教師ラベルなし単言語学習データのみでの cross-language コードクローン検出の試み

劉 宇澤　千葉 滋

近年，同じプロジェクトでも複数の言語に関わることが増え，教師あり学習で cross-language コードクローンを検出する手法が研究され始めている．しかしながら，教師あり学習はデータセットの質に依存するため，良いデータセットが存在しない今の状況では，実用に応用するにはまだ現実ではない．そこで，本研究は教師データが必要ない教師なし学習での cross-language コードクローンの検出を試みた．まだ研究されたことのないタスクであるため，その可能性を追求するに，二つの仮説を立てた．そして，二つの仮説の元に，LSTM を使った autoencoder や Transformer を使った言語モデルとそれぞれの改善法合計五つの手法について実験した．結果はかなり高い精度で cross-language のコードクローンを検出することができた．

1　はじめに

コードクローンはソフトウェアの開発および保守に悪影響を与える恐れがある．同一言語でのクローン検出はたくさん研究されている [2] [13] [18] [12] [5] [19]．近年，同じプロジェクトでも複数の言語に関わることが増えている．こういう背景の中に，cross-language コードクローンを検出する手法が研究され始めている [14] [1] [4] [21] [17] [16]．そして昨年に，教師データを使って，教師あり学習で cross-language コードクローンを検出する論文が二つ出た [17] [16]．しかしながら，教師あり学習はデータセットの質に依存するため，良いデータセットが存在しない今の状況では，実用に応用するにはまだ現実ではない．

そこで，本研究は教師データが必要ない教師なし学習での cross-language コードクローンの検出を試みた．まだ研究されたことのないタスクであるため，その可能性を追求するに，二つの仮説を立てた．一つ目は，自然言語と違い，プログラミング言語は違う言語でもたくさんの共通点があり，これらの共通点をもとに，言語の間の違い部分もベクトル空間上の近い位置に学習できるという仮説である．二つ目は，プログラミング言語は分布仮説に従っていて，英語ベースで付けられた変数名や関数名は隠しラベルの役目を果たす．

学習データとして GitHub[†1] からアルゴリズムに関係する Java と Python のコードを集め，LSTM [11] を使った autoencoder [10] や Transformer [20] を使った言語モデルとそれぞれの改善法合計五つの手法について実験した．評価は LeetCode[†2] の解答を使って行った．結果はかなり高い精度で cross-language のコードクローンを検出することができた．

2　背景

コードクローンはソフトウェアの開発および保守に悪影響を与える恐れがある．同一言語でのクローン検出はたくさん研究されている．

近年，同じプロジェクトでも複数の言語に関わることが増えている．例えば，プロトタイプコードとプロダクトコードは違う言語を使うことがある．プロトタイプコードは Python などのスクリプト言語を使い，実際のプロダクトコードは C++や Java などの

Yuze Liu, 東京大学大学院情報理工学系研究科, Graduate School Information and Science Technology, The University of Tokyo.

†1 https://github.com

†2 https://leetcode.com

速い言語で実装する．また，アプリケーションが動く環境によって，使える言語が限られている．同じ機能のソフトウェアでも，iOS アプリケーションは Swift や Objective C を使い，Android アプリケーションは Java や Kotlin で実装し，ブラウザアプリケーションは JavaScript で書く．

こういう背景の中に cross-language のコードクローンが存在する可能性が増えている．cross-language のコードクローンでもソフトウェアの開発に悪影響を与える恐れがあるから，検出することが大切だ．

例として，同じ機能のソフトウェアが複数言語で実装される場合は，一つ言語のコードが修正される時，他の言語の同じ部分も修正される必要がある．多言語のプロジェクトは通常複数のチームで協力して開発ことが多い．開発者は自分が専門している言語の部分だけ開発している．一つ言語のコードが修正された時，開発者は自分が詳しくない別の言語のコードからコードクローンを探し出す必要がある．システムアーキテクチャや別の言語の予備知識を知る必要があるため，同一言語のコードクローンよりも時間消費やコストが大きい．

