UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Vitali Kuzmin

Item2Vec-based Approach to a Recommender System

Bachelor's Thesis (9 ECTS)

Supervisor: Carlos Bentes, MSc
Supervisor: Mark Fišel, PhD

Tartu 2017

Item2Vec lähenemine soovitussüsteemi jaoks

Lühikokkuvõte:

Lõputöö eesmärk oli välja arendada sõnade vektoriesitusel põhinev mudel soovitussüsteemi jaoks. Mudelit rakendati Item2Vec meetodi alusel, kuna viimased uuringud on näidanud, et Item2Veci abil on võimalik saada konkurentsivõimelisi tulemusi. Töö koosnebki valitud lähenemise rakendamisest, selle tarkvara tehnoloogia paigaldamise kirjeldamisest Tarkvara Tehnoloogia Arenduskeskuse (STACC) tootele ning testandmete hindamise ülevaatest. Hindamise tulemused näitavad, et rakendatud mudel tõepoolest sobib soovitussüsteemide arendamiseks.

Võtmesõnad:

Soovitussüsteem, Item2Vec, kasutajate-poolne filtreerimine, sõnade vektoriesitused

CERCS: P176 Tehisintellekt

Item2Vec-based Approach to a Recommender System

Abstract:

The aim of this thesis was to develop a vector space representation model for a recommender system. The model implementation was based on the method called Item2Vec; this approach was chosen because recent studies have shown that it displays competitive results. The work consists of implementing the approach, evaluating it on test data and deploying it into production in the Software Technology and Applications Competence Center. Evaluation results show that the implemented model is indeed competitive and is suitable for building recommender systems.

Keywords:

Recommender system, Item2Vec, collaborative filtering, neural word embedding

CERCS: P176 Artificial intelligence

Acknowledgments

First, I want to say words of gratitude to the research group from STACC, Irene Teinemaa and Anna Leontjeva for helping me plunge into the world of machine learning. For Christian Safka and my supervisors Carlos Bentes and Mark Fišel, for giving timely assistance when it was needed and willingness to provide support at any time.

Contents

1 Introduction

Machine learning is a branch of the artificial intelligence field that is responsible for finding patterns with the ability to be learned [Bis06]. The core idea is that it is possible for a computer to learn without being explicitly programmed. This principle was introduced long ago, but the field has started to expand significantly in the last few years. Because machine learning requires a lot of data and computational resources, it was very slow in the past to run experiments.

Due to a significant increase in computational power and data availability, machine learning is a rapidly expanding field. In this thesis, the author focuses on a single application of machine learning, recommender systems, which tasks are the same. Using this general aim of machine learning, and considering recommender systems as a subsection, the system's tasks boil down to training a predictive model so that it could make forecasts, restore dependencies, make automatic decisions with new input, etc. A more human understandable explanation of what a predictive model is could be 'a model that learned from data to predict or suggest something'. In the case of recommender systems, it is generating personalized suggestions.

Today almost everyone uses the Internet, which makes it possible to buy goods from all over the world or read information about anything. However, there is often too much information. Recommender systems help people by filtering the content and picking out the most relevant pieces for everyone personally. The aim of this thesis is to develop and evaluate an Item2Vec model to be used in Software Technology and Applications Competence Center's (shortly STACC, that is "Created in collaboration with University of Tartu, Tallinn University of Technology and industry representatives, the technology development centre" [STA, UT]) recommender system, using open source tools, based on Word2Vec [MCCD13]. The Word2Vec tool "takes a text corpus as input and produces the word vectors as output". Word2Vec can learn which words are more likely to follow after others, or in what context they are commonly present. The words in the text corpus could be anything, for example in this thesis words are just movie indices (numbers) represented as strings, but it could even be pressed keys on the piano translated to 'words'. Word2Vec will learn the model using those 'words' and after a lot of examples the model will be able to play well, learning on big corpora which keys are pressed more likely after others [Yot].

As a result, the model based on Item2Vec that was implemented during the internship in STACC, is able to recommend items according to user preferences. Because of the fact that the model should work not only theoretically, but in practice as well, factors such as speed, ease of use, and ability to update the model with new items and events in real time were taken into account in the implemen-

tation in addition to general performance. Furthermore, the model is compared to STACC's cosine similarity model and other publicized implementations [Hug17].

The second chapter contains an overview of two common recommender system building approaches and the current market state. In the third section, several evaluation techniques are described along with an explanation of why they were chosen. Also included in the section is a theoretical comparison of this approach with another similar solution. The final part of the third chapter will recount which tools were used in development. The fourth chapter contains information about how the data was processed and the steps taken to train the model. Finally, the experimental results are displayed, in addition to a description of the integration process into an existing system. A conclusion is written to summarize the results and provide a brief overview of the completed work.

