Implicit Skills Extraction Using Document Embedding and Its Use in Job Recommendation

Akshay Gugnani,[1] **Hemant Misra**[2*]
[1]IBM Research - AI, [2]Applied Research, Swiggy, India
aksgug22@in.ibm.com, hemant.misra@swiggy.in

Abstract

This paper presents a job recommender system to match resumes to job descriptions (JD), both of which are non-standard and unstructured/semi-structured in form. First, the paper proposes a combination of natural language processing (NLP) techniques for the task of skill extraction. The performance of the combined techniques on an industrial scale dataset yielded a precision and recall of 0.78 and 0.88 respectively. The paper then introduces the concept of extracting implicit skills – the skills which are not explicitly mentioned in a JD but may be implicit in the context of geography, industry or role. To mine and infer implicit skills for a JD, we find the other JDs similar to this JD. This similarity match is done in the semantic space. A Doc2Vec model is trained on 1.1 Million JDs covering several domains crawled from the web, and all the JDs are projected onto this semantic space. The skills absent in the JD but present in similar JDs are obtained, and the obtained skills are weighted using several techniques to obtain the set of final implicit skills. Finally, several similarity measures are explored to match the skills extracted from a candidate's resume to explicit and implicit skills of JDs. Empirical results for matching resumes and JDs demonstrate that the proposed approach gives a mean reciprocal rank of 0.88, an improvement of 29.4% when compared to the performance of a baseline method that uses only explicit skills.

1 Introduction

Formal job search and application typically involves matching one's profile or curriculum vitae (CV) with the available job descriptions (JD), and then applying for those job opportunities whose JDs are the closest match to one's CV, and also considering his/her needs, constraints, and aspirations. A few of the things that a person may consider while doing this matching are: a) required skills mentioned in the JDs and skills possessed by self, b) current salary versus salary offered in the new job, c) future prospects after joining the new job, etc. Some of the entities are easy to extract from a JD, for example, the salary offered in a job. However, some other entities, for example, skill extraction (are Python and Java an animal and an island in Indonesia, respectively, or two object-oriented programming languages) and future prospects of a company (it is subjective as well as dependent upon market conditions), need serious consideration.

Though tremendous progress has been made in general purpose search engines, job search engines have made only modest progress. The reasons for this could be several, and some of them are: a) CVs and JDs are typically not written the way well-formatted articles are written in newspapers etc, b) CVs may contain tables and other formatting features to make them look attractive, but this makes it difficult to obtain relevant information from them, c) matching skill keywords between CVs and JDs may not yield good results because of the complex link between the skills, d) the JDs may be too descriptive or too simplistic, and may not uncover the essence of the offered positions and roles. Considering all of these, and also other factors, automated job search engines need research investment to make them robust and improve their performance. This paper proposes methods to overcome some of the above-mentioned challenges.

The fundamental premise this paper builds upon is that skills are one of the most important aspects while matching CVs to JDs, and play a major role in recommending JDs which are the best match for a CV. The following are the important contributions of this paper:

- A new methodology which combines several natural language processing (NLP) techniques for robust skill extraction from natural occurring texts in CVs and JDs is proposed
- An approach for inferring implicit skills in a JD (skills not explicitly mentioned in the JD) is introduced, and a method to extract these implicit skills from other similar JDs is presented
- A bi-directional matching algorithm to match skills between CVs and JDs is suggested to obtain the most relevant job recommendation for each CV

The rest of the paper is organized as follows: In Section 2, related work from the areas of skill extraction and job recommendation is briefly described. Data sources used in this paper and proposed approach are explained in Sections 3 and 4, respectively. The performance of the proposed job recommendation system is presented in Section 5 followed by conclusions.

2 Related Work

We have divided the related work into two broad sections:

1. Skills Extraction or Taxonomy Generation Systems
2. Job Recommendation Systems

[*]This author contributed during his tenure in IBM

2.1 Skills Extraction/Taxonomy Generation

A candidate acquires skills through formal education, vocation, internships, and/or previous jobs' experience. In due course of time, the candidate may start identifying (new) relevant jobs based on the basis of these acquired skills. The key function of a job search engine is to help the candidate by recommending those jobs which are the closest match to the candidate's existing skill set. This recommendation can be provided by matching skills of the candidate with the skills mentioned in the available JDs. A common approach while doing a skill match is to use standard keyword-matching or information retrieval framework as explained in Salton and Buckley (1988). A few challenges of this kind of approaches are: a) The skill may be mentioned in different forms or in terms of synonyms (e.g. cplusplus, c++; programming, scripting, etc.) in CVs and JDs, b) There could be skills that may not be specified in a candidate's profile or a JD, but can be easily determined by business knowledge (for example, 'java' being an object-oriented programming (OOP) language, its experience also indicates experience of OOP), and c) A skill could be an out of dictionary skill, that is, a not-so-common skill-term missing in the dictionary or from a new unseen domain for which the system may not have skills. A framework for skill extraction and normalization was proposed in Zhao et al. (2015). In this paper, a taxonomy of skill was built and Wikipedia was utilized for skill normalization. In Kivimäki et al. (2013), authors proposed a system for skill extraction from documents primarily targeting towards hiring and capacity management in an organization. The system first computes similarities between an input document and the texts of Wikipedia pages and then uses a biased, hub-avoiding version of the Spreading Activation algorithm on the Wikipedia graph to associate the input document with skills.

