Matching GitHub developer profiles to job advertisements

Claudia Hauff
Delft University of Technology
the Netherlands
Email: c.hauff@tudelft.nl

Georgios Gousios
Radboud University Nijmegen
the Netherlands
Email: g.gousios@cs.ru.nl

***Abstract*—GitHub is a social coding platform that enables developers to efficiently work on projects, connect with other developers, collaborate and generally “be seen” by the community. This visibility also extends to prospective employers and HR personnel who may use GitHub to learn more about a developer’s skills and interests. We propose a pipeline that automatizes this process and automatically suggests matching job advertisements to developers, based on signals extracting from their activities on GitHub.**

I. Introduction

Today, social coding platforms have become an important tool for developers to showcase their work and become visible in the developer community. GitHub[1] in particular has become an established way for developers to create a portfolio of their work to be considered during the hiring process by potential employers [1]. In order to find potential employers, developers search for job openings in various online job portals and compare their desires, experiences and activities with the described position. This is a cumbersome process as many job advertisements are lengthy, mentioning a plethora of programming languages, libraries and techniques that the perfect candidate should be familiar with. Moreover, each of these items is usually conditioned on the number of years of experience or the level of expertise and may fall into the “required” or “preferred” skill category. Over the years, job advertisements have asked for a larger number of skills from prospective employees. This has led to a situation where a developer matching half of the described requirements may actually be a very well qualified candidate for the advertised position. In such cases having insights into how well other potential candidates fit the position may help the developer to judge whether to apply or not. Another complicating factor is the fact that job advertisements’ writing style may be influenced by the numerous people involved in the creation of a job profile (managers, developers, HR personnel, etc.). Here a “semantic gap” may exist between search terms a developer is using to find suitable advertisements in job portals and the terms that actually appear in an advertisement.

Similarly, judging the qualification of an applicant based on his or her GitHub profile is equally challenging [2]. GitHub provides several user-based summary statistics such as *Contributions in the last year*, *Number of forked projects*, *Number of followers*, however, the usefulness of this information is very limited, as it does not offer immediate insights into the developer’s programming abilities, the particular languages the developer is regularly using or the type of development toolchain the developer is using. Marlow et al. [3] investigated how more detailed publicly accessible signals about a developer’s activites on GitHub are used by employers in the recruitment process. In an interview-based study with several IT employers (active in the open-source community) they identified four main insights that employers can *reliably* gain from a study of developer GitHub profiles: (1) Shared open source values, (2) Community acceptance of work & contribution quality, (3) Project management skills, and (4) Passion for coding. Though again, the limiting factor in this setup is the time required to manually inspect each developer’s profile.

Business-oriented social networks such as LinkedIn[2] are using recommender engines to *push* job advertisements to its users (in addition to the traditional *pull*-based model where users are actively searching among the available advertisements). Recommender algorithms determine the *similarity* between pairs of user and advertisement profiles and recommend the job to the user if the similarity is high. While this process moves the burden of determining the degree of matching away from the user, it is limited in its abilities due to the lack of detailed user profile data as statements such as “Experienced Java developer” or “Embedded Software Engineer” contain relatively little information.

We conclude that considering the vast amounts of job advertisements in the IT sector (as well as the large number of developers), finding a job advertisement that is a good match with one’s own abilities and desires is currently an inefficient process and likewise, determining how well an applicant from a group of applicants matches the position based on available GitHub user data is cumbersome and time-consuming. At the same time though, GitHub user data provides detailed insights that are not possible to be gained from other social Web sources.

In this paper we propose a pipeline that *automatically* mines GitHub user profiles and job advertisements for relevant information. We employ an approach that “translates” both the