また，ウェブアプリケーションはマイクロサービスアーキテクチャを採用することがある [17]．サービスごとに違うチームで違い言語を使って開発するため，サービスの間に機能的に重複するコードが存在する可能性がある．これらのコードは単一責任原則を違反しているから，cross-language コードクローンを検出する技術を使って，避けるべきだ．

Cross-language コードクローンの検出は研究され始めている．そして，昨年に，教師データを使って，教師あり学習で cross-language コードクローンを検出する論文が二つ出た [17] [16] が，あまり性能が良くない．論文は二つとも教師データとして，AtCoder[†3] や Google CodeJam[†4] などの競技プログラミングのデータを使って，学習した．競技プログラミングのコードはコーディングスタイルや解決方法に差が大きすぎるため，良いデータではない．そして，教師あり学習は学習されていない他の領域のパターンに弱い．競技プログラミングで学習されたモデルは，競技プログラミングのコードに使えるが，実用的の領域には使えない．実用的の領域にはまだ学習データが存在していないため，実用に応用するにはまだ現実ではない．

†3 https://atcoder.jp

†4 https://www.go-hero.net/jam/10/languages/0

3 教師なし学習による cross-language コードクローン検出

教師ラベルなし単言語学習データのみを使った教師なし学習による cross-language コードクローン検出を提案する．

教師なし学習による cross-language コードクローン検出の基本的考えは，教師データ不要のあるタスクをモデルに学習させて，その副産物としてコードの分散表現が得られる．コードの分散表現のコサイン類似度を計算して，コードクローンを判断する．

教師なし学習は教師あり学習に比べて，主に以下三つの利点がある：

1） 性能に期待できる．教師あり学習は教師データの質に依存するため，まだ良い教師データが存在しない今の状況では，性能は高くない．一方，教師ラベルなしの単言語データはたくさん存在する．教師なし学習の精度が期待できます．

2） 汎用性に期待できる．教師あり学習は学習されたパターンに強く，学習されていない他の領域のパターンに弱い．今までの論文は競技プログラミングの教師データを使って学習した．実用的の領域にはまだ学習データが存在していないため，実用に応用することはできない．一方，教師ラベルなしの単言語データはどの領域でもたくさん存在している．実用にすぐ応用できる．

3） モデルに通す計算量が低い，大規模の検出にも応用できる．教師あり学習はコードクローンのペアをインプットとして，クローンかどうかをアウトプットする．検出候補のコードを二つずつ組み合わせてモデルに通すため，モデルに通す計算量は $O(n^2)$．教師なし学習はコードを一つずつモデルに通して，コードの分散表現をアウトプットする．後は分散表現のコサイン類似度を計算して，コードクローンかどうかを判断する．モデルに通す計算量は $O(n)$ である．モデル

に通す時間コストはコサイン類似度を計算するより遥かに大きいため，モデルに通す計算量を抑えると，検出全体の時間は大幅に抑えられる．大規模のコードクローン検出にも応用できる．

教師なし学習による cross-language コードクローン検出は我々が知る限り，まだ研究されたことのないタスクであるため，その可能性を追求するに，二つの仮説を立てた．

3.1 仮説 1

自然言語と違い，プログラミング言語は違う言語でもたくさんの共通点があり，これらの共通点をもとに，言語の間の違い部分もベクトル空間上の近い位置に学習できる．

自然言語における cross-language タスクは，主に二つの言語を関連付けた教師データを使い，教師あり学習を行う．例えば，言語翻訳タスクは，事前に与えられた二つの言語の翻訳文ペアを学習データとして使い，そうすることで二つの言語を関連付ける．自然言語では，言語によって使う文字や文法は全然違うため，教師データをなしに二つの言語を関連付けるのは難しい．だが，プログラミング言語はそうではない．プログラミング言語は違う言語でもたくさんの共通点がある．特に，汎用的な言語は共通点が多い．

共通点の例として：

- 入れ子構造がある
- 式を使ってデータ処理をする
- if, for, while, function, class などの statement がある
- 英語ベースで変数名や関数名を付ける
- AST に変換して，実行する

教師なし学習では，トークンの意味をベクトル化する．言語の間の共通点をもとに，違う言語のベクトル空間は独立なものでは無くなって，関連付けられる．言語の間の違い部分もベクトル空間上の近い位置に学習できる．

