Semantic approaches to software component retrieval with English queries

Huijing Deng, Grzegorz Chrupała

Management Information Systems, Tilburg Center for Cognition and Communication
ETH Zürich, Tilburg University
huijingdeng@ethz.ch, g.chrupala@uvt.nl

Abstract

Enabling code reuse is an important goal in software engineering, and it depends crucially on effective code search interfaces. We propose to ground word meanings in source code and use such language-code mappings in order to enable a search engine for programming library code where users can pose queries in English. We exploit the fact that there are large programming language libraries which are documented both via formally specified function or method signatures as well as descriptions written in natural language. Automatically learned associations between words in descriptions and items in signatures allows us to use queries formulated in English to retrieve methods which are not documented via natural language descriptions, only based on their signatures. We show that the rankings returned by our model substantially outperforms a strong term-matching baseline.

Keywords: natural language code search; software component retrieval; semantic grounding

1. Introduction

Enabling code reuse is of paramount importance in modern software engineering. Effective code search interfaces can make a major contribution towards this goal. Here we propose to answer natural language queries in an Application Programming Interface (API) search system via a simple form of grounding the meanings of words in programming language constructs.

We exploit the fact that there are large programming language libraries which are documented both via formally specified function or method signatures as well as descriptions written in a natural language such as English. This allows us to learn to associate words with elements of method signatures which they frequently co-occur with. These associations enable the use of queries formulated in English to retrieve methods which have no corresponding descriptions, only based on their signatures.

In this paper we focus on the Java Standard Library, but we believe that our approach is applicable to other similar API collections. Java methods are documented via a format known as Javadoc. A simplified example of documentation in this format for the method `charAt` is shown in Figure 1. The method signature contains the method name (`charAt`), qualifier (`public`), return type (`char`), as well as parameter type (`int`) and names (`index`). The method description consists of one or more sentences written in English. The first sentence typically describes what the method does. It is followed by any additional detailed information.

We collected such Javadoc documentation for the Java Standard Library. As our baseline we use the simple term-matching model which assumes that the method description and signature are written in same language. We then developed two models which do not make this assumption but rather learn to connect words in the method Javadoc description with terms in the corresponding signature. These are the Polylingual Latent Dirichlet Allocation (PLDA) model (Mimno et al. 2009) and IBM model 1 (Brown et al. 1993).

Class `java.lang.String`
Signature `public char charAt(int index)`
Description Returns the `char` value at the specified index. An index ranges from 0 to `length() - 1`. The first char value of the sequence is at index 0, the next at index 1, and so on, as for array indexing.

Figure 1: Javadoc documentation for method `charAt` in the class `java.lang.String`

Initially developed for machine translation, IBM model 1 also proved effective in cross-lingual document retrieval (Berger and Lafferty 1999). The PLDA model attempts to find topics shared across languages. These two models provide two different ways of discovering cross-lingual associations from parallel corpora.

We test the models on the task of API search via English-language queries. We aim to enable a user to use a query such as *return the character at a specified index in a string*, and to receive a ranked list of methods which implement the desired functionality.

We simulate such queries by taking the first sentence of the description of a held-out portion of the Javadocs. The first sentence typically states succinctly, in simple English, what the method does, and thus is similar in form to a user query. For each such query we rank method signatures from the whole collection according to how semantically related they are to the natural language query.

We show that the ranking based on the translation model between words and signature terms substantially outperforms a strong term-matching baseline and achieves Mean Reciprocal Rank scores of nearly 50% and accuracy@10 of almost 80%.

2. Models

The task is to retrieve methods using queries formulated in natural language. We rank methods via a language model-

ing approach (Song and Croft 1999), according to the probability of the method d given query q:

$$p(d|q) = \frac{p(q|d)p(d)}{p(q)}.$$

By assuming a uniform prior $p(d)$ and ignoring the normalization constant we can use the likelihood term $p(q|d)$ to rank methods.[1] With a unigram language model for generating q from d we get:

$$p(q|d) = \prod_{w \in q} p(w|d). \quad (1)$$

Term-matching baseline As a baseline for $p(w|d)$ we use a term-matching approach which relies on the fact that many words have the same form as signature terms.
With this model we set $p(w|d)$ to the maximum likelihood estimate (MLE) with Jelinek-Mercer (Zhai and Lafferty 2001) smoothing:

$$p(w|d) = (1 - \lambda) f(w|d) + \lambda f(w|D),$$

where $f(w|d)$ is the relative frequency of w in method signature d and $f(w|D)$ is its relative frequency in the whole method collection.

