Similarity Computation Exploiting the Semantic and Syntactic Inherent Structure Among Job Titles

Sarthak Ahuja[1](✉), Joydeep Mondal[1], Sudhanshu Shekhar Singh[1], and David Glenn George[2]

[1] IBM Research Lab, New Delhi, India
sarahuja@in.ibm.com
[2] IBM Talent Management Solutions, Portsmouth, UK

Abstract. Solutions providing hiring analytics involve mapping company provided job descriptions to a standard job framework, thereby requiring computation of a similarity score between two jobs. Most systems doing so apply document similarity computation methods to all pairs of provided job descriptions. This approach can be computationally expensive and adversely impacted by the quality of the job descriptions which often include information not relevant to the job or candidate qualifications. We propose a method to narrow down pairs of job descriptions to be compared by comparing job titles first. The observation that each job title can be decomposed into three components, domain, function and attribute, forms the basis of our method. Our proposal focuses on training the machine learning models to identify these three components of any given job title. Next we do a semantic match between the three identified components, and use those match scores to create a composite similarity score between any two pair of job titles. The elegance of this solution lies in the fact that job titles are the most concise definition of the job and the resulting matches can easily be verified by human experts. Our results show that the approach provides extremely reliable results.

1 Introduction

The problem of finding similarity between a pair of documents lays groundwork for the problem of clustering similar documents together. Most of the initial research in this domain was based on standard document similarity computing methods such as tf- IDF, LSA, LDA etc. In certain specific scenarios, such as job descriptions in recruitment domain, the documents have very precise titles as well. Doing a preliminary match between pairs of titles can greatly reduce the effort required to eventually compare documents for similarity.

In our work for the recruitment analytics domain, and the recent developments therein, one problem that we have faced time and again is that of identifying which job requisitions are similar. This problem arises in two contexts:

M. Maximilien et al. (Eds.): ICSOC 2017, LNCS 10601, pp. 3–18, 2017.
https://doi.org/10.1007/978-3-319-69035-3_1

1. Machine Learning models to identify good candidates: A typical application of machine learning in hiring is to learn success models for various jobs. To be meaningful, the models need to be learnt at a job group level instead of job level, so that sufficient data can be obtained for training the models.
2. Candidates' previous jobs need to be matched with the opening they apply to (or to the openings that can be recommended to them). This requires comparing an applicant's job previous jobs to the job openings available in the applicant tracking system.

Job requisitions typically consist of several well-defined components: required skills, years of experience, job title, job location and a job description. Since required skills, years of experience and job location are well-defined structured fields, the complexity comes in matching job title and job descriptions across jobs. In this paper, we present a Parts Of Title (POT) tagging and wordnet based matching technique to create a match score for two job titles.

Our work here is based on the basic premise that any given job title can be broken into three components Attribute, Function and Domain. Attribute typically denotes some sense of hierarchy (Senior, Junior, lead etc.), function denotes functionality (Manager, supervisor, director etc.), while domain is about the core job area. For example, a senior software engineer is a software (domain) engineer (function) at senior level. A senior electrical engineer is an electrical (domain) engineer at senior level. Although the two job titles have two out of three common words, they are obviously not the same jobs. A software engineer or even a junior software developer is a much closer job to senior software engineer than senior electrical engineer.

These three components might contains multiple words, or can also be null depending upon the context. Our work here describes a method that utilizes semantic match scoring between the three components, and combining those scores using logical domain insights to create a match score between a pair of job titles.

This paper is organized as follows. The next section describes some literature on title or phrase similarity/clustering. Section 3 describes the complete pipeline and methodology. Section 4 explains the methodology, Sect. 5 presents our evaluation and results. Section 6 concludes and discusses some future work.

2 Literature Survey

In typical text document classification and clustering tasks, the definition of a distance or similarity measure is essential. The most common methods employ keyword matching techniques. Methods such TFIDF [2] 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. Researchers have also extended this to N-gram based models, 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. As could be seen from the senior software engineer versus senior electrical engineer example in the previous section, these traditional document similarity methods do not work so well when matching short snippets of text, such as job titles. There are methods involving web based kernel function [10], wherein results of web search query are used to provide context to the short terms being compared. This paper defines a semantic similarity kernel function based on query search results, mathematically analyze some of its properties (similarity score going to 1 for similar queries as the query results sets cover all the relevant documents; the kernel measuring mean topical distance between the queries), and provide examples of its efficacy.

An alternative classification system [3] employs lazy learning from labeled phrases, and present a strong argument in favor of their method when the property of near sufficiency (most of information on document labels is captured in phrases) holds. They also reveal that in all practical cases from small-scale to very large-scale manual labeling of phrases is feasible as natural language constrains the number of common phrases composed of a vocabulary to grow linearly with the size of the vocabulary. Variants of phrase based classification have been studied in Information retrieval [9] and it has the advantage of ease of explainability.

