The Entropy/Naturalness of Software Documentation

# Abstract

Although Software documentation is written in the Natural languages, it exhibits predictability and is regular in nature. We start with this hypothesis and do few experiments on software documentation. Experiments were performed with Stackoverflow posts used as software documentation corpus. Corpus was divided into 2 parts of 90% and 10% based on its size. We used widely recognized N-gram model approach to train statistical model on the 90% portion, and validate the model on the 10% portion. Results were averaged out over 10-fold cross validation and it shows that software documentation is indeed regular. We also extracted the code elements from the software documentation using ACE from Rigby *et al*. [2] to evaluate the cross entropy of the extracted code elements. Then we plot a graph between cross entropy and n-gram order to compare it with the graph from Hindle *et al.* [1] which provides an evidence for the research hypothesis which we state in the next section.

# Introduction

Natural languages such as English possess a great wealth of expressivity.

Apparently software documentation is a natural product given the fact that it is written in natural languages such as English. Despite its naturalness, software documentation is largely regular and predictable even though it is written in such a complex language which includes an abundance of vague and indefinite phrases. Primary reasons that contribute towards its predictable and regular nature include (1) people have limited set of vocabulary and they tend to use same structures time and again owning to cognitive constraints , (2) Given its nature, software documentation tends to use lots of code elements. (Software are comprised of code elements and far regular than natural language from Hindle *et al.* [1]) (3) Software documentation is directed towards specific topic.

If programmers writing code are repetitive and predictable, we would argue that that questions and answers about code are even more repetitive and predictable because people tend to ask and answer the same questions repeatedly (i.e. people learn in similar ways). Thus any documentation comprises of such questions and answers would be regular and predictable. Posts on famous question- answer websites such as StackOverflow[10] can be considered instances of such documentation. We would like to extend our argument that given there is English as well as code fragments in documentation, documentation would have higher entropy than source code. In effect, we hypothesize that the English between the code elements is obscuring the repetitive and simple nature of the topics covered by the documentation.

## Research Hypotheses

We expect the cross-entropy of the following to be in decreasing order:

1. English text
2. Software Documentation
3. Source code of API
4. Code Elements found in Software Documentation

This paper is structured as follows. In Section (Background) we discuss briefly about Language models, N-gram model, Concept of perplexity and cross entropy. In Section (Literature) we discuss about related research work that has been done using language models. Applications of such language models include automatic syntax completion, generation of readable strings for test oracle to address the human oracle cost problem. In Section (Methodology) we discuss in detail about experiments we performed to provide the evidence to support our hypothesis. We performed a series of experiments with software documentation and code element corpuses. In Section (Findings and Discussion) we compare results of software documentation corpus with code elements corpus. To Gain further insight on how does result stand in comparison to English text and software code, we compare our results with findings from Hindle *et al*. [1]. In the end we also compare results of software documentation for different projects.

# Background and Literature

**Language Model**

In this paper, we use term language model (LM), which is nothing but probability distributions over sequence of **m** tokens P (k**1**, k**2**, …., k**m**). Language Model is trained on a corpus of sequences of tokens from the language, with the goal of assigning high probability to tokens with maximum likelihood, and low probability to tokens with minimum likelihood. Language models are needed to model the uncertainty of the language by determining the most probable sequence of tokens for a given input.

**N-gram model**

Consider the sequence of tokens k**1**, k**2,**k**3** … km-1, km in a document (D). N-gram model statistically calculate likelihood of tokens to follow other tokens. Thus, we can estimate the probability of a document based on the product of a series of conditional probabilities:

p(D) = p(k1) p(k2|k1)p(k3|k1 k2) …… p(kn|k1, k2 ….. kn-1)

Where p (D) is the probability of document and p (ki) is the conditional probability of tokens.

We can transform above equation to following more general form of equation.

P (k1, k2, k3 … km-1, km) = ∑i=1..m P(ki | k1, ….., kn-1)

In this transformation it is assumed that token occurrences are influenced only by limited prefix of length n. This assumption is known as **Markov Property** as described by Zhang et al [11].

