Network Of Words

Ido Terner, Chanoch Baranes, Oren Glickman, Reuven Cohen, Simi Haber
September 2021

Abstract

We constructed a vector space using artificial intelligence methods for four different languages (English, German, French, and Spanish) in each, we built a suitable graph according to the similarity between the vectors, after which we calculated for each the following features: Degree Distribution, Connected Components, Graph Diameter, Centrality, Clustering Coefficient, and Average Path Length respectively. In addition, we clustered each graph according to K-Means algorithm. Finally, we used feature extraction methods to find similarities between the languages to which we found that there is a similarity that isn't trivial.

`https://github.com/iTerner/Network-Of-Words`

1 Introduction

To achieve our goal we used several definitions and techniques from various branches of Mathematics and Computer Science field such as Graph Theory and Natural Language Processing also known as NLP. In the following paragraphs we will represent a brief description of those branches.

1.1 Graph Similarity

Graph similarity has numerous applications in diverse fields (such as social networks, image processing, biological networks, chemical compounds, and computer vision), and therefore there have been suggested many algorithms and similarity measures. The proposed techniques can be classified into three main categories: edit distance/graph isomorphism, feature extraction, and iterative methods. In our project, we selected to compute the similarity based on feature extraction.

The key idea behind feature extraction is that similar graphs probably share certain properties, such as degree distribution, diameter, centrality, etc. After extracting these features, a similarity measure is applied to assess the similarity between the aggregated statistics and, equivalently, the similarity between the graphs. These methods are powerful and scale well, as they map the graphs to several statistics that are much smaller in size than the graphs. However, depending on the statistics that are chosen, it is possible to get results that are not intuitive. For instance, it is possible to get high similarity between two graphs that have very different node-set sizes, which is not always desirable.

Extracting features from graphs is completely different from normal data. Each node is interconnected with each other and this is important information that we can't just ignore. Fortunately, many feature extraction methods have been created for graphs. These techniques can be divided into *graph level*, *node level*, and *edge level* features as explained respectively.

Graph Level: What if we want to capture information about the whole graph instead of looking at individual nodes? Fortunately, there are many methods available that aggregate information about the whole graph. The features we chose in our research are Degree Distribution, Connected Components, and Graph Diameter.

Definition 1.1 (Degree Distribution)**.** The degree distribution of a graph is defined to be the fraction of nodes in the graph with degree k.

Definition 1.2 (Connected Component)**.** In graph theory, a component of an undirected graph is an induced sub-graph in which any two vertices are connected by paths, and which is connected to no additional vertices in the rest of the graph.

Definition 1.3 (Giant Connected Component)**.** In network theory, a giant component is a connected component of a given graph that contains a finite fraction of the entire graph's vertices.

Definition 1.4 (Graph Diameter)**.** The diameter d of a graph is the maximum eccentricity of any vertex in the graph. That is, d is the greatest distance between any pair of vertices.

Node Level: One of the simplest ways to capture information from graphs is to create individual features for each node. The features we chose to compare are Centrality and the Clustering Coefficient.

Definition 1.5 (Centrality)**.** Centrality assigns numbers or rankings to nodes within a graph corresponding to their network position.

Definition 1.6 (Clustering Coefficient)**.** In graph theory, a clustering coefficient is a measure of the degree to which nodes in a graph tend to cluster together.

Edge Level: The node and graph level features fail to gather information about the relationship between neighboring nodes. The Edge Level features to address this problem and measure both local and global overlaps in the graph. The feature we select is the Average Path Length.

Definition 1.7 (Average Path Length)**.** The average path length is a concept in graph topology that is defined as the average number of steps along the shortest paths for all possible pairs of graph nodes. It is a measure of the efficiency of information or mass transport on a graph.

$$l_G = \frac{1}{n*(n-1)} * \sum_{i \neq j} d(v_i, v_j)$$

Where $d(v_i, v_j)$ is the shortest distance between v_i and v_j and n is the number of vertices in the graph.

