Word embeddings and transformer models for optimal learning
CS433 - Machine Learning

Romain Gros, Antony Doukhan, Dora Lourenço
EPFL - Lausanne, Switzerland
Supervisor: Maxime Gabella, Magma Learning

***Abstract*—The recent pandemic has encouraged teachers and student to develop new tools to facilitate learning. Showcased at The Startup Champions Seed Night of 2020, Magma Learning proposes different tools to personally assist the user in the learning procedure. One of them is the vector embedding of topics and their visualisation in a colored 3 dimensional space.**

I. Introduction

When used for a particular task, Natural Language Processing algorithm perform better when trained on a predefined corpus. Given a list of topics that we want to embed in a high dimensional vector space, we present methods to optimize the creation of the corpus as well as the NLP model.

II. Article Set Generation

A. Collection of Wikipedia Dumps

One first improvement for the corpus creation is to replace the use of the Wikipedia API with the Wikipedia dumps. Wikipedia dumps are, in most cases, XML files that contain Wikipedia information. By using downloaded files instead of API calls, the corpus creation should theoretically be much faster. To obtain the dumps we access Wikipedia Dumps. The list of mirror servers that contain the dumps are available at Mirrors. The mirror we are using is https://ftp.acc.umu.se/mirror/wikimedia.org/. The files downloaded for our project are the ones found in the folder `dumps/enwiki/20201101/`. Articles in this folder were collected on the 1st of November 2020, the latest version at that time. These files follow the format "enwiki-20201101-pages-articles-multistream*number1*.xml-p*number2*p*number3*.bz2". One file, `enwiki-20201101-pages-articles-multistream.xml.bz2` contains all the articles and takes up 18Gb of space. The files we collected have each one a set of articles of the bigger file. There are two options: download all the smaller files or download the bigger one. We choose to download the smaller files. The number of articles we need to create the corpus is a very small percentage of the total number of articles. We only want around 6700 articles. Therefore, working with the files as they are would be inefficient in terms of time. We would need to iterate over a huge number of articles that we would never use. Because of this, we choose to use a script to extract only the articles we want and write them to XML files that can be parsed. If we run the script for the bigger article, we run into problems. The script needs to parse the content and, because of this, the file needs to be uncompressed. The bigger file takes up 75GB of memory (uncompressed). The RAM in a regular computer (usually 8GB) in not enough to run our script with the bigger file. Also, this process could not be parallelized. For these reasons, we choose to download the smaller files. The biggest ones take up 2GB of memory (uncompressed). We have in total 59 dump files downloaded. To create the new files, which only contain the articles we want, we run the script for each of the downloaded files. To parallelize the process, we run this in a cluster. We use 59 computers and, in each one, the script is run once for the respective file. It takes around 2h30 to run. If done sequentially, it takes up to 67h to run. The script is the file `parse_dumps.py` present in the Github repository. After the new files are created, they occupy a total of around 250MB, much smaller than the original 75GB. To read the articles from this files there are several Python libraries at our disposal. From the ones that can be found online, only two seemed useful in our opinion: gensim.corpora and wiki-dump-reader. We choose to use wiki-dump-reader, as gensim.corpora only allows to obtain cleaned text. When we go to a Wikipedia page and click on the option "Edit", we can see a version of the text that is not cleaned. It contains useful extra indications that are not showed directly in the article page. It allows, as an example, to find

	Time to process 5 articles
Wikipedia API	28.17s
Wikipedia dump	10.73s

Table I: *Speed comparison between Wikipedia API and Wikipedia dump*

synonyms of the topic. The new extracted files can be downloaded from Drive articles. After obtaining the new files, we need to test if using Wikipedia dumps is, in fact, faster than using the Wikipedia API. To test this, we create two cells, one with the original code to create the corpus and another with the same code, but using Wikipedia dumps. We ran the function 10 times with 5 articles and calculated the mean for each. Using Wikipedia dump leads to slower computation time than using API (Table I). None of the articles where disambiguation pages or duplicates.

