UNIVERSIDAD POLITÉCNICA DE MADRID

Profile Based Content Retrieval

Antón Aba Varela, anton.abav@alumnos.upm.es

Index

Introduction

Motivation

Nowadays, news outlets cover a wide range of disciplines all throughout the world and readers can select between the largest volume that has ever been seen. For instance, whereas our grandparents' repertoire may have been limited to the local or national radios and newspapers, we have the chance to access the almost unlimited content located in the worldwide web.

One of the few problems derived from this volume increase is the increasing problem of finding the news that you actually want to receive. I myself am a person who enjoys watching and reading multiple news outlets concerning many sports, such as basketball, but I am not interested in each team's nor in every player's news.

That is the main problem for media apps or similar news providers, they want to feed the user with the most relevant articles in order to maximize the time spent on the platform and the probability of coming back to that source. That is why most of them, if not all, have developed recommendation systems based on the user's profile to deliver the appropriate news snippets.

The recommendation is usually based on the settings chosen by each user and relies on very specific terms. For each topic of your interest, you are able to choose the sport and then the particulars, such as the division or league, the team(s) and the player(s); sometimes combining terms allowing to maximize the search engine.

For example, if a user likes "football", "Cristiano" he may be interested in all news regarding Cristiano Ronaldo; but if he adds "Champions" he may be more concerned the articles about the player's performance in such competition. Therefore, the articles shown should take into consideration this and deliver the appropriate snippets.

Datasets used

In order to develop and test the validity of the program I have selected the dataset created by (Yara Rizk, 2018). It contains the raw text of 1000 sports articles, all of them written in English, which we will be using in order to develop our model, as it is large enough to be considered a significant sample.

The profiles' information will be generated by me, as the proposed structure would consist of a simple solution which replicates the example provided two paragraphs above. This why multiple combinations can be tried in order to prove the validity of the code and we do not have to rely on one specific source.

Design

Preprocessing

In order to construct the model for our profile base retrieval we will need to preprocess the dataset. The goal will be to remove from each article the unsignificant words, keeping only those which provide an added meaning. For instance, from the sentence “I want a piece of cheese.” would remain something along the lines of “want piece cheese” which encapsules the whole meaning. Additionally, special characters such as numbers or punctuation should be removed from the raw text.

Furthermore, we would like to make sure that words that come from the same root are understood by the program as the same term. Given the following words: “produces”, “production”, “produce” they all come from the same origin “produc-” and it would be wrong to understand them as if this was not the case. Therefore, we should try to stem or lemmatize each token present after the removal of unsignificant terms.

Embedding model

First, I considered creating a simple frequency-based model such as TF-IDF weighting because it can reliably represent the importance of each token on each document relativized with respect to the appearances in other documents. Although it is feasible, I found 2 very concerning problem with this approach.

TF-IDF weighting simply measures the words that appear and does not understand the order or context around them. That could be solved combining it with a Bag-of-Words so it can learn which words tend to be around others, but the result is largely inefficient since it relies of a high-dimensional feature vector which is mostly zeros due to the size of the vocabulary. In conclusion, it solves one problem by creating another one and our second issue remains unsolved.

The meaning of the whole article and correlations between words remain undetected. For instance, if I were interested on tennis and the Roland Garros finals were being played, I would very much want to obtain news regarding this event. However, if the word tennis did not appear on the articles talking about the match then it will not be delivered to me, even though any human would easily say that I would be very interested on the snippet. (InterviewBuddy, 2019)

Therefore, I focused on the possible deep learning designs seen in class such as the Word2Vec model. It is one of the most effective techniques at representing words, as it is highly efficient at learning word embeddings, using Continuous BOW or Skipgram architectures. In one hand it solved all of the problems from the previous design, since it would not have the same error as the example given above and computationally it is much better.

However, I believed that this could not be my final choice since Word2Vec was capturing the information for every word and my actual goal was capturing the information for every document. Therefore, I was clearly getting closer to my goal but not quite there, I still had to deal on how to use the embeddings independently from the size of the query vector and all of the news articles.

That was solved when I found Doc2Vec which was built on top of word2vec. It proposes a paragraph vector, such as an unsupervised algorithm that learns fixed-length feature representations from variable length documents. It takes into consideration the ordering of words within a narrow context, similar to an n-gram model. The combined result generalizes and has a lower dimensionality but still is of a fixed length, ending up with a much simpler and

more generalized vector, reducing the necessary memory space and complexity tremendously when querying the database (Ayari, 2020).

The proposed solution was creating the model based on this architecture, which carries another extra benefit. As more news articles are written, they can be both quickly transformed by the model into their embeddings and used to make the model keep learning and improving itself continuously.