2 Background research

As said in the recommender systems course from University of Minnesota: "Recommender systems help people find things when the process of finding the information you need to make choices might be a little bit challenging, because there's too many choices, too many alternatives" [JAKb]. That aspect makes recommender systems in demand.

There are two basic types of recommender systems: content-based and collaborative filtering. Both are briefly overviewed in this chapter.

2.1 Content-based filtering (CBF)

The main idea of content-based filtering is to recommend items to customer, that are similar to previously liked items. An example of content-based filtering could be recommending movies. The recommendations could then use information such as actors, genres, tags given to the movie, or even media about the movie in websites, blogs, news, etc. The similarity will be calculated based on matching content. Even person recommendations could be done by checking mutual friends, or comparing interests. Using this information it is possible to predict the probability that you know each other or would be glad to know each other. There are some popular measures of distances between two items used in recommender systems: cosine similarity, Euclidean distance, Manhattan distance, and Jaccard similarity [Pol15].

All similarities are based on item profiles, and this is one of the cons of the content-based filtering approach. It requires rich item metadata so that they could be compared to other items and recommended properly. At the same time it leads to only similar content, because features of the items that the user saw previously will compose his/her profile (which features of the items person likes, which dislikes), and the system will search only for the items similar by some features (metadata). But some of CBF advantages make this system competitive and work well despite the disadvantages. Content-based filtering approach uses both implicit data (data that was collected from user without him/her explicitly providing it, for example clicks on links) and explicit data (that one user him/her self provided, like ratings). CBF is able to solve the cold start problem (when system has not much users/history and a lot of items are still not rated/viewed/purchased). Also, content-based filtering based systems are able to change recommendations in a short period of time if a user changes preferences, because with new content, the system will collect new user profile (metadata of items user likes) and based on this will receive new recommendations. As well CBF solves the problem of privacy, when a user doesn't want to share his/her profile, the system is still able to give recommendations based on implicit data (clicks). A final advantage of CBF is

its ability to provide explanations why an item was recommended. Some famous examples of systems that use content-based filtering for recommending items are Pandora radio and Internet Movie Database (IMDB) [LRU, IFO15, wik].

2.2 Collaborative filtering (CF)

Collaborative filtering also has two types. The first is item-based, which outputs similar items (similarity based on user ratings) to what the user liked as recommendations. Second type is user-based collaborative filtering, where recommendations are given based on items that similar users rated highly and the recommendee has not yet seen.

The main advantage of collaborative filtering is that it is domain free, which means that it could provide recommendations anywhere there is enough users and history collected about their behavior. In CF, similarity is commonly measured with cosine similarity, mean squared difference, or Pearson correlation coefficient (similarity based on user actions). One advantage of collaborative filtering is that it is possible to create a recommender system even with minimal domain knowledge, because all of the similarity calculations are based on user actions. Collaborative filtering works well in most cases (when there is enough users and their actions history) because if people make many similar choices, across a lot of data, it will classify the user to the right group of similar users. In most cases if previous choices were similar then future ones will be similar as well. One more opportunity of CF is that it does not require rich item descriptions or well structured user profile, because similarities are based on user actions, not on item features or user profile. Collaborative filtering faces the problem named *gray sheeps*, which are users whose opinions don't match any group of users. They rate items unpredictably, and the system can't find anything suitable to recommend to that sort of user. Other cons of collaborative filtering are the scalability problem, because calculations of nearest neighbor grow with both number of users and items . Some well-known systems that use collaborative filtering are YouTube and Reddit [Ama14, SK09, wik].

2.3 Similar solutions

At the time of writing this thesis, only one other implementation was found based on the Item2Vec article [BK16]. However in that implementation, another architecture was used- continuous bag-of-words (CBOW). A link to that repository can be found in [Junb]. The model described in this thesis uses a different architecture than Doosan Jung's solution. In this thesis, skip-gram with negative sampling (SGNS) is used, which should provide better results than the CBOW used by Doosan. According to Tomas Mikolov, the author of Word2Vec, "Skip-

gram: works well with small amount of the training data, represents well even rare words or phrases" [MSC+13]. The dataset of movies used in this thesis (see Section 4) may be classified as a dataset with "rare words or phrases", because there are many more films than user can rate and as a result we have a sparse matrix of user ratings for the items.

