SortingHat: A Deep Matching Framework to Match Labeled Concepts (Demo Paper)

Sumant Kulkarni*
International Institute of Information Technology Bangalore,
26/C, Electronics City, Bangalore India
sumant.k@iiitb.org

Srinath Srinivasa
International Institute of Information Technology Bangalore,
26/C, Electronics City, Bangalore India
sri@iiitb.org

ABSTRACT

We report a framework called *SortingHat* to perform semantic matching between labeled concepts in a partially labeled corpora such as workflow data. We create a labeled term co-occurrence graph as the representative data-structure of the given corpus. The semantic matching between concepts is performed using a variant of random walk algorithm on the term co-occurrence graph. The SortingHat system takes a set of concepts as input and generates a semantically matching set of concepts for them. In this experiment, we use data from bug tracking system of a large enterprise to demonstrate the results.

Keywords: Semantic Matching, Labeled Concept Matching, Deep Matching, Text Mining, Concept Matching

1. INTRODUCTION

Semantic matching can be defined as the identification of semantically related objects of a domain for a given set of input objects in the same domain based on the latent semantics. Some examples of such semantic matching problems and their domains are given in table 1.

Large amounts of textual data like bug tracking system snapshot, banking customer care logs, and tourist review logs for the destinations get generated in the above mentioned domains. We can use these text corpora to solve the kind of problems listed in table 1. Solving these problems help to speed up the processes in many such domains. Hence, semantic matching is considered an important problem in text domain.

Semantic matching in text corpora aims to identify set of semantically related concepts for given set of concepts. Many times the textual data contains additional labeled information, like *types* associated with the data values, which can be used in semantic matching process. There also can exist some data values for which labels do not meaningfully capture the type. We call such a label as *pseudo-type*. Together, this kind of data is called "partially labeled data". For example consider a customer care system in banking domain. Table 2 shows hypothetical data collected over several transactions. This log contains data like complaint id, customer id, resolver, priority, which have *type* information. However, there can be additional data columns like summary of complaint which are *pseudo-type*s.

Sl. No	Semantic Matching Problem	Domain
1	Identification of the resolvers for an issue in software industry	Software
2	Identifying the best customer care executive for the given type of complaints like online banking login failure	Banking
3	Identification of the best tourist destination for given calendar month	Tourism

Table 1: Examples of Semantic Matching Problems.

SortingHat aims to perform semantic matching for such partially labeled data. An example of semantic matching between concepts on the data in table 2 could be – identifying the best resolvers for a complaint related to "credit card" account type which also has "high" priority. Kulkarni et.al [1] proposed a term co-occurrence graph based approach to perform the concept matching. The approach includes creating a co-occurrence graph of labeled and unlabeled terms and running a "cash leaking" random walk on it to generate the matching output concepts. There have been earlier attempts to use random walk to mine different kinds of semantics. In [2, 3], authors propose a "cash leaking" random walk to identify the topical anchor of a set of terms. Yazdani et.al [4] present a random walk based framework for semantic similarity calculation.

The SortingHat implementation is inspired from [1]. We have been using a snapshot of a bug tracking dataset from a big enterprise. The implementation aims at identifying the best resolvers who can take up newly reported bugs. The system also extends its functionality in identifying other

*We thank Paras Mittal and Dipesh Joshi for their help in implementation.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
The 20th International Conference on Management of Data (COMAD), 17th-19th Dec 2014 at Hyderabad, India.
Copyright ©2014 Computer Society of India (CSI).

Comp ID	Cust ID	Resolver	Priority	Account Type	..	Conversation Summary	..
aaa	xxxx	Anand	High	Corporate	..	Faced issues with online access of account information. System was giving error 307. Requested customer to login after 3 hours.	..
bbb	yyyy	Sahana	Medium	Credit Card	..	Communication address needs to be changed. Requested employee to send mail to customer care email id with new address proof.	..

Table 2: Data from a hypothetical banking customer care system.

important aspects of a new bug like component, probable priority and so on.

