Suggesting a trip destination with word embeddings

Alejandro Gonzlez Rogel, *Student, IEEE* Dennis Doerrich, *Student, IEEE*

***Abstract*—Existing travel search engines depend on fixing the location or comparing different destinations by often only 1-dimensional pre-defined parameters like price-level. We propose a model that is able to accept free user input and matches it to the most suitable place, based on similarity between input and spot-description in our data-set. This model is presented as a Telegram-Bot to be easy accessible across multiple platforms.**
The similarity measure was obtained by encoding descriptions and each query into a vector representation of fixed length. This conversion was achieved by an auto-encoder, using LSTM networks that receive vector word-embeddings as input.

I. Introduction

There are multiple applications that allow you to look for information about a city or location. And some of them also allow you to compare similar destinations usually based on proximity or price-level. However, all these applications usually have a similar problem: the query the user can make to those applications is quite limited. Usually the user need to know the name of the place he is referring to or needs to input a set of fixed parameters that can be used to determine the ideal place for him. This can be an issue for those people that just want to explore the world but do not have any specific place in mind.

During the last decades, we have seen multiple improvements in the field of Natural Language Processing (NLP) and, a directly connected field, dialog systems (such as chatbots). This technology can help us to develop an application the user can communicate with in a more realistic and open way. And, although we have not achieved yet the point where a computer can completely understand human language, it is easy to find user assistance and information retrieval applications in our every day live.

The final goal of this project was to make use of Natural Language Processing (NLP) techniques to create an application that would suggest a location based on a query given by the user. We believe that the use of this technology could improve both the performance of our model and the experience of the user. More in detail, we expect to add create a model that can accept a more flexible input (making it easier to use), generate a better analysis of what the user is asking for and, that way, be able to provide an accurate answer.

This paper is structured as follows: section II provides an introduction to all the concepts that will be necessary for the understanding of this work. Section III explains in more detail the insides of the system and, subsection D, its final results. Section IV presents a deeper analysis of these results and sections V and VI will close by providing a general overview of the system and discussing improvements and future work. As requested, an appendix has been added at the end (section VI) stating the distribution of the work.

Cognitive Computational Modeling of Language and Web Interaction, 19/06/2017, and George Kachergis.
email: a.gonzalezrogel@student.ru.nl

Cognitive Computational Modeling of Language and Web Interaction, 19/06/2017, and George Kachergis.
email: ddorrich@student.ru.nl

II. Background

Although there has been a noticeable improvement in the field of natural language processing (NLP), this includes its applications to chatbots, the truth is that we still have not reached the point where an algorithm will be able to keep an open domain conversation and easily pass the Turing test [1]. However, there are several applications for which we do not need such complex systems for achieving a good performance. This is the case for assisting chatbots. This models usually operates in a closed domain (they just fulfill a really specific task) and retrieve predefined information instead of generating is itself, i.e, it is a retrieval-based models. The system that we explain and analyse in the following sections have this two characteristics.

But first, in order to handle a text corupus, we need a way to represent that data in such a way that we are able to operate with it. Recently, structures such as word2vec [2] or GloVe [3] have been developed to provide a vector representation of words that can represent semantic and syntactic properties of the words. For this project, however, we do not only need to use word embeddings but a system that is able to represent, in a limited space, the essential properties of a complete text. This has been done by several authors before, and for different purposes. For instance, Google researchers developed a paragraph vector that would help to predict words in a document [4]. Closely related to this project, we find the work done by [5] where they came out with a technique to represent business reviews in a vector by locating and processing key phrases in those reviews.

Due to the non-supervised nature of our problem, we are limited to use a solution that do not require of labeled data. Our approach will be more similar to the one found in sequence-to-sequence or sequence-to-vector models. Specially relevant is the work done in [6], where a structure consisting of two recurrent neural networks was used to solve a translation problem. The first network, named encoder, would process the input and generate a vector representation of it so the second network, the decoder, generates a desired output. This architecture is called sequence-to-sequence autoencoder. The hidden layers of such a network were composed by Long short-term memory cells (LSTM)[7], a special type of layer that is capable of learning long term dependencies.

