Benchmarking of a Novel POS Tagging Based Semantic Similarity Approach for Job Description Similarity Computation

Joydeep Mondal, Sarthak Ahuja, Kushal Mukherjee, Sudhanshu Shekhar Singh

IBM Research Lab, India

Abstract. Most solutions providing hiring analytics involve mapping provided job descriptions to a standard job framework, thereby requiring computation of a document similarity score between two job descriptions. Finding semantic similarity between a pair of documents is a problem that is yet to be solved satisfactorily over all possible domains/contexts. Most document similarity calculation exercises require a large corpus of data for training the underlying models. In this paper we compare three methods of document similarity for job descriptions - topic modeling (LDA), doc2vec, and a novel part-of-speech tagging based document similarity (POSDC) calculation method. LDA and doc2vec require a large corpus of data to train, while POCDC exploits a domain specific property of descriptive documents (such as job descriptions) that enables us to compare two documents in isolation. POSDC method is based on an action-object-attribute representation of documents, that allows meaningful comparisons. We use stanford Core NLP and NLTK Wordnet to do a multilevel semantic match between the actions and corresponding objects. We use sklearn for topic modeling and gensim for doc2vec. We compare the results from these three methods based on IBM Kenexa Talent frameworks job taxonomy

1 Introduction

Several contexts require finding similarity between a pair of documents. The problem of finding similarity between a pair of documents also lays groundwork for the problem of clustering similar documents together. Most of the initial research in this domain was based on cosine distance with tf-idf term vectors. Topic modeling based techniques such as LSA and LDA learn an intuitive set of topics from a given corpus of documents, and the topic distribution vectors of documents can be used to find document similarities or cluster documents together. More recently, word2vec and doc2vec based document similarity methods have been gaining popularity.

All the document clustering techniques group similar documents together, while keeping dissimilar documents in different groups. Various document similarity measures such as cosine similarity, Dice's coefficient and Jaccard's coefficient [13] have been used in literature to evaluate document similarity. However, the definition of a pair of documents being similar depends on the problem context. For example, finding the same joke [10] told differently is a vastly different problem from finding whether term paper submissions from two students are the same. Irrespective of the clustering technique used, most document similarity learning methods e.g. LDA [17], Doc2Vec [15] etc. require a large corpus of data to learn document features well. No document similarity computation methods work well if the corpus is small, or if only two documents are to be compared.

In our work of enabling hiring solutions via cognitive collaboration, wherein several agents/players such as job match, diversity champion, cultural assessment agent come together to make holistic hiring decision, one problem that we have faced many times is that of identifying which job requisitions are similar. This problem arises in two contexts:

1. Grouping jobs together: A typical application of machine learning in hiring is to learn success models for various jobs. To be meaningful, the models need to be learned at a sufficient level of granularity. Thus, arises the need to cluster jobs together. Grouping jobs together also arises necessity for a cultural assessment agent as similar assessments can be used for alike jobs.
2. Candidates' previous jobs need to be matched with the opening they apply to (or to the openings that will be recommended to them). This requires comparing job description from their previous jobs to the job openings available in the ATS (applicant tracking system).

Job requisitions typically consist of several well defined components: skill and years of experience requirement, job location and a job description. With the rest being structured fields, job title (covered in [8]) and job description, which typically consists of roles and responsibilities the job entails, becomes the primary component that needs to be matched across jobs. RISE [19] proposes a method of job classification followed by similarity establishment processes that leverages both structured and unstructured components of a job.

In this paper, we compare novel part-of-speech tagging based docu- ment similarity (POSDC) calculation method with doc2vec and LDA based topic modeling method. POSDC analyzes the actions (verbs), objects of each action (nouns) and attributes of the objects (adjectives) that appear in the two job description to be compared.

This paper is organized as follows. In the next section, we describe the literature on document similarity/clustering. Section 3 describes our job description similarity computation methodology and experimental setup. Section 4 explains the evaluation criteria. Section 5 concludes and discusses some future work.

2 Literature Survey

In typical text document classification and clustering tasks, a definition of a distance or similarity measure is essential. The most common methods employ keyword matching techniques. Methods such TFIDF [9] leverage the frequency of words occurring in a document to infer on similarity. The assumption is that if two documents have a similar distribution of words or have common keywords, then they are similar. Researches have also extended this to N-gram based models [16], where group of consecutive words are taken together to capture the context. With large N gram models, typically large corpus of documents are required to obtain sufficient statistical information

[16][17] extended these approaches to include a probabilistic generative model that would explain the frequency of occurrence of words. These methods include PLSI (probabilistic latent semantic indexing) and LSA (latent semantic analysis). The assumption is that there are an underlying latent set of topics (with their individual distribution of words describing the topic) and each document is generated from a mixture of these topics. The above described methods fall in the category of bag-of words models. The major limitation of bag-of-words models is that the text is essentially represent as an un-ordered set of words (or n-words, for n gram models). The long-range word relations are not captured leading to loss of information. Another issue with these techniques is that they rely on the surface information of the words and not its semantics. That is, words with two or more meaning (polysemy) are represented in the same way and two or more words with the same meaning (synonymy) are denoted differently. In other words, the distance among the three words 'bold' , 'brave' and 'timid' are the same.