IBM model 1 For this model we use the parallel corpus of method signatures and the corresponding descriptions to extract word–signature-term associations: we build a translation table which gives the probability of word w given signature term u: $p(w|u)$. IBM model 1 assumes that all alignments between two strings are equiprobable, and bootstraps translation probability estimates using Expectation Maximization (Brown et al. 1993).
When using the translation model, we marginalize over the terms u in the signature:

$$p(w|d) = (1 - \lambda) \left[\sum_{u \in d} p(w|u) f(u|d) \right] + \lambda f(w|D), \quad (2)$$

where $p(w|u)$ is given by the translation table and $f(u|d)$ is the relative frequency of u in d. Similarly to the baseline model, we apply Jelinek-Mercer smoothing to $p(w|d)$. By putting together equations 1 and 2, the ranking is determined by:

$$p(q|d) = \prod_{w \in q} (1 - \lambda) \left[\sum_{u \in d} p(w|u) f(u|d) \right] + \lambda f(w|D).$$

Interpolated PLDA We also implemented a model based on Polylingual LDA (Mimno et al. 2009), in which the associations between words and signature terms are mediated by topics. After training PLDA on the parallel training data, we can infer the distribution over topics $p(t|d)$ for a method signature d. We also use the word distributions of the topics $p(w|t)$. By putting the two together and marginalizing over the topics we get:

$$p(q|d) = \prod_{w \in q} \left[\sum_{t \in T} p(w|t) p(t|d) \right]. \quad (3)$$

[1] Of course given suitable data to estimate its parameters, a more informative prior could be used.

A topic collapses a number of words: for document retrieval, it may not be as precise as a word-level representation. Wei and Croft (2006) show clear improvements for retrieval when linearly interpolating a topic model and a term-matching model. We thus interpolate PLDA with our baseline model:

$$p(q|d) = (1 - \alpha) \times p_{\text{PLDA}}(q|d) + \alpha \times p_{\text{BASELINE}}(q|d). \quad (4)$$

3. Experiments

We collect the documentation of six packages from the Java standard library (Standard Edition 6 API Specification): `io`, `lang`, `math`, `net`, `text`, `util`. We extract signatures and their corresponding descriptions for 7183 methods. We split this data set into training set (60%), validation set (20%), and test set (20%). The models are trained only on the description-signature pairs in the training set. The validation set is used to determine the optimal parameters in the models. Finally we test all the models with their optimal parameters on test set.
To simulate queries, we take the first sentence of descriptions in the test (or validation) set (see Figure 1). We discard queries which contain less than three words.
When testing the models, for each query in the test set we use the model to rank the *union of the method signatures in the training and test set*. This way of ranking a large number of methods makes the evaluation more realistic and the task much more challenging.
We preprocess all the data by removing punctuation and HTML tags from the whole corpus and by splitting compound camel-cased terms, and lowercasing all terms: e.g. we turn `markSupported` into `mark` and `supported`. For method signatures, we include the package and class they belong to, as well as the superclass. We also tested different vocabulary reduction settings by filtering out most frequent and infrequence words from the preprocessed signature and description separately. For term-matching model and IBM model, non-filtered data achieved the best MRR scores. But for PLDA model, we found out that the optimal condition was to train on data where we filter out the twenty most frequent words while always preserving the first two words of the description (which usually correspond to the method name). After the preprocessing steps the signature for `charAt` from Figure 1 would look like this: `char at lang public object string char index int`.
Since each query has exactly one relevant signature, we use mean reciprocal rank (MRR) as our evaluation metric:

$$MRR = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{\text{rank}(i)}.$$

where N is the number of queries, $\text{rank}(i)$ is the rank of the correct method signature given the i^{th} query within the list of candidate signatures ranked according to the score produced by the model.

3.1. Results

Figure 2 plots the validation performance of Baseline and IBM model 1 as a function of smoothing parameter λ.

k	300	500	800	1000	1500
MRR	0.278	**0.291**	0.262	0.280	0.217

Table 1: PLDA on validation set with varying number of topics k

α	0.0	0.1	0.3	0.5	0.7	0.9
MRR	0.291	**0.375**	0.373	0.372	0.370	0.366

Table 2: Interpolated PLDA on validation set with varying α

There are two versions of IBM model 1: after one iteration of EM and after 15 iterations of EM. We tested iterations ranging from 1 to 30; we only show the case of 15 iterations, since past 15 iterations, MRR scores did not change appreciably.

