Scalable Recommendation of Wikipedia Articles to Editors Using Representation Learning

Oleksii Moskalenko
Ukrainian Catholic University
Lviv, Ukraine

Denis Parra
Pontificia Universidad Catolica de Chile & IMFD
Santiago, Chile

Diego Saez-Trumper
Wikimedia Foundation
San Francisco, USA

ABSTRACT

Wikipedia is edited by volunteer editors around the world. Considering the large amount of existing content (*e.g.* over 5M articles in English Wikipedia), deciding what to edit next can be difficult, both for experienced users that usually have a huge backlog of articles to prioritize, as well as for newcomers who that might need guidance in selecting the next article to contribute. Therefore, helping editors to find relevant articles should improve their performance and help in the retention of new editors. In this paper, we address the problem of recommending relevant articles to editors. To do this, we develop a scalable system on top of Graph Convolutional Networks and Doc2Vec, learning how to represent Wikipedia articles and deliver personalized recommendations for editors. We test our model on editors' histories, predicting their most recent edits based on their prior edits. We outperform competitive implicit-feedback collaborative-filtering methods such as WMRF based on ALS, as well as a traditional IR-method such as content-based filtering based on BM25. All of the data used on this paper is publicly available, including graph embeddings for Wikipedia articles, and we release our code to support replication of our experiments. Moreover, we contribute with a scalable implementation of a state-of-art graph embedding algorithm as current ones cannot efficiently handle the sheer size of the Wikipedia graph.

KEYWORDS

Wikipedia, RecSys, Graph Convolutional Neural Network, Representation Learning

1 INTRODUCTION

Wikipedia is edited by hundreds of thousands of volunteers around the world. While the level of expertise, motivations, and time dedicated to that task varies among users, most of them experience challenges in deciding which articles to edit next. For example, many experienced users have huge backlogs[1] of work with a large number of articles to improve or review. Prioritizing the articles in this backlog, as via a personalized article ranking system, would potentially be of great help for these editors. On the other hand, newcomers might experience difficulties deciding what do to after their first contribution, as evidenced by the many efforts to understand how to improve the retention of newcomers [5, 22].

While previous work on recommendations in Wikipedia has focused on finding articles for translation [32] or content to be added to existing articles [26], there are still important unsolved problems: i) creating a *scalable* recommender system that can deal efficiently with the large number of Wikipedia editors (over 400K monthly just in the English Wikipedia) and articles [12] ii) having good coverage of articles beyond just the most popular articles, and, iii) being able to provide good recommendations for newcomers, facing the classical user cold-start problem [30].

To address these problems, we have created an efficient and scalable implementation of a state-of-art convolutional graph embedding algorithm [14] that is able to deal with the large Wikipedia article graph. We combine this with a document embedding model that allows us to learn representations for articles and editors, and does not require retraining when new users are added in the system. With only a few edited articles then, the system is able to produce personalized recommendations, similar to Youtube deep recommendation model [7]. We test our algorithm in English Wikipedia (the largest one with almost 6 million articlesshowing that we can overcame well-established content-based filtering methods as well as collaborative filtering approaches. Moreover, we evaluate our recommendation measuring the top-100 items, to support a robust evaluation against popularity bias [31].

In summary, the main contributions of this paper are: *(i)* Introduce a model which learns representations (graph and content-based) of Wikipedia articles and makes personalized recommendations to editors; *(ii)* Evaluate it with a large corpus, comparing with competitive baselines *(iii)* and release a scalable implementation of GraphSage, a state-of-art graph embedding system, that in previous implementations was unable to deal with the large graph of Wikipedia page[2].

2 RELATED WORK

There are several projects trying to solve the task of recommending items to users at real-world scales of millions of users and millions of items. For instance, Ying et al. for Pinterest [34] created an extension of GraphSAGE [14], a type of Graph Convolutional Network [18] (GCN); researchers at YouTube [7] built a system based on regular deep neural networks that jointly learns users' and items' representations from users' previous history of views. However, in both examples, the model learns in a supervised setup, whereas we lack a sufficiently comprehensive dataset of previous interactions because 94% of Wikipedia contributors are associated with less that 10 interactions in last 3 years [12]. eBay's recommendation system covers a similar gap by using TF-IDF for similar item search, which does not require training [4].