Finally, because we want to provide a clean and easy to use interface to the user, we need to mention the Telegram Bot Platform[1]. There are several libraries that offer the possibility to set up a public chatbot without programming the interaction between our algorithm and the user and this platform is one of the most popular ones.

III. Project

For this project we wanted to create a retrieval-based, closed domain chatbot model. By feeding the system with a large database of places and their brief descriptions, our model would be able to analyse user's input when asking for recommendations and compare such an input with the database so it can find the most suitable place to visit. When handling text, we will use an encoder network that should be able to generate a vector representation of a sentence given the word-embeddings of the words that form such sentence.

A. Pipeline

Our baseline so to say was called Geobot, that was created for a former assignment by author Dennis Doerrich. It already was able to give recommend a city, based on weather and distance to the user using keyword matching and APIs for the weather and travel data[2][3]. This bot would interact by detecting pre-defined keywords in the user's input. We hence took only the already existing architecture to communicate with telegram, process new input, give response etc. and extended it by the above mentioned free text input to find a match (see more in section C). In order to submit a more clear application, the old functionalities were also deleted.

B. Data and Preprocessing

To build a functional application we needed a dataset of locations with their respective meaningful description. We used data corresponding to the travel guide Wikivoyage [4], which backup dump can be obtained from Wikimedia [5]. This raw data contained more than 50.000 entries of Wikivoyage, although not all of those had information about cities.

Preprocessing was performed over the data to filtrate all those pages that did not correspond to a city or place. From those, only the initial paragraphs before the start of a first section were considered. We assumed these paragraphs to contain the general information of a location. Over that text, we performed additional operations to eliminate entries with small descriptions, to remove stopwords (using the list provided by the python package SPACY [6]), and to remove any additional item that could be found in the text (format, images, hyperlink to other webpages). Our final dataset contained almost 11.000 locations.

C. Matching

In order to match the best city to the user input we needed to transform both the preprocessed city descriptions and the text input into a single vector each. For that, we worked at a word level and downloaded a GloVe vector representation of each word from the Standford Webpage [7]. We used the dataset with 6 billion tokens of 200 dimensions.

Two different strategies were considered when creating the description vector representation: extracting that vector from the mean value of all the vectors the description was made of and using an autoencoder network. The autoencoder allows input sentences up to 200 words (descriptions will be zero-padded or shortened if necessary) and both encoder and decoder contain just one LSTM layer. Once the network has been trained, the result produced by the encoder part of the autodecoder is considered the vector of the description. The encoder also produces vectors of 200 dimensions so the results are comparable to the ones obtained by the first strategy. This network was trained using stochastic gradient descent and using a batch size of 32. The autoencoder was trained to generate, as an output, the same sentence it was receiving as an input.

Once the network has been trained and all the locations have a vector representation associated to them, the chatbot can be tested. Figure 1 shows the internal behaviour of the application: the user will ask for a recommendation giving some information about the features he expects to find. This input will be transformed into a vector using the neural network that we mentioned above and such vector will then be compared to the ones generated for the descriptions of the places, which must be persistently stored so they are not generated each time the user ask interacts with the program. We will then select the three places whose descriptions are closer to the user's query according to their euclidean distance.

D. Evaluation

A major difficulty when evaluating performance is to decide the degree of correctness of a given answer. One might argue that labeling a proposal's quality is rather subjective and the results could not be representative enough if not enough people is asked during the labeling process. For this reason, we decided to try a different approach: we can measure quantitatively how precise our system recognizes a city by utilizing parts of their description as user input. As the fraction length grows and gets closer to the Training length we expect it to converge towards 100 percent correct match.

We tested on several descriptions length, although details are explained in section IV.