[1]https://github.com/

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

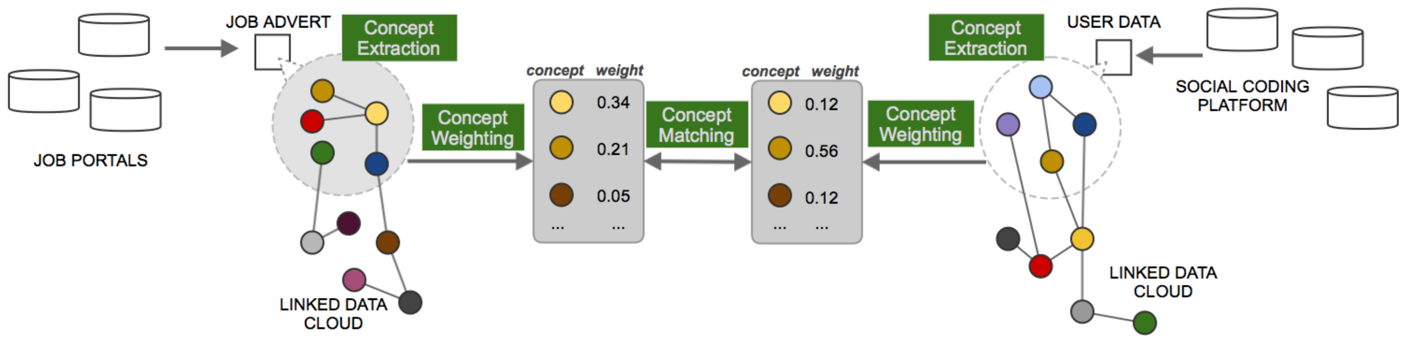

Fig. 1: Overview of our pipeline

developer profile and the advertisement into the Linked Open Data (LOD) [4] space, where we can exploit the background information available in the LOD cloud to bridge the semantic gap mentioned earlier. Additionally, this setup allows us to (partially) rely on well-tested algorithms and toolkits and it provides a natural mechanism to determine the similarity between a natural language job advertisement and a developer's GitHub profile.

In the following section we describe our proof-of-concept and provide an overview of the challenges that need to be overcome.

II. Approach

The general overview of our pipeline is shown in Figure 1 with the three main components being:

- **Extraction** of concepts from job advertisements and social coding user data
- **Weighting** of concepts in such a way that more important concepts receive higher weights
- **Matching** of the two (job and coding profile-based) concept vectors

A. Concept extraction

On the left-hand side in Figure 1, we take as input a job advertisement in natural language text and extract the entities (or concepts) that appear in it. *Named entity recognition* (that is, determining which word or phrase refers to some entity) in combination with *named entity disambiguation* (that is determining to which concrete entity a particular word/phrase refers to) have been shown to be powerful tools to turn natural language text into a structured representation that machines can reason about.

As a concrete example, consider the following excerpt from one of the job advertisements in our data set and its corresponding automatically derived Named Entity annotations. Shown in bold are all phrases that were recognized as referring to entities (or concepts). The annotations relevant to our scenario are shown in blue, while non-relevant entities are shown in red.

The **successful**	[dbr/Successful_(song)]
candidates	[dbr/Candidate]
will have **experience**	[dbr/Experience]
of **OOP**	[dbr/Object-oriented_programming]
in at least one of **PHP**	[dbr/PHP]
(or another comparable,	
dynamic language),	[dbr/Dynamic_programming_language]
Java	[dbr/Java_(programming_language)]
(ideally with **GWT**)	[dbr/Google_Web_Toolkit]
or **C++**	[dbr/C++]
(ideally with **Win32**).	[Windows_API]
They will also be well versed in	
Test Driven Development	[dbr/Test-driven_development]
and advanced practices of	
Object Oriented Programming	[dbr/Object-oriented_programming]
such as Design Patterns and **Refactoring**.	[dbr/Martin_Fowler]

We annotated the above text with DBPedia Spotlight[3], one of the most commonly used open-source annotation toolkits for natural language text. The phrases (the so-called *surface form* of an entity) are not simply matched against a list of entity names, instead they are disambiguated based on the context a word or phrase appears in — for instance, "Java" is recognized as the programming language entity (instead of the island, the coffee type or another one of the more than 30 different entities that have the surface form "Java"). Each entity is uniquely identified through its URI as a particular Linked Open Data concept which in the case of DBPedia Spotlight is prefixed by `http://dbpedia.org/resource/` (for brevity, we replaced this prefix with `dbr` above). Concepts are linked to each other through different types of properties, thus forming a densely connected graph structure. The large-scale DBpedia ontology is automatically derived from Wikipedia and based on the English Wikipedia edition, currently contains more than 4.5 million elements ("things") and nearly 600 million links between them. Figure 2 contains a small excerpt of this graph – even though C++ and the Java programming language are not directly linked, we can determine some degree of relatedness by a walk across the graph (similarly, we can observe a

[3]http://dbpedia-spotlight.github.io/

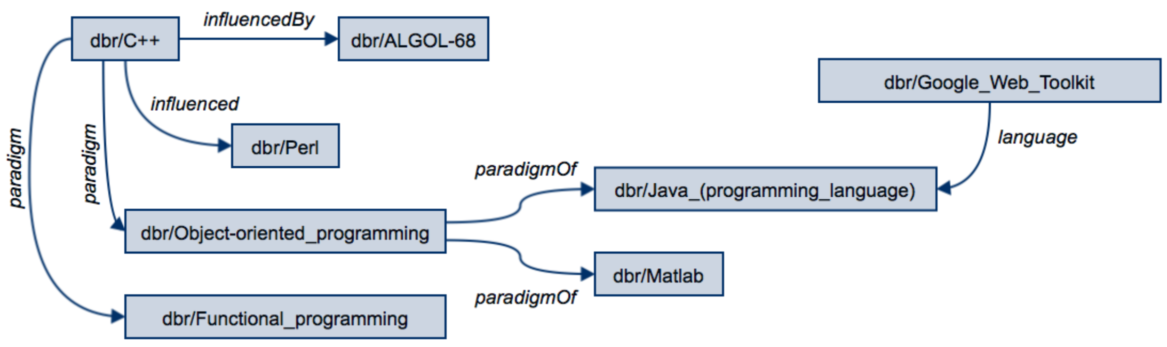

Fig. 2: An excerpt of DBPedia's graph structure. Each node is a concept, the properties between two concepts are captured in the form of a labelled edge.

