UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Maksym Sukhorukov

A network science and document similarity based hybrid job recommendation system

Master's Thesis (30 ECTS)

Supervisor: Rajesh Sharma, PhD

Tartu 2018

A network science and document similarity based hybrid job recommendation system

Abstract:
Job recommendation systems mainly use different sources of data in order to give the better content for the end user. Developing the well-performing system requires complex hybrid approaches of representing similarity based on the content of job postings and resumes as well as interactions between them. We develop an efficient hybrid network-based job recommendation system which uses Personalized PageRank algorithm in order to rank vacancies for the users based on the similarity between resumes and job posts as textual documents, along with previous interactions of users with vacancies. Our approach achieved the recall of 50% and generated more applies for the jobs during the online A/B test than previous algorithms.

Keywords:
Job recommendation system, network science, document similarity.

CERCS::P170, Computer science, numerical analysis, systems, control

Võrguteadusel ja dokumentide sarnasusel põhinev töökohtade soovitussüsteem

Lühikokkuvõte:
Tööde soovitussüsteemid kasutavad erinevaid andmeallikaid lõppkasutajale parema sisu tagamiseks. Hästi toimiva soovitussüsteemi arendamine nõuab keerulisi hübriidseid lähenemisi sarnasuse kujutamisele põhinedes töökuulutuste ja resümeede sisudele ja nendevahelistele interaktsioonidele. Antud töö tulemina arendati efektiivne võrgul baseeruv töökohtade soovitussüsteem, mis kasutab Personalized PageRank algoritmi töökohtade järjestamiseks põhinedes tööotsija resümee ja töökuulutuse kui tekstiliste dokumentide sarnasustele ning eelnevatele kasutaja ja töökuulutuste vahelistele interaktsioonidele. Meie lähenemine saavutas 50%-lise saagise ja tekitas online A/B testi jooksul rohkem kandideerimisi kui eelmised algoritmid.

Võtmesõnad:
Töökohtade soovitussüsteem, võrguteadusel, dokumentide sarnasusel

CERCS::P170, Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Acknowledgement

I would first like to say thank you to my thesis supervisor senior research fellow Rajesh Sharma for guidance, valuable ideas, and help.

Also, I would like to express my gratitude to my colleagues Alex and Tigran, who helped me with a lot of struggle. and pushed my ideas into the life.

Finally, I want to thank my family and my girlfriend Anastasiia for the support, love, patience and understanding.

Contents

1 Introduction

In the modern era the process of searching for a job has moved from traditional offline job search offices to various online platforms. There is a high competitiveness in the industry and every company tries to hold their market positions by providing better experience both for the job seekers and employee seekers. Most of the job search platforms store a lot of textual data such as resumes, job advertisements and interaction with them. The main challenge of such platforms is to match users' interests with the relevant job.

Recommendation systems are the common information retrieval domain which can be explored for such problems. These systems have been drastically improving in past years with the simplification of data collection and storing, cloud platform development, as well as advances in machine learning. We are able to explore the help of the recommender systems in our daily life searching for movies on Netflix[1], listening to music on Spotify[2] or buying goods on Amazon[3]. Most of the system use interaction between users and items in order to provide the user with similar items, which forms a sub-domain called Collaborative Filtering. Some systems focus heavily on the features of the items and trying to find preferred features for users in order to make recommendations, which forms a sub-domain of Content-based recommendation systems.

Talking about job recommendation systems, most of them try to use hybrid approaches which would allow combining both contents of the job posting, resumes, and interactions between them. There are plenty of possible approaches with their advantages and disadvantages. The choice of approach to building a job recommendation system varies depending on the structure of data, amount of users and market specifications.

In this thesis, we explore how network-based approach in combination with appropriately learned content similarities of job postings and resumes can improve the recommendation system. The main idea of the approach is to use Personalized PageRank algorithm on the graph of interactions between users and vacancies. We also enrich the graph with edges which shows the similarity between entities. We try to find similar documents by using the compact vector representation of text and discovering which of the existing approach would work better in our case.

We evaluated our approach using a real dataset of online job search platform, located in Europe. The platform serves tens of thousands of unique users daily. During our experiment, we were able to achieve the recall of 50% in off-line setup, as well to generate more applies then the previous system during the online test on real users also achieving a conversion rate of 17.5%

This allowed company to start researching more deeply into alternative approaches for the recommendation systems which would help them to increase key business metrics such as user activity and satisfaction.

[1] `https://www.netflix.com/`

[2] `https://www.spotify.com/`

[3] `https://www.amazon.com/`

The rest of this thesis is organized in the following manner:

1. Related work: The chapter describes the findings of the researchers in the specific field of job recommendation systems;

2. Theoretical Background: In this chapter, we discuss common approaches in recommender systems, state-of-the-art methods of getting vectors from the texts, along with the common graph-based approaches from network science, which can be applied to recommender systems;

3. Datasets description: The chapter describes the data we used for building recommender systems and the process of cleaning textual data for the similarity retrieval module

4. Methodology: The chapter describes an architecture of the system along with the approaches we have tested during development

5. Experiment and Evaluation: In this chapter, we describe the approach for splitting the dataset into training and validation, so that it would reflect on-line approach as close as possible. We discuss evaluation metrics we used in order to measure performance as well as final results of experiments.

6. Conclusion: The chapter describes key findings and future work

2 Related work

Since the rise of the digital era, online job portals always needed designing automatic recommendation system in order to serve their users the best content and provide the job posters with candidates as quickly as possible. That has led to a decent amount of research in this particular field. The approaches used in the previous related research could be completely different from each other and depend on the type of data job portal operates market specification, architecture patterns, business needs, etc. However, all of them can share useful insights on how the field is developing and which techniques are most commonly used.

1. Siting, Wenxing et al. (2012) surveyed known recommender systems and overviewed existing approaches to learning user profiles and performing recommendations based on them. They state that current personalized recommender systems focus on learning better user profiles by using collaborative filtering, content-based approaches, and reciprocal recommender. The notice that the best user profile can be learned by using hybrid features. [SWNF12]

2. Another survey paper by Al-Otaibi & Ykhlef (2012) which covers significant findings in the domain by that time. Authors discuss different types of data, which are commonly used by job recommendation systems, also talk about types of recommender systems and again point out how hybrid systems outperform pure collaborative or content-based systems. Moreover, they overview some of the hybrid approaches: such as probabilistic approach, semantic systems, fuzzy methods. They state that collaborative filtering uses an insufficient amount of data and in such domain, one has to use content-based features in order to achieve better performance. They also claim that there is a high potential solution for creating better systems in the application of machine learning algorithms.[Alo12]

3. Lu, Yao et al. (2013) proposed an exciting graph-based approach. They construct a graph, consisting of various links based on different criteria's. This links might be a content-based link: similarity between CV forms a connection in the graph, and also collaborative links: interactions between users and vacancies also forms a link. They later use the PageRank algorithm in order to rank recommendation for users. [LEHG13]

4. Another graph-based approach was proposed by Shalaby et al. (2018). They use the homogeneous approach where nodes are job posts, and edges are measures of similarity between them. They rely on the behavioral data, having weights as the number of co-interactions between the job in the user session, as well as on the content of job posts learned using deep learning embeddings of job posts. They aggregate the scores using heuristic and then perform recommendations via PageRank algorithm, using users history of interactions between jobs. [**?**]

5. Elsafty, Riedl and Biemann focused on improving their existing content-based recommender system using dense vector representation of job postings. They compared different text vector representations such as classical TF-IDF, doc2vec, word2vec, and modifications of these approaches. They also discussed the ways of evaluating their approach both offline and online using A/B and A/A tests. [ERB18]

6. Schmitt, Caillou, and Sebag (2016) proposed a hybrid approach of matching jobs and resumes using a deep neural network. First of all, they discussed the crucial insights that helped us with one of the ideas. They state that job posters and job seekers speak different languages, which means that one can not compare document embeddings directly from job posts and resumes. Then, they have used the different approach for document embeddings and also they added collaborative learned vectors, which they receive with the help of SVD. They also present a novel approach to estimate collaborative filtering vectors in order to fight cold-start problem. [SCS16]

Considering the reviewed papers above, we can assume that in the field, hybrid systems perform better than using solely content-based or collaborative filtering approach. We also can note that many state-of-the-art systems focus on improving item and user representation. Other valuable insights are that Natural Language Processing techniques and especially document embeddings into vector spaces are used a lot in order to obtain similar job posts or users by their resume. Much research found out that network-based approaches perform very well for the job recommendation task. All in all, we designed our job recommendation system with the help of useful insights received from related works.

3 Theoretical Background

In the first part of this chapter, we give a general overview of recommendation systems, conventional approaches, and types. Next, we discuss state-of-the-art methods of receiving numerical vectors from documents their advantages and limitations since our approach includes finding similar documents based on the text. Finally, we briefly overview common network-based approaches which we can map on recommendation problems.

3.1 Recommendation systems

A recommendation system is a group of models, which usually uses two kinds of data: the first one is an interaction between users and items, such as clicks, purchases, views, listening, etc. and the second one is information about these users and items also called attributes. Using this features, it tries either to predict some items, which are the most relevant for a user (user-item recommendation system), some users which are the most relevant for an item (item-user recommendation system), some items which are relevant for an item (item-item recommendation system) or some users which are relevant for a user (user-user recommendation system).

Depending on which data recommendation system is using, it is divided into three groups: collaborative filtering methods, content-based methods and finally hybrid based methods, which unite those two. [RRS15] This chapter describes the most commonly used methods.

3.1.1 Collaborative filtering

Collaborative filtering is a method, which predicts user preferences in item selections, based on the already known user feedback for an item, which can be divided into two basic types of algorithms: memory-based and model-based. [BHK98] In addition to this, algorithms are often distinguished by the type of feedback: it could be either implicit or explicit feedback. Explicit feedback means that user gave a rating to an item, for example, 5 - excellent, 1-bad; while implicit feedback is based on user purchase history, click history, etc. In this work, we deal with implicit feedback.

Memory-based algorithms. In the basic version of memory-based user-item prediction, we have a data of all users interaction with items. Our goal is to calculate prediction on a rating (in case of explicit feedback) or probability of action (in case of implicit feedback). In the first phase of memory-based algorithms, we compute similarities among all users (using the vector of user action), then we select top N similar users, then we remove already known probabilities of user reacting to item (those which happened), and calculate predictions.

