NLP3 - Lab1

Mathieu Rivier
mathieu.rivier@epita.fr

Moustapha Diop
moustapha.diop@epita.fr

Marius Dubosc
mariuus.dubosc@epita.fr

Philippe Bernet
philippe.bernet@epita.fr

December 2022

```
Running Truncated SVD over 8185 words...
Done.
```

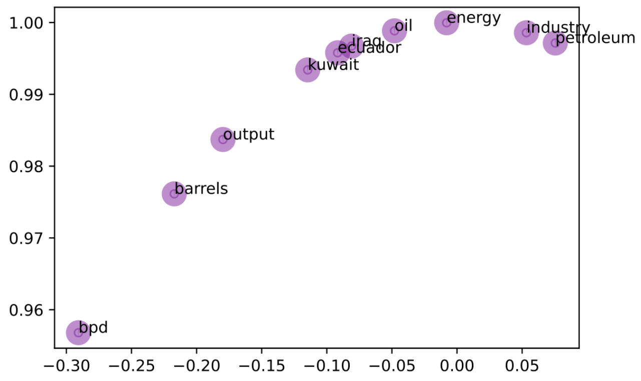


Figure 1: 2D co-occurence embeddings plot for selected words.

Abstract

This is the final course of a series of three on Natural Language Processing (NLP). This report represents the first explanation hand-out out of two practicals and explains the attached Jupyter-notebook. This report aims to explore 4 common ways of studying a corpus using NLP. Those will be treated in order and answer the questions from the notebook.

All the results mentioned in this report are reproducible in the above mentioned Jupyter-notebook.

Contents

1 Keywords Extraction

1.1 Preprocessing data in a meaningful way [code]

1.2 Build a top N words based on occurence [code]

1.3 What are some of the limits of raw counts? How could we improve the approach through preprocessing?

Counting the number of times each word appears in a document is a very simple approach to text processing, but it has several limitations. First, it does not account for the order of the words in the document, so two documents with the same words in different orders will be considered to be identical. Second, it does not account for different forms of the same word, so 'fish' and 'fishing' will be considered to be different words. Third, raw counts could be very low for rare words, and this could also skew the similarity results. Finally, it does not account for the context of the words in the document, so two documents with the same words but different meanings will be considered to be identical.

One way to improve the approach is to use a technique called "stemming" to reduce each word to its base form before counting. This will account for different forms of the same word. Further, we could use a weighting scheme such as tf-idf. This will palliate rare words getting low scores and skewing the results. Another way to improve the approach is to use a technique called "stopword removal" to remove common words such as 'a', 'the', and 'of' before counting. This will account for the context of the words in the document.

In question 1.1 we have implemented both the removal of stop words and the lemmatization which is the process of grouping together the different forms of a word so they can be analysed as a single item.

1.4 How can you find an optimal *max_df*? Why are we using a sparse matrix instead of a regular matrix?

The *max_df* parameter in a *CountVectorizer* indicates the maximum frequency of a word allowed in order to be included in the vocabulary. A higher *max_df* results in a smaller vocabulary.

One way to find an optimal *max_df* is to iterate over different values and compare the results. Another way is to use a grid search to test a range of values and find the best one.

There are a few reasons for using a sparse matrix with a *CountVectorizer*. First, a regular matrix requires a lot of memory, and a sparse matrix uses much less memory (because 0 values do not take up memory). Second, a sparse matrix can be created much faster than a regular matrix. Finally, a sparse matrix can be used with a much larger dataset than a regular matrix.

1.5 Find an example where there is a noticeable difference between tf-idf and raw counts? Justify which method you would choose yourself (there is no bad and good answer here)

We can see in this example (text 4) that the general words look similar. Although, upon inspection we realise that the words from Raw counts lack context and meaning for example we see learning as a keyword but what learning are we talking about machine learning, active learning, ...? On the other end, tf-idf is able to classify active learning as one of its keywords which actually gives more information on the contents of the article! We can further see those differences with error and generalisation error.

Overall at least on this example, tf-idf seems to give way more context than raw counts and seems to be a much better choice in this particular case.

