Wikipedia Query Engine

Team 2 - EE-558 - A Network Tour of Data Science EPFL

EL Amrani Ayyoub Micheli Vincent Myotte Frédéric Sinnathamby Karthigan

Repository Link

Visualisation Link

Abstract

Wikipedia has become a state-of-the-art dataset when it comes to variety of data and completeness of information. The richness of Wikipedia encyclopedia makes its graph very connected and full of material that we can leverage from. By using its graph power, we propose a new data product for users to quickly access relevant articles based on a given query. In particular, we propose a Wikipedia query engine that is fast and intelligent based on Graph Machine Learning.

I. Introduction - Motivation

Graphs are a beautiful data structure to represent information that cannot be represented otherwise. Given that our data is extracted from Wikipedia pages, modeling it with graphs seems to be the best representation to get rich information and also rich visualizations. What's more, the essential component of our data product is the training of machine learning algorithms on our graph. Applying machine learning to graph data is an exciting field full of promise, it is interdisciplinary and can take advantage of the structure of the graphs to extract interesting features based on nodes attributes and edges structure.

Our idea is to propose to the user different interesting recommendations extracted from a bunch of special features that are implicitly hidden through the richness of our graph. In order to do that, the main challenge is to get interesting embeddings of our pages that could absorb most of the explanatory features. Through this report, we will walk you through our exploration and exploitation work.

After having constructed our graph, we first perform an exploration of its fundamental characteristics. We then do the exploitation work by constructing our model using different frameworks and compare the different results obtained. To be able to play with the designed models, and see query results dynamically on the graph, a visualization of the data product has been made, and is also available at this *link*.

We are here proposing a new way for users to quickly access relevant articles based on a given query. There already exists a query engine within Wikipedia, but this one seems to be rudimentary, as it is word-based: exact matching or proposition of articles comporting the queried words. We want an engine that is fast and intelligent, that takes power from not only the graph structure (pages connectivity) but in some case also the semantic similarity; all of this thanks to powerful embedding methods given by Graph ML.

II. Acquisition - Dataset

The initial dataset is obtained from the online tool Seealsology [9]. Given few seeds, it crawls the Wikipedia database and extracts articles as nodes and their "See also" sections as sets of edges.

We chose to use this tool to create our graph, as we wanted to start with a graph constructed in a meaningful way. Here, the links between nodes (that are the pages) should be relatively logical, as "See also" section propose related pages (in terms of content). We did not want to use the "full" graph from Wikipedia: for a given page, each link to another page creates an edge in the graph. That would have resulted in a very dense graph, with a lot of links that creates an unwanted divergence. For example, if a page mentions a location, then it would point to the corresponding country, that would point to a continent, and so on. Moreover, we would need far more computational power in that case. Therefore, Seealsology was an interesting tool to use to create the graph the way we wanted to.

We focus on *Data Science* related topics by scrapping every article at a distance of 2 from the following seeds: *Machine learning, Natural language processing, Artificial intelligence, Artificial neural network, Chatbot, Intelligent agent, Data visualization*.

After some trivial preprocessing (indexing the nodes), we have the node and edges of our graph.

As for the node attributes, we extracted the summary section of the article using the Wikipedia Python API [10]. We proceed as follows:

- Each summary goes through a traditional NLP pipeline consisting of tokenization, stop-words removal and lemmatization.
- We compute the TF-IDF matrix of this new corpus of documents.
- For each summary, we chose the top-10 keywords extracted from the matrix.
- For each article, we consider the concatenation of the key words and the article name as the attribute for each node.

At the end, we have one dataframe that gives the edge structure (source, target) and another the node structure (title, url and keywords of the page). You can see samples of both dataframes in *Appendix* on TABLE V and TABLE VI. With this, we have built the entire graph used in our product.

III. Exploration

We will now explore this graph and show a few fundamentals properties and analysis. For this, we have used the NetworkX [5] Python package. An interactive visualization of the graph is available at this *link*.

A. Graph properties

1) Connected components & diameter: First, our graph is fully connected, and that is important regarding the goal of our product, which is to provide recommendations that could interest the person that makes the queries. Indeed, having unused nodes is pointless, as it would never be used by the query engine. It is quite intuitive to have a fully connected graph, as we know that Wikipedia graphs usually form a ball, and we took related topics to generate the graph, so it would have been surprising to have more than one connected component.