3 Word2Vec

Word2Vec is a tool that creates an Artificial Neural Network (ANN) able to learn on input corpora. This tool creates vectors from words, learns what words are frequently followed by others and calculates similarities between them. Similarities may be used for many different purposes, i.e. in Natural Language Processing (NLP) they may be used for improving machine translation. Word2Vec can use one of two techniques: continuous bag-of-words or skip-gram.

CBOW and skip-gram are neural architectures that describe how to learn the model. They have different working principles, which are seen in Figure 1. CBOW tries to predict one word from context and Skip-gram tries to predict context by word. Skip-gram can be trained with negative sampling, a mechanism that presents negative samples created in a random way (same as the training samples, but all are negative and not included in training) to the model [GL14].

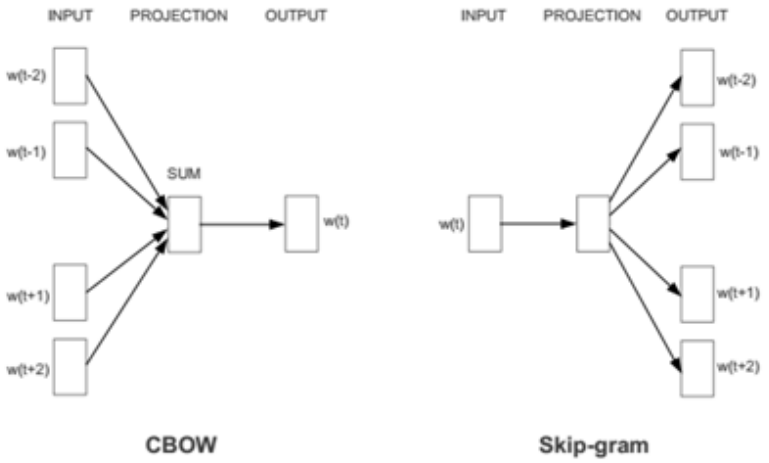

Figure 1: CBOW and Skip-gram architectures working principles [Cze]

The workflow of Word2Vec model looks like this:

1. Corpus reading, counting how often words appear in text,
2. Words sorted by word frequency in the corpus,
3. From vocabulary, create Huffman Binary Tree (more details here [MCCD13]),
4. Sentence by sentence corpus reading into the model and sub-sampling (process of eliminating most frequent words from analysis),

5. Algorithm goes through the sentence with specified window (model parameter), which is a number of words algorithm takes into account around the current word,

6. Using Feed-forward Neural Network (Artificial Neural Network (ANN) where connections do not form a cycle between the units) [Nik17].

3.1 Item2Vec

Item2Vec creates vectors of items and learns on input in the same way as Word2Vec. In this thesis the Item2Vec model produces recommendations by learning on history of what users liked, taking into account context (another user's rated movies). A visualization of how it works can be seen in Figure 2. In the experimental results chapter (see Section 5) there is some examples shown of how the model works in practice.

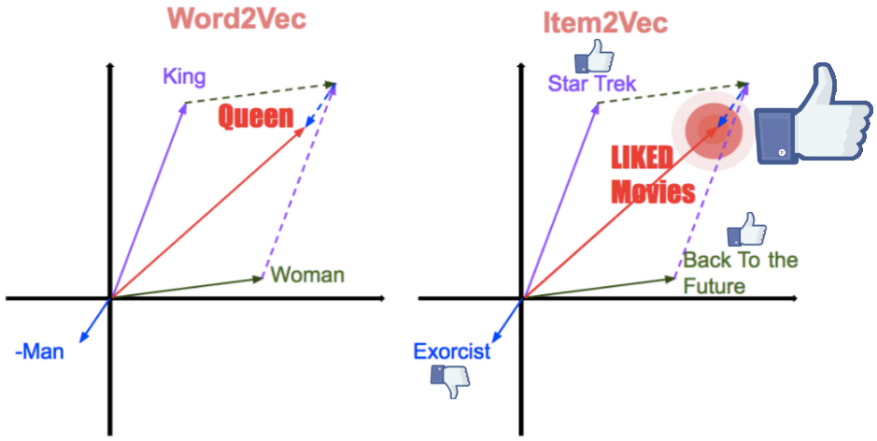

Figure 2: Item2Vec works on the same principle as Word2Vec. Modified original from [Juna]

3.2 Score prediction

For scores predictions was used Equation 1 from the recommender systems book [Agg16]. The idea of predicting user rating using items neighborhood, as it is written in above mentioned book, is to identify similar items to the target item in the list of user rated items, multiply their similarity by the rating of the user's rated item centered to a mean of zero. This divided by the absolute value of similarity between the items gives the mean-centered rating of item for the user. To make the prediction more accurate, this procedure calculates rating using all items in the neighborhood. Then, adding the mean rating of the film to this result returns the real rating.

$$\hat{r}_{uj} = \eta_j + \frac{\sum_{i \in I_u} \text{Sim}(i,j) \times s_{ui}}{\sum_{i \in I_u} |\text{Sim}(i,j)|} \tag{1}$$

Here:

$\hat{r}_{uj}$ is predicted user's u rating to an item j

η_j is a mean rating to an item j

$\text{Sim}(i,j)$ is similarities between items i and j, where i is in the set of items rated by user u.

s_{ui} is the mean-centered rating of user u to the item i.