Colucci et al. (2003) introduced the concept of *implicit skills*. Inspired by their work we have explored a new method in this paper to mine implicit skills using word and document embeddings. In Lau and Sure (2002), authors described a methodology for application-driven development of ontologies, with a sample instantiation of the methodology for skills ontology development. In Bastian et al. (2014), the team at LinkedIn built a large-scale topic extraction pipeline that included constructing a folksonomy of skills and expertise and implementing an inference and recommender system for skills.

2.2 Job Recommendation Systems

The main idea of a job recommendation system is to provide a set of (job) recommendations in response to a user's current profile. In these systems, the users typically can upload their skills or resume or their job search criterion; similarly, the employers or their agents can upload JDs or skills set needed etc along with information such as location, position and other job specific details.

In the job matching space, many studies have been conducted to discuss different recommender systems related to the recruiting problem as explored in the survey by Al-Otaibi and Ykhlef (2012). Among them, Malinowski et al. (2006), discussed a bilateral matching recommendation systems to bring people together with jobs using an Expectation Maximization (EM) algorithm, while Göleç and Kahya (2007) delineated a fuzzy model for competency based employee evaluation and selection with fuzzy rules. Paparrizos et al. (2011) used Decision Table/Naive Bayes (DTNB) as hybrid classifier. To accomplish the task of job recommendation, these systems used many manual attributes and various information retrieval techniques. However, such recommendation systems typically are only effective within a single organization where there are standardized job roles. At an industry sector level such as Information Technology or across such different industry sectors (such as retail, insurance, health care), mining and recommending the most relevant career paths for a user is still an unsolved research challenge.

In the literature, many authors have used machine learning algorithms also for developing a job recommendation system. The idea of modeling the career paths and predicting the job transition was proposed in Mimno and McCallum (2008). In this paper, a topic model was used for discovering latent skills from resumes. In Liu et al. (2016), a time-based ranking model was applied to historical observations, and a recurrent neural network (RNN) was used to model sequence properties to perform the task of job recommendation. In our previous work, Gugnani et al. (2018), we proposed the generation and representation of an individual user profile in context of their skills, and generated a temporal skill-graph that can be used to recommend optimal career path by suggesting next set of skills to acquire. Recently Lin et al. (2016) used a convolutional neural network (CNN) to match resumes with relevant JDs. Recent advances in job recommendation are covered in the survey by Al-Otaibi and Ykhlef (2012).

3 Data Sources

We mined the web to extract a heterogeneous mixture of JDs from various open-source websites. The entire dataset consists of 1.1 Million mined JDs. It has a substantial mix from multiple domains like IT/Software, Health-care, Recruiting, Education and 48 other such domains. This data is used to train our Word2Vec and Doc2Vec models which are explained further in Section 4. Since no standard large open source dataset exists for the task of CV to JD matching, we approached a research team (Maheshwary and Misra (2018)) who had worked on this problem using deep Siamese Network. The dataset borrowed from them consists of 1314 resumes which came in as a part of summer research intern application at their company and a set of 3809 JDs from various domains. We have used this dataset for our full job recommender system evaluation so that we can compare our results with some existing published results.

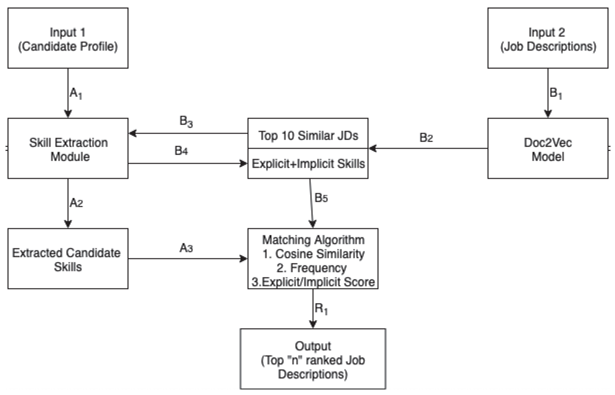
In order to identify a term or phrase as a skill, we created a base skill dictionary as proposed in Gugnani et al. (2018). This skill dictionary is created by mining the web for datasources rich in skill phrases and field terminologies (Onet [1]

[1]https://www.onetonline.org/

and ComputerHope [2]). The validation of these skill-terms was done with the help of various techniques explained in Section 4. This skill dictionary contains 53,293 skill-terms, and has a diverse mixture of skill-terms across various domains ranging from basic soft skills to domain-specific skills.