Knowing that our recommendations should be relevant and precise, we would like to have a reasonable diameter in order to limit wideness. Here, our graph has a diameter of 9. This is convenient, given that our graph has 1166 nodes and 1439 edges, that means that our graph is not too wide, and that we should be able to distinguish some formations of groups, hopefully related in terms of semantic context.

2) Sparsity & degree distribution: As we can see in Figure 1, the graph is very sparse in both cases due to the small number of edges. Removing hubs makes it even sparser. This is consistent with the fact that as mentioned before, we used seeds in order to construct a graph in a spanning-tree fashion. That is why we should consider keeping the hubs, as the network is not too dense already, and every link is important because of the construction choice we made for the graph.

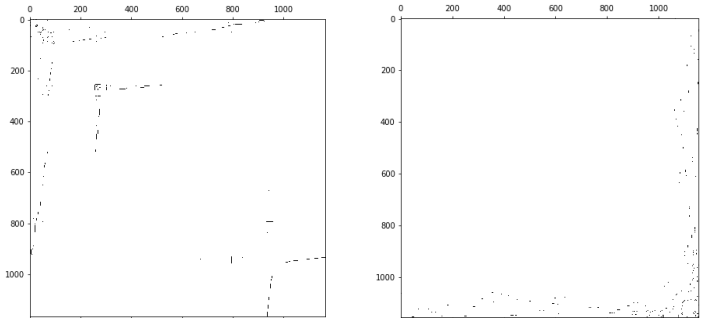


Fig. 1: Sparsity of the original graph and its pruned version. **Left**: Original graph - **Right**: Graph with top 10 hubs pruned.

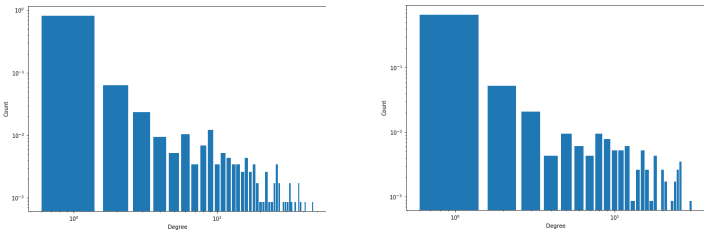


Fig. 2: Degree distributions of the original graph (**Left**) and its pruned version (**Right**).

In figure 2, we can see that by removing hubs, the degree distribution slightly changes, as it is removing nodes with the highest degree. Considering that the degree distribution is not significantly changing, and that most connected nodes are undoubtedly important in our network in terms of links creation between nodes, we should consider not to remove hubs.

After having studied the graph sparsity and its degree distribution with and without hubs, we have decided to work with the whole network. Furthermore, because of how we chose to construct the graph, every edge has an importance, and that is one more reason why it is not useful to remove hubs from our network.

B. Type of graph identification

After having looked at the log-log degree distribution of our graph (see figure 2), we have noticed that it has a linear decay, which means that the degree distribution probably follows a power-law. In order to confirm this observation, we tried to fit a network whose degree distribution follows a power-law to our graph, using Networkx "powerlaw_cluster_graph". By correctly setting the different parameters of Networkx function, we managed to generate a scale-free network that has the same number of nodes, roughly the same number of edges, and a close degree distribution. That strengthens our first guess that our network is scale-free.

C. Nodes properties

1) Clustering coefficient: The average clustering coefficient of our graph is 4.75%. That means that in average, nodes neighborhood is not fully connected, it is more likely that there are a few links between neighbor nodes. Considering the graph construction which gives "spanning-tree" like structure, it is consistent (as a given node would be linked to a couple of other nodes in a tree fashion, but nothing guarantees that linked nodes are themselves linked to each other) with the average clustering coefficient we have found.

On the other hand, the distribution of the clustering coefficient of our nodes (see figure 3) tells us that there are several fully-connected neighborhoods, even if most of the nodes have a clustering coefficient less than or equal to 4%. Nodes that have a fully-connected neighborhood could be the nodes we generate the network from (Seealsology seeds), or recurrent nodes that appear a lot in "See also" sections of pages used to generate the graph. That is a good point, as it means that we have some central topics in our graph, and that should help models that we use after to construct interesting nodes features to get a well-performing recommender system.