To use above equation we need to know conditional probabilities values for each token for each possible n-gram. The conditional probability can be calculated from n-gram frequency counts.

P (ki | ki-(n-1), …… ki-1) =

Bigram and Trigram language models can be modeled with value of n=2 and 3 respectively.

**Perplexity and Cross Entropy**

Perplexity evaluates the language model. The best language model is one that best predicts an undiscovered test data. To assess the quality of a given language modeling technique, the likelihood of new data is most commonly used [6].

Following expressions of perplexity are from Ronald *et.al* [6]

Perplexity = 2 cross-entropy (D.Pm)

Where cross-entropy (D;Pm) = ∑ P(D) . Log Pm (D)

Here Pm is the model distribution

**Literature**

From a Language Model viewpoint, the innovative work of Hindle *et al.* [1] wherein they applied language models to programming languages, demonstrated that source code is far more regular and predictable than natural languages such as English. Language Models could be leveraged to improve upon the auto syntax completion functionality in Eclipse.

In another prominent work by Joshua *et al*. [3], they also used the same approach based on the n-gram language statistical model. This same approach has been more effective on source code than it is on natural language. They trained a model on compilable source code and then evaluated on new code to see the probability of new tokens in the model. The essence of the experiment was to check source code that did not compile which should be surprising to n-gram language model that compiled. And one of the areas where these techniques could be used is to produce plugin in IDEs such as eclipse to assist software engineers by coloring background of erroneous code lines.

Allamanis *et al*. [4] introduced first giga –token LM over source code corpus five times more than the corpus selected by Hindle *et al.* [1]. They presented a corpus of 14,807 java project comprising over 350 million lines of source code. They estimated that project with an average size introduced 56 original identifiers per KLOC. They trained over one billion tokens and found out that giga token model is much better at capturing the statistical properties of code than smaller scale model. They also found that method names are much more predictable than type and variable names as API calls are easy to predict.

Sheeva *et al.* [8] asserted to use natural language model while generating readable strings for test oracle to address the human oracle cost problem. The language model estimates the likelihood of how closely a string belongs to a language depending upon the character combination. They also found out that human oracle more quickly evaluates strings generated by language models in certain cases.

# Methodology

**Software Documentation**

StackOverflow is a fast-growing network of question and answer sites on diverse topics from software programming. Not only Programmers can ask and answer questions but also can vote on the quality of each post. Questions are tagged under categories and one question can have multiple tags therefore multiple categories assigned to it. In Order to perform the experiments, we downloaded the data dump in XML format released by StackExchange community.

For Software Documentation we chose StackOverflow posts as a corpus that we used to train language model and perform test on it. StackOverflow posts contain both text in the form of discussion, and code snippet that mostly are central to the discussion. StackOverflow posts tagged under lucene and android category were used as corpus for software documentation.

**Table -1:** StackOverflow posts used as software documentation corpus

|  |  |  |
| --- | --- | --- |
| Projects | Release Date | Post count |
| Lucene | June 05, 2013 | 2426 |
| Android | June 05, 2013 | 266988 |

**Extraction of Code Elements from Software Documentation**

Code elements are valid tokens or sequence of tokens that resemble legal programming language construct. It includes methods, classes, enums, annotations, keywords and other valid language constructs. we used an automated code resolution tool called ACE built by Rigby et al [2] to extract code elements from documents that contain free-form text as well as code fragments may have syntax errors or other compilation issues.

Software documents (lucene and android stack overflow posts), were fed to ACE to extract code elements. ACE provides an output file containing extracted code elements separated by space character for each post. Each post in the output file is a separate line. So we have sequence of tokens separated by space on each line. Schulam *et al.* [7] describes that it is the common data format language modeling toolkits. These output files were then used in experiments to train language model and estimating perplexity of extracted code elements.

