Web Intelligence
TDT4215

REPORT – NEWS RECOMMENDATION GROUP 10

April 7, 2017

Lemei Zhang
Eivind Tveita Havikbotn
Fredrik Gram Larsen

NTNU – Trondheim
Norwegian University of
Science and Technology

Contents

List of Figures

1 Introduction

The following report is a part of a group project delivery in the course TDT4215 Web Intelligence held at Norwegian University of Science and Technology (NTNU) in spring 2017. The project involves the students signing up for the CLEF NewsREEL[1] competition and implementing a news recommendation system in an accompanied framework.

1.1 CLEF NewsREEL

Conference and Labs of the Evaluation Forums[2] (CLEF) maintains frameworks for information access systems, and promotes evaluation and development by organising yearly contests in which the participants receives tasks relevant for the retrieval systems. The contest held in 2017 is called CLEF NewsREEL – or CLEF News Recommendation Evaluation Lab – and the overarching goal is to implement a *news recommendation system*. This suggests that whenever a specific user reads a news article, other articles that may be of interest to that user are to be retrieved by the system.

1.2 The Project Task

CLEF provides two tasks for the challenge; the first involving a real-time setting, the second allowing for replay of a fixed data set. The students are required to solve the second task: ***Making a system that recommends news articles in a simulated live setting, where a sampled data stream are replayed by a reference framework***. The reference framework is called Idomaar[3] and will be further discussed in section section 4.1.

[1] http://www.clef-newsreel.org/
[2] http://clef2017.clef-initiative.eu/
[3] http://rf.crowdrec.eu/

2 Preliminary Studies

This section provides a look into the news domain and the challenges it presents for recommendation systems, followed by an analysis of the domain specific data provided for the task.

2.1 The News Domain

The news domain involves a huge portion of the ever-expanding bulk of information available on the Web and, accordingly, requires robust recommendation systems in order to navigate users to content that may interest them. Although the news domain bear many similarities to other research areas, i.e. movie databases or music streaming services, it differs in some respects:

- Users are often anonymous, as they are not necessarily prompted to log into a news portal.
- The news domain is very dynamic, which means that articles are generated very frequently and are constantly subjected to revision.
- News articles expires after time, and old articles are highly likely to not be relevant for a user, even if they erroneously may be deemed so by traditional recommendation systems. This also works the other way around; a very fresh article that does not fit a user profile might still be of interest for the user.
- News articles are largely unstructured; they consists almost entirely of text, and valuable information is often found in the text.

2.1.1 News Recommendation Challenges

In accordance to the unusual nature of its domain, news recommendation systems brings a whole new set of challenges to the table, most of which are due to the the *dynamic* and *unpredictable* qualities of the news domain. For instance, in the duration of a short time period, numerous news articles may have been created or updated, which enforces ***strict requirements to efficiency***. Computational time is therefore important to take into consideration when designing the recommender.

Another challenge often encountered in the news domain is the ***rapid change in popularity and relevancy*** of news articles. Recommender systems applied in, for instance movie databases or music streaming services, operates on items whose demands decrease or increase steadily over time. News recommendation, however, needs to take into account the sudden changes that may happen in few days, or even hours, due to the short relevancy of news.

One of the most prominent issues when working with news articles, however, relates to the ***cold-start problem***, which embodies the challenges in making recommendations when insufficient information is available. This is a common problem for most systems that are trying

to recommend an item lacking in ratings. In the news domain, the issue is exacerbated to an even higher degree due to the frequent and numerous articles generated constantly – articles not yet rated are quite commonplace in news portals. The recommendations also have to deal with cold-start users, since many news sites don't require the reader to log in in order to view an article. These users are regarded as anonymous and are therefore lacking in information for which to use in recommendation.

2.2 Data Analysis

The data set provided by CLEF for the NewsREEL Replay task includes event logs from several German news sites, gathered over 28 days in February 2016. Additionally, the 100 000 first events from the first day are included as a separate file for testing. The data stream consist of *three* types of events:

1. ***Item update***: Notifies the system that the publisher on a specific news site has published an article. The most useful fields are *domainID*, *itemID*, *created-date* and *flag* (stating whether the item can be recommended or not).

2. ***Recommendation request***: Notifies the system that a certain user is browsing a certain article, and requests a recommendation for the next article to read. This event contains a lot of contextual information about the user, such as browser-type, age-probability etc, but since it is desirable to focus mostly on the content, these fields are excluded. The most important fields are *userID*, *itemID* (the article currently read by the user), *domianID*, *timestamp*, and *recommendation-limit* (how many articles the user wants recommended). Additionally, each recommendation request comes with a set of keywords and their counts. These keywords are – for the most part – static to the article the current user is reading, and accordingly these are assigned to the items as content.