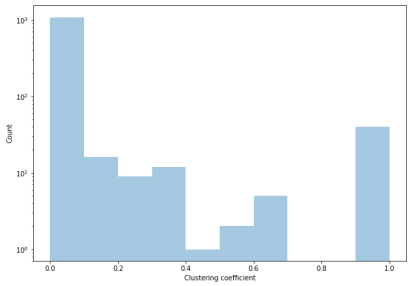


Fig. 3: Distribution of the clustering coefficient of nodes

node	centrality
artificial neural network	3.94 %
cognitive science	3.51%
analytics	3.26%
simulated reality	3.17%
universal basic income	3.17%

TABLE I: top 5 nodes with highest centrality

2) Centrality: Centrality is another interesting metric to identify the most important vertices in our network. As we can see on TABLE I, terms that have the highest centrality seems to be general central concepts of the pages we chose to generate the network from. On the other hand, as the highest centrality coefficients are quite reasonable, we can confirm that our graph is not only gravitating around a few very important nodes, but more distributed around a considerable number of important contexts, and that is what we need to make our product interesting.

3) Communities: The label propagation algorithm (from Networkx) allowed us to get the communities among the nodes of our network. These communities naturally have nodes that are strongly related to each other. One good metric that is worth

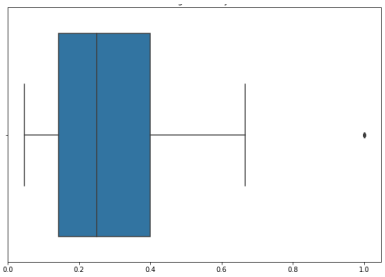


Fig. 4: Distribution of the average centrality of communities

looking at in this case is the average centrality among each community.

On the boxplot that shows the distribution of average centrality of communities (see figure 4), we can observe that we have strong relations between members of communities, as most communities have an average centrality that stands in a reasonable interval. That means that these communities were not only formed by a few links between each other, but that we have some groups of nodes that are particularly linked to each other.

D. Attributes Analysis

In this subsection, we perform an analysis of the nodes attributes of our graph, which are the keywords of the corresponding page. By regrouping all these words, we can study the frequency distribution and see the most frequent terms. We can visually represent the importance of the keywords based on their frequency distributions by using WordCloud (see figure 5 below).

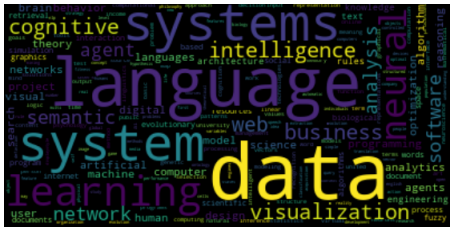


Fig. 5: WordCloud of the graph node attributes

Here, noticeable terms are ”data”, ”language”, ”system”, ”learning” and ”intelligence”, which are highly related to the seeds we chose to generate our graph. This is an expected result considering how we constructed our graph. At the same time, this concrete analysis is necessary keeping in mind the semantic similarity method we are going to use (cf FastText Walks IV-A3).

IV. Exploitation

The query engine is composed of two parts: models producing node embeddings and a recommendation algorithm.

A. Models - Node Embeddings

Three different models were used to learn low-dimensional continuous representations for nodes in the graph. Table II displays their respective hyperparameters.

1) Spectral Clustering: We establish a baseline based on the Spectral Clustering algorithm seen in the course. More precisely, we focus on the spectral embedding part.

- We compute the eigenvectors $u_1, u_2, ..., u_k$ associated with the k smallest eigenvalues of the normalized laplacian of our graph.
- We set $U_k = [u_1|u_2|...|u_k] \in \mathbb{R}^{N \times k}$
- Embed the i-th node to $y_i = \frac{U_k(i,:)^T}{||U_k(i,:)||^2}$

Model	Hyperparameters
Node2Vec	walk_length = 40, num_walks = 10 p = 1, q = 1 window = 5, size = 100, min_count = 5 sg = 1, negative = 10, iter = 20
Spectral	k = 100
FastText	walk_length = 40, num_walks = 10 p = 1, q = 1 source_weight = 0.75

TABLE II: Hyperparameters (set to library's default if not specified)

One should note that this approach is not scalable. With a larger dataset, the diagonalization of the Laplacian would become computationally very expensive.

