ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS
DEPARTMENT OF INFORMATICS
MSc IN DATA SCIENCE

"Job Recommendation System using Deep Learning"

Capstone Project Report
Athanasia Farmaki

Academic Supervisor: Dr. Ion Androutsopoulos
Company Supervisor: Jason Andriopoulos

Company: Workable

Athens, 2018

Table of Contents

List of Figures

Acknowledgements

I would like to thank my supervisor Ion Androutsopoulos for his continuous guidance and support. I would also like to thank the members of AUEB's Natural Language Processing group and especially John Koutsikakis and Manolis Kyriakakis for their aid and guidance. Finally, a great "thank you" goes to my friends and of course to my family for their support and patience.

Abstract

This is the report of a capstone project carried out for the MSc in Data Science at Athens University of Economics and Business. The objective of this project was to design a recommendation engine which, given information about a candidate's profile, would be able to recommend relevant jobs. The task can be divided into two parts, an Information Retrieval (IR) part and a Machine Learning part. For the information retrieval part, Elastic Search engine was used in order to retrieve top 100 relevant jobs for a candidate. Then a deep neural network model called Deep Relevance Matching Model (DRMM) proposed by Guo et al was applied to rerank the results and retrieve the top 10 jobs for the candidate. The data used are given by the software recruitment company Workable and the evaluation of the results is done both automatically using typical IR measures and manually by examining the results one by one.

Περίληψη

Η παρούσα διπλωματική εργασία πραγματοποιήθηκε στα πλαίσια του μεταπτυχιακού στην Επιστήμη των Δεδομένων στο Οικονομικό Πανεπιστήμιο Αθηνών. Το αντικείμενο της διπλωματικής είναι ο σχεδιασμός ενός συστήματος προσωποποιημένης πρότασης αγγελιών εργασίας σε χρήστες της πλατφόρμας εύρεσης εργασίας της εταιρείας Workable. Η υλοποίηση του συστήματος αυτού αποτελείται από δύο μέρη. Το πρώτο αφορά στην εξαγωγή εγγράφων με τη χρήση συστήματος Ανάκτησης Πληροφοριών και το δεύτερο αφορά στην ανακατάταξη των εγγράφων αυτών με τη χρήση μεθοδων Μηχανικής Μάθησης. Το σύστημα ανάκτησης πληροφοριών που χρησιμοποιήθηκε ήταν το Elasticsearch ενώ ως σύστημα μηχανικής μάθησης χρησιμοποιήθηκε το μοντέλο Deep Relevance Matching Model το οποίο προτάθηκε από τους Guo et al. Τα δεδομένα που χρησιμοποιούνται για την δημιουργία του συστήματος ανήκουν στη εταιρεία Workable και η αξιολόγηση των μοντέλου θα γίνει χρησιμοποιώντας κλασικές μεθόδους αξιολόγησης μοντέλων εξόρυξης δεδομένων καθώς και με την παρατήρηση των αποτελεσμάτων από τον προγραμματιστή.

1 Introduction

This thesis examines the use of a Deep Neural Network in order to efficiently tackle the job proposals project. Job Proposals is a project that aims at enhancing user's experience with Workable's Job Search Platform. Workable is a global recruitment software company that provides tools needed to manage multiple hiring pipelines. In this project the objective is to design a recommendation engine which, given information about a candidate's interests, will recommend relevant jobs. So as shown below the system should take as input information about a candidate's career interests and goals and output 10 jobs that should be recommended to her/him.

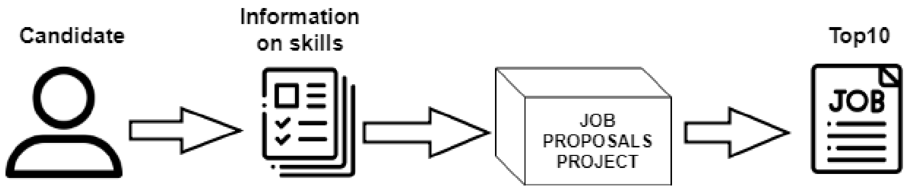
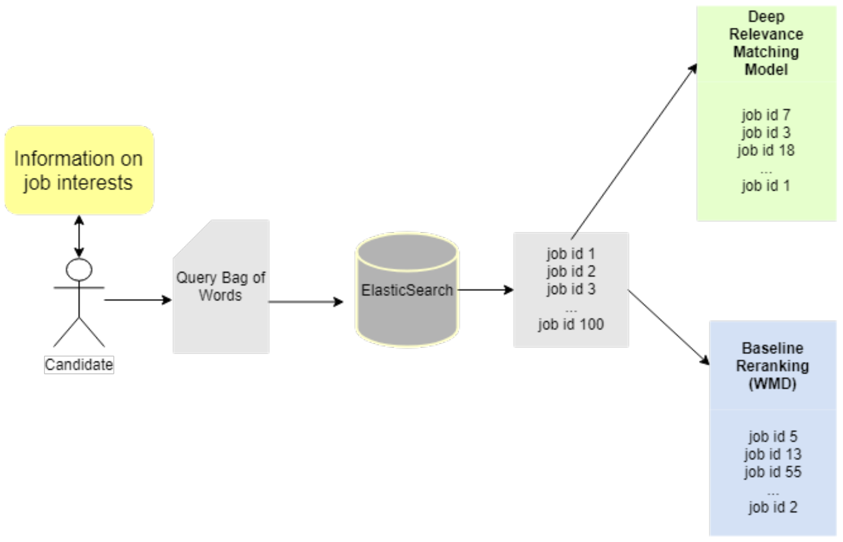


Figure 1: System description

1.1 Thesis Scope

Jobs recommendation system can be seen as a typical Information Retrieval problem whose objective is to be able to rank the most relevant documents given a query. The scope of the thesis is to efficiently solve this problem by using Deep Learning methods. A major issue in the implementation of Deep Learning is the fact that Neural Networks are computationally expensive. Therefore, in order to approach this problem using machine learning, a subset of possibly relevant jobs needs to be retrieved first. For this purpose Elastic Search Engine is used and a subset of 100 relevant jobs is retrieved using a simple relevance metric called BM25 (Robertson and Zaragoza, 2009) (section 3). Then two rerankings of this subset will be calculated. A simple one using a metric called Word Mover's Distance (Kusner et al., 2015) (section 3) that will be used as baseline and a less trivial one using Deep Relevance Matching Model (DRMM) (Guo et al., 2017) which has demonstrated great results in similar problems. So the project can be divided into two parts. The first part is the Information Retrieval section and the second part is the implementation of a Deep Learning technique.



