CS846
Machine Learning for Software Engineering

Pengyu Nie

Software Engineering Datasets and Metrics

Datasets/Benchmarks: Huggingface; HumanEval & MultiPL-E

Metrics: CodeBLEU; Pass@K

HuggingFace Datasets

- "Standard" repository for datasets
- Download datasets collected by others
  - https://huggingface.co/datasets
  - Full-text serach: https://huggingface.co/search/full-text?type=dataset
- Upload your collected/processed datasets
  - https://huggingface.co/docs/datasets/upload_dataset

Training Datasets

- The Stack: https://huggingface.co/datasets/bigcode/the-stack-v2
  - Essentially all open-source code from GitHub
  - You likely want a (small) subset of it for fine-tuning
- Glaive code assistant:
  https://huggingface.co/datasets/glaiveai/glaive-code-assistant-v3
- Magicoder:
  https://huggingface.co/datasets/ise-uiuc/Magicoder-Evol-Instruct-110K
  https://huggingface.co/datasets/ise-uiuc/Magicoder-OSS-Instruct-75K
  - Mostly for instruction fine-tuning
- Processing existing datasets / GitHub code
  with parsing, static analysis, dynamic analysis… (next few lectures)

Benchmark: HumanEval

- Dataset: https://huggingface.co/datasets/openai/openai_humaneval
- Benchmark scripts: https://github.com/openai/human-eval
- Introduced with the Codex model
- Does not involve human during computing evaluation metrics

prompt

```
from typing import List
def concatenate(strings: List[str]) -> str:
"""
Concatenate list of strings into a single string
>>> concatenate([])
''
>>> concatenate(['a', 'b', 'c'])
'abc'
"""
```

solution

```
return ''.join(strings)
```

tests

```
def check(candidate):
  assert candidate([]) == ''
  assert candidate(['x', 'y', 'z']) == 'xyz'
  assert candidate(['x', 'y', 'z', 'w', 'k']) == 'xyzwk'
```


Chen et al. 2021. Evaluating large language models trained on code. https://arxiv.org/pdf/2107.03374

Benchmark: MultiPL-E

- Dataset: https://huggingface.co/datasets/nuprl/MultiPL-E
- Benchmark scripts: https://github.com/nuprl/MultiPL-E
- HumanEval, but in multiple (19) programming languages

prompt

```
/// Concatenate vector of strings into a single string
/// >>> concatenate(vec![])
/// String::from("")
/// >>> concatenate(vec![String::from("a"), String::from("b"), String::from("c")])
/// String::from("abc")
fn concatenate(strings: Vec<String>) -> String {
```

solution

```
  strings.concat()
```

tests

```
} fn main() {
  let candidate = concatenate;
  assert_eq!(candidate(Vec::<String>::new()), String::from(""));
  assert_eq!(
    candidate(vec![String::from("x"), String::from("y"), String::from("z")]),
    String::from("xyz"));
  assert_eq!(
    candidate(vec![String::from("x"), String::from("y"), String::from("z"),
      String::from("w"), String::from("k")]),
    String::from("xyzwk"));
}
```

Demo (tutorial):
https://nuprl.github.io/MultiPL-E/tutorial.html

Cassano et al. TSE 2023. MultiPL-E: a scalable and polyglot approach to benchmarking neural code generation. https://arxiv.org/pdf/2208.08227

Evaluation Metrics

- $\text{metric}(\text{gold}, \text{prediction}) \in [0, 100]$
- Syntactic similarity
  - Exact match (EM, XMatch)
  - BLEU
  - CodeBLEU
  - Edit similarity
- Functional correctness
  - Compile%
  - Pass%
  - Pass@K

BLEU

- Designed for natural language
- Percentage of n-gram overlaps
- Brevity penalty

$$p_n = \frac{|\text{ngram co-occuring in g and p}|}{|\text{ngram in g}|}$$

$$BP = \begin{cases} 1 & if\ |p| > |g| \\ e^{1-|g|/|p|} & if\ |p| \leq |g| \end{cases}$$

