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

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.

Business-oriented social networks, such as LinkedIn[1], are using recommender engines to *push* job advertisements to their 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.

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

In the recent years, social coding platforms have become an important tool for developers to become visible in the developer community [1]. Developers use sites such as GitHub and BitBucket to showcase their work in the hope that this will help them in the hiring process by potential employers [2]. However, judging the qualification of an applicant based on his or her GitHub profile is equally challenging [3]. 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 neither does it offer immediate insights into the developer's programming abilities nor does it highlight the particular languages or tool chains the developer knows.

To address the process shortcomings we identified above, we propose a pipeline that *automatically* mines GitHub user profiles and job advertisements for relevant information. We employ an approach that "translates" both the developer profile and the advertisement into the Linked Open Data (LOD) [5] 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, while it provides a natural mechanism to determine the similarity between a natural language job advertisement and a developer's GitHub profile.

II. Approach

The general overview of our pipeline is shown in Figure 1. The three main components are:

- **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 Overview

1) The DBPedia Ontology: Our approach is based on two techniques from natural language processing namely, *named entity recognition* (NER) in combination with *named entity disambiguation* (NED). Given an unstructured text (e.g. a job

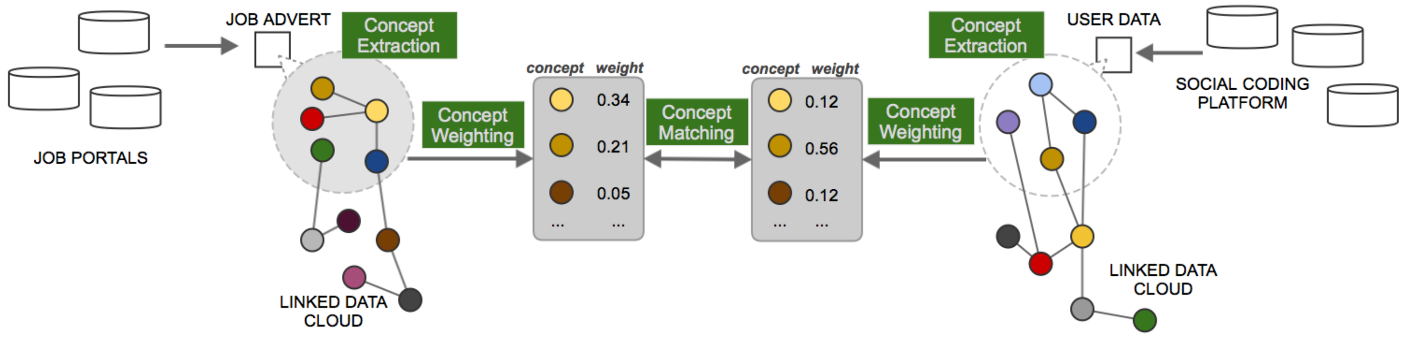

Fig. 1: Overview of our pipeline

advertisement) NER determines which words or phrases in the text refer to an entity, which can be any real-world entity. NED determines which concrete entity a particular word or phrase refers to. The combination of both techniques have been shown [6] to be a powerful mechanism to turn natural language text into a structured representation that machines can reason about.

We rely on DBPedia Spotlight,[2] one of the most commonly used open-source annotation toolkits for natural language text, as a concrete implementation of concept extraction. The large-scale DBpedia ontology [7] behind the DBPedia Spotlight service is automatically derived from Wikipedia and (based on the English Wikipedia edition), currently contains more than 4.5 million entities ("things") and nearly 600 million links between them. Wikipedia (and thus DBPedia) contain entries covering most if not all programming languages, 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.

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

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 concept (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 as a particular LOD concept. Concepts are linked to each other through different types of properties, thus forming a densely connected graph structure. 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 or by considering the textual similarity between each entity's description [8] (similarly, we can observe a lower relatedness degree between C++ and the GWT, as the distance between the two concepts is larger). 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*.

B. Developer Profile Extraction

While the annotation of the job advertisements is straightforward (as just shown in the example), 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.xml` files). Furthermore, identifying the degree of familiarity a developer has with particular concepts (e.g. WebSocket programming, the TCP protocol, Bloom filters) requires deep parsing and semantic tagging of the developer's source code contributions; parsing at scale requires vast computing resources while concept extraction from source code is neither precise nor complete [9].

