Naturalness of Code

CS454 AI-Based Software Engineering

Shin Yoo

What is “natural” about language?

- Natural language refers to ordinary languages that occur naturally in human community “by process of use, repetition, and change without conscious planning of premeditation” (Wikipedia)
- From the statistical point of view, it means that most of our utterances are simple, repetitive, and therefore predictable.
  - Surely this is how we all learn language.

3:19
duolingo
LTE
5
Complete the sentence
¡Lo siento! ¡ !
Agua
Leche
Perdón
Pan
CONTINUE

3:20
duolingo
LTE
5
HARD EXERCISE
Complete the sentence
Yo inglés. ¿Tú
español?
hablas
hablo
CONTINUE

3:21
duolingo
LTE
5
HARD EXERCISE
Complete the sentence
Yo español. ¿
Tú español?
hablo
hablas
CHECK

John: Hi, nice to meet you. How are you?

Mary: I'm ____, _____ ___. ___ ___?

a) fine, thank you. And you?
b) okay, I guess. Why care?

Do yuo fnid tihs smilpe to raed? Bceuase of the phaonmneal pweor of the hmuan mnid, msot plepoe do.

What about code?

- It is not “natural”, in the sense that we have artificially created the grammar for programming languages.
  - Programming languages do evolve, but how?
    - Intentionally? New grammars, language consortiums, etc…
    - Gradually? Languages do affect each other, a newer and more popular style eventually gets accepted, etc…

Python: for _ __ _____ …

a) i in range
b) ??

Java: for _ ___ _ _ _ _ …

a) (int i = 0;
b) ??

On the Naturalness of Software

Hindle et al., ICSE 2012

- “Programming languages, in theory, are complex, flexible and powerful, but the programs that real people actually write are mostly simple and rather repetitive, and thus they have usefully predictable statistical properties that can be captured in statistical language models and leveraged for software engineering tasks.”

Language Model

- Given a set of tokens, $\mathcal{T}$, a set of possible utterances, $\mathcal{T}*$, and a set of actual utterances, $\mathcal{S} \subset \mathcal{T}$, a language model is a probability distribution p over utterances $s \in \mathcal{S}$, i.e., $\forall s \in \mathcal{S}[0 < p(s) < 1 \wedge \sum_{s \in \mathcal{S}} p(s) = 1$
    - That is, given all possible sequences of tokens, $\mathcal{T}*$, how likely is it that someone says a specific sentence, $s \in \mathcal{S}$?

Probability of Utterance

- An utterance (or a sentence) is a sequence of tokens (or words). Suppose we have N tokens, $a_1, a_2, \ldots, a_N$ that consist s. What is $p(s)$?
  - $p(s) = p(a_1)p(a_2 \;\; a_1)p(a_3 \;\; a_1 \, . \, a_2)p(a_4 \;\; a_1, a_2, a_3)\ldots p(a_N \;\; a_1 \ldots a_{N-1})$
  - But these conditional probabilities are hard to calculate: the only feasible approach would be count each utterance that qualifies, but $\mathcal{S}$ is too big, let alone $\mathcal{T}*$.

N-Gram

- Assumes Markov property, i.e., the next token is influenced only by those came immediately before (say, within the window of n tokens)!
  - $p(a_i \;\; a_1 \dots a_{i-1}) \simeq p(a_i \;\; a_{i-3}a_{i-2}a_{i-1})$
- This is now much more tractable:
  - $p(a_i \;\; a_{i-3}a_{i-2}a_{i-1}) = \dfrac{count(a_{i-3}, a_{i-2}, a_{i-1}, a_i)}{count(a_{i-3}, a_{i-2}, a_{i-1}, *)}$

How surprising a sentence is…

- Given a language model $\mathscr{M}$ and a sentence s, we can define the entropy of the sentence:

- $H_{\mathscr{M}}(s) = -\frac{1}{n}\log p_{\mathscr{M}}(a_1 \ldots a_n)$

- Under the n-gram model:

- $H_{\mathscr{M}}(s) = -\frac{1}{n}\log p_{\mathscr{M}}(a_1 \ldots a_n) \simeq -\frac{1}{n}\sum_{1}^{n}\log p_{\mathscr{M}}(a_i \quad a_{i-3}, a_{i-2}, a_{i-1})$

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

Table I: 10 Java Projects, C code from 10 Ubuntu 10.10 Categories, 3 English Corpus used in our study. English is the concatenation of Brown and Gutenberg. Ubuntu 10.10 Maverick was released on 2010/10/10.