1.2 Word2Vec

Word2Vec is a shallow, two-layer neural network that is trained to reconstruct linguistic contexts of words. It takes as its input a large corpus of words and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located in close proximity to one another in the space. The Word2Vec algorithm uses the *skip-gram* model which predicts surrounding context words from the target words. Statistically, skip-gram treats each context-target pair as a new observation, and this tends to do better when we have larger datasets. To produce the embedding, the algorithm performs several steps. First of all, we cannot feed a word as a string into a neural network. Instead, we feed words as one-hot vectors, which is a vector of the same length as the vocabulary, filled with zeros except at the index that represents the word we want to represent, which is assigned "1". The hidden layer is a standard fully connected (Dense) layer whose weights are the word embeddings. The output layer outputs probabilities for the target words from the vocabulary. The use of different model parameters and different corpus sizes can greatly affect the quality of a word2vec model, such as *Dimensionality* and *Context window* The dimensionality is the number of dimensions of each word vector, typically between 100-1000, and the context window determines how many words before and after a given word would be included as context words of the given word.

2 Preparing The Data

To construct the graphs, we must first find a word list (also known as corpus) large enough for us to use it as our language. The reason we need a large corpus is that to have an

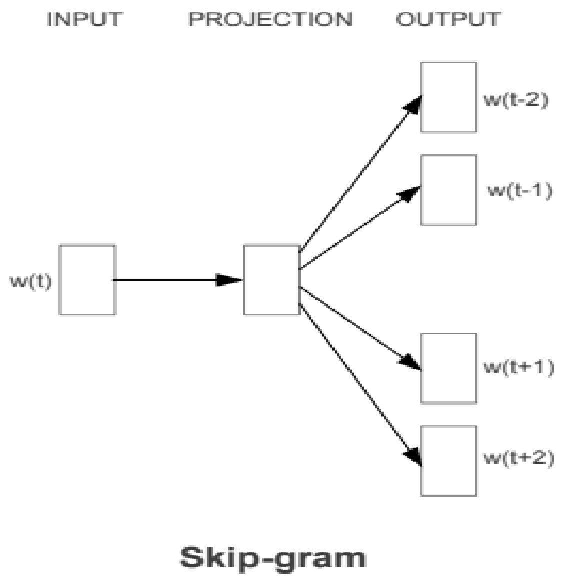


Figure 1: A simple visualization of the skip-gram model

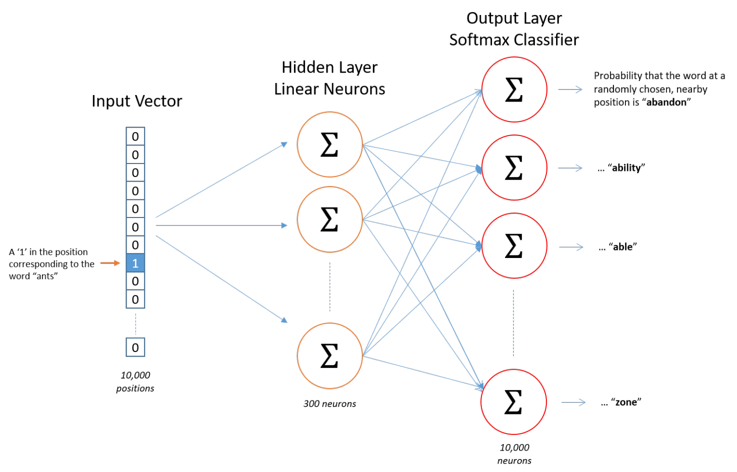


Figure 2: A simple visualization of the Word2Vec skip-gram algorithm. The input to this network is a one-hot vector representing the input word, and the label is also a one-hot vector representing the target word, however, the network's output is a probability distribution of target words, not necessarily a one-hot vector like the labels.

accurate embedding we need a large dataset to train it, so the bigger the corpus, the greater embedding. Next, we will have to convert the language into a vector space and run the graph construction algorithm. We parse the corpus from Wikipedia by processing over 10,000 articles and converted them into a frequency dictionary so we could transform them into a vector space and then select the words from them that appeared most frequently. After various tests on the best minimum threshold for the data, we found that the optimal number of times for a vector word to be sufficiently accurate stands at 1200 times.