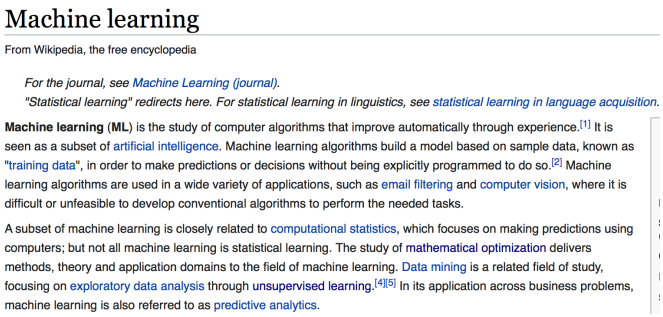
B. Backlinks

To improve our corpus, we also use backlinks. Backlinks are the links of the articles that have a link to the current article. In Wikipedia, these can be found in *What links here* for each article. The idea is to use backlinks to collect more articles, and therefore, increase the number of documents in the corpus. Unfortunately, there are no dumps for backlinks, so we had to resort to the Wikipedia API. The documentation of the backlinks API call can be found at API:Backlinks. We use this API to collect all the titles of backlinks for our topics. In the following sentences, we refer to the title of a backlink just as a backlink. After the collection of backlinks, we calculate the frequency of each backlink. The most useful ones are the ones with the highest frequency, i.e. the ones that relate the biggest number of topics. We then exclude the articles that start with "Outline of", "List of", "Index of", "Glossary of", "Lists of" and "Very Short Introductions", because these are lists in alphabetical order. Articles like "2013 in science" are also excluded, as they state a list of several events (usually each one taking just a few lines) by chronological order, that have no correlation with the events that antecede and precede them. The articles starting with "History of" are not a list of events like "2013 in science". So we don't exclude them. Afterwards, we select the 2000 most frequent backlinks. From these backlinks, we only keep the ones that we don't already have. We obtain a list of around 825 titles to use to collect articles using the script. The titles are stored in the file `backlinks.pickle`, retrievable in the same Google Drive.

III. Data Pre-processing

A. Addition of Acronyms and Synonyms

To optimize the creation of the corpus, we implement the addition of synonyms and acronyms denoting the topics. The use of an acronyms of synonym is redudant on the same wikipedia page and tracking those is essential for learning the context of words. The use of `XML` wikipedia dumps allows to retrieve all the page names (synonyms) redirecting to the topic considered, as well as acronyms and synonyms appearing at the begining of an article in bold. For example, looking at Machine Learning page (Figure 1), the head states *"Statistical Learning" redirects here* and the first lines of the introductory paragraph lists 'ML' written in bold. The expressions found are added to the array associated to a topic name. The acronyms are stored in lower case. Thus to avoid confusion and false positive presence, we drop the acronyms if they are part of the top 1000 most frequent words of US dictionary.


Machine learning

From Wikipedia, the free encyclopedia

For the journal, see Machine Learning (journal).
"Statistical learning" redirects here. For statistical learning in linguistics, see statistical learning in language acquisition.

Machine learning (**ML**) is the study of computer algorithms that improve automatically through experience.[1] It is seen as a subset of artificial intelligence. Machine learning algorithms build a model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to do so.[2] Machine learning algorithms are used in a wide variety of applications, such as email filtering and computer vision, where it is difficult or unfeasible to develop conventional algorithms to perform the needed tasks.

A subset of machine learning is closely related to computational statistics, which focuses on making predictions using computers; but not all machine learning is statistical learning. The study of mathematical optimization delivers methods, theory and application domains to the field of machine learning. Data mining is a related field of study, focusing on exploratory data analysis through unsupervised learning.[4][5] In its application across business problems, machine learning is also referred to as predictive analytics.

Figure 1: *Wikipedia Machine Learning page header*

B. Stemming

Next, we perform (or not) a stemming step, using *Porter's algorithm*, before searching for the words in the different articles. Although this approach may add some noise to our data, the information that is gained by performing may allow to make models of a better quality. In addition, pluralization (and lemmatization) of the topics are no more to be considered. Note that with this stemming step, we remove 297 acronyms and synonyms to the list, since they have the same stemmed expression as one of the original topics. To understand if the model is indeed better, we will optimize the models using both the stemmed and the unstemmed version of the corpus and search for the best one.