3.2 仮説 2

プログラミング言語は分布仮説に従っていて，英語ベースで付けられた変数名や関数名は隠しラベルの役目を果たす．

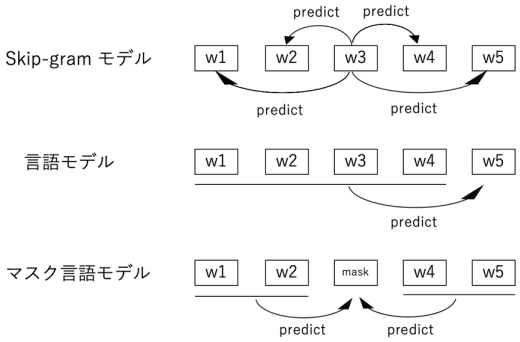


図 1　教師なし学習手法

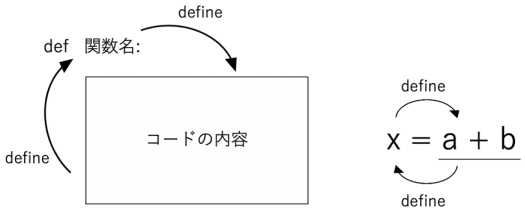


図 2　プログラミング言語の形

分布仮説とは，同じ文脈で出現する単語は意味も類似する [9]．言い換えれば，単語の意味は周辺文脈によって定義される [7]．分布仮説は統計的意味論 [22] の基礎である．

分布仮説は近年発展している教師なし学習手法と強く関係していると考える．図 1 に示した通り，例えば，skip-gram モデル [15] は，単語をインプットとして，その単語の周りの単語を予測する．言語モデルは，今までの文脈をインプットとして，次の単語を予測する．マスク言語モデル [6] は，マスクされていない単語をインプットとして，マスクされた単語を予測する．

プログラミング言語は，形から見ると，分布仮説に従っていると思われる．図 2 に示した通り，例えば，関数を定義する時，関数名を書いてから，後ろにコードの内容を書く．関数名の意味は後ろのコードによって定義される．そして，後ろのコードの意味も関数名によって定義される．式の場合も同じ．例えば，x = a + b の場合は，x の意味は a + b によって定義される．a + b の意味もまた x によって定義される．

違う言語でも，同じく英語ベースで関数名や変数名を付けている．そして，機能的に近いコードは，関数名や変数名も関係している．分布仮説が成立すれば，変数名や関数名は隠しラベルの役目を果たす．これで，教師なし学習が可能になる．

4 手法と実験結果

LSTM を使った autoencoder や Transformer を使った言語モデルとそれぞれの改善法合計五つの手法について実験した．4.1 は学習データ，評価データ，データの前処理について紹介する．4.2 から 4.6 まではそれぞれの手法と実験結果について紹介する．4.7 は実験結果について議論する．

4.1 データセット

4.1.1 学習データ

学習データは GitHub から集めた．GitHub から LeetCode や algorithm のキーワードを含めたリポジトリを Java と Python それぞれ 550 個を集めた．長さ 512 トークン以上のファイルを取り除いて，Java は 123623 ファイル，Python は 83398 ファイルを学習データとして使用した．

LeetCode[†5] は IT 企業のコーディング面接を練習するためのサイトである．LeetCode を学習データとして選んだ理由は，競技プログラミングと違って，LeetCode は時間を競うではなく，コーディングスタイルを重視している．LeetCode は競技プログラミングより，実際の仕事現場で書いたコードに近いである．

4.1.2 評価データ

評価データは LeetCode の解答を使う．同じ問題を解く Java と Python のコードは cross-language コードクローンと見なす．103 個の問題にそれぞれ一つ Java の解答と一つ Python の解答がある．組み合わせて，全部 10609 ペアの中に 103 ペアだけの正解ペアを検出するというタスクになる．実際の実用の時も，コードクローンの数に対してコードの種類数が遥かに多いである．そのため，たくさん種類のコードの中に，数少ないコードクローンを見つかられるかどうかを評価する．

4.1.3 BPE(byte pair encoding)

BPE(byte pair encoding) [8] はデータ圧縮法の一つである．自然言語では，BPE を使って，非頻出語をサブワードに分けることで未知語を対応することがある．