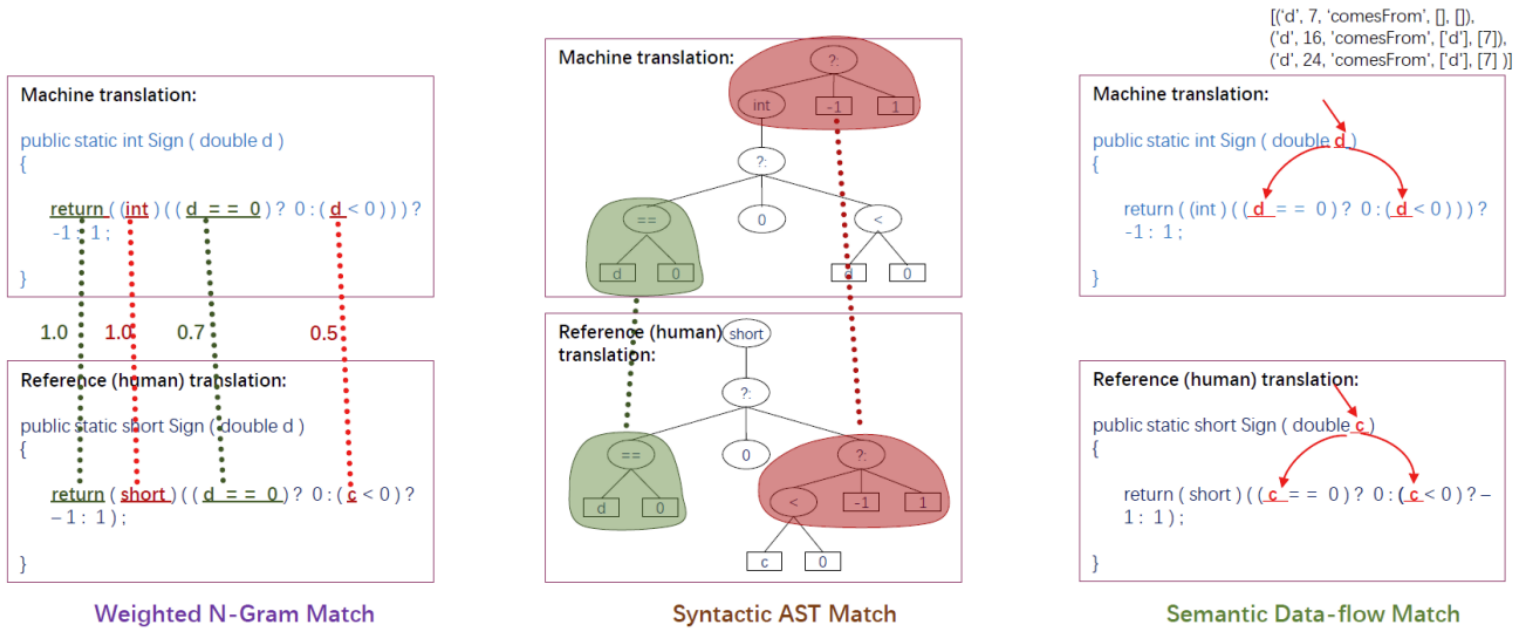
$$\text{BLEU} = BP \cdot \exp \sum_{n=1}^{N} w_n \ln p_n$$

Usually $N = 4, w_1 = w_2 = w_3 = w_4 = 0.25$

- Implementation: nltk
  https://www.nltk.org/api/nltk.translate.bleu_score.html

CodeBLEU

- Considers keywords, AST, and data-flow in code



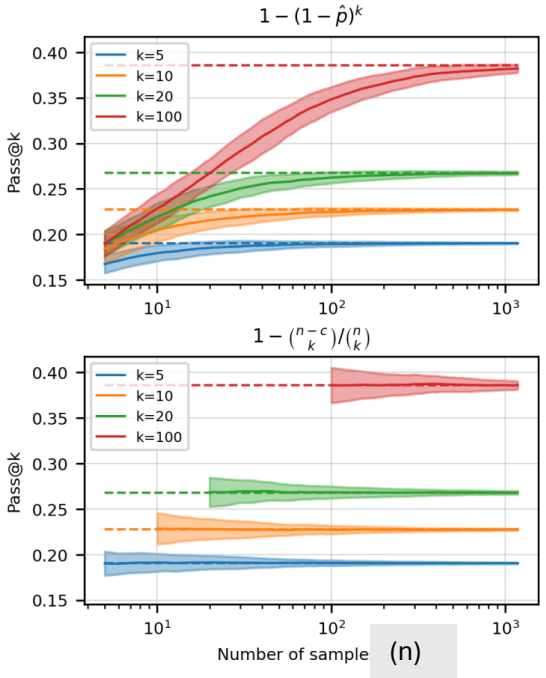
$$\text{CodeBLEU} = \alpha \cdot \text{N-Gram Match (BLEU)} + \beta \cdot \text{Weighted N-Gram Match} + \gamma \cdot \text{Syntactic AST Match} + \delta \cdot \text{Semantic Data-flow Match}$$

- Implementation: https://github.com/microsoft/CodeXGLUE/tree/main/Code-Code/code-to-code-trans/evaluator

Ren et al. ACL 2021. CodeBLEU: a method for automatic evaluation of code synthesis. https://arxiv.org/pdf/2009.10297

pass@k

- Generate n predictions by sampling decoding, the probability of at least one prediction pass the tests when selecting k predictions
  - n: hidden parameter, n >> k (e.g., n = 200 and k = 1, 10, 100)
- What if…
  - n = k? -> high variance
  - Use pass@1 to estimate pass@k?
    $pass@k = 1 - (1 - pass@1)^k$ -> underestimate



Chen et al. 2021. Evaluating large language models trained on code. https://arxiv.org/pdf/2107.03374

pass@k

- Generate n predictions by sampling decoding, the probability of at least one prediction pass the tests when selecting k predictions
  - n: hidden parameter, n >> k (e.g., n = 200 and k = 1, 10, 100)
- Unbiased estimation:

$$Pass@K = 1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}$$

$$= 1 - \frac{\frac{(n-c)!}{(n-c-k)!\,k!}}{\frac{n!}{k!\,(n-k)!}} = 1 - \frac{(n-c)!\,(n-k)!}{n!\,(n-c-k)!}$$

$$= 1 - \frac{(n-c+1-k)\cdot(n-c+2-k)\cdot\ldots\cdot(n-1-k)\cdot(n-k)}{(n-c+1)\cdot(n-c+2)\cdot\ldots\cdot(n-1)n}$$

$$= 1 - \prod_{i=1}^{c}\frac{n-c+i-k}{n-c+i}$$

$$= 1 - \prod_{i=1}^{c}\left(1-\frac{k}{n-c+i}\right)$$

```
def pass_at_k(n, c, k):
    """
    :param n: total number of samples
    :param c: number of correct samples
    :param k: k in pass@$k$
    """
    if n - c < k: return 1.0
    return 1.0 - np.prod(1.0 - k /
        np.arange(n - c + 1, n + 1))
```


Chen et al. 2021. Evaluating large language models trained on code. https://arxiv.org/pdf/2107.03374