Rich document representations and similarity measures are also an option for job title classification [11]. Semantic enrichment strategies replace the bag of words (BOW) representation that is more popular text classification as it is less adept at handling synonyms, polysemous words and multi word expressions. A machine learning- based semi-supervised job title classification system [4], leveraging a varied collection of classification and clustering tools and techniques, can be used to tackle the challenges of designing a scalable classification system for a large taxonomy of job categories.

A technical report [8] on learning compound noun semantics discusses an annotation scheme for compound nouns to derive compound relations (BE, HAVE, IN, ABOUT, ACTOR, (INST(rument))), and uses this annotation scheme to meaningfully compare compound nouns. This report inspired us to create learners to tag the three components of a job descriptions for a meaningful comparison between them. The final paper [7] combines pattern-based extraction and bootstrapping for noun compounds interpretation. They use a two-step algorithm to jointly harvest NCs and patterns (verbs and prepositions) that interpret them for a given abstract relation.

3 Methodology

Our proposed approach to generate a similarity score between two titles T and T' is illustrated in Figs. 1 and 2. In the former we illustrate the steps for setting up the system and in the latter we depict the steps that take place in the deployed system.

We start with a labeled dataset of titles, where each constituent keyword has been labeled with a particular context in which it occurs. For our training phase as described in Fig. 1 we use 90% of this data, while the remaining 10%