3. ***Event notification***: In theory, several types of events, but in the data set only click events are actually present. A click events tells us that a user has successfully clicked on a recommendation, and it contains the fields *domainID*, *itemID* (for the source article), *userID*, *timestamp* and a list of the *recommendations* (articles) the user has clicked on.

After investigating the data and running the data stream on the evaluation framework, it was discovered that roughly half of all recommendation requests have a source article without preceding item update. The data is summarised and compared across the different days in the appendix.

As seen in the statistics, the amount of users starts off as 550 000 observed the first day, 800k users the second day, and thereafter growing by 100-150 thousand users per day. Presume that the activity is equally distributed among new and recurring users, then the activity for unknown user for the last 14 days in the data set would account for no more than 4 percent. Regarding the

articles, there is an average growth of 280 items per day, which is a bit higher the first days than in the last half, most likely due to articles being updated and kept in the system for reuse. There is roughly 6 million recommendation request per day, which certainly will pose a challenge in terms of computational complexity. Lastly, as for the keywords, 67 percent are being presented the first 14 days, and since each article contains a number of keywords, only training a model on 67 percent should be enough to generalise over new articles.

3 Design and Ideas

In this section we will go through our three proposed solutions to the news recommendation challenge. Our goal was to produce a set of recommenders, used to select recent articles both popular among similar users (Collaborative filtering) and select articles that are related to the long term topics each user has shown interest in (Content-based SVM model) – either as independent models and hybrid systems.

3.1 Approach 1: Collaborative filtering

In the first approach we had planed to build a neighbourhood-based user-user collaborative filtering system. As described in the textbook, collaborative filtering is done by first performing a user-user similarity measure between all users (or between a pre-computed group), either by using Pearson or Cosine similarity. In order to avoid user bias when dealing with continuous-, interval- or ordinal ratings, the ratings are often scaled by reducing the average (mean centring). Thereafter, the rating for a particular item can be calculated by multiplying the similarity of the neighbouring user with their rating value for the specific item, averaged over all neighbouring users.

Potentially, this kind of method can perform well when the data set is more or less static, and, preferably, when there are numerical rating values). For large scale collaborative filtering systems there will usually be an offline phase and an online phase, where the offline phase computes the similarity between all users (with a complexity of O(nm^2), and the online phase computes the similarity between the offline selected k-most similar users (with complexity O(nk^2) where k << m). The problem when applying this to the news domain is that the content is rapidly shifting and that the recency of articles is an important factor. Additionally, many neighbouring users may not be active during the session when computing the recommended articles in the online phase.

For our system we planned to use the first half of the dataset for building the initial neighbourhood profiles for each user. Then in the online phase, we will find the n-most similar active users from the k-sized neighourhood profile, where n < k. Thereafter we planned to compute the rating for each potential article, by the weighted similarity score of the neighbouring user and a time decay factor of $(1 - t * w)^2$ where t is number of minutes and w is an empirical set weight according results during testing (start value for w = 0.01).

3.2 Approach 2: Keyword based Support Vector Machine

Support Vector Machine (SVM) is a classic machine learning method in dealing with classification problems. Traditionally, many classification problems try to solve the two or multi-class situation. As in our case, clef data set provides us with the items the user is reading or clicking, which could be regarded as user's interesting items. However, other items that this user hasn't looked at, are not representations of what the user doesn't like, which means we only have data of one

class and the goal is to find out new items whether it is alike or not like the clicking ones. Under this condition, one class SVM [4] is a better way to distinguish interesting items for a user from the others.

The basic idea for one class SVM is that it will detect the soft boundary of the training set so as to classify new points as belonging to that set or not. Considering a data set $\omega = (x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$; $x_i \in \mathbb{R}^k$ for $y_1 = y_2 = ... = y_n$, where x_i is the i-th input data point and is a k-dimensional vector, $x_i = (d_1, d_2, ..., d_k)$, where $d_i \in \mathbb{F}$ and $\mathbb{F}$ represents the feature space. To simplify the problem, we could represent input data set as $X \in \mathbb{R}^{n \times k}$ for the coordinates of n points in k dimensions. The problem can be explained as to separate all the data points in feature space $\mathbb{F}$ and and create a hyperplane that maximizes the distance between the outlier points the ordinary ones.The original data can be projected into a lower space through a function Φ from the original space with higher dimensions. This algorithm is based on the assumption that different classes of data are separated according to their density.