BPE は特にプログラミング言語に適していると考える．プログラミング言語は開発者が自分で名前を付けるため，未知語は避けられない．今までの研究は非頻出語を identifier などの型に変えることが多い．しかし，そうすると多くの情報が損失することになる．プログラミング言語で名前を付ける時よくキャメルケースやスネークケースを使って意味を表現する．BPE を使えば，上手く名前を意味のあるサブワードに分けられるではないかと思って，実験をした．結果は，ほとんどの変数名や関数名は意味のあるサブワードに分けることができた．

学習データに BPE を使うことにした．Java と Python のコードの語彙数は合わせて 157347 である．BPE を使って，30829 にした．

4.2 手法 1

LSTM [11] を使った autoencoder [10] について実験した．図 3 で示した通り，コードを LSTM encoder に入れて，一つの code embedding にエンコードする．そして，LSTM decoder は code embedding を元に戻すように学習する．学習の結果として，コードをコードの意味を含めた一つの code embedding にエンコードできる encoder を得られる．評価するとき，その encoder を使って，コードをエンコードする．得られた code embedding のコサイン類似度を計算して，コードクローンかどうかを判断する．レイヤーは

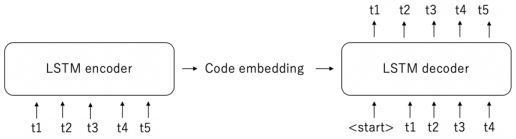
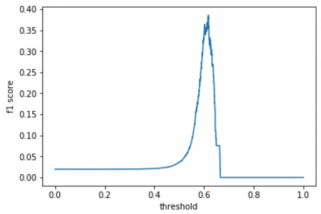


図 3 手法 1

†5 https://leetcode.com



Threshold: 0.6173
Precision: 47.8%
Recall: 32.3%
F1 score: 38.6%

Highest pair's similarity: 0.6653

図 4　手法 1 の結果

一つで，embedding は次元数は 512 である．予想として，二つの言語を一つのモデルで学習することで，仮説 1 で言語の間の共通点をもとに，言語の間の違い部分もベクトル空間上の近い位置に学習できる．

結果は図で示した．F1 は 38.6%である．10609 ペアの中に 103 ペアが正解なので，ランダムで判断する場合は正解率約 1%である．したがって，性能は確かにあった．しかし，高いとは言えない．そして，一番近いペアの類似度は 0.66 なので，Java のコードと Python のコードの距離が離れていることが分かった．二つの言語の間の違いが code embedding に含まれていると考えられる．

4.3　手法 2

図 5 で示した通り，言語ごとに違う decoder を使用することにした．二つの言語のコードは同じ encoder でエンコードするが，元に戻す時にそれぞれの decoder を使う．手法 1 で二つの言語の間の違いが code embedding に含まれていることが分かった．二つの decoder を使うことで，言語の間の違い部分が decoder で吸収され，意味に関わる部分だけ code

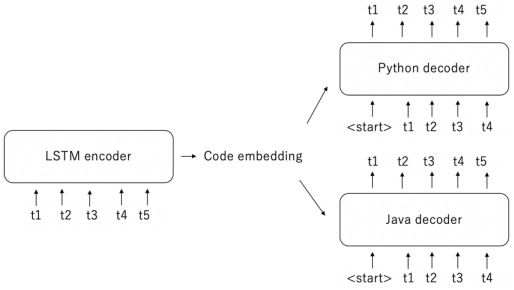
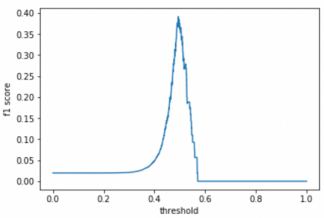


図 5　手法 2



Threshold: 0.4952
Precision: 42.5%
Recall: 36.3%
F1 score: 39.2%

Highest pair's similarity: 0.5844

図 6　手法 2 の結果

embedding に記憶させる．

結果は図 6 で示した．F1 は 39.2%でちょっと上がったが，一番近いペアの類似度は 0.58 で逆に下がりました．二つの decoder を使うことは言語の間の距離を縮めないことが分かった．