4 Proposed Approach

Our proposed system consists of 3 core units that are shown in the system architecture in Figure 1.



Input 1 (Candidate Profile)
Input 2 (Job Descriptions)
A_1
B_1
Skill Extraction Module
B_3
B_4
Top 10 Similar JDs
Explicit+Implicit Skills
B_2
Doc2Vec Model
A_2
B_5
Extracted Candidate Skills
A_3
Matching Algorithm 1. Cosine Similarity 2. Frequency 3.Explicit/Implicit Score
R_1
Output (Top "n" ranked Job Descriptions)

Figure 1: System Architecture and Flow Diagram

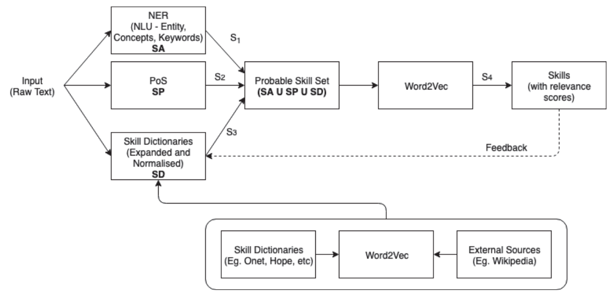
The system includes:

- Skill extraction module
- Module identifying similar JDs given a JD
- Module matching skills from candidate profile to skills from JDs

Each of these modules is elaborated in the following sub-sections. As shown in Figure 1, there are two inputs to the system. Input 1 is a candidate profile and input 2 is a set of JDs. The system is designed such that different inputs are processed asynchronously. In Figure 1, the steps for processing input 1 are labelled with Steps A_1 to A_3, and the steps for processing input 2 are labelled with Steps B_1 to B_5. When a candidate's CV is passed to the Skill Extractor module, it extracts skill-terms from the CV and assigns them to the candidate's profile. Input 2, which has the input JDs, is processed first by the Doc2Vec model and then for every JD best matching 'n' JDs are retrieved, This is followed by the process of extracting implicit skill-terms for each JD – the detailed processing steps of this process are explained in the below sub-section.

4.1 Skill Extraction

The Skill Extractor module is used to identify and extract skill-terms from a given piece of natural unstructured text – these skill-terms can be single-word or multi-word phrases. As shown in Figure 1, the Skill Extraction module processes both the candidate profile as well as the JDs to identify potential skill-terms from the unstructured input text. The module parses each individual sentence in the input as a single unit of raw text. The Skill Extraction module leverages several natural language processing (NLP) techniques to identify skill-terms (when we refer to a skill-term or a skill-phrase, we imply a single word or multiple words which may be a skill). As shown in Figure 2, we have broadly classified them into the following four modules: a) Named Entity Recognition (NER), b) Part of Speech (PoS) Tagger, c) Word2Vec (W2V), and d) Skill Dictionary.



Input (Raw Text)
NER (NLU - Entity, Concepts, Keywords) SA
PoS SP
Skill Dictionaries (Expanded and Normalised) SD
S_1
S_2
S_3
Probable Skill Set (SA U SP U SD)
Word2Vec
S_4
Skills (with relevance scores)
Feedback
Skill Dictionaries (Eg. Onet, Hope, etc)
Word2Vec
External Sources (Eg. Wikipedia)

Figure 2: Skill Identification Flow Diagram

From the raw text, each of the front three modules extracts a set of terms along with a module-specific "score" for each extracted term. Based on this score, the term/phrase is identified as a "Probable Skill" as shown in Figure 2. Combining the four module-specific scores, we compute an overall "relevance score" which indicates how likely is an identified term/phrase a skill.

Named Entity Recognition (NER)
NER is a subtask of information extraction that seeks to locate and classify terms occurring in natural text into pre-defined categories such as names of persons, organizations, locations, expressions of times, quantities, monetary values, percentages, etc. Apart from usual NER, the Watson NLU [3] services also identify keywords, entities and concepts from natural text. We leverage them to identify and validate noun-phrases as skills or technological terms. We parse the input text through this NLU system and generate a list of terms that are identified in these three categories. We classify this list as a set of "Probable Skills" **SA** (in Figure 2). These terms are further processed with a Word2Vec model to assign them a relevance score. This step is explained with an example in the sub-section on Combined Flow (Section 4.1).

Part of Speech (PoS) Tagger
PoS Tagging is the process of marking up a word in a given text (corpus) as corresponding to a particular part of speech, based on both its definition and its context, which is its relationship with adjacent and related words in a phrase, sentence, or paragraph. A simplified form of this is the identification of words as nouns, verbs, adjectives, adverbs, etc.