Different from the multi-classification SVM, the objection function of one class problem is the following minimization formulation:

$$min_{w,\epsilon_i,\rho} \frac{\|w\|^2}{2} + \frac{1}{vn} \sum_{i=1}^{n} \epsilon_i - \rho$$

subject to

$$(w^T \Phi(x_i)) \geq \rho - \epsilon_i, \; for \, i = 1, 2, ..., n$$

$$\epsilon_i \geq 0, \; for \, i = 1, 2, ..., n$$

where $\frac{1}{vn} > 0$ is the constant determine the trade-off between maximazing the margin and the number of training data points within that margin. n is the total number of points. The $w^T \Phi(x_i)$ is the hyperplane which could separate different points that belong to various classes. It should be noticed that it is the parameter v that characterized the solution for the first, it sets an upper bound on the fraction of the outliers, and the second, it is a lower bound on the number of training examples used as Support Vector.

Thus, the decision function becomes:

$$f(x) = sgn((w^T \dot{\Phi}(x_i)) - \rho)$$

where $w^T \Phi(x_i)$ could also reffer to $\sum_{(}^{n} i = 1) \alpha_i K(x, x_i)$, and $K(\dot{})$ is the kennel function and α_i is the weight. And according to Chapelle[5], the time complexity is $O(max(n, d) min(n, d)^2)$.

[4] Schölkopf, Bernhard, et al. "Support vector method for novelty detection." NIPS. Vol. 12. 1999.
[5] Chapelle, Olivier. "Training a support vector machine in the primal." Neural computation 19.5 (2007): 1155-1178.

3.3 Approach 3: Deep Learning in Collaborative similarity space

A third approach we decided to explore was to utilise Deep Learning to map the content of the articles into a latent feature space constrained by the similarities found collaborative filtering. As described in chapter 2, one of the key problems with news recommendation systems are that new content is arriving continuously thereby limiting the system by the fact that enough users would have had to read the new article before user-user recommendations can be produced. Our proposed solution is to overcome this problem by learning a mapping between the content of each article and the collaborative filtering space.

First we want to select a substantial part of the data to use for training, for example the first 14 of the total 28 days for the CLEF dataset. By this data partition we can construct the unary rating matrix m x n, where m is the number of users and n is the number of items. By using a dimensionality reduction technique such as SVD or PCA, the matrix can be reduced to d x n where d $<<$ m, typically 100-300 dimensions. Then we can build a neural network model, with the content as input, connected to several layers of hidden nodes, and an output layer with size d. By training this model on each article, the model should be able to generalise over new unseen articles with different/mixed content and correctly map the newly published article into the collaborative space.

As for the content, and the topology or the network, we would first like to adapt this methodology on the keywords associated with each article. For this, a simple 2-3 hidden layer network should suffice, and the input dimensions should be the size of the number of keywords observed so far, 5000. Since the keywords occur with multiple frequencies, the quantity of each keyword should be squashed between 0.5 and 1,(for n occurrences > 0). However, given enough data, one should also be able to adapt this method to do content extraction from the title and text. A standard approach for doing this is to use Word2Vec word-embeddings, and a LSTM over the textual input.

Once the model has been trained, we should then be able to model the users path through the collaborative space by extracting the neural net representation from each article the user visits. In order to perform recommendations, a simple approach would be to perform a recency-oriented weighted average over all the visited articles. This can then be used to compare against the embedding representation of potential recommendable articles.

If one would like to extend the model even further, instead of doing weighted average for the recommended collaborative representation, one could use the article representations as input to recurrent neural network, and then optimize the network to learn the sequence of user visits. Due to the representation space now has embedded much of the user-user similarities, the model should converge faster. In addition data for the time of visit and duration could also be included in the input vector to further boost the recommendation. Hopefully the RNN model should then be able predict the continuous path for the user, which would be used for retrieving the most

appropriate article to recommend.

3.4 Approach 4: Graph-based Collaborative Filtering

User-item relationships can be represented as heterogeneous information network. In such network, the edge between nodes represents some form of relationship. In our graph, user-item relationship are adopted during the recommendation process in order to find the preference propagation over the network.

Before introducing graph-based recommendation system, one substantial concept need to introduce, which is data embedding. It is used in machine learning applications to create low-dimensional feature representation. Meanwhile, data embedding approach keeps the structure of data point in their original space. The main goal of data embedding task is to learn the mapping functions to project data from different modalities to a common space so that similarities between objects can be directly measured.