Although we can generalise this observation as: raw counts are the number of times a term

```
Raw counts: [
    ('ensemble', 56),
    ('network', 52),
    ('error', 51),
    ('generalization', 40),
    ('set', 27),
    ('ambiguity', 26),
    ('weight', 26),
    ('learning', 22),
    ('training', 22),
    ('example', 22)
]
```

Figure 2: Raw counts top-n keywords for the same article as Figure 3

appears in a document which appear relatively naive. tf-idf is a measure of how important a term is to a document in a collection of documents which seems to be a more complete overview. It is the product of the term's raw count and a document frequency weight. tf-idf is thus better because it takes into account how often a term appears in a document.

```
tf-idf : [
    ('ensemble', 0.525),
    ('generalization error', 0.356),
    ('ambiguity', 0.302),
    ('generalization', 0.232),
    ('network', 0.21),
    ('error', 0.203),
    ('active learning', 0.195),
    ('cross validation', 0.14),
    ('active', 0.136),
    ('individual', 0.125)
]
```

Figure 3: Tf-idf top-n keywords for the same article as Figure 2

1.6 Apply KeyBERT to the a sample of the dataset [code] (1 point)

1.7 Comparison of multilple techniques [written] (4 points)

1.7.1 Draw a table of the solution, the quality score that you defined and the time taken to find keywords across a sample of 1000 of the original dataset.

Test	Time in sec
raw counts	0.89
tf-idf	2.50
Key BERT	3.25 min

Table 1: Time to run each method

Table 1 shows the different times it takes to run the different methods. It is apparent that the better the method is in this case, the more time it takes.

Accuracy	**raw counts**	**tf-idf**	**Key BERT**
raw counts	/	10.9%	3.6%
tf-idf	/	/	4.4%
Key BERT	/	/	/

Table 2: Accuracy between different models

Table 2 depicts the accuracy between each pair of method. This enables to have a comparison between the performance of the methods against one another. This is the only reasonable method to treat a considerable amount of data in this case. The other way being comparing them by hand, or asking users for feedback. A small program on top of the normal accuracy_score from sklearn was built to be able to more easily test. It jumps at us straight that tf-idf and raw-counts produce much closer keywords which makes sense since they are algorithmically closer, and more naive than KeyBERT. Although, the better method from this testing alone would be tf-idf.

Matching	**raw counts**	**tf-idf**	**Key BERT**
raw counts	/	56%	22%
tf-idf	/	/	23%
Key BERT	/	/	/

Table 3: Matching elements between different models

Table 3 portrays a metric that was invented and coded for this specific test. It checks how many of the keywords each method has in common for a specific document. The reader can find its implementation in the attached notebook. This metric is a good complement to accuracy to understand what exactly is different between the different methods. The results are much higher, which indicate that actually, the predictions are rather similar especially for tf-idf and raw counts, meaning the order of keywords is the main difference between the two methods. A significant improvement is further noted in comparisons with the KeyBERT model.

1.7.2 Based on the above table and lecture 1, what do you think is the most appropriate solution for keywords extraction? Why?

From those results alone, it would be tempting to classify tf-idf as the best solution for key-word extraction. It is faster and "tests" better. Although, it is important to remember from lecture 1, that KeyBERT is a much better model in terms of accuracy and relevance of the keywords produced. As such the difference between it and the two others should be seen as under performance of the two other methods rather than a defect of KeyBERT. This said KeyBERT is much slower than both of the other methods. Making it hard to justify for very voluminous tasks (with small hardware).

The decision thus resides on the use case. If the goal is to achieve best in class predictions regardless of the cost, then KeyBERT is no doubt the best solution that has been studied in this paper. If time and hardware are more of a concern though, tf-idf has shown reasonable performance (see question 1.5, while being much faster.

2 Word Vectors

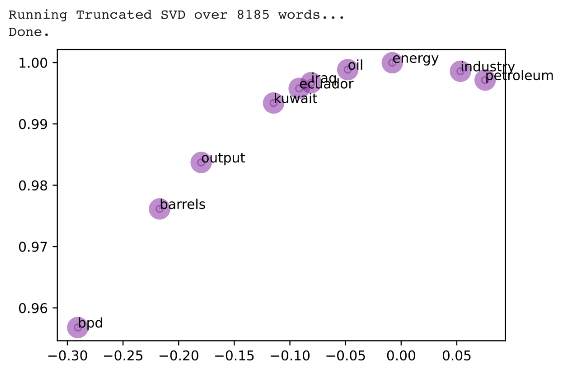
2.1 Implement `distinct_words` [code]