On the document representation task, we can highlight several approaches: Doc2Vec[19] is method for obtaining content-based

[1] https://en.wikipedia.org/wiki/Wikipedia:Backlog

[2] https://github.com/digitalTranshumant/WikiRecNet-ComplexRec2020

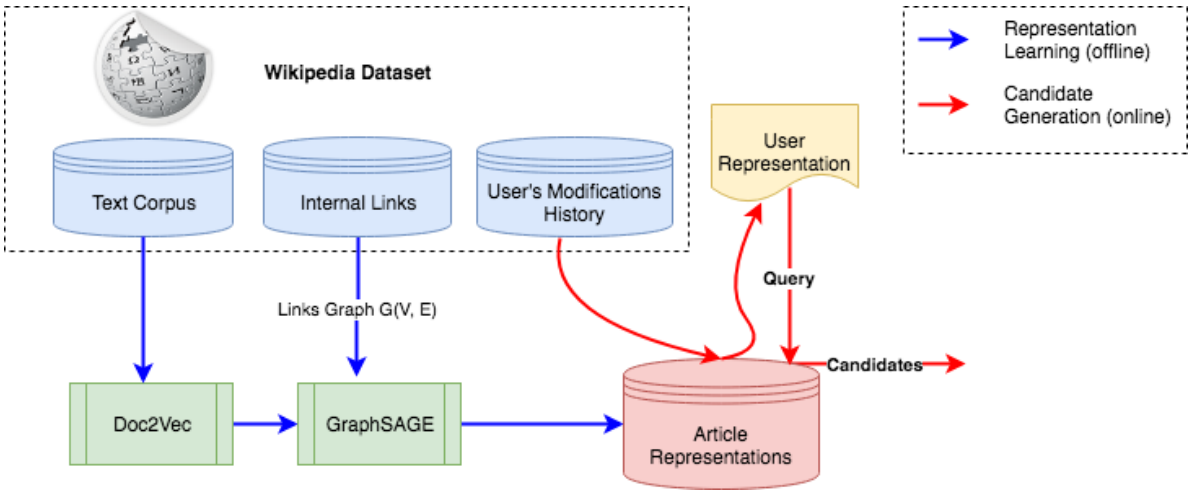

Figure 1: Flow of Candidate Generation for Wikipedia articles recommendation: Doc2Vec embeddings are trained on Wiki Text Corpus and then passed as input into GraphSAGE model. Received articles' representations are then used in Nearest Neighbors Search to produce candidates

representations of paragraph or longer text in vector space. However, one main advantage of our dataset is the availability of structural knowledge [6] - i.e. links among articles that could potentially tell more about the article beyond its content. Those links can be represented as a graph, where nodes are articles and edges are links between them. Thus, the task of learning document representations can be transformed into learning the representation of a node in the graph. Node2vec [13] is a recent approach to learn such a representation. However, its scalability is still limited [35] and the main drawback for our use case is the necessity of full retraining after changes in the structure of the graph. Node2vec also omit the content part of articles (node features), which is a huge part of our dataset.

GCN [10, 18] are a recent approach to solve many machine learning tasks like classification or clustering of a graph's nodes via a message-passing architecture that uses shared filters on each pass. It combines initial node features and structural knowledge to learn comprehensive representations of the nodes. However, the original GCN architecture is still not applicable to large-scale graphs because it implies operations with a full adjacency matrix of the graph. To tackle these limitations, GraphSAGE model was introduced [14] in a way that only some fixed-sized sample of neighbors is utilized on the convolutional Layer. Because of the fixed-size samples, we also have fixed-sized weights that are generalized and could be applied to a new, unknown part of the graph or even completely different graph. Thus, with inductive learning, we can train the model on a sub-graph, which means less computation resources are required, and evaluate generalization on the full graph.

3 WIKIRECNET DESCRIPTION

Here we introduce *WikiRecNet*, a scalable system for providing personalized article recommendations in Wikipedia, built on top of GCN and Doc2Vec. The design of our solution is inspired by a classic *Information Retrieval* architecture. First we represent users by the articles that they edited, then we generate a list of candidates from the article pool by comparing that user representation with the article representations. Next, we sort the article candidates accordingly to the user preferences and generate a list of *top-n best candidates* recommendation.