3.3 Recommender systems evaluation

Principle idea of evaluating a recommender system is to use existing data collected from users (ratings of items, logs of clicks, etc.) to simulate user behavior and see whether the recommender system can predict correct behavior of the user or not. The more user actions predicted correctly on test (hidden) data, the more likely the recommender system will work fine in live mode. In practice, existing data is split to train and test sets, and usually the train set is noticeably bigger than test set. When the recommender system is trained, it predicts user behavior from test data and then measures difference between the predicted and actual values to see how well the model performed [JAKa].

There are several different metrics for measuring model performance:

- Prediction accuracy is when the system tries to predict rating, and compares it to the ground truth rating.
- Predict a list of items a user may like and see how many of them user actually bought in the test data.
- Measure how close items that the user actually purchased were to the top of the recommended list.

For this thesis, the prediction accuracy metric was chosen. Therefore, the models are compared by their ability to predict the score a user would give to an item. There are two common metrics of prediction accuracy: Mean Absolute Error (MAE) illustrated in Equation 2 and Root Mean Squared Error (RMSE) presented in Equation 3 [JAKc].

$$MAE = \frac{\sum_{ratings} |P - R|}{|ratings|} \tag{2}$$

$$RMSE = \sqrt{\frac{\sum_{ratings} (P - R)^2}{|ratings|}} \tag{3}$$

Here:

P is predicted rating

R is actual rating

$|ratings|$ is total number of terms in numerator

MSE is not observed because it is mostly used only to explain RMSE, which shows almost the same thing, but is a more human readable and intuitive metric. Both of the other metrics MAE and RMSE were calculated for the developed model. It is good practice in systems evaluation process to use cross-validation, which provides more accurate results. With cross-validation, there is 100% coverage of data during the evaluation process, while without it might use some unrepresentative data (i.e. a lot of information missed) for training the model and make it very hard to test properly, leaving a result that would not show how the system really performs.

3.4 Development tools

For model development, Linux Mint was used because it is easier to install and hold the environment on the Unix-based operating system. The language chosen was Python 3.5 [PG07], because it was used to create the system and has a lot of efficient packages for machine learning tasks. Scikit-learn [PVG+11], which is built for machine learning tasks, NumPy [WCV11]- very good library for scientific computations, pandas is developed for data analysis, SciPy [JOP+]- scientific computing tool for Python, gensim [ŘS10] is an easy to use library for generating similarities. Something very useful is that all these libraries communicate perfectly with each other. As a main development environment, Pycharm CE 2017.1 was used because it highlights errors, suggests better code styling, provides autocomplete and includes other comfortable features. Anaconda [Dis16] is a virtual

environment system for python to handle package installation. It was used to install all necessary libraries, as well as the Jupyter Notebook. Jupyter Notebook helps to save intermediate results and allows to use them in other *code cells* inside one notebook, what is very convenient and important for data analysts, because it saves a lot of time that otherwise would be used saving and loading files for every intermediate result.

4 Data

This section describes what data was used for model performance testing and why, as well as how the data was processed.

4.1 Choice of a dataset

MovieLens dataset was chosen for model development and evaluation [HK15]. It is a dataset collected by GroupLens research group in the Department of Computer Science and Engineering at the University of Minnesota. This dataset was chosen because it is popular for testing recommender system performance and has a lot of published results for different algorithms and estimation metrics. The choice fell in favor of 1M (contains 1,000,000 ratings) dataset instead of the full one that contains 24,000,000 ratings. At one point this decision was made because of the original repository [Hug17] that contains different algorithms tested on MovieLens 1M dataset with calculated Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) for each algorithm. Moreover, full dataset processing takes a lot of time even on a powerful server. Data is stored in different files with *.dat* extension and full size is about 24 MB which is much less comparing to the full dataset at 0.9 GB.

4.2 Dataset description

MovieLens dataset provided by GroupLens Research group [HK15] from University of Minnesota. It is data about how users used online movie recommender web service, rated and tagged movies there. Also provided in the dataset is anonymous data of users like gender, age group, and others that might be relevant.