C. Final topics list

The final stemmed topic list contains 8937 "words" that will be searched for in articles during corpus generation. Among these 8937 words, 6769 corresponds to the original topics that we found in `topics_epfl.txt`". The 2168 remaining words are either synonyms or acronyms of these original words. The final unstemmed contains topic list contains 9270 "words". Less words are removed due to the absence of stemming.

IV. Corpus generation

A. Disambiguation Pages

Some of the articles extracted are disambiguation pages. Disambiguation pages are Wikipedia articles that contain a list of possible disambiguations for the topic. Useless for the corpus creation, they can be ignored. In the dumps, we can keep track of this property if no related category is mentioned in the page.

B. Duplicates

The script selects the articles by comparing the titles in lowercase. This leads to obtaining more articles than we want. For example, we obtain the articles with titles Microscope, MicroScope and MICROSCOPE. Out of these 3, we are only interested in one. The others we call duplicates. What we do to solve this problem is finding all the articles that have duplicates and create an array with the duplicates' titles. This array is stored in the pickle `duplicates_to_ignore.pickle`. The array is then used in the function for the creation of the corpus to skip these articles that we are not interested in.

C. Parallelization

In order to make the corpus creation faster, the dump files are split between different processes running in parallel. The function passed to create the corpus is CPU intensive. Because of this, the time spent writing in the shared variable, i.e adding the generated document to the corpus, is very small compared to the time spent on the rest of the function. Therefore, this parallelization results in a reduction of the total running time of the corpus creation. The number of processes created is the total size of the files divided by the size of the biggest file. In our case, we use 3 processes to compute the corpus. Theoretically, the running time is around 3 times shorter, assuming a multi-core CPU is used.

D. Disambiguation

Some of the `topics_epfl.txt` topics contain parenthesis, that are used for disambiguation. As an example, the topic *Work* is present three times, corresponding to three different contexts (*Thermodynamics*, *Physics* and *Project Management*). During the loop, if one of the topics is in this case, we apply an algorithm to perform disambiguation. First, we search for all the possible word in parenthesis in the text. If at least one of them is found, we take the one that is the most present and add it to the word. If none of the words are found, then the topic is not added to the corpus. For our example, if the word *Thermodynamics* is mentioned five times in an article in which *Work* is found, then it's likely that the use *Work* was related to *Thermodynamics*.

E. Final corpus

The final versions of the corpus (unstemmed version and stemmed version) are divided in 3 different files, for each version, that can be downloaded at the same Google Drive (`corpus` folder). The words are stored as numbers and can be mapped back to words using the function `read_topics_corpus`. They are built from 50% of the total amount of articles for the stemmed corpus (25% for the unstemmed corpus), due to computational power limitations. However, the same code can be ran to build the corpus for all the articles using a more powerful computer (with 16GB RAM memory).

V. Model training

For this project, we decided to use and compare three commonly used natural language processing (NLP) algorithms:

1) CBOW Word2vec (Mikolov et al., 2013) : the embedding is based on local interpolation from neighboring words (we used gensim implementation of Word2Vec).
2) Glove (Pennington et al., 2014) : based on global word to word co-occurence counts over the corpus (GloVe implementation).
3) FastText (Bojanowski et al., 2017) : treats each word as composition of n-grams, and makes an embedding based on the latter (we used gensim implementation of FastText).

All the models are trained using previously generated corpus, and optimized following a procedure described in the next section.