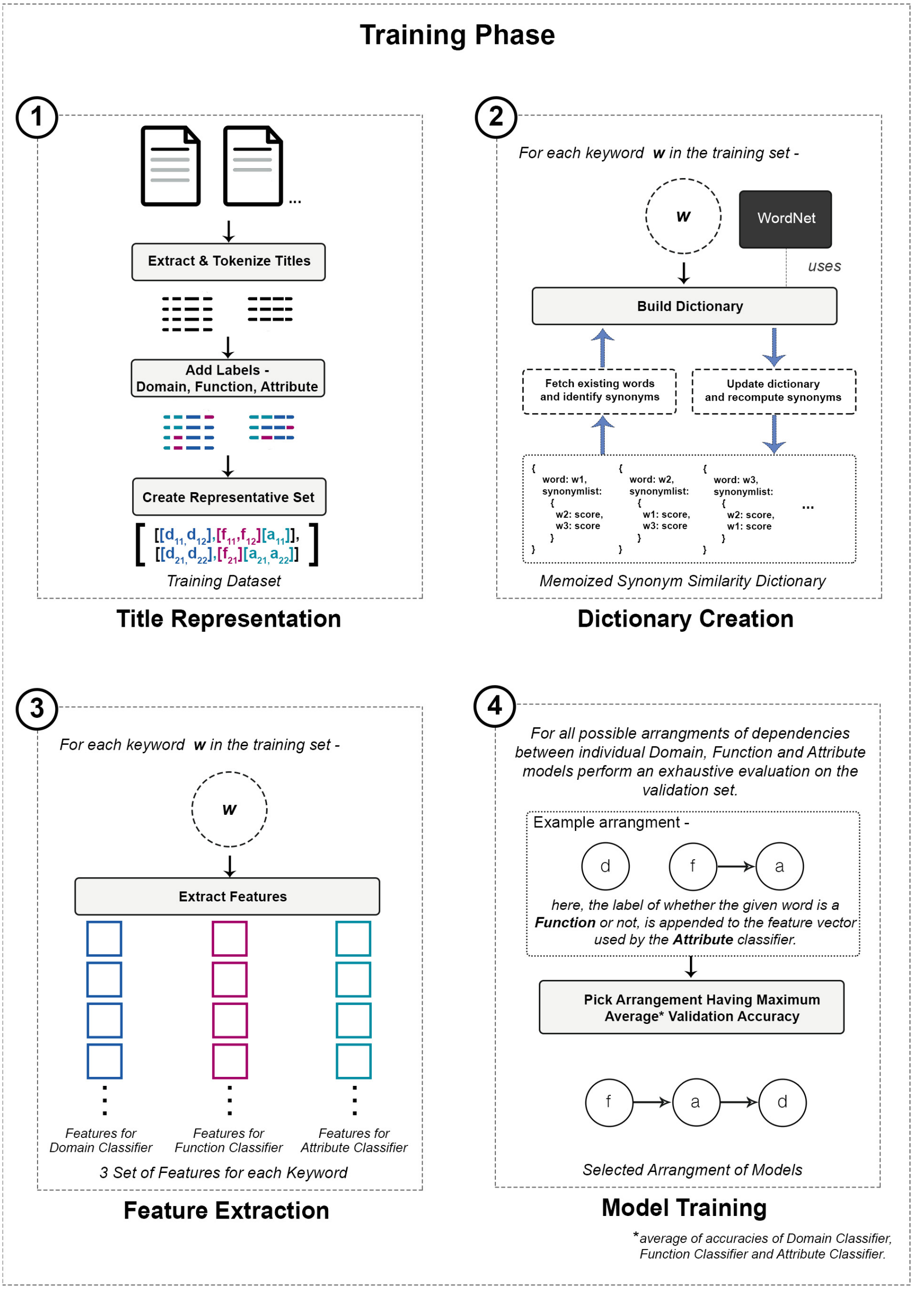

Training Phase
1
Extract & Tokenize Titles
Add Labels -
Domain, Function, Attribute
Create Representative Set
[[d11,d12],[f11,f12][a11]],
[[d21,d22],[f21][a21,a22]]
Training Dataset
Title Representation
2
For each keyword w in the training set -
w
WordNet
uses
Build Dictionary
Fetch existing words and 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
For each keyword w in the training set -
w
Extract Features
Features for Domain Classifier
Features for Function Classifier
Features for Attribute Classifier
3 Set of Features for each Keyword
Feature Extraction
4
For all possible arrangments of dependencies between individual Domain, Function and Attribute models perform an exhaustive evaluation on the validation set.
Example arrangment -
d
f
a
here, the label of whether the given word is a Function or not, is appended to the feature vector used by the Attribute classifier.
Pick Arrangement Having Maximum Average* Validation Accuracy
f
a
d
Selected Arrangment of Models
Model Training
*average of accuracies of Domain Classifier, Function Classifier and Attribute Classifier.

Fig. 1. Training Phase. In (1) titles for all documents available in the training set are extracted and their keywords are labeled with the ground truth context. Next in (2) for each of these keywords a dictionary is built using WordNet to maintain top synonyms and corresponding similarity scores on a cloud database for faster computation. Later in (3) feature vectors for each keyword are extracted and passed onto the Model Training phase. Finally, all permutations of arrangement of models are trained on the dataset and the arrangement with the highest validation accuracy is stored for deployment and evaluation on the testing dataset.

