RISE: Resolution of Identity Through Similarity Establishment on Unstructured Job Descriptions

Rakesh Rameshrao Pimplikar[1](✉), Kalapriya Kannan[2], Abhik Mondal[3], Joydeep Mondal[1], Sushant Saxena[4], Gyana Parija[1], and Chandra Devulapalli[5]

[1] IBM Research, Bangalore, India
rakesh.pimplikar@gmail.com, {jomondal,gyana.parija}@in.ibm.com
[2] Hewlett Packard Enterprise, Bangalore, India
kalapriya@gmail.com
[3] Department of Computer Science, IIT Madras, Chennai, India
abhik.mondal1992@gmail.com
[4] Department of Computer Science, IIT Delhi, New Delhi, India
sushant3012@gmail.com
[5] IBM Software Lab, Bangalore, India
cdevulap@in.ibm.com

Abstract. Identity resolution of job description involving cross organizational data would go a long way in addressing several high valued business problems. Job data normalization/sanitation, automated creation of better job descriptions with context preference, description reuse and validation across different sources, semantic classification of jobs, routing of candidates to suitable jobs across different organization etc. are some of the business centric functionalities that can be efficiently built by resolving job description identities. Job descriptions are highly unstructured with free flow textual data consisting of lines describing important attributes of job requirements, like education, skills, experience, role, responsibility etc. Much of the problem is due to the highly unstructured nature of job descriptions. Further, the attributes that are representative of the information in a job description are not readily available from the description. Thus, the process of resolution involves deep data cleansing, classification, attributes identification, and building highly scalable similarity detection algorithms. In this paper, we propose RISE - that uses values of attributes in the underlying job description data and similarity observed in the attributes to resolve identities across organizations. It proposes classification followed by similarity establishment processes that eventually provides high quality of resolution. Through extensive experiments performed on corpus of job descriptions from several real world recruitment systems, we demonstrate that RISE can resolve the identities with high precision and recall.

K. Kannan—A part of the work was done when the author was an employee at IBM Research - India.

A. Mondal and S. Saxena—A part of the work was done when the authors were interns at IBM Research - India.

M. Maximilien et al. (Eds.): ICSOC 2017, LNCS 10601, pp. 19–36, 2017.
https://doi.org/10.1007/978-3-319-69035-3_2

1 Introduction

Identifying right job positions is a key to right opportunity. Job descriptions (JDs) expose job positions by providing information about the positions. To-date job descriptions are prescriptive and dependent solely on expression of job details from employers or recruiting systems. As a result job descriptions differ significantly from one employer to another even for the same role or responsibility, making it difficult for job seekers to identify the right set of jobs. Thus, Job Data Normalization would go a long way assisting the community of job seekers to identify the right set of opportunities for their profiles through standardization of job requirements.

Functional commonalities observed in recruiting systems such as hiring, selection etc., among organizations result in data (JDs, candidate CVs, processes for hiring etc.) that exhibit high commonalities. Such data is often non-standard and each organization chooses its own identifier to refer to each of the records. Our own analysis of about 27000 JDs across 28 different organizations has revealed that terms used to refer to roles vary significantly by name. Thus identifying similar JDs by role names becomes difficult if not impossible. Often times, manual inspection of data is employed to assess contents of JDs and establish similarity and thus identities.

In this paper, we address one such critical problem of resolving identities of JDs and normalizing them across organizations. We present RISE, an identity resolution engine that uses underlying similarity in the nature of the data, representing attributes as a fundamental concept to resolve identities. It establishes novel methods to process unstructured JDs, identify attributes and convert unstructured textual descriptions into structured information. Similarity is established against the first class attributes identified to represent the data. Identities (Job Titles/Department) pertaining to JDs, which are established as similar, are used to build rules to construct equivalence. We enumerate each of the steps in the process in detail and show that our approach identifies similarity across JDs with high accuracy.

Our contributions in this paper can be summarized as follows:

1. Identity resolution of JDs
   (a) Identification of important attributes that are descriptive of the information in JDs.
   (b) Build highly accurate classifier that labels the unstructured text into one or more of the attributes.
   (c) Identify and extract keywords for each of the attributes from unstructured text.
   (d) Establish similarity among JDs based on extracted keywords.
   (e) Establish equivalence among identity titles/roles of JDs.
2. Experiments on real world data sets
   (a) We performed each of above 5 steps on real world data sets collected across 28 different organizations.
   (b) We extensively validated results at each step to ensure that the overall process derives similarity with high accuracy.

Rest of the paper is organized as follows. The system and steps are presented in detail in Sect. 2. Section 3 is used to present the algorithms that we have used. We review some of the existing literature as applicable for our work in Sect. 5. Section 4 presents the details of the experiments and the results. We conclude with directions to future work in Sect. 6.

2 System Overview and Approach

Before providing system details, we present a list of terms along with their definitions in Table 1. We use these terms throughout our paper. It will help readers not to get confused with terms having literally similar meaning.

Table 1. Important terms and definitions

Terms	Definitions
Job Description (JD)	It is an unstructured textual information describing job requirements that a candidate profile should satisfy in order to be considered for the job
Keywords	We use this term to refer to a set of words from an unstructured line of a JD. In general, keywords provide some specific information
Category	It represents a concept/topic for a combination of certain keywords/single key word. We are going to use categories as a feature set as described in Sect. 3.2
Attributes	It is a set of independent identifiable variables that can be used to tag the information provided by every line in a JD. It is also used as a set of labels as described in Sect. 3.2

Our system RISE comprises six different phases. Figure 1 shows all the phases and respective steps involved in extracting the relevant information from highly unstructured text describing a job requirement. Details of every phase are as follows.