2.2 Implement `compute_co_occurrence_matrix` [code]

2.3 Implement `reduce_to_k_dim` [code]

2.4 Implement `plot_embeddings` [code]

2.5 Co-Occurrence Plot Analysis



```
Running Truncated SVD over 8185 words...
Done.
```

Figure 4: 2D co-occurence embeddings plot for the following words: 'barrels', 'bpd', 'ecuador', 'energy', 'industry', 'kuwait', 'oil', 'output', 'petroleum', 'iraq'.

A 2-dimensional embedding space is a graphical representation of a set of word embeddings, where each word is represented by a point in the space. Word embeddings are numerical representations of words that capture the meaning and context in which the words are used.

A 2-dimensional embedding space, is read by:

1. Looking at the x-axis and y-axis labels. These should correspond to the dimensions of the embedding space.
2. Observing the positions of the points in the space. Words that are closer together in the space are typically more similar to each other in meaning or context, while words that are farther apart are typically less similar.
3. Looking for patterns in the positions of the points. For example, you might notice that certain words tend to cluster together in certain areas of the space. This could indicate that these words are related or have a similar meaning.
4. looking at the distances between points to get a sense of the similarity between different words is also possible. For example, if two points are very close together, it could indicate that the words represented by those points are very similar in meaning.

It's important to keep in mind that the positions of the points in the embedding space are determined by the algorithm that was used to generate the embeddings. Different algorithms may produce different results, so it is important to consider the specific context in which the embeddings were created when interpreting the results.

In figure 4's case that means all the words are very similar since they are located in the upper center of a [-1; 1] 2-dimentional space.

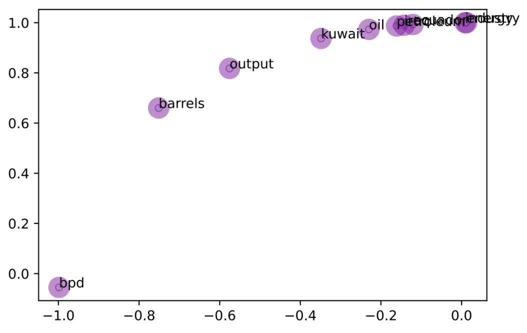
Most words must be very similar in usage since only 3 out of the 10 words seem to be slightly off from the others. In fact, when looking closely these words ('bpd', 'barrels' and 'output') are related in that they represent or imply measures or output. 'berrels' is a quantity, 'bpd' is the barrels per day – daily production of barrels – which represent a quantity per day, and output represents what has been produced or said differently a quantity. This may be considered as a first cluster of outcasts since they are linked with one another. A reason they may be outcasts is that they might be used in a slighly different manner from the other words, being used to evaluate quality or profitability of an exploitation rather than describing it.

Another cluster seems to be 'iraq', 'kuwait', 'ecuador', and oil are locations that may in fact be used in the same way in a sentence, and for oil be used often with for locations.

A third and final cluster can be seen as 'oil', 'energy', 'industry', 'petroleum' which would make sense since those words would be used similarly in the documents studied here.

3 Prediction-based word vectors

3.1 GloVe Plot Analysis



```
<Figure size 432x288 with 0 Axes>
```

Figure 5: 2D GloVe embeddings plot for the following words: 'barrels', 'bpd', 'ecuador', 'energy', 'industry', 'kuwait', 'oil', 'output', 'petroleum', 'iraq'.

As mentioned in question 2.5, the closeness of words to one another represents a similar meaning or usage in sentences. It can also mean they are often used together.

First, it is important to point out that while the scale $[-1, 1]$ two-dimensional remains the same as in question 2.5, the graph is although representing a different range x-axis: $[-1, 0.1]$, y-axis: $[-0.2, 1]$ compared to x-axis: $[-0.30, 0.1]$, y-axis: $[0.95, 1]$ for question 2.5. The values are thus more distributed throughout the range.

Similarly to previous graph figure 4, most words are really close together in the upper center of the scale. Furthermore, the words that stick out from the pack are the same from those in figure 4.