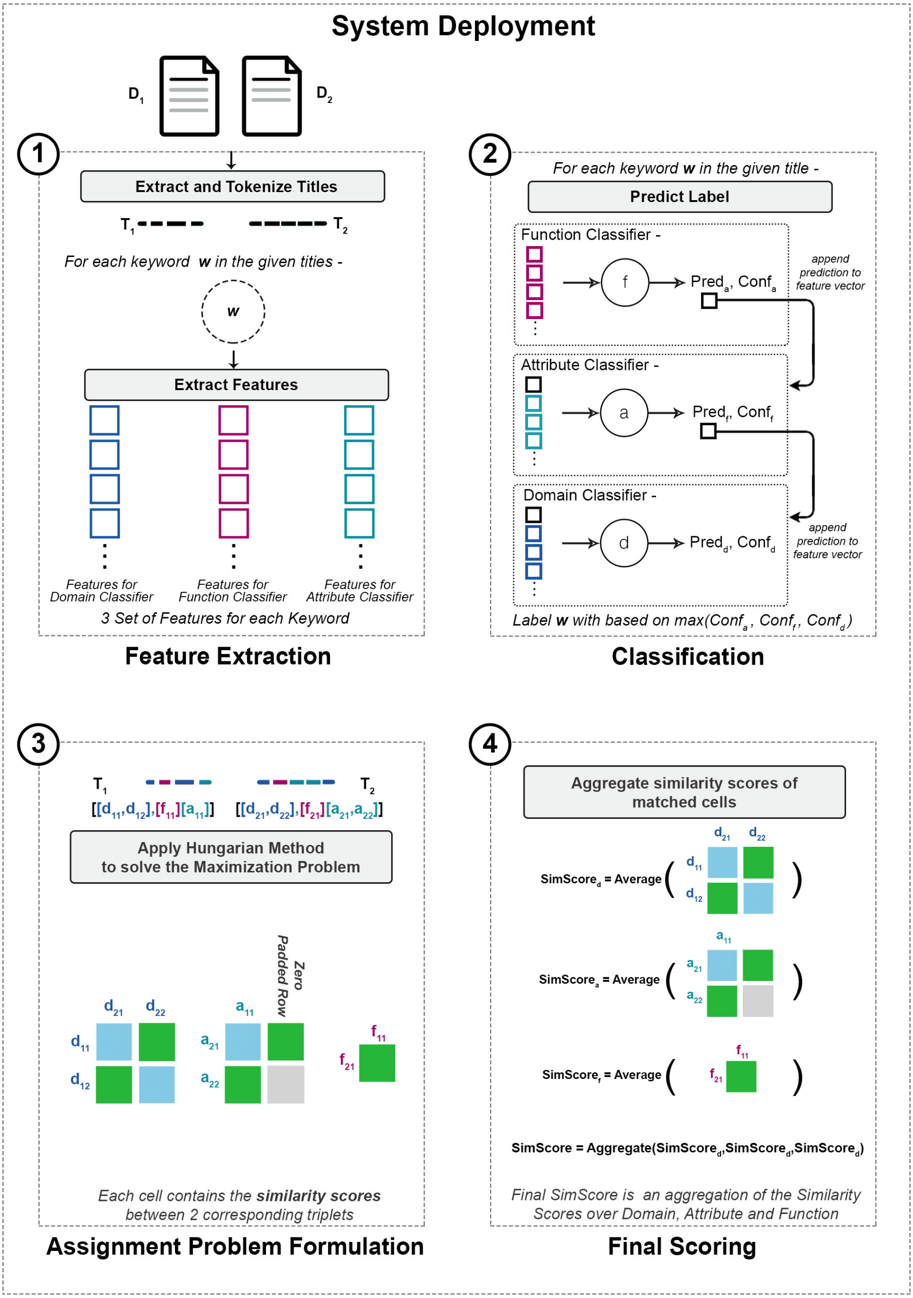

System Deployment
D1
D2
1
Extract and Tokenize Titles
T1
T2
For each keyword w in the given tities -
w
Extract Features
Features for Domain Classifier
Features for Function Classifier
Features for Attribute Classifier
3 Set of Features for each Keyword
Feature Extraction
2
For each keyword w in the given title -
Predict Label
Function Classifier -
f
Pred_a, Conf_a
append prediction to feature vector
Attribute Classifier -
a
Pred_f, Conf_f
Domain Classifier -
d
Pred_d, Conf_d
append prediction to feature vector
Label w with based on max(Conf_a, Conf_f, Conf_d)
Classification
3
T1
T2
[[d11,d12],[f11][a11]]
[[d21,d22],[f21][a21,a22]]
Apply Hungarian Method to solve the Maximization Problem
Zero Padded Row
d21 d22
d11
d12
a11
a21
a22
f11
f21
Each cell contains the similarity scores between 2 corresponding triplets
Assignment Problem Formulation
4
Aggregate similarity scores of matched cells
SimScore_d = Average ( )
SimScore_a = Average ( )
SimScore_f = Average ( )
SimScore = Aggregate(SimScore_d,SimScore_d,SimScore_d)
Final SimScore is an aggregation of the Similarity Scores over Domain, Attribute and Function
Final Scoring

Fig. 2. System Deployment. In (1) incoming documents have their titles extracted and feature vectors for their constituent keywords created. Moving to (2), based on the arrangement of the models, the keywords are labeled with their context. Next in (3), for each context the assignment problem is formulated and solved to identify one-to-one matches for keywords within a particular context. Here, these matches are depicted in green. Finally in (4) the similarity scores over each context are computed using the dictionary created over the training phase and later aggregated to generate an overall similarity score.

is used for testing. Our goal is to create classification models which are able to label keywords of a jobTitle with their context. Once we have a system capable of doing this type of labeling, we move on to using these contextual labels to calculate similarity scores over these contexts and finally aggregate these scores to generate an overall similarity score.

In this section we explain these steps alongside their system implementation in greater detail.

3.1 Title Representation

Each title T_i is treated as a set of sequenced keywords $\mathbf{K}(k_{i1}, k_{i2}, ...k_{in})$. When comparing two titles, it is imperative that the similarity between two keywords that have the same context contributes towards the final score and not the similarity among two keywords with a different context. In the specific case of **Job Titles**, we hypothesize that a constituent keyword can have three contexts i.e. each title can be represented as a collection of keywords organized into three sets *domain, function and attribute* as depicted below

Assistant Software Engineer can be represented as -
Domain - [*Software*]
Function - [*Engineer*]
Attribute - [*Assistant*]
similarly *Senior Call Center Consultant* can be represented as -
Domain - [*Call, Center*]
Function - [*Consultant*]
Attribute - [*Senior*]

Here domain symbolizes words which are representative of the field/industry of work, function symbolizes the line/position of work, while attribute corresponds to any supporting characteristic of the function and domain. Given the diversity of job titles that appear on resumes and job databases, any two of the three sets may be empty. Throughout this paper, we will refer to this collection of sets representation of a title T_i as R_i, consisting of R_i^D, R_i^F and R_i^A, with constituent elements being denoted as $[d_{i1}, d_{i2}...d_{ij}]$, $[f_{i1}, f_{i2}...f_{ik}]$ and $[a_{i1}, a_{i2}...a_{il}]$ respectively as depicted in Fig. 1.

3.2 Preprocessing

For each keyword in a title we perform basic preprocessing to clean the data:

1. **Lower Case:** All titles are converted to lower case characters and trailing spaces are trimmed off.
2. **Abbreviation Expansion:** We expand common abbreviations such as **sr.** to **senior**, **jr.** to **junior** using a hard coded list of common abbreviations.
3. **Punctuation and Number Pruning:** Punctuation marks like '_', '-', etc. are removed by pruning all non-alphabet characters.

Once the data is cleaned, we move on to constructing the feature vectors on which the classification models will be trained.

3.3 Feature Extraction

In our approach we create 3 separate binary classification models for each of the aforementioned contexts - *domain, function* and *attribute*. For each title T_i, the three classifiers label each constituent keyword K_i. The label with the highest confidence for the positive class is taken as the label for the keyword. This section elaborates on the features extracted features and the intuition behind choosing them.