2) Node2Vec: Node2Vec [3] follows a two-step process:

- Random walks: For each node, multiple biased random walks are generated. At every timestep, the transition probabilities depend on whether we want to emphasize homophily or structural equivalence. Under the homophily hypothesis, nodes belonging to similar communities should be embedded together. Under the structural equivalence hypothesis, nodes having similar structural roles should be embedded together. In this project we gave the same weight to both hypotheses.
- Word2Vec: The walks generated can be seen as corpus of nodes sequences. These sequences are then fed to a Skip-Gram Word2Vec model [4]. Ultimately node embeddings are derived from this model.

We used the implementation provided by the Node2Vec library [6].

Input:
An embedding map M. A node n_{source}.
A list of walks L starting from n_{source}.
Output: A new embedding for n_{source}.
buffer = empty_list()

for *walk in* L **do**
  source_contribution = source_weight $\times M(n_{source})$;
  walk_contribution = $\frac{1-source\_weight)}{length(walk)-1} \times \sum_{i=1}^{length(walk)} M(walk[i])$;
  new_embedding = source_contribution + walk_contribution;
  buffer.append(new_embedding);
end
return buffer.mean()

Algorithm 1: Walk averaged node embedding

3) fastText Walks: The previous algorithms do not make use of node attributes and that is why we designed this new approach in order to leverage graph structure and textual information at the same time. It relies on the fastText library [1] which was designed as a fast and lightweight alternative to Word2Vec with the ability to handle Out-of-Vocabulary forms. It is also known to generate better word embeddings for rare words.

- fastText embeddings: The fastText team published pretrained word vectors in many languages. In our case we resort to 1 million word vectors trained with subword information on Wikipedia 2017 and other sources [7]. We use these embeddings to map keywords to vector representations. Then for each article its keywords vectors are averaged into a single vector.

- Walk averaging: We make use of the previously computed node2vec random walks. That is, we enrich the fastText nodes representations based on the context in which they appear. The new embeddings are obtained by applying Algorithm 1 to every node in the graph.

Again this new representation depends on whether we want to emphasize homophily or structural equivalence. Besides, the $source\_weight$ parameter should be tuned based on the importance we want to give to the context of a node.

The interface with the pretrained embeddings was provided by the pyMagnitude library [8].

B. Recommendation Algorithm

The three models give us an embedding for each node. Now given a query we need to recommend pages to the user.

First we compute a query embedding. In the case of the fastText method, we take the mean of the fastText embeddings of each word present in the query. For the two other methods, we split the query on commas and look for the embedding of each splitted token if it corresponds to a page title present in the corpus. The comma is a choice of design of our product. The retrieved embeddings are then averaged.

Ultimately articles are recommended based on the cosine similarity between their respective embeddings and the query embedding.

One can notice that the Fasttext approach is more flexible than the other ones. Indeed it does not require page titles to be present in the query to make recommendations.

C. Evaluation & Results

1) Evaluation method: One quantitative way to evaluate our system would have been to perform link prediction. That is, we cut edges of the graph and we try to predict those links based on the given recommendations. However, this assumes that our product will not be better than the "See Also" section of Wikipedia pages, which our graph is built on. Eventually, this is not a good way to evaluate our product. We could have used Google Search results which is also the current search engine of Wikipedia. However, as said in the introduction, it is most likely word-based. Thus, after some thoughts and trials, we went for a qualitative method of evaluation.

We chose sets of pages from different topics and performed dimensionality reduction on their embeddings to visualize them. The idea was to analyze whether pages with similar topics would be located in the same regions of the embedding space. We used two dimensionality reduction methods seen and explained in the course: PCA and t-SNE. PCA is defined as an orthogonal linear transformation that maps the data to a new coordinate system so that the first coordinate captures the greatest amount of variance in the data (second greatest amount for second coordinate and so on). t-SNE is a non-linear technique minimizing the divergence between two distributions: one measuring pairwise similarities in the input space and another one measuring pairwise similarities of the corresponding low-dimensional points in the embedding space.

Another simpler way to evaluate the models is to look at the answers for different queries and to judge if they are good enough. Therefore, we propose two different types of results: clustering of central topics and answers to predefined queries.

