CitySearcher: A City Search Engine For Interests

Mohamed Abdel Maksoud*
Codoma.tech Advanced Technologies
Egypt
mohamed@amaksoud.com

Gaurav Pandey*
University of Jyvaskyla
Finland
gaurav.g.pandey@student.jyu.fi

Shuaiqiang Wang
University of Manchester
UK
shuaiqiang.wang@manchester.ac.uk

ABSTRACT

We introduce *CitySearcher*, a vertical search engine that searches for cities when queried for an interest. Generally in search engines, utilization of semantics between words is favorable for performance improvement. Even though ambiguous query words have multiple semantic meanings, search engines can return diversified results to satisfy different users' information needs. But for *CitySearcher*, mismatched semantic relationships can lead to extremely unsatisfactory results. For example, the city *Sale* would incorrectly rank high for the interest *shopping* because of semantic interpretations of the words. Thus in our system, the main challenge is to eliminate the mismatched semantic relationships resulting from the side effect of the semantic models. In the previous case, we aim to ignore the semantics of a city's name which is not indicative of the city's characteristics. In *CitySearcher*, we use *word2vec*, a very popular word embedding technique to estimate the semantics of the words and create the initial ranks of the cities. To reduce the effect of the mismatched semantic relationships, we generate a set of features for learning based on a novel clustering-based method. With the generated features, we then utilize learning to rank algorithms to rerank the cities for return. We use the English version of Wikivoyage dataset for evaluation of our system, where we sample a very small dataset for training. Experimental results demonstrate the performance gain of our system over various standard retrieval techniques.

CCS CONCEPTS

• **Information systems → Retrieval tasks and goals**;

KEYWORDS

Information Retrieval, word2vec, Feature Engineering

ACM Reference format:
Mohamed Abdel Maksoud, Gaurav Pandey, and Shuaiqiang Wang. 2017. CitySearcher: A City Search Engine For Interests. In *Proceedings of SIGIR '17, Shinjuku, Tokyo, Japan, August 07-11, 2017*, 4 pages.
https://doi.org/10.1145/3077136.3080742

*Indicates equal contribution

1 INTRODUCTION

In this paper, we present *CitySearcher*, a vertical search engine that generates a ranking of cities in response to an interest given by the user. The interest acts as a query on a travel document corpus, which mostly contains documents that represent and contain information about a particular city. For example, for the interest *'History'* the top 3 cities presented are *'Rome'*, *'Lviv'* and *'Oslo'*.

Generally in search engines, utilization of semantics between words is beneficial for an information retrieval system for performance improvement. To illustrate, the presence of sentences like *'The city has bright sea shores'* in a document should contribute in giving it a high ranking score for the query *'sunny beach'*. Even though the sentence and the query have no common words, but there is semantic similarity between the words *'bright'* and *'sunny'* as well as *'shores'* and *'beach'*. Even though ambiguous query words have multiple semantic meanings, search engines can return diversified results to satisfy different users' information needs.

However in *CitySearcher*, mismatched semantic relationships can lead to extremely unsatisfactory results. This happens particularly when the city names gets incorrectly semantically interpreted. As an example, the city *'Sale'* would incorrectly rank high for an interest like *'shopping'*, because of semantic relationships between the two words as well as repetition of the city's name in its document. Thus for *CitySearcher*, the main challenge is to eliminate the mismatched semantic relationships resulting from the side effect of the semantic models. In the previous case, it is rather desired that the semantics of city name are ignored completely for matching it with the interest. This is because, the meaning of a city name if any, can be completely unrelated to its characteristics.

In *CitySearcher*, we we use *word2vec* [7, 8] to estimate the semantics of the words. *Word2vec* is a widely used word embedding technique, and it is shown that the vectors generated by *word2vec* preserve the semantic relations between the words, *e.g. vec('Paris') - vec('France') + vec('Italy')* is very close to *vec('Rome')* [7].