Although, the difference with 'bpd' in this graph, figure 5 compared to 'bpd' in figure 4 is that the point is much further down from other points meaning less related in meaning or usage. What is observed here is that 'bpd' is excluded from its similar words. Furthermore, 'barrels' and 'output' are also separated from the other points but in a lesser manner. 'bpd' and 'barrel' should be close to one another since they are both measures of quantity. The output should also be relatively close as 'bpd' represents the raw output for a day.

While looking very similar to the previous graph figure 4 when omitting scale, 'bpd' cannot be considered part of a cluster anymore.

A possible reason for the difference between figure 4 from the previous question 4 and figure 5 is that while co-occurence was used previously meaning the times words occured next to each other, in this graph prediction-based word embedding is used, and the method to compute similarity may be slightly different.

3.2 Words with Multiple Meanings [code + written]

It is most probable that since the explored texts are published articles, most polysemous words are used in a certain sense and usually would not appear in both senses. For example 'sound' which both means safe and a noise is only found as its noise definition. Although, in the case of express we can see that both the speed and transport definitions appear in the top 10.

```
[
    ('expressing',
    ↪  0.5767321586608887),
    ('expressed', 0.5290095210075378),
    ('train', 0.5205782651901245),
    ('sympathy', 0.5139451622962952),
    ('trains', 0.5050660371780396),
    ('expresses',
    ↪  0.49935826659202576),
    ('bus', 0.4954938292503357),
    ('regret', 0.49006834626197815),
    ('sorrow', 0.4864082932472229),
    ('travel', 0.47741514444351196)
]
```

Figure 6: Multiple meanings of the word 'express': 1. 'expressing' – emotions and 2. 'travel' – speed

3.3 Synonyms Antonyms [code + written]

The cosine distance between two words is a measure of the distance between the vectors representing those words in a vector space. It is calculated by taking the angle between the vectors and converting it to a value between 0 and 1 using the formula $1 - cos(angle)$. Cosine distance ranges from 0 to 1, with a value of 0 indicating that the vectors are identical and a value of 1 indicating that the vectors are completely opposite.

```
Cosine Distance w1-w2:
↪  0.6250409185886383
Cosine Distance w1-w3:
↪  0.5616524815559387
cosine Distance w1-w3 < w1-w2: True
```

Figure 7: Cosine distance with synonyms & antonyms with `w1 = "hello"` – `w2 = "hi"` – `w3 = "bye"`

It is possible for the cosine distance between the words 'hello' and 'bye' to be lower than the cosine distance between the words 'hello' and 'hi', even though 'bye' and 'hello' are antonyms and ‘hi’ and ‘hello’ are synonyms, depending on the vectors that are used to represent these words in a vector space. See figure 7.

If the vectors for the words ‘hello’ and ‘hi’ are pointing in similar directions, and the vectors for the words ‘hello’ and ‘bye’ are also pointing in similar directions, but more similar than the vectors for ‘hello’ and ‘hi’ the cosine distance between ‘hello’ and ‘bye’ will be lower than the cosine distance between ‘hello’ and ‘hi’ This could happen if the vectors for ‘bye’ and ‘hello’ are more similar to each other than the vectors for ‘hi’ and ‘hello’ for example.

It’s worth noting that the exact cosine distance between two words can vary depending on the context in which they are used and the other words that appear in the same context. In some cases, two words that are normally considered very different might have a lower cosine distance if they are used in similar contexts or if there are other words present that make the vectors for those words more similar.

3.4 Analogies with Word Vectors

The expression with which we maximise cosine similarity is the following:

$$x = k - m + w$$

If the `man : king :: woman : x` analogy is taken as example, and with the above expression in mind:

- **x** would be the prediction (i.e : ‘Queen’)
- **k** would be the word to transfer (i.e: ‘king’)
- **m** would be the genre of the initial word (i.e: ‘man’)
- **w** would be the targeted genre (i.e: ‘woman’)

3.5 Finding Analogies [code + written]

```
First analogy: "man : prince :: woman
↪   : x":
[('princess', 0.7453499436378479),
 ('duchess', 0.6067375540733337),
 ('daughter', 0.5600142478942871),
 ('queen', 0.5452842712402344),
 ('hrh', 0.5299034118652344),
 ('wife', 0.5115962028503418),
 ('marry', 0.5082696676254272),
 ('naruhito', 0.5037658214569092),
 ('mistress', 0.5026963949203491),
 ('crown', 0.4981675446033478)
]
```