In the next parts of this assignment the data will be described and the methods used will be explained. In chapter 4 the Information Retrieval part will be presented and in chapter 5 the Deep Learning model will be explained. In chapter 6 the data preparation for the models will be analyzed and in chapters 7 and 8 we will present the evaluation metrics and the results accordingly.

2 Data Description

Our dataset consists of two main entities, job advertisements (we will refer to them as jobs) and user information (we will refer to them as candidates since users expressing interest for a job advertisement are potential candidates). The available information for jobs is semi-structured text describing a job opening. The available information for users/candidates is the jobs they have expressed interest for. At this point we need to mention that the goal of the Capstone project is not to recommend jobs based on the candidate's profile but based on his interests. Another important fact (that has impact on some critical decisions we will explain in the following sections) is that for the vast majority of available jobs the level of interest of the candidate is unknown to us. For instance we know that a candidate is interested in a certain job, but we do not know whether he/she would be less, equally or more interested in another, random job from the database.

Jobs Data:

The **jobs' dataset** has 466101 jobs and consists of the following fields:

- Job id
- Title
- Function: The field of work
- Requirement Summary
- Description: General description of the position
- Keywords: words extracted by the system automatically based on part-of-speech(POS) tagging

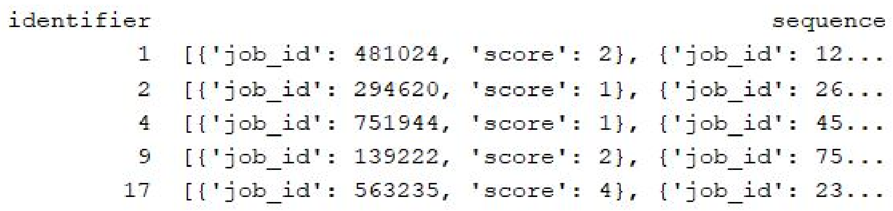
  Below there is an example of what keywords field contain:

```
['python', 'developer', 'ui', 'xml', 'datasets', 'mongodb', 'mathematica', 'flask', 'd3', 'react']
```

In order to bring the dataset into a convenient format a basic cleaning is applied where capital letters turn to lowercase and numbers and punctuation are excluded.

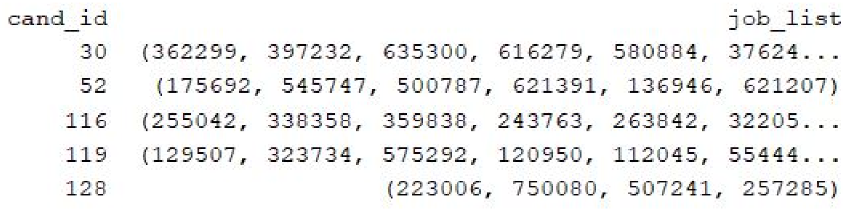
Candidate data:

The information given on a candidate is jobs for which he/she has expressed interest for. More specifically, we have a chronologically sorted list of the jobs a user has taken interest in and a score for each one of those jobs. Scores represent the level of interest that a user has expressed for a job advertisement and are integer values usually between 1 - 6 with a few outliers (values between 7 - 10). The higher the score, the more interested the user is. We consider scores with value greater than 4 to be indicators of high interest. So in order to sum up the information about the candidate, we will have to construct a bag of words using this information. The number of candidates we have at our disposal is 7388266 and the format of the dataset is the following:



```
identifier                                                  sequence
         1  [{'job_id': 481024, 'score': 2}, {'job_id': 12...
         2  [{'job_id': 294620, 'score': 1}, {'job_id': 26...
         4  [{'job_id': 751944, 'score': 1}, {'job_id': 45...
         9  [{'job_id': 139222, 'score': 2}, {'job_id': 75...
        17  [{'job_id': 563235, 'score': 4}, {'job_id': 23...
```

In order to reduce data noise we keep only the candidates for which we have at least 3 jobs in the list and for which the highest scored job has at least a score of 4. Furthermore we observed the existence of candidates that have expressed interest in jobs of different nature, which can be confusing for the training of the deep learning model. Intuitively, relevant jobs contain similar skills in the announcements and belong to similar fields of study. Practically, in order to discard the irrelevant jobs from the list we apply **word mover's distance** (Kusner et al., 2015)(section 3) to check whether all job ids within a candidate have relevant content between them. We consider as jobs with relevant content the ones with average Word Mover's Similarity [1] greater than 0.45 from the rest of the jobs in the list. Also for the calculation of the distance in this phase we use only the words of fields of the job document that are considered as the most informative such as title, function and keywords. When we track a job irrelevant to the rest, we discard that job. After this phase of data processing the format of the data is the following and the number of candidates in our possession is 600000:



```
cand_id                                              job_list
     30  (362299, 397232, 635300, 616279, 580884, 37624...
     52   (175692, 545747, 500787, 621391, 136946, 621207)
    116  (255042, 338358, 359838, 243763, 263842, 32205...
    119  (129507, 323734, 575292, 120950, 112045, 55444...
    128                    (223006, 750080, 507241, 257285)
```

[1]Word Mover's Similarity is in fact the negative of word movers distance normalized to score between [0,1]

3 Methods

In this section all the methods used during the Information Retrieval and the Deep Learning part are analyzed.

3.1 Inverse Document Frequency:

Idf is calculated per word and it is used within the DRMM model on the term gating layer explained in section 5. It is defined as the logarithm of the Number of documents in the corpus divided by the number of documents where the examining word appeared (document frequency)

$$idf_t = log(N/df_t)$$

3.2 BM25 Scoring Function

As mentioned in the introduction, BM25 is the relevance function used in Elasticsearch engine that ranks a set of documents based on the query terms appearing in each document. It is not a single function, but actually a whole family of scoring functions, with slightly different components and parameters. Here we present BM25 version that is used in Elasticsearch engine. In brief, in order to produce a relevance score between a query and a document, BM25 takes into account the number of times a query term appears in the document, how rare that query term is in the corpus and the length of the document compared to the average length of a document in the corpus. In mathematical terms:

$$BM25(q, D) = \sum_{k=1}^{n} IDF_{q_i} \frac{f(q_i,D)*(k1+1)}{f(q_i,D)+k1*(1-b+b*\frac{docLen}{avgLen})}$$

where:

IDF: Inverse Document Frequency of query term i.

f(q_i,D): The number of time term q_i appears in document D.

$\frac{docLen}{avgLen}$: this represents the length of a document relative to the average document length.

b: it is a parameter that we can set according to how much we want the length of the document to be taken into consideration for the calculation of relevance score. If set to 0 we don't take it into consideration at all. In Elasticsearch the default option is 0.75.

k1: it is a variable which limits how much a single query term can affect the score of a given document.