To alleviate the drawback of bag-of-words model, Le et. al. [15] proposed Paragraph to Vector, an unsupervised algorithm that learns feature representations from variable-length pieces of texts, such as paragraphs. The algorithm represents each paragraph by a dense vector which may be used to predict words in the paragraph. Its construction captures semantics and has the potential to overcome the weaknesses of bag-of-words models. Paragraph Vectors outperform bag-of-words models as well as other techniques for text representations and have achieved state-of-the-art results on text classification and sentiment analysis tasks [15].

Another approach to document similarity is via concept modeling (Wikipedia concepts [11], IBM watson natural language understanding service [4]). The main idea is to use many concepts from Wikipedia or any other encyclopedia to construct a reference space, where each document is mapped from a keyword vector to a concept vector. This captures the semantic information contained in the document. [18] have demonstrated the effectiveness of concept matching to overcome the semantic mismatch problem. However, the concepts themselves are not independent. [12] extended Wikipedia matching to document clustering by enriching the feature vector of a text document by using the correlation information between concept articles.

3 Methodology

Our proposed approach to generate a similarity score among two job description D and D' can be divided into four parts as illustrated in Figure 1. In this section we explain these steps alongside their system implementation in greater detail.

3.1 Document Representation

Each document is treated as a collection of sentences set_{sent}, with each sentence $sent$ being further represented as a collection of sets of triplets - **action, object and attributes**. Consider the following illustrative example.

Job Description Document :

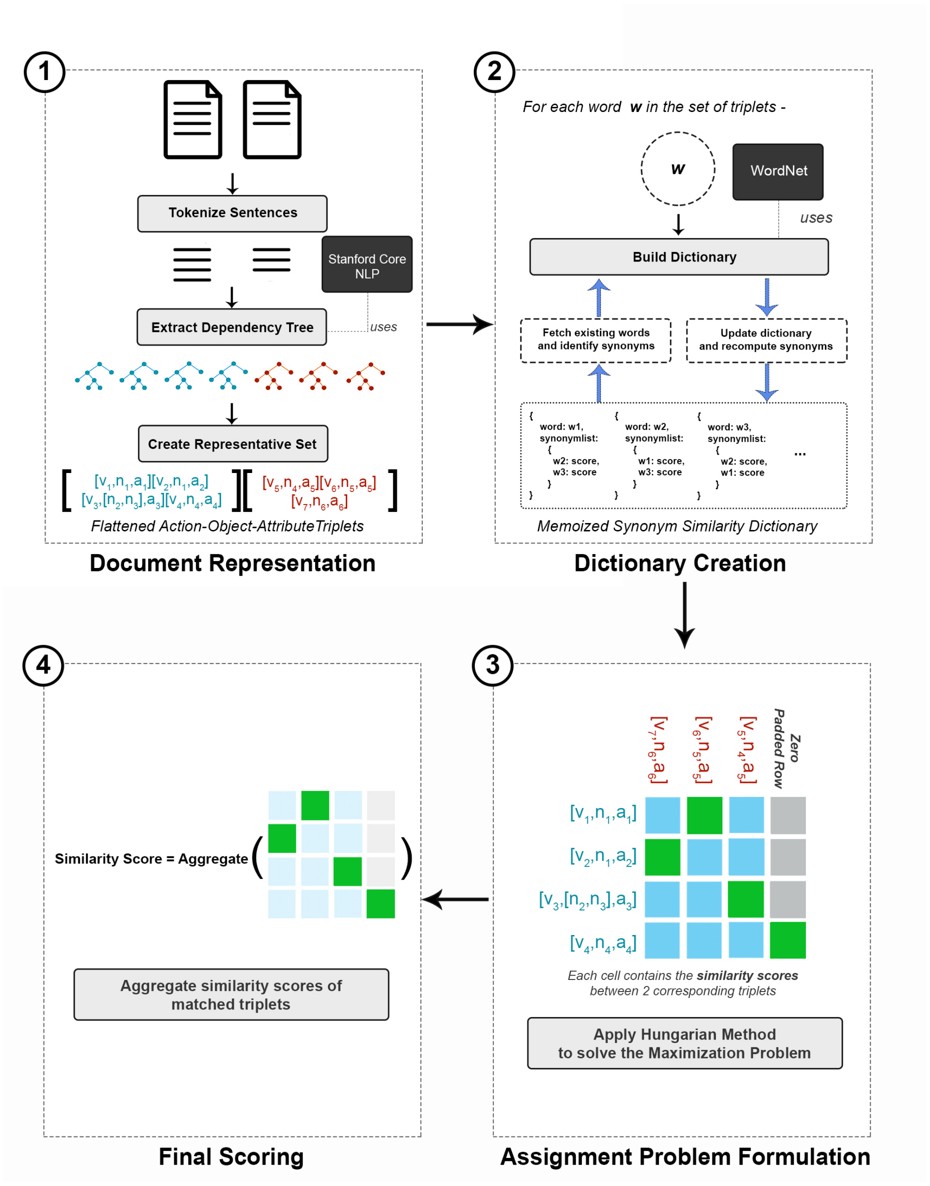
1
Tokenize Sentences
Stanford Core NLP
Extract Dependency Tree
uses
Create Representative Set
$[v_1,n_1,a_1][v_2,n_1,a_2]$ $[v_3,[n_2,n_3],a_3][v_4,n_4,a_4]$
$[v_5,n_4,a_5][v_6,n_5,a_5]$ $[v_7,n_6,a_6]$
Flattened Action-Object-Attribute Triplets
Document Representation
2
For each word w in the set of triplets -
w
WordNet
uses
Build Dictionary
Fetch existing words and identify synonyms
Update dictionary and recompute synonyms
{ word: w1, synonymlist: { w2: score, w3: score } }
{ word: w2, synonymlist: { w1: score, w3: score } }
{ word: w3, synonymlist: { w2: score, w1: score } }
...
Memoized Synonym Similarity Dictionary
Dictionary Creation
3
$[v_7,n_6,a_6]$
$[v_6,n_5,a_5]$
$[v_5,n_4,a_5]$
Zero Padded Row
$[v_1,n_1,a_1]$
$[v_2,n_1,a_2]$
$[v_3,[n_2,n_3],a_3]$
$[v_4,n_4,a_4]$
Each cell contains the similarity scores between 2 corresponding triplets
Apply Hungarian Method to solve the Maximization Problem
Assignment Problem Formulation
4
Similarity Score = Aggregate
Aggregate similarity scores of matched triplets
Final Scoring