3 Building The Graph

Before creating the graph we first must determine how we connect the words. In recent studies, the most popular ways to do so is by using *cosine similarity* which is computed by using the formula

$$cosinesimilarity(A, B) = \frac{A \cdot B}{||A|| * ||B||}$$

In the process of building the graph, we first had to normalize the vectors and computed the cosine similarity between all the vectors, afterward we computed the mean and standard deviation of each similarity vector. To connect two nodes, we need to define a threshold that will tell us which nodes are supposed to be connected and which are not. after some testing we found out that the best threshold is created by the given function

$$threshold_i = mean_i + 3 * standarddeviation_i$$

where $mean_i$ is the average of the i'th similarity vector and $standarddeviation_i$ is the standard deviation of the i'th similarity vector.

After we defined the threshold, we connected between two nodes i, j if

$$i \cdot j > threshold_i \wedge i \cdot j > threshold_j$$

The reason we constructed the edges that way is because we wanted the graph to be undirected and connect only the closest nodes.

4 Results

4.1 Parameters Optimization

To optimize the correlation between our graph and the language as much as possible, we need to find 2 main parameters, the *vector size* and the *window size*, when the vector size is the more fundamental one. The window size parameter is usually between 1 - 10, we checked this range of values and discovered that the optimal window size value is 5.

After we set the window size parameter, we can start to establish an accurate vector size. The optimal value for the window size we chose by observing the average degree and the standard deviation of each graph. As you can see in Figure 3, there is a logarithmic increase/decrease in the average degree / standard diversion respectively, In addition, we observed a divergence when reaching the size 300, thus we chose 300 as the vector size value. These findings of the values of the vector size and window size are matching the most common values.

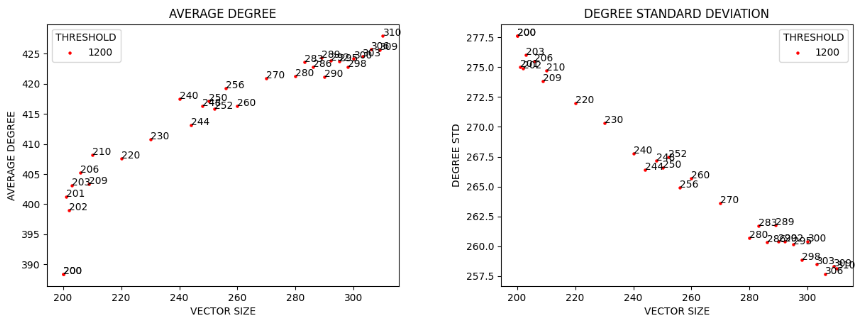


Figure 3: The average degree and the standard deviation of each vector size value with the proper threshold we defined

With those parameters, we can build graphs for different languages and compare them.

4.2 Comparison Results

4.2.1 Corpus Comparison

First, to be able to compare languages, we have to build graphs from several languages. The languages we have chosen are English, German, French, and Spanish. For each language, we obtained different results in the graph structure as described in the following table. As you can see in Table 1[1], each graph has different sizes both in terms of the number of nodes And in terms of the number of edges, the main reason for this is the size of the corpus. The larger the corpus, the larger the graph, but because our graphs are Approximately the same size (10^4 nodes and 10^6 edges) A comparison can be made. Compression Results

Language	Corpus Size	Number Of Nodes	Number Of Edges
English	19 GB	$4.8 * 10^4$	$10 * 10^6$
German	6 GB	$3.8 * 10^4$	$6 * 10^6$
French	5 GB	$3.0 * 10^4$	$4 * 10^6$
Spanish	4 GB	$2.6 * 10^4$	$3 * 10^6$

Table 1: The graphs and their basic features obtained from the languages.

4.2.2 Graph-Level Features Comparison

In this section, we provide a comparison of the Graph-Level features of our graphs. We will begin with the degree distribution. For each distribution, we tried to find the most fitted statistical distribution. In Figure 4 we present the degree distribution of each graph with the five most accurate statistical distributions (beta, gamma, chi, skew-normal and logistic).