3.1 Article and user representation

The primary challenge for our system is producing good user and article representations. It is an especially big problem for user representation since most of Wikipedia contributors do not fill any additional information about themselves except their login credentials [3], and around 28% of all revisions in our English Wikipedia dataset, are done by anonymous users [12]. The only useful information that could uniquely characterize the user is the history of his editions. Hence, most of our efforts were dedicated to learning articles' representations, and then representing the user based on the articles edited. One effective approaches to construct good user and item representations is to learn them with recommendation supervision [7, 34]. However, it is not possible to follow this approach due to the lack of the required comprehensive-enough dataset of previous interactions. History of users editions in Wikipedia is far from exhaustive (88% of users of English Wikipedia have done less than 5 major editions [12]) and too sparse, in a way that it is hard to model user's area of interest. Therefore, the additional challenge is to conduct representation learning [2] in an unsupervised way in relation to our final task.

3.2 Candidate Generation

Similar to YouTube's deep learning recommender [7], *WikiRecNet* first generates candidates for a final personalized ranking in a second stage. To generate candidates we first calculate representation vectors (content-based and graph-based) for all articles in our

[3] https://en.wikipedia.org/wiki/Wikipedia:Wikipedia_is_anonymous

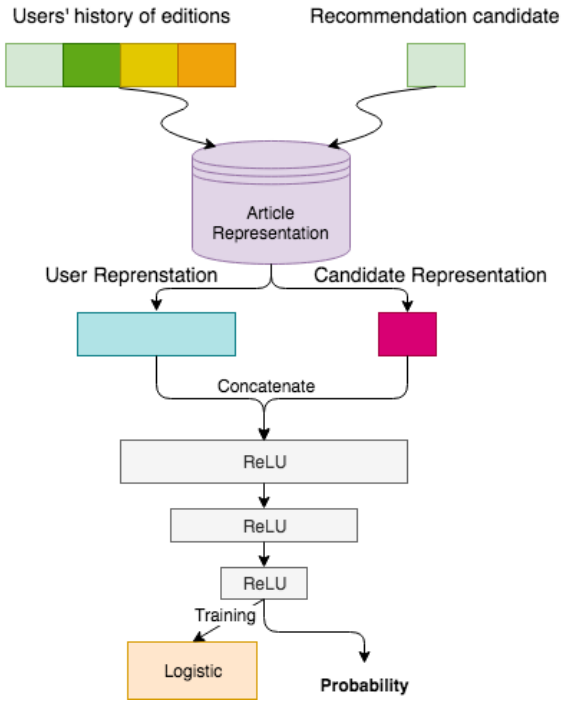

Figure 2: Candidate Ranking: user history along with candidate are passed through articles' representation database (Embedding Layer) and then through several fully-connected layers to train in the log-regression setup.

Table 1: Performance of different algorithms for K-NN search. All tests were conducted with English Wikipedia articles ($|V| = 5,251,875$). Setup is measured in seconds. Secs. req. means seconds per request.

Algorithm	Setup	Secs./req.	Recall	MRR
Exact search	3.91	0.81	0.224	0.0220
IVF	207.02	0.07	0.206	0.0212
HNSW	232.68	0.04	0.224	0.0220
LSH	472.31	0.15	0.215	0.0219

Table 2: Specifications of built Wikipedia Graph

Specification	English Wikipedia
Amount of vertices ($\|V\|$)	5,251,875
Amount of Edges ($\|E\|$)	458,867,626
Average Degree ($\overline{d_{all}}$)	174
Median Degree ($\widetilde{d_{all}}$)	60
Approx. Diameter (D)	23
Amount of labeled nodes	4,652,604

dataset, a process conducted off-line which is presented as *Representation Learning* in Figure 1. Then, for every user we define her representation as an aggregation of representation vectors of corresponding articles that were edited by this user. Next, we conduct Nearest Neighbors search with user representation as a *query* in the articles' representation database, a procedure we call *Candidate Generation* and which is conducted online, as shown in Figure 2.