There are different variations of basic model: for example correlation between users past action can be used to predict future actions [RIS+94], vectors can be viewed in the same way as text vectors or even weighted using inverse document frequency approach. [SB88]. Approaches also vary from the choice of distance metric in order to find similar users: it could be Pearson correlation, cosine distance, Jacquard index and their various modifications.

This approach suffers a lot from data sparsity which leads to poor prediction quality. In addition to this, model-based approaches only counts exact match on older actions, this problem can be addressed using some weighting factor or threshold which would give smaller similarity weights to users being similar having small number of elements in common. [MKL07]

Another extension to the memory-based algorithm, which uses correlation - is introducing approaches from graph theory in order to predict user preferences. For instance, the ItemRank algorithm sees the problem as a bipartite graph consisting of user nodes and item nodes, where links are an interaction between users and items. It uses the Random Walk approach on the graph to find new connections and give them a score. [GP07]

Model-based algorithms While memory-based algorithms focus on finding similarity between users, model-based algorithms use machine learning models, which requires training the model and fitting its parameters in order to achieve better accuracy of recommendations. [LSL12]

First of all, collaborative filtering could be possibly generalized as a classification or regression problem. In classical machine learning, we have features or independent values which tries to predict missing dependent value. In recommendation problem - any of the columns, which is usually treated as independent value can be called a dependent one, for one sample (user or item) or as independent for another. So the core idea of generalizing standard machine learning regression and classification technique is to try to impute potentially missing values from rating matrix R by using known entries as independent features. In addition to this there is a big difference in having the test and train rows because rows in collaborative filtering can contain any amount of missing entities, so we can instead talk about the test and train entries.[Agg16]

Figure 1 illustrates the difference between classical machine learning and recommendation systems modeling. Considering this, we can run any supervised algorithm, like logistic regression [BMS06], decision tree [GML+10], Bayesian models [MP00] or deep neural network [HLZ+17]. We can treat each column of the initial matrix of ratings R as independent value and other entries as a dependent one and in this way predict all the missing entries. Of course, this approach extremely suffers from high-dimensionality and is very complex computationally.

Since generalizing the collaborative filtering problem to classification and regression

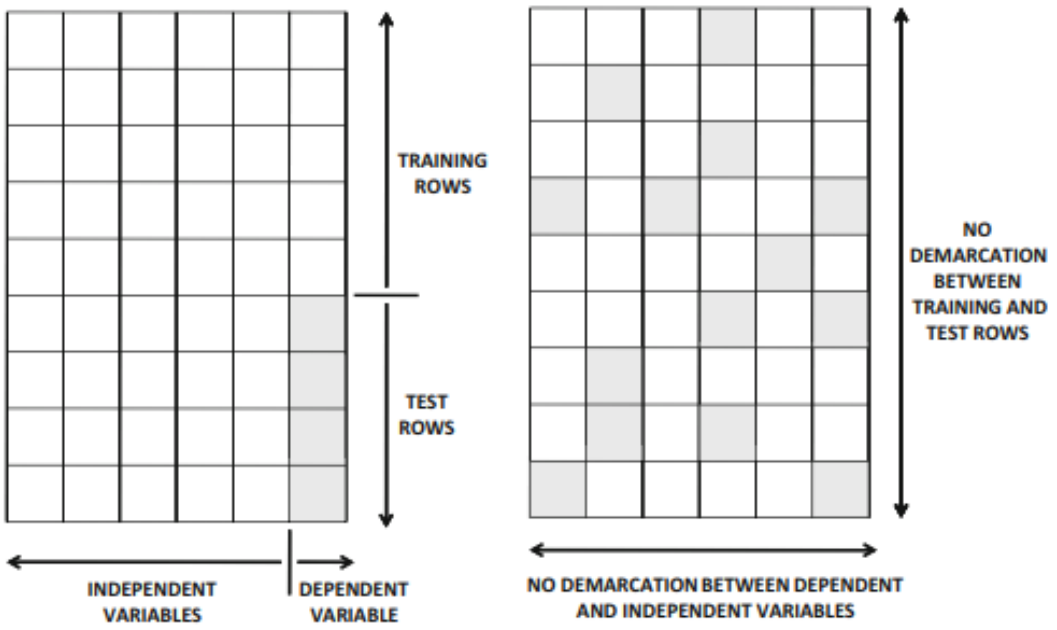

Figure 1. Comparing the traditional classification problem with collaborative filtering. Shaded entries are missing and need to be predicted [Agg16].

problems is often problematic, Matrix Factorization has become one of the most popular and commonly used model-based algorithms for collaborative filtering. In its basic form matrix factorization characterizes items and users with some factor matrix which is based on the rating matrix, in this way high correspondence between obtained factors leads to a recommendation or prediction with high score. [KBV09]

Matrix factorization aims to map both users and items to a joint latent space, where interactions between users and items are modeled as a simple dot-product of less sparse matrices than initial rating matrix. So we can formalize it with the following equation:

$$R \approx PQ^T = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_f \end{bmatrix} \begin{bmatrix} q_1^T & q_2^T & \cdots q_m^T \end{bmatrix} \tag{1}$$

Where P is a user factor matrix, and Q is an item factor matrix. The idea of the matrix factorization algorithm is to learn those user and item factor matrices so that their product would be as close as possible to initial rating matrix R. [ZdLCG+14]. In this way, as noted the prediction of rating for each user to an item previously can be simply

calculated as dot-product of corresponding vectors.

There are many ways to get user and item matrix. One of them is SVD (singular value decomposition). Using SVD we try to reconstruct our matrix R as a form of $SVD(R) = P\Sigma Q^T$ where P and Q are orthogonal matrices and diagonal values of Σ are called the singular values of matrix R, which are placed in descending order. The columns of P and Q are called left and right singular values. We can formulate an optimization problem for singular value decomposition as to minimize $\frac{1}{2}\|R - UV^T\|^2$ in such way that columns of P and V are mutually orthogonal. In other words, we are aiming to minimize the sum of the squares of the residual matrix $R - UV^T$. This optimization problem is usually solved by applying stochastic gradient descent or alternate least square with the coordinate descent. [Agg16]

In addition to performing classical SVD, we can also map R to even lower dimensional space, which is called Truncated SVD by only using first k rows of matrices P and Q, since their singular values are in order. [BGM15]. With SVD the prediction and recommendation task is done by imputing missing values in the rating matrix and then performing matrix factorization.

Another addition to SVD would be a Non-Negative Matrix Factorization, which is very similar to standard SVD but has additional constraint that all values in matrices P and Q should be non-negative. This increases interpretability and can perform better in problems with an implicit dataset. Also, different regularization technique during the optimization process can be applied to SVD in order to deal with over-fitting and in learning the better model. There are plenty of other techniques to perform matrix factorization, including probabilistic models, such as PLSA which instead of minimizing Frobenius norm objective, maximizes maximum-likelihood or maximum margin method inspired by support vector machine, which uses hinge loss as an objective. Nevertheless, all the methods of Matrix Factorization share the same idea and their extensions and restrictions try to optimize solution and could work better or worse depending on the particular problem. [Agg16]

3.1.2 Content based recommendation systems

As discussed previously, while collaborative filtering methods use users rating for an item in order to recommend for the user, content-based models use information about user and items as the main data source. In general, content-based models try to describe the items that might be recommended, and then it is creating a profile of the user that describes the types of items the user prefers, and a means of comparing items to the user profile to determine what exactly to recommend. [PB07] So similarity between users in content-based models is represented not as a correlation between given ratings or interactions, but by attributes of items those users liked or interacted with.

Most content-based recommender systems use the textual description of the items, for example, movies description, job posting description, meta-information: i.e music

genre, keywords from scientific article etc. [Agg16].

Content-based models rise in problems with the big amount of information about the item, and they could better deal with a cold-start problem (bad quality of recommendations for a new user). However, unlike the collaborative filtering models, content-based models requires much preprocessing, feature extraction, working with textual data. The common process of creating content based models usually consists of three steps: item similarity modeling, user profile modeling, and models for giving recommendations with data obtained after the first two steps. [LSDJ06]

Common approaches for the first step, item representation modeling are vector-space models, keyword-based models, semantic models, network models. In vector space modeling the main goal is to reconstruct text to vector space. In content-based recommender systems which rely on the vector-space model, both items and users are represented as weighted term vectors. Predictions of a user's interest in a particular item can be derived by computing the cosine or any other type of similarity. [FR15]

The process of learning user profiles is usually done using machine learning classification approaches. For instance, in recommender systems user somehow labels items, expressing his likes or dislikes, so in this way, we can transform it into the form of binary classification. Items, as discussed previously are usually represented as some numerical entities. Then, the learned user profile for a specific a user is a machine learning model, developed based on this user interaction with items. The most common approaches for learning user profiles are Bayes classifiers, nearest-neighbor classifiers, rule-based classifiers, regression-based classifiers and deep learning models. [Agg16]

Content-based models and collaborative models have both advantages and disadvantages. For example, both approaches suffer heavily from the cold-start problem: when the user has not made any interaction with items yet, but content-based systems do not suffer from the item cold-start problem since interactions are usually not required to learn item profile. Also, content-based systems are easy to explain, while it is often not achievable with collaborative filtering. Content-based systems also suffer from overspecialization, meaning that it would recommend the narrow scope of items to a user. In some cases recommending the similar items all over again is not the best option. Despite all of the mentioned disadvantages, content-based systems can complement collaborative systems with information about items and users and both approaches can be united together into a hybrid recommender system. [Agg16]

3.2 Vector representations of texts

In order to retrieve the similarity of desired text documents: job postings and resume texts, we have to represent texts in vector or numerical form, such that it would be easier to compare them with each other by finding distances. In this section, we discuss state of the art methods to receive document embedding vectors.

Given a pair or set of some text documents, it is possible for a human with some domain language to find which are similar to each other, and even identify some degree of similarity or rank them. In order to find out which documents are similar automatically, have to compute some similarity coefficient between them, and to compute it, we have to represent this documents as vectors. Thus, we have to find out what is the best representation of the particular set of documents in some vector space.

