Improving Free Text Recommendation Time by Means of Clustering Algorithms

Israel Mendonça^{a,*}, Antoine Trouvé^b, Kazuaki Murakami^b, Akira Fukuda^a

^aKyushu University, 744-Motoooka, Nishi-ku, Fukuoka, 819-0395, Japan

^bTeam Aibod, 1-8-7-Daimyo, Chuo-ku, Fukuoka, 810-0041, Japan

*Corresponding Author: Israel.m.san@gmail.com

Abstract

In this paper, we study the effects of applying clustering algorithms to free text recommendation systems. Usually recommendation systems do not scale well as the size of the recommendation space grows. One of the main techniques to scale them is by applying clustering, however clustering usually have a negative impact on the accuracy when applied without taking into consideration the recommended items. We construct a simple recommendation system for documents and propose partition its search space using kMeans. We vary the number of clusters and analyze how it affects performance in relation of recommendation time and accuracy. We apply a word-embedding-based technique to represent the document's bag-of-words, and therefore be able to compare how clustering algorithms performs in the task of partitioning these documents. One of the main findings of this work is that using clustering we can improve the recommendation time in almost 4 times without losing much off its initial accuracy. Another interesting finding is that the increment of the number of clusters is not directly translated into linear performance

Keywords: Clustering, Recommendation-Systems, Word-Embedding.

1. Introduction

Recommendation Systems are systems specialized in retrieving items from a database that may be of interest of users (top- n recommendation problem). A typical Recommendation System will collect information from an user inputs and match this information with a set of heuristics to fetch the items it believes will best suit the user needs. Efficient recommendation systems can retrieve meaningful items in a short amount of item, i.e., the accuracy and the

retrieval time of the system are widely used as metrics to evaluate recommendation systems.

The structure of the data directly influences the capacity of a system to analyze the items parameters and produce recommendations. However, natural text documents do not have a clear constitutional rule, and can vary greatly in terms of size, contents and internal text structure. Because of that, when one wants to create a recommendation system, she re-quires to apply techniques that are able to generate a structure that represents these free text documents. The most classical representation approaches are: Bag of words (BOW), in which each document is represented as the set of its internal words, with variations removing or not stop-words, and term frequency-inverse document frequency (tf-idf), in which each word in the document has its importance calculated by the frequency it appears in the text against the frequency it appears in the corpora.

Classical document representations often fail to capture the whole complexity that exists in the natural language text field, like synonyms or *polysemous* words (words with several meanings). Word2Vec models were introduced in 2013 by Mikolov, *et. al.* [1] with the main objective of generating some structure to represent words that could carry their se- mantic meaning. In their technique, words are represented as a high-dimensional vector, that are generated from the probabilities of their existence in a corpora. From their work, different authors employed word-embedding to calculate the distance between documents.

When we recommend documents by finding the closest documents in the search space, the amount of documents to compare directly influences the recommendation time. If we represent a document as a vector of n dimensions and we have d documents in our search space, the complexity of calculating the most similar items is defined as $O(n \times d)$. However, if we reduce this search space by a fraction k of its original value, the search time is also reduced by $\frac{n}{k}$. Yet,

dividing the search space into k fractions can hurt the accuracy of the system. For instance, the clustering algorithm could generate highly heterogeneous clusters, which could spread documents of the same classes among the clusters and reduce the probability of them being recommended when we provide recommendations from a single cluster.

In this paper, we apply a clustering technique to divide the search space of a recommendation system and calculate the influences on precision and recommendation time on this system. We use word-embedding to represent documents.

The rest of the paper is organized as follows. Section 2 presents a review of the literature regarding clustering and recommendation systems. Section 3 briefly explains some of the necessary background to better understand this work. Section 4 gives a more detailed explanation about the techniques we use in this paper. In Section 5, we introduce the datasets, the experimental setup, and the set of experiments conducted. In Section 6 we discuss the results of our experiments, we show that our methodology was able to improve the recommendation time by almost 4 times for our tested dataset. Finally, we conclude by highlighting the main points of our work, and identifying some possible perspectives.