Content-based articles representation: Doc2Vec. For learning the content-based article representation, text features are needed to be extracted first. This can be conducted with traditional document vector space model [29] or by using word embeddings such as Word2vec [21] and GloVe [25] and performing an additional step of aggregation. Another option is using directly a full text embedding model and with that goal we use Doc2Vec [19]. There are two distinct approaches for learning document embedding with this model. One is Paragraph Vector Distributed Bag-of-Words model (PV-DBOW) model, which is based on word2vec's Continuous Bag-of-Words approach [21] but instead of word input it accepts paragraph vector and predicts context words for this paragraph. In the second approach, Distributed Memory (PV-DM), which is based on word2vec skip-gram model, the model predicts middle word based on context and paragraph vector given as input. Later on this paper (Section 5) we show that PV-DBOW is the best fit for our task. We train Doc2Vec-DBOW model on the Corpus of all Wikipedia articles in a given language. Output vectors of Doc2Vec are being passed as input features to the GNC.

Graph-based article representation: GraphSAGE. GraphSAGE has been used as GCN due to its ability to learn with an inductive approach and construct embeddings for unseen nodes. During the pre-processing of the input dataset –snapshot of Wikipedia Dataset– we create a graph $G(V, E)$ where V denotes the set of articles, and E the set of links between them. GraphSAGE utilizes structural knowledge from graph G and produces new vectors that preserve both text and structural representations. Due to the inductive nature of GraphSAGE architecture, we do not need to retrain the model every time after adding a new article into the database, this is very important for applying *WikiRecNet* in real scenarios, where new articles are constantly added [12].

After producing the document vector and updating the Graph G structure, we can run GraphSAGE model as is, with already trained weights. GCN is a multi-layer network, where each layer can be formulated as:

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l+1)})$$

where $\tilde{A} = A + I$ is the adjacency matrix with self-connections (I), $\tilde{D} = \sum_j \tilde{A}_{ij}$, W are trainable weights and H is the output of previous layer or $H^{(0)} = X$ is input, X represents node features. An intuitive explanation of this process is that each node collects features of its neighbors that were propagated through trainable filters (convolutions) so called, message passing. On each step node collects knowledge of its neighborhood and propagates its state further on the next step. Thus, properties of 1st, 2nd, ..., nth proximity are being incorporated into node's state along with preserving original features of node's community.

Optimizing candidate retrieval. In serving time, recommendation candidates will be produced by applying K-Nearest Neighbors (K-NN) search to find the most similar articles to the user representation vector in the pre-computed database of all articles' representations. K-NN search is one of the main parts of candidate article

generation, since its performance in terms of time and resource consumption is very critical for online recommendation in a high-load system. We conducted experiments with different optimizations for K-NN candidate search using FAISS library [16] : Locality-Sensitive Hashing (LSH), Inverted file with exact post-verification (IVF), Hierarchical Navigable Small World graph exploration (HNSW). Our tests showed that HNSW gives the best speed along with exactly the same recall and MRR as exact search, so with no trade-off in performance we achieved 20x times improvement in speed. Results of these experiments are shown in Table 1.

3.3 Ranking of Candidate Articles

After learning content and graph-based representations for Wikipedia articles, in the second part of our system we are trying to model user preferences based on the previous edit history of Wikipedia contributors. With given previous editions and articles, we produce a relevant a list of candidates, ranked by its relevance for a given user. Our model is trained on binary labels - *relevant / not relevant* (logistic regression), as shown in Figure 1, but on serving time it will produce probabilities of user interest, which are used as a preference ranking score.

This approach is inspired by Pointwise ranking [20] and is implemented in many similar recommender systems: YouTube[7], eBay[4]. The model is shown on Figure 2 and consists of several fully-connected layers with Batch Normalization and ReLU activation after each layer except for the last layer, where sigmoid activation is used. The final model's architecture was selected as follows: 1024 ReLU -> 512 ReLU -> 256 ReLU. As input model accept a concatenated vector of user and candidate representations.