Bag of words. The advance of text representation for the proper computer-readable format has begun from most straightforward BOW (bag of words). [Har54] It was one of the first fixed length text vector representation and remains one of the most common baselines for document similarity [WM12] despite its simplicity. This approach compares documents by the set of words used in them, representing how much some word contributes to the semantics of a document, which might perform well in most cases. However, it fails in capturing word order in documents, their context, as well as the similarity between synonymous words and phrases. Another significant drawback of the bag of words is its frequent near orthogonality [GC06]. This approach can also be advanced to the bag of n-grams [Car00], which could lead to statistically significant improvements only with quite a specific dataset. [BA03]

Term frequency-inverse document frequency. Another baseline representation of text is the term weighting approach using inverse document frequency [SB88], which is also known as TF-IDF. The value of TF for each word in corpus can be calculated as

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \tag{2}$$

which is the frequency of term occurring in a particular document. Where inverse document frequency is calculated as

$$idf_i = \log \frac{|D|}{|d : t_i \in d|} \tag{3}$$

which is the logarithm of the total number of texts we have divided by the number of texts where this particular term occurs. The logarithm is taken in order to smooth big and small frequencies. IDF also could be interpreted as the amount of information of the term. [Rob04] By the definition, TF-IDF is a multiplication of noted previously terms $tf_{i,j}(2)$ and $idf_i(3)$. Still, after introducing some term weighting, TF-IDF representation suffers from the same problem as BOW, since it also sees documents just in the form of the bag of words. Considering this fact, we can say that it still suffers from polysemy, synonymity, and order problems. However, the introduction of global weighting can improve vector representation of text comparing to the bag of words and give more accurate results.

BM25 There is another weighting scheme for bag-of-words, which is mainly used for search but could also be used to compare documents with each other, called BM25. It is very similar to TF-IDF but introduces several hyper-parameters to be tuned. So, for BM25 the TF part is calculated as:

$$tf* = \frac{tf(k+1)}{k(1-b+\frac{bDL}{AVDL}+tf)} \tag{4}$$

Where k and b are some hyper-parameters, DL is a document length, and AVDL is average document length. Hyper-parameters are introduced to normalize document length and to ensure the monotonic growth of TF part. In this case, idf is calculated similarly as in TF-IDF. So the BM25 vector would be the multiplication of (5) and (6).

Latent semantic analysis. To address the problems mentioned above, the Latent Semantic Indexing was introduced, which uses singular value decomposition technique to reduce dimensionality in the way that synonymous word occurs are presented near each other, which means that its eigendecomposes bag of words feature space. After computing co-occurrence matrix, we compute the standard SVD

$$N = U\Sigma V^t \tag{5}$$

Where U and V are orthogonal matrices $U^tU = V^tV = I$ and the diagonal matrix Σ has the singular values of N. Then the Latent Semantic approximation of N is calculated by assuming that all but the largest K singular values are zero, which would output a low-dimensional vector of documents and this could help with the curse of dimensionality, which is one more problem, TF-IDF is suffering with. Latent Semantic Analysis also tries to solve the problem that multiple terms can be referring to the same object. [STW$^+$]

Later, the Probabilistic Latent Semantic Analysis model was introduced, which presented Tempered Expectation-Maximization as a fitting procedure, which allows it to be better with large datasets. [Hof01].

We can generalize all of the techniques mentioned above into a single group called Vector Space Models, which has been a standard in information retrieval for years and is continuing to be still. Ignoring the word order and having many problems with semantic understanding of words, these models still are simple to implement, easy to use at the high scale as well as to parallelize.

Word2Vec There were quite a lot of trying to introduce Deep Learning and Neural Networks to text analysis and language modeling [BDVJ03]. It did not achieve immense popularity until the Word2Vec model was introduced in 2013. [MCCD13].

They have introduced two new architecture, which can transform a word or phrase (n-gram) to vector. The first one is a Continuous bag of words (CBOW model) and the second is skip-gram, which are schematically shown in Figure 1.

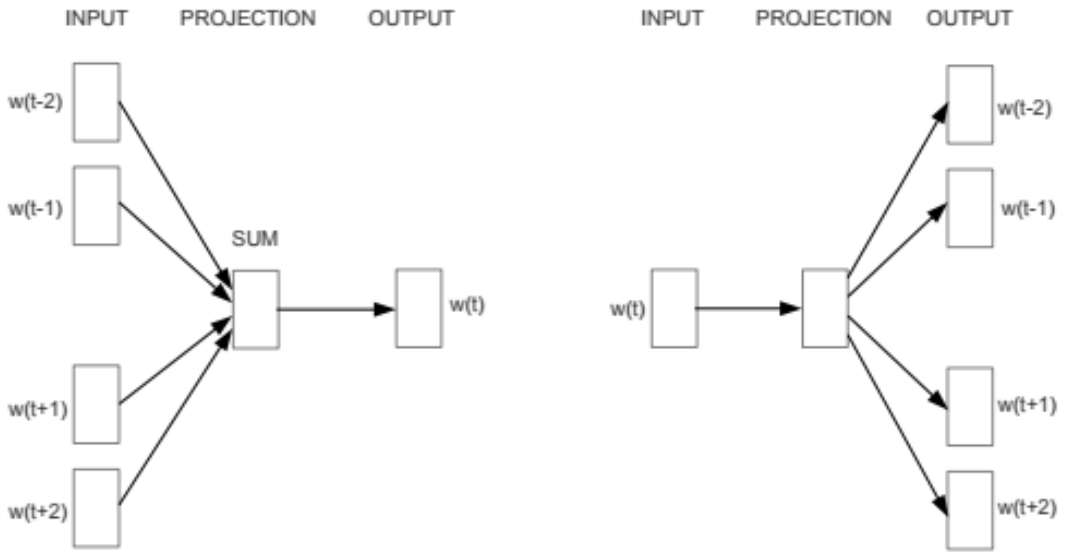

Figure 2. The CBOW and skip-gram Word2Vec architectures [MCCD13].

For every task there is some sequence of word, the model would be trained on. $w_1, w_2, ...w_T$. The CBOW models predicts the word based on its context. It takes the context vectors, calculate the sum of them and predicts the target. The objective function to maximize in this case is

$$\frac{1}{T}\sum_{t=1}^{T}\log p(\omega_t|\sum_{-c\leqslant j\leqslant c, j\neq 0}\omega_{t+j}) \tag{6}$$

where c is some window of context for each word, in which the sum will be calculated.

In CBOW the input layer of Neural Network represents the bag of words of the surrounding c words, and contains one input element per word. Then it is projected linearly to the hidden layer, which uses the Huffman tree to encode the word, which reduces complexity. Then, after the desired number of iterations, the network converges and middle encoding hidden layer represents each word. [WHBD14]

For Skip-gram, on the other hand, each word is predicted separately given its context.

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leqslant j\leqslant c, j\neq 0}\log p(\omega_{t+j}|\omega_t) \tag{7}$$

Softmax function defines the probability in here. They also use a more computationally efficient approximation in order to decrease complexity. [MSC$^+$13]

Both models also use negative sub-sampling, which randomly deletes words from context with a probability which is based on their frequency in other documents.

It is possible to iterate through the documents while learning word embeddings multiple amounts of times and to control dimensionality of vectors.

So, these models extract semantic relationships between words that co-occur with similar words, meaning they have the same context.

In order to receive a vector representing document from the word embeddings, one has to do some algebraic operations on these vectors, such as sum them, take an average of vectors in documents, or to use weighting scheme. This would allow later to compare the vector of documents and similarly to Vector Space Model - tell which documents are more similar to each other.

Doc2vec. After the success of word embeddings, the similar technique was introduced, that would learn fixed length representation not only for words but for sentences, paragraphs, and documents, called Paragraph Vector or Doc2Vec. [LM14]

The first model they use is a distributed-memory model, in which every paragraph is mapped to a vector, which is a column in some matrix D and every word is similarly mapped to some vector in matrix W. Then the vectors are concatenated to predict the next word in context. The logic is very similar to CBoW model, but instead of using just words in context to predict next word, it uses some previous word concatenated with unique document vector to predict this single word as presented in Figure 2. In here, each word is shared between paragraphs, but each paragraph is not shared between all corpus.

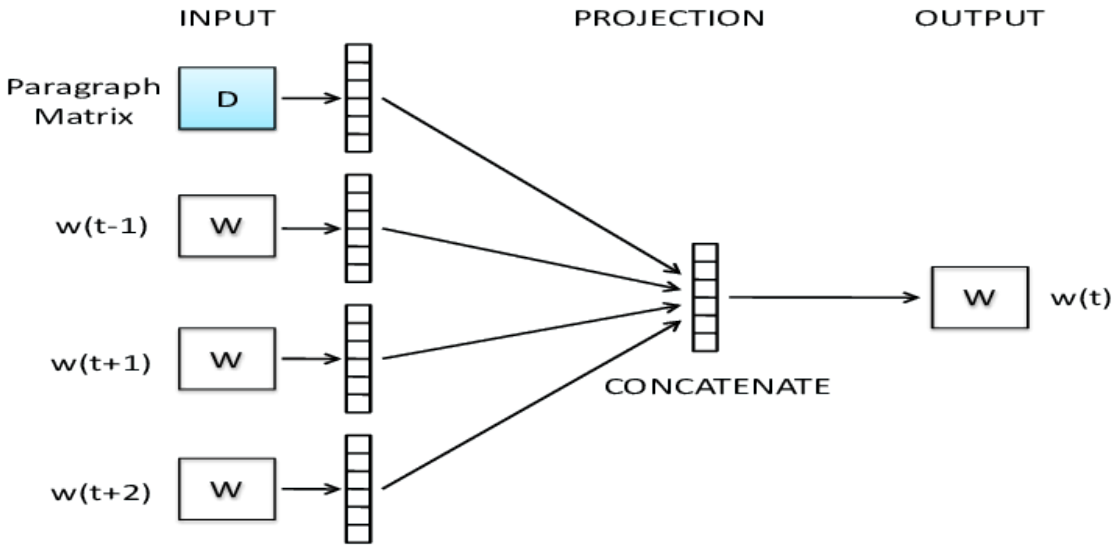

Figure 3. Doc2vec model with a distributed memory. [WKR+17]

In another model, called The Distributed Bag of Words the goal is to predict randomly sampled words from the context using the paragraph vector as shown in Figure 3, which is quite similar to Skip-gram model.

