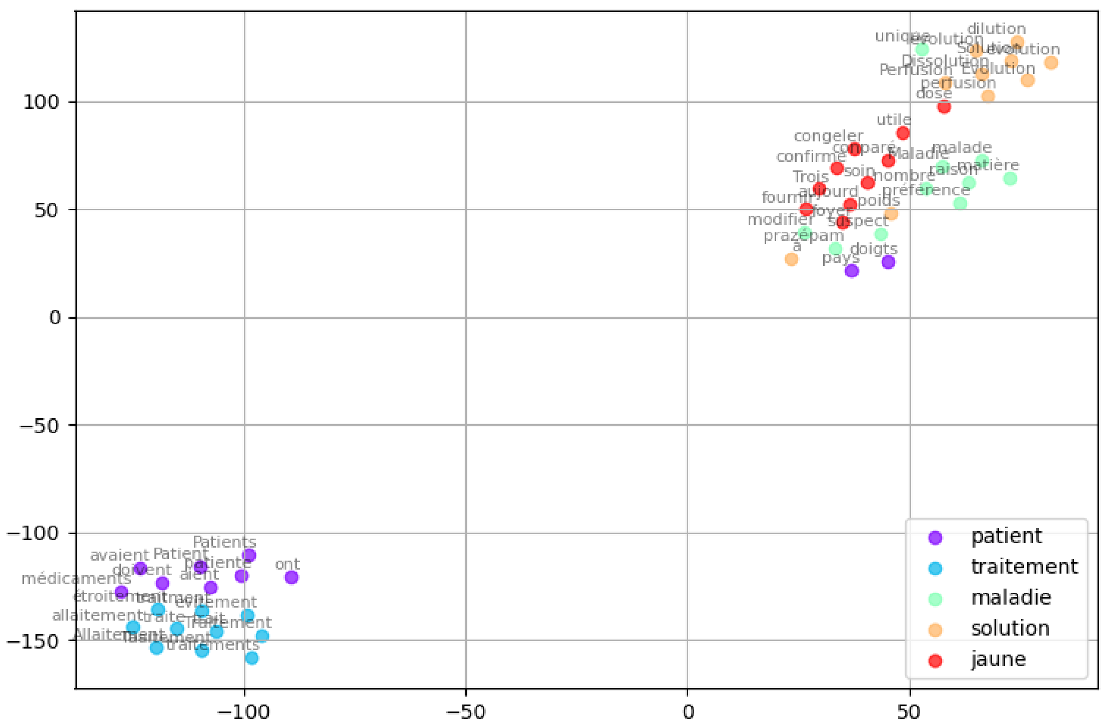

Figure 5: Visualizing Word Embeddings for fastText (French Medical Corpus)

Bibliography

[1] Tomás Mikolov et al. "Distributed Representations of Words and Phrases and their Compositionality". In: *CoRR* abs/1310.4546 (2013). arXiv: `1310.4546`. URL: `http://arxiv.org/abs/1310.4546`.

[2] Aurélie Névéol et al. "The Quaero French Medical Corpus : A Ressource for Medical Entity Recognition and Normalization". In: 2014.

[3] Radim Řehůřek and Petr Sojka. "Software Framework for Topic Modelling with Large Corpora". English. In: *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. `http://is.muni.cz/publication/884893/en`. Valletta, Malta: ELRA, May 2010, pp. 45–50.

[4] Sergey Smetanin. *Google News and Leo Tolstoy: Visualizing Word2Vec Word Embeddings using t-SNE*. 2018. URL: `https://towardsdatascience.com/google-news-and-leo-tolstoy-visualizing-word2vec-word-embeddings-with-t-sne-11558d8bd4d`.

[5] Lena Voita. *NLP Course Yandex School*. 2020. URL: `https://lena-voita.github.io/nlp_course.html`.

[6] Lilian Weng. *Learning Word Embedding*. 2017. URL: `https://lilianweng.github.io/lil-log/2017/10/15/learning-word-embedding.html`.