Fig. 1: Main Flow Describing the Four Steps for computation of Job Description Similarity

$$
\begin{array}{l}
[\\
\quad w_1\text{: } syn_{w1}\text{:}\{w_2\text{:}sim_{sem\ w1,w2}, w_3\text{:}sim_{sem\ w1,w3}\},\\
\quad w_2\text{: } syn_{w2}\text{:}\{w_1\text{:}sim_{sem\ w1,w2}\},\\
\quad w_3\text{: } syn_{w3}\text{:}\{w_1\text{:}sim_{sem\ w1,w3}\}\\
]
\end{array}
$$

Fig. 2: Dictionary Structure of Keywords and Their Synonyms

```
{
  "title": "academic",
  "synonymlist":{
    "coach": 0.319
    "managed": 0.316
  }
}
```

Fig. 3: JSON Structure of a Dictionary Entry

Determines operational feasibility by evaluating analysis, problem definition, requirements, solution development, and proposed solutions.

Representation of Job Description Document :

1. ***Action****: determines,* ***Object****: feasibility,* ***Attributes****: [operational]*
2. ***Action****: evaluating ,* ***Object****: problem definition,* ***Attributes****: [ ]*
3. ***Action****: evaluating ,* ***Object****: requirements,* ***Attributes****: [ ]*
4. ***Action****: evaluating ,* ***Object****: solution development,* ***Attributes****: [ ]*
5. ***Action****: evaluating ,* ***Object****: solutions,* ***Attributes****: [proposed]*

where $action$ symbolizes the main activity described by that particular $sent$, $object$ represents the entity on which the activity has been acted upon and $attributes$ corresponds to the characteristics of the $object$. We will refer the triplet as t_{POS} , set of triplets corresponding to a sen as $sent_{t_{POS}}$ and set of triplets corresponding to a document D as $D_{t_{POS}}$. Our hypothesis is that these sets of triplets can properly describe a job description document. To verify this statement, we performed a small experiment. We chose five people (experts in Job analytics domain) and gave them the generated set of triplets for 10 job descriptions. Without seeing the original job description documents, they could easily extract the actual essence out of these triplet sets.

As all the job description documents were in English, without loss of generality it can be said that the main activity of a $sent$ i.e $action$ is represented by the non-auxiliary verb v. The entity on which the activity v has been acted upon is generally the object noun corresponding to v in $sent$. Characteristics of an entity are portrayed by the adjectives in English. So, we depicted $attribute$ of an $object$ as the adjectives corresponding to the object noun present in $sent$. We assume all the sentences in job description documents were in . In cases where an entity of t_{POS} contains multiple elements, such as in the case of compound nouns, a list of elements is created instead.

For faster computation we used Apache Spark [1] environment to parallalize the sequential loop in Algorithm 1. Apache Spark provides programmers with an application programming interface centered on a data structure called the resilient distributed dataset (RDD), a read-only multiset of data items distributed over a cluster of machines, that is maintained in a fault-tolerant way [2]. We modified Algorithm 1 to Algorithm 3 to incorporate distributed system capabilities.