4.4　手法 3

図 7 で示した通り，言語ごとに違うスタートトークンを使うことにした．手法 2 で，二つの decoder を使うことは言語の間の距離を縮めないことが分かったので，言語の間の違いを decoder に記憶させるではなく，スタートトークンの embedding に記憶させる．decoder は二つのインプットがある．一つは code embedding で，一つはスタートトークンの word embedding である．スタートトークンの embedding は code embedding と同じ記憶空間がある．言語ごとに違うスタートトークンを使うことで，言語に関する情報はスタートトークンの embedding に記憶させ，意味に関する情報は code embedding に流れさせる．

結果は図 8 で示した．F1 は 51.2%で，大幅に上がりました．そして，一番近いペアの類似度は 0.79 ま

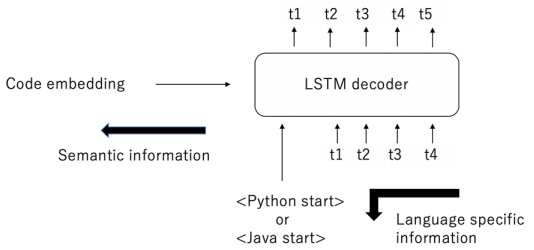


図 7　手法 3

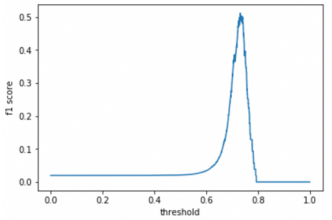


図 8　手法 3 の結果

で上がった．予想通りに，言語ごとに違うスタートトークンを使うことで，言語の間の距離を縮めることができた．

4.5　手法 4

Transformer [20] を使った language model について実験をした．language model とは前の文脈を見て，次の単語を予測し続けるモデルである．モデルは Transformer の decoder だけを使った．レイヤーは 6 で，次元数は 512 である．評価する時に，後ろの padding を取り除いたアウトプットを各トークンの平均を取ってコードの分散表現にする．得られたコードの分散表現のコサイン類似度を計算して，コードクローンかどうかを判断する．予想として，二つの言語を一つのモデルで学習することで，仮説 1 で言語の間の共通点をもとに，言語の間の違い部分もベクトル空間上の近い位置に学習できる．そして，language model を使うことで，仮説 2 で分布仮説が成立するのおかげで，変数名や関数名は隠しラベルの役目を果たす．

結果は図 9 で示した．F1 は 82.4%でかなり高い性能が出た．しかも一番近いペアの類似度は 0.918 である．二つの仮説で予想した通り，違う言語のコードでも距離がかなり近くなった．

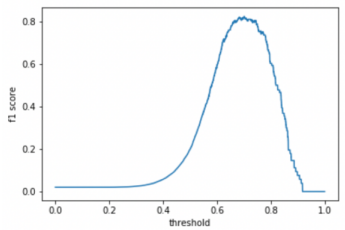


図 9　手法 4 の結果

4.6　手法 5

手法 4 で得られたコードの分散表現の 512 次元の中に言語に関わる特徴と意味に関わる特徴があると予想した．言語に関わる特徴の影響を減らせる実験をした．

コードの分散表現をインプットとして，一つの linear 層の classifier を使って，Python か Java かと分類させる．その linear 層の重みの絶対値が高いところは言語に関係強い特徴であると予想した．学習時に L1 正則化をかけた．L1 正則化をかけることで，予測に影響が弱い特徴の重みは 0 に近く，予測に影響が強い特徴だけ重みが高く学習できる．

結果は精度 99.97%で分類できる．重みは図 10 に示した通り，ほとんどの重みは 0 に近いである．残りの重みの絶対値が高い特徴は言語に関わる特徴だと思われる．

重みの絶対値が高い特徴は以下の式で減らせる．

$$v_i' = (1 - \min(|\, w_i \,| + \alpha, 1)) \cdot v_i \qquad (1)$$

式の中，v_i はコードの分散表現にインデックスが i である要素．v_i' は減らした後の要素である．w_i は linear 層の重みにインデックスが i である要素．α はハイパーパラメーターである．重みの絶対値が $1 - \alpha$ 以上の特徴は取り除かれる．残りの特徴は重みの絶対値が大きければ大きいほど減らされる．結果として，α が 0.95 の時一番高い性能が得られた．