As it's written in the MovieLens dataset readme file, movieId is a unique number from 1 to total number of movies (some numbers are skipped), and is used as an identifier for movies. All ratings are contained in the file "ratings.dat", which has 4 fields. Namely,

- userId: also a unique number, from 1 to number of users (without any skips),
- movieId: the movie the user rated,
- rating: user opinion about the movie in a 5-star scale,
- timestamp: (UNIX time) when user rated the movie.

All information about the movies is stored in the file "movie.dat", which includes

- movieId: used to identify the same movie across whole dataset,
- title: movie title concatenated with the year the movie was released,
- genres: pipe-separated list of genres (18 different genres starting from Action and Adventure and ending with War and Western), if movie has more than one.

These two files were used for extracting all information required to train the model.

4.3 Data processing

For the model to learn from the data in the *.dat* files, they were read into python arrays and all item indices needed to be transformed into strings, because Word2Vec only operates with string data. Because of the fact that Word2Vec operates with unlabeled data, ratings could not be considered during model training, and that definitely affects performance of the model in a negative way.

The solution for this unlabeled data problem was found in the recommender systems book [Agg16]. Users may have different scales of ratings and according to the author words, one user might be biased toward liking most items, whereas another user might be biased oppositely. It is possible to calculate a mean-centered matrix, which is useful in solving this problem. Mean-centered matrix means that the average rating of each item in the ratings matrix is subtracted from each rating.

Liked and disliked lists of products for every user compose a training corpus (usually for Word2Vec this corpus is called 'sentences', i.e. in documentation and other sources) where for each user, two arrays of movie indices (sentences) are created which are positively rated items in the first array and negatively rated items in the second. These sentences suit for model vocabulary building as well, which is one of the most important parts of the model. According to documentation, *build_vocab* method calls *finalize_vocab* method, which builds tables and model weights based on final vocabulary settings, meaning that similarities will be calculated for all items [ŘS10]. For positive sentences, only four and higher star user ratings and for negative sentences, three and lower star ratings.

5 Experimental results

First, there is an example shown of what the model will recommend based on a user that liked "Star Trek" and "Back to the Future" and disliked "Exorcist".

```
liked_movies = ['Star Trek: Generations (1994)', 'Back to the Future
    (1985)']
disliked_movies = ['Exorcist, The (1973)']
recommendations = give_recommendations(liked = liked_movies, disliked
    = disliked_movies, number_of_recommendations = 3)
```

This code asks the model for recommendations based on the given input, and it returned the following movies as recommendations:

1. Stargate (1994) (Action|Adventure|Sci-Fi)
2. Star Trek: First Contact (1996) (Action|Adventure|Sci-Fi)
3. Star Trek VI: The Undiscovered Country (1991) (Action|Adventure|Sci-Fi)

The recommender work could be visually seen from the Figure 3 in a simple form.

For implemented model evaluation, metrics Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) were selected. The model was evaluated using 5-fold cross-validation procedure. Every user history was divided into 5 equally long arrays of items where 20% was used for testing and 80% for model training. Each fold, the model was retrained with new input and tested on validation data hidden from the train set. Implemented functions for model evaluation are described in recommender system evaluation course from the University of Minnesota [HKTR04].

The developed model shows better results for RMSE compared to STACC's cosine similarity model Figure 4, as well as 8 out of 9 other algorithm implementations found in [Hug17]. If compared with MAE, then the results are better than 6 out of 9 of the same algorithms. This can be seen in the supplemented Table 1 from the above mentioned repository.

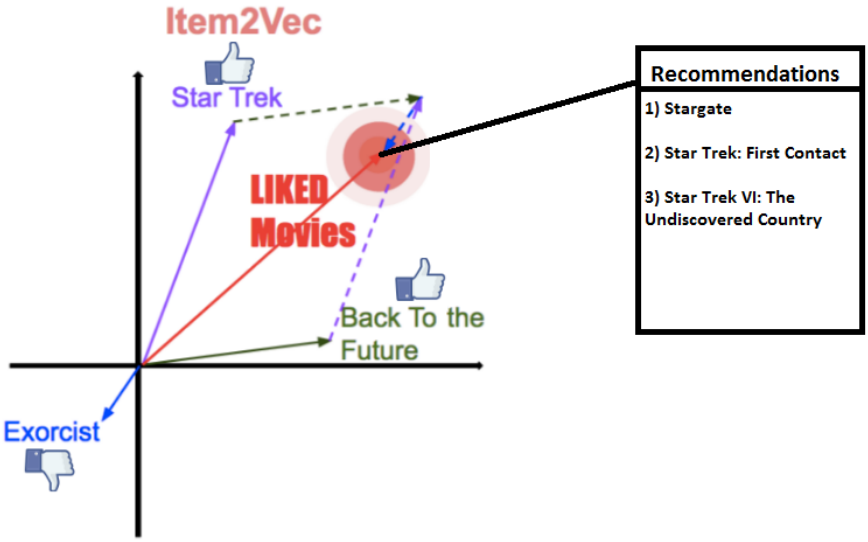

Figure 3: Item2Vec recommender system in practice. The model combines the user liked movies (Star Trek and Back to the Future) with the user disliked moved (Exorcist) to find the space of recommendations (red vector). Modified from original, found on [Juna]