3.3 Word Mover's Distance

Word Mover's Distance (WMD) is the metric used both during the preprocessing phase and as a baseline reranking method. It is a measure used to find the distance between 2 documents and is able to match synonyms between the documents by using word embeddings instead of the actual words. It is defined as the minimal cumulative distance that the words of the first document need to travel to reach the words of the second document. It is advised to exclude stop words from both documents before applying the measure. In this task we use WMD during the preprocessing to discard the jobs within a candidate that are not relevant to the rest of the jobs where he/she applied in the past. We also use WMD as a baseline re-ranking method. It is important to note that on the implementation in python when using wmd on the preprocessing of the data, instead of using gensim's wmdistance which is Word Mover's Distance we use gensim's library wmdsimilarity. Wmd similarity is the negative of word movers distance normalized to score between [0,1].

3.4 Cosine Similarity

Cosine Similarity is used within the DRMM model. It is a measure that calculates the cosine of the angle between two vectors. This metric can be seen as a measure of similarity between documents on a normalized space. It is important to note that even if one word embedding was pointing to a point far from the other, they could still be similar to each other if they have a small angle between them.

4 Information Retrieval

As mentioned this project consists of two parts, an Information Retrieval part and a Machine Learning part. As an academic field of study Information retrieval (IR) is defined as a task of finding documents of a structured, semi-structured or unstructured nature that is relevant to some information given, from within large collection of documents.

The term "unstructured data" refers to data which do not have a clear structure. The most common unstructured data is text. Structured data can be the ones derived from a relational database and as semi-structured data we define the documents with some fields as a structure. In this project we will be dealing with semi - structured data.

The most standard information retrieval task is called ad-hoc retrieval in which given a corpus (a collection of documents) we wish to retrieve the most relevant documents according to some relevance metric, by making a query. The ad-hoc term refers to the fact that the number of ways to construct a query is huge and it is up to the user to decide in which way to construct the query. In this project the IR task mentioned is an ad-hoc retrieval task for which we will use a very fast text search engine called Elasticsearch engine. But first it is important to understand how search engines work and why they can be so fast.

In order to perform information retrieval an initial thought would be to grep (linearly scan) all the words of all the documents in a database and find the wanted ones according to some criterion. As the collection of documents that the IR systems normally search in is very large that practice would be totally inefficient. The way to avoid linear scanning of the documents for each query is to index the documents in advance. The standard indexing method in information retrieval nowadays is called inverted index. The idea of the inverted index is to create a dictionary of terms where for each term, we have a list that records in which documents each term occurs. Each term in the list is called a posting. The dictionary also records some statistics, such as the number of documents containing each term and other metrics that will be used during the computation of the relevance metric used in the corresponding search engine. In this way the retrieval becomes surprisingly fast and the only defect is the increased time to index the data and the storage of that dictionary on the memory during processing. It is important to mention that during indexing the data are tokenized and preprocessed in order to produce a list of normalized tokens.

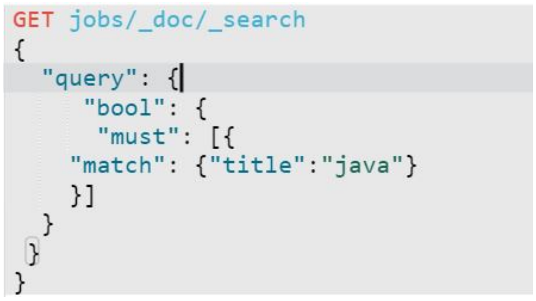
Elasticserarch uses as relevance metric the BM25 score. Previously used metrics for the IR task were boolean operators, term frequency(TF) and term frequency–inverse document frequency (TF-IDF) metrics.

4.1 Elastic Search Engine

As mentioned, in order to apply Deep Learning, we first need to retrieve the top 100 most relevant jobs given a query and for this purpose Elasticsearch along with Logstash will be used. Elasticsearch is an open source search engine and Logstash is an open source data collection engine.

ElasticSearch is a highly scalable open-source text search engine that allows the storage and search of big volumes of data in near real time. In order to use Elasticsearch we first have to index the data as described previously. The term index in Elasticsearch instructions is a collection of documents with similar characteristics, in our case job announcements with specified fields such as title, function, description etc. An index can be created automatically while inserting the data through Logstash or manually by the user who describes the format of the data in the database. Logstash is used to ingest data into Elasticsearch either by using predefined indices or by creating an index while inserting the data.

After having indexed and inserted the data of the database we need to somehow access them. The way this is done is by creating queries in accordance to the ad-hoc retrieval task described. Elasticsearch provides a full Query Domain Specific Language(DSL) based on JSON format to define queries. The following is an example of a query where we retrieve the jobs where the word "java" exists in the field "title" of the index called jobs:



```
GET jobs/_doc/_search
{
  "query": {
      "bool": {
       "must": [{
     "match": {"title":"java"}
     }]
  }
 }
}
```

5 Deep Learning

For the second part of this project, a Deep Neural Network called Deep Relevance Matching Model will be used.

5.1 Deep Relevance Matching Model (DRMM)

DRMM is a deep neural network with which Elasticsearch results will be reranked to achieve a recommendation system with more relevant results. Neural networks used for the task of document re-ranking can be divided into two categories, representation focused models and interaction focused models. A **representation focused** model uses a neural network to build representation of the query and the document separately and then uses a simple similarity function to calculate relevance. **Interaction focused** models create some representation of the relation of the query and the document and then uses a neural network to calculate a relevance score.
If we define as g the function that creates the representation of the query and the document and F the function that produces the relevance score then the representation focused model has a simple F and a complex g while the interaction based model has a simple g to combine the documents and a complex F :

$$\text{relevance score} = F(g(q), g(d))$$

DRMM is an interaction-focused model for ad-hoc retrieval. It takes as input a query q and a documents d and outputs a score indicating the relevance between them.

$q = w_1, w_2, ..., w_m$ where w_i is a pre-trained word embedding for the word i of the query
$d = w_1, w_2, ..., w_m$ where w_i is the pre-trained word embedding for the word i of the document.