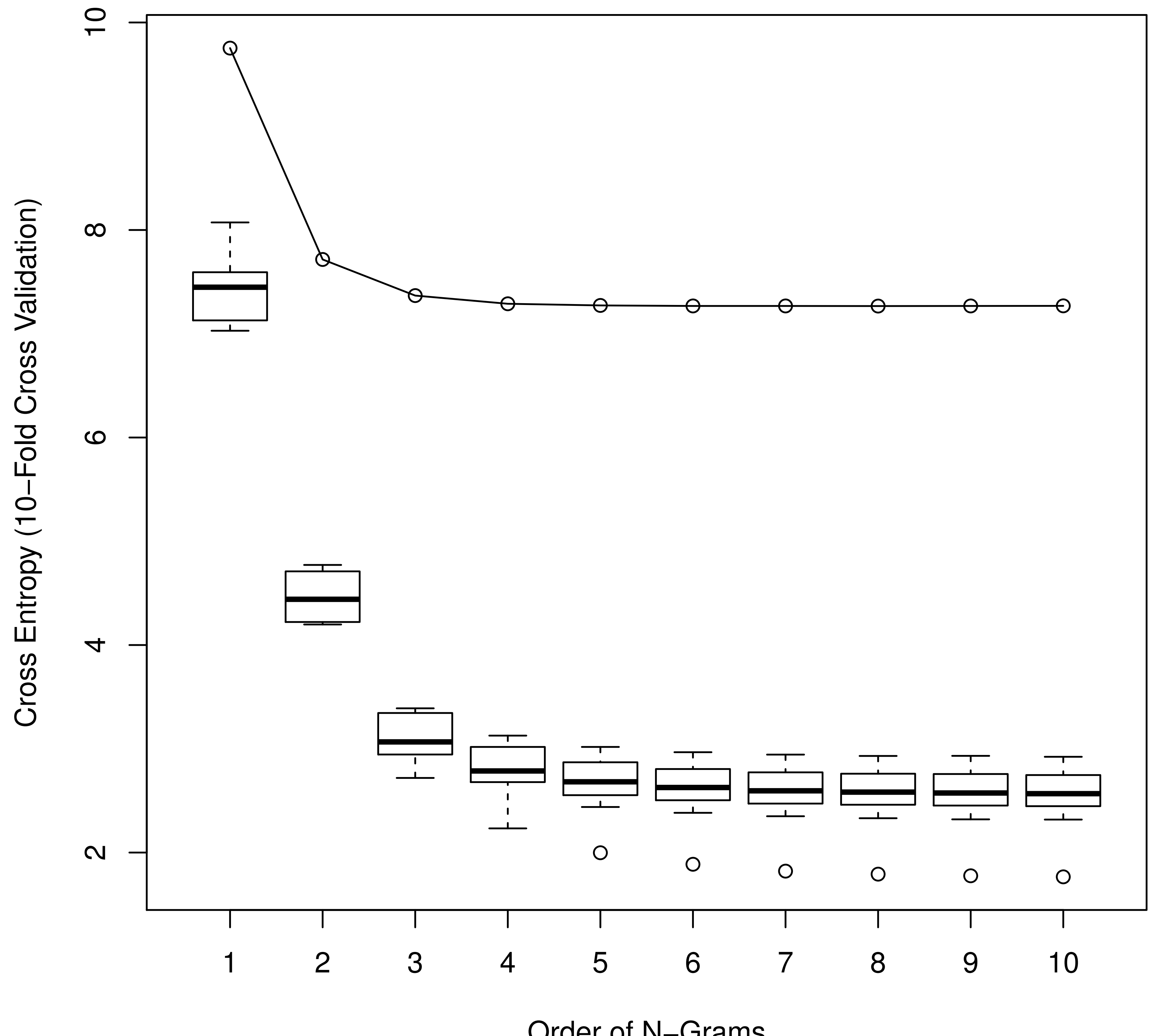


Figure 1: Comparison of English Cross-Entropy versus the Code Cross Entropy of 10 projects.