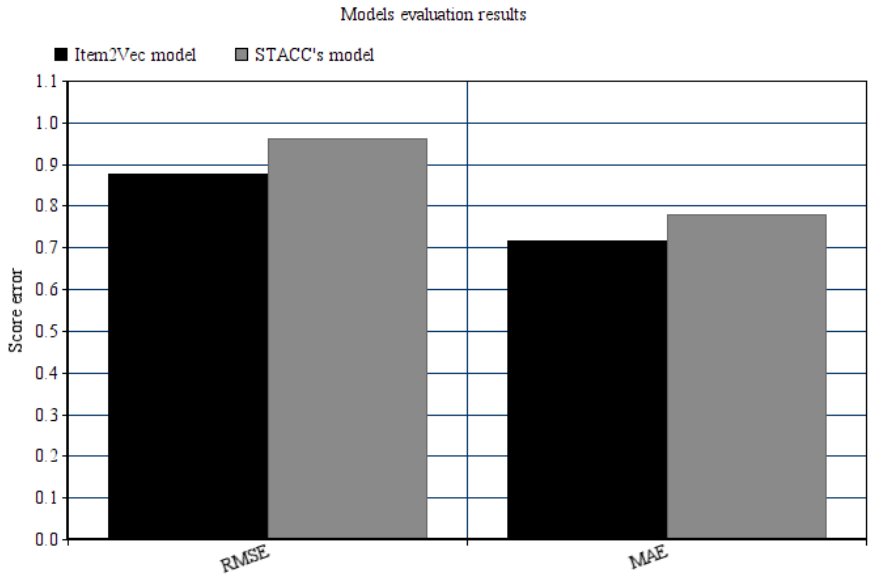

Figure 4: Performance comparison of the developed model (black columns) and STACC's cosine similarity model (gray columns)

In the Item2Vec article, it was mentioned that an Item2Vec based model should be competitive with SVD (Singular Value Decomposition) based model, and these words were confirmed as seen from the Table 1. SVD has RMSE of 0.8738 while Item2Vec shows 0.8741, and MAE scores are 0.6858 and 0.7147 respectively.

Table 1: Comparison between different algorithms performance tested on Movielens 1M dataset (all numbers, but last taken from [Hug17])

Movielens 1M	RMSE	MAE
NormalPredictor	1.5037	1.2051
BaselineOnly	.9086	.7194
KNNBasic	.9207	.7250
KNNWithMeans	.9292	.7386
KNNBaseline	.8949	.7063
SVD	.8738	.6858
NMF	.9155	.7232
Slope One	.9065	.7144
Co clustering	.9155	.7174
Item2Vec	**.8741**	**.7147**

Item2Vec model evaluation time was slower, comparing to the cosine similarity. Item2Vec spent one hour for 1,000,000 scores prediction ($5 \times 20\% = 100\%$, and there is 1,000,000 ratings in initial dataset), while STACC's model did it in 6 minutes. Visually it is seen from the Figure 5.

Adding new items to the system was a bit challenging, for both Item2Vec and cosine similarity model. Figure 6 shows how much time was spent on adding to both algorithms n new items. However Item2Vec model showed better results here.

In addition to time saving, Item2Vec model requires just a few lines of code to be retrained and updated with new items:

```
from gensim.models import Word2Vec
item2vec_model = Word2Vec.load('name_of_the_model')
item2vec_model.build_vocab(new_items, update=True)
item2vec_model.train(new_items)
item2vec_model.save('updated_name_of_the_model')
```

This simple example with cosine similarity however takes around 40-50 lines of code and much more time when including the necessity to read all the data once again, create new user-item matrices, and re-calculate similarities between every item.