Model	Result: cosine score
Node2Vec	machine learning: 0.99 machine learning in bioinformatics: 0.83 explanation-based learning: 0.77 one-shot learning: 0.75 model selection: 0.69 quantum machine learning: 0.69 quantum image: 0.68 hyperparameter optimization: 0.67 quantum annealing: 0.66 automated machine learning: 0.65
Spectral	machine learning: 0.99 machine learning in bioinformatics: 0.91 explanation-based learning: 0.83 one-shot learning: 0.55 automated machine learning: 0.42 gene expression programming: 0.37 quantum machine learning: 0.35 weak ai: 0.33 parallel distributed processing: 0.31 hyperparameter optimization: 0.31
FastText	automated machine learning: 0.86 machine learning: 0.86 quantum machine learning: 0.85 rule-based machine learning: 0.85 applications of machine learning: 0.84 machine learning in bioinformatics: 0.84 computer-assisted language learning: 0.82 never-ending language learning: 0.81 representation learning: 0.81 virtual world language learning: 0.81

TABLE III: Answers for the query: "machine learning"

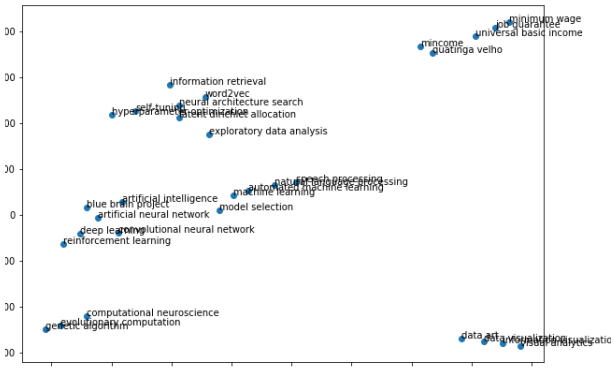
Model	Result: cosine score
Node2Vec	natural language processing: 0.77 artificial intelligence: 0.74 1 the road: 0.67 spoken dialogue system: 0.65 truecasing: 0.65 philosophy of artificial intelligence: 0.64 printing press check: 0.63 computer-assisted reviewing: 0.62 foreign language writing aid: 0.62 natural language user interface: 0.62
Spectral	natural language processing: 0.72 artificial intelligence: 0.69 1 the road: 0.53 automated essay scoring: 0.53 biomedical text mining: 0.53 language and communication technologies: 0.53 language technology: 0.53 spoken dialogue system: 0.53 transformer (machine learning model): 0.53 truecasing: 0.53
FastText	natural language processing: 0.92 natural-language processing: 0.90 philosophy of artificial intelligence: 0.89 marketing and artificial intelligence: 0.88 natural computation: 0.88 artificial development: 0.87 computational models of language acquisition: 0.87 existential risk from artificial general intelligence: 0.87 personality computing: 0.87

TABLE IV: Answers for the query: "artificial intelligence, natural language processing"

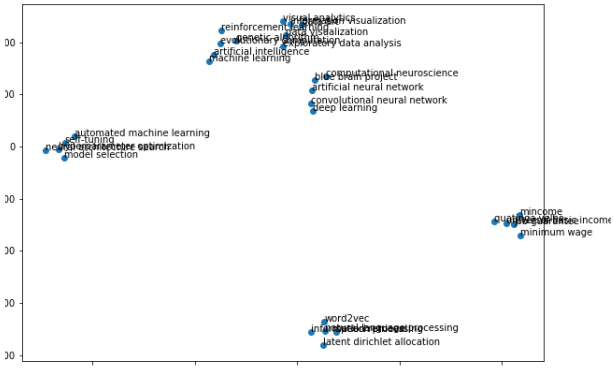
2) Results: Query results are shown in Table III and V. Other query results can be found in the *Appendix* in Tables X, XI and XII.

Figures 6 show the visualisations obtained with t-SNE for each embedding model. The PCA projection can be found in the *Appendix* in Figure 7.

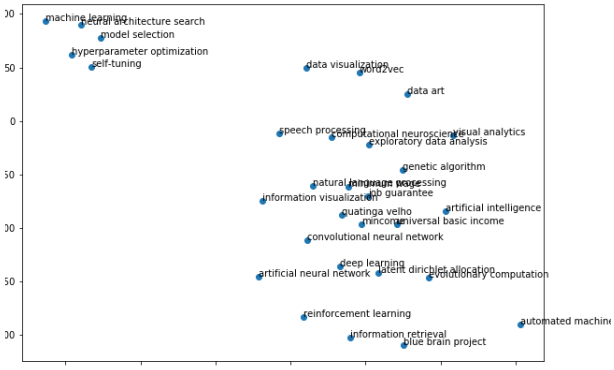
Overall t-SNE yields more defined clusters than PCA which goes to show the advantages of non-linear methods. It will be the technique of choice to visualize and compare the embeddings.



