

Freedville Blog

Andrew R. Freed's Blog

January 13, 2017

Demo of natural language processing with rules and machine-learning based approaches

Introduction

In this cognitive area, many people are interested in using natural language processing (NLP) to extract insights from their large collections of unstructured text. There are two main approaches to natural language processing: rules-based NLP and machine-learning-based NLP. I decided to put together a brief example of how both techniques work so that you can compare the two.

Problem setup

The problem I want to solve is to find all the times computer programming languages are mentioned in a block of text, including languages that I have never heard of before. Rather than using a predefined list of programming languages (this list called a 'dictionary' in NLP terminology), I want the NLP system to use only semantic clues from the text itself to determine when a programming language is being discussed.

Further, I add two constraints to keep the example simple. First, I only process one document: the [Wikipedia article on programming languages](#). Second, I use only one rule in my rules-based system. This low amount of training keeps both techniques on equal footing.

Rules-based NLP

Rules-based NLP is performed by expert rules developers. These developers scan source documents and try to discover rules that will help extract key data points, balancing rules that extract too little vs rules that extract too much. For fun, skim [the article](#) and see what rule(s) you might try.

I settled on the following simple rule:

For every sentence containing the word “language”, remove the first word, and any remaining capitalized words are programming languages.

The results are interesting. I’ll select snippets of text and **bold** the “programming languages” detected by my NLP.

Sometimes this rule is good:

“ *The language above is **Python**.*

Sometimes this rule is great:

“ *In 2013 the ten most popular programming languages are (in descending order by overall popularity): **C, Java, PHP, JavaScript, C++, Python, Shell, Ruby, Objective-C** and **C#**.*

Sometimes the rule is way off (none of the bold words are programming languages):

“ *Edsger **Dijkstra**, in a famous 1968 letter published in the **Communications** of the **ACM**, argued that **GOTO** statements should be eliminated from all “higher level” programming languages.*

And the rule also misses some easy ones (Java is a programming language):

“ *Java came to be used for server-side programming.*

Still, the rule was simple to write with only a few lines of code, and it performed reasonably well. I manually counted the number of programming languages mentioned in the article as 103. The rule found 106 programming languages, 56

correctly detected, 47 incorrectly, thus giving precision of 52.8%, recall of 54.4%, and F1 score of 0.537. (F1 score is our accuracy metric.) Not bad for one rule.

See full code listing: [ProgrammingLanguageRulesBasedAnnotator.java](#).

See video demonstration:

Developing Natural Language Processing With Rules



Machine-learning-based NLP

Machine-learning based NLP does not use any rules – rather it “learns”, or is “trained” by, source documents “annotated” by subject matter experts. Think of annotating as using a highlighter on a source document, highlighting every concept you want the machine learning model to learn. (Use a different colored highlighter for every “type” of concept you want to learn.) For my machine-learning-based NLP, I created a demo instance of [Watson Knowledge Studio](#).

Watson Knowledge Studio suggests you break training documents into 2,000 word sub-documents for optimal machine-learning performance. I actually broke my document into 20-line segments, averaging 500-600 words. I did this mostly because annotating can get tedious and annotating smaller documents at a time gives you more breaks, but also so that Watson Knowledge Studio could better randomize my documents into training sets.