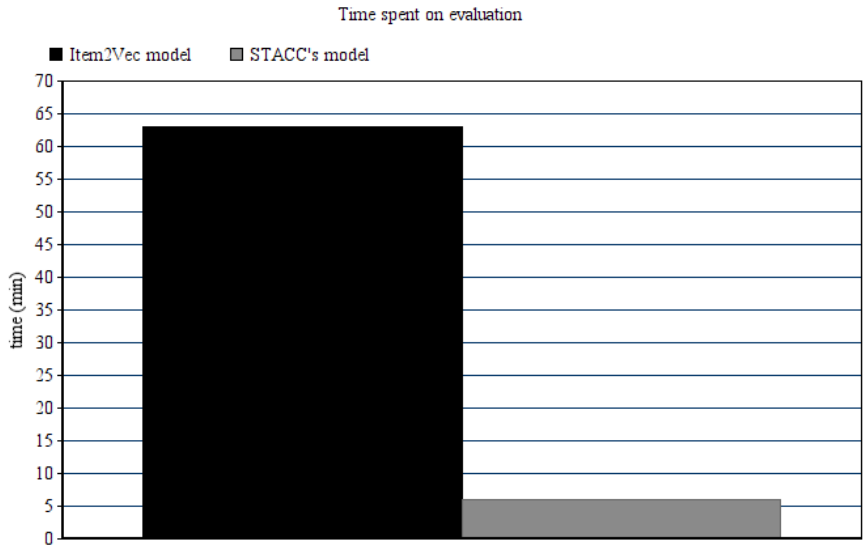

Figure 5: Comparison of the developed model (black columns) and STACC's cosine similarity model (gray columns)

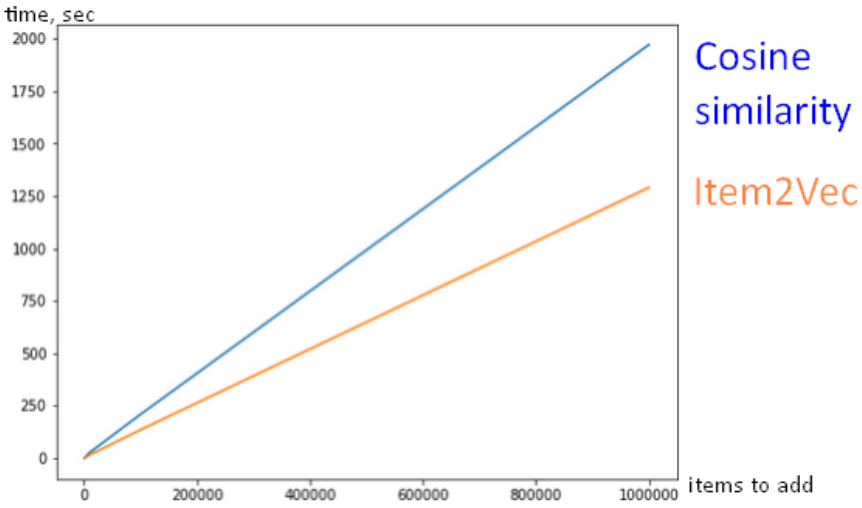

Figure 6: Models performance test by adding new items to the system

6 Integration

Elasticsearch [Ela] is an open source, distributed, scalable, document-oriented engine. By distributed and scalable, it is meant that it is possible to grow the system without requiring newer and newer hardware. This only makes sense up to a certain point, because after that it's not suitable to continue without upgrading. Elasticsearch is able to not only scale vertically, but horizontally as well, which means that when buying additional hardware, it is possible to use it alongside your current hardware, instead of replacing it. Document-oriented means that everything you do with Elasticsearch in terms of communication is JSON, which most developers are very familiar with, and both back-end and front-end developers could use.

In the recommender system there are two parts, online and offline:

Online part is responsible for giving recommendations and taking into account recent user actions when there is no time to add them to the model and recalculate similarities between items.

Offline part is responsible for updating and evaluating models, which can't be done on the fly (in milliseconds) as everything in online part, but takes some time for calculations.

Elasticsearch fits the need to get online recommendations in milliseconds, because it has extremely fast searching even when there is a lot of information stored. Model storing used the Elasticsearch engine as well, which in addition to all the listed advantages may be used as NoSQL database [Bra]. NoSQL database is defined as "Not only SQL"(SQL- Structured Query Language). That are points explaining why Elasticsearch was chosen by STACC.

Model implemented during this thesis should have been able to be stored inside Elasticsearch as well. The first k top similar items from the model are stored in Elasticsearch. This will not affect recommendations, because they are usually top n similar items, and $n < k$. This is shown in the GitHub repository of the model's code (See Appendix B.)

7 Conclusion

In this thesis, an Item2Vec based model was implemented and compared to STACC's current and other models. In the section about data processing there is an overview of the model development process, along with the evaluation methods in the subsection. The results section shows that the developed model is better than previous STACC ones as well as many others. However it has it's own cons, like long evaluation time. In the future, model performance might be improved by transforming it to a hybrid system (system that uses both collaborative filtering and content-based filtering).