Model name	Parameters to be optimized	Values
Word2Vec	min count	2, 4
	window	2, 5, 10
	Stemming	True, False
GloVe	window	2, 5, 10
	no components	50, 100
	learning rate	0.05, 0.01, 0.005
	Stemming	True, False
FastText	embedding size	10, 20, 50
	window size	2, 5, 10
	Stemming	True, False

Table II: *List of parameters to be optimized*

Model name	Best parameters		Correlation
Word2Vec	min count	2	0.41
	window	2	
	stemming	False	
GloVe	window	5	0.295
	no components	50	
	learning rate	0.005	
	stemming	True	
FastText	embedding size	20	0.292
	window size	5	
	stemming	True	

Table III: *Optimization results*

VI. Objective Function Analysis

A. Description of our approach

To select our model and optimize its parameters, we chose to use a word similarity evaluation, that has been widely used in the literature [1] [2]. The human ability to judge relatedness between different topics can be used as a gold standard when comparing different models. At first, we used some of the online available datasets described in Wang et al. [2] for this task (WS-353, MEN-LEM, EN-RW and EN-TRUK). Unfortunately, we could not get more than 73 matches between our topic list and one of the dataset. Also, the pairs appearing both in our vocabulary and the dataset did not mention scientific topics which are highly represented in our corpus. For these reasons, we decided to create our own dataset. To do so, we randomly generate 102 pairs of topics, and each of us assigns a similarity grade on a scale 0-10. Although each grade is very subjective, the mean of our grades can be considered, to some extent, as measurement of the similarity of two topics. Correlation between our grades and similarity obtained from the model is then computed. The goal is then to optimize this similarity measurement, using a grid search approach. Few parameters (see Table II below), are optimized for each model. Due to computational power and RAM memory limitations, we use 20% of the articles. The final model can only be ran with a computer owing at least 16GB of RAM. Google Colab is not powerful enough for that task, as it only proposes 12GB of RAM.

B. Results and model selection

Obtained results (the best correlation obtained for each model), are indicated in Table III. According to these, one should chose the best model, i.e. Word2Vec with $window = 2$, $mincount = 2$ and $stemming = False$, yielding the higher correlation values for further visualization. Note that different values for the parameters might be found when running for the full corpus.

Well, Software, Europe, Analysis, Availability
Time, Property, Text mining, Law
Science, System, Population, World

Table IV: *12 most similar words to* `Computer`

VII. Visualization

We apply PCA (eventually followed by t-SNE) to the matrix obtained after Word2Vec training in order to obtain data suitable for 3D (or 2D) representation. This interactive representation can be found in the Github repository under the name `Learnet.html`. One can navigate between the topics and see similar topics being located close to each other (to some extent). Note that this visualization should be way more relevant if trained on the full Wikipedia corpus (and not only on 20% of the articles). Another example of visualization is the list of the 12 most similar topics to `Computer` and `Operation (Mathematics)`, which are represented in Table IV and Table V. One can observe that most of the topics present in the list above are indeed related to the `Computer Science` and the `Mathematics` world, respectively.

Ring (mathematics), Group (mathematics), Change (mathematics)
Set (psychology), Value (economics), Point (geometry)
Matrix (mathematics), Expression (mathematics), Measure (mathematics)
Reaction (physics), Variable (mathematics), Equation

Table V: *12 most similar words to* `Operation (mathematics)`

VIII. Conclusion and perspectives

We managed to optimize the corpus creation at two levels. First by sharpening the research of words throughout the Wikipedia articles : adding acronyms, synonyms (and eventually) stemming. All this methods have to be carefully used, as they may induces bias and repeated error during the creation if not controlled. Secondly, we reduced the time needed for the construction of our corpus by using Wikipedia dumps data base, and splitting the job between processors. In addition, we compared three NLP models performed a hyper-parameter tuning with respect to a well-suited objective function that we developed. Of course, the dataset generated should be improved if this project should become a commercial product. Another notable aspect that we did not mention in this report is the opportunity to use a Network structure based on number of mentions of topics, hyperlinks connecting the articles, etc. We believe that this intrinsic property of the corpus that is created can be used for further applications, in particular dimensionality reduction. We would like to thank Maxime Gabella, CEO of Magma learning, for the opportunity of working on an interesting applied problem, and we hope that the improvements made will help Magma Learning develop further its personal AI assistant.

References

[1] David Milne. “Computing Semantic Relatedness using Wikipedia Link Structure”. In: (Jan. 2009).

[2] Bin Wang et al. “Evaluating word embedding models: methods and experimental results”. In: *APSIPA Transactions on Signal and Information Processing* 8 (2019), e19. DOI: 10.1017/ATSIP.2019.12.