Figure 8: First correct analogy: looking for `x = princess`

Both `man : prince :: woman : x` figure 8 and `girl : sister :: boy : x` figure 9 correctly identify the analogies. We thus have the feminine version of prince being princess and the masculine version of sister being brother.

```
Second Analogy: "girl : sister :: boy
↪   : x":
[('brother', 0.7562326192855835),
 ('father', 0.7254698276519775),
 ('mother', 0.7100989818572998),
 ('son', 0.7058944702148438),
 ('cousin', 0.7000033855438232),
 ('daughter', 0.6871263384819031),
 ('uncle', 0.6804952025413513),
 ('siblings', 0.677232027053833),
 ('elder', 0.6522819995880127),
 ('nephew', 0.6461338996887207)
]
```

Figure 9: Second correct analogy: looking for `x = brother`

3.6 Incorrect Analogy [code + written]

```
First analogy: "woman : wife :: man :
↪  b":
[('father', 0.7467501163482666),
 ('brother', 0.745741069316864),
 ('husband', 0.7437037825584412),
 ('son', 0.7166001796722412),
 ('friend', 0.7085943222045898),
 ('his', 0.6684678792953491),
 ('cousin', 0.6388828754425049),
 ('daughter', 0.6335204243659973),
 ('mother', 0.6283376216888428),
 ('uncle', 0.6279417276382446)
]
Got "father" when expected "husband".
```

Figure 10: First incorrect analogy: looking for `b != husband`

Both `woman : wife :: man : b` and `girl : brother :: boy : b` wrongly identified the analogies. We thus have the masculine version of wife being father instead of husband and feminine the version of brother being daughter instead of sister.

```
Second Analogy: "girl : brother :: boy
↪  : b":
[('daughter', 0.8058238625526428),
 ('cousin', 0.7796305418014526),
 ('son', 0.7571232914924622),
 ('wife', 0.7448716163635254),
 ('sister', 0.7441308498382568),
 ('father', 0.7369656562805176),
 ('niece', 0.7302137613296509),
 ('nephew', 0.7301906943321228),
 ('husband', 0.7162861824035645),
 ('mother', 0.7117223739624023)
]
Got "daughter" when expected "sister".
```

Figure 11: Second incorrect analogy: looking for `b != sister`

3.7 Guided Analysis of Bias in Word Vectors

```
[('employee', 0.6375863552093506),
 ('workers', 0.6068919897079468),
 ('nurse', 0.5837947726249695),
 ('pregnant', 0.5363885164260864),
 ('mother', 0.5321309566497803),
 ('employer', 0.5127025842666626),
 ('teacher', 0.5099576711654663),
 ('child', 0.5096741914749146),
 ('homemaker', 0.5019454956054688),
 ('nurses', 0.4970572590827942)
 ]
```

Figure 12: Study of bias on `man : worker :: woman : x`

```
[('workers', 0.6113258004188538),
 ('employee', 0.5983108282089233),
 ('working', 0.5615328550338745),
 ('laborer', 0.5442320108413696),
 ('unemployed', 0.5368517637252808),
 ('job', 0.5278826951980591),
 ('work', 0.5223963260650635),
 ('mechanic', 0.5088937282562256),
 ('worked', 0.505452036857605),
 ('factory', 0.4940453767776489)
 ]
```

Figure 13: Study of bias on `woman : worker :: man : x`

The list of female-associated words in Figure 12 includes terms such as ‘nurse’, ‘mother’, ‘teacher’, and ‘homemaker’ which are often traditionally considered to be female-dominated occupations or roles. The list of male-associated words in Figure 13 includes terms such as ‘mechanic’ and ‘laborer’ which are also traditionally considered to be male-dominated occupations or roles.

This difference between the two lists reflects gender bias, as it reflects the societal biases and stereotypes that have been encoded in the data used to train the word embeddings. These biases can have harmful consequences, as they can reinforce and perpetuate harmful stereotypes and discriminate against certain groups of people. It is important to be aware of these biases and take steps to mitigate them in any application that uses word embeddings or other language models.