**MITLM** the MIT Language Model package [5]

Language models are widely applied in natural language processing. There are many popular toolkits available including SRILM, MITLM, and RANDLM. Kenneth *et al* [9] describes that MITLM is mostly designed for accurate model estimation, but can also compute perplexity. He also does comparison of these tools. MITLM was chosen for estimating the perplexity and cross entropy of StackOverflow posts. Hindle *et al* [1], Nelson et al [4] used MITLM for training the model and computing perplexity (and further cross entropy).

**10-fold Cross-Validation**

Cross validation is a statistical method of evaluating learning algorithms by dividing data into two segments: one segment is used to learn or train model and the other segment is used to validate the model. In general sense, k-fold cross validation the data is first partitioned into k-equally sized folds. After that k iterations are performed of training and validations. Under each iteration, model is trained on k-1 data sets and tested on remaining data sets.

10-fold cross-validation is the most common approach in machine learning. In each of the 10 iterations data is divided into 10 folds so that model can be trained on 9 data sets and validated on 1 data set. Resultant values are averaged out over each 10 iteration.

**Plot**

We performed few experiments to compare the naturalness of software documentation with the code elements extracted from those documents. To look at the bigger picture we compared our results with the results found by Hindle *et al.* [1]. They compared the naturalness of English language with software code.

After loading the data dump, released by stackexchange, into the database, data files were formed using ACE for lucene and android projects. Two types of data files were generated using ACE tool. First type of file contained all the SO Posts (StackOverflow posts) and second one contained all the extracted code elements from those SO Posts. Data files for both android and lucene projects were generated.

**Cross-Entropy of documentation and code elements**

Cross entropy is a measure of how surprising a test documents is to a distribution model estimated from a corpus [1]. In all experiments we measured the cross entropy by averaging out over a 10-fold cross validation. We split the data files into 10 sets of size n/10. Trained the 9 data sets and tested on 1. Repeated 10 times and took the average.

# Findings and Discussion

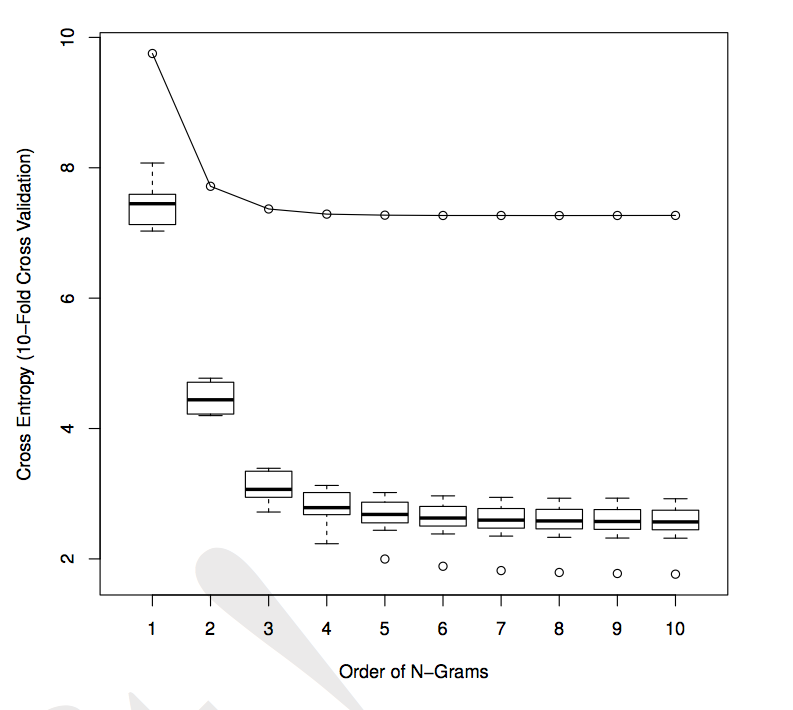
Figure 1: English Cross-Entropy versus the Code Cross Entropy from Hindle *et al.* [1]

Figure 2: Software Documentation Cross-Entropy versus Code-Element Cross Entropy

**RQ 1. Software documentation (and code element) is regular.**

Results in Figure 2 show that statistical model captures the regularities in both software documentation and code elements. We estimated n-gram models for 1 to 10 values of n over StackOverflow posts mentioned in the Table-1, using averages over 10-fold cross validation as described above.