2. Related Works

Cluster documents for reducing search time has been studied by different authors. On Cutting et. al. [2], and Hearst and Pedersen [3], clustering is used to build a table-of-contents of the documents collection and hence facilitate the browsing through the collection. Zamir and Etzioni [4] introduces a new clustering algorithm called the Suffix Tree Clustering (STC) which first identifies sets of documents that shares common phrases, and then clusters according to these phrases. Although we do not use the phrases for clustering, we do use the document's combined bag of word to generate the clusters.

Many authors use content-based clustering for retrieving information from the web [5]. On [6] the author ranks the search query and then generate results based on this rank, they then merge these results to generate a cluster. We also use the contents of the document to cluster the documents, but instead of clustering when we return the document, we pre-cluster them based on their distances.

To the best of our knowledge, we are the first authors to use a word-embedding document recommendation for recommendation systems.

3. Background

3.1 Word2Vec

Word2Vec is a word-embedding technique that was introduced by Mikolov et al. [1] and is based on the skip-gram model with sampling. They propose a simple neural network, with emphasis on efficiency at the training phase rather than precision. The neural network architecture consists of an input layer, a projection layer, and an output layer to predict nearby words. They train the vectors with the aim of maximizing the log probability of neighboring words within a specific corpus. More formally, given a sequence of words $w_1, \dots, w_T$, it tries to maximize the average log probability.

$$\frac{1}{T} \sum_{(t=1)}^T \sum_{-c \leq j \leq c, j \neq 0} \log p(w_j | w_t)$$

Where c is the size of the training context and $p(w_j | w_t)$ is the hierarchical softmax function of the associated words w_j and w_t . Since the model is usually trained over large datasets, the words representation are able to learn complex word relationships and also carry semantic meanings. The learning phase of word embedding is unsupervised and it can be computed on different corpus based on the user interest or pre-computed in advance. In this paper we use a pre-trained version of Google's word2vec [7] as our word embedder, but other embedding techniques are also available (Collobert and Weston, 2008 [8]; Mnih and Hinton, 2009 [9]; Turian et al., 2010 [10]).

3.2 K-Means

Clustering is the unsupervised classification of patterns into groups (clusters) [11]. The k-Means clustering is an iterative algorithm that initially assigns random groups to all the points, and then proceeds to its update step by computing the initial mean of the center each random assigned cluster. This update step will be run until the group's center doesn't varies much from each iteration. On termination the algorithm returns a list of each cluster it assigned to the data points.

More formally, given a set of dimensional training input vectors $x_1, x_2, \dots, x_n$ the k-Means clustering algorithm partitions the n training set into k sets of data points or clusters $S = S_1, S_2, \dots, S_n$, where $k \leq n$, such that the within cluster sum of squares is minimized. That is:

$$\operatorname{argmin}(s) \sum_{i=1}^k \sum_{x \in S_i} \|x - c_i\|^2$$

where c_i is the centroid or mean of data point in cluster S_i . The algorithm tries to improve the result by updating the centroids described in Equation 2.

4. Document Clustering for Recommendation Systems

4.1 A simple content-based recommendation system

In a content-based document recommendation system, when an input arrives, with respect to an input document, also called a query-document, a set of meta-data related to this query document is extracted and used by the system so it can, based on a set of internal beliefs, judge which items may be of the interest of the user. Then, the system needs to proceed to a search step in which it will look for its database for the items it intends to recommend. Figure 1 graphically displays the aforementioned steps.

In this work, we will use a function based on distance to rank items and make recommendations. The closer a document from the recommendation space is from the query document, the higher will be its rank on the system. To calculate the distance, we use the k-neighbors algorithm [12].

Since there is no straightforward mean of calculating the distance of two free-text documents, we had to generate a structure to represent documents, which provides us a tool to compare them. We use a technique based on word2vec to convert every document used in the system, both the query documents and the ones stored in the database. This technique is explained in details on Subsection 4.2.