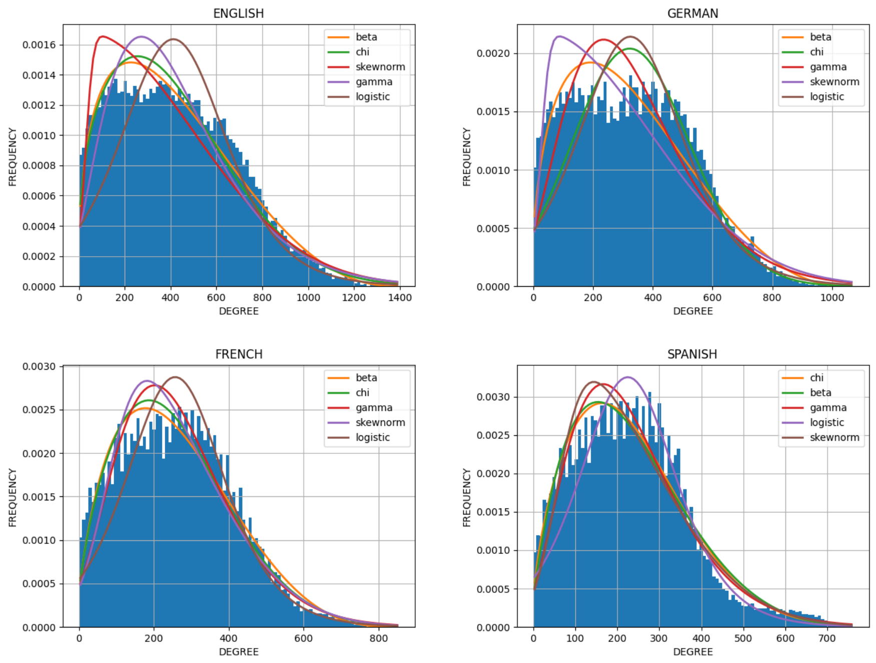


Figure 4: The degree distribution of each graph and the 5 most fitted statistical distributions (beta, gamma, chi, skew normal and logistic).

Furthermore, In Table 2 we present the parameters of the most suitable statistical distribution using the square mean error, $MSE = \frac{1}{n} * \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$.

[1]The reason the English corpus is considerably large relatively is that the corpus contains a great many unique words which don't affect the graph comparison due to their low-frequency appearance.

Language	Distribution	Alpha	Beta	Error Rate
English	Beta	1.46	3.38	$1.11 * 10^{-6}$
German	Beta	1.55	3.48	$4.12 * 10^{-6}$
French	Beta	1.85	4.12	$5.13 * 10^{-6}$
Spanish	Beta	2.061	6.242	$8.56 * 10^{-6}$

Table 2: The best statistical distribution, its parameters, and the error rate for each graph.

After computing and detecting the degree distribution for each graph, we now continue to other Graph-Level features which are the Diameter and Connected Components. In Table 3 we show the features results for each graph.

Language	Diameter	Connected Components	GCC percentage
English	7	44	0.999
German	7	30	0.999
French	7	22	0.999
Spanish	8	25	0.998

Table 3: The Graph-Level features for each graph.

4.2.3 Node-Level Features Comparison

In this section, we provide a comparison of the Node-Level features of our graphs. We will present the Centrality results first. The Centrality calculation is for each node individually, so to properly represent the data we will show the distribution (see Figure 5), Mean, Standard Deviation, Median, Min value, and the Max value for each graph as shown in Table 4.