3.8 Independent Analysis of Bias in Word Vectors [code + written]

```
[('affluent', 0.6492576599121094),
 ('socialite', 0.5705443024635315),
 ('middle-class', 0.5464404225349426),
 ('aristocratic', 0.5330350399017334),
 ('businesswoman',
 ↪  0.5314464569091797),
 ('marry', 0.5257409811019897),
 ('wealthier', 0.5169167518615723),
 ('elderly', 0.5112395882606506),
 ('marrying', 0.510892927646637),
 ('married', 0.5068483948707581)
]
```

Figure 14: Study of bias on `man : wealthy :: woman : x`

```
[('wealthiest', 0.6119896173477173),
 ('businessman', 0.5537059307098389),
 ('rich', 0.5525563955307007),
 ('businessmen', 0.5454820394515991),
 ('affluent', 0.5427639484405518),
 ('richest', 0.5321693420410156),
 ('tycoons', 0.5197821855545044),
 ('billionaire', 0.5065683126449585),
 ('millionaire', 0.49499309062957764),
 ('fortune', 0.4918596148490906)
]
```

Figure 15: Study of bias on `woman : wealthy :: man : x`

The list of female-associated words in Figure 14 includes terms such as marry and marrying, married, which are often traditionally associated with wealthy women. Supposing that woman cannot make their own money and have to rely on men which is completely false. The list of male-associated words Figure 15 includes terms such as businessman, tycoons, affluent, and billionaire, which are also traditionally associated with men and wealth. But this time putting glory on the men again. This is hugely biased because a man could be a son or a husband as much as a woman can be a businesswoman or affluent of her own right.

3.9 Thinking About Bias

One way that bias can get into word vectors is through the data that is used to train the word embedding model. If the data used to train the model reflects societal biases and stereotypes, then the word vectors produced by the model may also reflect these biases.

One experiment that could be used to test for or measure this source of bias is to evaluate the word vectors on a set of benchmarks specifically designed to measure bias. There are several such benchmarks available, such as the Word Embedding Association Test (WEAT) and the Word Embedding Association Test for Gender (WEAT-G). These benchmarks allows one to quantitatively measure the degree of bias present in a given set of word vectors by comparing the similarity between pairs of words that are related to a particular bias (e.g. gender, race, sexual orientation, etc.) against the similarity between pairs of words that are not related to that bias.

Another way to test for or measure bias in word vectors is to manually examine the most similar words for a given word and see if the resulting list reflects societal biases and stereotypes. For example, if we find that the most similar words for the word 'woman' include terms such as 'nurse', 'homemaker', and 'secretary', while the most similar words for the word 'man' include terms such as 'businessman', 'engineer', and 'doctor' this may indicate that the word vectors are reflecting societal biases and stereotypes about the roles and occupations of men and women, which is what we have observed in this practical.

4 Prediction-based sentence vectors

4.1 How would you represent a sentence with Glove? What are the limits of your proposed implementation?

To represent a sentence with GloVe (Global Vectors for Word Representation), we can simply take the mean of the word vectors for all of the words in the sentence. This will give us a single vector that represents the overall meaning of the sentence.

For example, given the sentence "The cat sat on the mat" we can represent it with GloVe by taking the mean of the word vectors for 'the', 'cat', 'sat', 'on', 'the', and 'mat'. This will give us a single vector that captures the overall meaning of the sentence.

One limitation of this approach is that it does not take into account the order or arrangement of the words in the sentence. The meaning of a sentence can often be influenced by the order in which its words are arranged, and this information is lost when we simply take the mean of the word vectors. Another limitation is that this approach does not capture any context beyond the words themselves. In natural language, the meaning of a word can be influenced by the words that come before or after it, and this information is not captured by simply taking the mean of the word vectors.

To overcome these limitations, more advanced approaches to sentence embedding may be used, such as using recurrent neural networks (RNNs) or transformers to encode the entire sentence and capture the context and order of the words within it.

4.2 Evaluate clustering quality of SentenceBERT. What makes it good at clustering sentences? Which method of the two below would you go for?

4.2.1 Evaluate clustering quality of SentenceBERT. What makes it good at clustering sentences?

SentenceBERT is a type of transformer-based language model that has been pre-trained on a large dataset of sentences. It is designed to be able to understand the meaning and context of a sentence and to be able to represent that meaning in a numerical form, known as an embedding. This ability to represent the meaning of a sentence in a numerical form makes SentenceBERT well-suited for use in natural language processing tasks such as sentence clustering.