1. Attribute Identification Phase (*AIP*). It uses Principal Component Analysis (PCA) to identify attributes that are representatives of the information in JDs (Step 1 in Fig. 1). This is done with the help of the domain and subject matter experts. Algorithm for identification of these attributes is presented in detail in Sect. 3.1. The five attributes those were determined through this analysis are **{*Education, Skills, Experience, Roles, Responsibilities*}**.

2. Classifier Training Phase (*CTP*). It is responsible for training a multi-label ensemble classifier to assign one or more attributes to each line in a JD. The input to this step is training data where every line of historical JDs is already labeled. Output is a classifier model. Ground truth, collected through manual labeling (as described in Sect. 4.1), is used to train the classifier.

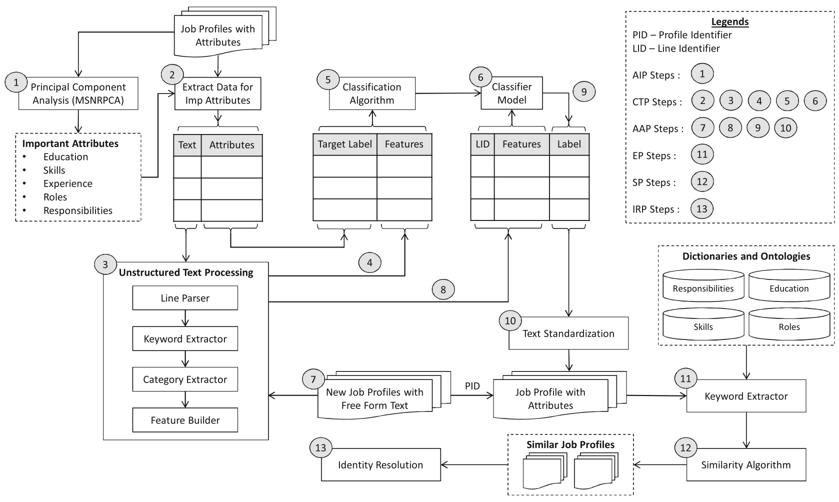


Fig. 1. Phases and steps involved in processing unstructured job descriptions

In step 2 (refer to Fig. 1), we extract unstructured text information from the JDs only for attributes identified in *AIP*. Step 3 involves unstructured text processing, where text is parsed to be broken into a set of lines. Delimiters that have been used to break the unstructured text into lines are {, . ; newline}. Based on the keywords present in a line, a set of categories are extracted for that line. These categories can be in hierarchical order. Eventually a binary feature set is built for every line where every entry in a feature set indicates whether the corresponding category is present or absent. In step 4, this feature set is used to create training data for a classification algorithm. In step 5, a classifier is trained to output a classifier model as shown in step 6. Details of the algorithms that are involved in category extraction, feature set generation, and classifier training are presented in Sect. 3.2.

3. Attribute Association Phase (*AAP*). It uses the classifier model built in step 5 of *CTP* for the classification. Every new JD is passed through unstructured text processing (step 7), which extracts the features against each line (step 8). The extracted features are passed through the classifier which associates each line with one or more attributes (step 9). The output of classifier passes through text standardization (step 10). The main functionality of this component is to convert the keywords available in each labeled line into standard recognizable forms. Trivial differences such as multiple spaces between keywords, presence of delimiters are also cleaned by the text standardization process. Algorithmic details of the text standardization process is presented in Sect. 3.3.

4. Extraction Phase (*EP*). The text is still in the form of unstructured lines after *AAP*. This textual lines now with labels are passed into the Extraction Phase (step 11). In this phase, keywords referring to each of the attributes are identified and extracted from the text. This step converts the unstructured text into a structured JD. The keywords for each of the attributes are stored in the form of comma separated values. Section 3.4 presents the complete set of algorithms for extracting keywords related to each of the attributes from the lines.

5. Similarity Phase (*SP*). Step 12 of Fig. 1 represents a similarity algorithm to find similarity between any two JDs. We have used *Jaccard* similarity measure to determine the similarity of two JDs based on the attributes education, skills, roles, and responsibilities. For similarity measures on the experience, we have provided our own approach of computing similarity based on the number of years of experience. Algorithmic details are provided in Sect. 3.6.

6. Identity Resolution Phase (*IRP*). In this phase, we establish and build a set of rules that can be used to easily identify two equivalent JDs (Step 13 in Fig. 1). Each job description is identified by its job title (a role oriented descriptor) and department. For every pair of similar JDs identified in previous phase, their job titles and departments are stored as a rule in the rules repository. We don't have to analyze any two JDs for similarity in future, if their jobs titles and departments are already present among rules.

3 Algorithms

In this section, we primarily describe 4 algorithms, (1) for identifying important attributes for JDs, (2) for tagging unstructured lines of JDs as one of 5 attributes {*Education, Skills, Experience, Roles, Responsibilities*}, (3) for creating structured job descriptions, and (4) for finding similarity between two job descriptions.

3.1 Identifying Important Attributes for Job Descriptions

This algorithm is used in the *AIP* described in Sect. 2. Our aim is to identify a set of important attributes that are representatives of the information in JDs. *Principal Component Analysis* (PCA) is used in dimensionality reduction. We use a hybrid feature reduction method MSNRPCA based on the combination of feature ranking with PCA. This method was proposed by Yang et al. [20].

We use labeled data set of JDs, as described in Sect. 4.1 to run MSNRPCA on it. Labeled data has values of every attribute for all JDs. An exhaustive list of different attributes derived by qualitative analysis on these JDs is as shown below.

