Extracting keywords from Reddit submissions

Rok Hudobivnik

ABSTRACT

Implementing a good automatic keyword extraction algorithm is no trivial task, it has been attempted many times with various degrees of accuracy. The aim of this paper is to present, evaluate and compare some well known methods for this specific problem. In this paper we compare methods RAKE, TextRank and keyword extraction using a LSTM neural network. The LSTM network comparison is then further expanded by implementing a bidirectional LSTM network. Given all of this, the evaluation shows us that the bidirectional LSTM networks are by far the best choice for the automatic keyword extraction out of the compared algorithms.

KEYWORDS

datasets, neural networks, RAKE, automatic keyword extraction, word embedding, TextRank, LSTM

1 INTRODUCTION

Implementing a good automatic keyword or key-phrase extraction algorithm is no trivial task. It is a quite complex problem of discerning which word out of many is the one that conveys the meaning of the sentence or a text. The desire to solve this problem comes from the need for an automatic method that can scan a lengthy text and summarize with a couple of words or phrases that still retain the meaning of the initial text. For such a task human extraction of such words or phrases would be just too slow.

To achieve a solution to this problem, a lot of different attempts were made, from relying solely on frequencies of words in text, using the texts to built a network of interconnected words, to using neural networks. In the following sections some of these approaches will be explained and later on the results of these approaches will be compared to one another.

2 RELATED WORK

2.1 Rapid automatic keyword extraction (RAKE)

In their paper Rose et al. [7] propose an unsupervised, domain-independent, and language-independent method for extracting keywords from individual documents. The proposed approach provides a computationally inexpensive method of keyword extraction over single documents, but the good aspects are somewhat overshadowed by the accuracy of the algorithm. Nonetheless, this algorithm provides a quick and computationally inexpensive way of acquiring a baseline for our comparison to the algorithms listed later on. This approach selects keywords candidates from the input text and constructs a co-occurrence graph. The main part of the algorithm then calculates multiple measures based on that graph, like the degree of nodes, frequency of occurrence, etc. Based on those metrics, the candidate keywords are then ranked by their word score.

2.2 TextRank

Mihalcea and Tarau [5] show an implementation of a novel unsupervised graph-based methods of keyword and sentence extraction. An important aspect of this approach is that it does not require deep linguistic knowledge. Its weakness however stems from ignoring semantic similarities between different texts. This approach takes its inspiration from the PageRank [6] algorithm for ranking web pages based on the number and quality of links to a certain page. The idea behind TextRank is the same, build a network out of words and create connections between them based on co-occurrence of those words in a sentence or a text. Given that, estimates which words are more important that the others, the most central in the constructed network. Unlike PageRank, the edges of the network created by TextRank are undirected. The intuition behind this keyword extraction algorithm is, that the created co-occurrence network will contain densely connected regions for words, terms, that appear often and in different contexts.

2.3 LSTM

Long short-term memory neural networks or LSTM for short were first proposed by Hochreiter and Schmidhuber in 1997 [3]. Since then the LSTM architecture has been improved and built upon, the most notable change was the addition of a forget gate by Schmidt and Schmidhuber [2]. In the recent years the architecture has been adapted for the in natural language processing due to its ability to keep track of dependencies between elements of input sequences, without the vanishing gradient problem that occurs in the simpler recurrent neural networks (RNN) for longer sequences. Not going to much into the details about the math behind a single LSTM cell (Image 1), the main intuition behind this architecture is that, as mentioned before, each cell keeps track of the dependencies between the elements in the input sequence. Inside a single LSTM cell we have an input, an output and a forget gate. These gates control the flow of information from the input to the cell (input gate), the extent to which the input stays in the cell (forget gate) and the extent to which the value in the cell is then used to compute the output activation of that LSTM cell (output gate). The connections from and into the gates are weighted and can be recurrent. Said gates need to be learned during the training phase. For the purposes of this report a slightly modified version of the LSTM network will be used (bidirectional LSTM [1]) in addition to the LSTM network described above.

3 METHODOLOGY

The methods used in this report are mostly implementations of related works described in the previous section. The only notable exception comes in the form of LSTM netowrks where some additional changes were made due to results acquired during preliminary testing.

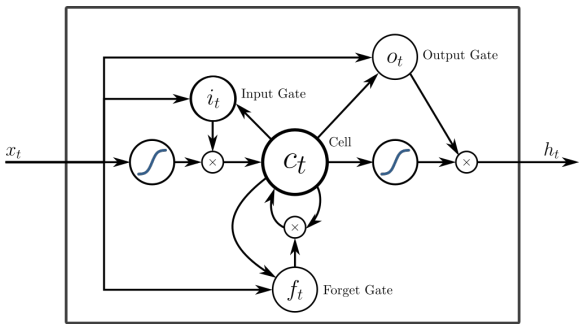

Figure 1: An illustration of a single LSTM cell.

3.1 Datasets

In our paper we will mostly be dealing with a dataset of submissions and comments from the platform Reddit[1], more specifically from a Subreddit called Movie Suggestions[2]. The dataset is a product of crowdsourced work and is comprised of submissions, comments and hand denoted keywords. The texts are relatively short (with average around 60 words per text) in length and contain between 0 and 18 keywords (with the average around 3). For further references, this will be the dataset used in the experiments and the calculation of the results.