(a) FastText clusters



(b) Node2Vec clusters



(c) Spectral clusters

Fig. 6: Embeddings dimensionality reduction using t-SNE

The Spectral Clustering model fails to cluster similar pages together whereas the other two models map topics into different regions of the space.

Regarding queries, the three models give meaningful recommendations even though the baseline Spectral Clustering is still lagging behind. Moreover, the fastText method is the best approach when it comes to queries that do not mention page titles present in the corpus. In those instances, the model is able to capture the semantic content of the queried words and to recommend similar pages.

Overall fastText method seems to be the best which is to be expected as it takes into account both the graph structure and the semantic content of the pages. This is a nice illustration of the power of Graph ML.

D. Limitations

- Graph acquisition: As mentioned in part II, it is not a trivial task to obtain an insightful graph from Wikipedia. We believe that improving the graph acquisition procedure would drastically increase the quality of this system. Ultimately a query engine is just as good as the dataset it is built upon.
- User-friendliness: As mentioned in this part, the Spectral Clustering and Node2Vec models are not user-friendly since they require prior knowledge about article names. An area of improvement would be to implement a corrector module replacing Out-of-Corpus titles by the closest known titles.
- Scalibility: As mentioned previously, Spectral Clustering is not scalable in particular due to the Laplacian Computation. Even though, techniques exist to make the computation faster, the algorithm relies sensibly on the whole adjacency matrix. As such, for bigger datasets, Spectral Clustering would have been even less performant than the two other methods.

V. Data Product

We have published the query engine at this *link* along with an interactive visualisation of the graph. It has been implemented with Plotly and Dash technologies [2]. The code and the ReadMe for the product is also available in the *repository*.

VI. Conclusion

The proposed query engine seems to be powerful in extracting meaningful recommendations. It is due to the strength of the embeddings of the article nodes given by the models. The exploitation of the graph structure is undeniably important as all the methods are performing quite well. In particular, fastText walks seems to be very promising due to its use of semantic content as well. The graph power of Wikipedia along with some Machine Learning lead to strong a data product.

This query engine could be optimized even more. For instance, we have only used summary pages for the attributes of each node concerning the fastText method. A suggestion would be to think of more attributes in order to enrich our embeddings to reach even better performance and more desired recommendations for the user.

References

[1] Piotr Bojanowski et al. *Enriching Word Vectors with Subword Information*. 2016. arXiv: 1607.04606 [cs.CL].

[2] *Dash by Plotly*. URL: https://plot.ly/dash/.

[3] Aditya Grover and Jure Leskovec. "node2vec: Scalable feature learning for networks". In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2016, pp. 855–864.

[4] Tomas Mikolov et al. "Distributed representations of words and phrases and their compositionality". In: *Advances in neural information processing systems*. 2013, pp. 3111–3119.

[5] *NetworkX*. URL: https://networkx.github.io/documentation/stable/.

[6] *Node2Vec library*. URL: https://github.com/eliorc/node2vec.

[7] *Pretrained FastText Model*. URL: https://fasttext.cc/docs/en/english-vectors.html.

[8] *pyMagnitude library*. URL: https://github.com/plasticityai/magnitude.

[9] *Seealsology*. URL: https://densitydesign.github.io/strumentalia-seealsology/.

[10] *Wikipedia Python API*. URL: https://pypi.org/project/wikipedia/.

APPENDIX

source	target	depth
intelligent agent	software agent	1
intelligent agent	cognitive architecture	1
intelligent agent	cognitive radio	1
intelligent agent	cybernetics	1
intelligent agent	computer science	1

TABLE V: Source/Target table

source	url	keywords
intelligent agent	/wiki/Intelligent_agent	'intelligent', 'agents' ..
chatbot	/wiki/Chatbot	chatbots, apps..
artificial intelligence	/wiki/Artificial_intelligence	'ai', 'intelligence'..
weak ai	/wiki/Weak_AI	'ai', 'narrow', 'weak',..