DeepWalk is a popular way to handle data embedding problems for heterogeneous network. After network construction phase, random walk are used to generate nodes clusters which is similar among each other or have close relationships among each clusters. Then, DeepWalk feeds it to word2vec to generate embedding. word2vec, a shallow neural network-based set of algorithms has been applied outside of the natural language processing (NLP) domain to, effectively map latent features hidden in the paths over the information network into a vector space.

4 Development Phase and Experimentation

During the implementation process, we adopted python and java two different programming languages into the same framework, and meanwhile two kind of tools are mainly used throughout the recommendation process:

- Spark[6]. An open source framework for analytics of data sets of considerable size.
- Scikit-learn[7]. It is a free software machine learning library for Python. It features various classification, regression and clustering algorithms.

4.1 The CLEF Framework - Idomaar

We started our work by looking into the Idomaar framework provided by the CLEF organisation, and we spent much of our initial time working and trying to accommodate for the different faults for this software.

The framework consist off a data-streaming module that reads the data files, and a computation environment where we can place our own recommendation module. The data-streaming module transmits the event lines in the .log files provided by CLEF over a http connection and waits to receive recommendations as a response. It then calculates whether the correct recommendation has been returned and calculates the Click-through-rate for the session for each domain(news outlet).

In addition the CLEF framework has a codebase for the recommendation server, which includes a table structure for the incoming recommendation requests and a simple recommendation algorithm called DirtyRingBuffer that returns the most recent items as a baseline. In essence we were given the impression that simply learning how the framework functioned, we could easily implement our own algorithms in the excising codebase, and then run the entire dataset by using the out of the box components. Unfortunately this was not the case.

During our deployment process with CLEF framework, a lot of problems were occurred as organized as follows:

1. ***Virtual Machine***: The CLEF framework should be built and run on Virtual Box. However, two framework cannot run on the same virtual machine, even changing the connection port. Besides, we also find one virtual machine cannot build upon another.
2. ***Data Format***: When we deployed the evaluator on the server according to the tutorial, and test the evaluator with two different data format in 2014-07-01.data.idomaar_1k.txt and nr100000lines.log files, the recommender threw the *ArrayIndexOutOfBoundsException* exceptions.

[6]http://http://spark.apache.org/
[7]http://scikit-learn.org/stable/index.html

3. ***Dirty Ring Buffer***: When we run the recommender algorithms, most item lists in DirtyRingBuffer according to different publishers are empty, and thus recommender system could not return any recommendation results. But this should not be correct, because there are 2086 item_update types of data in nr2016-02-01.log file in total and among them, 1203 are unique items, whereas the DirtyRingBuffer only contains a few number of items after scanning the whole file.

4.2 Building our own Framework

Due to the difficulties experienced with the Idomaar system, we looked into whether it could be a preferable option to build our own recommendation framework. In addition to being unstable and complicated to understand due to the bad documentation, the Idomaar evaluator was also relatively slow, talking a couple of minutes to run only 100k recommendation requests. When considering the size of the entire dataset, where there are roughly 170 million requests, and they all should be processed within a reasonable time-frame, it was clear that the provided components would not suffice.

The solution was that we spent a full weekend putting together a completely new framework in Java, with an architecture optimised for performance. The main advantages was that instead of reading and parsing the JSON lines each time we ran the framework, the essential content from the data was extracted and stored at disk as serialised objects. So when loading the data in future runs, the data could be retrieved from disk in only 15 seconds per dataset compared to several minutes as for the Idomaar module. In addition the new framework utilised an object oriented design for storing Users and Articles, where the relationships were stored by pointers, making easier to work with, and extremely fast to extract what articles has been read by certain users, and visa versa.

In addition we implemented a Python server to deal with communication between the Java framework and our recommendation components written in python. This so the Java framework can act as a client upon recommendation requests, by forwarding the requests to the python module, which then runs the algorithm and responds with the result.

