ON THE NATURALNESS OF SOFTWARE

ABRAM HINDLE, EARL T. BARR, MARK GABEL, ZHENDONG SU, AND PREMKUMAR DEVANBU

BACKGROUND

THE EVOLUTION OF LANGUAGE

- Language evolved under noise, urgency, and efficiency -> simple, repetitive, and expedient patterns
- History of NLP:
  - 1960s -> rule based approaches
  - 1970s – 1980s -> logic and formal semantics
  - 1980s – present -> data-driven statistical approach
- Can NLP techniques be applied to programming languages?

BACKGROUND

LANGUAGE MODELS

- Probability distribution over a sequence of tokens
  - Lexical models
  - Syntactic models
  - Semantic models
- N-gram models:
  - $p(s)=p(a1)p(a2|a1)p(a3|a1,a2)...p(an|a1,...,an-1)$
- Markovian assumption:
  - $p(ai|a1,...,ai-1)\approx p(ai|ai-3,ai-2,ai-1)$
- Evaluation:
  - Cross-entropy
  - Perplexity

RESEARCH APPROACH

THE CORPORA USED

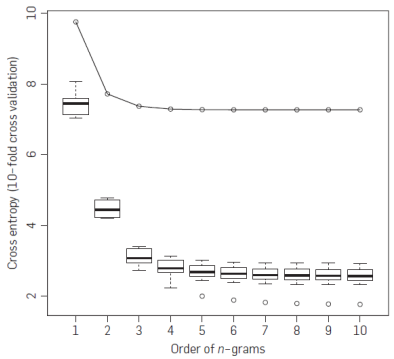
			Tokens	
Java Project	**Version**	**Lines**	**Total**	**Unique**
Ant	20110123	254457	919148	27008
Batik	20110118	367293	1384554	30298
Cassandra	20110122	135992	697498	13002
Eclipse-E4	20110426	1543206	6807301	98652
Log4J	20101119	68528	247001	8056
Lucene	20100319	429957	2130349	32676
Maven2	20101118	61622	263831	7637
Maven3	20110122	114527	462397	10839
Xalan-J	20091212	349837	1085022	39383
Xerces	20110111	257572	992623	19542

			Tokens	
Ubuntu Domain	**Version**	**Lines**	**Total**	**Unique**
Admin	10.10	9092325	41208531	1140555
Doc	10.10	87192	362501	15373
Graphics	10.10	1422514	7453031	188792
Interpreters	10.10	1416361	6388351	201538
Mail	10.10	1049136	4408776	137324
Net	10.10	5012473	20666917	541896
Sound	10.10	1698584	29310969	436377
Tex	10.10	1405674	14342943	375845
Text	10.10	1325700	6291804	155177
Web	10.10	1743376	11361332	216474

			Tokens	
English Corpus	**Version**	**Lines**	**Total**	**Unique**
Brown	20101101	81851	1161192	56057
Gutenberg	20101101	55578	2621613	51156

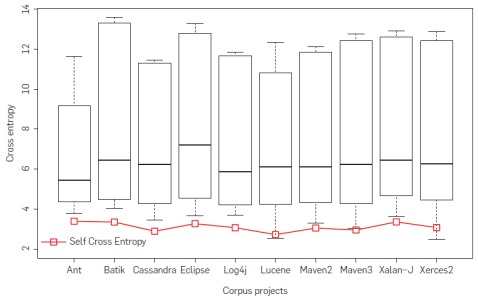
CODE VS NATURAL LANGUAGE

- Java has a **simpler and more structured syntax** compared to English.
- Regular patterns (e.g., function calls, loops) appear frequently.
- Variable and function names tend to repeat, unlike the vast vocabulary of English.



PROJECT-SPECIFIC REGULARITY

- **Hypothesis:** If Java's low entropy is purely due to syntax, then training a model on one Java project and testing on another should yield similar results.



DOMAIN-SPECIFIC REGULARITY

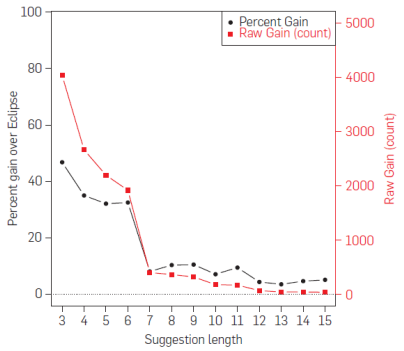
- Compare **within-domain entropy** (training and testing within the same domain).
- Compare **cross-domain entropy** (training on one domain, testing on another).
- Findings:
  - Projects within the same domain (e.g., networking applications) share common patterns.
  - Cross-domain entropy is **about 1 bit higher** than within-domain entropy.
  - Web applications have **especially low entropy**, suggesting more repetitive patterns.

TOKEN SUGGESTION FEATURE

- **Eclipse** is a popular IDE used for software development, which has an in-built suggestion engine
- Corpus-Based n-gram Model (NGSE):
  - A suggestion engine built using an n-gram model, which predicts the next token based on the two previous tokens entered.
  - Study used a trigram model
- Merging NGSE and ECSE:
  - Aims to balance the short suggestions from NGSE (n-gram model) and the longer suggestions from ECSE (Eclipse's built-in engine)

RESULTS

- The experiments were conducted on 5 different open-source projects: Ant, Maven, Log4J, Xalan, and Xerces.
- For each project, 40 files were set aside as a test set to evaluate the performance.



FUTURE WORK

- Improved language models
  - Smoothing techniques
  - Deep learning approaches
- Language models for accessibility
- Applications of machine translation
  - Code summarization
  - Code retrieval
- Software tools

POSITIVE POINTS

- The paper explores statistical language models for code, which laid the groundwork for tools like GitHub Copilot, CodeT5, and Codex. This early insight into "naturalness" in code could have helped inspire deep learning-based approaches that now dominate AI-assisted programming.
- The paper suggests that statistical models can capture different aspects of code structure (syntax, type, scope, and semantics). Modern approaches, such as transformer-based models, have indeed shown this to be true, validating the paper's hypothesis.
- Instead of just focusing on autocompletion, the paper explores applications in error detection, accessibility, and machine translation of code, which are still relevant research directions.
- The idea of using statistical methods to approximate expensive static analysis tools is insightful. Even today, researchers are trying to balance soundness with efficiency in static analysis tools.

NEGATIVE POINTS

- The paper briefly mentions deep learning as a possible future direction, but by 2016, neural networks were already proving effective for natural language tasks.
- The primary approach discussed is based on n-gram language models, which are effective for local patterns but struggle with long-range dependencies.
- The paper does not address issues like bias in training data, security risks, or ethical concerns about AI-generated code.
- The paper does not address the computational cost of training and using such models in real-world software engineering workflows.

RATING

4/5

DISCUSSION

- Given the paper's focus on statistical approaches, are there cases where simple statistical methods might still be preferable to deep learning models?
- The trade-off between speed and code quality in AI-assisted programming
- Could AI coding tools reinforce biases in software?
- AI-driven software maintenance: who will maintain AI-written code?
- Is "naturalness" in code always a good thing?