3.2 Document Parsing and Dictionary Creation

Given a document D, it's corresponding representation discussed in 3.1 is obtained by parsing the output tree $tree_{dep}$ generated by Stanford Dependency Parser from the NLTK Library [3] for each sentence. Algorithm for creating $D_{t_{POS}}$ is described by below Algorithm 1.

Algorithm 1 Document Representation Algorithm-Sequential

```
 1: procedure DocRepSeqProc
 2:     Input: $D$
 3:     Output: $D_{t_{POS}}$
 4:     $set_{sent}$ ← sentence tokenizer (D)
 5:     $D_{t_{POS}}$ ← null
 6:     for each sent of $set_{sent}$ do
 7:         $sent_{t_{POS}} \leftarrow SenRepProc(sent)$
 8:         $D_{t_{POS}} \leftarrow D_{t_{POS}} \cup sent_{t_{POS}}$
 9:     end for
10: end procedure
```

Algorithm 2 Sentence Representation Algorithm

```
1: procedure SenRepProc
2:     Input: $sent$
3:     Output: $sent_{t_{POS}}$
4:     $Tree_{dep}$ ← Stanford POS Dependency Tree (sent)
5:     $sent_{t_{POS}}$ ← modify $Tree_{dep}$  ▷ action, object and attributes are extracted from the tree
6: end procedure
```

After obtaining the D_{tPOS} we used *memoization* and *precomputation* techniques to build a dictionary $Dict$ of words present in the $D_{t_{POS}}$. The structure of the dictionary is depicted as in Figure 2. In the dictionary, every word w has been stored with its synonym list syn_w. We used Wordnet dictionary from NLTK [5] to get syn_w for a given w. We used cloudant Database to store this dictionary as JSONs. The structure of the JSON is in Figure 3. syn_w for a w consists of only the words which exist in $Dict$ and cross a threshold of semantic similarity score (sim_{sem}). Algorithm to update the dictionary is given in Algorithm 5.

Algorithm 3 Document Representation Algorithm- Parallel

```
1: procedure DocRepParProc
2:     Input: $D$
3:     Output: $D_{t_{POS}}$
4:     $set_{sent} \leftarrow$ sentence tokenizer $(D)$
5:     $D_{t_{POS}} \leftarrow$ null
6:     $Map_{set_{sent}}(SenRepProc)$   ▷ it is executed in cluster machines of Spark environment in parallel for each sentence
7: end procedure
```

WordNet is a large lexical database of English language. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms which is called $synsets$. Each $synsets$ expresses a distinct concept which interlinked by means of conceptual-semantic and lexical relations. Wordnet provides synsets for a given English word [6]. To calculate sim_{sem} between w_1 and w_2 we calculate *wup similarity* score between two synsets corresponding to w_1 and w_2. *Wu Palmer Similarity* or *wup similarity* provides a score denoting how similar two word senses are, based on the depth of the two senses in the taxonomy and that of their Least Common Subsumer (most specific ancestor node) [7]. After getting the scores between each sysnset we took an average of the scores to get the semantic similarity score between w_1 and w_2 and denoted it as $sim_{sem_{w_1,w_2}}$. Algorithm to find sim_{sem} is described in Algorithm 4.

When $Dict$ is empty and the algorithm encounters a new word it creates $Dict$ and stores an entry corresponding to the word. When $Dict$ exists in the cloudant database and algorithm encounters a w then it first checks whether it is present in $Dict$ or not. If w is not present in $Dict$ then it will create an entry for w and will generate a corresponding syn_w by calculating sim_{sem} with every other words in $Dict$. The sim_{sem} of every other words of $Dict$ will also be updated accordingly. While processing each entry in D_{tPOS}, we precompute the semantic similarity scores among the words and store them in a database. Processing of D_{tPOS} is described in Algorithm 6

3.3 Assignment Problem Formulation

After the document parsing stage, the two documents,D and D', are now represented as two sets of triplets, D_{tPOS} and D'_{tPOS}, each with elements t_{POS_1}, t_{POS_2}, ... t_{POS_n} and t'_{POS_1}, t'_{POS_2}, ... t'_{POS_m} respectively. The similarity score between the two job des can be now interpreted as the similarity score between these two sets. The similarity function is explained in detail in the next subsection, and is denoted by **F** for now. To calculate the similarity score between two sets, a naive approach would be to calculate the similarity score between each pair of elements from two sets (S and S'), greedily pick the pair with the highest similarity score and repeat the process till either one of the sets has no element left. This greedy approach, although simple, does not provide an optimal match between the sets being compared. We assume that there are no repeating descriptions in the descriptive document, hence, the representative set for a document too will not have synonymous elements i.e. no same $action$ on the same $object$. This