To identify how skill-terms are represented in a plain text or JD, we did a manual annotation exercise to identify text patterns of skill-terms occurrence. We asked five domain experts to manually label and annotate in few hundred JDs the

[2]https://www.computerhope.com/

[3]https://console.bluemix.net/docs/services/natural-language-understanding/categories.html

terms/keywords that they recognize as skill-phrases. During this exercise we observed that the skills are very subjective and vary not only based on an industry or job requirement but also in context of the person evaluating it. For instance, “writing” may not be regarded as a skill or important skill for a “Software-Developer” role. Hence to identify a term as skill we took the help of inter-annotator agreement - for a term to be a skill, majority of the annotators must identify it as a skill.

We then processed the same set of JDs through the Stanford Core NLP Parser or PoS Tagging to identify noun, verbs and adverb phrases and obtained their parsed tree. We analyzed these JDs and manually annotated how the skills commonly occurred in terms of PoS tags. Based on this data we developed generalized set of rules to identify potential occurrence of skills in a sentence. Based on the inter-annotator agreement we identified and defined rules where three or more annotators had identified a skill-term and the PoS tag occurrence of those terms. We leveraged these patterns/rules in plain text to identify potential new skill-terms that may not be present in our skill dictionary. An example of a rule can be -

If a sentence has a comma separated list of nouns, where one or more nouns is a skill then the other set of nouns are probably skills.

These rules weer programmed into our system and executed on encountering such an instance in any new text. An example case is discussed in the sub-section on Combined Flow (Section 4.1).

Word2Vec (W2V)

W2V, in the work of Mikolov et al. (2013), is a group of related models that are used to produce word embeddings. W2V takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in that vector space. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located in close proximity to one another in the vector space.

As depicted in Figure 2, our W2V model plays an integral role in identifying and learning new skill phrases. It is trained on a corpus consisting of 1.1 Million JDs, varied over multiple domains, including, but not limited to, IT/Software, Health-care, Recruiting, Education and 48 other such domains. It also includes the Wikipedia pages of the terms present in the skill-dictionary.

A W2V model tokenizes using white spaces, hence identifying a single-word skill-terms is straightforward. However, in order to improve our results, we operate under the assumption that a skill-term can be a set of words or phrase as well. Some instance of multi-word skills can be “Web Development”, “Computer Programming”, “Proficient at Client Retention”, “Hard Working”, “Ability to Work Under Pressure” etc. Since we are unaware as to which phrases may be a skill-term we assume that if vectors for individual words from the skill phrase are close to each other in the vector space than their average will be close to this cluster and would be a good representation of the skill phrase. Hence for a multi-word phrase detected as potential skill-phrase, we break the multi-word phrase to single words and average the vectors of single words to obtain a vector for the phrase.

In the embedded W2V space, every potential skill-term is compared with the skill-terms existing in the skill dictionary. The highest W2V cosine similarity of the potential skill-term with these known skill-terms is assigned as the potential skill-term’s skill-score. We also leverage user feedback mechanism to learn new skills and improve the scoring of new identified skills over-time. The assigned skill-score indicates relevance score of each potential skill-term, which in turn indicates how likely is the identified term or phrase a skill.

Skill Dictionary

We curated a base knowledge source of skill-terms that we use as the initial Skill Dictionary. The base skills were curated by mining various online public dataset resources which had well classified labels on terms identified as skills. These terms were then validated by a team of three annotators as skill-terms by using Wikipedia page and category information as an additional validation step. The selected keywords made the initial Skill Dictionary. We focused on Information Technology and soft-skills as the starting skill domains to create the initial Skill Dictionary. Few of the resources we mined were Onet and Hope list. This initial dictionary had 53,293 skill-terms.

This initial Skill Dictionary is further enhanced by our skill-learning and feedback mechanism, which results in a dynamically expanding skill dictionary that autonomously expands to skills of new domains - the process for this is explained in the following sub-sections.

Combined Flow

In this section we illustrate how the Skill Extraction system works. Consider we have the following sentence from a JD: ***“Need candidates with ability to code in Python, Java, and Octave.”***

The above sentence is sent as input raw text to the Skill Extraction system (Figure 2). It passes through its different modules as described below:

- The sentence is passed to the NER module. The NER module identifies and generates a list of Entities, Keywords and Concepts. A combined list of terms thus generated is referred to as **SA**. Each term in the list is also assigned a score denoted by S_1 which indicates the confidence level of the identified term to be a probable skill. The value S_1 can take depends upon how many services (Entity, Keyword and Concept) identified it as a probable skill-term. For the sample sentence, the following terms are identified as probable skills:

$$\textbf{SA} = \left[candidate, code, python, java\right]$$

- The same sentence is also passed to the PoS tagger module. Here the sentence’s syntactical structure is compared with the previously learned pre-defined rules. For the sample sentence, the system rule identifies the sentence to have a comma-separated list of nouns. The rule then checks with the skill dictionary to find if any of the nouns in the comma separated list is a known skill. Two nouns from the list map to known skills "Python" and "Java". This being in complete agreement with the pre-defined rule, the rule gets fired suggesting that the third term is also a likely skill, which in this case is "Octave". Hence the rule is fired for potential skill set, which is referred to as **SP**.