[1] https://telegram.org/blog/bot-revolution
[2] https://openweathermap.org/api
[3] https://developers.google.com/maps/documentation/distance-matrix
[4] https://www.wikivoyage.org/
[5] https://dumps.wikimedia.org/enwikivoyage
[6] https://spacy.io/
[7] https://nlp.stanford.edu/projects/glove/

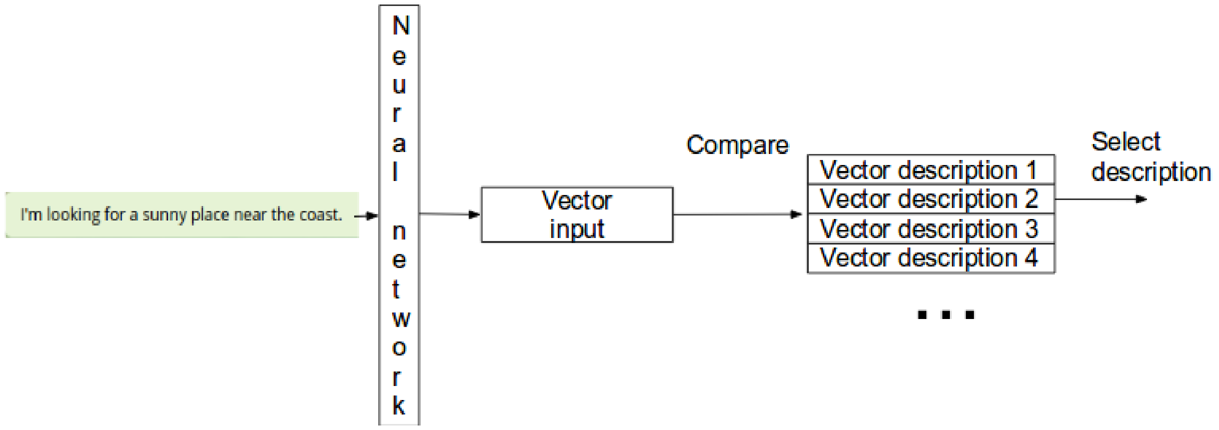
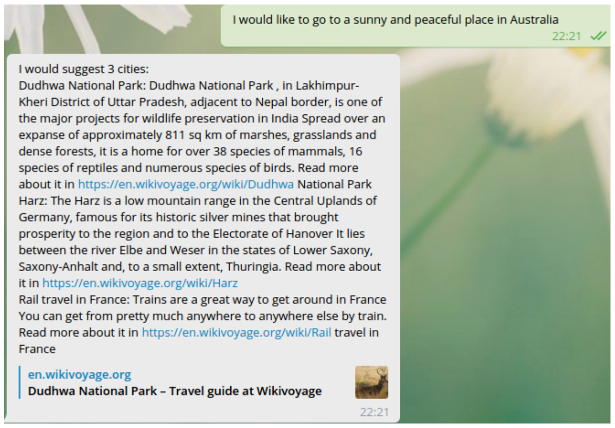


Fig. 1. Pipeline of the system: The user's input is encoded into a vector that is later compared to the ones generated by the city descriptions, selecting those whose distance is lower.

E. Examples

In figure 2 we provide an example of the model's response to a query:



I would like to go to a sunny and peaceful place in Australia
22:21

I would suggest 3 cities:
Dudhwa National Park: Dudhwa National Park , in Lakhimpur-Kheri District of Uttar Pradesh, adjacent to Nepal border, is one of the major projects for wildlife preservation in India Spread over an expanse of approximately 811 sq km of marshes, grasslands and dense forests, it is a home for over 38 species of mammals, 16 species of reptiles and numerous species of birds. Read more about it in https://en.wikivoyage.org/wiki/Dudhwa National Park
Harz: The Harz is a low mountain range in the Central Uplands of Germany, famous for its historic silver mines that brought prosperity to the region and to the Electorate of Hanover It lies between the river Elbe and Weser in the states of Lower Saxony, Saxony-Anhalt and, to a small extent, Thuringia. Read more about it in https://en.wikivoyage.org/wiki/Harz
Rail travel in France: Trains are a great way to get around in France You can get from pretty much anywhere to anywhere else by train. Read more about it in https://en.wikivoyage.org/wiki/Rail travel in France

en.wikivoyage.org
Dudhwa National Park – Travel guide at Wikivoyage
22:21

Fig. 2. Answer of a query given by a system that uses vectors generated by an encoder network.