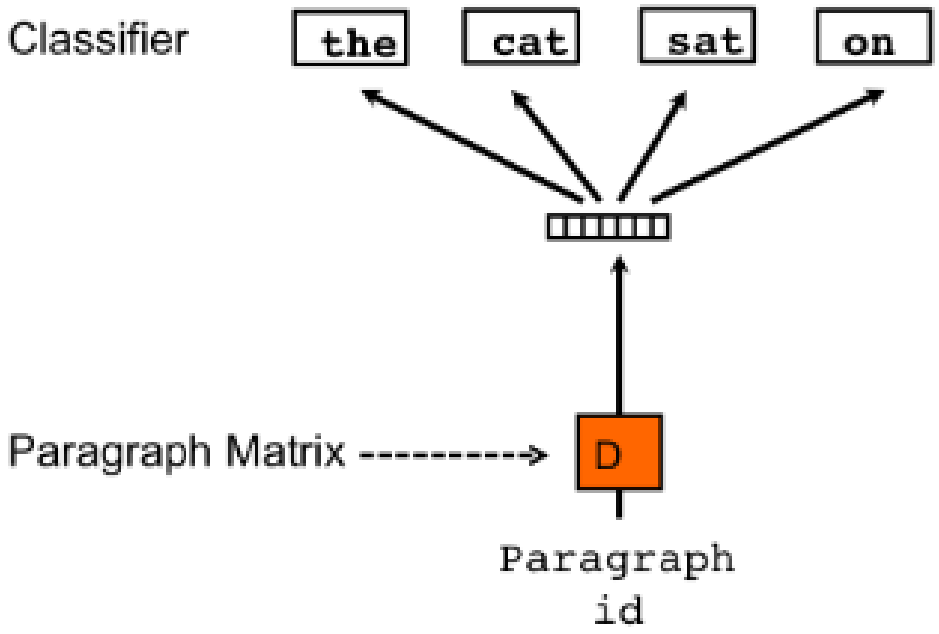

Figure 4. Doc2vec model with a distributed bag of words [LM14].

Both models are again trained via stochastic gradient descent, where the gradient is obtained through backpropagation. The authors also suggest, that PV-DM and PV-DBoW works best and are more consistent in combination, but could be possibly used separately with PV-DM being significantly better than PV-DBoW. There is also some variation of PV-DM model, which uses sum instead of concatenation.

Autoencoders: Another method to directly generate document embeddings is using autoencoders. The simplest autoencoder is a shallow neural network, which goal is to try to reconstruct the input at the output layer. An autoencoder has an encoder part, which maps an input x to a hidden layer: $z = g(Wx + b)$ and a decoder which goal is to reconstruct input: $\hat{x} = o(W'z + c)$ where b and c are some bias terms, W is weight matrix between input and hidden layer, while W' is a matrix between hidden and output layer. [CZ17]

In order to deal with overfitting, there has to be some regularization in autoencoders. Depending on what to use as a regularization, there are different types of autoencoders.

The designing autoencoder has core idea of having corrupted data as input and comparing the output of corrupted and non-corrupted versions. [VLL+10]. The variational autoencoder is using a stochastic variational inference, and the encoder in this model tries to approximate the true posterior, while the decoder generates data. [KW13] The k-sparse autoencoders use only k highest activities during feedforward phase and thus achieves exact scarcity. [MF13]

Comparing to using autoencoders with images, autoencoding texts is a much harder task, due to a big sparsity and other problems. The core goal of autoencoders in document representation as vectors is not only to minimize the error of reconstruction but also to extract the best features, which would allow comparing documents correctly. The KATE: k-competitive autoencoder model tries to overcome this limitation by adding constraints in training phase via mutual competitions. It means that in the end, each neuron should represent some pattern of data different from others. The core idea is to have neurons with large absolute positive and negative values of weight as the most competitive, thus discarding the low valued weights. [CZ17]

3.3 Graph-based recommendation methods

In general, graph-based approaches means that the data is represented in the form of a graph where nodes are users, items or both, and edges encode the interactions or similarities between the users and items. [FR15] Graphs are the compelling and simple abstraction that also allows using many algorithms from the network domain.

There are three standard approaches to building graphs for recommendation problems: user-item graphs, user-user graphs and item-item graphs for Neighborhood-Based Methods. [Agg16]

The user-item graph is usually defined in the form of an undirected bipartite graph with a set of nodes representing users and another set of nodes representing items. All edges in this approach usually exist only between users and items only if the user has rated number. This representation helps a lot in case of sparse datasets since the number of edges is equal to a number of observed interaction and there is no need in keeping large sparse matrices. In order to retrieve recommendation for the user, we have to use walk-based approaches to find indirect connectivity between nodes. This approach uses random-walk measures as well as Katz measure which makes these problems identical to the problem of link prediction in social networking analysis. [Agg16]

In order to find path-based similarity, the distance between two nodes of the graph is defined as a function of the number of paths connecting the two nodes as well as the length of those paths. The association between user u and item i can be defined as the sum of the weights of all distinctive paths connecting u to i (allowing nodes to appear more than once in the path), whose length is no more than a given maximum length K. [FR15] Maximum length is used because longer paths tend to become noisy for the predictions. The intuition is that if two users belong to the same neighborhood based on

walk-based connectivity, then there should possibly be the link between them. The Katz measure between nodes i and j is defined as $Katz(i,j) = \sum_{t=1}^{\infty} \beta^t n_{ij}^{(t)}$ where $n_{ij}^{(t)}$ is a number of walks of length t between nodes i and j and β is some user-defined parameter. In this way, we can predict top N user-item links that are likely to be formed. We can also enhance the performance by forming content-based edges between users. [Agg16]

Another way to obtain the similarity between nodes is a random walk. In this approach, the similarity between users and items is defined as a probability of reaching these nodes in a random walk. A random walk over a graph is a stochastic process in which we know the initial node, and the similarity between initial node and any other is explained by the likelihood of transiting from node i to node j in the graph. [ZZAP13] The random walk is usually described by a Markov chain describing the sequence of nodes visited by a random walker is called a random walk. A random variable $s(t)$ contains the current state of the Markov chain at time t. If the random walker is in this i state at some time defined by t, then $s(t) = i$. The random walk is defined with the following single-step transition probabilities of jumping from any state or node $i = s(t)$ to an adjacent node $j = s(t+1) : P(s(t+1) = j | s(t) = i) = a_{i,j}/a_i = p_{i,j}$ where $a_{i,j}$ is the element of symmetric adjacency matrix of the graph and $a_i = sum_{j=1}^{n} a_{i,j}$ [FPRS07]

The PageRank is a popular extension of Random Walk algorithm, and Google Founders developed it, Their idea is to present the algorithm which models the importance of nodes with the use of the linkage structure in the Web graph. [BP98] It deals with a common problem of simple Random walk algorithm, such as deadlocks, where random surfer stucks in the infinite loop. [Agg16] In order to overcome this, the model has a probability of surfer to restart the process and jump to a random node. So the formal definition of the random walk process in PageRank algorithm is following:

$$\vec{r} = cP^T\vec{r} + (1-c)\frac{1}{n}\vec{(e)} \tag{8}$$

In the above equation, $\vec{r}$ represents the PageRank vector, n is the rank score for a node, $\vec{(e)} = (1, 1, \ldots, 1)$ is a probability to jump to any other node and c is a damping factor constant. In order to retrieve personal recommendations for the user using the PageRank algorithm, we have to use its modification, called Personalized PageRank. The only difference is that we always randomly jump to the starting node of interests. So in other words $\vec{(e_1)} = (1, 0, 0, \ldots, 0)$ and $\vec{(e_i)}$ would have 1 in the i-th element if vector and 0 at others. [LM14].

4 Dataset

In this chapter we describe the dataset we have used along with the steps done in order to clean the textual data for the document embedding models.

4.1 Data description

The given dataset consists of the following main entities: job postings, resumes and interactions between them: applies to job post using the resume.

The example of the job posting data can be viewed in the Figure 5. The data consists of the following entities:

- id of the job posting;
- name or title of the vacancy;
- city, where the vacancy is opened;
- textual description of the job post in the free form;
- one of 34 branches selected by job poster;
- offered salary;
- date of posting;
- type of schedule selected by job poster.

As can be seen in the Figure 5 description comes in raw HTML format, which would require additional filtering and cleaning, city, schedule, and branch are mapped to a numerical representation. The meaning of each branch can be seen in Table 1. These branches are mainly used to help with navigation in the site. While they cover all possible job postings into groups, it is also quite clear, that these branches are too general meaning that job postings inside one branch can be completely different, but this is still a beneficial meta information. The schedule type might be one of the following: full-time, part-time, project job, remote job, internship. Salary field is the only one, which can be missing, in this case, its value is imputed with 0.

Another type of data we are dealing with is resume information. It has very similar entities as a job posting and consists of the following:

- id of the resume;
- specialty of the user;

- city, where the job seeker wants to work;
- text of resume;
- one of 34 branches selected by job seeker;
- date of posting.

In addition to this, we have a simple table of interactions between job posts and resumes, which are in the form pairs of vacancy ids and resume ids. An interaction means an application of the user to the job, which reflects the most significant possible interest.

Moreover, we have a filtered list of 1953 most popular search queries used in the website, as well as the list of synonym glossary consisting of 2723 groups. This data was created semi-manually, using the search statistics data.

4.2 Data preprocessing

Talking about the last part of the dataset - user and item interactions, we first decided to remove jobs which don't have interactions with users from the modeling. The removing is done since we assume that these postings are irrelevant, so we have taken away those job posts which have no applies.

Secondly, we remove resumes which were applied to many jobs, since this possibly could be spam, injections, irrelevant resumes. We filter out users with an overall number of applies that exceed the 99th percentile of all users. We consider these users as outliers.

Table 2 shows the size of the dataset after removing outliers. The mean amount of applies for user after cleaning the data is close to 3, and we also have 18 applies per job posting on average.

Much more significant amount of preprocessing should be done while working with text in order to transform it into numerical data. Following techniques has been applied to both job post data and resume data.

First of all, we remove punctuation, HTML tags, extra white-space characters, special characters since they don't give us any useful information and add noise while learning a model. However, in both job descriptions and resumes, there are words, which includes special characters by nature. So, the best decision was to use additional information from popular search queries. In this way we can protect words like: C#, C++, .NET, ASP.NET, Procter & Gamble and others. All this has been done by using regular expressions.

Another thing to do is to lowercase all of the words so that the uppercase letters will not affect the performance of any model by treating same words as different tokens. In some problems, this should also be done carefully, since some words starting with upper and lower case letters can mean different things, but this is problem rarely occurs in job posting texts.

Next things which add a lot of noise in textual similarity systems are numbers. We approached this with replacing all of the numbers with placeholder "num" and we also used the same approach as with special characters - protection of often occurring words. In this way, we have protected words like B2C, J2E, J2EE and similar entities which include numbers. Replacing numbers with a placeholder helped us not to lose some context in the text.

Another import thing in text preprocessing is stemming. Stemming is usually referred to reducing the word to its most basic form. In this task we use an algorithmic approach in order to stem all of the words in the text, it is called Snowball stemmer. [FP01], stemming is reducing the size of vocabulary and eliminates some noise from the text.