The term-matching model is quite a strong baseline in this evaluation: the MRR with the optimal value of λ on the validation set is 34.4%, i.e. the harmonic mean of the rank of the correct method is around 3 (out of more than 5000). In the Java Standard Library, methods and their arguments typically have informative, English-based names and thus tend to match words in the query. This may not always be the case for other libraries and in such cases we would expect the term-matching model to perform less well.

We tuned the number of topics for the pure PLDA model: as shown in Table 1 the best k was 500 with MRR at 29.1%. Table 2 shows the validation set performance of Interpolated PLDA with $k = 500$ as a function of α. With the best setting Interpolated PLDA gives a moderate gain over the baseline, reaching 37.5%.

In contrast, IBM model 1 substantially outperms the baseline and Interpolated PLDA, and brings the MRR to 52%. The model can deal effectively with queries where the baseline fails badly. Figure 3 shows two example queries where the rank of the correct method was improved dramatically by the translation model compared to the baseline. In both cases there are few code-terms corresponding to words in the query, and the ones that are shared are not informative (such as *object*).

Table 3 shows the retrieval performance of the models on the test set. Besides MRR scores, it also shows the accuracy@1 and accuracy@10, which indicate the proportion of queries that have the correct answer at position 1, or within the top 10 results. For IBM model 1 returns the correct answer within the top 10 results for almost 80% of the queries.

Model	**MRR**	**Acc@1**	**Acc@10**
Baseline ($\lambda = 0.7$)	0.332	0.223	0.530
Interp. PLDA ($\alpha = 0.1$)	0.352	0.242	0.562
IBM model 1 ($\lambda = 0.3$)	**0.493**	**0.339**	**0.793**

Table 3: Results on test data

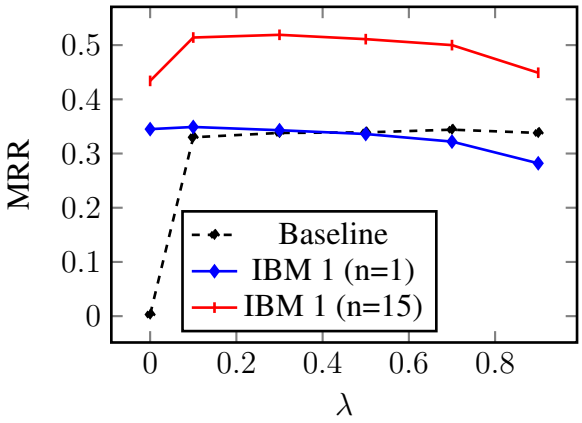

Figure 2: Results on validation data

Query returns the trigonometric tangent of an angle
Signature public static double tan(double a)
Signature terms tan lang public object strict math double public a double
Rank change 1998 → 2

Query compares the specified object with this map for equality as per the definition in the map interface
Signature public boolean equals(Object o)
Signature terms equals util public dictionary hashtable boolean public o object
Rank change 1718 → 1

Figure 3: Example queries where baseline fails

4. Related work

Our work builds on several strands of research from somewhat disconnected areas. Firstly, it can be seen as implementing a simple form of grounding for natural language. There are too many approaches to this problem to discuss here, but the most related work in this domain is perhaps on translating natural into formal languages, e.g. (Matuszek et al. 2010, Branavan et al. 2010, Liang et al. 2011). These works are concerned with full-fledged mappings between sentences and a formally specified semantic interpretation. We are more interersted in learning distributional meaning representations of words in terms of elements of executable source code. Our use of a word-alignment model to associate words with meanings is also reminiscent of the work of Fazly et al. (2010) which models cross-situational word learning by infants.

Secondly, we apply our word representation to API retrieval. Casting information retrieval as a translation problem in order to account for systematic differences in language between queries and documents dates back to Berger and Lafferty (1999). Different versions of this basic approach have been since explored, e.g. in Lafferty and Zhai (2001), Momtazi and Klakow (2011). Our contribution here is to apply this idea to the problem of API search via natural language queries.

API search and retrieval have been of interest within the

software engineering community, e.g. Stylos and Myers (2006), Bajracharya et al. (2006), McMillan et al. (2011). These works build prototypes of API retrieval systems using basic techniques from NLP and IR, analogous to what we implemented in our baseline, i.e. relying on code terms matching query terms. Other approaches rely on code-specific query languages (Mandelin et al. 2005). None of these works specifically address natural language queries or exploit language-code translation.