Depth of knowledge, Process, Tools and Technologies, Skills, Domain knowledge, Experience, Business knowledge, Efficiency of communication, Roles expected to perform, Performance expected, Project Management, Schedule Management, Training undergone, Education level, Education streams, Responsibilities

MSNRPCA assigns a score to each of these attributes, based on the importance of every attribute. Higher the score, higher is the importance. For robustness, we create 5 sets of labeled JDs, by randomly sampling 80% of total JDs every time. MSNRPCA is used to assign an importance score to every attribute for every sampled instance of labeled JDs. To simplify the analysis, we map all scores on a rating of 10. Table 2 captures all such scores. You can think of these scores as ratings given to every attribute by 5 different domain experts. Co-related variables are removed from this list thus reducing the set of variables to a minimum number of independent attributes. We perform factor analysis on these scores to eventually identify 5 important attributes. Those are {*Education, Skills, Experience, Roles, Responsibilities*}. Skills attribute can further be classified into "Technical Skills" and "Soft Skills". All skills that involve a known technology, tools, product or methodology are categorized under technical skills. Soft skills include those which do not involve a known tool, but are gained through experience and personal affiliation. Examples of such skills include management skill, communication skill, etc. In this paper we do not discuss this classification of skills and consider only 5 important attributes. We refer these attributes using the notations $y_{\text{edu}}, y_{\text{skill}}, y_{\text{exp}}, y_{\text{role}}$ and y_{resp} respectively.

3.2 Unstructured Text Classification

The bunch of algorithms presented here are used in the *CTP* and *AAP* phases explained in Sect. 2. A job description generally contains unstructured text describing the requirements of an open job position in terms of important attributes {*Education, Skills, Experience, Roles, Responsibilities*}. It is observed that every line of such a job description describes one or more attributes. So in order to create a structured description out of an unstructured one, we first identify which line describes what attributes. This leads to a multi-label classification problem where we need to assign one or more labels to every line L of a job description. In our case, a set of possible labels is $\mathcal{Y} = \{y_{\text{edu}}, y_{\text{skill}}, y_{\text{exp}}, y_{\text{role}}, y_{\text{resp}}\}$. As described in [19], there are two main methods for tackling multi-label classification problem, (1) problem transformation methods that transform the multi-label problem into a set of binary classification problems and (2) algorithm adaptation methods that adapt the algorithms to directly perform multi-label classification. We use the problem transformation method, where we create 5 binary classifiers one for each label in $\mathcal{Y}$. All the steps involved in multi-label classification are explained in detail below.

Feature Extraction. This section provides an approach to unstructured text processing as mentioned in Sect. 2. A set of features is required for a line L to use any standard classification algorithm. So feature extraction is an important step in our approach. In a way, our labels $\mathcal{Y}$ are the categories which we have to identify for every line. Such a category identification needs a mapping between categories and keywords as an input. It looks for keywords in text and based on mapping it figures out most appropriate category. We use Naive Bayes classifier

Table 2. Importance scoring of all attributes

Depth of knowl-edge	Process	Tools and technologies	Skills	Domain knowl-edge	Experience	Business knowledge	Efficiency of commu-nication	Roles	Perform-ance expected	Project manage-ment	Schedule manage-ment	Training under-gone	Education level	Education streams	Responsi-bilities
2	2	5	8	2	8	4	2	8	4	5	2	3	8	7	7
4	3	5	8	4	6	5	3	6	3	4	2	1	7	7	8
5	4	6	6	3	7	6	2	8	3	6	3	4	8	9	8
2	3	6	8	4	7	2	2	6	2	6	2	2	7	8	9
5	3	6	8	4	7	3	3	7	2	5	1	2	7	7	7

to classify text into one of the categories [9]. There are mainly two problems with this approach, (1) one keyword may be mapped to multiple categories, resulting in more than one possible categories for L, and (2) it is very difficult to come up with an exhaustive list of keywords for every category. Thus, using category identification approach for our problem leads to poor results.

Instead we can have a taxonomy for several different categories (including $\mathcal{Y}$), such as *academics*, *products*, *work*, *business*, etc. An example of such a taxonomy is shown in Fig. 2. Similar taxonomy can easily be found in public domain. Every parent node in a taxonomy can be considered as a category and children can be considered as relevant keywords. Using this taxonomy, we can find a set of most suitable categories for a line L. Resultant categories may or may not have categories from $\mathcal{Y}$, but we can deduce categories from $\mathcal{Y}$ provided we have some knowledge about which combination of categories result in which categories from $\mathcal{Y}$.

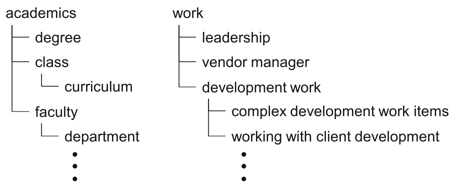


Fig. 2. Taxonomy of categories

Being a good indicator of information present in L, we can use extracted categories as features f^L for L. If given taxonomy has m categories, then there will be m binary features for every L. For all $i \in \{1, ..., m\}$, feature $f_i = 1$ if category c_i is extracted for L, otherwise $f_i = 0$. It creates a feature vector $f^L = \{f_1, f_2, ..., f_m\}$.