Words that occur very frequently and doesn't give any meaningful information called stop-words and are usually removed from texts. The first thing to remove is functional words like "the", "from", "for", "a", "to", "our", "was" etc. It is also really common to remove domain-specific words, which are repeated often and doesn't influence text a lot. We have again used popular search queries list but also we have run TF-IDF and looked at biggest IDF scores manually selecting which words to remove. Therefore we have removed words like: "job", "description", "position", "need", "have", "must", "required", "contact", "requirements", "experienced", "flexible" etc.

Finally, since we are going to use words or unigrams as tokens we can protect some of the bi- and three- grams. Again, the dataset of search queries is really helpful, we can manually select which phrases to protect. In this way, we can use phrases like "machine learning", "data scientist", "sales manager", "category B" and many others as tokens since we assume that splitting them to words might make models harder to learn.

In addition to this since the company provided us with the small internal glossary of synonyms and common spelling errors consisting of 2723 groups, which for instance contains following synonym groups: "telecom, telecommunication, telecommunications ,telecommunicatios, telecomunication, telecomunications, telekom" or "digital marketing, internet marketing, online marketing", we are able to replace all the synonyms to single meaning and eliminate even more noise from texts.

Here is how our example from Figure 5 looks like after preprocessing step: "develop framework captur natur languag processer machine_learning process research math intellig machine_learning comprehend research paper build prototyp algorithm data structur mathemat big_data python deep learn framework". As one can notice it is now much harder to understand text for human, while it is clear that it contains only relevant information without much noise.

As shown in Algorithm 1, we firstly perform `synonymReplace` function replaces the key with the value in the string, it uses the key-value glossary in the format of the dictionary; `stringReplace` function looking for protected words in the document. This function replaces whitespace with "_" signs, numbers with their textual meaning, i.e "2" is replaced with "two" so that later tokenizer treats them as unigram.

Algorithm 1: TextCleaning

Input: List of raw strings of job or resume descriptions, dictionary of synonyms, list of protected words, list of stop-words, list of bad symbols

Result: List of preprocessed tokens for each document

```
1  foreach document string document_ do
2      lowercase string;
3      foreach key, value k, v in dictionary of synonyms do synonymReplace(k, v,
         document);
4      ;
5      foreach protected word s do stringReplace(s, document);
6      ;
7      foreach character c in document do if c in list of bad symbols then
8          replace c with whitespace
9      else
10         pass
11     ;
12     ;
13     regexp: remove extra white spaces;
14     split string to list l on whitespaces;
15     foreach token t in l do snowballStemmer(t, l);
16     ;
```

```
            "Salary": 0,
            "Description": "<p> </p><p>Job Description:</p><p
> </p><p>Develop a framework for capturing knowledge using
natural language processors and machine learning.</p><p> </p
><p>What you need for this position:</p><ul><li>Experience in NLP
and ideas of how to apply learning algorithms to Natural Language
Processing tasks</li><li>Research mindset and math-oriented
intelligence</li><li>Knowledge of machine learning in theory and
practice</li><li>Ability to comprehend research papers and build
prototypes</li><li>Strong algorithms and data structures skills
</li><li>Strong Mathematical skills</li></ul><p> </p><p
>Preferred Skills:</p><ul><li>Familiarity with Big Data.</li><li
>Python</li><li>Experience with Deep Learning frameworks.</li></ul
><p> </p><p> </p>",
            "AddDate": "2016-07-05T17:59:46.670000",
            "CityID": 1,
            "VacancyName": "Machine Learning Engineer",
            "BranchID": 47,
            "ID": 6242943,
            "ScheduleID": 1
```


Figure 5. Job posting data example

Table 1. Possible for selection job posting and resume branches.

BranchId	**Branch Name**
47	IT
48	Telecommunication
49	HR specialists - Business-Trainers
50	Security Services / Armed Forces
51	Logistics - Customs - Warehousing
52	Accounting - Taxes - Financial enterprises
53	Sports - Beauty - Health
54	Hotels - Restaurants - Cafes / HoReCa
55	Health Care - Pharmacy
56	Science - Teaching - Translation
57	Administrative Specialists - Drivers - Couriers
58	Top-management
59	Non - profit organization - Public
60	Consulting - Analytics - Audit
61	Design - Graphics - Photo
62	Commerce
63	Sales - Client Management
64	Banks - Investments - Leasing
65	Insurance
66	Household Staff
67	Show business - Entertainment
68	Media - Publishing
69	Tourism - Travel
70	Marketing - Advertisement - PR
71	Maritime
72	Agriculture - Agro-business - Forestry
73	Construction - Architecture
74	Real Estate
75	Lawyers and notaries
76	Students - Career start - Without Experience
77	Purchases - Supplies
78	Production - Engineers - Technologists
79	Auto business - Auto Service
80	Government agencies - Local self-government

Table 2. Statistic of counts of data in the dataset after removing outliers.

Total job posts count	**Total resume count**	**Total interaction count**
56447	333430	1038323

5 Methodology

In this chapter, we present our methodology which we used to retrieve the recommendations for users based on their resumes from the initial dataset. The overall system chart is presented on Figure 6. It consists of 4 main steps: data cleaning and preprocessing, document embedding algorithm, neighbors retrieval algorithm and graph-based recommendation ranking algorithm.

5.1 Document embeddings modeling

In order to find similar texts we have used following approaches of document embedding models and their modification.

We firstly use the simplest representation - bag of words and its gensim implementation [ŘS10], we also normalized the counts of words and transformed them to vector. As the result, our vocabulary size for job postings was 9856 words and 13877 words for the resume. The difference is, as mentioned earlier is based on bigger document size for resumes, a different language used by job posters and job seekers, as well using synonym glossary and protected words list which was mostly conducted using job posting data.

Then, we have used TF/IDF weighting and its gensim implementation [ŘS10] on the previously retrieved bag of words as our second approach in generating document embeddings.

Much more computationally expensive procedure was training word2vec vectors. We have tried four different word2vec models for job posters and job seekers, depending on datasets used and the training algorithm. Talking about a dataset used, we also took past job posts and CVs (already deleted and closed) in order to receive better word vectors. In this way, we increased the number of job posts from 56447 to 1059552 and CVs from 333430 to 1575518. We also tried using both CBOW and Skip-gram algorithm. In order to train word2vec models, we have used gensim implementation [ŘS10] with default parameters, only changing the training algorithm and number of training documents.

The following parameters were set for training:

- dimensionality of word vectors: 100;
- maximum distance between the current and predicted word within a sentence: 5;
- minimum count: 5, which means that words occurring less than 5 times are ignored;
- learning rate: 0.025 which would linearly drop to 0.0001;
- negative sampling with 5 noise words drawn, also 0.75 exponents used to shape the negative sampling distribution;

- threshold for higher-frequency words, which are randomly down-sampled is 0.0001;
- for CBOW algorithm we use mean instead of the sum of context word vectors;
- 5 epochs over the corpus are made with sorted final vocabulary and 10000 maximum batch size.

Then, in order to retrieve document vectors from word2vec vectors, we firstly use a common approach - weighing all the words with TF-IDF weights and then averaging all the word vectors in a document with respect to TF-IDF vector. This is a pretty naive approach, but it is used very often. Secondly, we use a simple smoothing approach inspired from the paper by Arora, Liang, and Ma (2017). They called their approach "smooth inverse frequency" and they used pretty simple word weight calculation, they just compute the weighted average of the word vectors in the sentence and then remove the projections of the average vectors on their first principal component.[ALM17]. We have used python implementation made by authors of the paper. [4]

Finally, we have used doc2vec model, with both algorithm modification PV-DM and PV-DBOW. For doc2vec model we use all of the available documents including historical ones in order to receive better embeddings for currently posted documents and CVs. We use default gensim implementation [ŘS10] with following parameters:

- dimensionality of document vectors: 100;
- 5 was the maximum distance between the predicted word and context words used for prediction within a document;
- minimum count: 5, which means that words occurring less than 5 times are ignored;
- learning rate: 0.025 which would linearly drop to 0.0001;
- negative sampling with 5 noise words drawn, also 0.75 exponents used to shape the negative sampling distribution;
- threshold for higher-frequency words, which are randomly down-sampled is 0.0001;
- for PV-DM algorithm we use mean instead of sum or concatenation of context word vectors;
- 5 epochs over the corpus.

[4] `https://github.com/PrincetonML/SIF`

5.2 Finding similar documents

In order to find distances between all pairs of job postings and vacancies, we would have to construct a large sparse matrix and do a lot of computations to find all the similar pairs. The complexity of such operations is $O(N^2)$. This is highly inefficient and memory intense. In order to overcome this problem, we have to use approximations and more efficient data structures.

We have chosen Spotify's python implementation of Approximate Nearest Neighbors algorithm called ANNOY[5] . It supports cosine distance similarity by transforming its vectors to Euclidean space by doing simple operation $\sqrt{2(1 - cos(u, v))}$ for some vectors u and v, which is most commonly used when comparing document vectors with each other and fits our task really well. It works fast enough and uses a moderate amount of resources.

It uses the random projection, the basic idea of which is to choose some random hyperplane and use the hyperplane to hash input vectors. Every hyperplane is chosen by sampling points from the subset and taking the hyperplane equidistant from them.

There are just two hyper-parameters to choose for the ANNOY algorithm: number of trees for indexing, which is provided during build time and affects the build time and the index size. A larger value will give more accurate results, but larger indexes. Basically, we can balance between precision and search time depending on the task. Another one, amount of nodes to inspect bound for search: is provided in runtime and affects the search performance. A larger value will give more accurate results but will take a longer time to return.

The implementation allows following operations:

- building an index with a defined number of trees by adding vectors. after this operation no more vectors could be added;
- saving and loading the index;
- getting n closest items for a vector after performing the desired number of nodes inspections;
- getting vectors for an item;
- getting distance between 2 items;

Since we have selected by job posters branches of vacancies and selected by job seekers branches of resumes, we have decided to build separate ANNOY random projection tree. We believe it should increase the performance of document similarity module as well as the precision of ANNOY algorithm.

[5] `https://github.com/spotify/annoy`

This algorithm allowed us to retrieve N similar nodes for every node. For every random projection forest, we use a number of trees such that it would we 100 nodes in every tree. We choose $N = 10$ for the desired number of similar objects with additional distance cutoff of 0.5.