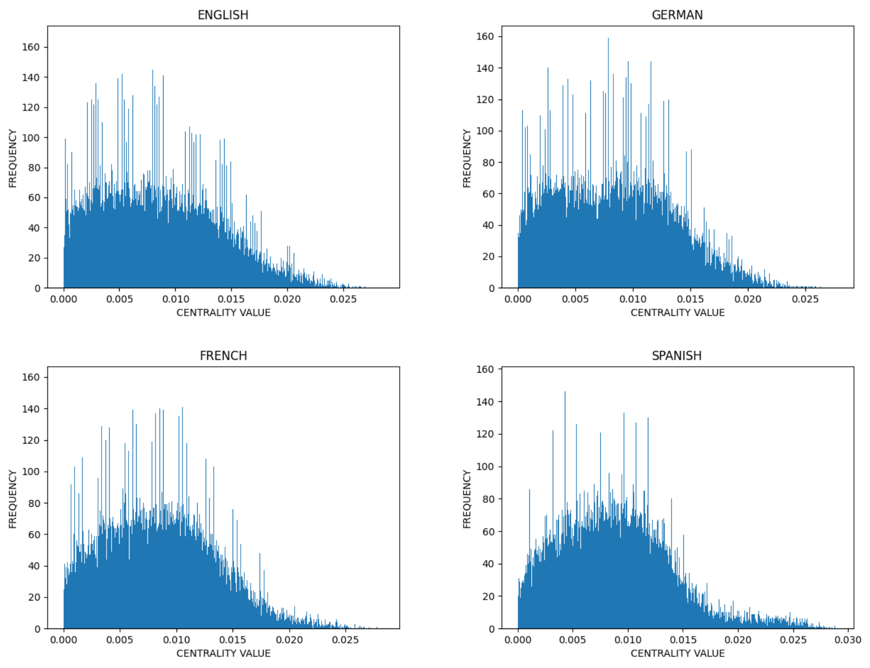


Figure 5: The centrality distribution for each graph.

After computing and representing the centrality for each graph, we now continue to other Node-Level features which is the Average Clustering Coefficient. In Table 5 we show the feature results for each graph.

Language	Mean	Standard Deviation	Median	Min Value	Max Value
English	0.0086	0.0.0053	0.0082	0	0.0285
German	0.0086	0.0051	0.0084	0	0.0278
French	0.0087	0.0049	0.0085	0	0.283
Spanish	0.0088	0.0052	0.0085	0	0.029

Table 4: The Centrality Mean, Standard Deviation, Median, Min value and the Max value for each graph.

Language	Average Clustering Coefficient
English	0.352
German	0.339
French	0.327
Spanish	0.317

Table 5: The Node-Level feature for each graph.

4.2.4 Edge-Level Features Comparison

In this section, we provide a comparison of the Edge-Level features of our graphs. In Table 6 we can see the Average Path length results for each graph.

Language	Average Path Length
English	2.759
German	2.811
French	2.807
Spanish	2.824

Table 6: The Edge-Level feature for each graph.

4.3 Clustering Of The Graph

First of all, we will present the definition of clustering.

Definition 4.1 (Clustering)**.** The process of dividing a set of input data into possibly overlapping, subsets, where elements in each subset are considered related by some similarity measure.

Performing clustering on graphs is further complex than 'normal' clustering since nodes have no vector representation. To obtain such a vector representation, there are several ways including the node2vec algorithm, Adjacency matrix, Laplacian matrix, etc. In our case, we chose to use the Adjacency matrix where each node will be represented as the corresponding row in the matrix. The clustering algorithm we chose is *K-Means*.

Definition 4.2 (K-Means Clustering)**.** The main concept of the K-means algorithm is to represent each cluster by the vector of mean attribute values of all training instances for numeric attributes and by the vector of modal (most frequent) values for nominal attributes that are assigned to that cluster. This cluster representation is called cluster center.