In addition to the Reddit submissions data, we also used the INSPEC datset [4], a well known keyword extraction dataset, for the purpose of transfer learning. A method of supervised learning where a model is firstly trained on a well established dataset and afterwards it is fine tuned on the target dataset. in practice such an approach is often used with smaller or less well defined datasets, such as, in our case, the Reddit dataset.

To prepare the mentioned datasets for keyword extraciton, several preprocessing techniques were implemented, most notably tokenization, the process of dividing a text/sentence into tokens and a simple substitution methods, used to remove markdown tags and some other special characters that might appear in the texts. For the first two approaches the set of keywords for each text was left untouched. For the use in deep learning, the part of the dataset, that contains the keywords of the current text was transformed into a two dimensional binary array in which the keywords of certain text are marked with 1 and other words with 0.

3.2 Word embedding

To be able to train the deep learning models, we would need vectorized representations of the input texts , since the normal string versions of words cannot be used. A simple solution to this would be the use of one-hot encoding of words. That kind of encoding unfortunately does not retain any dependencies between words, i.e. words "bird" and "birds" would be considered completely equally different to one another as perhaps words "computer" and "person". Since we would like to train the neural network to learn about connections between words, we will need to use a better encoding method.

[1]https://www.reddit.com/
[2]https://www.reddit.com/r/MovieSuggestions/

One such method is Word2Vec, a method of training a model on a large dataset, to predict what words come next in a certain sequence. By then taking the second to last layer of this neural network model, we can extract word encodings. For training the deep learnign models we will be using a pre-trained Word2Vec model, that was trained on 100 billion words from a Google News dataset.

3.3 LSTM

As we described in the beginning of this report, we will use a modification of the original LSTM [3] network called the bidirectional LSTM (bi-LSTM) network [1]. This architecture uses uses 2 LSTM layers for the input data, one for sequences from the original data and the second one for the same sequences, but this time reversed. In the end the outputs of the two LSTMs are joined togehter. This is done to capture any additional hidden features that might not be apparent from just a normal pass over a sequence.

Additionally, during the fine tuning of the network we have discovered that we could obtain slightly better results if we used two bi-LSTM layers one after another instead of just one. Given that, the final architecture of the network that we used started with two bi-LSTM layers, followed by a dropout layer, to prevent overfitting, and afterwards a fully connected layer which is then connected to a softmax output layer. For the fully connected layer, we have discovered, similarly to the bi-LSTM article [1], that 150 neurons works best for this problem.

4 EXPERIMENTS AND RESULTS

For the experimental part of this report we have compared the implementations of the algorithms described above using precision, recall and the F1 score. The measures were calculated by comparing the ground truth keywords of the original Reddit dataset and the keywords extracted by the implemented algorithms.

Method	Precision	Recall	F1-score
Rake	**0.797**	0.063	0.117
TextRank	0.683	0.089	0.158
LSTM	0.622	0.213	0.317
bidirectional LSTM	0.684	0.258	0.375
LSTM (transfer)	0.623	0.323	0.425
bidirectional LSTM (transfer)	0.518	**0.438**	**0.475**

Table 1: Evaluation results of the described keyword extraction algorithms

As we can see from the table of results (Table 1), the two simpler algorithms TextRank and RAKE get a very high precision but a very low recall, meaning that the keywords that are chosen by these two algorithms are highly likely the true keywords, but at the same time, they are unable to find more than a handful of these. In contrast the LSTM algorithms have slightly lower precision, but for that reason a very high recall compared to the first two, also resulting in a high F1 score.

Additionally we can also see big differences in the F1 score between the normally trained LSTM networks and the ones trained using transfer learning. On the other hand, the differences between

LSTM and bi-LSTM network become more ambiguous with transfer learning, where the LSTM network has a noticeably higher precision, but still a lower recall.

5 CONCLUSION

To wrap things up, the algorithms performed as expected, with the simpler two algorithms being on very similar terms in regard to the results (Table 1) and the deep learning ones performing very much better. Concluding this report, it needs to be mentioned that the number of neurons and the architecture of the neural networks are by no means optimal and could be improved with more extensive testing.

REFERENCES

[1] Marco Basaldella, Elisa Antolli, Giuseppe Serra, and Carlo Tasso. 2018. Bidirectional lstm recurrent neural network for keyphrase extraction. In *Italian Research Conference on Digital Libraries*. Springer, 180–187.

[2] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. 1999. Learning to forget: Continual prediction with LSTM. (1999).

[3] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[4] Anette Hulth. 2003. Improved automatic keyword extraction given more linguistic knowledge. In *Proceedings of the 2003 conference on Empirical methods in natural language processing*. Association for Computational Linguistics, 216–223.

[5] Rada Mihalcea and Paul Tarau. 2004. Textrank: Bringing order into text. In *Proceedings of the 2004 conference on empirical methods in natural language processing*.

[6] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank citation ranking: Bringing order to the web.* Technical Report. Stanford InfoLab.

[7] Stuart Rose, Dave Engel, Nick Cramer, and Wendy Cowley. 2010. Automatic keyword extraction from individual documents. *Text Mining: Applications and Theory* (2010), 1–20.