lower relatedness degree between C++ and the GWT, as the distance between the two concepts is larger). Such graph walks allow us to determine the relatedness between virtually any two concepts. Since in our particular use case we are mostly interested in concepts related to information technology, we restrict the annotations to concepts that have the type *computer* or *internet*. Wikipedia (and thus DBPedia) contains entries covering most if not all programming languages, many important programming frameworks and libraries as well as many computer science concepts. We thus consider it a suitable ontology to use for our specific use case.

While the annotation of the job advertisements is straightforward, annotating a developer's GitHub data with concepts from the same ontology (to allow a simple matching strategy) is more challenging. While programming language identification can largely be considered a solved problem, more fine-grained information such as the libraries used by a particular project often require language-specific solutions (e.g. Maven projects requires the parsing of pom files, ...). Furthermore, identifying how well a developer has internalized particular concepts (e.g. WebSocket programming, the TCP protocol, Bloom filters) requires a deep parsing and semantic tagging of the developer's source code contributions. We conducted a preliminary study (described later) that suggests that a project's README can be utilized as a reasonable approximation of a project's content. Since such READMEs are usually in natural language form, we are able to use exactly the same process as for the job advertisements, only instead of extracting the concepts of a single piece of text, we process all README files contained in all projects the developer is *associated* with. Here, association can either mean a project the developer created, forked or watches.

We turn to the vector space model as algebraic formulation of the concepts found in the job advertisement as well as the GitHub developer data: $job_i = (w_{1,i}, w_{2,i}, ..., w_{n,i})$ and $dev_j = (w_{1,j}, w_{2,j}, ..., w_{n,j})$. Each dimension corresponds to one DBPedia concept and if a concept c_s is found in the job advertisement (or developer profile), the corresponding weight w_s is set to a value above zero. Since these vectors are extremely sparse, we also investigate the use of random walks on the DBPedia graph to "switch on" related concepts that are not explicitly mentioned in the advertisement or developer profile. How to exactly set the weights is the task of the concept weighting component discussed next.

B. *Concept weighting*

Not all concepts extracted from an advertisement or developer profile are equally important. We use the well-known TF.IDF weighting scheme to determine non-binary concept weights (benefitting concepts which occur rarely across the entire corpus of documents, but often within particular documents); these weights are computed separately for the corpus of job advertisements and the corpus of developer profiles. Additionally, we enrich the concept vectors by adding for each found concepts up to t semantically related concepts based on a particular relatedness measure.

C. *Concept matching*

The vector space provides a natural mechanism to determine the similarity between the devised job and developer profile vectors, namely the cosine of the angle between the two vectors (the so-called cosine similarity), which is bounded to a value in $[0, 1]$ where a larger score indicates higher similarity. We can now for each job advertisement rank all developer profiles as well as the other way round.

III. Data

We crawled XXX job advertisement in early 2015 containing the phrase *Software Developer* from the UK version of the job portal Indeed[4]. In total, $9,848$ job

A. *Developer profiles*

Mining developer profiles from GitHub's publicly available data sources incurs the challenge of how to derive features from this raw data that are useful in the context of matching job advertisements to developers. Prior work has considered a number of high-level concepts that recruiters consider when looking for developers, such as ....

[4]http://www.indeed.co.uk

B. Job profiles

C. Matching developer and job profiles

IV. Preliminary Work

V. Related Work

VI. Conclusions

References

[1] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Social coding in github: transparency and collaboration in an open software repository," in *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*. ACM, 2012, pp. 1277–1286.

[2] L. Singer, F. Figueira Filho, B. Cleary, C. Treude, M.-A. Storey, and K. Schneider, "Mutual assessment in the social programmer ecosystem: An empirical investigation of developer profile aggregators," in *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*, ser. CSCW '13. New York, NY, USA: ACM, 2013, pp. 103–116. [Online]. Available: http://doi.acm.org/10.1145/2441776.2441791

[3] J. Marlow and L. Dabbish, "Activity traces and signals in software developer recruitment and hiring," in *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*, ser. CSCW '13. New York, NY, USA: ACM, 2013, pp. 145–156. [Online]. Available: http://doi.acm.org/10.1145/2441776.2441794

[4] C. Bizer, T. Heath, and T. Berners-Lee, "Linked data-the story so far," *International Journal on Semantic Web and Information Systems (IJSWIS)*, vol. 5, no. 3, pp. 1–22, 2009.