Once we have means to compare documents, and a function that can rank documents for recommendation, the last thing that the system requires is a tool to fetch items that will be passed along the query document to the rank function and produce a recommendation. On Subsections 4.3 and 4 we will introduce the techniques we use in this work to retrieve documents from the database.

4.2 Document Representation

To be able to compare the distance of documents, we

generate a numeric representation of documents using word2vec. This allows us to compute the distance between documents using well-established mathematical techniques. Word2vec is largely used to extract the meaning of individual words and represent them as vectors. We try to extend this concept, by representing a document as a matrix M , in which each row of M contains the vectorial representation a word in the bag-of- words of the document.

Figure 2 displays how we used word2vec to represent a document. First, we convert the bag-of-words of the document into a matrix M of word-embedding with dimensions' $w \times n$, in which w is the number of words in a document and n is the number of dimensions of the word-embedding vector. From the matrix we extract the mean of each column of the matrix which results in a vector of length n . This document representation is expected to keep the topics of the document, and also provide a good base to compare document's distances.

4.3 Baseline Recommendation Characteristics

As stated in Subsection 4.1, to be able to perform recommendations, the system needs to fetch documents into a database. Our baseline approach will fetch the whole database when making a recommendation.

To have an idea of the efficiency of this approach, we can calculate the complexity of the search on this set. Since we need to compare n documents, represented as vectors v , the complexity of the baseline algorithm is $O(n \times v)$. Also, we need to perform a sorting operation after the distances are calculated. We use Timsort, which has complexity $O(n \times \log(n))$ for the average and worst case. So, the total complexity of a recommendation for our baseline case is $O(n \times v) + O(n \times \log(n))$.

4.4 Our approach Characteristics

In our approach, we try to reduce the search time by clustering the results and searching on a fraction of the recommendation space.

Since we need to draw a list of candidates from the recommendation space, if we reduce the number of candidates on the search, it will reduce the amount of calculations the system needs in order to make a recommendation. From a complexity point of view, when we divide the search space equally into c different clusters, the

Fig. 1. A simple recommendation. The system combines the query document and the database document into a rank function to produce recommendations.

search complexity is reduced to exactly $O(\frac{n \times m}{c})$. Which can be translated into a theoretical speedup of c times for search. Also, the sorting step would also be improved since it has $\frac{n}{c}$ items to compare now.

In our approach, we decided to use kMeans to cluster the whole train data. Along to the file, we will also store the label in which the document belongs. We added a new step to the recommendation, which assign a cluster to the query document based on its contents. The system will retrieve only a subset of documents from the database, that belongs to the same cluster as the query document, and proceed to the distance calculation and sort steps on this subset. We expect this operation to improve the recommendation time for our system.

Fig. 2. Document Representation. First, the document is converted to a matrix of its bag-of-words, then we convert it into a single vector.

5. Dataset and Experimental Setup

This section will give details of the dataset we used as well as the experimental setup we used for our experiments.

5.1 Dataset

In the remainder of this article we use the Reuters-22172 [13] dataset to evaluate the performance of the recommendation system. Reuters is a classic news dataset labeled by news topics (we use the 8-class version). It contains 8 classes, 5485 training documents and 2189 test documents. We will evaluate the quality of the recommendations based on whether or not the recommended documents are in the same class as the query document.

5.2 Experimental Setup

To test our model, we created a reference implementation of our method. We created a script that convert all the documents from a folder to the representation explained before and store them in a database. Then another script clusters those documents, and store the model that assign clusters to the file system. The information related to the cluster that each train document belongs is then stored into the database. Finally, we run another script that takes the whole test set, convert each document to the representation introduced before and then calls a recommendation routine for each of them. The recommendation routine is responsible for reading the query document, and using the cluster model to assign a cluster to this document. It then retrieves all the documents from the assigned cluster from the database. It will then calculate the distance between each document and the query document, sort the results, and return the k first documents.

To extract the bag-of-words of a document, we used the NLTK's sentence tokenizer and extracted all the words from sentences. For all the steps that required clustering, we used the sci-kit learn's version of KMeans. For the steps that required database for storing documents and labels, we used MongoDB.