4.3 Implementation of Recommendation Algorithms

4.3.1 Approach 1

For the Collaborative filtering implementation, we simply extended the framework with an additional class that handled all the necessary methods and datastorage. We began by building the offline profiles for each user. We first split the dataset into two, 14 days for initial training(profile building) and 14 days for testing. As we can see from the appendix, the total amount of users observed during the first 14 days are over 2.5 million. Many of these however does not have enough activity in order to contribute well to collaborative filtering. By setting a minimum

requirement of read articles to 20, we reduced the number of user by 91% to 222 000 users for the first 14 days. Since the complexity is quadratic, this results in a substantial reduction in computation time. The users are compared by cosine similarity, and since we are dealing with binary vectors, the cosine equation can be simplified to:

$$cos(u, v) = \frac{size(I_u \cup I_v)}{\sqrt{size(I_u)} + \sqrt{size(I_v)}}$$

which is quite fast to compute. In addition, by utilising multithreading, the similarities can be calculated in parallel at even grater efficiency.

The run-time for computing the similarities matrix for the amount of users described above is roughly 6-8 hours on a 16 core machine with 128GB of ram. Unfortunately we did not have enough time to both implement the remaining online similarities calculations and the final recommendation functions, so we could not test the model properly.

4.3.2 Approach 2

In second approach, we intended to use one class SVM to analyse keywords extracted from data streams. Figure1 shows the overview of the approach used for news recommendations.

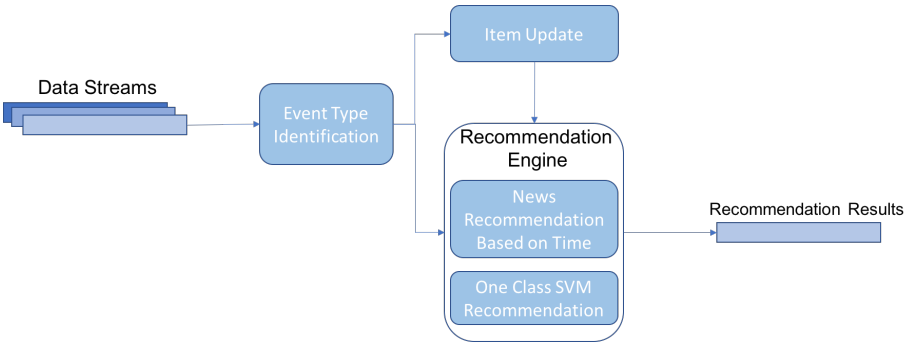

Figure 1: The overview of SVM-based approach

When the server receives data streams from client called by java functions, the event type identification module will send the data into different process modules according to the type of the data. For "item_update", item update module will extract domain id, item id, text, title and update time from json format data, and save this data into item map. For other kinds of data, recommendation engine module will form and update user profile and newest topic map by extracting user id, item id which the user is reading, and the correspondent keywords of this item. User profile includes the blacklist of the items that the user read before and now, and the profile contains all the items and their correspondence information, as shown in Figure2

From the beginning, the user profile is nearly empty, then the recommend results are based on the newest arrived items extracted from newest topic map. When the size of user profile begins growing, and the number of items the user have read is bigger then 10, which is the self-defined thresh hold, one class SVM is considered to use. In our case, one class SVM is trained by user's

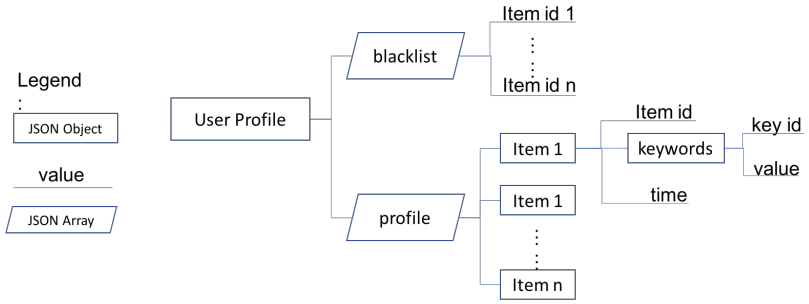

Figure 2: The data structure of user profile

historical items, and the trained model is used to predict the rest of the items in newest item map which are not in the user's blacklist.

We need to note that the keywords cannot be extracted from "item_update" items, and they can only be acquired from analysing other kinds of items in data streams. Furthermore, when the user historical records (items in user profile) are not representative enough for model training, which means only few item candidates or none of the item candidates are suitable to put in the recommendation list according to the results predicted by svm model, then newest item recommendation method is used to make up the recommendation results list.