Algorithm 4 Semantic Similarity Between Two Words

```
1: procedure SemSImProc
2:     Input: $w_1, w_2$
3:     Output: $sim_{sem_{w_1,w_2}}$
4:     $sim_{sem_{w_1,w_2}} \leftarrow 0$
5:     $synSets_{w_1} \leftarrow$ null
6:     $synSets_{w_2} \leftarrow$ null
7:     $synSets_{w_1} \leftarrow$ synsets from Wordnet for $w_1$
8:     $synSets_{w_2} \leftarrow$ synsets from Wordnet for $w_2$
9:     $div \leftarrow 0$
10:    for each $synSet_{w_1}$ of $synSets_{w_1}$ do
11:        for each $synSet_{w_2}$ of $synSets_{w_2}$ do
12:            $wup_{score} \leftarrow$ wup similarity between $synSet_{w_1}$ & $synSet_{w_2}$
13:            if $wup_{score}$ is not null then
14:                $sim_{sem_{w_1,w_2}} \leftarrow sim_{sem_{w_1,w_2}} + wup_{score}$
15:                $div \leftarrow div + 1$
16:            end if
17:        end for
18:    end for
19:    $sim_{sem_{w_1,w_2}} \leftarrow \frac{sim_{sem_{w_1,w_2}}}{div}$
20: end procedure
```

Algorithm 5 Dictionary Update Algorithm

```
1: procedure DictUpdateProc
2:     Input: $w, Dict$
3:     if $w$ doesn't exist in $Dict$ then
4:         $syn_w \leftarrow$ null
5:         for every word $w_i$ in $Dict$ do
6:             $sim_{sem_{w,w_i}} \leftarrow$ SemSImProc$(w, w_i)$
7:             if $sim_{sem_{w,w_i}} >$ threshold then
8:                 append $w_i$ to $syn_w$
9:                 append $w$ to $syn_{w_i}$
10:            end if
11:        end for
12:        add $w$ and $syn_w$ to $Dict$
13:    end if
14: end procedure
```

assumption motivates a one-to-one mapping among the two sets being compared for similarity.

Algorithm 6 Document-Triplet-Set Processing Algorithm

```
 1: procedure PROCTRIPPROC
 2:     Input: $D_{t_{POS}}$
 3:     for each $t_{POS}$ in $D_{t_{POS}}$ do
 4:         $v \leftarrow action$
 5:         $Noun \leftarrow object$
 6:         $Adj \leftarrow attribute$
 7:         DictUpdateProc($v$, $Dict$)
 8:         if $Noun$ is a compound noun then
 9:             for each noun of $Noun$ do
10:                 DictUpdateProc($noun$, $Dict$)
11:             end for
12:         else
13:             DictUpdateProc($Noun$, $Dict$)
14:         end if
15:         for each adj of $Adj$ do
16:             DictUpdateProc($adj$, $Dict$)
17:         end for
18:     end for
19: end procedure
```

To find an optimum one-to-one mapping among the aforementioned two sets, we formulate the problem as an assignment problem [14]. In a generic assignment problem, given the cost of assignment among each pair of elements in two sets, the task is to find an optimal one-to-one assignment among the elements that maximizes/minimizes the total cost of assignment. Our problem of finding such a one-to-one mapping among the representative sets of the descriptive documents can be formulated in a similar way - given **F** as the cost of assignment function among each pair of elements in the two representative sets, the task is to find an optimal one-to-one assignment among the elements that maximizes the aggregate similarity score. Since the two sets being compared can have unequal number of elements, this is a case of an imbalanced assignment problem.

After formulating the problem as a similarity score maximization assignment problem, we use the *Hungarian Method* [14] to extract out the matches. This method takes as input a nxn square cost matrix and post applying a set of matrix operations, outputs an optimal set of n assignments, one per row and column, which offer a maximum cumulative assignment score. Since ours is a case of an imbalanced assignment problem, given 2 sets with m and n triplets each, we start with a mxn cost matrix, where each cell contains the similarity score between the corresponding row and column elements of the matrix. Without loss of generality, we assume $n > m$, and add zero padding to extend the mxn matrix to a nxn one. Rest of the steps for applying the *Hungarian Method* remain the same, as for a typical score maximization assignment problem. We will refer *Hungarian Method* as $Assign_{Hung.}$ in the rest of the paper.

Post this assignment, the following subsection defines the similarity and aggregation functions.

3.4 Score Calculation

We define four similarity functions for calculating similarity score between two job description documents D and D'. We will refer sim_{sem} function as calculating semantic similarity between two words described in Algorithm 4. The definition of the functions are following,