5.3 Retrieving recommendations from graph

After obtaining all similar pairs of resumes and job postings, we construct a graph, consisting of an undirected graph of resumes nodes and items nodes. The connection between resume nodes is based on the similarity retrieved from CVs document vector representations. Similarly, the connection between job postings is retrieved from vacancies document vector representations. In addition to this, it contains user-item edges which detect the fact of applying to particular job posts from the user with a particular resume.

In this way we have an average of 3 edges from user to the item, an average of 20 edges from an item to a user and at the maximum of 10 connections between users and items. Then the goal was to obtain n recommended job postings for a user based on his resume. We have chosen the following approach:

- get a node from a graph corresponding to a user;
- get all of its direct item neighbors from user-item edges;
- get all of its direct user neighbors from user-user edges;
- run a Modified Personalized PageRank algorithm. We modified Personalized PageRank in the way that we defined personalized teleport vector, meaning that during a random walk process we have a random probability to return not only to the initial user node but also to its user-user direct neighbors and user-item direct neighbors. We define elements of teleport vector as follows:

  $\vec{q}$ as $q_i = \begin{cases} 1, & \text{if } v_i \text{ is direct neighbor of the node of interest} \\ 0, & \text{if } v_i \text{is not a direct neighbor of the node of interest} \end{cases}$

  So, the complete equatlion for Personalized PageRank vector is:

  $$\vec{r} = cP^T\vec{r} + (1-c)\frac{1}{n}\vec{q} \tag{9}$$

  We also used the numbers of 0.60 as dumping factor and 100 as a number of iterations of random walk. For the modeling we use Spark and its graphframes[6] extension
- filter results to only item nodes and exclude direct user-item edge neighbors;

[6] https://graphframes.github.io/

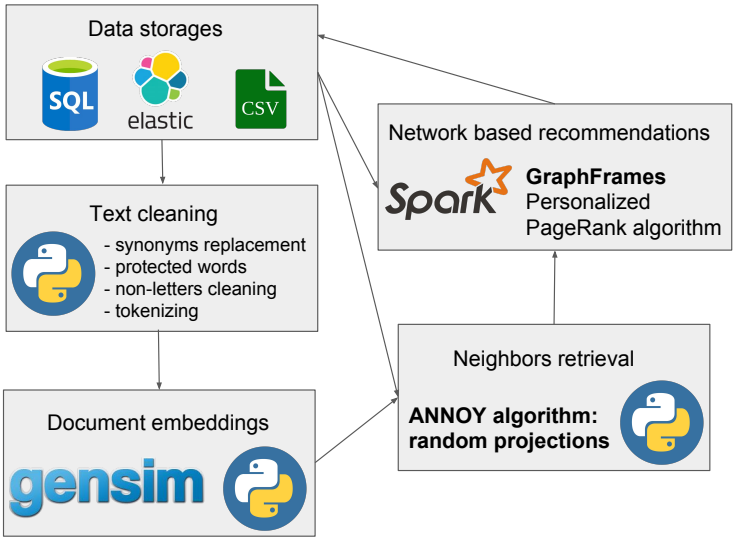

Figure 6. Architecture

6 Experiment and Evaluation

In this chapter, we describe the experimental setup for the online experiment, which includes choosing the control and experiment group. We also discuss offline experiment and methods of splitting data to the train and validation test. Then, we talk about the chosen evaluation metric and describe the results we have achieved.

6.1 Experimental setup

The evaluation of recommender system requires proper splitting of data to train and holdout dataset in order to correctly evaluate the system and prevent over-fitting. Wrong experimental setup would naturally lead to misjudgments and wrong results. In our case, we had to design a proper evaluation system which would reflect an online setup as closely as possible.

6.1.1 Online experiment

The online experiment was done using daily job recommendation mailing. The users have subscribed to the list automatically when doing their first apply and creating the resume on the website. The user also can unsubscribe from the mailing list manually or be unsubscribed due to inactivity, meaning that he has not opened emails

We split the desired users into two equal groups at random based on their IDs. In this way, we have a Control and Experiment group, also called "A" group and "B" group. The whole process is also known as A/B testing. We calculate recommendation for an "A" group with a control algorithm and check if our system works better on the Experiment group.

Another important thing has weekly seasonality in website activity. It means that user behaves differently depending on the day of the week. Considering this fact, we usually need a week timespan to measure all the effects of the new systems, since the activity of users is drastically different on Mondays and Saturdays. In addition to this, the control and experiment group should be fixed at the beginning of the experiment, which means that if the user was in control group in the first day of the experiment, he remains there at the last day as well.

For the business, there is a core measure for evaluation called Conversion Rate, which would decide if the approach was successful for users or not. This approach measures how many clicks from emails to job posts converts to applies for the job. The primary logic behind this approach is that the better the algorithm more applies it generates, because of the better quality of content it recommends for the user.

From the technical side, we only had to change an API end-point so that for users in the control group, mailer system would request recommendations from another source, formed by us.

We also have decided to choose a smaller amount of audience, and to have an experiment for the users with selected branches of their resumes being "IT" and "Sales - Client Management" as shown in Table 1. Even though those are the most popular branches of the website, this still leaves us with the total of approximately 18% of all users.

6.1.2 Offline experiment

In order to make a proper off-line experiment we have to mirror the on-line setup and split all of the datasets used in making recommendations to training and validation sets accordingly. The following steps were taken in order to setup off-line experiment:

- Choose seven consecutive days for validation datasets.
- Make seven validation datasets of interactions between user. Every dataset captures the length in one week from chosen day
- For every validation day form a training dataset with all the posted and actual vacancies and resumes for that day, along with their features and interactions between them.

So, the core idea of the off-line evaluation method is to split the data into training and testing based on time, identically to the real-life setup.

As shown in Figure 7 we try to model the off-line environment as closely as possible, but still it does have some problems and drawbacks. The user who applies for the job from emails more often is our audience of focus in the online experiment. Those users are reacting to the content of mail and making their decision to apply at the moment of viewing the content. In offline setup, users cannot see the recommended content. Thus we instead predict how well we are guessing the same job posts that were sent by our Control system.

In order to overcome this problem, we have to measure the quality of our recommender system not only for those users, who are active users of mailing programs but for all users on the website. That would help us to see how our system guesses natural or organic applies rather than those, which were forced by the system to some extent.

6.2 Evaluation metrics

It has always been a challenging task to find a proper metric, which would evaluate the performance of the recommender system. In case of explicit feedback datasets, it is possible to measure an accuracy metrics: such as RMSE (root mean squared error) or MAE (mean absolute-error) since the holdout dataset has known ratings and we can compare recommender systems predictions similarly to any classification task. However,

in the case of implicit feedback, measuring accuracy is also possible considering rating being 1 in case of an interaction between user and item and 0 in case of absence of the interaction. [Agg16]

However, considering the domain of job recommender system, this is not a proper metric to use. We are limited to show only several numbers of recommended items since the user would probably lose attention and would not scroll to all of the possible recommended items. In other words, the user can interact with the limited subset of items, and our goal, in this case, would be to match this subset as closely as possible.

In order to evaluate recommender system, we decided to use precision and recall modified metrics. Precision, in this case, would be calculated using a number of recommendations specified in the process of designing the system, which would match those, the user has interacted with. The recall, in this case, would be calculated as the number of items which users had interacted with which were present in top k recommended items.

Formally, the evaluation metrics are given bellow:

$$Precision_k = \frac{\{relevant\_items\}_n \cap \{recommended\_items\}_k}{k} \tag{10}$$

$$Recall_k = \frac{\{relevant\_items\}_n \cap \{recommended\_items\}_k}{\{relevant\_items\}_n} \tag{11}$$

$$F1_k = \frac{2 \cdot Recall_k \cdot Precision_k}{Precision_k + Recall_k} \tag{12}$$

The value of $Precision_k$ is not always monotonic for the top k because of both the numerator and denominator may change with k differently. There is a natural trade-off between precision and recall metrics, and this trade-off is also not necessarily monotonic. It means that an increase in recall does not always lead to a reduction in precision. In order to present a single metric, which would describe the trade-off and overall performance of the system by its own, we use the F1-measure, which is the harmonic mean between the precision and the recall. [Agg16]

6.3 Results

In this section we present the results of our evaluation performed using different methods.

6.3.1 Offline results

First of all, we wanted to check which document embedding algorithm would perform better in our dataset. We believe that choosing appropriate document vector representation would lead to more accurate neighbor graph and better recommendations for users based on their CV and historical data. In order to do this we fix other parameters as follows:

- amount of recommendation made is equal to 10.
- amount of neighbors from document embedding similarity retrieval module is 10
- personalized PageRank algorithm dumping factor is equal to 0.50, a number of iterations is equal to 100.
- the data for word2vec and doc2vec training included only currently published resumes and vacancies

The variables in the first experiment were methods of document embeddings. We have checked the following setup:

- BOW (bag of words);
- TF-IDF;
- Skip-gram word2vec averaging;
- Skip-gram word2vec with TF-IDF weights;
- Skip-gram word2vec with smooth inverse frequency weights;
- PV-DM doc2vec;
- PV-DBOW doc2vec;

We have expected that more advanced algorithms like doc2vec and word2vec with smooth inverse frequency weights would be superior to more straightforward bag of words and TF-IDF.

As can be seen at Figure 8 in best case scenario we have about one out of ten job recommendations being correct. Because there is an average of only three applies per user during its lifetime which shows that those results are decent. We observe that word2vec skip-gram model with and smooth inverse frequency algorithm as the weighting scheme and PV-DM being superior to other document embedding schemes.

As one can see at Figure 9 in best case scenario we achieve recall of 0.3, which means we are guessing correctly one out of three applies for user on average. The recall is more important metric in our case since the activity of users is usually low and we are choosing k larger than average amount of applies from users to job posts - we expect to maximize recall, which means to have all of the vacancies of interest in our recommended job posts.

Another thing we wanted to measure was the impact of larger training vocabulary on the performance of word2vec and doc2vec models, as they give the best quality of recommendations. We expected a significant increase in metrics.