Regarding word-embeddings, we download the pre-trained one from Google's webpage. To load the model and furthermore be able to convert words to its respective embeddings, we used Gensim, an open-source tool that can be used to load word pre-trained embedding and converts words to an embedding vector.

All the code necessary to run those experiments, and all the experimental results introduced on the next sections is public and available at github1.

Our experiments were run in a Ubuntu 17.10 virtual machine hosted in a dedicated 10-core 3.2GHZ Intel i7-6900K with 16Gb of memory.

6. Experimental Results

In this section, we present the experimental results for the system and dataset introduced in Section 5. We analyze how the performance is affected in terms of time and accuracy.

For the data described in Subsection 5.1, we recommend 10 documents for each document of the test set. This means that for our test set of 2189 documents, we will make 2189 recommendations, each of them returning up to 10 documents.

6.1 Time Efficiency

We start our analysis by evaluating how much clustering influences recommendation time. Figure 3 displays the sum of each individual recommendation time for against the number of clusters. For our first case, the baseline in which all the documents are grouped into a single cluster, the total recommendation time is 795 seconds, which averages a time of 0.36 second per recommendation.

After the baseline, we increase the number of clusters from 8 to 104 (with steps of 8) and display their time. We notice that by using clustering only, regardless of the number of selected clusters, we have a positive impact over the recommendation time, as expected. We can also see that although the total recommendation time keeps decreasing from 8 clusters until we have 24 clusters, the recommendation time of the system starts to increase until we reach 48 clusters, after that it reduces to 258 seconds and stabilizes on numbers near 270 seconds for the rest of the experiment.

Despite having significant improvements in terms of time, reaching a maximum speedup of 3.83 when we had 24 clusters, this speedup is near half of the theoretical linear speedup. To better understand the effects of the partition configuration over the time, we take a closer look into the configuration of the partitions.

Fig. 3. Reduction of the size of the document representation by means of using our model. The model is able to reduce the size of the corpora from a 3D matrix to a 2D one. 129

6.2 Assessing Cluster Partitions

Figure 6.2 shows the distribution of members among the clusters for the following cluster sizes: 8, 24, 40 and 104 clusters. The ideal linear speedup is obtained when we have the same amount of documents inside each cluster, however as Figure 6.2 shows, the distribution of elements is not even among the clusters, which causes an imbalance on the recommendation time. Small clusters yield shorter execution times, while larger clusters have recommendation time closer to the base-line. In the end, the overall recommendation time is heavily influenced by the size of the clusters selected for the search stage.

To reason over the difference in time of a small and a big cluster, we analyze the recommendation time of each cluster size for two out of four cases aforementioned. Figures 5 and 6 displays the recommendation time separated by cluster size for each kMeans version. The recommendation time grows as the number of elements inside a cluster grows. For 8 clusters, we can see that the bigger cluster has 2 times the size of the smallest one, but it can reach up to 4 times the recommendation time. For 24 clusters, this difference cannot be so easily perceived because the clusters have a more similar size, and consequently a more similar recommendation time.

Fig. 4. Documents distribution for each individual cluster of kMeans. The cluster size parameter is varied and is 8(a), 24(b), 40(c) and 104(d). The higher the number of clusters passed as parameter, the more distributed the individuals should be among the clusters.

6.3 Effects on Precision

We compute the precision by calculating the amount of correct guesses inside the set of recommended items. If we recommend 10 items, and 8 are from the same label as the query document, we say that the recommendation had 80% of precision.

Since we artificially separated the data, we expected the precision to be negatively impacted. However, the impact on the precision were minimal. The results for the baseline alongside with all the cluster sizes is displayed on Table 1.

Regardless of the cluster size, the precision is not strongly affected. From the table we can draw two important results: The first one regards the baseline representation, which yield 90% precision, which means that the document representation that we chose, was sufficient to capture the distinction that separates documents for this dataset; The second observation concerns the fact that the precision did not change significantly as the number of clusters changed, this means that kMeans was able to take advantage of the document representation and generate highly accurate partitions.