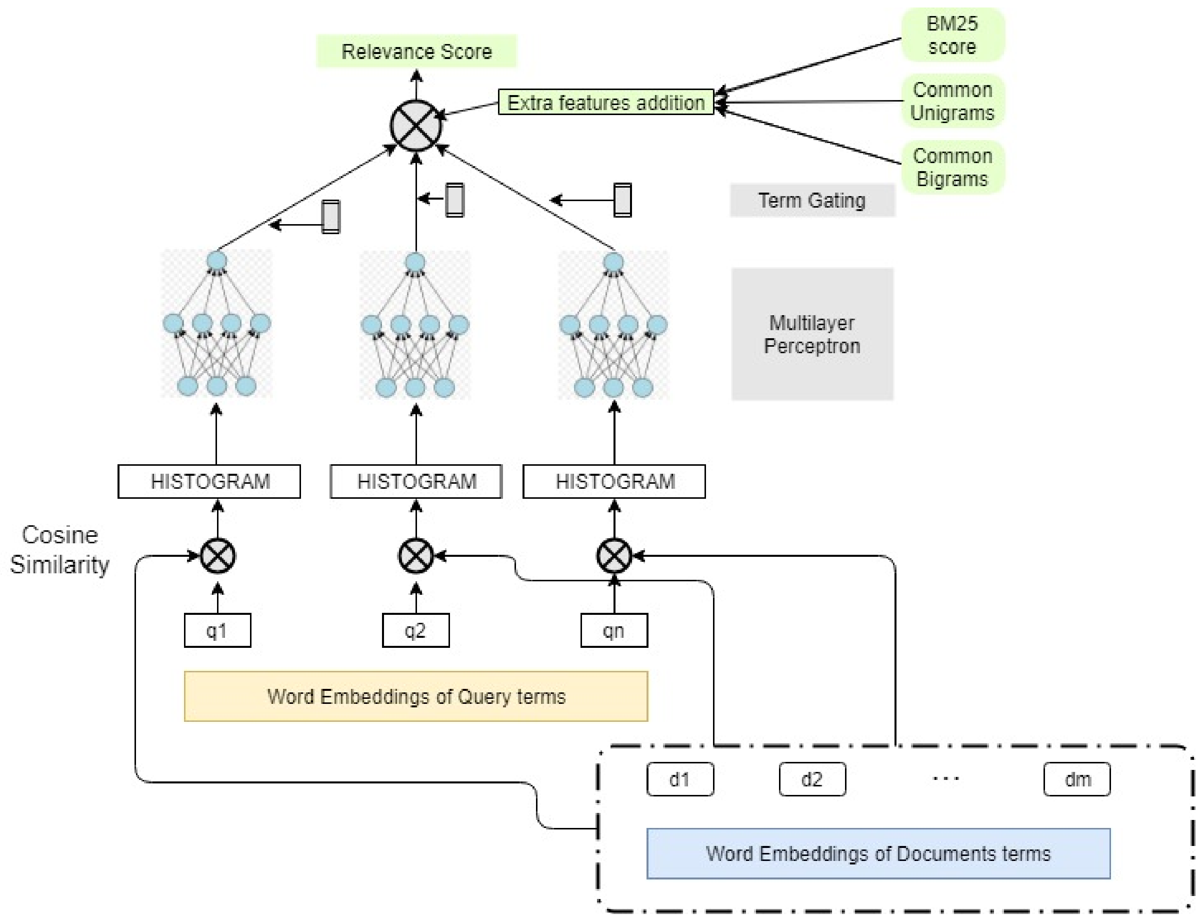


Figure 2: DRMM

In accordance to interaction-focused models, DRMM builds a representation of the interaction of the query with a document and feed it into a Multilayer Perceptron. In more detail, for each query term we calculate the cosine similarity between the word embedding of the query term and the word embeddings [2] of every document term producing as many cosine similarity scores as the document terms. We then construct a histogram that will be fed into a multilayered perceptron(MLP) to produce a score for that query term. The histogram is constructed as follows. First the range of cosine similarity, which is $[-1, 1]$ is discretized into equal sized bins. Then in each bin we assign a similarity score provided that it belongs to the interval of that bin. An additional bin is also used for the scores that are exactly 1. Repeating this process for each query term we end up having an equal sized histograms per query term. There are three version of the number calculated in each bin of the histogram:

- **Count-based Histogram:** In each bin, the number stored is the count of the doc terms whose similarity score belongs to the corresponding bin range.
- **Normalized Histogram:** In each bin, the number stored is the count of each bin normalized by being divided with the total number of doc terms.
- **LogCount-based Histogram:** In each bin, the logarithm of the corresponding count based histogram bin is stored.

[2]using word2vec of gensim library with vector size 100 and a window of 5

Multilayer Perceptron

After constructing the histogram we feed each histogram into a Multilayer Perceptron, the same for every term. In this way a score per query term is produced.

Term Gating Network

In order to combine the scores, a gating network is used. It produces an aggregation weight for each query term controlling how much the relevance score on that query term contributes to the final relevance score. The function(g) used is softmax and we can either use as x the **term vector** of the query term or the **idf** score. If we use term vector, then x_i is the word embedding of query term i and thus the weight w is a vector. If we use the idf score of the query term then x_i is the idf of query term i and thus the weight w is a single parameter.

$$g_i = \frac{exp(w * x_i)}{\sum_{j=1}^{m} exp(w * x_j)}$$

Extra Features Layer

In some cases apart from the score produced by the multiple MLPs using the histogram representations we also wish to introduce extra features such as the number of common bigrams between the query and the document. To do this we will introduce an extra layer of weights one for each feature. If we introduce for example the number of common unigrams and bigrams mentioned the extra layer will be:

$$docscore * w_1 + unigrams * w_2 + bigrams * w_3$$

These weights will also be trained by the model to define the level of participation of each score to the final output score of the model.

Training

As DRMM is used to solve a ranking problem, we have to use a pairwise ranking loss function. Such functions take as input a query(q) and two documents, a positive (d^+) and a negative (d^-) and try to separate them by scoring the positive document higher than the negative. So in order to train the model we have to construct a triple of (q, d^+,d^-) and then use hinge loss function(L) to train the model:

$$L(q, d^+, d^-) = max(0, 1 - s(q, d^+) + s(q, d^-))$$

where:

s(q, d) denotes the predicted matching score for (q, d)

Sometimes it happens that the neural network overfits in one particular feature ignoring the rest. In order to avoid this behavior we can apply a technique called dropout. **Dropout** (Srivastava et al., 2014) is a regularization technique used to reduce the overfitting phenomenon of neural networks by preventing complex co-adaptations on training data. The way dropout works is by letting a unit active with a probability p. In the cases where the unit is not active the corresponding weight is not updated and therefore the network tries to separate the positive and the negative document using only the features for which the units are active.

6 Data Preprocessing

6.1 DRMM Data Preprocessing

Out of 600000 candidates remained in the previous cleaning stage, a smaller number of candidates will be kept as a further cleaning is necessary to exclude the noisy data.