$$\textbf{SP} = \left[octave, python, java\right]$$

Each term in the list is assigned a score denoted by S_2 which indicates the confidence level of the identified term to be a probable skill. The value that S_2 can take depends upon how many pre-defined rules matched the syntactic structure and identified the term as a probable skill-term

- The same sentence is also parsed by the 3 skill-dictionaries (Onet, Hope and Wikipedia), which identify "Python" and "Java" as skills terms. For the sample sentence, **SD** will hold the terms "Python" and "Java".

$$\textbf{SD} = \left[python, java\right]$$

A term may occur in only one dictionary or two out of the three dictionaries or all the three dictionaries. Each dictionary in which a term occurs assigns a weight to that term. For example, if a term occurs in all the three dictionaries, all the three dictionaries assign a weight to it. However, if a term occurs only in one of the dictionaries, only that dictionary assigns the weight to it, The combined score assigned by the three dictionaries to each term is denoted by S_3.

S_1, S_2, S_3 can have values after normalization in the range of [0,1] where 0 indicates a term is less likely to be a probable skill while 1 indicates a term is more likely to be a probable skill.

Once a sentence has been processed by the three parallel modules, the union of these lists forms the "Probable Skill Set".

Probable Skill Set = SA U SP U SD

Probable Skill Set =[candidate, code, python, java, octave]

Using W2V model described earlier, vector representation of each phrase in the Probable Skill Set is obtained. This is compared with W2V vector representation of each skill-phrase found in the Skill Dictionary. For each phrase (of the Probable Skill Set), the cosine similarity between the phrase and all the skill-phrases found in the Skill Dictionary is computed in the vector space, and the max cosine similarity score is found. This max cosine similarity score for each phrase of Probable Skill Set is stored in S_4 - it represents the max cosine similarity score of the given phrase with the skill-phrases of the Skill Dictionary.

Each identified phrase is assigned a "Relevance Score", which indicates how likely it is for the given phrase to be a skill. This score is computed by the following formula:

$$x = \frac{\alpha S_1 + \beta S_2 + \sum_{n=1}^{3} \gamma_n S_3 n + \lambda S_4}{\alpha + \beta + \sum_{n=1}^{3} \gamma_n + \lambda} \tag{1}$$

where x is the Relevance Score.

Table 1 shows the weights assigned to the outputs (terms identified as "Probable Skills") of various modules. The weights were arrived through empirical evaluations, and those giving the best results were chosen and are shown here.

Table 1: Parameters' Weight to Compute Relevance Score

Module	Weight Symbol	Weight Value
PoS	α	1
NER	β	1
ONet Dictionary	γ_1	20
Hope Dictionary	γ_2	10
Wikipedia Dictionary	γ_3	20
W2V	λ	2

Using the formula defined for "Relevance Score", a final score is computed for each phrase present in the Probable Skill Set. This score is a normalized value and ranges between [0,1]. Based on empirical evaluation, we kept a threshold of relevancy at 0.35 - any term with relevance score below that value is removed from the "Skill List". For the sample sentence, from the "Probable Skill Set", the phrase "candidate" has a relevancy score of 0.24 and hence gets removed from the final skill list generated.

$$\textbf{Skills} = \left[python, java, code, octave\right]$$

Performance of Skill Extraction Module

We used the Skill Extraction system described in the previous sections on a set of 100 JDs. These JDs were selected from a corpus that is not used for training the Skill Extraction system and is a part of the test dataset. In addition, these JDs were not seen by the developers who developed the Skill Extraction system. The JDs and the skill-terms extracted by the system were then given to four selected annotators to score the extracted skill-terms in terms of relevance as a skill. A score of 0 was given for a term that has been incorrectly extracted as a skill (false positive) while score of 1 was given for a term that has been correctly extracted as a skill (true positive). We also asked the annotators to analyze and annotate skill-terms which were missed or not extracted but should have been extracted by the system.

Table 2: Skill Extraction Results for some Sub-Systems and Full-System for 100 JDs

Skill	Dictionary Match	NLU	Dic+ NLU+ W2V	Full System: Dic+NLU+ W2V+PoS
Yes	811	695	1002	1158
No	0	849	392	320
Total	811	1544	1394	1474
Missed	395	511	204	158
Precision	1.00	0.45	0.72	0.78
Recall	0.67	0.58	0.83	0.88
F1-Score	0.80	0.51	0.77	0.83

Table 2 shows the analysis of the output of the skill extractor module (**Full System: Dic+NLU+W2V+PoS**) run on 100 JDs. For the 100 JD dataset, the module extracted total of 1474 skills out of which 1158 were identified as skills (**Yes**) and 320 were not identified as skills (**No**) by the annotators. As per the annotators', there were 158 skills which the module failed to identify. The performance in terms of precision/recall/F1-score (F1-score is the harmonic average of the precision and recall) is also shown in the table. Performance of some sub-systems are also shown in the same table for comparison. As expected, dictionary based system (**Dictionary Match**) has very high precision but low recall. On its own, NLU based system **NLU**) performs very poorly. Combined systems (**Dic+NLU+W2V** and **Full System: Dic+NLU+W2V+PoS**) show better performance than individual systems (**Dictionary Match** and **NLU**), and **Full System** performs the best among all the systems.