Fig. 5. Recommendation time for every cluster size generated using kMeans with 8 clusters. We can see a pattern that the recommendation time increases as function the cluster size.

7. Conclusion

In this paper, we have studied the effects of adding clustering when doing recommendations of text files. Our main goal was to reduce the search time, without greatly affecting the performance. We investigated how different

number of clusters for kMeans can affect the performance in relation of time and accuracy. We empirically tested our hypothesis using a real- world dataset and assessed that the time can be improved up to almost 4 times when using a correct configuration of clustering. We also verified that disregarding the number of clusters, using kMeans to partition the documents usually yields improvements in terms of time, without affecting much of the accuracy.

Fig. 6. Recommendation for every cluster size generated using kMeans with 24 clusters. We can see a pattern that the recommendation time increases as function of the cluster size.

Table 1. Precision, Recall and F1-Score.

# Clusters	Precision (%)	Recall (%)	F1-Score
1 (Baseline)	90.65	1.00	0.02
8	90.36	1.63	0.03
16	90.74	3.31	0.06
24	90.62	4.16	0.08
32	90.48	4.07	0.08
40	90.77	4.97	0.09
48	90.62	5.80	0.11
56	90.53	5.67	0.11
64	90.84	7.47	0.14
72	90.66	7.45	0.14
80	90.61	7.81	0.14
88	90.79	7.08	0.13
96	90.58	8.76	0.16
104	89.02	8.17	0.15

Acknowledgment

The authors are grateful to the National Council of Scientific and Technological Development (CNPQ) for its financial support through the Science Without Borders program, see (<http://www.cienciasemfronteiras.gov.br>).

References

- (1) Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean, “Efficient estimation of word representations in vector space,” arXiv preprint arXiv:1301.3781, 2013.
- (2) Douglas R. Cutting, David R. Karger, and Jan O. Pedersen, “Constant interaction-time scatter/gather browsing of very large document collections,” in Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, New York, NY, USA, 1993, SIGIR ’93, pp. 126–134, ACM.
- (3) Marti A Hearst and Jan O Pedersen, “Reexamining the cluster hypothesis: scatter/gather on retrieval results,” in Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval. ACM, 1996, pp. 76–84.
- (4) Oren Zamir and Oren Etzioni, “Web document clustering: A feasibility demonstration,” in Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval. ACM, 1998, pp. 46–54.
- (5) Mei Kobayashi and Koichi Takeda, “Information retrieval on the web,” ACM Computing Surveys (CSUR), vol. 32, no. 2, pp. 144–173, 2000.
- (6) Hua-Jun Zeng, Qi-Cai He, Zheng Chen, Wei-Ying Ma, and Jinwen Ma, “Learning to cluster web search results,” in SIGIR 2004: Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Sheffield, UK, July 25-29, 2004, 2004, pp. 210–217.
- (7) Google inc., “word2vec,” 2013, <https://code.google.com/archive/p/word2vec/>.
- (8) Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa, “Natural language processing (almost) from scratch,” Journal of Machine Learning Research, vol. 12, no. Aug, pp. 2493–2537, 2011.
- (9) Andriy Mnih and Geoffrey E Hinton, “A scalable hierarchical distributed language model,” in Advances in neural information processing systems, 2009, pp. 1081–1088.
- (10) Joseph Turian, Lev Ratinov, and Yoshua Bengio, “Word representations: a simple and general method for semi-supervised learning,” in Proceedings of the 48th annual meeting of the association for computational linguistics. Association for Computational Linguistics, 2010, pp. 384–394.
- (11) Anil K Jain, M Narasimha Murty, and Patrick J Flynn, “Data clustering: a review,” ACM computing surveys (CSUR), vol. 31, no. 3, pp. 264–323, 1999.
- (12) Naomi S Altman, “An introduction to kernel and nearest-neighbor nonparametric regression,” The American Statistician, vol. 46, no. 3, pp. 175–185, 1992.
- (13) D Lewis, “Reuters-21578,” 1997, <https://archive.ics.uci.edu/ml/datasets/Reuters-21578+Text+Categorization+Collection>.