Querying

The database of news articles has already been encoded and saved, which means that the only task left is to encode the profiles in order to properly query the results. For that matter, we will be first applying the same preprocessing as it was done with the corpus of news articles in order to keep the consistency. It does not seem reasonable to keep both the user's actual profile and the preprocessed one in memory all the time as the computation is done almost instantly.

Then, for each topic of interest we will transform it using our Doc2Vec model, which should not be kept in memory for the same reasons as the ones explained above. Using that vectorized input we will retrieve the positions of the most similar documents located in the database of articles.

Delivery

Using these positions, we will be retrieving the original documents from the database, not the preprocessed ones. Bear in mind that it is important to do so because there is no reader who would want to read and article with all "meaningless" words removed and the remaining ones reduced to its root, that would be real a nuisance.

The user will receive the top N snippets for each of his interests and next to it the similarity percentage, in a similar fashion as Netflix does.

Implementation

Preprocessing

In order to start the program, the executor could provide the location of the news corpus in order to be preprocessed. That is necessary in order to have in memory the whole dataset for the delivery of the original articles. The next steps will be described below, however, it is not entirely necessary to run them since the model containing the embeddings of the articles has been saved and will be provided along this document.

First, in order to standardize the documents, we will be keeping only those terms composed simply by letters (which may be lower or capital case). This will successfully be removing all special characters and numbers using the *re* package and its *sub()* method using the regression formula provided in other notebooks. Then we will be removing all whitespaces applying *strip*() and convert the whole text to lower case using the *lower*() function, both from the string package.

Then we can begin treating each of the remaining words within the document and for that matter we will be using the *nltk* package. We will instantiate a *WordPunctTokenizer()* to split the document in the different tokens keeping only those whose length is greater than 2, as those are in almost all cases very frequent throughout texts and meaningless. Moreover, the token must not belong to the list of English stop words compiled in the *stopwords* list provided by the package.

Then, if it passes the filters, the token will be stemmed using the instance of the *PorterStemmer()* and reevaluate whether or not the remaining string belongs to the stopwords list. This was decided because while adjusting the code it was detected that terms, such as “she’s”, were passing the filter although they clearly belonged to the stopwords list. For each document, a list of the remaining and preprocessed tokens is saved, creating a list of lists to keep them all.

Embedding model

Again, it is not entirely necessary to run it since the model containing the embeddings of the articles has been saved and will be provided. We will be using the *gensim* package which in its module *models.doc2vec* you can find the methods *TaggedDocument* and *Doc2Vec*, which will allow us to process the corpus.

First, I have used a compression list to tag the whole preprocessed corpus so that it can be fed to the Doc2Vec model and keeping the order from the original dataset. Secondly, we will create a model with the Doc2Vec method and adjusting some of its parameters, mainly the size of the embedding vector, the window size for the bag of words to learn the context around the tokens.

Using a double for-loop with values of the vector size being 25, 50, 100 or 150 and the window size being 1, 2, 3, 4 and 5. It was determined that the most accurate results were obtained while using a vector of 100 dimensions and a window which looked 3 places to the right and left for each token.

In order to remove extremely rare tokens, the *min_count* parameter was set as 2, meaning that a term must appear at least this number of times in the corpus in order to not be ignored. The number of epochs was stablished at 100 since the results were more than sufficient, although it can be trained for as longer as it is desired.

Then we build the necessary vocabulary for training the model, using the tagged corpus and the function *build_vocab()*. Once this is completed, the model was trained using the tagged corpus as source, stablishing the number of sentences with the *total_examples* parameter as the attribute *corpus_count* from the current model and the number of epochs that were previously specified.

After the training is completed, we will be saving the model in order to not need it to recompute all this processing again, since it has transformed each of the articles in the corpus to its respective embedding. If we wanted to add new texts, we will be needing to append it to the original corpus and feed it to the model, which can simply transform it or also use it to learn a bit more.

Querying

I created the following profiles as a validation set, although any other imaginable options can be tried. It is formed by 2 basic cases and a complex one

"Antonio": [["baseball"]]. Which means that his only interest is in Baseball, the sport.

"Javier": [["cristiano"]]. He is only interested in Cristiano Ronaldo.

"Pablo": [["basketball","nba","lebron"], ["tennis"]]. Pablo is interested in the sport of basketball, more particularly in the NBA and specially on Lebron James. He is oalso interested in tennis.

First, we will be loading the saved model and then for each user and for each of his interest we will be preprocess the interest in the same fashion as it was done during the preprocessing of the tokenized corpus, transforming each token to its stemmed version so that the terms can be understood by the embedding model.