Figure 3 is the implementation of training and predicting svm model, and Figure 4 shows the implementation of newest item recommendation method.

```
109 def rec_svm(user_array, item_array, item_list):
110     from sklearn import svm
111     print ">>>recommendation with one-class svm...",
112     if len(user_array) == 0 or len(item_array) == 0:
113         return []
114     num = len(user_array)
115     starttime = datetime.datetime.now()
116     clf = svm.OneClassSVM(nu=0.1, kernel="rbf", gamma=0.1, tol=0.0001, max_iter=5000)
117     clf.fit(user_array)
118     endtrain = datetime.datetime.now()
119     print ">>>training time: ", (endtrain-starttime)
120     item_predict = clf.predict(item_array)
121     endpred = datetime.datetime.now()
122     print ">>>predict time (per item): ", (endpred-endtrain)/0.1*len(item_list)
123     recommend_items = []
124     for i in range(len(item_predict)):
125         if item_predict[i] == 1: recommend_items.append(str(item_list[i]))
126     return recommend_items
```

Figure 3: The implementation of one class svm

During the development process, we planed to deploy SVM training and predicting process in an online environment from the beginning. However, as more and more data streams went through the recommendation engine, the size of the user profile for a specific user increased along with the size of item candidates map. Meanwhile, the feature space $\mathbb{F}$ also expanded

```
def cmp_datetime(a, b):
    a_datetime = datetime.datetime.strptime(a, '%Y-%m-%d %H:%M:%S')
    b_datetime = datetime.datetime.strptime(a, '%Y-%m-%d %H:%M:%S')
    if a_datetime > b_datetime: return -1
    elif a_datetime < b_datetime: return 1
    else: return 0

def rec_hot_new(domainid, blacklist):
    global new_topic, hot_topic, item_topic
    print '>>>rec hot news... ', domainid
    if domainid not in item_topic: return []
    item_list = item_topic[domainid]
    item_list.sort(cmp=cmp_datetime, key=operator.itemgetter('updated_at'))
    recommend_items = []
    for item in item_list:
        if len(recommend_items) >= 6: break
        if item["itemid"] in blacklist: continue
        else: recommend_items.append(str(item["itemid"]))
    return recommend_items
```

Figure 4: The implementation of newest item recommendation

as more keywords were explored among the items. As a result, both training and prediction time increased dramatically. Case in point, to train a $10 \times 40,847,958$ matrix would require 1.5s. Since the average recommendation request is about 7,000,000, the training time would be more than $1.5 \times 7,000,000$ which is approximately 10,500,000s – or, equally, more than 121.5 day. Therefore, this approach is too time-consuming to be used in an online environment. Also the cpu is also nearly 100 percent occupied during the SVM-recommendation process, which substantiate the the need for more efficient methods.

4.3.3 Approach 3

We started the experimentation with the third approach by installing Apache Spark on one of our servers. Spark is high performance distributed computation framework that runs on top of the Apache Haadop ecosystem. The Spark framework has a package call MLlib, which includes several algorithms for for machine learning methods. In our case we were interested in using the Principal Component Analysis(PCA) and Singular vector decomposition(SVD) algorithms for reducing the m x n rating matrix into a d x n matrix.

After the dependencies for Spark where up and running, we made some additional components to our custom built framework in order to efficiently extract the binary rating matrix. Luckily spark has a sparse vector class that enables us to store the user representation (in terms of articles read) as indexes instead of the full vector representation. Since there are 2.5 million users in the training set where many has only read a few articles, performing the calculations on all is most like not going to yield any good results. We therefore pruned the rating matrix for any users that has read less then 20 articles, and that has had a minimum of 2 sessions. In addition we only considered articles that has been announced by item-update, and the remain-

ing articles which were only observed during recommendation requests were discarded. This left us with a little more than 102 000 users, and 4900 articles. The rating matrix was 0.989 sparse.

Our goal was now to reduce the 100k x 4900 rating matrix into a d x 4900 representation. First we applied the SVD method with d=100. Unfortunately this only yielded very low values for all items at each dimension, typically in the range of 10^{-3} to 10^{-6} . Thereafter we experimented with setting the target dimension d to lover values such as 50 and 10, and higher values such as 200, 500 and 1000. In all cases the resulting right-handen SVD components yielded very low and unusable values, so we thought the current data might not be applicable to this approach.

As we had failed to project our rating matrix into a lower dimensional feature space with SVD, we tried the same approach with Principal Component Analysis. Unfortunately, after many attempts with different target dimensions, we were only able to extract embeddings with extremely low values. Since this work was just a few days before the project deadline, we concluded that the approach was not going to give adequate results within the time we had left, and decided not to work on it further.

In retrospect, using dimensionality reduction is not the only way for the neural network models to learn the collaborative embedding space. One possible approach is to calculate all item-item cosine similarities, and further construct a siamese neural network where two identical weight-shared sub-networks with different articles as inputs are trained to map with the correct distance in the collaborative space. This might be something to look into in future works.