Classifier Training. We provide a multi-label classifier training algorithm in this section, which is used by step 5 of *CTP* phase as described in Sect. 2. As mentioned in the previous section, if we know the rules that map combinations of categories into one of the categories from $\mathcal{Y}$, we can easily assign a label from $\mathcal{Y}$ to L. Decision Tree is a good choice to learn such a set of rules from the given data. It also generates a classification tree, which can be used to classify L into one of the labels from $\mathcal{Y}$. Decision tree assigns only one label to L, while we need multiple labels. So we create a decision tree for every label in $\mathcal{Y}$. Though there are several flavors of Decision Tree available in literature, we have used its generic form for simplicity of explanations. However we have presented results for C5.0 [1], CHAID [12] and C&RT [5] in Sect. 4.

Decision Tree requires labeled data for training. So we parse several job descriptions to get a set of lines. During a ground truth collection phase, we receive a multi-label set $y^L \subseteq \mathcal{Y}$ for every line L. Thus we get pairs $\{L, y^L\}$ in training dataset $\mathcal{D}_{\text{train}}$. To train a particular classifier for $y_i \in \mathcal{Y}$, we replace y^L from every $\{L, y^L\}$ pair with a binary value b_i^L where $b_i^L = 1$ if $y_i \in y^L$, otherwise $b_i^L = 0$. This gives us pairs $\{L, b_i^L\}$ in training dataset $\mathcal{D}_{\text{train}}^i$ for label y_i.

Having all labeled data with us and features vectors for every line as described in Sect. 3.2, we build a decision tree T_i for every label y_i.

Multi-label Classification. Given a new unseen line L, we classify it using each of the decision trees built in training phase. Decision Tree T_i classifies a line L and provides label b_i^L. For example, consider a decision tree T_{edu} for y_{edu}. For any L, decision tree returns $b_{\text{edu}}^L = 1$ indicating that L describes education requirements and it returns $b_{\text{edu}}^L = 0$ when L is not about education. We combine all such labels for L from all decision trees and generate a multi-label set y^L where y^L contains a label y_i if $b_i^L = 1$. This approach can be used in step 9 of AAP phase as mentioned in Sect. 2.

3.3 Text Standardization

To address the problem of standardizing text in step 10 of AAP phase as mentioned in Sect. 2, we use ontologies like WordNet[1] and Yago[2]. For example, some recruiters may write "MS Office" and others may write "Microsoft Office". If we don't standardize words like 'MS' into 'Microsoft', it would be difficult to find similarity between two job descriptions, which is our final goal. Ontologies are useful, because they usually contain common entities and their abbreviations. WordNet can be used to find even synonyms which can replace certain keywords in a line. We also use Jaro-Winkler distance [6] on keywords to group similar keywords together. Input to this distance estimator are keywords from lines and dictionary of keywords collected through large databases from organizations. For instance, names of all skills relevant to an organization can be made available in the form of a dictionary. Each of the keywords of the lines are compared with the keywords in the dictionary using the Jaro-Winkler distance estimator to determine the closeness. If two keywords are identified as similar by the algorithm with high confidence level, the keyword in the line is replaced with the keyword from the dictionary. This ensures uniform representation of keywords across text.

3.4 Building Structured JDs Using Keywords Extraction

Once we have every line L of every job description classified as one or more labels from $\mathcal{Y}$, our next task is to extract certain keywords from L, which precisely tells about the job requirements. This set of algorithms relate to the EP phase in Sect. 2. For example, consider following line of a job description.

... **Masters in statistics** or quant-heavy social science program, **bachelor grad** must have extensive research assistant experience; experience of programming in **SPSS** or **SAS**, **C**, **C++** or **Visual BASIC** required...

This line should ideally receive labels y_{edu}, y_{skill} and y_{exp}, as it is talking about education, skills and experience requirements. After labeling, we should extract bold keywords from this line so as to organize it as follows. Observe that though the line is labeled as y_{exp}, we do not extract any keywords for experience. It is because we extract only numeric information for experience attribute,

[1] http://wordnet.princeton.edu.
[2] www.mpi-inf.mpg.de/yago.

for example, number of years of experience. Line given in this example doesn't contain any such information.

Education: **Masters in statistics**, **bachelor grad**
Skills: **SPSS**, **SAS**, **C**, **C++**, **Visual BASIC**
Experience:

We get a set of keywords S_i^L for each attribute y_i from $\mathcal{Y}$ for every line L. Eventually we take union of all sets S_i^* over all lines to get a final set of keywords S_i for attribute y_i. Such sets for all y_i together forms a structured job description. Next we describe how we can extract important keywords from a line L after it has been classified into a set of attributes y^L.

Keywords Extraction for Education. It is observed that education is usually specified in following format.

`<Degree> {of,in,...} <Field>`

For example, "Bachelor of Engineering". It is easier to get an exhaustive list of possible values of degree, while set of possible values of field/stream/department can be huge and we may not be able to create an exhaustive list. But we can utilize the correlation among keywords of education phrase. It is very clear from the above format that parts of speech of an education phrase are Noun-Preposition-Noun. We use NLP (Natural Language Processing) based part of speech (POS) tagging [3] to tag every phrase of a line, which is labeled as y_{edu}. We use OpenNLP[3] tool, which can tag every keyword from a set of 36 different POS tags. We pick all the Noun-Preposition-Noun phrases and lookup for degree related keywords in Noun phrases. For this purpose, we maintain a dictionary of keywords for degree. This dictionary is used for lookup. Once we find a degree keyword that must have been tagged as Noun in a phrase, we can tag other Noun of the phrase as field of the degree. Finally we extract all such phrases, where we can find degree and field combination.

Another possible format for education phrase can be only `<Degree>`. It is applicable for education level lower than graduation where they don't have any specialization. This case is easier to handle by having only dictionary lookup for degree.