As mentioned above DRMM takes as input a query and a document, it creates a histogram for each query term and it feeds each histogram into a Multilayer Perceptron in order to produce a score per query term. Then it combines these scores and outputs a similarity score between the whole query and the document. As we use hinge loss to train the model, we have to extract for each query, a relevant to the query document called positive and a less relevant document called negative. Also as it is demonstrated by previous applications it is beneficial to create additional features to enhance the model's performance. In our application the additional features to use is BM25 score of the document and the number of common unigrams and bigrams between the query and the document. So in order to train and test the model in this phase we need to retrieve for each candidate the **query** that describes him/her better, a **positive job**, a **negative job** and their corresponding **BM25 scores** calculated by Elastic Search Engine.

Query Bag Of Words

To create a bag of words describing the candidate we use the jobs in the job list of each candidate apart from the one with the highest score that will be used as the positive job. From each job we get a bag of words(bow) from the 4 most informative fields of the job(title, function, requirement summary, keywords) excluding stopwords[3](very common words). This bow will then be used to retrieve relative jobs from the jobs' database using Elasticsearch engine. But as we don't wish for the query to be too long (> 300 terms) we sometimes need to discard a few terms. The decision on which words to discard is non trivial and three options were examined in this application:

- Discard the words contained in a field of the data where less descriptive words are contained, in our case requirement summary.
- Use the inverse document frequency(idf) of the words in the query to find the words that are most common(small idf) and discard them.
- Keep in the query the words that are common among the multiple job ids describing a candidate.

In this application, the words of the requirement summary field are discarded first and then if the query is still too large(>300) words with small idf are discarded. But we make sure to keep the words that appear in multiple job ids of a candidate even if their idf is small.

Positive job

As positive job for each candidate we use the job in which the candidate had the highest score. It is important to mention that we discard the candidates for which the positive job is not among the top 100 jobs retrieved by the Information Retrieval engine as it may be the case that the query bag of words does not describe the candidate efficiently.

[3] using nltk's english and greek stopwords

Negative job

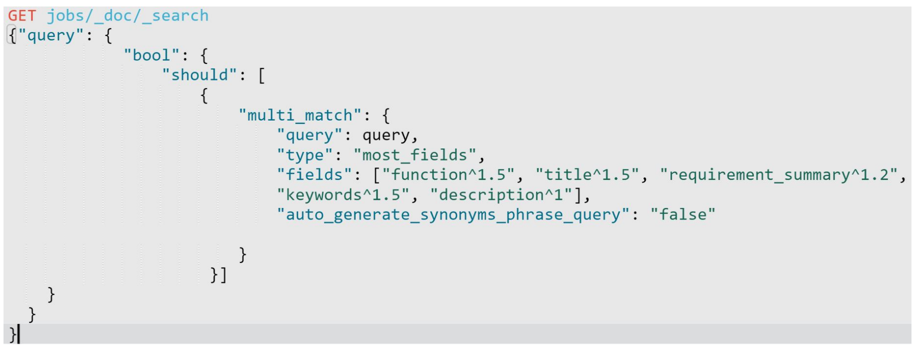
The choice of the negative job can be done in many different ways. In this project three choices were examined:

- Use a job that was ranked low in the results of the fully informative query in Elasticsearch but this does not guarantee that it will be less relevant than the positive job.
- Choose a job randomly
- Make a less informative query and choose randomly a job from Elasticsearch engine results.

In this application we have chosen to extract a negative job from the results of a less informative query. More specifically we create a query for the candidate using 1/6 of the original query and retrieve 100 jobs from the database. Then we choose one job id randomly keeping also its BM25 score given by Elastic Search Engine. In this way we hope that the documents retrieved will be less relevant to the candidate but not entirely irrelevant.

ElasticSearch Query formation

For the formation of the DSL query(the query that we will use into elasticsearch) we use multimatch command. Multimatch command takes candidate's bow, creates tokens and tries to find the tokens in the fields given in the tab "fields". As we have a "should" and not a "must" in the bool tab it is not necessary that a document will contain all the query terms given. But if common terms are found between the query and the fields of the document - job, the document will be assigned a higher score and therefore ranked higher on the results. As observed in the image above, during the formation of the query, weights are also assigned so that BM25 score, scores higher, when a term is found on the field for example title than on the field description. This was added because as we know a term found on the title of a job is more important for the matching of a candidate's skill with the job than the case where the term is found on the description of the job. Therefore, in this query the weights assigned are, a weight of 1.5 on the fields "function", "title", "keywords", a weight of 1.2 on the "requirement summary" field and a default weight of 1 on the "description" field which often contains noise.



```
GET jobs/_doc/_search
{"query": {
            "bool": {
                "should": [
                    {
                        "multi_match": {
                            "query": query,
                            "type": "most_fields",
                            "fields": ["function^1.5", "title^1.5", "requirement_summary^1.2",
                            "keywords^1.5", "description^1"],
                            "auto_generate_synonyms_phrase_query": "false"

                        }
                     }]
    }
  }
}
```

Figure 3: Main query

Training - Test Data

In order to train and evaluate the model we create two types of dataset. The first one is used both on the training stage and on the evaluation stage and the second one is used to evaluate both the baseline and DRMM models by calculating a metric called mean average precision explained in section 7.

Pairs Preprocessing:

This prepossessing is used to create a dataset consisting of the triplet (q,d^+,d^-) that will be used both for DRMM training and evaluation. For evaluation, these data are used to calculate the accuracy considering as success the fact that the positive job's score is higher than the negative. The process is demonstrated in the chart below:

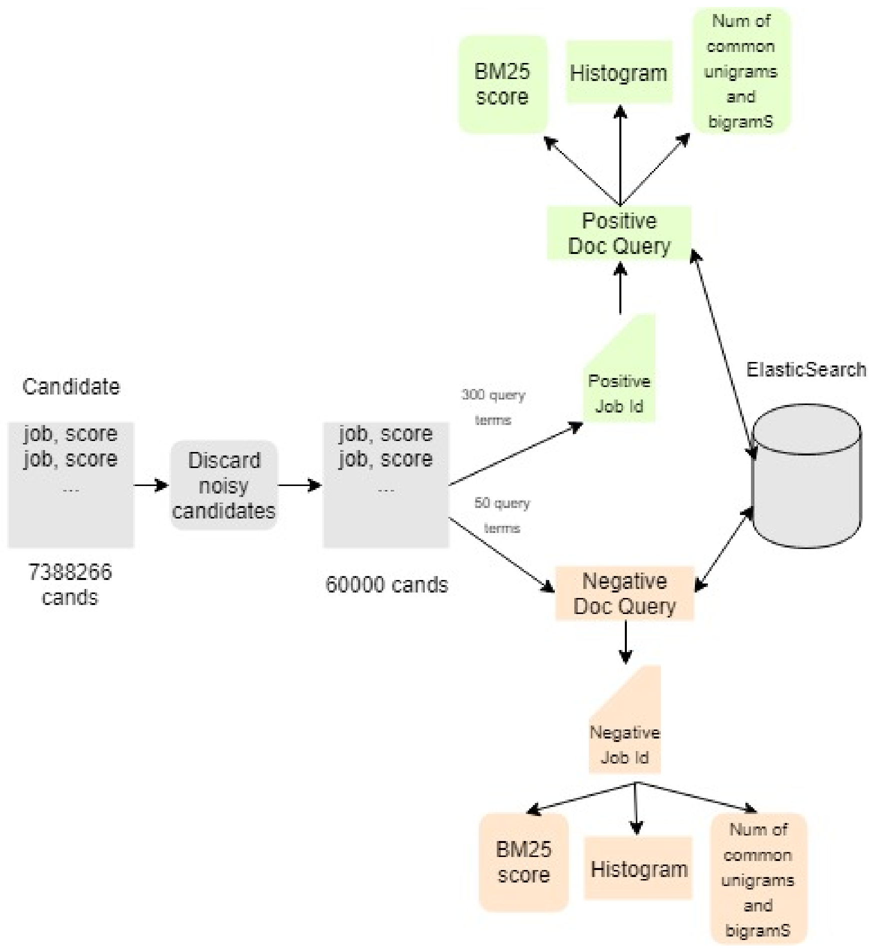


Figure 4: Pairs data preprocessing flowchart

As shown in figure 6, after discarding the noisy data we choose from the bucket of the previous applications of the candidate, as positive, the top scored job. Using the rest of the job ids in the bucket, we create a query bag of words(bow) containing 300 terms. Using this bow we search in jobs database using Elasticsearch and retrieve the 100 most relevant jobs along with their BM25 score. If the job id of the job that we have chosen as positive exists among these results, we store its BM25 score that we wish to try to use as an extra feature in DRMM model and keep the candidate. If not, we discard that candidate.

After retrieving the positive document along with its bag of words we use its terms along with the terms of the query to calculate a log-countbased histogram for DRMM. We also use the positive job's bow to count the number of common unigrams and bigrams with the query in order to try to use them as features in DRMM model.

In order to choose a negative document as mentioned we construct a less informative query, retrieve the top 100 jobs of that query using Elasticsearch and then we randomly choose one as negative. We store its BM25 score, and using its bag of words along with the full informative query we create a histogram per query term, and calculate the number of common unigrams and bigrams as in the positive document.

MAP Preprocessing:

This preprocessing will be used to calculate the mean average precision (MAP) explained in chapter 7.

In order to evaluate the performance of the models an experiment has been designed. For each candidate we retrieve 50 jobs by using his/hers fully informative query bag of words that we consider as positive. We also retrieve 50 negative jobs using a less informative query with words chosen randomly among the original query bag of words, created for that candidate. Then we rerank the 100 jobs using both the baseline and the DRMM, models and calculate the average precision on the top 10 per query and the mean average precision on multiple evaluation samples. Below this data processing is demonstrated.

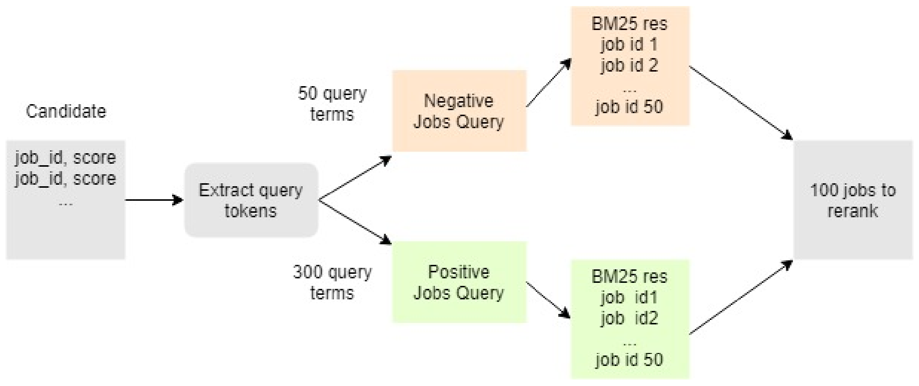


Figure 5: MAP data preprocessing flowchart

Using pairs preprocessing we extract 27000 training examples, 3000 tuning examples and 3000 testing examples while MAP preprocessing is used to extract only evaluation data, therefore 3000 tuning data and 3000 testing data.

7 Evaluation Metrics

The measures used to evaluate the performance of the models are accuracy, mean average precision at 10, precision at 10 that is manually calculated and an extra metric introduced that will be called "diversity measure".

7.1 Accuracy:

Accuracy is the number of correct predictions made divided by the total number of predictions, multiplied by one hundred (100) to turn it into a percentage. In this project, we consider as correct prediction the fact that given a triple of (q, d^+, d^-), the positive job (d^+) is scored higher than the corresponding negative(d^-).

7.2 Precision at k

Precision at k is the fraction of relevant instances among the k retrieved instances where k is the number of documents where we would like to count precision. In this project we use k = 10 as we would like to examine the relevance of the top 10 jobs recommended. It is a useful metric for the ranking task but it fails to take into account the positions of the relevant documents among the top k.

$$\text{precision}=\frac{|\{\text{relevant documents}\}\cap\{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$$

7.3 Average Precision at k

This measure calculates precision in each position until we reach the kth position and then it calculates the average. Therefore it takes into account the position of the ranking by giving more value to the relevant documents positioned on top of the ranking.

$$\text{AveP} = \frac{\sum_{i=1}^{k}(P(i)*rel(i))}{R}$$

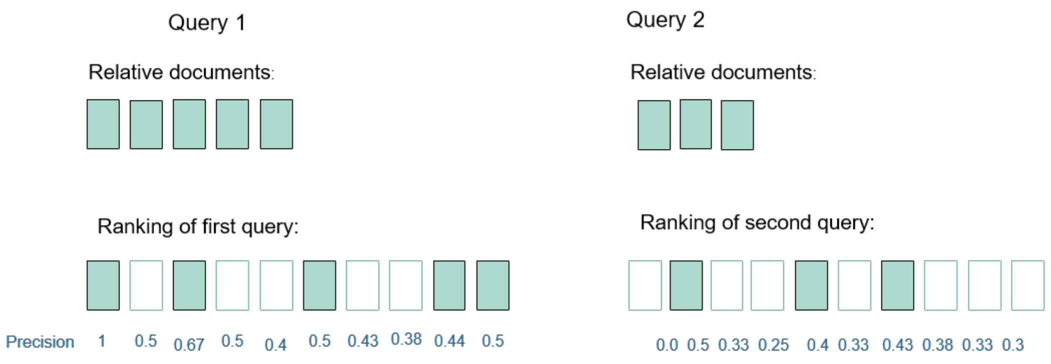
where:
R is the number of relevant documents,
k is the number documents per query (10 as we calculate average precision at 10),
rel(i) is 1 if document is relevant to the query , 0 if document is not relevant and
P(i) is the precision until point i

7.4 Mean Average Precision (MAP):

Mean Average Precision: The mean of the Average precision.