IV. Results

As previously explained in subsection D, we will present to our model 100 random definitions of places that is already has stored. This definitions will not always be complete, i.e, we will just present a limited number of words of such descriptions. We will then measure the percentage of hits the network has, i.e, the number of times that the network finds the place which description is currently processing. Results are shown in table I

Number of words of the description	% of hits
Complete desription	26%
20 words	%
10 words	0%
5 words	24%

TABLE I

V. Summary

With this work we tried to provide a complete system that would allow the user to input a highly flexible query asking for a location and would provide a clear and accurate suggestion based on the information we had about several locations. We tried to achieve that by using the encoder network of a trained autoencoder. Although the whole system works and it is able to run without any complications, the suggestions made by our bot are mainly random, which certainly means that the network was not able to create a meaningful representation of the location descriptions.

VI. Conclusion and Outlook

We created a system that, although it is able to perform all the tasks that is was programmed to do, it is unable to create a meaningful representation of paragraphs (descriptions). This is, without a doubt, the aspect of the system the next iteration should focus on.

As often in NLP challenges, the simple approached worked better than the LSTM network. From example Kaggle challenges we know that those complex approaches can outperform the easier ones but only if all parameters are tuned well. Thus it is still open to play with deepness of the network, number of lstm-units, dropout configuration and batch normalization to prevent overfitting, early stopping and other changes that make the prediction more accurate. As can be seen from this paper, and due to the time constrains, there is a lot to try out when it comes to tuning of the structure and the parameters of our network.

We used euclidean distance to measure distance between vectors. This is probably not the best approach and it might have a strong influence in our result. A different measure system should be considered in the future.

When planning this project we came to realize how hard it would be for a chatbot to determine which action to perform in the case he was programmed to reply to different types of queries. This is probably one important restriction to consider if we would like to make a commercial use of the bot.

Next to the matching process, which admittedly is the heart of our machine, there are several functionalities that can be improved for a better user experience in the future. Considering for instance a place's popularity, travel-time, weather predictions or local price-level, more features can be taken into account than the free input. Whether these preferences are fixed or asked once per user or even adapt automatically to user behavior is still open.

The today state of the bot gives only one short description per city that we extracted from the dataset. Wikivoyage on the other hand offers travel descriptions by usually similar sections like 'Get Around' or 'See'. Enabling the option to get more information about a specific section within the app instead of refering to links can enhance the usability.

Nevertheless we achieved to build a Chatbot that in its given domain accepts free text-input and gives reasonable answers.

References

[1] Alan M Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.

[2] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. word2vec, 2014.

[3] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014.

[4] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1188–1196, 2014.

[5] Neal Khosla and Vignesh Venkataraman. Learning sentence vector representations to summarize yelp reviews. 2015.

[6] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

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

Appendix

A. Author Contributions

Although this report was read entirely by both authors and all its were modified by both of us, there were sections where one person contributed more than the other. Dennis wrote the abstract, several sections of the Project section and the conclusion where Alejandro wrote the Introduction, Background, the other half of the Project section, Results and the summary.

About the code, Dennis provided the first version of the preprocessing script, the baseline approach of the vectorization of the descriptions, the pipeline of the bot and the code for averaging vector representations and measuring their distance. Alejandro provided an improved version of the preprocessing, the code to vectorize text using the autoencoder (this includes the neural network itself) and methods to present the user the results of the algorithm.

B. Repository

The code for this project can be found in https://github.com/AlexGonRo/CCMWLI_final_project