2. THE METHOD

As discussed earlier, semantic matching in text domain is the process of identifying a set of concepts R for a given set of concepts Q in a given corpus based on the latent semantics. Identifying the probable defaulters from a banking transaction dataset, identifying defective components using the server logs, identifying an application software similar to a given application software based on the description and reviews are example of semantic matching.

This work uses partially labeled data as the corpus. Let C be the set of concepts in the corpus. There are some concepts which are identifiable with labels (or have attached type information). We label the unlabeled concepts with special label "concepts". The set of all unique labels in the corpus is represented as L.

We build a co-occurrence graph using C. In this graph, every labeled concept co-occurs with every other labeled concept in the same context. This co-occurrence graph is formally represented as,

$$G = (C, E, w, L, \gamma) \quad (1)$$

where, $E \subseteq [C]^2$ is pairwise co-occurrences of labeled concepts. $\gamma : C \rightarrow L$ represents a functions which assigns label types to concepts. The function $w : E \rightarrow Z^+$ assigns edge weight $w(t, u)$ which is same as the co-occurrence count, where $t, u \in C$. This edges are undirected. We convert the graph into generatability graph, using the procedure explained in [3].

The SortingHat algorithm takes a set of labeled concepts $Q = \{q_1, q_2, \ldots, q_m\}$ as input and generates semantically matching labeled concepts $R = \{r_1, r_2, \ldots, r_p\}$ as result from the co-occurrence graph G. For each q_i, the algorithm generates a set of neighbors $N(q_i)$. A new set $N(Q^*)$, called as *neighbourhood closure*, is generated by taking union of all these set of neighbours. We generate the *semantic context* $S(Q^*)$ by considering all the edges between all the concepts in $N(Q^*)$. The details of the calculation of generatability, neighborhood closure and semantic context are available in [1].

To identify the semantically matching concepts R for Q, we run "cash leaking" random walk algorithm [2]. We start the random walk by distributing equal cash to each query concept and zero cash to every other concept in $S(Q^*)$. After reaching stationary distribution, we identify concepts with higher cash accumulation and output them as R. These are the concepts which are semantically related to the set of labeled query concepts Q.

3. THE TOOL

The SortingHat tool is developed in Java environment. The noun phrases are extracted using the Apache openNLP [1]. We generate the co-occurrence graph and store it in MySQL [2]. We are also parallely attempting to use Neo4j [3] to store co-occurrence graph. We have developed this tool as a web application. Figure 1 shows the webpage to input the query. The input concepts are entered as $\langle term \rangle : \langle type \rangle$. For example, *performance:concept* represents a concept named *performance* of type *concept*.

The tool runs the "cash leaking" random walk using the input and generates the semantically matching output for it. The output is displayed in the form of a table, where each column represents a type, as shown in figure 2. We have intentionally blanked out some columns from the output screen shot to hide sensitive information. The concepts in the columns are ranked as per their semantic relatedness to the query.

Entity	Description
Project	The name of the project.
Issue	The Issue Id of the issue.
Resolver	The person who resolved the issue.
Resolution	State of the issue after resolution (Not a Bug, Duplicate, Fixed etc.,).
Component	Name of the project component.
Time Taken	Time taken to fix issue.
Issue Priority	Importance of the issue.
Classification	Type of the fix (API, Code etc.,).
Root Cause	Reason for the issue to arise.

Table 3: The different labeled concepts in bug tracking dataset.