MAP $= \frac{\sum_{q=1}^{Q}(AveP(q))}{Q}$ where Q is the number of queries.

Example:



Average Precison Query 1 = (1 + 0.67 + 0.5 + 0.44 +0.5) / 5 = 0.62
Average Precison Query 2 = (0.5 + 0.4 + 0.43) / 3 = 0.44

Mean Average Precision = (0.62 + 0.44) / 2 = **0.53**

7.5 Diversity measure

Apart from the typical information retrieval measures in this project we would like to be able to check the difference between the top ten results of our models and the ranking returned by Elasticsearch (using BM25 metric) to avoid overfitting to the initial results. During the creation of the experiment to calculate MAP we consider as ground truth the ranking returned by Elasticsearch as we get 50 positives and 50 negatives based on the large difference that their BM25 scores have. So, it becomes clear, that creating exactly the same ranking as Elasticsearch would have a great mean average precision implying great performance from the corresponding model while in fact the results are of no use as we seek for a possible improvement of the model based on the BM25 score. In order to do that we will define a measure that counts the average number of non-common job ids between a reranking model and the ranking based on the BM25 score. In the following example, we have a list of the top ten job ids ranked by DRMM and BM25. The common ids between them are 3 and therefore diversity measure for just one sample is 10 - 3 = 7. If we examined more samples we would take the average. Therefore, this measure takes values on the interval [0, 10].

```
DRMM top10 job ids: [268249, 268242, 586030, 425455, 415567, 423177, 151136, 564627, 287302, 236996]
BM25 top10 job ids [268249, 377566, 585118, 586030, 226761, 425455, 485859, 496419, 241922, 151136]
Common ids: {268249, 268242, 236996}
```

8 Experiments

In this section the results of DRMM model and of a baseline model are presented and compared in terms of the MAP, accuracy and diversity metrics explained above. Also precision at 10 is calculated manually using samples of Elasticsearch ranking and DRMM results.

8.1 Baseline ranking results

As a baseline ranking model a reranking of the 100 most relevant jobs per candidate has been calculated using gensim's library word mover's distance. Then a baseline Mean Average Precision at 10 and a diversity measure is calculated and compared to the results of Deep Relevance Matching Model. Baseline ranking results can be found on the results tables of DRMM models.

8.2 DRMM Results

For the MLP of the DRMM model, 10 hidden layers are used along with 10 units per layer. The maximum number of epochs used is 50 but with an early stopping rule in which if the accuracy on the development set on the current epoch has changed less than 0.008 from the previous epoch the training is interrupted on this epoch.

In order to find the best version of DRMM model, four executions where tried:

Version 1:

In the first version the simplest form of DRMM was executed where the only feature used was the extracted histograms.

Version 2:

In the second version we introduce as extra features the number of common unigrams and bigrams between the query bow and the document and BM25 score. But as our experiment is designed in such a way that the relevant jobs always have higher BM25 score than the non relevant we have to prevent the model from overfitting on the BM25 score as it will produce the same ranking as Elasticsearch engine. In order to do that we apply a dropout with p=0.3 where p is the probability of keeping a unit active. This version of DRMM has a diversity metric of 2 which means that it overfits on BM25 scores producing similar to the Elasticsearch top 10 results.

Version 3:

In the third version the number of common unigrams and bigrams are used as extra features. But this time we separate the training into two phases.

In the first phase we freeze the weights of the last layer where the extra features are introduced keeping only the histogram scores weights active. Then we train the model and store the weights of the MLP network.

In the second phase we freeze all the weights that were previously trained and unfreeze the ones of the last layer.

This version of DRMM turned out to be inefficient producing low accuracies of 75 percent on the training and development set and a MAP of 0.8.

Version 4:

In the forth version of DRMM we use as extra features the number of bigrams and BM25 score. Then we train the model exactly as in version 3 but this time as we have introduced BM25 score as feature we apply dropout in the second phase of training. The dropout applied on the weight controlling the participation of BM25 score has p=0.2.

At first it seems that the results are satisfying as we have high accuracies of 0.95 both for the train and the development set. But the diversity measure is 2 which means that the model produces on average almost the same results as Elasticsearch ranking.

So it seems that only version 1 is effective and we display the results below.

DRMM-version 1 results

On this version of DRMM, on the training phase the model uses as its only feature the histogram. The following table is used to construct the learning curves found bellow. As observed, the metrics calculated are accuracy on the training set, accuracy on the development set and MAP on development set. The number of development data is 3000. In the table below, we observe that the model's performance is very satisfying even while using only 9000 training samples. The accuracy on the training set is almost the same as on the development set and both the accuracy on development data and mean average precision, which are the measures of interest, are satisfying.

num_of_training_data	train_accuracy	accuracy_on_development	map_on_development
9000	0.932111	0.932	0.965704
18000	0.929333	0.939	0.968045
27000	0.926667	0.945	0.966276

Learning Curve on Train Accuracy - MAP on development set

The following learning curve demonstrates in the blue line the accuracy on the training set and in the orange line the Mean Average Precision at 10 on the development set.