Definition 1. *If v_1 and v_2 are two single-word verbs and V_{sim} is the similarity function between two verbs then V_{sim} is defined as,*

$$V_{sim}(v_1, v_2) = sim_{sem_{v_1,v_2}} \tag{1}$$

Definition 2. *If N_1 and N_2 are two sets of nouns (nouns can be a set in t_{POS} in case of compound noun) and N_{sim} is the similarity function between two noun sets then N_{sim} is defined as,*

$$N_{sim}(N_1, N_2) = \frac{(1 * |N_1 \cap N_2| + \frac{\sum_{i=1}^{n} \sum_{j=1}^{m} sim_{sem_{N'_{1_i}, N'_{2_j}}}}{|N'_1| * |N'_2|})}{(1 + |N_1 \cap N_2|)} \tag{2}$$

where, $N'_1 = N_1 - (N_1 \cap N_2)$ and $N'_2 = N_2 - (N_1 \cap N_2)$

Definition 3. *If A_1 and A_2 are two sets of adjectives (adjectives can be a set in t_{POS} in case of multiple adjectives corresponding to a noun) and A_{sim} is the similarity function between two adjective sets then A_{sim} is defined as,*

$$A_{sim}(A_1, A_2) = \frac{(1 * |A_1 \cap A_2| + \frac{\sum_{i=1}^{n} \sum_{j=1}^{m} sim_{sem_{A'_{1_i}, A'_{2_j}}}}{|A'_1| * |A'_2|})}{(1 + |A_1 \cap A_2|)} \tag{3}$$

where, $A'_1 = A_1 - (A_1 \cap A_2)$ and $A'_2 = A_2 - (A_1 \cap A_2)$

Definition 4. *If t_{POS_1} and t_{POS_2} are two sets of triplets consisting of (v_1, N_1, A_1) and (v_2, N_2, A_2) respectively.Then, t_{sim} is the similarity function between two triplet sets and t_{sim} is defined as,*

$$\begin{aligned} t_{sim}(t_{POS_1}, t_{POS_2}) = & \frac{1}{(2 + \mathbf{1}_{A_1 \cup A_2 \neq null})} \\ & * (V_{sim}(v_1, v_2) * (1 + N_{sim}(N_1, N_2) \\ & * (1 + A_{sim}(A_1, A_2)))) \end{aligned} \tag{4}$$

where $\mathbf{1}_{A_1 \cup A_2 \neq null} = 0$, if $A_1 \cup A_2 = null$, 1 otherwise.

Calculating triplet similarity includes finding semantic similarity between $action$, $object$ and $attributes$, where $attributes$ set can be null but others can't be null. We have already discussed in section 3.1 that $action$ are actually nothing but *verbs*, $object$ are nothing but *nouns* and $attributes$ are nothing but *adjectives*. So, calculating similarity between $action$, $object$ and $attributes$ boils down to finding semantic similarity between verbs, corresponding nouns and corresponding adjectives. A triplet t_{POS} consists of exactly one $action$ or one *verb*,one $object$ or a set of *nouns* (in case of compound noun), and a set of $attributes$ or a set of adjectives (in case of multiple adjectives). Calculating semantic similarity between two *verbs* is straight forward using $sim_{sem_{w_1,w_2}}$ discussed in section 3.2 and as described in Definition 1. On the other hand, calculating semantic similarity between two noun sets or two adjective sets in (Defitnition 2, Definition 3) is a bit tricky. Both follows the same rule. So, we will discuss about the noun similarity calculation here. We compute the intersection $N_1 \cap N_2$ between two sets N_1 and N_2. We also compute the set difference between both the sets N_1', N_2' as $N_1 - N_1 \cap N_2$ and $N_2 - N_1 \cap N_2$ respectively. Then we compute the pairwise semantic similarity among elements of N_1' and N_2' using $sim_{sem_{w_1,w_2}}$ function. Next, we take the average of these pair wise semantic similarity and treat it as one entity $sim_{sem_{nonIntersec}}$, where

$$sim_{sem_{nonIntersec}} = \frac{\sum_{i=1}^{n} \sum_{j=1}^{m} sim_{sem_{N'_{1_i},N'_{2_j}}}}{|N_1'| * |N_2'|} \tag{5}$$

The other entity is semantic similarity score for $N_1 \cap N_2$ which is 1. Finally, we compute the weighted average of 1 and $sim_{sem_{nonIntersec}}$ where the weights are $|N_1 \cap N_2|$ and 1. After calculating these individual similarity scores we aggregate them to compute the similarity score between two triplets such that $action$ gets the highest importance, followed by $object$ and $attributes$.

Given two documents D and D' we first compute their corresponding triplet representation $D_{t_{POS}}$ and $D'_{t_{POS}}$. Lets say, $|D_{t_{POS}}| = n$ and $|D'_{t_{POS}}| = m$. Without loss of generality, it can also be stated that $n \geq m$. Then, the similarity matrix Mat_{sim} has been calculated as,

$$\begin{aligned} &Mat_{sim^{i,j}} = t_{sim}(D_{t^i_{POS}}, D'_{t^j_{POS}}) \\ &\forall i, j \in n, m \end{aligned} \tag{6}$$

After Mat_{sim} calculation, it has been provided as the input matrix to $Assign_{Hung.}$ algorithm, which returns a unique $1-1$ mapping $Map_{D,D'}$. Now the final similarity score between two documents $sim_{D,D'}$is calculated using the following equation 7

$$\begin{aligned} &sim_{D,D'} = \frac{\sum_{k=1}^{m} t_{sim}(D_{t^i_{POS}}, D'_{t^j_{POS}})}{n} \\ &\textit{where } Map^k_{D,D'} : D_{t^i_{POS}} -> D'_{t^j_{POS}} \\ &\forall i, j \in n, m \end{aligned} \tag{7}$$

Following algorithm 7 actually describes the procedure to calculate the similarity between two job description documents.