Position. The position of a keyword comes out to be an important parameter in determining it's context. For example, in the job titles - ***Assistant*** *Software Engineer* and ***Assistant*** *Manager*, the word ***Assistant*** acts as an *attribute* when it appears in the beginning of the word, while in the job title *Lab* ***Assistant***, it acts as a *function*.

It is important to understand at this juncture, that the same word can appear in different contexts depending on it's position, hence making **position** an important feature. In our approach we define 3 boolean features - *position_begin* which denotes whether the keyword appears at the beginning of the title, *position_end* which denotes whether the keyword appears at the end of the title and *position_between* which denotes whether the keyword appears in the middle of the title.

Suffix. We listed a set of common keywords found in Domains, Functions and Attributes and noticed some patterns with the suffixes. For example:

Words labeled as Domain: Ophthalmologi**st**, Denti**st**, Psychiatri**st**, etc.
Words labeled as Function: Engine**er**, Doct**or**, Manag**er**, etc.
Words labeled as Attribute: Juni**or**, Seni**or**, etc.

Our first observation was the pattern of **-ist** suffix for the *Domain* words. This observation is consistent with the definition of **-ist** being forming nouns denoting a member of a profession or business activity [1]. However for the *Functions* and *Attributes* the suffix usage is tightly correlated with its context. As example, **-or** can be used to denote a person or thing performing the action of a verb [1]. It also can be used to form comparative adjectives.

Based on these observations, in our approach we define two suffix lists for the three contexts and for each define the feature as a boolean on whether the suffix of the keyword is present in the concerned list. For example, the suffix list defined for the *function* and *attribute* classifier was defined as [**'or', 'er', 'ors', 'ers', 'ar', 'ars'**] and hence, the feature vector for the keyword ***Manager*** marks the **suffix** feature as **1** for the *function* classifier. Similar list is created for the *domain* classifier as [**'ist', 'ists'**].

It should be noted that we limit the size of these lists, to only include the most common observations using basic knowledge of English grammar and vocabulary, and do not mine for any suffix patterns explicitly.

Keyword POS Tag. Next we append to the feature vector a Part-Of-Speech Tag for the corresponding keyword using the Stanford POS Tagger. The tagger can tag the keyword with 1 of the 36 labels.

An important fact we note is that the POS Tag of the keyword is at times different from the POS Tag of other versions of the same keyword. For example, **manager** is tagged as **noun** while **manage** is tagged as **verb**. Considering that ideally both should be labeled with the same context, besides the original POS Tag, we reduce the keyword to it's root and add the POS Tag of the root word as well. Hence, for this feature we append two values to the feature vector, POS Tag of the keyword and POS Tag of the root of the keyword.

In our implementation, we refer to vocabulary.com to extract the root of a keyword.

3.4 Feature Vector Construction

The features explained in the previous subsection are combined to form a feature vector. If we assume that all three classification models are independent of each other we get each keyword being depicted in the form of 3 feature vectors each of length 7. But we observe that as humans we do not label all the keywords independently. Once we have obtained the prediction for one or two of the context labels for a keyword in the title, we get contextual information that increases our confidence to label it for a different context. For example, for **Assistant Software Developer**, once we identify that **Software** is not the *function* word, we get more confident about labeling it as *domain*. Given our problem, we assert that this *ruling out* step will play a crucial role in improving the classifier accuracies. Keeping this in mind, based on the dependencies amongst classifiers, the feature vector for a classifier may also have the predicted label of a previously checked classifier. In the previously mentioned example, if we identify that the *domain* classifier is dependent on the *function* classifier, we will see the feature vector being fed into the former be of length (7 + 1), after appending the prediction of the *function* classifier to the feature vector. In the training stage, instead of actual model predictions we use the ground truth value, while in the testing (system deployment) stage, we use the actual prediction of the trained models. Figure 3 displays the possible arrangements of the models.

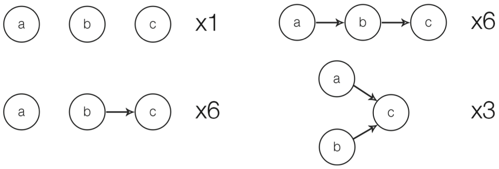


Fig. 3. All possible dependency relationships that can exist between three models. Here *a*, *b* and *c* can represent any of *Domain*, *Function* and *Attribute*, hence generating a total of 16 possible arrangements.

In our case we perform an evaluation on all 16 possible arrangements, explained in Sect. 5 and choose the best arrangement based on validation accuracy. This selected arrangement of models represents the identified dependencies among the context classifiers, and is deployed in the system and used on the testing dataset for our final evaluation.

The next section explains the training procedure for each individual model.

3.5 Model Creation

For each context, we train models using SPSS's autoclassifier module which trains a bunch of classifiers on the data - neural net, C5, Logistic Regression, CHAID, Quest, C&R, Bayesian Network, Decision List. We use 90% of our labeled dataset for this model creation. We apply a 10-fold validation on this dataset and select the model with the highest average validation accuracy as depicted in Table 1. Our selected arrangement of models is depicted in Figs. 1 and 2. The best classifier for *Domain* and *Function* context comes out to be C5, while for *Attribute* it comes out to be a neural net.