A Hands-on for Natural Language

- N-gram Language Model with NLTK: https://www.kaggle.com/code/alvations/n-gram-language-model-with-nltk
  - NLTK is a famous NLP toolkit for Python
  - We will leave this for you to try later

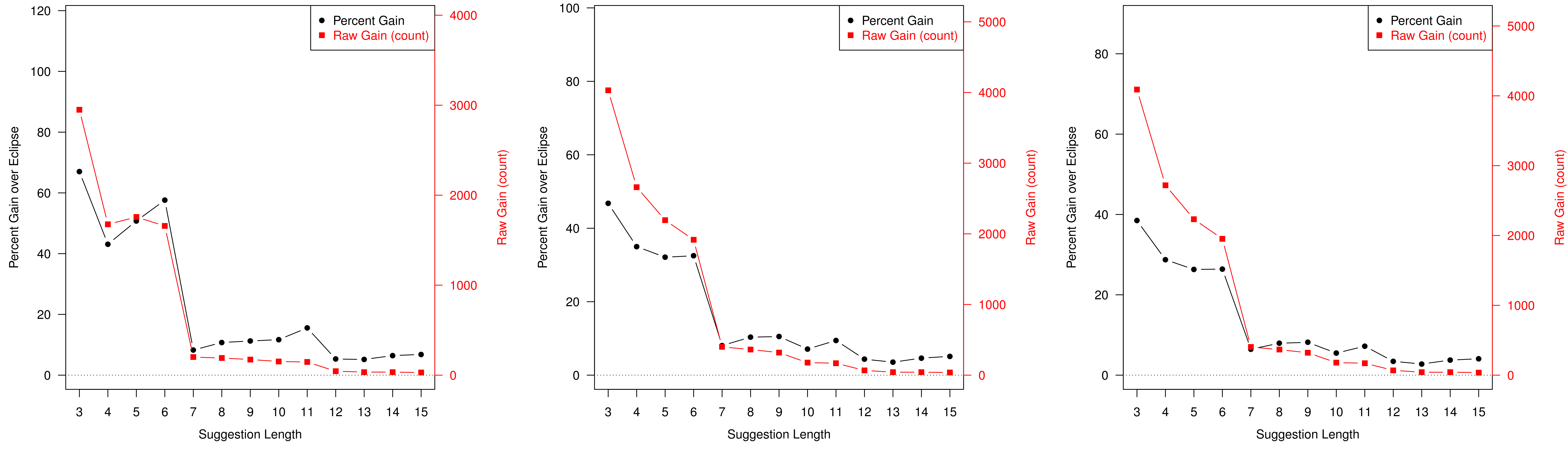
A Hands-on for Python Code

- We will use Python corpus from CodeSearchNet (https://github.com/github/CodeSearchNet)
- The hands-on script is available from: https://github.com/coinse/cs454-ngrams

So what can you do with this?

(On practical level) Autocompletion

- Given a few preceding tokens, you can quickly compute the likelihood of a specific token to follow the given tokens: autocompletion!
- In practice, there are much more extra information on top of ngram analysis if we want to do implement autocompletion within an IDE. What?
  - Type
  - Variable scope
  - Vocabulary <— (we will come back to this)



(a) Gain using top 2 suggestions.

(b) Gain using top 6 suggestions.

(c) Gain using top 10 suggestions.

Figure 4: Suggestion gains from merging n-gram suggestions into those of Eclipse.

(On curiosity level) What is the usual entropy?