Preference score We define our preference ranking score as the probability that a user u finds a wikipedia article a_i relevant:

$$score(u, a_i) = P(a_i = relevant|u) = \frac{1}{1 + e^{-\Phi(a_i,u)}} \quad (1)$$

where u represents the user, a_i a candidate wikipedia article, A is the set of all articles to be ranked, and $\Phi(\cdot)$ a weighted sum of the values in the last network layer of the *Candidate ranking* neural network, shown in Figure 2. We train the model with a traditional loss for a binary logistic regression.

4 EXPERIMENTS

We worked with the English version of Wikipedia not only because is the most popular one. In addition, is the most challenging in terms of data processing, and if our system is able to deal with the largest Wikipedia it would be easy to apply later in smaller projects.

4.1 Dataset

All data used has been downloaded from the official Wikimedia Dumps [11] which are snapshots of the full Wikipedia. Some of the objects in the dump (like articles' links) are stored in SQL format, others, with deeper structure (like articles' text) are stored in XML.

First of all, for representation learning we built a graph $G(V, E)$, where the set of nodes V is the set of all Wikipedia pages belonging to article namespace[4] and E is the set of directed links between them. The SQL dumps of *page*, *pagelinks*, and *redirects* tables were parsed to organize this data. During pre-processing stage, all links to redirect pages[5] were replaced by their actual destinations. "Category pages", that consists only of links to other pages and do not have their own content, were detected and filtered out. We used Apache Spark and GraphX for parallel parsing of SQL dumps and discovering and cleaning Article Graph respectively. The output Graph was converted into binary format with graph-tool [24] to achieve fast processing (see Table 2). For extracting the articles' texts we took the latest revision[6] per each article from XML dump. We used *Gensim* [27] to tokenize and lemmatize text and prepare for the Doc2vec training.

To the facilitate the evaluation in an end-to-end fashion we reorganized the data into a *revisions-per-user* dataset. Only revisions that were created after Jan. 2015 were kept in this dataset, so our recommendations that are based on the latest snapshot of article graph (Oct. 2018) will not recommend too many articles that did not exists on that moment.

We found that 88% of contributors are not regular users, since they edited fewer than 5 different articles for selected dates. We also calculated diversity [3] of users' contribution based on vector representations obtained from Doc2vec. The set of contributors that fits to our needs has mostly edited from 5 to 40 different articles, though diversity of those articles is rather high. That is the main cause our representations cannot be trained against this data like it was done in previous work [7]. Unlike the work by Covington et al. [7], our training dataset is small (around 60K users) and users' areas of interest are very diverse.

4.2 Training

For all training experiments with GraphSAGE we generated a sampled adjacency matrix based on Articles' graph G. The width of the adjacency matrix was determined by our experiments, based available memory resources as well as on graph statistics (Table 2). We selected 128 as maximum amount of neighbors in this matrix. If a node had more than that, then we used random subsampling. Our GCN architecture consists of two convolutional layers. On each convolutional step we picked a random sample of 25 neighbors from this adjacency matrix. This 25-neighbors sample is being resampled on each new batch. For better generalization we used a batch size of 512, since experiments with dropout between convolutional layers led to no improvement in generalization. We set the size for all output vectors to 512 considering a balance between better resolution and available memory.

Document representations from Doc2Vec-DBOW, trained with vector size 300 and window size 8, were passed as initial node states and graph edges played the role of labels when the model was trying to predict those edges. We utilized max-margin loss [34], as target for training GraphSAGE in link-prediction setup. Model parameters were tuned with the Adam optimizer[17].

$$J(z_u z_i) = E_{v_n \sim P_n(u)} max\{0, z_u \cdot z_{v_u} - z_u \cdot z_i + \Delta\} \quad (2)$$

For training the ranking model, a dataset from users' history was constructed. As input this model takes 5 articles edited by a user (representing users' preferences) and 1 candidate that might

[4] https://en.wikipedia.org/wiki/Wikipedia:Namespace

[5] https://en.wikipedia.org/wiki/Wikipedia:Redirect

[6] Article's revision is a specific version of article's content after each *modification*

interest the user. The model tries to predict the probability of relevance of this candidate to the current user. Those 6 input articles are passed through an *Embedding Layer* populated with representations received from GraphSAGE and then concatenated into one vector. We chose positive candidates from actual user history and generated negative candidates with kNN search on constructed articles' representations. Logistic (binary cross-entropy) regression with class-weights (due to high class imbalance) was used as loss function.