3.6 Assignment Problem Formulation

As depicted in Fig. 2, the two documents, D_1 and D_2, are represented by their tokenized titles T_1 and T_2. After our selected arrangement of the three context classifiers labels these keywords as either *Domain*,*Function* or *Attribute* they are then represented as two sets of triplets, $S_1([R_1^D,\ R_1^F,\ R_1^A])$ and $S_2([R_2^D,\ R_2^F,\ R_2^A])$, each with elements $[[d_{11}, d_{12}...d_{1j}],\ [f_{11}, f_{12}...f_{1k}],\ [a_{11}, a_{12}...a_{1l}]]$ and $[[d_{21}, d_{22}...d_{2j}],\ [f_{21}, f_{22}...f_{2k}],\ [a_{21}, a_{22}...a_{2l}]]$ respectively. Similarity score between these two documents can be interpreted as the similarity score between these two sets of triplets. The similarity function is explained in detail in the next subsection, and is denoted by $\mathbf{F}$ for now. As mentioned earlier, when comparing two titles, it is imperative that the similarity between two keywords that have the same context contributes towards the final score. There is no relevance in the similarity among two keywords with a different context. To calculate the similarity score between two sets, we calculate the similarity score independently amongst the three contexts and aggregate them to get a final similarity score. This aggregator function is explained in detail in the next subsection, and is denoted by $\mathbf{A}$ for now. To find the similarity between two sets of the same context, a naive approach would be to calculate the similarity score between each pair of elements from two sets (example, R_1^D and R_2^D), 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 keywords in the Job Title, hence, the representative set for a document too will not have synonymous elements. This assumption motivates a one-to-one mapping among the two sets being compared for similarity.

To find this mapping among the aforementioned two sets, we formulate the problem as an assignment problem. 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 one of the context sets of the S_1 and S_2 can be formulated in a similar way - given **F** as the cost of assignment function among each pair of elements in the two sets, the task is to find an optimal one-to-one assignment among the elements that maximizes the aggregate similarity score.

Next, we use the *Hungarian Method* 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. Given ours is an imbalanced assignment problem, the 2 sets with m and n keywords 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.

This assignment task is done independently for each of the three contexts. Post this assignment, the following subsection defines the similarity and aggregation functions.

3.7 Final Score Computation

In this subsection we define the previously mention similarity(**F**) and aggregator(**A**) functions.

Similarity Function. We use WordNet as the basis of our similarity function to compute a semantic similarity score between two keywords. Any other methods such as Word2Vec, etc. which provide a semantic similarity score between two words could be possible alternatives to WordNet. 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). 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 1. On obtaining the sim_{sem} score between matches provided by the solution of the assignment problem, the final Similarity Score for a context is taken to be the average of all the sim_{sem} scores. We denote the Similarity Score for each context - *Domain*, *Function* and *Attribute* as $SimScore_d$, $SimScore_f$ and $SimScore_a$ respectively.

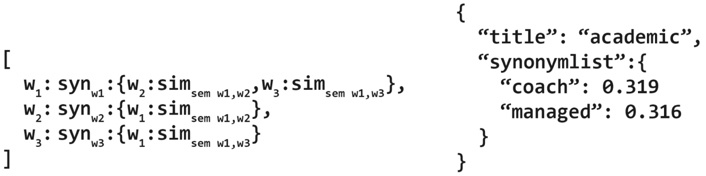
To make the computation of this Similarity Score faster we employed some optimizations in the scoring process. After obtaining the tokenized keywords from a Job Title we used *memoization* and *precomputation* techniques to build a dictionary *Dict* of keywords. The structure of the dictionary is depicted as in Fig. 4. 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 Fig. 4. syn_w for a w consists of only the words which exist in *Dict* and cross a threshold of semantic similarity score (sim_{sem}).

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 keyword, we precompute the semantic similarity scores among the words and store them in a database.

Algorithm 1. SimSem Function

```
procedure SimSemFunction
    Input: $w_1, w_2$
    Output: $sim_{sem_{w_1,w_2}}$
    $sim_{sem_{w_1,w_2}} \leftarrow 0$
    $synSets_{w_1} \leftarrow null$
    $synSets_{w_2} \leftarrow null$
    $synSets_{w_1} \leftarrow$ synsets from Wordnet for $w_1$
    $synSets_{w_2} \leftarrow$ synsets from Wordnet for $w_2$
    $div \leftarrow 0$
    for each $synSet_{w_1}$ of $synSets_{w_1}$ do
        for each $synSet_{w_2}$ of $synSets_{w_2}$ do
            $wup_{score} \leftarrow$ wup similarity between $synSet_{w_1}$ & $synSet_{w_2}$
            if $wup_{score}$ is not null then
                $sim_{sem_{w_1,w_2}} \leftarrow sim_{sem_{w_1,w_2}} + wup_{score}$
                $div \leftarrow div + 1$
            end if
        end for
    end for
    $sim_{sem_{w_1,w_2}} \leftarrow \frac{sim_{sem_{w_1,w_2}}}{div}$
end procedure
```