4.3.4 Approach 4

In this approach, we plan to use collaborative filtering model to do the recommendation. The main steps are as follows:

1. Constructing a heterogeneous network containing the nodes which represent users, item-click, domain id, update days and other entities related to those items.
2. Use DeepWalk algorithm to generate random walks over this graph.
3. Based on the results from random walks, embed the graph in a low dimensional space using word2vec

The heterogeneous network are presented in Figure 5. Each edge represents the two connection nodes have relationship between each other.

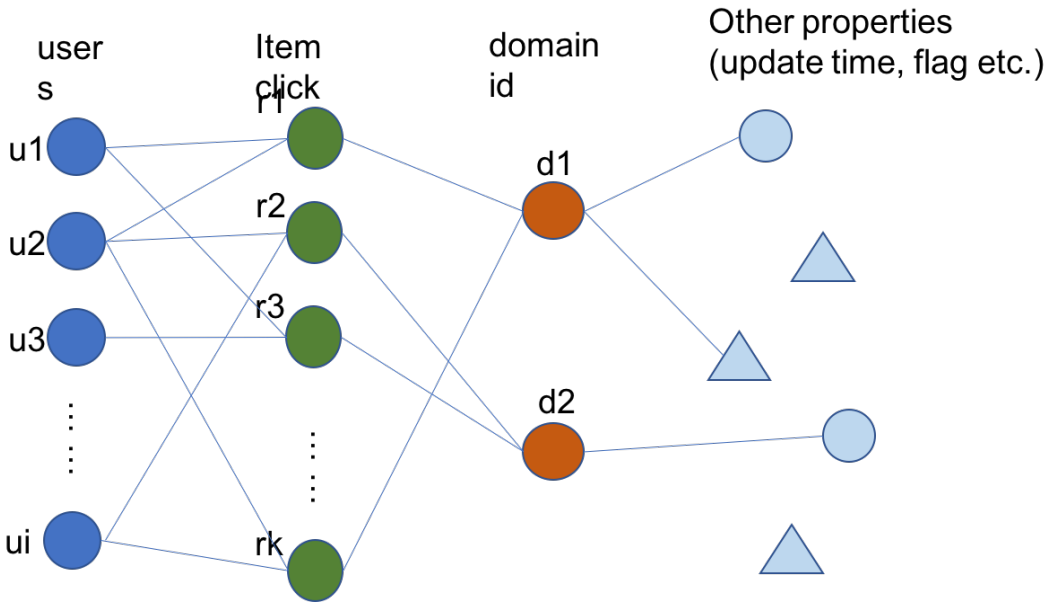

Figure 5: The heterogeneous network in out approach

In building this network, we use 100,000 dataset for model training and 100,000 dataset for predicting.

5 Results and Conclusion

As for now, the code delivered together with the report will not give any fruitful results or be able to perform evaluation. This is of course regrettable, although much fault lies in the CLEF NewsREEL. Idomaar has great premise, but ultimately proved to be more of a liability than an asset, as a number of issues occurred in the first half of the the projects duration, forcing us to decipher the maze-like structures of the framework to find the needle in the hay.

A week and a half before deadline, however, we decided to pivot, and build our own framework from the very ground. Although time consuming, we managed to make a code base which worked ideally in the context of the project. However, there was barely any time left to implement the designs we had though of by the time the framework was up and running. We think that it would have been fun to see how well these recommendation systems would perform, and hope that they will be tested out in the future.

At last, the following results are from approach 4:

```
260440
Word2Vec(vocab=26044, size=64, alpha=0.025)
MSE = 1.707113
accuracy = 0.740403
[[74041  7942  5768  8043  4207]
 [    0     0     0     0     0]
 [    0     0     0     0     0]
 [    0     0     0     0     0]
 [    0     0     0     0     0]]
```