Keywords Extraction for Experience. We extract years or months of experience required for a job position if a line is labeled as y_{exp}. To find experience phrases in a line, we can use an approach similar to what we do for extracting education phrases. Formats of experience phrases are observed to be as follows.

```
<Number> {years, months}
<Number> - <Number> {years, months}
<Number> {to} <Number> {years, months}
```

[3] http://opennlp.apache.org.

For example, "... **5 years** of experience in Java...", "... 2–3 years of experience in Databases...", etc. A number can be written either in digits or in words. So we again use NLP based POS tagging to find phrases those are tagged as numbers. If a number phrase is found along with 'year' or 'months' keywords then we extract such number as experience. This can be ambiguous sometime when a time duration is not associated with experience, for example, "... candidate should be at least **25 years** old...". To resolve such ambiguities and boost our confidence, we also look for skills or work related keywords in the vicinity of experience number. Skills and work related keywords can be found using taxonomy of categories mentioned in Sect. 3.2. If we find two numbers separated by '-' or keywords like 'to' as shown in possible formats above, we extract the average of both numbers as experience. For example, we extract keywords "2.5 years" from a line "... 2–3 years of experience in Databases".

Keywords Extraction for Skills and Roles. Skills required for a job position and roles in an organization can be very specific and recruiters use them again and again while writing job descriptions for several job positions. Hence, it is easier to maintain dictionaries of exhaustive keywords for skills and roles. If a line in a job description is labeled as y_{skill}, we lookup into skills dictionary to check if any keywords from dictionary are present in the line. We extract all such matching keywords to tag them as skills. We follow the same procedure for the lines which are labeled as y_{role}.

Responsibilities of a job position are well understood from entire line instead of few keywords. Hence we don't extract any specific keywords for responsibilities attribute. We consider entire line among responsibilities if the line is labeled as y_{resp}. We use Jaro-Winkler distance [6] based string similarity for all dictionary lookups, because it takes into consideration minor spelling mistakes and white spacing between keywords.

3.5 Enriching Dictionaries

The present dictionaries of exhaustive keywords for skills, roles and education may not be exhaustive tomorrow due to ever evolving needs of new skills, roles and education. We propose a way to keep enriching these dictionaries with new keywords by analyzing the frequent occurrences of nouns in lines labeled as one or more of y_{skill}, y_{role} and y_{edu}. As described in algorithm 1, if a noun is not in any of the dictionaries, we count its frequency in the context of different labels. For every such noun, we find a label where the noun has maximum frequency and insert it into the dictionary corresponding to that label, if maximum frequency is above certain threshold. Frequency based analysis is important, because every line can be assigned multiple labels and it can be confusing do decide which dictionary a noun should be inserted into. For simplicity and accuracy, we assume that a noun belongs to only one dictionary.

Algorithm 1. Enrich Dictionaries

```
Input  : Dictionaries $\text{Dict}^i$ $\forall y_i \in \{y_{\text{skill}}, y_{\text{role}}, y_{\text{edu}}\}$, set of lines $\mathcal{L}$, label vector
         $y^L$ $\forall L \in \mathcal{L}$
Output : Updated dictionaries
Counts of keywords for different labels, $C \leftarrow 0$
forall $L \in \mathcal{L}$ do
    $L_{\text{tagged}} \leftarrow$ Tag all keywords in $L$ with part of speech [3]
    $N \leftarrow$ All nouns from $L_{\text{tagged}}$
    forall $n \in N$ do
        if $n \notin \text{Dict}^i$ $\forall y_i \in \{y_{skill}, y_{role}, y_{edu}\}$ then
            forall $y_j \in y^L$ do
                $C_{n,j} \leftarrow C_{n,j} + 1$
            end
        end
    end
end
forall $C_n \neq 0$ do
    $i \leftarrow \text{argmax}_j\, C_{n,j}$
    if $C_{n,i} \geq$ *threshold* then
        $\text{Dict}^i \leftarrow \text{Dict}^i \cup n$
    end
end
```

Following the keywords extraction methods for skill, roles and education, as described in Sect. 3.4, we run the process of enriching dictionaries and then again try to extract keywords. It helps in extracting those keywords, which we could not extract in previous iteration due to lack of their presence in relevant dictionaries.

3.6 Similarity of Job Descriptions

Given two job descriptions J_1 and J_2, our aim is to find how similar they are in terms of attributes y_{edu}, y_{skill}, y_{exp}, y_{role} and y_{resp}. We have provided a detailed procedure in Sects. 3.2 and 3.4 about how to arrive at a structured job description which has sets of keywords S_{edu}, S_{skill}, S_{exp}, S_{role} and S_{resp} for respective attributes. Having these keyword sets where text has been standardized using ontologies as mentioned in Sect. 3.3, we just have to find keywords based overlap between respective sets of job descriptions. $S_i^{J_k}$ represents a set of keywords for job description J_k and attribute y_i. We compute *Jaccard* similarity score between two respective sets $S_i^{J_k}$ and $S_i^{J_l}$ of job descriptions J_k and J_l to get a score $\text{sim}_i^{k,l}$ as follows. *Cosine* similarity [2] can also be used instead of *Jaccard*. For the ease of explanation we mention only *Jaccard* similarity here.

$$\mathrm{sim}_i^{k,l} = \frac{|S_i^{J_k} \bigcap S_i^{J_l}|}{|S_i^{J_k} \bigcup S_i^{J_l}|} \tag{1}$$

This is repeated for all y_i except y_{exp}, because we extract only numbers for experience attribute and not keywords. So *Jaccard* does not work for y_{exp}. Instead we propose a novel similarity measure for finding similarity based on the numeric values.