We observe that the skill extraction system on the dataset of 100 JDs has a precision of 0.78 and a recall of 0.88, giving an F1-score of 0.83. This may be a better performance than that of a related system Javed et al. (2017). However, it has to be noted that the lack of their evaluation dataset makes the direct comparison difficult.

In order to check the generalization of the module's performance, the same experiment was repeated on a larger dataset of 275 JDs. The results obtained were similar and consistent with our initial findings on 100 JDs' dataset, showing the convergence of statistics on a larger dataset. In fact we observe that the performance-metrics are slightly better on 275 JDs. A possible reason for this could be that the randomly selected 100 JDs for the first experiment were from a rather difficult sample.

Table 3: Skill Extraction Performance

System	Precision	Recall	Accuracy	F1-Score
100 JDs	0.78	0.88	0.71	0.83
275 JDs	0.80	0.93	0.75	0.86

4.2 Identifying Similar JDs

The premise of our proposed approach is to identify implicit or latent skills and use them to improve job recommendation for candidates. We define an implicit/latent skill as one which has not been directly or explicitly mentioned in a JD but may be relevant for the job role. For instance, if a JD is for the role of an assistant and mentions that the suitable candidate needs to be well versed with Microsoft Word, this JD has an inherent implicit skill of ability to operate a computer. Our system populates every JD with relevant implicit skills and uses this knowledge for better recommendations.

To identify an approach to extract latent or implicit skills we analyzed hundreds of JDs. We found that for similar JDs often certain skills were omitted depending on how the JDs were written. We theorized that a union of skills of similar JDs would result in a set of skills which are consistent with the original job role. This premise was evaluated by taking a few JDs and analyzing skills from them and their similar JDs – the hypothesis was found to be true in general.

In order to effectively obtain relevant similar JDs, we crawled, mined and curated a list of over 1.1 Million JDs from various online portals and job listings as described in Section 3. These jobs were selectively extracted from multiple sources, spreading over a wide and vast set of domain and job roles. To identify similar JDs we decided to try with a generic approach of finding similar documents which is to cluster similar jobs based on simple information retrieval framework based on term frequency inverse document frequency (TF-IDF). We generated results with modified query representations in Apache Lucene. For comparison, we also performed an analysis in the Doc2Vec vector space to find similarity between JDs. We evaluated both the methods by calculating Mean Reciprocal Rank (MRR).MRR is a statistic measure for evaluating any process that produces a list of possible responses to a sample of queries, ordered by the probability of correctness. We observed that the MRR of Doc2Vec's ranked output was 0.63 and was much higher than that of Lucene (0.51). Therefore we decided to use Doc2Vec to find similar JDs.

Doc2Vec model was trained on 1.1 million JDs. We performed experiments for hyper-parameter tuning to identify suitable parameter values for Doc2Vec. The following were found suitable: Vector Size=300, Window=8, min_count=5, alpha=0.0254, min_alpha=0.001, Workers=25 and train_epoch=500. Based on this model of Doc2Vec, each individual JD from the input set is passed to the Doc2Vec to find upto 10 similar JDs from the training corpus. We use a threshold value of 0.59 similarity or higher – this value was obtained by manual validation of the top matches and their relevance when compared to the input JD. These top 10 similar JDs are tagged with the input JD. We pass these (input JD and the similar JDs) to the Skill Extractor system. The skills obtained from each input JD is tagged as explicit skills and the skills extracted from the top 10 similar JDs are tagged as the "probable implicit skills". These explicit and implicit skills are then passed to the Matching algorithm.

4.3 Matching Candidate CV and JD

At this stage, we have skills extracted from a candidate's profile and the explicit and probable implicit skills extracted from each JD. We need a method to compare and match the skills at both sides and generate a list of best matching JDs for each CV. This is a bipartite graph matching problem.

During experiments, it was observed that between a set of candidate's skills and a set of JD's skills, it is highly likely that one set has a greater number of skills than the other set. This would imply that if we did matching of skills along 1-way, then in general a JD with relatively more number of skills than the other JDs would likely to match better with all the CVs. Similarly if a candidate's skills are more than that of the JDs, there will be a likelihood of more JD matches (and higher scores) for that candidate. Both these scenarios could result in a poor matching leading to a poor job recommendation. To overcome this challenge, we decided to perform matching from both sets and then compute an affinity score to remove the variance. We performed a greedy maximal match from the smaller set of skills to the larger set and a maximum matching from the larger set to the smaller set of skills. Once both these scores are computed, an "Affinity Score" is generated for the JD by averaging out scores from both the matching pairs. The "Affinity Score" measures how suitable the recommendation is for the given candidate and JDs, and ranges from [0,1] with 0 being a poor match and 1 being a strong match.