In this experiment the variables were:

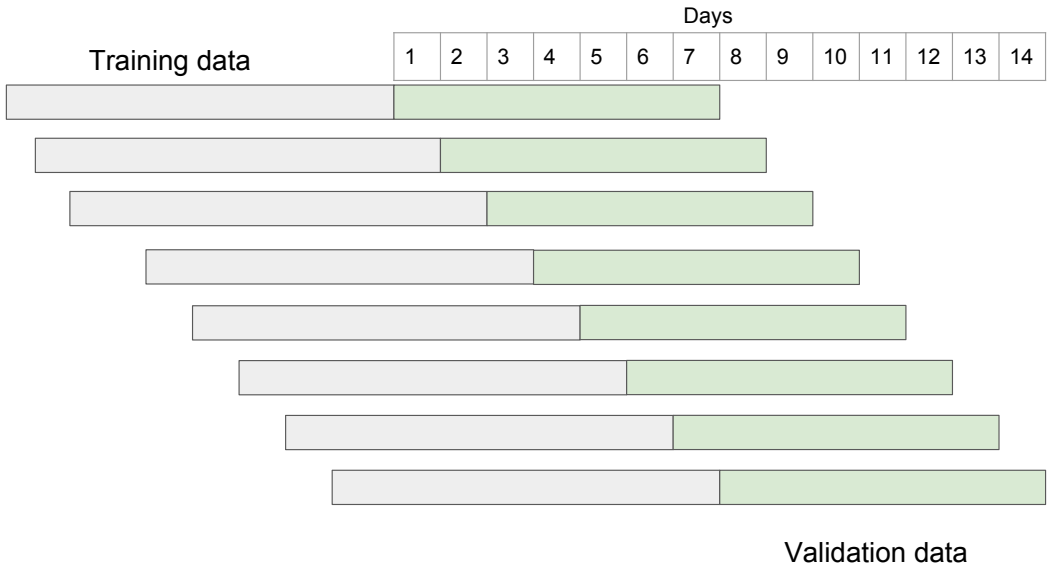

Figure 7. Offline evaluation

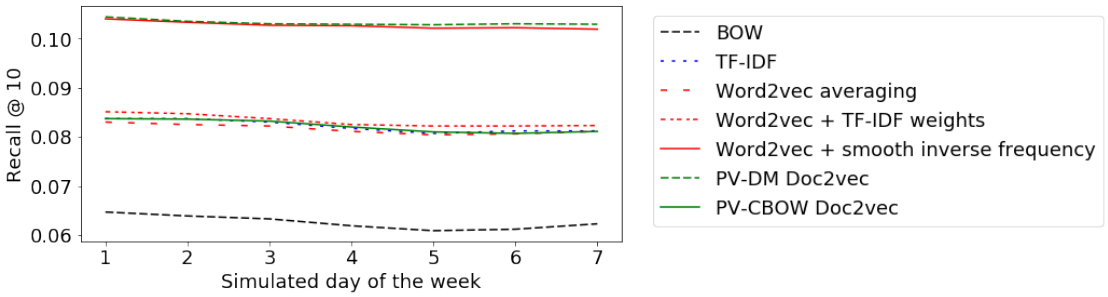

Figure 8. Precision comparison for different document embeddings

- Skip-gram word2vec with smooth inverse frequency weights trained on the actual dataset;
- Skip-gram word2vec with smooth inverse frequency weights trained on the actual and historical dataset;
- PV-DM doc2vec trained on the actual dataset;
- PV-DM doc2vec trained on the actual and historical dataset;

As we can see at Figure 10 and Figure 11 word2vec scales much better with more significant dataset since it can learn better word representation, doc2vec on the other hand scale worse depending on the size of the corpus. The result of the experiment gives us the recall being nearly fifty percent, which means that we match half of the user interests by our recommendations.

Next, we tried to inspect how the Personalized PageRank algorithm parameters depend on the chosen metrics of precision and recall. We try to vary the number of iterations from 50 to 150 and dumping factor from 0.6 to 1.0. In this experiment, we fix the document embedding algorithm to word2vec trained on historical with actual data with smooth inverse frequency and the damping factor is 0.60. We also take an average Recall and Precision for ten recommendations of seven experiments.

Figure 12 shows that the result stabilize starting from 75 iterations. Meaning that there is no need to make an algorithm to do more iterations. It is clear from Figure 13 that with the growth of the dumping factor the result is getting worse. It means that Random Walk takes more steps further to the tree giving more non-relevant recommendations as a result.

6.3.2 Online results

In order to measure our results in real-life setup, we have stopped on the best performing configuration. We have chosen word2vec with smooth inverse frequency algorithm for document embedding module and parameters of Personalized PageRank algorithm as 0.5 and 100 for dumping factor and number of iterations respectively. We did not reduce the number of iterations since the setup did not require faster calculations.

The existing email program, as mentioned previously require the person being not unprescribed from email, which is in total 150101 users. We also decided to check an algorithm only for several branches which left us with 27048 emails daily on average.

The algorithm for the control group (existing architecture) was a content-based algorithm, which used ElasticSearch[7] as a tool for retrieving similar job-posts. The algorithm can be described as the following:

[7] `https://www.elastic.co/`

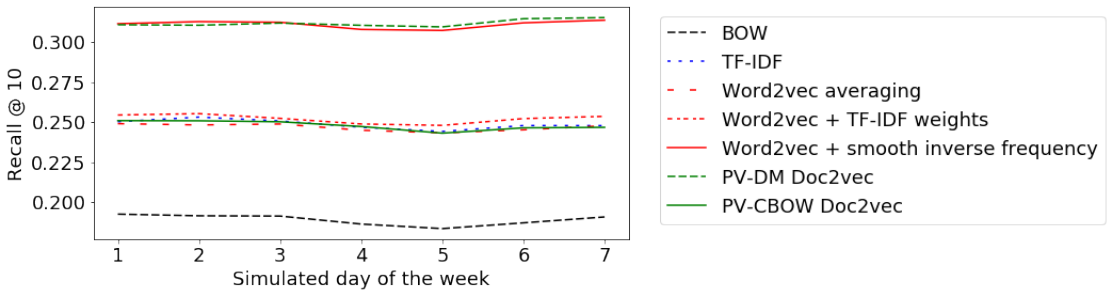

Figure 9. Recall comparison for different document embeddings

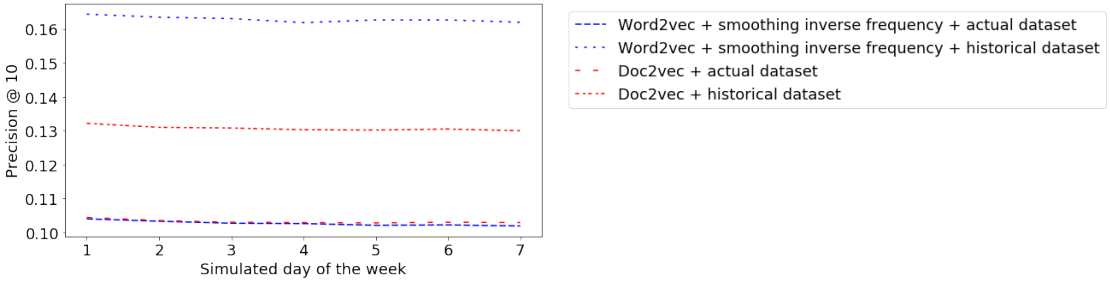

Figure 10. Precision comparison for different document embeddings using actual and historical datasets

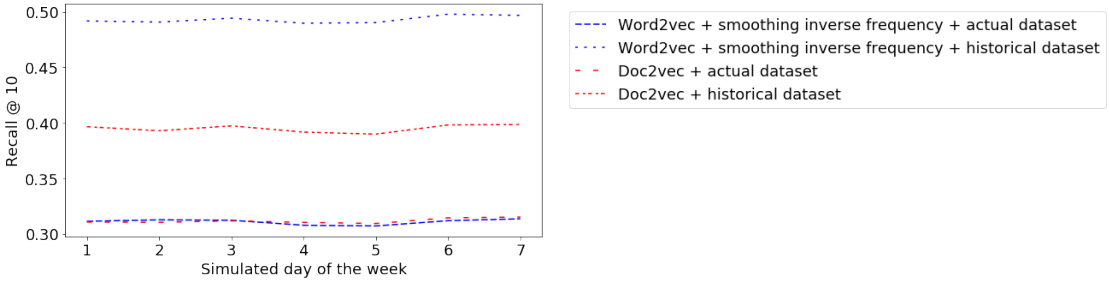

Figure 11. Recall comparison for different document embeddings using actual and historical datasets

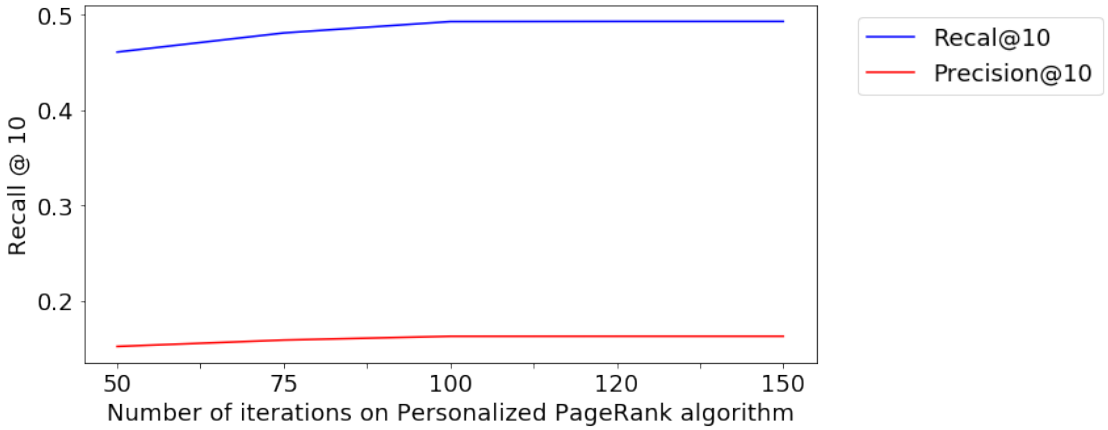

Figure 12. Number of iteration comparison for Personalized PageRank algorithm

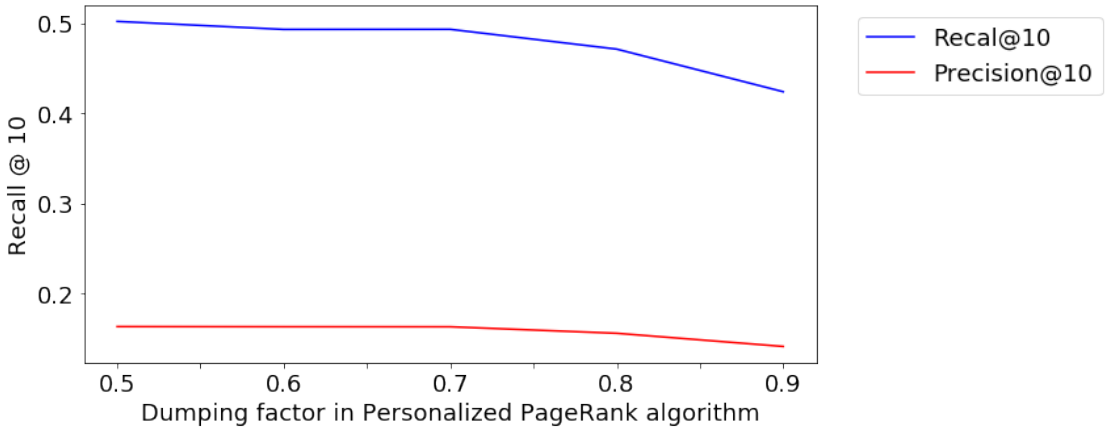

Figure 13. Dumping factor effect for Personalized PageRank algorithm

- For a selected user retrieve job posts he had applied to.
- Run "More Like This" query in order to retrieve similar job posts.
- Choose top 10 job relevant job posts based on the score given by ElasticSearch engine.

