DATA ENGINEERING KODEBEAGLE LABS

NLP and Code Analysis — Part I

by
Sachin Tyagi

posted on
July 20, 2015 (July 20, 2015)

In the previous post I represented the idea that human written code (in programming languages like Java, C etc.) has many of the same properties that the documents written in natural languages (like Hindi or English etc.) have — i.e. predictability, repetitiveness and a propensity towards being used idiomatically. And as was pointed out, this makes a given code corpus amenable to statistical analysis in general and natural language processing in particular.

Moreover, software engineering best practices like separation of concerns, high cohesion and loose coupling [1] imply that much as the documents written in natural languages aim to achieve subject coherence, documents written in computer code (the collection of classes and script files etc.) also tends towards functional coherence. In other words — as documents written in natural languages tend to achieve cohesion by focusing on limited number of 'topics', the written code documents also try to achieve such coherence by grouping their functionality around a 'concern'. Once we accept this observation, it then becomes plausible that the topic modelling methods that have achieved success in NLP should also have utility in the analysis of computer code. And this applies at each structural level of the software — a software project is generally focused on a particular domain, a package or a module on a more specific area of functionality, and the classes and methods tends to be even more specific. But each tend to focus on a small set of 'concerns' (or 'topics').

So what is topic modelling?

Topic modelling consists of a set of methods that collectively aim to discover the underlying themes within a set of documents. A trivial method (though one that would involve humongous effort) to discover these underlying themes would be hand annotate each document. But this is clearly unfeasible if one wishes to analyze a large corpus — say all the java projects on Github or all the pages of Wikipedia. To overcome this, many probabilistic topic models have been developed. These models, instead of relying of human judgement for each document, leverage the statistical properties of the underlying data to discover the themes or 'topics' in that data.

So how is such topical analysis of code useful?

One obvious utility of such a model would be in an information retrieval system for the written code. Once we have a topic model in place, we can then improve on a simple text search through the code. For a given user query we can determine the topics distribution that best represents the query and then search for the code that matches this distribution closest. And this can be applied across the heirarchy of code structures — i.e project, modules, classes, or even a combination of all these.

Second use case would be for summation of a higher order code structures in terms of their components. For example, with a topic model in hand it then become possible to summarize a class in terms of its functions (or more specifically AST nodes[3]) that best represent the same overall topic distrbution as the class itself. Again this idea can be extended to the projects and modules that can then be summarized in terms of the most representative files. This can arguably help those unfamiliar with a project to quickly familiarize themselves with the project. A basic implementation of these ideas can be found in Tassal [4] (for class summarization) and basic extension of this idea to project summarization here [5].

Third, once we have a hierarchical topic distribution across a code corpus, we can then analyze the same in a project. It can be argued that at some point when more than a certain number of topics start finding representation within a project, its

decomposition into independent projects becomes desirable. It would be interesting to explore this idea in the context of current thinking regarding service decomposition in microservices architecture.

Finally, another possible application can be a similarity analysis across different code structures.

In the next post, I would talk about one of the more prevalent topic model — Latent Dirichlet Allocation (or LDA) and its variations.

[1] https://en.wikipedia.org/wiki/Separation_of_concerns
[2] Daid Blei, Probabilistic Topic Models – http://www.cs.columbia.edu/~blei/papers/Blei2012.pdf
[3] An algorithm for these is presented by Jaroslav Fowkes, Razvan Ranca, Miltiadis Allamanis, Mirella Lapata and Charles Sutton in “Autofolding for Source Code Summarization” – http://arxiv.org/pdf/1403.4503v3.pdf
[4] A basic implementation of algorithms presented in [3] – https://github.com/mast-group/tassal
[5] Natural extension of [4] for project summarization – https://github.com/sachintyagi22/tassal

Read Comments Leave a Comment

TAGGED IN

ML NLP software engineering

Sachin Tyagi

1 COMMENT

Pingback: LDA: NLP and Code Analysis – Part II | experience@imaginea