With the semantic vectorizations of the words, the *CitySearcher* algorithm firstly represents each city as the words in the corresponding document describing the target city, and then calculates the initial ranking scores for each city-interest pair with the similarities between the vectors of the words in the city document and the interest. While *word2vec* provides semantic interpretations in retrieval for *CitySearcher*, it also introduces the issue of certain mismatched semantic similarities, especially between interest and city names. Thus besides the initial ranking scores between cities and interest queries, we also propose a set of new features for ranking of the cities with the given interest queries. In particular, we create a set of topics by clustering all of the vectors of words in the vocabulary. For each interest, we choose a subset of closest topics and calculate the similarity from the city to the topics as new

features. We finally create a training set by collecting users' relevance assessments (ratings) for city-interest pairs, and use machine learning (e.g. different regression algorithms) to rerank the results of the cities for return.

While *CitySearcher* could preserve semantic relationships in general and improve retrieval performance, it is also expected to show capability of reducing the effect of undesired semantic relationships. With our proposed features, the semantics of the interest query words are interpreted with a set of closest topics (clusters of words), allowing us calculating the similarity between cities and interest queries with multiple enriched dimensions.

We use the English version of Wikivoyage dataset [1] for evaluation of our system. With a very small sampled dataset for training, experimental results demonstrate the performance gain of our system over various standard retrieval techniques.

2 PROBLEM STATEMENT

Let D be a corpus containing travel documents doc_{c_1}, doc_{c_2} ... doc_{c_z} having information about the cities $c_1, c_2 \dots c_z$, respectively. There is one-to-one mapping between documents and cities, i.e. each document represents only one city and each city has only one document for it. For a word representing an interest itr, retrieve a ranking of cities in decreasing order of relevance.

The main challenges or targets for the retrieval task along with their proposed solutions are:

(a) The semantic relationships between words are beneficial for retrieval, and hence should be utilized.
Solution: Create vector representations of the words in corpus using *word2vec* (that preserves semantic relationships) and then use them for retrieval.

(b) However, consideration of semantic similarity between certain words, especially city and interest, lead to irrelevant results. Such effect mismatched of semantic relationships needs to be reduced.
Solution: To reduce the effect of the undesired semantic relationships, train a ranking model using machine learning. For the training process, collection of ratings from the users for city-document pairs is a prerequisite.

(c) There are few features for city-document pairs as well as limited number of ratings from the users. This limits the efficiency of training.
Solution: Generate new features for city-document pairs using vector representations of words in the vocabulary.

3 METHOD

Mokolov et al [7, 8] introduced *word2vec*, a prominent procedure to generate vector representations of words. They introduced the continuous *skip-gram*, which is a neural network model consisting of input, projection and output layers that predict the surrounding words. This is achieved by maximizing the average log probability for the words $(w_1, w_2 \dots w_N)$ present in the corpus:

$$\frac{1}{N} \sum_{i=1}^{N} \sum_{j \in Sur(i,z)} log\, p(w_j|w_i) \tag{1}$$

[1] https://en.wikivoyage.org/wiki/Main_Page

Here, $Sur(i, z)$ represents a context window for training consisting of z words that surround the word i. Also, $p(w_j|w_i)$ is the hierarchical softmax of the respective word vectors $v(w_j)$ and $v(w_i)$. One of the key benefits of generating word vectors is that the training is completely unsupervised. Moreover, the generated vectors enable the calculation of similarities between words, by simply calculating the similarity of vectors.

CitySearcher utilizes *word2vec* on a corpus comprised mainly of travel destination documents (each document has information about a particular city), so that vector representations of each word present in the vocabulary is computed. That is to say, for all the words present in the corpus: $w_1, w_2 \dots w_N$, the corresponding vector representations: $v(w_1), v(w_2) \dots v(w_N)$ are generated. The sections that follow, describe how these vector representations are used to generate ranking of cities for an interest.

3.1 Basic Algorithm: Using Vector Similarity

The similarity between two words $w1$ and w_2 from the corpus can be calculated on the basis of cosine distance between their vectors:

$$sim(w_1, w_2) = 1 - cosineDistance(v(w_1), v(w_2)) \tag{2}$$

To rank cities against an interest itr (a word representing an interest, e.g. music, history etc.), we need to find the ranking score for itr and the cities. This can be done by calculating the average similarity score for itr and the words closest to it in the city document. If doc_c is the document representing the city c and contains words $w_1, w_2 \dots w_n$, the similarity scores are calculated between itr and the words of the document: $sim(itr, w_1)$, $sim(itr, w_2)$ ... $sim(itr, w_n)$. Let $s_1, s_2 \dots s_k$ be the top k scores from these similarity scores. Then, the ranking score for the city-interest pair is calculated as the average of top-k similarities. This ranking score, $initialScore(c, itr)$ can be formulated as:

$$initialScore(c, itr) = \frac{1}{k} \sum_{i=1}^{k} s_i \tag{3}$$

The calculation of such scores for an interest corresponding to different cities, and then sorting the cities in a deacreasing order of the scores, enable ranking the cities for the interest.