4.3 Evaluation

To prepare the evaluation dataset, we subsampled windows of size 10 from user's history (from users that were not previously used for training or testing the Deep Ranking model). Our assumption if that the first 5 articles denoted users' area of interest. To compute a single user vector we took element-wise average of representations from the first 5 articles (GraphSAGE representations). We were trying to predict the rest 5. Algorithm can be expressed as follows: (i) take first 5 articles per user. Calculate average of their embeddings vectors, output this as the user vector representation, (ii) generate candidates by nearest neighbors search of user representation, (iii) sort candidates according to ranking algorithm and select the top K. In our evaluation we compare two ranking techniques: sort by cosine similarity, and sort by probability from Deep Ranking model, and (iv) compare Top-K recommendations with the 5 articles in the test set (from second half of the sampled window).

To measure the results we used several metrics: mean average precision (MAP), normalized discounted cumulative gain (nDCG)[1] and Recall@k [8]. We calculate these metrics at high k values, $k = 50$ an $k = 100$. Unlike traditional research on top-k recommendation systems usually focusing on small k values (k=10,20,30), we are specially interested in preventing popularity bias, i.e., having WikiRecNet biased to recommend mostly popular items. Valcarce et al. [31] showed recently that usual top-k ranking metrics measured at higher values of k (50, 100) are specially robust to popularity bias, and that is why we use them here.

.

5 RESULTS

Results of the evaluation are presented in Table 3. We first describe the competing methods:

Baselines. We used two well established methods. The first one is BM25[28], a probabilistic method used on information retrieval but also applied for content-based filtering in the area of recommendation [23]. A second baseline is implicit feedback collaborative filtering optimized with Alternative Least Squares (ALS) [15].

K-NN recommender. In addition, we implemented a simple K-NN recommender where the Wikipedia articles are represented by the Doc2vec vector embeddings. Each user u is represented by the articles she has edited, and we test two forms of aggregation to represent the user model: merging the user-edited articles (*merge*) and calculating the mean at each dimension of the document (*mean-pool*). We rank recommended articles by cosine similarity.

Aggregations. Finally, *WikiRecNet* is presented in 5 versions by varying the type of aggregation of articles to represent the user model (merge, mean-pool, max-pool), as well as the method for ranking (cosine similarity and Deep-Rank).

The results in Table 3 show that WikiRecNet, using *merge* aggregation and *Deep-Rank* ranking, outperforms the other methods in all metrics. We highlight the following the aspects in the evaluation:

- ALS implicit feedback collaborative filtering performs the worst among all methods. This result must be due to the extreme high sparsity of the dataset.
- BM25, despite being a simple and traditional content-based filtering method, performs well and remains very competitive.
- The simple K-NN based on Doc2Vec representation performs better than ALS, and mean-pool reports better results than merge but only at higher ranking positions (MAP@50, nDCG@50, Recall@50).
- Among the WikiRecNet variations, the max-pool aggregation seems to be the least helpful. In terms of nDCG@50 and nDCG@100 (the metric most robust to popularity bias [31]), merge aggregation seems more effective than mean-pool, and then the combination with DeepRank produce the best performance, with a 100% increase compared to the Doc2vec mean-pool reference method.

6 CONCLUSION

In this article we have introduced *WikiRecNet*, a neural-based model which aims at recommending Wikipedia articles to editors, in order to help them dealing with the sheer volume of potential articles that might need their attention. Our approach uses representation learning, i.e., finding alternative ways to represent the Wikipedia articles in order to produce a useful recommendation without requiring more information than the previous articles edited by targeted users. For this purpose, we used Doc2Vec [19] for a content-based representation and GraphSage [14], a graph convolutional network, for a graph-based representation.

WikiRecNet architecture is composed of two networks, a candidate generation network and a ranking network, and our implementation is able to deal with larges volumes of data, improving existing implementations that were not capable to work in such scenarios. Also, our approach does not need to be retrained when new items are added, facilitating its application in dynamic environments such as Wikipedia. To best of our knowledge, this is the first recommender system especially designed for Wikipedia editors that takes in account such applications constrains, and therefore, can be applied in real world scenarios.