Right away, we will use it as the parameter for the function *infer_vector()* from our Doc2Vec model, which will reduce the text to a vector with size 100 so that it can be compared to those in the embedded corpus. In order to so, we will use the built-in function *most_similiar()* from the model found in the *model.docvecs* module using the inferred vector as input and stablishing the number of documents to return with the *topn* attribute.

For this implementation I felt it was sufficient to deliver the list of the 3 most similar but the number may be changed regarding the desires of the deploying platform. We will then order them by its similarity to the querying profile (the 2nd argument for each element on the list). Finally, we retrieve the position of the chosen snippets in order to deliver the results to the user, it is the 1st argument of each element and it remains the same as in the original corpus.

Delivery of results

Using the profile, we will create a customized introduction to the delivery of articles using the following structure (in order to simplify the explanation an example is provided). I used bold in some areas in order to attract the attention of the user to the interest in question and the matching percentage, transformed to the percentile-scale using int(value * 100).

Pablo, I know you like basketball, nba and lebron. You should probably check out the following articles:

1. You have at least a **56% match**: whole text of article 1.
2. You have at least a **55% match**: whole text of article 2.
3. You have at least a **51% match**: whole text of article 3.

Conclusions

Analysis of results

For the first example, being baseball its only interest, the first article was a 60% match, and it delivered the news of Rafael Soriano, a very well-known pitcher, signing for the Washington Nationals in the MLB. The second article had a 59% similarity rating and it talk about a multi-sport agency which includes baseball. The third one, was the news about a rule change in the sport regarding pitching which would affect the discipline greatly.

The second profile's sole interest was Cristiano Ronaldo, which returned different articles regarding its first match against its former team Manchester United since the signing for Real Madrid. The firs 2 had a 64% match and the latter a 60%, it is understandable to receive these snippets since the corpus is written in English and it was one of the most relevant games for the British spectators. If they were written in Spanish or in other years, the results would include a larger variety of news, but I believe that when this corpus was assembled those were the main news.

Finally, the most complex user, which liked basketball-NBA-lebron and tennis. For, the latter (and simpler) the news retrieved first regarded the Australia Open event, one of the top 4 in the tennis landscape, with a 64% score. The other 2, with a 62 and 60% matching result, informed about first the retirement of the U.S. former world champion Justine Henin from the same Open and from tennis entirely, respectively, due to a series of injuries.

Finally, for his interest in the sport of basketball, more particularly in the NBA and specially on Lebron James, we obtained particularly good results, indicating that the more specific a profile is the better. The first one, with a 56% similarity described how famously LeBron hugged a fan who just won a $75,000 prize by hitting a full court shot, a viral moment in the athlete's career.

The second article had a 55% matching score, and it described the record-breaking winning streak that the Miami Heat (LeBron's team at the moment) were having and the teammates and coaching staff reflection on it. Finally, the last snippet (51% match) discussed the comparisons drawn between Michal Jordan and LeBron James, being both of them the frontrunner contestants for the best player in Basketball history.

As it can be seen, the result obtained were great results, as the are not only providing the information regarding the appropriate topic, but quality articles which describe events which were major news at the time of happening. Furthermore, they are obtained using a highly efficient model regarding both memory and computational resources. Finally, it is important to note that the output is natural and comprehensible to the user, providing a rationale behind the recommendations and trying to attract his attention.

Future steps

I believe that in order to be able to deploy it to a real platform we would need to stablish a few aspects. First, we need to take into consideration how recent the article was published and how many people with similar interests read it. On top of that, like / dislike buttons could be added in order to maximize the customization of the feeding experience for the users.

I would recommend that both tasks are studied and implemented by a team of data scientist and engineers. Furthermore, they should study whether these complements should be added with or after the system is deployed in their platform. Finally, as a reminder, they must not forget to continuously feed the model with new articles and users' reactions for its improvement.

References

Ayari, R. (2020). NLP: Word Embedding Techniques Demystified. *Towards Data Science*. Retrieved from https://towardsdatascience.com/nlp-embedding-techniques-51b7e6ec9f92

InterviewBuddy. (2019). What are the advantages and disadvantages using bag of words feature vector? *machine learning aptitude*. Retrieved from https://www.machinelearningaptitude.com/topics/natural-language-processing/what-are-some-advantages-and-disadvantages-using-bag-of-words-where-would-you-use-it-and-where-would-you-not/

Yara Rizk, M. A. (2018). *Sports articles for objectivity analysis Data Set.* American University of Beirut. Retrieved from https://archive.ics.uci.edu/ml/datasets/Sports+articles+for+objectivity+analysis#