- In other words, do we write unique code, or whatever we write are usually boilerplates, repetitive, expectable?
- Interestingly, a predecessor of the Naturalness paper asked this first: “A Study of the Uniqueness of Source Code” by Gabel and Su, FSE 2010
    - After abstracting unique identifiers for matching, you have to write over 30 tokens to get unique in general: this is about 5~6 lines of code.
    - A single line (=6 tokens) is almost always redundant (i.e., the same line can be found in the same program)

		Median Syntactic Redundancy (%)					
			Max Hamming Dist:				
	g	**Abstraction**	0	1	2	3	4
C	6	None	63.3	74.8	88.4	96.7	99.9
		Renamed IDs	98.3	98.7	99.0	99.6	99.9
	20	None	7.8	14.0	23.6	34.8	49.9
		Renamed IDs	59.5	79.6	90.8	96.1	98.5
	35	None	4.1	5.5	7.2	9.1	11.1
		Renamed IDs	14.8	19.5	25.0	30.8	37.3
	77	None	2.0	2.4	2.7	3.1	3.4
		Renamed IDs	4.5	5.0	5.6	6.0	6.5
	120	None	1.4	1.6	1.8	1.9	2.0
		Renamed IDs	2.7	2.9	3.1	3.2	3.4
C++	6	None	54.5	68.9	84.8	95.8	99.8
		Renamed IDs	97.9	98.5	99.2	99.8	100.0
	20	None	3.2	7.8	15.1	25.2	39.3
		Renamed IDs	48.1	68.2	83.6	92.4	96.9
	35	None	0.9	1.5	2.4	3.6	5.3
		Renamed IDs	9.8	13.3	18.0	22.4	27.8
	77	None	0.1	0.3	0.3	0.5	0.6
		Renamed IDs	1.6	1.8	2.1	2.3	2.6
	120	None	0.0	0.0	0.1	0.1	0.1
		Renamed IDs	0.7	0.8	0.9	0.9	1.0
Java	6	None	69.5	81.0	92.9	98.5	99.9
		Renamed IDs	98.2	98.5	98.8	99.5	99.9
	20	None	9.6	18.1	30.5	45.9	63.5
		Renamed IDs	72.2	88.1	95.4	98.1	99.2
	35	None	3.9	5.6	8.0	10.8	14.1
		Renamed IDs	23.0	30.4	39.7	48.5	56.5
	77	None	1.8	2.2	2.6	2.9	3.3
		Renamed IDs	4.9	5.3	5.9	6.4	7.0
	120	None	1.3	1.5	1.7	1.8	1.9
		Renamed IDs	2.6	2.9	3.1	3.3	3.5

Table 4: Median syntactic redundancy values for the 6,000 corpus projects.

Many related questions

- Does correct code have lower entropy, and vice versa? (Assuming that LM is trained on the whole corpus of code, and that there are more correct code than incorrect ones)
  - If so, can we use LM to write/fix code?
- Is buggy code unnatural??
  - If so, can we detect them simply by computing how unique they are?

Dual Channel Constraints

Casalnuovo et al., ICSE NIER 2020

- Next iterative refinement of the naturalness idea :)
- Source code communicates meaning over two different channels:
    - AL (Algorithmic) Channel: a human tells a computer what to do —> the semantic is complied into machine instructions and eventually executed
    - NL (Natural Language) Channel: a human tells **other humans** what the source code does —> others can read the code and understand
- Importantly, each channel constraints what is allowed in the other!
    - No one will use random variable names; no one will name a function `quick_sort` if the code actually implements insertion sort algorithm

An example of exploiting the dual channel

- Predictive Mutation Analysis (Kim et al., TOSEM 2022): which tests can kill this mutant? Can we predict without actually executing tests at all?
  - Concrete analysis based on AL channel: actually executes the test against the mutant, and observe the execution results - if the results are different from the results obtained from the original program, mutant is killed
  - Learnt analysis based on NL channel: based on previous concrete analysis, we try to learn the connection between vocabularies of mutants and tests that kill them