**Observations from Figure 2:**

1. The blue line depicts the average over 10 folds for the software documentation, begins at about close to 9 bits and then trails down to less than 8 bits for 10 gram.

# The red line depicts the average over 10 folds for the code elements, begins at about 7 bits and then trails down to less than 5 bits for 10 gram.

# The cross entropy values for both software documentation and code elements declines with 1 and 2-gram and saturates after trigrams.

# Code elements are comparably more regular than software documentation. This observation is analogous to what we see in the Figure 1 which shows that software is far more regular than English text.

Observations from figure 2 provide evidence to the support the claim that people tend to ask and answer same question repeatedly or people learn in the same way.

**RQ 2. We expect Cross Entropy of the following to be in decreasing order.**

1. English text
2. Software documentation
3. Source code of API
4. Code Elements

**Figure 1 versus Figure 2**

Figure 1 shows that software is far regular than English text. We present a graph that shows that code elements are regular than software documentation. Results in both the figures are based on experiments that use statistical language model trained on the 90% of the corpus and validated on the 10% of the corpus. Since the experiments were conducted on the related projects e.g. lucene therefore the results from Figure 2 can be compared to Figure 1 and few important observations can be deduced.

1. Software documentation more often than not contains code snippets pertaining to the context and code snippets are more regular than English text therefore software documentation is comparably regular than English text. We support our claim by comparing first line of Figure 1 with blue line of Figure 2. Comparison shows that software documentation has lower cross entropy than English text.
2. Code elements extracted from software documentation are regular than English text and Software documentation. Software is far regular to English text, Software documentation and Code elements. We support our claim by comparing each line of two figures to every other line. The red line ranks third if we try to plot all the lines on one graph.

**Contradiction between Findings and Hypothesis:**

As per Figure 2, Code elements have higher cross entropy than software code.

**Reasoning on why code elements have higher cross entropy than software:**

Code Elements extracted from software documentation need not be a compilable unit of source code because people tend to ask questions about problems in the source code. Also, software documentation includes code snippets to illustrate certain properties of the code. Thus code elements will have higher entropy than software code. Joshua *et al*. [2] also describes that invalid source code will often have higher cross-entropy than compilable source code given a corpus of only syntactically valid source code. This argument is supported by the evidence that code elements have higher cross entropy of 5 bits but source code cross entropy saturates around 2 bits.

Moreover, syntactically invalid source code will often have a higher cross-entropy than compilable source code if the language model is trained on corpus of only syntactically valid source code elements. It is the reason why defective source code looks unnatural to a natural language model trained on compilable source.

# Conclusion

Though Natural languages are complex, people use them in a predictable manner owning to cognitive constraints. In this paper we use language models to capture the regularities in software documentation which is a product of natural languages such as English. For future work, we hope to extend this work further and use it in area such as code summarizations and code search.

# References

[1] "On the Naturalness of Software" Hindle.

[2] “Discovering Essential Code Elements in Informal Documentation” Rigby.

[3] “Syntax Errors Just Aren’t Natural” Joshua.

[4] “Mining Source Code Repositories” Allamanis.

[5] “MITLM, MIT Language Model Toolkit: http://code.google.com/p/mitlm/” [MITLM].

[6] “Two Decades of Statistical Language Modeling” Ronald.

[7] “Building Statistical Language Models of Code” Schulam.

[8] “Evolving Readable String Test Inputs Using a Natural Language Model to Reduce Human Oracle Cost” Sheeva.

[9] “Faster and Smaller Language Model Queries” Kenneth.

[10] “<http://stackoverflow.com/>”.

[11] “Markov Property of Markov Chains and Its Test” Zhang