TABLE VI: Node attributes examples

Model	Result: cosine score
Node2Vec	machine learning: 0.99 machine learning in bioinformatics: 0.83 explanation-based learning: 0.77 one-shot learning: 0.75 model selection: 0.69 quantum machine learning: 0.69 quantum image: 0.68 hyperparameter optimization: 0.67 quantum annealing: 0.66 automated machine learning: 0.65
Spectral	machine learning: 0.99 machine learning in bioinformatics: 0.91 explanation-based learning: 0.83 one-shot learning: 0.55 automated machine learning: 0.42 gene expression programming: 0.37 quantum machine learning: 0.35 weak ai: 0.33 parallel distributed processing: 0.31 hyperparameter optimization: 0.31
FastText	automated machine learning: 0.86 machine learning: 0.86 quantum machine learning: 0.85 rule-based machine learning: 0.85 machine learning in bioinformatics: 0.84 applications of machine learning: 0.84 computer-assisted language learning: 0.82 never-ending language learning: 0.81 representation learning: 0.81 virtual world language learning: 0.81

TABLE VII: Answers for the query: "machine learning"

Model	Result: cosine score
Node2Vec	natural language processing: 0.77 artificial intelligence: 0.74 1 the road: 0.67 spoken dialogue system: 0.65 truecasing: 0.65 philosophy of artificial intelligence: 0.64 printing press check: 0.63 computer-assisted reviewing: 0.62 foreign language writing aid: 0.62 natural language user interface: 0.62
Spectral	natural language processing: 0.72 artificial intelligence: 0.69 automated essay scoring: 0.53 biomedical text mining: 0.53 language and communication technologies: 0.53 language technology: 0.53 1 the road: 0.53 spoken dialogue system: 0.53 transformer (machine learning model): 0.53 truecasing: 0.53
FastText	natural language processing: 0.92 natural-language processing: 0.90 philosophy of artificial intelligence: 0.89 marketing and artificial intelligence: 0.88 natural computation: 0.88 artificial intelligence marketing: 0.88 artificial development: 0.87 personality computing: 0.87 computational models of language acquisition: 0.87 existential risk from artificial general intelligence: 0.87

TABLE VIII: Answers for the query: "artificial intelligence, natural language processing"

Model	Result: cosine score
Node2Vec	natural language processing: 0.77 artificial intelligence: 0.74 1 the road: 0.67 spoken dialogue system: 0.65 truecasing: 0.65 philosophy of artificial intelligence: 0.64 printing press check: 0.63 computer-assisted reviewing: 0.62 foreign language writing aid: 0.62 natural language user interface: 0.62
Spectral	natural language processing: 0.72 artificial intelligence: 0.69 automated essay scoring: 0.53 biomedical text mining: 0.53 1 the road: 0.53 language and communication technologies: 0.53 language technology: 0.53 spoken dialogue system: 0.53 transformer (machine learning model): 0.53 truecasing: 0.53
FastText	natural language processing: 0.92 natural-language processing: 0.90 philosophy of artificial intelligence: 0.89 marketing and artificial intelligence: 0.88 artificial intelligence marketing: 0.88 natural computation: 0.88 personality computing: 0.87 computational models of language acquisition: 0.87 artificial development: 0.87 existential risk from artificial general intelligence: 0.87

TABLE IX: Answers for the query: "artificial intelligence, natural language processing"

Model	Result: cosine score
Node2Vec	social simulation: 0.89 intelligent agent: 0.88 journal of artificial societies and social simulation: 0.77 artificial reality: 0.76 artificial society: 0.75 synthetic environment for analysis and simulations: 0.75 virtual reality: 0.75 multiple-agent system: 0.74 agent-based computational economics: 0.74 fuzzy agent: 0.73
Spectral	social simulation: 0.77 intelligent agent: 0.70 artificial reality: 0.64 synthetic environment for analysis and simulations: 0.64 system dynamics: 0.64 cliodynamics: 0.64 journal of artificial societies and social simulation: 0.64 virtual reality: 0.64 fuzzy agent: 0.55 peas: 0.55
FastText	agent-based social simulation': 0.90 distributed multi-agent reasoning system': 0.86 fuzzy agent': 0.86 intelligent personal assistant': 0.86 internet relay chat bot': 0.86 intelligent agent': 0.85 interactive online characters': 0.85 multi-agent system': 0.85 social bot': 0.85 software agent': 0.85