3.2 Improvement using Machine Learning

The algorithm described in the previous section utilizes the semantic relationship between words, captured by their vector representations. Though this is expected to perform well in most cases, utilization of vector similarities can lead to fundamental problems in particular situations. Since the city's name is semantically interpreted and it is usually mentioned several times in the document, it has a significant influence on the ranking. As a result, a city called *Sale* ranks first for the interest *Shopping*. Similarly, a small village named *Chicken* appears in the top documents for the interest *Food* and a city called *Mobile* ranks highly for the interest *Technology*.

To circumvent these defects and improve the performance, the city rankings could be generated by training using regression algorithms like Kernel ridge regression [9] and Logistic regression [9], on the relevant assessments obtained from the users. A relevance assessment can be represented as a tuple: (c (*city*), itr (*interest*), r

(*rating*)) where r $\in \{1, -1\}$. The user gives a value 1 to r if she thinks that city c is relevant for the interest itr, and -1 if it is irrelevant. However, there is only one feature derived from vector representations that can be used in training i.e. the city-interest initial ranking score as calculated in Equation 3. The lack of features would limit the effectiveness of the training. Hence, enriching our training dataset would improve training effectiveness. To this end, in the next section we describe a novel approach employed by us, to generate a set of new features per relevance assessment, by utilizing vector representations of the words in the vocabulary.

3.3 CitySearcher Feature Generation

Our novel technique generates features corresponding to each relevance assessment (c, itr, r). We create a set of topics by clustering all of the vectors of words in the vocabulary. For the interest itr, we choose a subset of closest topics and calculate the similarity from the city c to these topics as new features. The intuition behind using this technique is that the relevance of a c to itr, should be related to the similarity of c to the topics closest to itr.

Firstly we use k-means [12] clustering algorithm to divide the entire vocabulary into M clusters. The clustering algorithm utilizes the precomputed vector representations of the words in the corpus and calculates the similarity between the words using Equation 2. The centroid of word vectors in each cluster is considered as a topic (also a vector), resulting in creating M topics. Also, each topic is representative of the words in its cluster. Thereafter, for the relevance assessment (c, itr, r) we find t_{itr} , the topic to whose cluster the interest word itr belongs. Then, we find the p topics closest to t_{itr}: $t_1, t_2 ... t_p$, which are arranged in increasing order of their cosine-distance from t_{itr}. It should be noted that the p closest topics to t_{itr} also includes t_{itr} itself, making $t_1 = t_{itr}$.

Since in our case, each city c is represented by only one document doc_c and also doc_c is representative only for c, the calculation of city to topic similarity becomes quite straightforward. It can be calculated as $topicSimilarity(c, t)$, which is the ratio of the number of words in doc_c that belong to the cluster for topic t:

$$topicSimilarity(c, t) = \frac{|(words\ in\ doc_c) \cap (words\ in\ cluster\ of\ t)|}{|words\ in\ doc_c|} \tag{4}$$

For the relevance assessment (c, itr, r), the similarities of c to the p closest topics to t_{itr} $(t_1, t_2 ... t_p)$ can be considered as the p newly generated features $(f_1, f_2 ... f_p)$:

$$f_i = topicSimilarity(c, t_i), i = 1, 2, \ldots p \tag{5}$$

It should be noted that other features can also be included for training along with the generated features. In our method we include two more features: (a) initial ranking score for city document pair, $initialScore(c, itr)$, using the basic approach (Section 3.1) (b) document length, as it is a strong indicator of importance for travel destination documents. This makes a total of $p + 2$ features. These features are used to form the training data, that in turn is used by regression algorithms, that generates improved ranking models.