5. Conclusion

Numerous advantages make component-based construction an dominant trend in software development: higher reliability, higher efficiency, lower cost, reduced skills requirements. An effective API search interface is the key to component reuse, and being able to specify requirements in a natural form is an important feature.

Our research demonstrates that a user-friendly natural language API search interface can be built by exploiting naturally occurring language–code parallel data to ground word meanings. The level of performance we have seen is already useful for a real-world application: 80% of queries receive the correct answer withing top 10 results.

Together with this paper we release the data and code developed for this work. The public repository can be accessed at https://bitbucket.org/gchrupala/codeine. We hope these resources will stimulate further research into applying NLP techniques to API retrieval.

Many questions remain to be explored. Firstly we would like to evaluate our approach on real user queries, such as those found on online programming QA forums. Secondly, it would be interesting to see how our method generalizes to other programming languages. We foresee challenges with the lack of static typing in e.g. Python or Javascript, where function signatures convey less information than in Java. In addition to standard library APIs, software developers increasingly rely on unofficial, third-party APIs. In many cases these APIs are not documented as well as official ones, and they could serve as a testbed for our natural-language based search approach. In the current work we have used the bag-of-words approximation to natural language documents and method signatures. More precise API search may be enabled by more structured representations.

Acknowledgments

As part of a Huijing Deng's master thesis, this work was supported by the chair of Spoken Language Systems, Saarland University. The chair of Management Information Systems, Swiss Federal Institute of Technology Zurich also co-funds the work.

References

Bajracharya, S., Ngo, T., Linstead, E., Dou, Y., Rigor, P., Baldi, P., and Lopes, C. (2006). Sourcerer: a search engine for open source code supporting structure-based search. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, pages 681–682. ACM.

Berger, A. L. and Lafferty, J. D. (1999). Information Retrieval as Statistical Translation. In *SIGIR*, pages 222–229.

Branavan, S., Zettlemoyer, L. S., and Barzilay, R. (2010). Reading between the lines: Learning to map high-level instructions to commands. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1268–1277. Association for Computational Linguistics.

Brown, P. F., Pietra, S. D., Pietra, V. J. D., Mercer, R. L., and Mercer, R. L. (1993). The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, pages 263–311.

Fazly, A., Alishahi, A., and Stevenson, S. (2010). A probabilistic computational model of cross-situational word learning. *Cognitive Science*, 34(6):1017–1063.

Lafferty, J. and Zhai, C. (2001). Document language models, query models, and risk minimization for information retrieval. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 111–119. ACM.

Liang, P., Jordan, M. I., and Klein, D. (2011). Learning dependency-based compositional semantics. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 590–599. Association for Computational Linguistics.

Mandelin, D., Xu, L., Bodík, R., and Kimelman, D. (2005). Jungloid mining: helping to navigate the API jungle. In *ACM SIGPLAN Notices*, volume 40, pages 48–61. ACM.

Matuszek, C., Fox, D., and Koscher, K. (2010). Following directions using statistical machine translation. In *Proceedings of the 5th ACM/IEEE international conference on Human-robot interaction*, pages 251–258. IEEE Press.

McMillan, C., Grechanik, M., Poshyvanyk, D., Xie, Q., and Fu, C. (2011). Portfolio: finding relevant functions and their usage. In *Software Engineering (ICSE), 2011 33rd International Conference on*, pages 111–120. IEEE.

Mimno, D., Wallach, H. M., Naradowsky, J., Smith, D. A., and McCallum, A. (2009). Polylingual topic models. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2-Volume 2*, pages 880–889. Association for Computational Linguistics.

Momtazi, S. and Klakow, D. (2011). Trained trigger language model for sentence retrieval in QA: bridging the vocabulary gap. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 2005–2008. ACM.

Song, F. and Croft, W. B. (1999). A general language model for information retrieval. In *Proceedings of the eighth international conference on Information and knowledge management*, pages 316–321. ACM.

Stylos, J. and Myers, B. A. (2006). Mica: A web-search tool for finding API components and examples. In *Visual Languages and Human-Centric Computing, 2006. VL/HCC 2006. IEEE Symposium on*, pages 195–202. IEEE.

Wei, X. and Croft, W. B. (2006). Lda-based document models for ad-hoc retrieval. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 178–185. ACM.

Zhai, C. and Lafferty, J. (2001). A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 334–342. ACM.