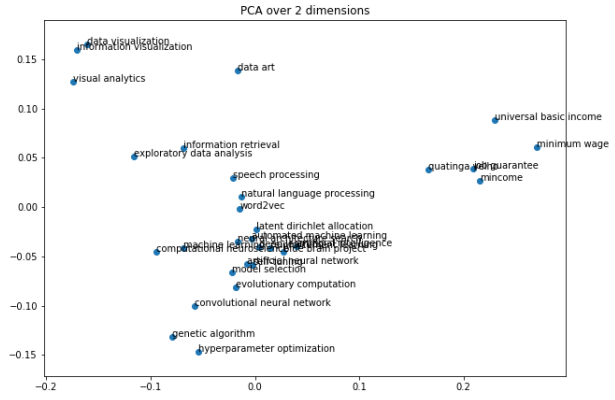
TABLE X: Answers for the query: "social simulation, intelligent agent, chat bot"

Model	Result: cosine score
Node2Vec	data visualization: 0.99 data warehouse: 0.76 data science: 0.73 climate change art: 0.71 statistical graphics: 0.71 information visualization: 0.70 craftivism: 0.69 interaction techniques: 0.69 statistical analysis: 0.69 warming stripes: 0.69
Spectral	data visualization: 0.99 data warehouse: 0.86 data profiling: 0.24 data science: 0.48 climate change art: 0.38 visual journalism: 0.29 statistical graphics: 0.28 warming stripes: 0.27 statistical analysis: 0.26 artificial neural network: 0.26
FastText	data visualization: 0.90 data presentation architecture: 0.88 information visualization: 0.88 visualization (graphic): 0.88 software visualization: 0.87 paraview: 0.86 interactive visual analysis: 0.85 scientific visualization: 0.85 sonargraph: 0.85 sourcetrail: 0.85

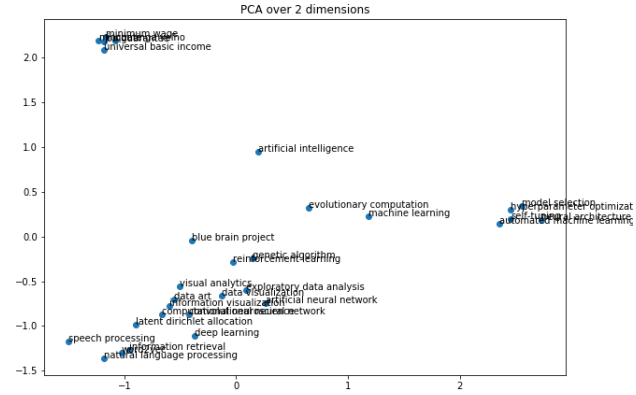
TABLE XI: Answers for the query: "data visualization"

Model	Result: cosine score
Node2Vec	None (not in the corpus)
Spectral	None (not in the corpus)
FastText	cash transfers: 0.74 redistribution of income and wealth: 0.67 revenue shortfall: 0.66 negative income tax: 0.65 universal credit: 0.65 fairtax: 0.64 post-scarcity economy: 0.64 consumer demand tests (animals): 0.63 guaranteed minimum income: 0.63 working time: 0.63

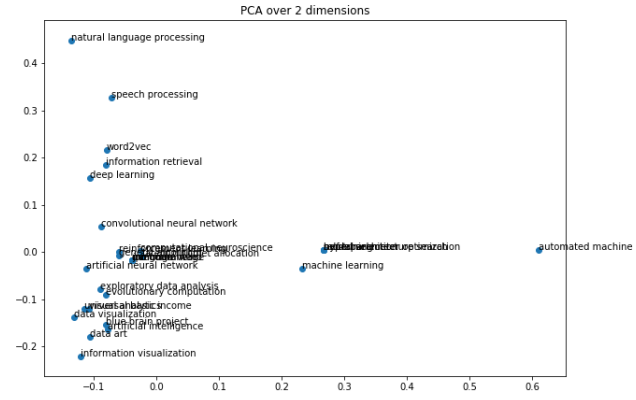
TABLE XII: Answers for the query: "money money money"



(a) FastText clusters



(b) Node2Vec clusters



(c) Spectral clusters

Fig. 7: Embeddings dimensionality reduction using PCA