There are several factors that contribute to the effectiveness of SentenceBERT for clustering sentences:

- Pre-training on a large dataset: By pre-training SentenceBERT on a large dataset of sentences, it has learned to understand the meaning and context of a wide range of sentences, which makes it better at understanding the meaning of new sentences it has not seen before.
- Use of transformer architecture: The transformer architecture used in SentenceBERT allows it to capture long-range dependencies between words in a sentence, which is important for understanding the meaning of a sentence.
- Attention mechanism: The attention mechanism in SentenceBERT allows it to focus on specific words or phrases in a sentence and to weight their importance when representing the meaning of the sentence. This can be useful for identifying key phrases that are important for clustering sentences.
- Sentence embeddings: SentenceBERT produces sentence embeddings that represent

the meaning of a sentence in a numerical form. These embeddings can be used to compare the similarity between sentences, which is useful for clustering.

Overall, the combination of these factors makes SentenceBERT a powerful tool for clustering sentences and for other natural language processing tasks.

4.2.2 Which method of the two below would you go for?

In the term of the answers both methods obtain the exact same results in this case. Although, there is a clear way to distinguish both technics:

- Agglomerative Clustering: "Recursively merges pair of clusters of sample data; uses linkage distance." - Agglomerative Clustering - sklearn documentation.
  - Although is computationally expensive. - 2.3.6. Hierarchical clustering - sklearn's doc
- K-means Clustering: "In practice, the k-means algorithm is very fast (one of the fastest clustering algorithms available), but it falls in local minima. That's why it can be useful to restart it several times." - KMeans – sklearn documentation

It thus means that a choice needs to be made between quality of prediction and optimisation / complexity. If the goal is to get a somewhat correct result very fast, then K-means is one of the fastest clustering algorithms. Although it has a limit in the number of connections it looks for meaning it may be less precise than Agglomerative Clustering. On its end agglomerative clustering does not fall in nearly as many problems in terms of local minima of prediction constraints. It will create as many pairs (clusters) as needed and compare each one again each other pair until having the best clusters. This makes it much more reliable but also much slower. If the importance is placed on results quality and the corpus is resonable, Agglomerative clustering may be a great option. On the other end, if speed is a top priority or the corpus is extremly large, using K-means may seem to be a better option.

4.3 SentenceBERT Plot Analysis

The goal here is to cluster similar sentences together. Each of the five plots represents one of the clusters plus the 'food' word as a reference across different plots.

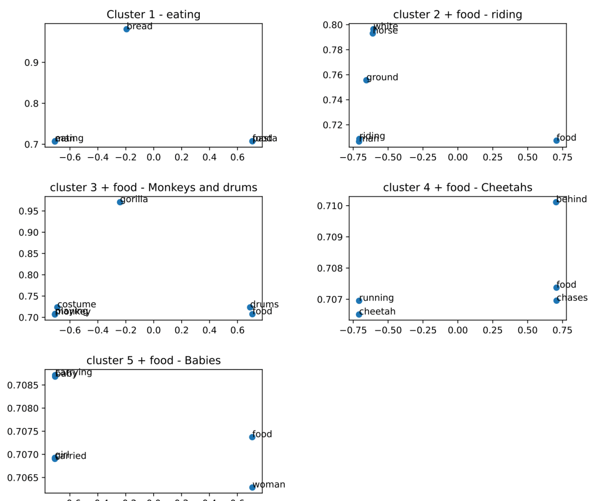


Figure 16: Plot of each Cluster's keywords

Study of figure 16:

- **Plot 1**:
  in this graph three clear clusters appear. The first one on the bottom left containing: 'man', 'eating'. The second one in the middle containing only 'bread'. The final one containing: 'food' and 'pasta'. Those clusters make sense for the most part. Except for 'bread' that should be in the same cluster as 'food' and 'pasta'.
- **Plot 2**:
  In this graph there are 2 clusters and two isolated words. The clusters are 'white', 'horse' and 'man', 'riding'. The isolated words are 'food' and 'ground'. The isolation of 'food' in this case makes sense while

the isolation of ‘ground’ is also understanding as it describes something different from the other words.