Similarity for Experience. Given two numeric values e_k and e_l of experience attributes for job descriptions J_k and J_l, dissimilarity of experience is equivalent to normalized gap between two values. As both are non-negative numbers, maximum gap is equal to $\max\{e_k, e_l\}$, which is used for normalization. Thus similarity of experience values can be formulated as follows.

$$\mathrm{sim}_{\mathrm{exp}}^{k,l} = 1 - \frac{|e_k - e_l|}{\max\{e_k, e_l\}} \tag{2}$$

We also define a weight vector $w = \{w_{\mathrm{edu}}, w_{\mathrm{skill}}, w_{\mathrm{exp}}, w_{\mathrm{role}}, w_{\mathrm{resp}}\}$ to specify importance of every attribute for all job descriptions. A weight can be any non-negative number. All similarity scores sim_i are scaled by weights w_i, added up and then normalized to get the final similarity score between two job descriptions. It can be summarized with following equation.

$$\mathrm{JobSim}(J_k, J_l) = \frac{\sum_{i \in \{\mathrm{edu,skill,exp,role,resp}\}} \left(w_i \times \mathrm{sim}_i^{k,l}\right)}{\sum_{i \in \{\mathrm{edu,skill,exp,role,resp}\}} w_i} \tag{3}$$

4 Experiments

We evaluated the performance of every phase of our system by running a set of experiments over a data set as described below. We categorize our experiments mainly into three sets. First set of experiments were conducted to assess the accuracy of classification algorithm (*CTP* and *AAP* phases). Second set of experiments were conducted to assess the accuracy of keyword extraction from labeled text for creating structured JDs (*EP* phase), and third set of experiments were conducted to assess the accuracy of similarity algorithm (*SP* phase). All of these experiments are described in following subsections.

4.1 Data Set

We collected approximately 27000 JDs from 28 organizations including IBM and its clients. Client names are not mentioned in this paper to preserve confidentiality. These JDs were picked from actual jobs posted by organizations for hiring candidates. Distribution of number of JDs picked from 28 organizations is 8000, 4000, 2500 and 500 each from remaining organizations. The organizations are in the area of Information Technology. Thus diverse set of JDs for a single domain were considered. It was observed that these JDs were highly unstructured with free flow text expressing requirements of the job. There was no explicit or consistent expression of lines as skills, experience, roles, etc. These JDs produced about 0.18 million lines which were used as data.

Approximately 5000 lines, chosen randomly, were manually tagged for collecting the ground truth. Human labelers assigned one or more of the labels $\{y_{\text{edu}}, y_{\text{skill}}, y_{\text{exp}}, y_{\text{role}}, y_{\text{resp}}\}$ to every line, depending on what a line was describing. Along with labels, human labelers also annotated the phrases that actually described the labels assigned. Total 6367 phrases were annotated. 10 people contributed in this ground truth collection activity. Every line is labeled and annotated by 2 labelers. Overall agreement on labels and annotations was 86%. We carefully resolved the conflicts while finalizing the ground truth.

4.2 Classifier Evaluation

We compare the performance of our classification algorithm as described in Sect. 3.2 with a baseline approach. Our algorithm uses decision tree classifier, that automatically generates a set of rules for assigning an attribute as a label to every line. On the contrary, baseline approach relies on a set of rules manually provided by domain experts for every attribute. Given a set of categories extracted for a line, baseline approach scans through rules for an attribute and if any rule is satisfied, that particular attribute is assigned to the line. This process is repeated for every attribute.

We conducted experiments of baseline and our algorithm over a ground truth of 5000 lines labeled manually. As rules are readily available, baseline approach doesn't require any training phase. Baseline predicted attributes for every line and later we compared them against the ground truth. Whereas a 5 fold cross validation was used to report precision and recall for our algorithm.

While comparing with baseline, we computed three different set of results for our classification algorithm by selecting a different decision tree algorithm every time. Namely we used C5.0 [1], CHAID [12] and C&RT [5] decision tree algorithms.

As this is multi-label classification problem, we report F1 score for every attribute as shown in Fig. 3. It is clear that F1 score of our algorithm beats baseline F1 score or at least at par with baseline F1 score in all three settings for all attributes except for experience. Improvement ranges from 0 for skills using C&RT to 0.23 for roles using CHAID. For experience, drop in F1 score ranges from 0.04 using CHAID to 0.15 using CR&T. Thus, our algorithm works better than baseline in most of the cases. In remaining cases, our algorithm is not far behind the baseline in terms of F1 score. Additionally, our algorithm can be used with larger data sets. Manual rules in baseline approach may not be exhaustive in case of larger data sets.

Comparing among three different settings for our algorithm, we can infer from above analysis that CHAID is the best suited for our algorithm and C&RT is the worst among three.

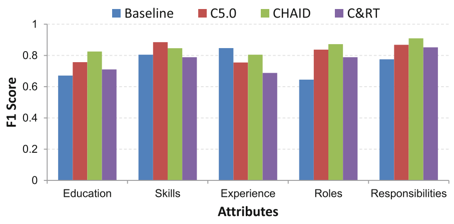
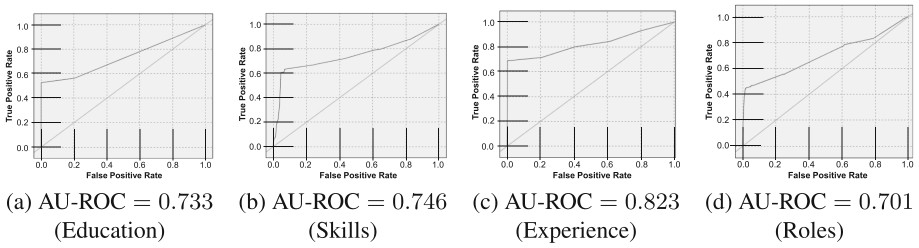


Fig. 3. F1 score based comparison

We plot ROC curves for decision tree classifiers for every attribute with CHAID technique. Classification scores obtained for every attribute and for every line are used for this purpose. These ROC curves are shown in Fig. 4. It is observed that area under ROC curve (AU-ROC) is high for all attributes, that establishes the quality of our algorithm for classification.