References

[Agg16] Charu C Aggarwal. *Recommender Systems*. Springer, 2016.

[Ama14] Xavier Amatriain. Recsys 2014 tutorial - the recommender problem revisited, 2014. [Online; accessed 2017.04.20].

[Bis06] Christopher M Bishop. Pattern recognition. *Machine Learning*, 128:1–58, 2006.

[BK16] Oren Barkan and Noam Koenigstein. Item2vec: Neural item embedding for collaborative filtering. 2016.

[Bra] Alex Brasetvik. Elasticsearch as a nosql database.

[Cze] Michael Czerny. Modern approaches to sentiment analysis. [Online; accessed 2017.05.10].

[Dis16] Anaconda Software Distribution. Computer software. Vers. 2-2.4, November 2016.

[Ela] Elasticsearch homepage. `https://www.elastic.co/`. [Online; accessed 2017.05.08].

[GL14] Yoav Goldberg and Omer Levy. word2vec explained: Deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*, 2014.

[HK15] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4):19:1–19:19, December 2015.

[HKTR04] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, January 2004.

[Hug17] Nicolas Hug. Surprise, a Python library for recommender systems. `http://surpriselib.com`, 2017.

[IFO15] F.O. Isinkaye, Y.O. Folajimi, and B.A. Ojokoh. Recommendation systems: Principles, methods and evaluation. *Egyptian Informatics Journal*, 16(3):261 – 273, 2015.

[JAKa] Michael D. Ekstrand Joseph A Konstan. Hidden data evaluation. [Online; accessed 2017.05.04].

[JAKb] Michael D. Ekstrand Joseph A Konstan. Introduction to recommender systems: Non-personalized and content-based. `https://www.coursera.org/learn/recommender-systems-introduction`. [Online; accessed 2017.04.18].

[JAKc] Michael D. Ekstrand Joseph A Konstan. Prediction accuracy metrics. [Online; accessed 2017.05.04].

[JOP+] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed 2017.05.08].

[Juna] Doosan Jung. Item2vec implementation. Presentation. [Online; accessed 2017.05.10].

[Junb] Doosan Jung. Item2vec project. `https://github.com/DoosanJung/I2V_project`. [Online; accessed 2017.05.08].

[LRU] Leskovec, Rajaraman, and Ullman. Content based recommendations.

[M+10] Wes McKinney et al. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. van der Voort S, Millman J, 2010.

[MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[MSC+13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[Nik17] Nikita Nikitinsky. http://nlpx.net/archives/179, 2017. [Online; accessed 2017.05.09].

[PG07] Fernando Perez and Brian E Granger. Ipython: a system for interactive scientific computing. *Computing in Science & Engineering*, 9(3), 2007.

[Pol15] Saimadhu Polamuri. Five most popular similarity measures implementation in python, 2015.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[ŘS10] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.

[SK09] Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. 2009. [Online; accessed 2017.04.20].

[STA] Stacc webpage. `https://www.stacc.ee`. [Online; accessed 2017.05.08].

[UT] University of tartu. `https://www.cs.ut.ee/en/research-and-collaboration/research-projects/stacc`. [Online; accessed 2017.05.08].

[WCV11] Stefan van der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.

[wik] Recommender System. Wikipedia. `https://en.wikipedia.org/wiki/Feedforward_neural_network`. [Online; accessed 2017.05.09].

[Yot] Yotam Mann with friends. Github page. a.i. duet. `https://github.com/googlecreativelab/aiexperiments-ai-duet`. [Online; accessed 2017.05.08].

Appendix A Glossary

STACC - Software Technology and Applications Competence Center
NLP - Natural Language Processing
CF - Collaborative Filtering
CBF - Content-based Filtering
ANN - Artificial Neural Network
NoSQL - Not only SQL
IMDB - Internet Movie Database
SQL - Structured Query Language
CBOW - Continuous Bag-of-Words
SGNS - Skip-gram with Negative Sampling
MAE - Mean Absolute Error
RMSE - Root Mean Squired Error
SVD - Singular Value Decomposition

Appendix B Github repository

Source code of this thesis is placed in GitHub: https://github.com/vtlkzmn/Thesis

Non-exclusive licence to reproduce thesis and make thesis public

I, Vitali Kuzmin (date of birth: 6th of August 1995),

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

1.1 reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

1.2 make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

Model for STACC's Recommender system

supervised by Carlos Bentes and Mark Fišel

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, 10.05.2017