- **Plot 3**:

  In this graph two clusters and one isolated word are present. The clusters are ‘costume’, ‘monkey’, ‘playing’ and ‘drums’, ‘food’. The clusters make sense. While a person might classify the isolated word ‘gorilla’ in the same cluster as ‘costume’ and ‘monkey’ in this case it does not really serve the same role in the sentence nor does it occupy the same place in the sentence.

- **Plot 4**:

  There are two clusters and one isolated word in this plot. The clusters are ‘running’, ‘cheetah’ and ‘food’, ‘chases’. The isolated word is ‘behind’. This clustering makes sense.

- **Plot 5**:

  There are two clusters and two isolated words in this plot. The clusters are: ‘baby’, ‘crying’ and ‘girl’, ‘carrying’. The isolated words are ‘food’ and ‘woman’. This is a miss-classification. In fact, ‘woman’ should be in the same cluster as ‘girl’ although as can be seen, the words were clustered with the verbs directly following.

It is further apparent that in most of the plots, the subject and action are gathered in the same cluster.

```
score: 0.5851 -- text: ['Vaccination
↪  is a barbarous practice, and it is
↪  one of the most fatal of all the
↪  delusions current in our time, not
↪  to be found even among the
↪  so-called savage races of the
↪  world...']
score: 0.5448 -- text: ['One thing is
↪  certain.  In the book "Vaccines,
↪  are they really safe and
↪  effective?"  page 20-21  Says that
↪  in 1959 ...']
score: 0.4961 -- text: ["The vaccine
↪  that are commonly used do use
↪  adjuvants. Often that leads to
↪  adverse effects.\nThe study that
↪  approved the Pandemrix® vaccine
↪  noted that the vaccine gave  21.5%
↪  of children between 6 and 9
↪  headaches..."]
score: 0.4497 -- text: ["Some studies
↪  into COVID and others into
↪  Influenza contradict the
↪  government position on the
↪  efficacy of masks suggest there's
↪  no benefit."]
score: 0.4431 -- text: ['Yes, but it
↪  goes even further back than
↪  that:\n\nMADRID (Reuters) -
↪  Spanish virologists have found
↪  traces of the novel...']
score: 0.4187 -- text: ["Note: This
↪  isn\'t intended as a complete
↪  answer, only a partial one (which
↪  is too extensive to place in
↪  comments).\n\nA Washington
↪  Post..."]
```

Figure 17: Study of bias on in stackexchange’s dataset: prompt - “The Covid vaccine is”

4.4 Independent Analysis of Bias in Word Vectors [code + written]

Like any machine learning model, SentenceBERT can be subject to various sources of bias, including:

- Dataset bias: If the dataset on which SentenceBERT was trained is not diverse or representative of the languages, dialects, and communities that the model will be applied to, the model may perform poorly or unfairly on those languages, dialects, or communities. To mitigate this bias, it is important to carefully select a diverse and representative dataset for training the model.

- Algorithmic bias: The algorithms used to train SentenceBERT, as well as the hyperparameters and optimization techniques used, can influence the model's performance and the biases it exhibits. For example, if the model is trained on a heavily imbalanced dataset, it may learn to favor the majority class, leading to biased predictions.

- Human bias: The data used to train SentenceBERT, as well as the labeling and annotation process, may be influenced by human biases. For example, if the dataset contains biased or stereotypical language, the model may learn and reproduce these biases in its predictions.

The bias studied here is the dataset bias. Biased data has been injected as the input corpus. As expected from the "Garbage in, Garbage out" rule, the outputs present bias. Before to look at the examples, it is interesting to note that finding bias with SentenceBERT was considerably harder than with the previous methods, possibly because the pretrained model used here is trained on good data that has reduced bias to a minimum. From the examples above it is possible to note that:

- "Vaccination is a barbarous practice, and it is one of the most fatal of all the delusions current in our time, not to be found even among the so-called savage..."
- "Some studies into COVID and others into Influenza contradict the government position on the efficacy of masks suggest there's no benefit"

and the other answers from Figure 17 are disproportionately against Vaccins in general where as studies have shown the positive effects of vaccines on society. Those results are not only biased but also provide miss-information.