3.4 Remarks

We have used a rather sophisticated method to construct the training set. The method was dictated by the relatively small number of ratings available. Potentially better results can be constructed if we build a dataset for each interest. In such setting, the ratios for each topic would have the same order throughout the dataset, which makes learning a model out of them more straightforward.

4 EXPERIMENTAL SETUP

4.1 Dataset

We have used the English version of Wikivoyage dataset for our experiments, which contains 6691 documents that predominantly represent different travel destinations (cities). It has a corpus size of 1832499 words and a vocabulary of 106634 words. Also, we created a list of 50 common travel interests, by taking inspiration from several famous travel webpages. Moreover, for relevance assessment, we collected 800 ratings for randomly selected city-interest pairs. These ratings were received from 40 people (Europeans, both genders, age 18-60) using the crowdsourcing platform clickworker [2]. Furthermore to enable training and testing, the ratings were divided into a training set (80%) and a test set (20%).

4.2 Evaluation Metric

We use the standard ranking accuracy metric: normalized discounted cumulative gain (NDCG@1-5) [6], to evaluate the rankings generated by our algorithms and the baselines.

4.3 CitySearcher Setup

Vector representations for the words in WikiVoyage dataset are created using *word2vec*, for a window size of 10. Firstly, the rankings of cities are generated for the interests by using the basic *CitySearcher* algorithm described in Section 3.1. For calculating the ranking scores, we used top 10 similarity scores between the interest and the words in the document, i.e. $k = 10$.

Then, to implement the feature generation technique described in Section 3.3 to enrich the training set, we clustered the word vectors and created 100 topics for the Wikivoyage dataset (i.e. $M = 100$). Thereafter, for each relevant assessment (c, itr, r), we generated features corresponding to all the 100 closest topics to the interest itr, i.e. value of p is also 100. It means that the similarities of c are calculated to all the topics (ordered from closest to farthest to the topic of interest itr), and considered as features. Two more features were included: the document length for doc_c (the document representing c) and the initial ranking score for city-interest pair using Equation 3. Thus, for each relevance assessment we have a total of 102 features. Admittedly, these values of k, M and p have been chosen intuitively.

The following two regression algorithms were applied on the training set with generated features to create the ranking models:

- **Kernel Ridge Regression**: Kernel ridge regression (KRR) [9] combines Ridge Regression (linear least squares with l2-norm regularization) with the kernel trick. It thus learns a linear function in the space induced by the respective kernel and the data. We used RBF kernels. The optimal parameters were found using 3-fold cross validation.

[2] https://clickworker.com

Table 1: Performance comparison for NDCG@1-5

NDCG	@1	@2	@3	@4	@5
TF-IDF	0.8000	0.7500	0.8055	0.8017	0.8234
Okapi-BM25	0.8000	0.8500	0.8380	0.8479	0.8542
LSI	0.8000	0.7500	0.8055	0.8017	0.8234
LDA	0.8000	0.7500	0.7620	0.7521	0.7821
LGD	**0.9000**	0.8000	0.8480	0.8403	0.8475
CS-Basic	0.8000	0.7500	0.7740	0.7861	0.8154
CS-KRR	**0.9000**	**0.9000**	0.8161	0.8465	0.8620
CS-LR	0.8000	0.8500	**0.8576**	**0.8614**	**0.8638**

- **Logistic Regression**: Logistic regression [5] is a classification algorithm, that nevertheless computes estimates of the class probabilities, that can be used for ranking.

For these regression algorithms, we have used the implementation provided by the Python package scikit-learn [3]. The models generated by these models are evaluated on the test set.

4.4 Baselines

To compare the rankings produced by our algorithms, we use the five widely used retrieval techniques as baselines:

- **TF-IDF**: Term frequency- Inverse Document Frequency [10] is a very famous ranking model and employs bag-of-words representation. The weight of a term increases proportionally to the number of its occurrences in the document, but also decreases in proportion to its frequency in the corpus.
- **Okapi-BM25**: Okapi BM25 [11], a well known ranking scheme, is based on probabilistic retrieval framework.
- **LSI**: Latent Semantic Indexing [4] uses singular value decomposition to identify patterns in the relationships between the terms to generate a semantic feature space.
- **LDA**: Latent Dirichlet Allocation [1] models each document using the underlying set of word topics.
- **LGD**: LGD [2, 3] weighting model is a high performing version of the log-logistic model.