(a) AU-ROC = 0.733 (Education) (b) AU-ROC = 0.746 (Skills) (c) AU-ROC = 0.823 (Experience) (d) AU-ROC = 0.701 (Roles)

Fig. 4. ROC curve along with AU-ROC value of a classifier for every attribute

4.3 Keyword Extractor Evaluation

We used 5000 lines from ground truth having attributes assigned to them. For each of these lines, we extracted keywords based on the attributes of lines. We compare the extracted keywords for every line with the annotated keywords for that line in the ground truth. We adopt the standard definition of precision to find the precision of our keyword extraction algorithm in terms of following formula.

$$\text{Precision} = \frac{\sum_{L\in\{\text{all lines}\}} \sum_{i\in\{\text{edu,skill,exp,role,resp}\}} N^{i,L}_{\text{anno, ext}}}{\sum_{L\in\{\text{all lines}\}} \sum_{i\in\{\text{edu,skill,exp,role,resp}\}} N^{i,L}_{\text{ext}}} \tag{4}$$

where $N^{i,L}_{\text{anno, ext}}$ is the total number of keywords those were annotated in the ground truth as well as extracted by our algorithm for a line L and for attribute i. $N^{i,L}_{\text{ext}}$ is the total number of keywords extracted by our algorithm for a line L and for attribute i. We calculated the value of above formula to find what fraction of total extracted keywords were actually describing the attributes of the lines. The value of Eq. 4 was computed to be as high as 0.954.

We also adopt the standard definition of recall to find the recall of our keyword extraction algorithm as follows.

$$\text{Recall} = \frac{\sum_{L\in\{\text{all lines}\}} \sum_{i\in\{\text{edu,skill,exp,role,resp}\}} N^{i,L}_{\text{anno, ext}}}{\sum_{L\in\{\text{all lines}\}} \sum_{i\in\{\text{edu,skill,exp,role,resp}\}} N^{i,L}_{\text{anno}}} \tag{5}$$

where $N_{\text{anno}}^{i,L}$ is the total number of keywords annotated in the ground truth for a line L and for attribute i. This gives us what fraction of total annotated keywords were actually recognized by our algorithm. The value of Eq. 5 was computed to be 0.842. This implies that our keyword extraction algorithm is highly effective with high precision and recall. F1 score can be computed to be 0.896.

4.4 Similarity Algorithm Evaluation

Similarity algorithm provides a score between 0 to 1 for a pair of JDs. Similarity is high, if the score high. One way to evaluate similarity algorithm is to find similarity scores of a JD with every other JD from our data set of 27000 JDs. We can set a threshold on similarity score to find all pair of similar JDs. Then manually find out how many of those pair are actually similar. There are two problems in this evaluation approach. First, setting a threshold value is tricky. One value for a pair of JDs may not be valid for other pair of JDs. Second problem is that inspecting all similar pairs manually is not feasible for possible 27000×27000 pairs. Collecting ground truth for those many pairs is also time consuming and need a lot many human resources.

We decided to go with ranking approach to address these two problems in evaluation. We randomly selected 50 JDs out of a data set of 27000 JDs. For every JD in this set of 50 JDs, we computed similarity scores with every other JD in 27000 set. For a selected JD, we ranked all JDs in decreasing of their similarity scores. We picked top 10 and manually observed how many of them were actually similar. We repeated this for each of the 50 selected JDs. Thus ground truth collection efforts was brought down to 50×10 from previous value 27000×27000. Setting a threshold value is also not required for this evaluation. Just that instead of computing precision and recall, we computed area under ROC curve (AU-ROC) in this setting for each of these 50 ranked lists with 10 JDs each.

We observed that minimum AU-ROC was 0.642, maximum AU-ROC was 0.9 and mean AU-ROC was 0.779. This highlights the effectiveness of similarity algorithm in ranking similar JDs at the top. Ranked list certainly does not provide exact list of similar job descriptions, but it provides an ordered list of JDs, which user can follow to find similar job descriptions. It reduces tremendous efforts of user of scanning all JDs in random order. Based on the application, a threshold value for similarity scores can as well be used or top k JDs can be picked. It will further reduces the screening efforts of user.

We also report precision of similarity algorithm for the sake of completeness, by setting up following experiment. Given above mentioned 50 ranked lists, we set a high threshold of 0.7 for two JDs to be similar. It gave us a set of pairs of JDs, that we predicted as similar. It shortlisted on an average top 15 JDs from every ranked list which increases manual labeling effort from 50×10 to 50×15 pairs. Based on manual labeling, we observed that 88% of predicted similar JDs were truly similar. This sets a high precision value for our similarity algorithm.