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

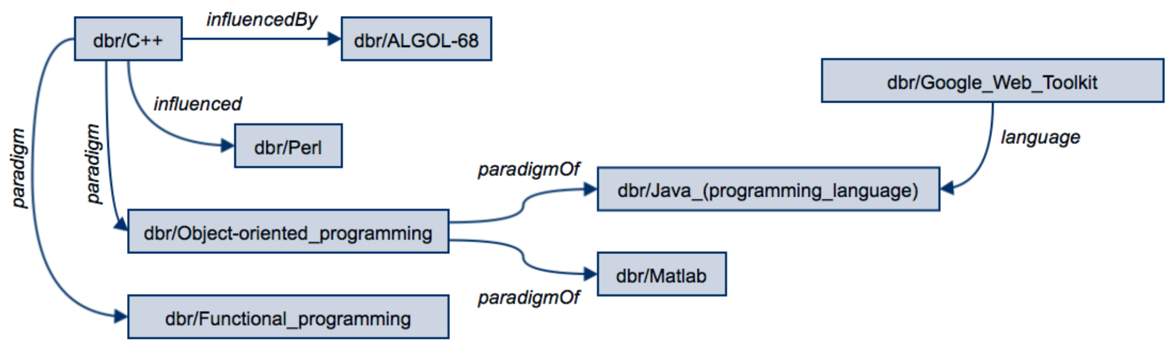

Fig. 2: An excerpt of DBPedia's graph structure. Each node is a concept, the properties between two concepts are captured in the form of a labelled edge.

We advocate that README (and in general, documentation) files in repositories in the developer's profile are a good proxy for the developer's familiarity with certain programming concepts, languages and toolkits. A typical README file contains information about what the software can do, description of technologies used in it, steps on how to recreate development environments and in general a wealth of information that the developers should be familiar with if they own a repository. Since such READMEs are in natural language form, we can 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.

C. *Concept weighting*

The extracted concepts from job descriptions and GitHub developer profiles are converted to vectors: $job_i = (w_{1,i}, w_{2,i}, ..., w_{n,i})$ 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 semantic relatedness measures on the DBPedia data to "switch on" related concepts that are not explicitly mentioned in the advertisement or developer profile.

Not all concepts extracted from an advertisement or developer profile are equally important. We use a basic concept weighting scheme commonly employed in information retrieval: TF.IDF [**?**] which gives a low weight to concepts that appear in many documents (as those are not very informative) while at the same time benefiting 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.

D. *Concept matching*

The vector space model [**?**] 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 according to the similarity as well the other way round.

Number of job adverts	$1,337$
Average number of words / advert	284.7
Aveage number of entities / advert	84.1
Average number of distinct entities / advert	59.0
Average number of IT entities / advert	14.4
Average number of distinct IT entities / advert	9.7
Total number of distinct IT entities found	486

TABLE I: Overview of our job advertisement data set.

III. Experiments

a) Job advertisements: We crawled job advertisements posted in early 2015 from the UK job portal Indeed[3]. Each search on Indeed returns up to $1,000$ search results (i.e. advertisements); we conducted five searches using the following search phrases: $\{$*software developer*, *software architect*, *software engineer*, *computer science*, *web developer*$\}$ and thus retrieved $5,000$ advertisements in total. Of those more than $1,300$ advertisements were posted directly on the Indeed platform (instead of referring to an externally placed advertisment). We annotated them with DBPedica concepts as discussed before; the main results are shown in Table I. Important for our work is the fact that on average ten IT concepts are mentioned in each advert. The ten most occurring concepts in this data set are shown in Table II (left column).

b) Developer profiles: We extracted the $1,000$ top committers from the GHTorrent [**?**] dataset. 863 of those users forked or created at least one project that contains a non-empty `REAdME.md` file, the remainder are either bots (63 users) or created/forked projects with only empty README.md files

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

Job advertisements	Developer profiles
Computer_software (41.8%)	Hypertext_Transfer_Protocol (78.0%)
Application_software (39.6%)	Scripting_language (67.0%)
Cascading_Style_Sheets (37.8%)	HTML (67.0%)
Business (36.4%)	COM_file (61.1%)
JavaScript (34.5%)	Debian (60.5%)
HTML (23.0%)	JavaScript (56.4%)
Debian (19.2%)	Library_computing (44.8%)
PHP (18.9%)	Git_(software) (29.2%)
Computing_platform (18.4%)	GNU_Project (29.2%)
.NET_Framework (18.0%)	Wiki (27.5%)

TABLE II: Overview of the ten most often occurring IT concepts in our job advertisement and developer profile data sets. Shown in brackets is the percentage of documents (job advertisements / developer profiles) that contain the concept at least once.