The existing system uses ElasticSearch and runs the bag of words with BM25 weighting on the selected features of the document in order to retrieve similar ones. This approach ignores data from CV and similarity between users based on their interactions with job post.

Table 3. Results of off-line experiment

	Emails sent	Views	Applies	Conversion
Control Group	94670	10876	2066	**0.190**
Experiment Group	94666	**11245**	**2082**	0.185

We have expected that our system would be superior to the existing one. As one can observe from Table 3 recommendation made by our algorithm generated more applies and more views. However, the conversion rate dropped. This does not necessarily mean worse overall performance since the number of users and data was limited. In addition to this, users which are usually using emails might behave differently from other users. Also, in the process of evaluation of the off-line experiment, we only counted direct applies and views from the email links, which contained an Urchin Tracking Module (UTM) parameters with the identification of experiment. It means, that if the user returned naturally to the job post later and applied for the job, we did not count it as the application from email.

7 Conclusion

To sum up, in this thesis we used the real dataset of job searching platform and tried to improve the current recommendation system and achieve better results in production. We have achieved an average recall of 50% for the ten recommended job posts and gained more job applies during an online test.

We have tried a hybrid network-based recommendation system, which also included a heavy focus on finding similar resumes and job posts using better document vector representation techniques. We have used a Personalized PageRank algorithm in order to rank job posts for the user based on his resume, interactions with job postings and similarity to other users.

We examine different document embeddings approaches such as bag-of-words, TF-IDF, word2vec, and doc2vec. For word2vec we tried various averaging techniques in order to retrieve a document embedding and found out that quite simple smooth inverse frequency weighting is superior for our corpus.

We conclude several iterative off-line experiments, trying to find the best document representation method as well as Personalized PageRank hyper-parameters. Using the better performing setup in off-line setup, we conducted an A/B test to check the performance of our system in real-time.

It is important to notice that the system we created could be possibly used to recommend resumes for the job post in order to make a better experience for the employers as well. we

For the future work, we can try to tune the parameters of the word2vec model to obtain better performance of the similarity retrieval module. In addition to this, we can obtain a better glossary of synonyms. We can also experiment with the network system and improve the ranking.

Another approach, different from graph-based could be tried to build a hybrid recommendation system. We can use recent advances in deep learning in order to build a complex system, which can outperform the current setup.

Talking about using the system in production, we are aiming to optimize the performance of algorithms and build the stable solution which would be possible to use in real-time.

References

[Agg16] Charu C. Aggarwal. *Recommender Systems: The Textbook*. Springer Publishing Company, Incorporated, 1st edition, 2016.

[ALM17] Sanjeev Arora, Yingyu Liang, and Tengyu Ma. A simple but tough-to-beat baseline for sentence embeddings. 2017.

[Alo12] Shaha Alotaibi. A survey of job recommender systems. 7, 07 2012.

[BA03] Ron Bekkerman and James Allan. Using bigrams in text categorization. 2003.

[BDVJ03] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, March 2003.

[BGM15] Dheeraj Bokde, Sheetal Girase, and Debajyoti Mukhopadhyay. Matrix factorization model in collaborative filtering algorithms: A survey. *Procedia Computer Science*, 49:136 – 146, 2015. Proceedings of 4th International Conference on Advances in Computing, Communication and Control (ICAC3'15).

[BHK98] John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI'98, pages 43–52, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[BMS06] Kevin Bartz, Vijay Murthi, and Shaji Sebastian. Logistic regression and collaborative filtering for sponsored search term recommendation. 01 2006.

[BP98] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst.*, 30(1-7):107–117, April 1998.

[Car00] Maria Fernanda Caropreso. A learner-independent evaluation of the usefulness of statistical phrases for automated text categorization. 2000.

[CZ17] Yu Chen and Mohammed J. Zaki. Kate: K-competitive autoencoder for text. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pages 85–94, New York, NY, USA, 2017. ACM.

[ERB18] Ahmed Elsafty, Martin Riedl, and Chris Biemann. Document-based recommender system for job postings using dense representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 3 (Industry Papers)*, pages 216–224. Association for Computational Linguistics, 2018.

[FP01] M F. Porter. Snowball: A language for stemming algorithms. 1, 01 2001.

[FPRS07] F. Fouss, A. Pirotte, J. Renders, and M. Saerens. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 19(3):355–369, March 2007.

[FR15] Bracha Shapira (eds.) Francesco Ricci, Lior Rokach. *Recommender Systems Handbook*. Springer US, 2 edition, 2015.

[GC06] Derek Greene and Pádraig Cunningham. Practical solutions to the problem of diagonal dominance in kernel document clustering. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 377–384, New York, NY, USA, 2006. ACM.

[GML+10] Amir Gershman, Amnon Meisels, Karl-Heinz Lüke, Lior Rokach, Alon Schclar, and Arnon Sturm. A decision tree based recommender system., 01 2010.

[GP07] Marco Gori and Augusto Pucci. Itemrank: A random-walk based scoring algorithm for recommender engines. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, pages 2766–2771, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[Har54] Zellig S. Harris. Distributional structure. <*i*>*WORD*</*i*>, 10(2-3):146–162, 1954.

[HLZ+17] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, pages 173–182, Republic and Canton of Geneva, Switzerland, 2017. International World Wide Web Conferences Steering Committee.

[Hof01] Thomas Hofmann. Unsupervised learning by probabilistic latent semantic analysis. *Machine Learning*, 42(1):177–196, Jan 2001.

[KBV09] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, August 2009.

[KW13] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2013.

[LEHG13] Yao Lu, Sandy El Helou, and Denis Gillet. A recommender system for job seeking and recruiting website. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13 Companion, pages 963–966, New York, NY, USA, 2013. ACM.

[LM14] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages II–1188–II–1196. JMLR.org, 2014.

[LSDJ06] Michael S. Lew, Nicu Sebe, Chabane Djeraba, and Ramesh Jain. Content-based multimedia information retrieval: State of the art and challenges. *ACM Trans. Multimedia Comput. Commun. Appl.*, 2(1):1–19, February 2006.

[LSL12] Joonseok Lee, Mingxuan Sun, and Guy Lebanon. A comparative study of collaborative filtering algorithms. 05 2012.

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

[MF13] Alireza Makhzani and Brendan Frey. k-sparse autoencoders. 12 2013.

[MKL07] Hao Ma, Irwin King, and Michael R. Lyu. Effective missing data prediction for collaborative filtering. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, pages 39–46, New York, NY, USA, 2007. ACM.

[MP00] Koji Miyahara and Michael J. Pazzani. Collaborative filtering with the simple bayesian classifier. In *Proceedings of the 6th Pacific Rim International Conference on Artificial Intelligence*, PRICAI'00, pages 679–689, Berlin, Heidelberg, 2000. Springer-Verlag.

[MSC+13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pages 3111–3119, USA, 2013. Curran Associates Inc.

[PB07] Michael J. Pazzani and Daniel Billsus. The adaptive web. chapter Content-based Recommendation Systems, pages 325–341. Springer-Verlag, Berlin, Heidelberg, 2007.

[RIS+94] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: An open architecture for collaborative filtering of netnews. 04 1994.

[Rob04] Stephen E. Robertson. Understanding inverse document frequency: on theoretical arguments for idf. *Journal of Documentation*, 60:503–520, 2004.

[RRS15] Francesco Ricci, Lior Rokach, and Bracha Shapira. *Recommender Systems Handbook*. Springer Publishing Company, Incorporated, 2nd edition, 2015.

[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`.

[SB88] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.*, 24(5):513–523, August 1988.

[SCS16] Thomas Schmitt, Phillipe Caillou, and Michele Sebag. Matching jobs and resumes: a deep collaborative filtering task. In Christoph Benzm\"uller, Geoff Sutcliffe, and Raul Rojas, editors, *GCAI 2016. 2nd Global Conference on Artificial Intelligence*, volume 41 of *EPiC Series in Computing*, pages 124–137. EasyChair, 2016.

[STW+] Deerwester Scott, Dumais Susan T., Furnas George W., Landauer Thomas K., and Harshman Richard. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407.

[SWNF12] Zheng Siting, Hong Wenxing, Zhang Ning, and Yang Fan. Job recommender systems: A survey, 07 2012.

[VLL+10] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, December 2010.

[WHBD14] Lior Wolf, Yair Hanani, Kfir Bar, and Nachum Dershowitz. Joint word2vec networks for bilingual semantic representations. *Int. J. Comput. Linguistics Appl.*, 5:27–42, 2014.

[WKR+17] Jinzhong Wang, Xiangjie Kong, Azizur Rahim, Feng Xia, Amr Tolba, and Zafer Al-Makhadmeh. Is2fun: Identification of subway station functions using massive urban data. PP:1–1, 10 2017.

[WM12] Sida Wang and Christopher D. Manning. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers - Volume 2*, ACL '12, pages 90–94, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[ZdLCG+14] R. Zhang, Q. d. Liu, Chun-Gui, J. X. Wei, and Huiyi-Ma. Collaborative filtering for recommender systems. In *2014 Second International Conference on Advanced Cloud and Big Data*, pages 301–308, Nov 2014.

[ZZAP13] Zhu Zhang, Daniel Zeng, Ahmed Abbasi, and Jing Peng. A random walk model for item recommendation in folksonomies. *CoRR*, abs/1310.7957, 2013.

Appendix

I. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Maksym Sukhorukov**,

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,

   of my thesis

   A network science and document similarity based hybrid job recommendation system

   supervised by Rajesh Sharma

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

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

Tartu, 09.08.2018