IBM Watson Knowledge Studio

Completed Close

Alpha... 14pt 1

Mention

Relation

Conference

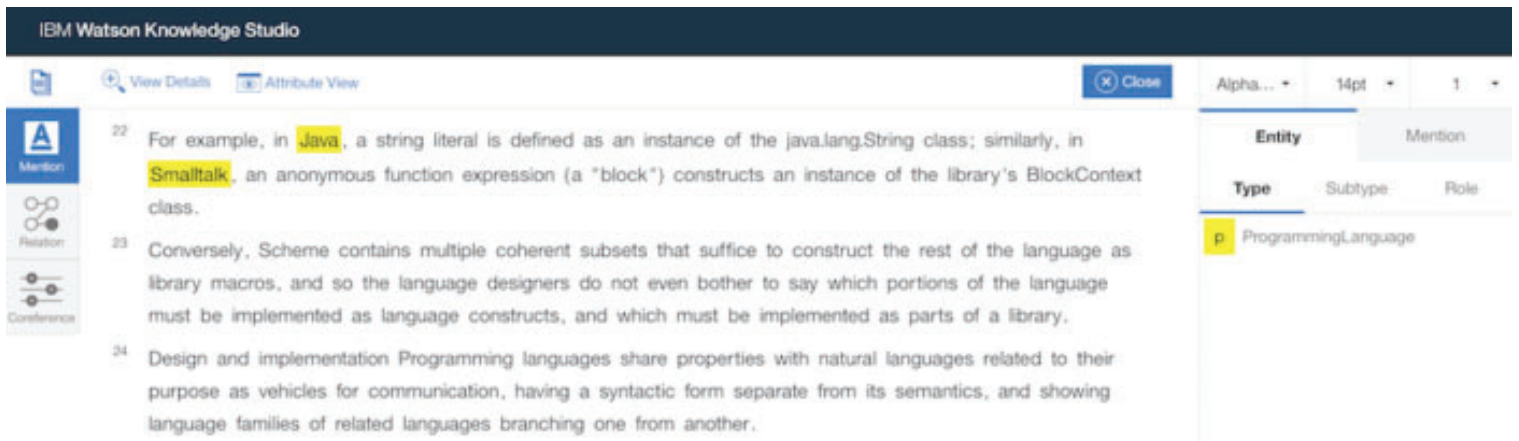
ProgrammingLanguages.part_ad.txt

- 1 The 1980s were years of relative consolidation.
- 2 C++ combined object-oriented and systems programming.
- 3 The United States government standardized Ada, a systems programming language derived from Pascal and intended for use by defense contractors.
- 4 In Japan and elsewhere, vast sums were spent investigating so-called "fifth generation" languages that incorporated logic programming constructs.
- 5 The functional languages community moved to standardize ML and Lisp.
- 6 Rather than inventing new paradigms, all of these movements elaborated upon the ideas invented in the previous decades.
- 7 One important trend in language design for programming large-scale systems during the 1980s was an increased focus on the use of modules, or large-scale organizational units of code.
- 8 Modula-2, Ada, and ML all developed notable module systems in the 1980s, which were often wedded to generic programming constructs.

Entity		
Type	Subtype	Role
p	ProgrammingLanguage	

A training document in Watson Knowledge Studio editor.

All of the documents I annotated became “ground truth” for the machine learning model. I submitted my document set to Watson Knowledge Studio for training, and in approximately 10 minutes I was able to review the results. A nice touch in Watson Knowledge Studio is that the same style of interface is used to review the machine learning results.



Watson Knowledge Studio training results for a single test document

In the example we can see that the machine learning model correctly identified the languages Java and Smalltalk, and missed the programming language Scheme. Incidentally, our single rule would have missed Java and Smalltalk but would have found Scheme! Watson Knowledge Studio also gives us overall performance numbers for the model.

Entity Types	F1	Precision	Recall	% of Total Annotations	% of Corpus Density (by number of words)	% of Documents that Contain This Type
ProgrammingLanguage	0.53	1	0.36	100% (24/24)	1% (24/1707)	100% (3/3)
Overall Statistics	0.53	1	0.36	100% (24/24)	1% (24/1707)	100% (3/3)

Overall accuracy results from Watson Knowledge Studio NLP model

In a shocking turn of events, the machine-learning model has 100% precision, meaning that every time it highlighted text as a programming language, it was correct. However, the recall of 36% means that it only highlighted 36% of the actual programming languages labeled in the ground truth. 100% precision is an unheard of result, surely due to my exceptionally small training data size. As the model improves its recall, precision will certainly drop a bit.

An interesting turn of events is that the rules-based model and the machine-learning-based model had nearly identical F1 scores at approximately 0.53. This is a modest result, not sufficient for a production system but not bad for giving about an hour to each method.

See video demonstration:

Conclusion

This post introduced a small problem to solve with natural language processing and demonstrated two different NLP approaches. Each produced modest results in a quick-and-dirty implementation. In a future post I will further discuss the [pros and cons of rules-based vs machine-learning-based](#) as well as discuss how I would go about [improving my results with each technique](#).

 Uncategorized

 AI, machine_learning, natural_language_processing



Andrew Freed

5 Comments

Pingback: [Comparing rules and machine learning natural language processing approaches – Freedville Blog](#)

Pingback: [Improving simple natural language processing models with rules or machine learning – Freedville Blog](#)

Pingback: [Reaching peak cognitive performance – are we there yet? – Freedville Blog](#)

Pingback: [Chatbot Texterkennung – Machine Learning oder Regelbasierend](#)

Pingback: [Chatbot Texterkennung - Machine Learning oder Regelbasierend?](#)

This site uses Akismet to reduce spam. [Learn how your comment data is processed.](#)

Powered by WordPress | Theme: Write by Themegraphy