Algorithm 7 Job Description Similarity Calculation Algorithm

```
1: procedure DOCSIMCALCPROC
2:     Input: $D, D'$
3:     Output: $sim_{D,D'}$
4:     $D_{t_{POS}} \leftarrow DocRepParProc(D)$
5:     $D'_{t_{POS}} \leftarrow DocRepParProc(D')$
6:     $ProcTripProc(D_{t_{POS}})$
7:     $ProcTripProc(D'_{t_{POS}})$
8:     calculate $sim_{D,D'}$ by equation 7
9: end procedure
```

3.5 Experimental Setup and Data-sets

We used a Spark cluster with 6 executors each having 8GB of RAM for running our experiments. Apache Spark frame work has been used to incorporate parallelism to carry out the experiments. All the codes have been written in python using pySpark library. Cloudant services have been incorporated as database resource. We also used Stanford Core NLP Parser and Wordnet from NLTK library.

Job description documents from IBM Talent Framework Data sets have been used to carry out all the experiments. All the sentences in the job description documents are grammatically incomplete in the sense that each of them starts with a verb. Subject noun is missing from each sentence, e.g. ”Require analytical skills”. So we add ”You” or ”You are” at the beginning of the each sentence depending upon the form of the verb. If the verb ends with ”ing” we added ”You are”, otherwise we added ”You” to make the sentences grammatically correct. In the cases where verbs end with ”s” (verb meant to be for third person singular number) ”You” has been treated as a name (third person singular number) and thus resolves the grammatical issue. Then these grammatically correct sentences are fed to the Stanford Core NLP Parser for generating dependency tree. As an estimate of the computation time in this setup, the action-object-attribute representation and calculation of job description similarity of 500 cross 500 jobs took 3892.33 secs.

4 Evaluation

For testing our method we do Job Family based evaluation. Since we are using IBM Kenexa talent frameworks, we can utilize its default clubbing of jobs into job families. The general expectation is that jobs within a family (intra) will have higher job description similarity scores than those outside the job family (inter). Let

- $F = \{F_1, F_2, ..., F_n\}$ be the set of all job families in the test set.
- $J_i = \{J_{i,1}, J_{i,2}, ..., J_{i,n_i}$ be the set of all jobs in family F_i.
- $Intra_i$ be the average similarity between all pairs of jobs within F_i
- $Inter_i$ be the average similarity between all pairs (A, B) of jobs such that $A \in F_i$ and $B \in F_j$ for all $j \neq i$
- $R_i = \frac{Intra_i}{Inter_i}$

Then the gross metric of interest to gauge effectiveness of a document similarity computation method is $S = \frac{\sum_i |F_i| \times R_i}{\sum_i |F_i|}$, computed over a common test set. So, higher S value means better performance of the similarity calculation approach.

Since we intend to benchmark our method against existing state of the art methods, we conduct several experiments with varying corpus of training data with $N_1 = 56$, $N_2 = 129$ and $N_3 = 430$ documents used for training. POSDC does not require any training corpus, therefore the corpus varying experiments are valid only for doc2vec and LDA. Note that the test set consisted of 500 randomly chosen jobs out of the 2344 available in IBM Kenexa talent frameworks, so that there is representation from each job family in the selected test set. The test sets selected did not have any of the jobs on which the models were trained, and were selected separately for the 3 experiments.

As is evident by the bar charts and table 1, when a large enough corpus is chosen, LDA gives the best overall performance. Otherwise POSDC performs better. When we looked at individual job families, neither LDA nor POSDC completely dominates the other. Doc2vec seems to be consistently inferior to both LDA and POSDC irrespective of the corpus size.

	N_1	N_2	N_3
POSDC	1.60	1.65	1.59
LDA	1.36	1.37	1.65
DOC2VEC	0.968	0.996	1.01

Table 1: Comparison of S value Across Methods

Total number of job families in IBM Kenexa Talent Frameworks is more than 100. But for the sake of brevity and clarity, we show the bar charts for ten largest job families (in terms of number of jobs included) for all three training corpus sizes.

The comparison of R_i values for ten of the biggest job families corresponding to N_1, N_2 and N_3 can be seen in figures 4, 5 and 6 respectively.