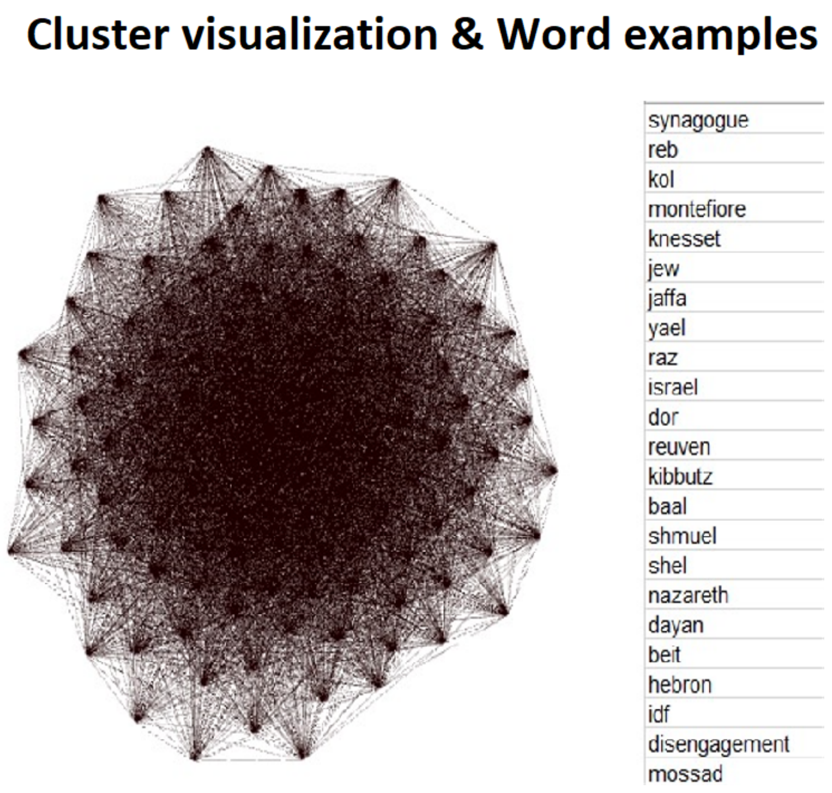
A major way of strengthening our core belief in the structure of the graph is by observing the clustering results[2], moreover, we found a strong pattern in each of our clusters which is that in each cluster all the words are under one category.

Before clustering the graphs, we first need to determine the number of clusters we will use. We wanted to find the number of categories that would fit the scale of our graphs, so we chose 1000 clusters when we expected each cluster to have about 200-500 words. After examining the results we saw that each cluster contained around 150 (scale of 10^2) words with one cluster containing the rest of the words (scale of 10^3). To exterminate the large

[2]In the report folder on the GitHub

cluster, we increased the number of categories. Unfortunately, we saw that this does not change the large cluster, but diminished the others. Therefore we concluded that the words that in the large cluster Are words that the algorithm does not know how to deal with and we can't do much about it.

To illustrate the results of the cluster, we present one of our clusters which can be seen in Figure 6. This sub-graph contains 116 nodes and 5646 edges with an average degree of 97.345, average path length 1.1, and diameter 2. all this information about the cluster indicating that it's well built.



synagogue
reb
kol
montefiore
knesset
jew
jaffa
yael
raz
israel
dor
reuven
kibbutz
baal
shmuel
shel
nazareth
dayan
beit
hebron
idf
disengagement
mossad

Figure 6: We can see a visualization of one of the clusters. This cluster category is Israel-related words which some of them you can see in the list.

5 Conclusion And Further Development

From a general look at all the features of the graphs, it can be seen that they are very similar. The degree distribution and centrality histograms are similar, In addition, the average path length and the GCC were similar to an error of 10^{-2} and 10^{-3} respectively and the graph diameter came out the same.

Moreover, the languages we used belong to the same category (analytic languages), and therefore the very fact that the features are similar shows us that the language structure is indeed similar - something that is not obvious because although the languages are analytic does not indicate that their construction is similar.

This project includes many different aspects since language is complex. unfortunately, from lack of time, we could not explore them all so there are things that are still interesting to check

- Find a corpus large enough for a non-analytic language and perform all the actions we have done for it.
- Use feature extraction methods that are more complex.
- Are there things that can be learned about language construction from clustering and graph features?
- Why there are 'jumps' in the degree distribution.

Those ideas could give us a deeper understanding of the languages and the differences between the languages.

References

[1] `https://www.cs.cmu.edu/~jingx/docs/DBreport.pdf`

[2] `https://www.sciencedirect.com/science/article/pii/S1574013707000020`

[3] `https://www.mediawiki.org/wiki/Download/pt`

[4] `https://www.researchgate.net/publication/3193784_A_SubGraph_Isomorphism_Algorithm_for_Matching_Large_Graphs`

[5] `https://networkx.org/documentation/stable/index.html`