Number of developers	863
Average number of project README files / developer	32.3
Average number *forked* project README files / developer	16.5
Average number *created* projet README files / developer	15.7
Average number of words / README file	384.5
Average number of non-code words / README file	320.2
Average number of entities / README file	52.3
Average number of distinct entities / README file	17.23
Average number of IT entities / README file	18.0
Average number of distinct IT entities / README file	6.0
Average number of entities /developer	1447.3
Average number of distinct entities / developer	291.3
Average number of IT entities / developer	500.1
Average number of distinct IT entities / developer	78.1
Total number of distinct IT entities found	764

TABLE III: Overview of our developer profile data set.

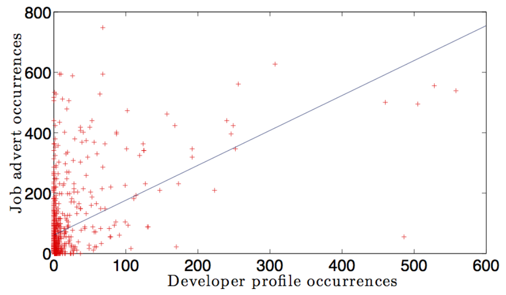

Fig. 3: Each marker represents one IT entities (851 in total) which represent the union of all concepts found in the job adverts and developer profile data sets. The scatter plot shows in how many developer profiles the entity appeared (at least once) vs. in how many job advertisements.

(70 users). Thus, overall for less than 10% of human GitHub it is not possible to use README's as source of developer profile data. Each `README.md` is associated with one project. The main statistics of our developer data set are shown in Table III. Importantly on average for each developer we are able to extract 500 IT entities across all the developer's README files. The 57 distinct IT entities found on average per developer indicate the high level of detailed information we are able to extract. The right column of Table II shows the most often occurring IT entities across developers. In support of our envisioned application, we find substantial overlap between the entities extracted from job advertisements and the entities extracted from developer profiles as evident in Figure 3; while some concepts appear either only among the developer profiles or among the job adverts, the majority of concepts appear at least once in both corpora. The linear correlation between the number of times a concept appears in developer profiles vs. job adverts is $r = 0.49$ (fitted line in Figure 3).

c) Matching developer and job profiles: Finally we briefly discuss the results of ranking all developers for a single

IV. Related Work

Profiling developers using trace data is an active field of research. Developer profiles have been built, among others, for building project-specific expertise knowledge bases [**?**], identify developers with similar expertise [**?**], measure develop productivity [**?**], and recommending developers for specific tasks [**?**] (e.g. bug resolution [**?**]). However, to the best of our knowledge, no work extracted developer profiles for matching with job advertisements.

In reference [1], Capiluppi et al. described a process (and its pitfalls) for assessing technical candidates using data, among others, from social coding sites. Marlow et al. [4] investigated how more detailed, publicly accessible signals about a developer's activities 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.

V. Conclusions

¡¡¡¡¡¡¡ HEAD =======
¿¿¿¿¿¿¿ c9906f194068069879d1ed1bd8acce9b65bb995b

References

[1] A. Capiluppi, A. Serebrenik, and L. Singer, "Assessing technical candidates on the social web," *Software, IEEE*, vol. 30, no. 1, pp. 45–51, Jan 2013.

[2] 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.

[3] 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

[4] 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

[5] 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.

[6] C. C. Aggarwal and C. Zhai, *Mining text data*. Springer Science & Business Media, 2012.

[7] P. N. Mendes, M. Jakob, A. García-Silva, and C. Bizer, “Dbpedia spotlight: shedding light on the web of documents,” in *Proceedings of the 7th International Conference on Semantic Systems*. ACM, 2011, pp. 1–8.

[8] E. Gabrilovich and S. Markovitch, “Computing semantic relatedness using wikipedia-based explicit semantic analysis.” in *IJCAI*, vol. 7, 2007, pp. 1606–1611.

[9] A. Kuhn, S. Ducasse, and T. Grba, “Semantic clustering: Identifying topics in source code,” *Information and Software Technology*, vol. 49, no. 3, pp. 230 – 243, 2007, 12th Working Conference on Reverse Engineering. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584906001820