Computing the Affinity Score

In greedy maximal match, given a bipartite graph, it gives the best match from left to right such that the net score of the match is highest. A maximum match is a matching that contains the largest possible number of edges. We may note that every maximum matching is maximal match, but not every maximal match is a maximum matching. We take into account various heuristics for the matching to take place and for an edge to be formed between a candidate's set of skills and a JD's set of skills.

If there is a skill which is present in both the sets (CV skills and JD skills) it will result in an "Edge Score" of 1 and will match to one another. We remove the common skills from both the sets and assign them an edge-weight of 1, and the count of total number of common skills are added in order to account for them in the affinity score.

It was noticed during experiments that there is a strong correlation of the following factors when comparing skills: a) Cosine Similarity using W2V, b) Frequency Factor of the skill in the document, and c) Boosting based on Explicit/Implicit skill. Hence the edge weight computation was designed to incorporate these factors. The following formula is defined for computing the edge-weight when a successful match has been found:

$$Y = (\omega_1 E_1 + \omega_2 E_2 + \omega_3 E_3)/(\omega_1 + \omega_2 + \omega_3) \quad (2)$$

where, Y = Edge Weight, E_1 is the cosine-similarity score between the skills obtained by the W2V model and E_2 is the frequency factor score for a skill that is calculated as the Total Frequency of the skill across all the documents divided by the number of documents. E_3 for explicit skills is given a value of 1 since they are directly relevant across both the documents. In contrast, for implicit skills, E_3 is given a value of 0.5. In the matching algorithm, an edge is only formed between two skills from either set based on their edge score. The other parameters for edge-scoring are listed in Table 4. The edge score is calculated by giving 50% value to the cosine similarity, 20% value to the frequency and 30% value to boosting values for being an explicit skill. The "Affinity Score" of the JD is then calculated as the average of all the Edge Weights it has.

Table 4: Parameters for Edge Weight Computation

Factor	Symbol	Weight
Cosine Similarity Score – W2V	ω_1	0.5
Frequency Score	ω_2	0.2
Explicit-Implicit Score	ω_3	0.3

5 Results: Matching CV and JD

In absence of any standard large open source dataset for job recommendation task, this paper uses the dataset from Maheshwary and Misra (2018) so that we can compare results. To validate the recommendation algorithm we created 2 sub-datasets. In the first sub-dataset we selected a set of 25 Candidate profiles and 100 JDs, and in the second we selected a set of 200 Candidate profiles and 10,000 JDs. We created these 2 separate sub-datasets to observe if performance varies across a small and large dataset.

We obtained the top 10 recommended JDs for a candidate resume from our system (a) first without using the implicit skills and (b) then using the implicit skills. We got them evaluated by 2 hiring experts and the results are shown in Table 5.

Table 5: Performance for 25 Candidates' Profiles

System	A@1	A@3	A@5
System without Implicit Skills	0.68	0.76	0.88
System with Implicit Skills	0.88	0.96	1

We observe that the accuracy of our system for the highest/first recommended job is 0.68 without the use of implicit skills. With the implicit skills being used the accuracy goes up to 0.88. We additionally observe that the accuracy at the 5th recommendation is at 1 when considering implicit skills – this means that while using the implicit skills our system is able to recommend a perfect job match within the top five recommendations.

When compared with the system of Maheshwary and Misra (2018), which uses siamese network for recommendation on the same dataset of 25 candidate profiles, our system shows an overall improvement of 6.67% in recommending relevant jobs.

The performance of the proposed system for the larger set of 200 candidate profiles is shown in Table 6, and the performance trend is seen to be similarly replicated. We observe the accuracy with implicit skills at the 1st recommendation is 0.84 and the accuracy at the 5th recommendation is at 0.98. The results of this experiment gives us confidence that the proposed method using implicit skills generalizes across the two datasets and hence would likely be consistent over a further larger industry application dataset.

Table 6: Performance for 200 Candidates' Profiles

System	A@1	A@3	A@5
System with Implicit Skills	0.84	0.95	0.98

6 Conclusions and Future Work

In this paper, we have proposed a novel framework for job recommendation – a skill extraction technique is introduced to identify and infer implicit skills for each JD that may not be explicitly mentioned in the original JD. When these implicit skills are used along with typically extracted explicit skills to generate the recommendations, our initial results indicate that an improved and better set of job recommendations are obtained on a locally established and previously published dataset.