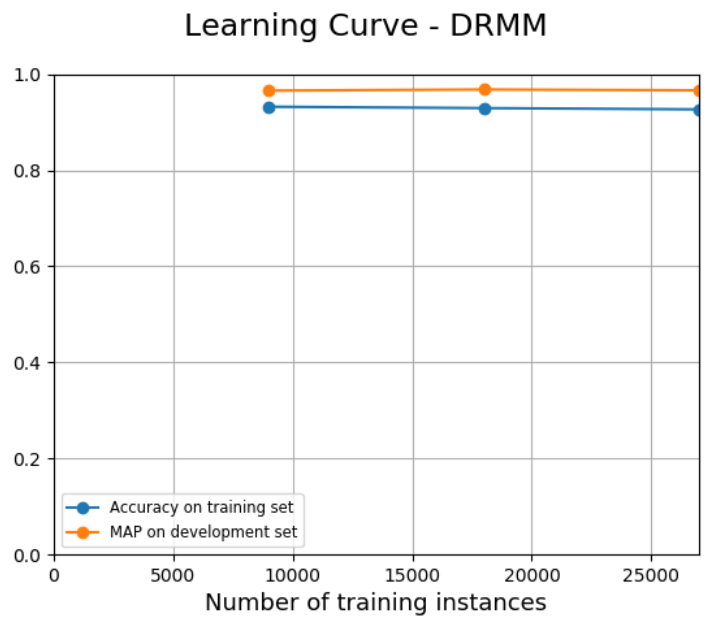


Figure 6: Learning Curve DRMM version 1

As observed MAP remains high throughout the training.

Learning Curve on Train Accuracy - Accuracy on development set

In this learning curve the blue line corresponds to the accuracy on the training set while the orange is the accuracy on development set.

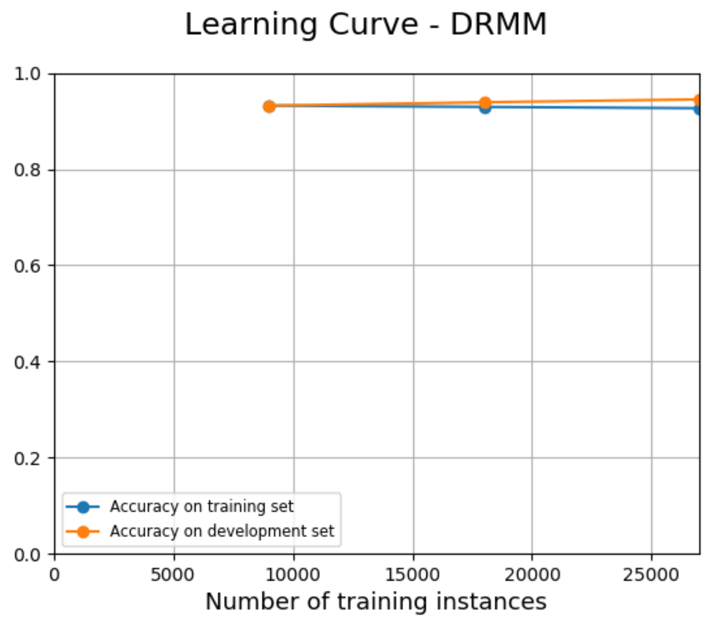
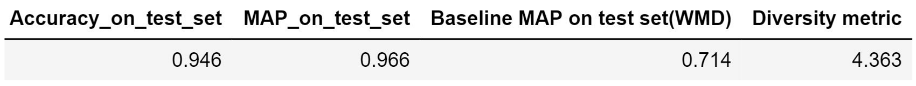


Figure 7: Learning Curve DRMM version 1

Considering both the Learning Curves and the table above, the accuracy on development set is increasing on adding more training samples while the training accuracy is almost the same.

Results on test set

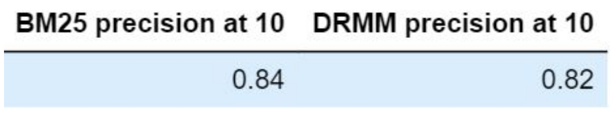


Accuracy_on_test_set	MAP_on_test_set	Baseline MAP on test set(WMD)	Diversity metric
0.946	0.966	0.714	4.363

Comments on test results
DRMM version 1 model has a great accuracy of 94.6 percent on scoring the positive document higher than the negative on the test set. Furthermore according to the MAP and the diversity metric the model does rerank the results produced by Elasticsearch bringing successfully on top relevant to the candidate jobs. The baseline model on the other hand fails in many cases producing a MAP of 0.71.

Manually Calculated Precision at 10

In order to conclude on whether the results of DRMM are useful we have to manually check the relevance of the results of both Elasticsearch ranking and DRMM. In detail, we get a sample of 100 candidates and calculate the precision at ten. It is important to highlight that the relevance of a candidate to the corresponding job announcement is not clear and if we wanted to accurately calculate precision we would have to let the users-candidates answer how much interested they are in the jobs proposed. But for this project we will just answer with a yes or no on whether the jobs on the results seem relevant in a subjective way. The results on the precision are the following:



BM25 precision at 10	DRMM precision at 10
0.84	0.82

Precision at 10 is almost the same in both methods and approximately 80 percent of the results are relevant to the candidate.

9 Conclusions

Based on the metrics produced above, we can conclude that DRMM model can separate effectively the positive and the negative documents while the baseline reranking method in many cases fails. This is an important observation as Word Mover's Distance is not a random metric. We can also observe that according to the diversity metric, DRMM models can rerank Elasticsearch results producing different top ten job announcements. But as we couldn't conclude from those metrics whether the baseline DRMM or BM25 results are better, we manually calculated precision at 10 for 100 randomly selected candidates. The conclusion was that BM25 and DRMM results have almost the same precision of approximately 80 percent. Considering that DRMM is a computationally expensive model we would have to ask the user for feedback on the recommendations to decide on whether DRMM model is worth applying.

10 Future Work

DRMM-fields

A possible future work for this project would be to try another model based on DRMM that instead of using the document as a whole, takes into consideration that each document(job announcement) is divided into fields(title, function, requirement summary etc). In this version instead of creating a histogram for each query term by calculating cosine similarities with all the terms of the document we divide the document into parts according to its fields. Then we calculate a histogram per query for each field separately. In this way DRMM will be able to adjust weights for the relationship of each query term to a part of the document and therefore it could for example weight the similarity to the "title" field more than the similarity to another field.

References

Brokos, G. (2018). Document reranking with deep learning in information retrieval. MSc thesis, Department of Informatics, Athens University of Economics and Business.

Guo, J., Fan, Y., Ai, Q., and Croft, W. B. (2017). A deep relevance matching model for ad-hoc retrieval. *CoRR*, abs/1711.08611.

Kusner, M., Sun, Y., Kolkin, N., and Weinberger, K. (2015). From word embeddings to document distances. In Bach, F. and Blei, D., editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 957–966, Lille, France. PMLR.

Manning, C. D., Raghavan, P., and Schutze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, UK.

McDonald, R., Brokos, G., and Androutsopoulos, I. (2018). Deep relevance ranking using enhanced document-query interactions. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1849–1860. Association for Computational Linguistics.

Robertson, S. and Zaragoza, H. (2009). The probabilistic relevance frameword: Bm25 and beyond. *Foundations and Trends in Information Retrieval*, pages Pages 333–389.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research.*