Aggregator Function. The aggregator function is meant to collate the similarity scores generated among matches provided by the solution of the assignment problem. The definition of this function is described in Algorithm 2. The equation is basically a weighted average of the three context similarity scores where more weight is given to the *Domain* similarity, *Function* similarity and



[

w_1: syn_{w1}:{w_2:$sim_{sem\ w1,w2}$,w_3:$sim_{sem\ w1,w3}$},

w_2: syn_{w2}:{w_1:$sim_{sem\ w1,w2}$},

w_3: syn_{w3}:{w_1:$sim_{sem\ w1,w3}$}

]

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

(a) Dictionary Structure for Keyword (b) JSON Structure for saving on Cloudant

Fig. 4. (a) Dictionary Structure for Keyword (b) JSON Structure for saving on Cloudant

then *Attribute* similarity, in that order. Special care is taken in the averaging process so that if both sets of a particular context turn out to be empty, they are not included in the normalizing denominator.

Algorithm 2. Aggregator Function

```
procedure AggregatorFunction
    Input: $SimScore_d, SimScore_f, SimScore_a$
    Output: $SimScore$
    $SimScore = \frac{1}{(\mathbf{1}_{SimScore_a \neq 0} + \mathbf{1}_{SimScore_f \neq 0} + \mathbf{1}_{SimScore_d \neq 0})} * (SimScore_d * (1 + SimScore_f * (1 + SimScore_a)))$
end procedure
```

4 Experimental Setup and Dataset

We used a Spark cluster with 6 executors each having 8GB of RAM for running our experiments. Apache Spark framework has been used to incorporate parallelization to carry out the experiments. All the codes have been written in python. We used PySpark library to include Apache Spark environment into our system. Cloudant services have been incorporated as database resource. We also used Standford 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. Our training and validation set consists of 4471 job titles, leading up to 16180 keywords. For our test set, we have 421 job titles corresponding to 71 different job families. Other details of the dataset can't be revealed here due to confidentiality issues.

5 Evaluation

As part of our evaluation we first present the results of the training phase in Table 1. Here for all 16 possible arrangements of model dependencies we calculate the training and validation accuracy. We observe that the maximum validation accuracy of

Table 1. Selecting best arrangement of models

Arrangement	Domain		Function		Attribute		Average	
	Training Accuracy	Validation Accuracy	Training Accuracy	Validation Accuracy	Training Accuracy	Validation Accuracy	Training Accuracy	Validation Accuracy
$d \quad f \quad a$	80.48	80.507	88.54	88.77	92.25	92.257	87.093	87.178
$d \quad f \rightarrow a$	80.48	80.507	88.54	88.77	92.77	**92.91**	87.26	87.39
$f \quad d \rightarrow a$	80.48	80.507	88.54	88.77	**92.80**	92.62	87.27	87.30
$a \quad d \rightarrow f$	80.48	80.507	90.96	91.94	92.25	92.257	87.9	87.96
$a \quad f \rightarrow d$	86.62	86.36	88.54	88.77	92.25	92.257	89.14	89.12
$d \quad a \rightarrow f$	80.48	80.507	89.18	89.227	92.25	92.257	87.30	87.33
$f \quad a \rightarrow d$	83.81	83.75	88.54	88.77	92.25	92.257	88.20	88.26
$f \rightarrow d \rightarrow a$	86.62	86.36	88.54	88.77	92.78	92.76	89.31	89.29
$f \rightarrow a \rightarrow d$	**90.71**	**90.59**	88.54	88.77	92.77	**92.91**	90.67	**90.76**
$a \rightarrow f \rightarrow d$	**90.71**	**90.59**	89.18	89.227	92.25	92.257	**90.71**	90.69
$a \rightarrow d \rightarrow f$	83.81	83.75	**92.89**	**92.79**	90.25	90.257	89.65	89.60
$d \rightarrow a \rightarrow f$	80.48	80.507	**92.89**	**92.79**	**92.80**	92.62	88.72	88.64
$d \rightarrow f \rightarrow a$	80.48	80.507	90.96	91.94	92.78	92.76	88.07	88.13
$a \rightarrow f \leftarrow d$	80.48	80.507	**92.89**	**92.79**	92.25	92.257	88.54	88.51
$f \rightarrow a \leftarrow d$	80.48	80.507	88.54	88.77	92.78	92.76	87.27	87.34
$f \rightarrow d \leftarrow a$	**90.71**	**90.59**	88.54	88.77	92.25	92.257	90.50	90.54

90.76% is obtained for a linear relationship among the context classifiers ($f \rightarrow a \rightarrow d$). This arrangement is hence chosen for deployment and testing.

For our chosen model, in the testing phase we observe

1. an accuracy of **78.04%** for the domain classifier
2. an accuracy of **87.01%** for the function classifier.
3. an accuracy of **93.43%** for the attribute classifier.