In addition, the paper proposed a generalizable ensemble method for skill extraction from unstructured text of resumes as well as JDs. Our ensemble consists of sub-modules such as a Watson-driven NLU system (for extracting concepts, keywords and entities), a PoS tagging system whose output PoS patterns were mapped for skill identification and an expandable dictionary whose base dictionary was seeded from several online open source knowledge bases. The performance of the proposed ensemble method was compared with that of the manual annotators – an accuracy of more than 0.90 and F1-score of more than 0.83 was obtained on two different job datasets. Moreover, several scoring algorithms were explored for matching skills extracted from resumes with (explicit and implicit) skills extracted from JDs.

Due to non-availability of standard large open source dataset for job recommendation task, we evaluated our system on a dataset used in Maheshwary and Misra (2018). Though our results on this dataset are better than the ones reported in the original paper, our immediate next step would involve using a more diverse dataset, a stronger evaluation of the system by including more resumes and leveraging additional techniques for skill identification and extraction. Our future work in this space will involve generating ranked recommendations on different career path options that optimally utilize the accumulated skill and experience of a candidate. We also intend to use skill graph for inferring professional growth of a user and leveraging that for better recommendations. This could help a professional in understanding where he/she stands when compared to his/her peers. Application of skill graph for professional growth inference could also help in comparing two organizations in terms of professional growth of their employees. We propose using these skill graphs to infer **Skill-Gap** in a candidate profile and use this as an additional recommendation to the user. Additionally the system can be used to analyze cost of acquiring a skill and recommend better skills on which to get trained.

7 Acknowledgments

This work was guided by our colleague Danish Contractor who provided expertise that greatly assisted this research. The deployment work was supported by our colleagues Padmanabha Venkatagiri Seshadri and Vivek Sharma.

We are also immensely grateful to Renuka Sindhgatta and Bikram Sengupta for their comments on an earlier versions of the manuscript and support for this work.

References

Al-Otaibi, S. T., and Ykhlef, M. 2012. A survey of job recommender systems. *International Journal of Physical Sciences* 7(29):5127–5142.

Bastian, M.; Hayes, M.; Vaughan, W.; Shah, S.; Skomoroch, P.; Kim, H.; Uryasev, S.; and Lloyd, C. 2014. Linkedin skills: large-scale topic extraction and inference. In *Proceedings of the 8th ACM Conference on Recommender systems*, 1–8. ACM.

Colucci, S.; Di Noia, T.; Di Sciascio, E.; Donini, F. M.; Mongiello, M.; and Mottola, M. 2003. A formal approach to ontology-based semantic match of skills descriptions. *J. UCS* 9(12):1437–1454.

Göleç, A., and Kahya, E. 2007. A fuzzy model for competency-based employee evaluation and selection. *Computers & Industrial Engineering* 52:143–161.

Gugnani, A.; Kasireddy, V. K. R.; and Ponnalagu, K. 2018. Generating unified candidate skill graph for career path recommendation. In *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, 328–333. IEEE.

Javed, F.; Hoang, P.; Mahoney, T.; and McNair, M. 2017. Large-scale occupational skills normalization for online recruitment. In *AAAI*, 4627–4634.

Kivimäki, I.; Panchenko, A.; Dessy, A.; Verdegem, D.; Francq, P.; Bersini, H.; and Saerens, M. 2013. A graph-based approach to skill extraction from text. *Proceedings of TextGraphs-8 Graph-based Methods for Natural Language Processing* 79–87.

Lau, T., and Sure, Y. 2002. Introducing ontology-based skills management at a large insurance company. In *In Proceedings of the Modellierung 2002*, 123–134.

Lin, Y.; Lei, H.; Clement Addo, P.; and Li, X. 2016. Machine Learned Resume-Job Matching Solution. *ArXiv e-prints*.

Liu, K.; Shi, X.; Kumar, A.; Zhu, L.; and Natarajan, P. 2016. Temporal learning and sequence modeling for a job recommender system. In *Proceedings of the Recommender Systems Challenge*, 7. ACM.

Maheshwary, S., and Misra, H. 2018. Matching resumes to jobs via deep siamese network. In *Companion of the The Web Conference 2018 on The Web Conference 2018*, 87–88. International World Wide Web Conferences Steering Committee.

Malinowski, J.; Keim, T.; Wendt, O.; and Weitzel, T. 2006. Matching people and jobs: A bilateral recommendation approach. In *HICSS*.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, 3111–3119.

Mimno, D., and McCallum, A. 2008. Modeling career path trajectories.

Paparrizos, I. K.; Cambazoglu, B. B.; and Gionis, A. 2011. Machine learned job recommendation. In *RecSys*.

Salton, G., and Buckley, C. 1988. Term-weighting approaches in automatic text retrieval. *Information processing & management* 24(5):513–523.

Zhao, M.; Javed, F.; Jacob, F.; and McNair, M. 2015. Skill: A system for skill identification and normalization. In *AAAI*, 4012–4018.