5 RESULTS

The results are shown in Table 1 for the metric NDCG@1-5. The results are presented for the baselines: TF-IDF, Okapi BM25, LSI, LDA and LGD. Also, the basic *CitySearcher* algorithm from Section 3.1 is denoted as CS-Basic. Moreover, the evaluations for the learning-to-rank models trained on generated features (Section 3.3) using Kernel Ridge Regression and Logistic Regression are presented as CS-KRR and CS-LR respectively.

We can see that the results for CS-Basic are better than the baseline LDA for all metrics, however they are worse than TF-IDF, Okapi-BM25, LSA and LGD in most cases. Also, LGD is the best performing method among the baselines. We observe the benefits of using machine learning on generated features, as the performances of both such methods, CS-KRR and CS-LR, are better than that of CS-Basic. CS-LR gives the best results for NDCG@3-5, while CS-KRR gives the best performance for NDCG@1-2. However, LGD ties with CS-KRR on NDCG@1. Overall, we can conclude that CS-LR is the best performing method because apart from giving best performances for NDCG@3-5, its performance is better than most baselines even for NDCG@1-2. CS-KRR on the other hand hand falls behind LGD as well as Okapi-BM25 for NDCG@3-5.

6 CONCLUSION

We introduced *CitySearcher*, a search engine that ranks cities for interests. It uses vector representations of words to estimate their semantics. The basic algorithm computes ranking scores for each city-interest pair using similarities between the vectors, but suffers because of mismatched semantic similarities. To solve this issue, we propose a set of new features to rerank cities for the interest. A set of topics is created by clustering all vectors of words in the vocabulary. Then, we choose a subset of closest topics to the interest and calculate the similarity from the city to the topics as new features. Even for a small training set, the results show the benefits of reranking using regression algorithms on generated features over the basic algorithm as well as the standard retrieval techniques.

In future work, we want to create an algorithm that can incorporate more than one interest in the query. Additionally, we would like to return a ranking of tours (group of cities), in response to multiple interests. Since we have a small training set in our experiments, we would like to experiment with larger training data. Moreover, the parameters in our experiments can be further explored and tuned to achieve further improvements.

ACKNOWLEDGMENTS

This work was supported by Codoma.tech Advanced Technologies in the context of the Travición project[4], the Academy of Finland (MineSocMed Grant No 268078) and the Natural Science Foundation of China (Grant No 71402083).

REFERENCES

[1] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent Dirichlet Allocation. *Journal of Machine Learning Research* 3 (2003), 993–1022.
[2] Stéphane Clinchant and Eric Gaussier. 2009. Bridging Language Modeling and Divergence from Randomness Models: A Log-Logistic Model for IR. In *Proceedings of the 2Nd International Conference on Theory of Information Retrieval: Advances in Information Retrieval Theory*. Springer-Verlag, 54–65.
[3] Stéphane Clinchant and Eric Gaussier. 2010. Information-based Models for Ad Hoc IR. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 234–241.
[4] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. 1990. Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science* 41, 6 (1990), 391–407.
[5] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. 2013. *Applied Logistic Regression*. Vol. 398. John Wiley & Sons.
[6] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated Gain-based Evaluation of IR Techniques. *ACM Transactions on Information Systems* 20, 4 (2002), 422–446.
[7] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *CoRR* abs/1301.3781 (2013).
[8] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems*. 3111–3119.
[9] Kevin P. Murphy. 2012. *Machine Learning: A Probabilistic Perspective*. The MIT Press.
[10] Gerard Salton and Christopher Buckley. 1988. Term-Weighting Approaches in Automatic Text Retrieval. *Information Processing and Management* 24, 5 (1988), 513–523.
[11] Amit Singhal. 2001. Modern Information Retrieval: A Brief Overview. *IEEE Data Eng. Bull.* 24, 4 (2001), 35–43.
[12] Douglas Steinley and Michael J. Brusco. 2007. Initializing K-means Batch Clustering: A Critical Evaluation of Several Techniques. *Journal of Classification* 24, 1 (2007), 99–121.

[3] http://scikit-learn.org/

[4] http://www.travicion.com