We did a job family based evaluation to test our method. Since we are using IBM Kenexa talent frameworks, we can utilize its default clustering of jobs into job families. We would expect jobs within a family (intra) to have higher title similarity scores than those outside the job family (inter). The scores for intra vs inter job family titles' similarity were calculated, and averaged for reporting. The comparison of scores for some of the biggest job families can be seen in Fig. 5.

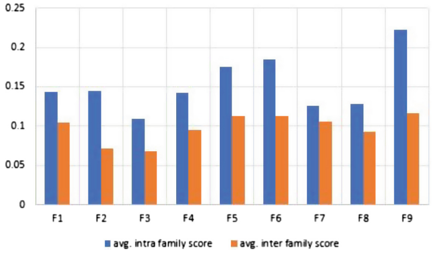


Fig. 5. Inter versus intra job family title comparison scores.

As expected, the scores for inter job family title distances are higher than those for intra job family distances. They are not very high, as there are many different roles and functions even within the same job family. For example, both "talent analyst senior" and "international human resources manager" in the job family "HR", but they are very different from each other.

In some of the smaller job families, the intra scores average was lower (or equal) than the inter scores average. On investigation we found out that those job families had only 2–3 jobs, and they all seemed very different. Where as those jobs seem to have several common domain synonyms with job titles in other families.

We compare our method to another approach of directly using WordNet semantic similarity. For this comparison, we use the ratio of the average inter and intra cluster similarity as our evaluation metric. The lower this ratio is, the better is the cluster quality. For WordNet based semantic similarity we take the average of all WordNet generated similarity scores for all possible pairs of keywords between two given titles. We show the results for the biggest job families in Fig. 6. We observe that our method leads to a consistently better cluster quality compared to the method of simply using WordNet based average score as the similarity metric.

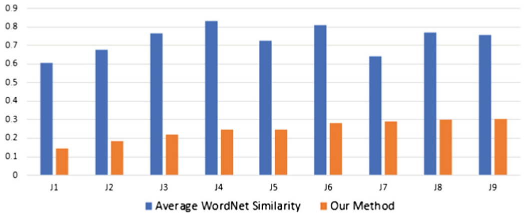


Fig. 6. Comparison based on inter/intra cluster similarity ratio

6 Conclusion and Future Work

In this paper, we described an approach to find similarity between job titles based on the observation that each job title consist of three components - domain, function and attribute. We used classifier models to identify the tokens in a job description as one of the three components. Then we used a hierarchical approach with domain, function and attribute as the levels of hierarchy to find the similarity score between any two jobs.

As we observed via the intra vs inter job title similarity scores, the approach gives fairly accurate results. In some of the smaller job families, the intra scores average were not higher than the inter scores average. The accuracy of overall matching scores depends on the accuracy of classifiers and the engine used to match the three components with each other.

We believe that another way of identifying the three components of a job description could be based on approaches used for finding similarities between compound nouns. That exact approach will not suffice since job titles do not just consist of nouns. Other semantic approaches to identify the three components, or compound noun based approach to find similarity between domain words can improve our results.

References

1. English Dictionary. https://en.oxforddictionaries.com/spelling/nouns-ending-in-er-or-and-ar
2. Aizawa, A.: An information-theoretic perspective of Tf-idf measures. Inf. Process. Manag. **39**, 45–65 (2003)
3. Bekkerman, R., Gavish, M.: High-precision phrase-based document classification on a modern scale. In: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM (2011)
4. Javed, F., Luo, Q., McNair, M., Jacob, F., Zhao, M., Kang, T.S.: Carotene: a job title classification system for the online recruitment domain. In: 2015 IEEE First International Conference on Big Data Computing Service and Applications (BigDataService), pp. 286–293. IEEE (2015)
5. Loper, E., Bird, S.: NLTK: the natural language toolkit. In: Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics, ETMTNLP 2002, Association for Computational Linguistics, Stroudsburg, PA, USA, vol. 1, pp. 63–70 (2002). http://dx.doi.org/10.3115/1118108.1118117
6. Miller, G.A.: WordNet: a lexical database for english. Commun. ACM **38**(11), 39–41 (1995)
7. Nakov, P.I., Hearst, M.A.: Semantic interpretation of noun compounds using verbal and other paraphrases. ACM Trans. Speech Lang. Process. (TSLP) **10**(3), 13 (2013)
8. Ó Séaghdha, D.: Learning compound noun semantics. Technical report, University of Cambridge, Computer Laboratory (2008)
9. Riloff, E., Lehnert, W.: Information extraction as a basis for high-precision text classification. ACM Trans. Inf. Syst. (TOIS) **12**(3), 296–333 (1994)

10. Sahami, M., Heilman, T.D.: A web-based kernel function for measuring the similarity of short text snippets. In: Proceedings of the 15th International Conference on World Wide Web, pp. 377–386. ACM (2006)
11. Zhu, Y., Javed, F., Ozturk, O.: Semantic similarity strategies for job title classification. arXiv preprint arXiv:1609.06268 (2016)