Figure 6: The result from graph-based collaborative filterin g

6 Appendix

6.1 Overview of The CLEF news data

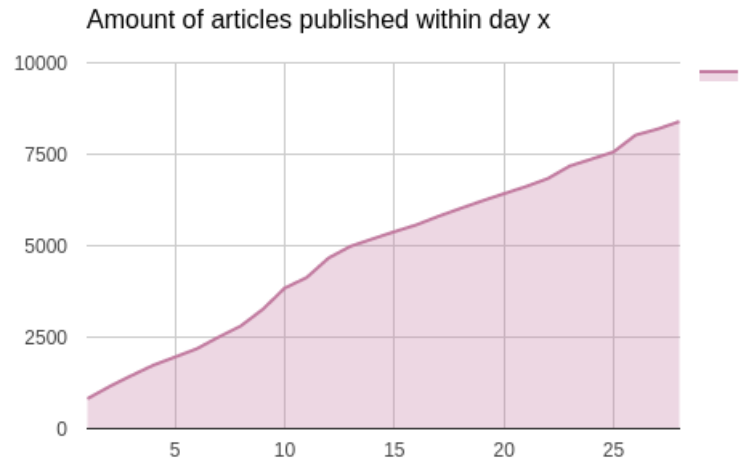

Figure 7: Statistics over total amount of articles presented by item-update

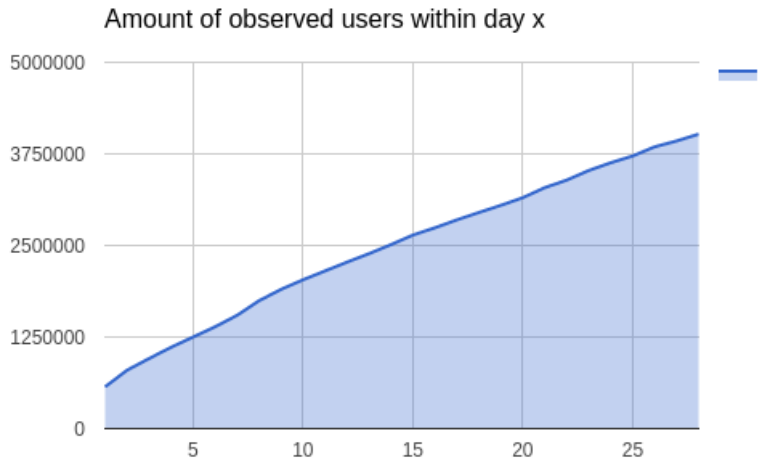

Figure 8: Statistics over total amount of observed users

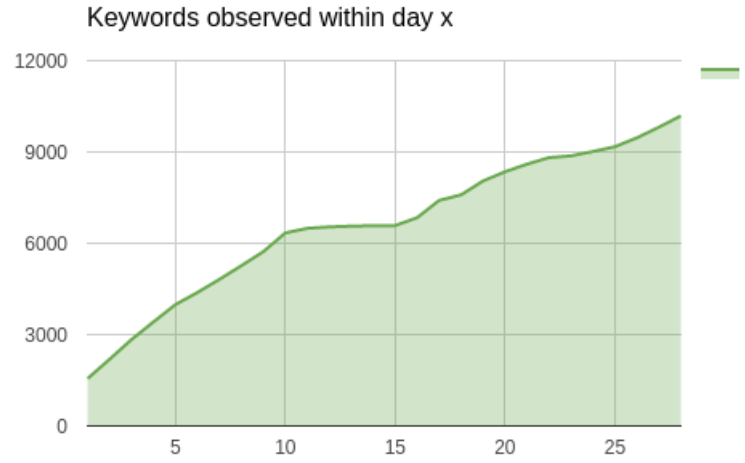

Figure 9: Statistics over total amount of observed keywords

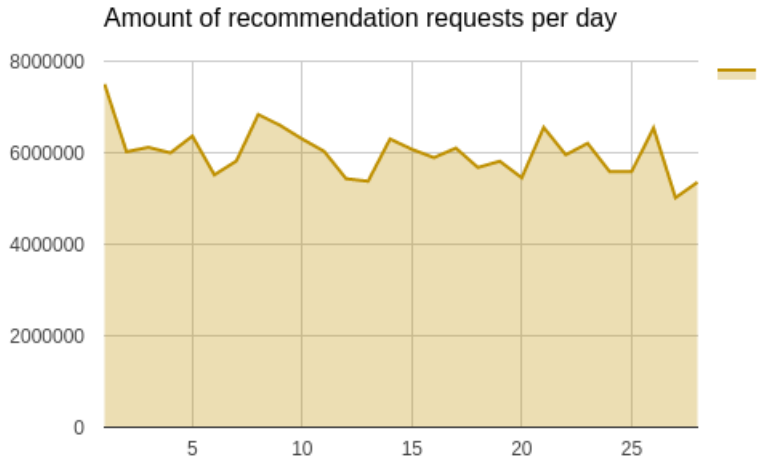

Figure 10: Statistics over number of recommendation requests per day