In order to contribute to the community, we provide our code and the graph embedding of each Wikipedia page used in this experiment[7] available in a public repository, as well as a working demo that can be tested by the Wikipedia editors community[8]. With respect to text embeddings, there have been important progresses in the latest years, so another idea for future work will be testing models like BERT [9] or XLNet [33].

[7] Embeddings in other languages would be also available under request.
[8] https://github.com/digitalTranshumant/WikiRecNet-ComplexRec2020

Table 3: Offline evaluation of generated recommendations on the task of predicting next 5 articles edited by user with percentage improvement over content-based model Doc2Vec (mean-pool) with cosine similarity.

			K=50			K=100		
Model	Aggregate	Rank	MAP	nDCG	Recall	MAP	nDCG	Recall
WikiRecNet	mean	cosine	0.0221	0.1361	0.0846	0.0238 (+78%)	0.1468 (+66%)	0.1179 (+99%)
	mean	deep-rank	0.0228	0.1363	0.0841	0.0243 (+82%)	0.1493 (+70%)	0.1134 (+92%)
	max	cosine	0.0192	0.1196	0.0672	0.0206 (+54 %)	0.1299 (+47%)	0.0923 (+56%)
	merge	cosine	0.0208	0.1412	0.0825	0.0227 (+70%)	0.1538 (+75%)	0.1175 (+99%)
	merge	deep-rank	**0.0262**	**0.1625**	**0.0935**	**0.0282 (+111%)**	**0.1760 (+100%)**	**0.1302 (+120%)**
Doc2Vec	merge	cosine	0.0085	0.0805	0.0438	0.0092	0.0883	0.0600
	mean	cosine	0.0126	0.0821	0.0436	0.0133	0.0880	0.0590
BM25			0.0251	0.1602	0.0921	0.0273	0.1710	0.1290
ALS MF			0.0027	0.0163	0.044	0.0063	0.0204	0.0609

7 ACKNOWLEDGMENTS

The author Denis Parra has been funded by the Millennium Institute for Foundational Research on Data (IMFD) and by the Chilean research agency ANID, FONDECYT grant 1191791.

REFERENCES

[1] Linas Baltrunas, Tadas Makcinskas, and Francesco Ricci. 2010. Group Recommendations with Rank Aggregation and Collaborative Filtering. In *Proceedings of the Fourth ACM Conference on Recommender Systems (RecSys '10)*. ACM, New York, NY, USA, 119–126. https://doi.org/10.1145/1864708.1864733

[2] Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2013. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence* 35, 8 (2013), 1798–1828.

[3] J Bobadilla, Francisco Serradilla, and J Bernal. 2010. A new collaborative filtering metric that improves the behavior of recommender systems. *Knowledge-Based Systems* 23 (08 2010), 520–528. https://doi.org/10.1016/j.knosys.2010.03.009

[4] Yuri M. Brovman, Marie Jacob, Natraj Srinivasan, Stephen Neola, Daniel Galron, Ryan Snyder, and Paul Wang. 2016. Optimizing Similar Item Recommendations in a Semi-structured Marketplace to Maximize Conversion. In *Proceedings of the 10th ACM Conference on Recommender Systems (RecSys '16)*. ACM, New York, NY, USA, 199–202. https://doi.org/10.1145/2959100.2959166

[5] Boreum Choi, Kira Alexander, Robert E Kraut, and John M Levine. 2010. Socialization tactics in wikipedia and their effects. In *Proceedings of the 2010 ACM conference on Computer supported cooperative work*. ACM, 107–116.

[6] Cristian Consonni, David Laniado, and Alberto Montresor. 2019. WikiLinkGraphs: A complete, longitudinal and multi-language dataset of the Wikipedia link networks. *arXiv preprint arXiv:1902.04298* (2019).

[7] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems (RecSys '16)*. ACM, New York, NY, USA, 191–198. https://doi.org/10.1145/2959100.2959190

[8] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. 2010. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the fourth ACM conference on Recommender systems*. ACM, 39–46.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[10] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alan Aspuru-Guzik, and Ryan P Adams. 2015. Convolutional Networks on Graphs for Learning Molecular Fingerprints. In *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.). Curran Associates, Inc., 2224–2232. http://papers.nips.cc/paper/5954-convolutional-networks-on-graphs-for-learning-molecular-fingerprints.pdf

[11] Wikimedia Foundation. 2018. Wikimedia Downloads. https://dumps.wikimedia.org [Online; accessed 14. Oct. 2019].