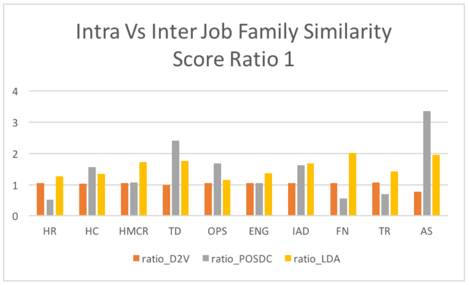

Fig. 4: 10 largest Job Families' R_i Values For N_1

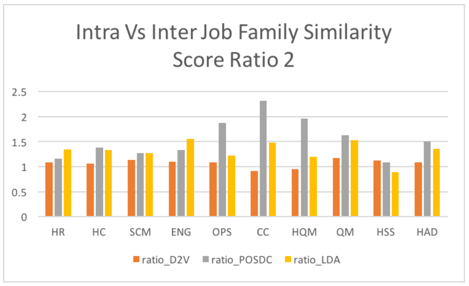

Fig. 5: 10 largest Job Families' R_i Values For N_2

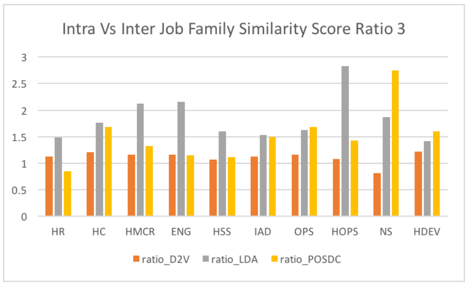

Fig. 6: 10 largest Job Families' R_i Values For N_3

5 Conclusion and Future Work

The core novelty of POSDC is that unlike LDA or doc2vec, it doesn't require any prior training on large corpus. It uses the inherent semantics of job descriptions to find the similarity using available dictionary. As can be seen in our results, it is consistently superior to doc2vec, and even superior to LDA based method when the corpus available to train is smaller. The future work in this direction would be to define similar paradigm(s) for other/generic documents.

In the current approach, we have assumed that there is no duplication or alternate description of the same action-object-attribute triplet within a document. If that is not the case, then effectively the same action-object-attribute triplet in one job may get matched to different ones in another job. This can be overcome by first matching a job description with itself, and removing pairs of action-object-attribute triplets that match with a score above a threshold.

Another possible future direction could be more domain specific rather than being problem specific. Since our motivation to tackle this problem is to find jobs that are similar, we could combine similarity between job title ([8]) and POSDC to improve upon RISE ([19]).

References

1. Apache Spark, accessed: 2017-05-23

2. Apache Spark Wiki, accessed: 2017-05-23
3. NLTK, accessed: 2017-05-23
4. Watson Natural Language Understanding Service, accessed: 2017-01-05
5. Wordnet NLTK, accessed: 2017-05-23
6. Wordnet Synsets, accessed: 2017-05-23
7. Wordnet WUP$similarity, accessed : 2017 - 05 - 23$
8. Ahuja, S., Mondal, J., Singh, S.S., George, D.G.: Similarity computation exploiting the semantic and syntactic inherent structure among job titles. In: Maximilien M., Vallecillo A., Wang J., Oriol M. (eds) Service-Oriented Computing. ICSOC 2017. Lecture Notes in Computer Science, vol 10601. Springer, Cham. pp. 3–18 (2017)
9. Aizawa, A.: An information-theoretic perspective of tfidf measures. Information Processing Management 39(1), 45 – 65 (2003), `http://www.sciencedirect.com/science/article/pii/S0306457302000213`
10. Friedland, L., Allan, J.: Joke retrieval: recognizing the same joke told differently. In: Proceedings of the 17th ACM conference on Information and knowledge management. pp. 883–892. ACM (2008)
11. Gabrilovich, E., Markovitch, S.: Computing semantic relatedness using wikipedia-based explicit semantic analysis. In: IJcAI. vol. 7, pp. 1606–1611 (2007)
12. Hu, J., Fang, L., Cao, Y., Zeng, H.J., Li, H., Yang, Q., Chen, Z.: Enhancing text clustering by leveraging wikipedia semantics. In: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval. pp. 179–186. ACM (2008)
13. Huang, A.: Similarity measures for text document clustering. In: Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand. pp. 49–56 (2008)
14. Kuhn, H.W.: The hungarian method for the assignment problem. Naval research logistics quarterly 2(1-2), 83–97 (1955)
15. Le, Q.V., Mikolov, T.: Distributed representations of sentences and documents. CoRR abs/1405.4053 (2014), `http://arxiv.org/abs/1405.4053`
16. Manning, C.D., Schtze, H.: Foundations of Statistical Natural Language Processing (1999)
17. Matveeva, I., Levow, G.A., Farahat, A., Royer, C.: Generalized latent semantic analysis for term representation. In: Proc. of RANLP (2005)
18. Pak, A.N., Chung, C.W.: A wikipedia matching approach to contextual advertising. World Wide Web 13(3), 251–274 (2010)
19. Pimplikar, R.R., Kannan, K., Mondal, A., Mondal, J., Saxena, S., Parija, G., Devulapalli, C.: Rise: Resolution of identity through similarity establishment on unstructured job descriptions. In: Maximilien M., Vallecillo A., Wang J., Oriol M. (eds) Service-Oriented Computing. ICSOC 2017. Lecture Notes in Computer Science, vol 10601. Springer, Cham. pp. 19–36 (2017)