As mentioned earlier, we are currently using a bug tracking system snapshot as the partially labeled dataset. We have manually identified 9 types(lables) [1] in the dataset as given in table 3. We similarly identified 3 fields in the

[1]https://opennlp.apache.org/
[2]http://www.mysql.com/
[3]http://www.neo4j.org/

Figure 1: Input Screen of SortingHat Web Application

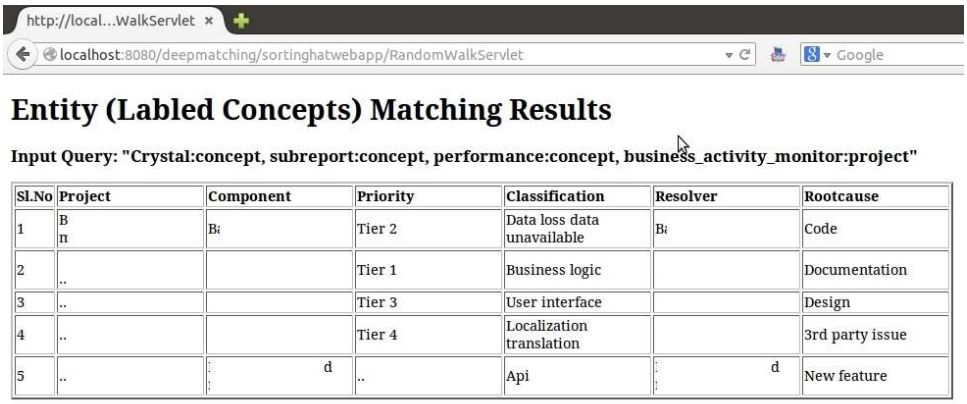

Figure 2: Output Screen of SortingHat Web Application

dataset from where "concepts" can be extracted. They are given in table 4.

The primary semantic matching problem in this data is to identify the best suited *resolver* for a given new issue, for which the information of *Project, Priority, Summary* and *Description* is available. This would help the managers to identify the resolver more efficiently. We can also extend this to identify the *component* of the issue. For both the use cases, the inputs are the values of *Project, Priority*, and *concepts* from unstructured fields - *Summary* and *Description*. The input contains comma separated concepts, where each concept has its type information separated by ":". An example query is given in figure 1. Further, we can also use this framework to identify many other semantic similarities like projects similar to other projects, resolvers similar to other resolvers and so on.

The SortingHat framework can handle many other types of generic semantic matching queries on given textual dataset. It can take any coherent set of concepts and generate semantically matching set of concepts for it. There are domains like insurance claim management, bank customer grievance resolution, and many other where such partially labeled textual data is available. There is a need to identify different labeled concepts like *claim resolver* for the given inputs like *insurance claim*. In such cases, SortingHat can come in handy as a supporting tool.

4. CONCLUSION AND FUTURE WORK

SortingHat is a tool developed to identify semantically related concepts for a given set of concepts. The tool addresses semantic matching in text domain. Many domains generate textual data, and hence this tool can play an important role in semi–automating large number of important tasks in those domains. The future work includes the de-

Column Name	Description
Summary	Short description of the issue.
Description	Detailed description of the issue.
Customer Facing Description for Release Notes	The message passed to the customer to convey the existence of the issue. It describes how the customer can observe the issues.

Table 4: The different fields contributing to concept space in JIRA dataset.

velopment of a functionality to identify similar concepts of same *type* for given input concept. This would help to suggest more number of semantically similar concepts for the input query.

5. REFERENCES

[1] S. Kulkarni, S. Srinivasa, J. N. Khasnabish, K. Nagal, and S. G. Kurdagi. Sortinghat: A framework for deep matching between classes of entities. In *Data Engineering Workshops (ICDEW), 2014 IEEE 30th International Conference on*, pages 90–93. IEEE, 2014.

[2] A. R. Rachakonda and S. Srinivasa. Finding the topical anchors of a context using lexical cooccurrence data. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 1741–1744. ACM, 2009.

[3] A. R. Rachakonda, S. Srinivasa, S. Kulkarni, and M. Srinivasan. A generic framework and methodology for extracting semantics from co-occurrences. *Data & Knowledge Engineering*, 2014.

[4] M. Yazdani and A. Popescu-Belis. A random walk framework to compute textual semantic similarity: a unified model for three benchmark tasks. In *Semantic Computing (ICSC), 2010 IEEE Fourth International Conference on*, pages 424–429. IEEE, 2010.