[12] Wikimedia Foundation. 2019. Wikimedia Statistics - All wikis. https://stats.wikimedia.org/v2/#/all-projects [Online; accessed 13. Oct. 2019].

[13] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. *CoRR* abs/1607.00653 (2016). http://dblp.uni-trier.de/db/journals/corr/corr1607.html#GroverL16

[14] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*.

[15] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative filtering for implicit feedback datasets. In *In IEEE International Conference on Data Mining (ICDM 2008*. 263–272.

[16] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with GPUs. *arXiv preprint arXiv:1702.08734* (2017).

[17] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. http://arxiv.org/abs/1412.6980

[18] Thomas N. Kipf and Max Welling. 2016. Semi-Supervised Classification with Graph Convolutional Networks. *CoRR* abs/1609.02907 (2016).

[19] Quoc V. Le and Tomas Mikolov. 2014. Distributed Representations of Sentences and Documents. *CoRR* abs/1405.4053 (2014). arXiv:1405.4053 http://arxiv.org/abs/1405.4053

[20] Tie-Yan Liu. 2009. Learning to Rank for Information Retrieval. *Found. Trends Inf. Retr.* 3, 3 (March 2009), 225–331. https://doi.org/10.1561/1500000016

[21] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *CoRR* abs/1301.3781 (2013). arXiv:1301.3781 http://arxiv.org/abs/1301.3781

[22] Jonathan T Morgan, Siko Bouterse, Heather Walls, and Sarah Stierch. 2013. Tea and sympathy: crafting positive new user experiences on wikipedia. In *Proceedings of the 2013 conference on Computer supported cooperative work*. ACM, 839–848.

[23] Denis Parra and Peter Brusilovsky. 2009. Collaborative filtering for social tagging systems: an experiment with CiteULike. In *Proceedings of the third ACM conference on Recommender systems*. ACM, 237–240.

[24] Tiago P. Peixoto. 2014. The graph-tool python library. *figshare* (2014). https://doi.org/10.6084/m9.figshare.1164194

[25] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[26] Tiziano Piccardi, Michele Catasta, Leila Zia, and Robert West. 2018. Structuring Wikipedia Articles with Section Recommendations. *arXiv preprint arXiv:1804.05995* (2018).

[27] Radim Řehůřek and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. ELRA, Valletta, Malta, 45–50. http://is.muni.cz/publication/884893/en.

[28] Stephen Robertson and Hugo Zaragoza. 2009. The Probabilistic Relevance Framework: BM25 and Beyond. *Found. Trends Inf. Retr.* 3, 4 (April 2009), 333–389. https://doi.org/10.1561/1500000019

[29] Gerard Salton, Anita Wong, and Chung-Shu Yang. 1975. A vector space model for automatic indexing. *Commun. ACM* 18, 11 (1975), 613–620.

[30] Andrew I Schein, Alexandrin Popescul, Lyle H Ungar, and David M Pennock. 2002. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 253–260.

[31] Daniel Valcarce, Alejandro Bellogín, Javier Parapar, and Pablo Castells. 2018. On the robustness and discriminative power of information retrieval metrics for top-N recommendation. In *Proceedings of the 12th ACM Conference on Recommender Systems*. ACM, 260–268.

[32] Ellery Wulczyn, Robert West, Leila Zia, and Jure Leskovec. 2016. Growing wikipedia across languages via recommendation. In *Proceedings of the 25th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 975–985.

[33] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le. 2019. XLNet: Generalized Autoregressive Pretraining for Language Understanding. *arXiv preprint arXiv:1906.08237* (2019).

[34] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. 2018. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In *KDD*.

[35] Dongyan Zhou, Songjie Niu, and Shimin Chen. 2018. Efficient Graph Computation for Node2Vec. *CoRR* abs/1805.00280 (2018). http://dblp.uni-trier.de/db/journals/corr/corr1805.html#abs-1805-00280