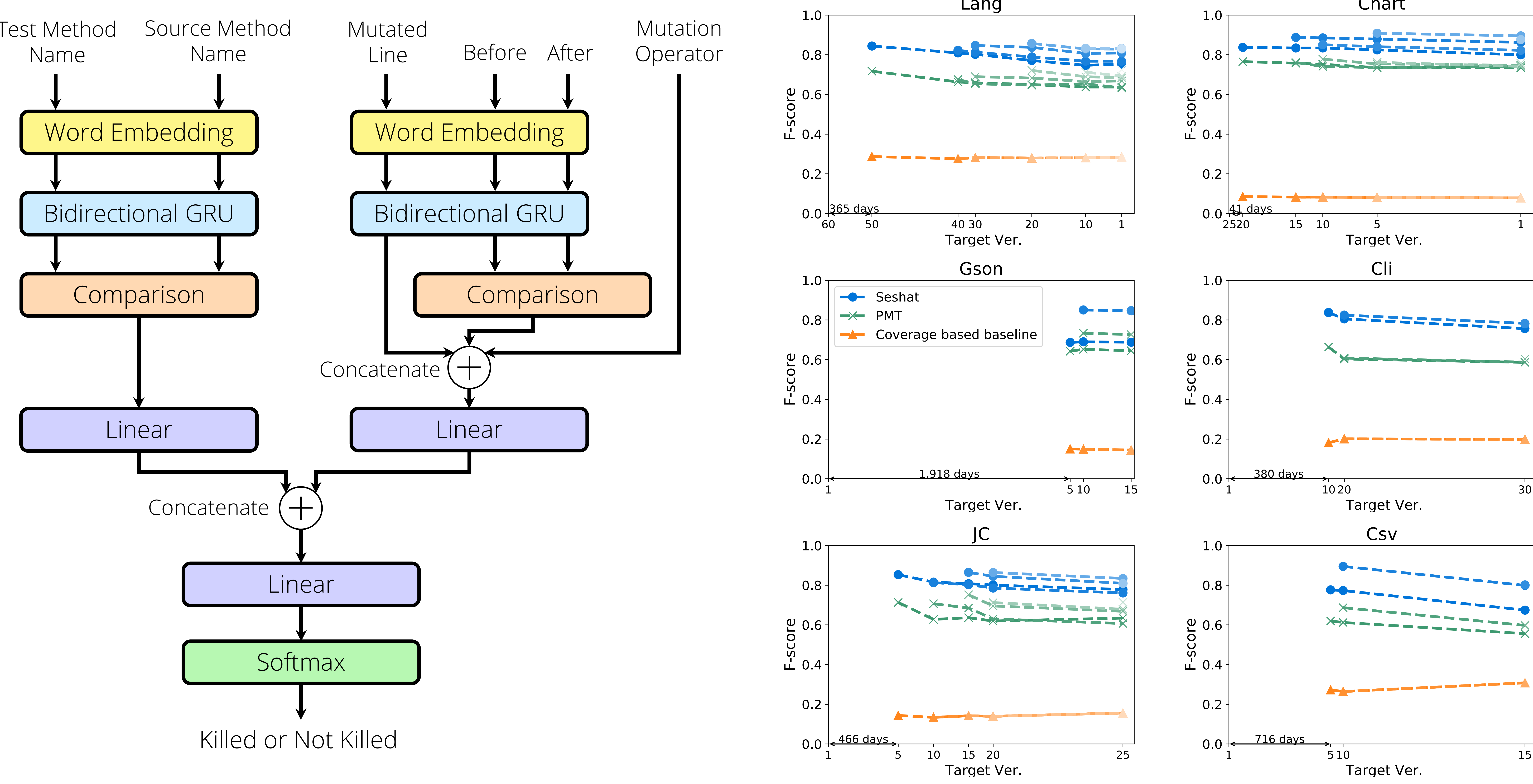


Fig. 3. Model architecture of Seshat

Fig. 4. Prediction of the full kill matrix on Major

Summary

- Statistical view of source code can help certain task a lot.
- N-gram is computationally attractive way of computing a Language Model (LM).
- But if we scale up LMs then we get… :)