5 Related Work

Importance of entity and identity resolution have been established earlier in several research works [4,13,17]. Our work is along the lines of recent approaches which are variants of Fellegi-Sunter Model [8]. In [8] identity resolution is solved as a classification problem - given a set of similarity scores for different attributes of two candidates, classify it as a match or a non-match. Several bodies of research work have advised, compared and learned similarity measures for use in entity resolution (example, [7,18]). Typically in such work, matching is performed individually on each of the attributes and then a transitive closure is used to eliminate inconsistencies. In our work, we establish these attributes through MSNRPCA [20], a hybrid approach for feature reduction based on the combination of feature ranking with PCA, and utilize the similarity to resolve identity. We train classification models for attributes using well established decision tree algorithms such as C5.0 [1], CHAID [12] and C&RT [5].

Entity resolution has been solved in several domains by various research works (example, [15,16]) and to different types of data, including text (example, [14]) and images (example, [11]). RISE targets resolution of entities in the domain of Job Descriptions in a recruiting system. We have highlighted the importance of the problem earlier and the goals of our work have been motivated by real world requirements of recruiting systems. There has been a pressing demand for identity resolution systems where identifying right candidates through one channel for specific organization can be routed to other job descriptions if not found suitable. Furthermore, there has been demands for creation of context sensitive job descriptions based on the existing job descriptions that had the best convergence. For all these purposes, one requires that similarities are established and identities are resolved. A big distinguishing factor is that our data source have been cross organizational. Thus we expect the identities of these job descriptions to be completely different from one organization to another. The problem is more challenging also due to the nature of the attributes. For instance, the numeric values for *experience* attribute requires different measures for obtaining similarity.

A group of researchers have focused on large databases and resolving identities in them. Methods were provided to avoid the quadratic number of comparisons between all pairs of entities (example, [10]). Such methods can be leveraged to reduce number of comparisons while finding similarities between every pair of JDs.

6 Conclusion and Future Work

We have built a system called **RISE** that addresses one of the key issues of identity resolution among job descriptions in recruitment systems. Recruitment systems typically employ technologies that allow centralized storage of data across different organizations. Although, centralized yet underlying unstructured data and lack of resolution techniques have rendered the data less usable for several valuable applications. RISEprovides an end-to-end system for establishing equivalence among identities and resolving them with high precision and recall.

Our future work includes enabling several key capabilities on top of this system such as automated creation of job description based on the context, routing of profiles across different jobs etc.

References

1. C5.0 Decision Tree Algorithm. http://www.rulequest.com/see5-info.html
2. Cosine Similarity Algorithm. http://en.wikipedia.org/wiki/Cosine_similarity
3. Stanford Log-linear Part-Of-Speech Tagger. http://nlp.stanford.edu/software/tagger.shtml
4. Benjelloun, O., Garcia-Molina, H., Menestrina, D., Su, Q., Whang, S.E., Widom, J.: Swoosh: a generic approach to entity resolution. VLDB J. **18**, 255–276 (2009)
5. Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification and Regression Trees. Statistics/Probability Series. Wadsworth Publishing Company, Belmont (1984)
6. Cohen, W.W., Ravikumar, P., Fienberg, S.E.: A comparison of string distance metrics for name-matching tasks. In: IJCAI 2003 Workshop on Information Integration, pp. 73–78 (2003)
7. Cohen, W.W., Ravikumar, P., Fienberg, S.E.: A comparison of string metrics for matching names and records. In: Proceedings of the KDD 2003 Workshop on Data, pp. 13–18 (2003)
8. Fellegi, I.P., Sunter, A.B.: A theory for record linkage. J. Am. Stat. Assoc. **64**(328), 1183–1210 (1969)
9. Frank, E., Bouckaert, R.R.: Naive Bayes for text classification with unbalanced classes. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) PKDD 2006. LNCS (LNAI), vol. 4213, pp. 503–510. Springer, Heidelberg (2006). doi:10.1007/11871637_49
10. Hernández, M.A., Stolfo, S.J.: The merge/purge problem for large databases. SIGMOD Rec. **24**(2), 127–138 (1995). http://doi.acm.org/10.1145/568271.223807
11. Huang, T., Russell, S.: Object identification: a Bayesian analysis with application to traffic surveillance. Artif. Intell. **103**(1–2), 77–93 (1998)
12. Kass, G.V.: An exploratory technique for investigating large quantities of categorical data. J. R. Stat. Soc. Ser. C **29**(2), 119–127 (1980)
13. Li, J., Wang, G.A., Chen, H.: Identity matching using personal and social identity features. Inf. Syst. Front. **13**(1), 101–113 (2011)
14. Li, X., Morie, P., Roth, D.: Semantic integration in text: from ambiguous names to identifiable entities. AI Mag. **26**(1), 45–58 (2005)
15. Norén, G.N., Orre, R., Bate, A.: A hit-miss model for duplicate detection in the who drug safety database. In: KDD 2005, pp. 459–468 (2005)
16. Ong, I.M., Page, D., Dutra, I., Costa, V.S.: Hyperpaths: extending pathfinding to moded languages. In: Proceedings of MRDM 2005, p. 57. ACM (2005)
17. Singla, P., Domingos, P.: Entity resolution with Markov logic. In: Proceedings of ICDM 2006, pp. 572–582. IEEE Computer Society (2006)
18. Tejada, S., Knoblock, C.A., Minton, S.: Learning domain-independent string transformation weights for high accuracy object identification. In: Proceedings of KDD 2002, pp. 350–359 (2002)
19. Tsoumakas, G., Katakis, I.: Multi label classification: an overview. Int. J. Data Warehouse Min. **3**(3), 1–13 (2007)
20. Yang, M.J., Zheng, H.R., Wang, H.Y., McClean, S., Harris, N.: Combining feature ranking with PCA: an application to gait analysis. In: ICMLC 2010, vol. 1, pp. 494–499 (2010)