結果は図 11 で示した．F1 は 84.2%で，性能がちょっ

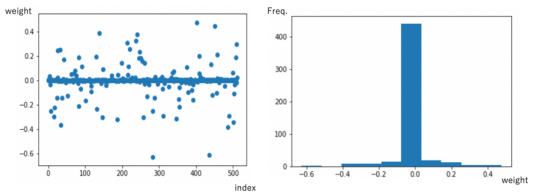
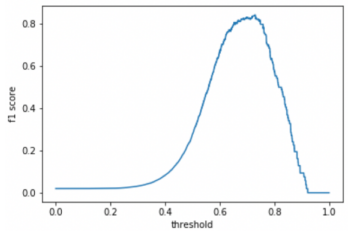


図 10　linear 層の重み



Threshold: 0.7305
Precision: 90.9%
Recall: 78.4%
F1 score: 84.21%

Highest pair's similarity: 0.9222

図 11　手法 5 の結果

と上がった．そして，一番近いペアの類似度は 0.922 で，もっと近くなった．性能はちょっとしか上がらなかったのはデータセットに原因があると思われる．4.7 で議論する．

4.7　議論

評価データでは，10609 ペアの中に 103 ペアが正解である．そのため，ランダムで判断する場合は正解率約 1%である．実験結果では，F1 は 84.2%を得られた．かなり高い精度と思われる．性能の数字だけじゃなく、もっと正確にモデルの性能を評価するため，モデルが正解だと判断したペアの中に，間違ったペアを見てみた．

図 12 から図 15 まではコードペアのサンプルが挙げられた．左は Python のコードで，右は Java のコードである．こちらのペアはモデルが近いと判定されたが、違う問題を解くコードなので、間違いだとカウントされた。でも、これらのペアは違う問題のコードだけど、実はかなり近いであった。

例えば，図 12 サンプル 1 のペアはどっちでも binary search のアルゴリズムを使ったコードである．図 13 サンプル 2 のペアは，問題が違うけど、実は同

```python
class Solution(object):
    def fixedPoint(self, A):
        l, h = 0, len(A) - 1
        while l <= h:
            mid = (l + h) // 2
            if A[mid] < mid:
                l = mid + 1
            elif A[mid] > mid:
                h = mid - 1
            else:
                return mid
        return -1
```

```java
class Solution {
    public int peakIndexInMountainArray(int[] A) {
        int lo = 0, hi = A.length - 1;
        while (lo < hi) {
            int mid = (lo + hi) / 2;
            if (A[mid] < A[mid + 1]) lo = mid + 1;
            else hi = mid;
        }
        return lo;
    }
}
```

図 12　サンプル 1

```python
class Solution(object):
    def topKFrequent(self, words, k):
        count = collections.Counter(words)
        heap = [(-freq, word) for word, freq in count.items()]
        heapq.heapify(heap)
        return [heapq.heappop(heap)[1] for _ in xrange(k)]
```

Python:
Problem 692:
Top K Frequent Words

```java
class Solution {
    public List<Integer> topKFrequent(int[] nums, int k) {
        HashMap<Integer, Integer> count = new HashMap();
        for (int n: nums) {
            count.put(n, count.getOrDefault(n, 0) + 1);
        }
        PriorityQueue<Integer> heap =
new  PriorityQueue<Integer>((n1, n2) -> count.get(n1) - count.get(n2));
        for (int n: count.keySet()) {
            heap.add(n);
            if (heap.size() > k)
                heap.poll();
        }
        List<Integer> top_k = new LinkedList();
        while (!heap.isEmpty())
            top_k.add(heap.poll());
        Collections.reverse(top_k);
        return top_k;
    }
}
```

Java:
Problem 347:
Top K Frequent Elements

図 13　サンプル 2

```python
class Solution(object):
    def findDisappearedNumbers(self, nums):
        res = []
        if nums:
            n = len(nums)
            for i in range(n):
                val = abs(nums[i]) - 1
                if nums[val] > 0:
                    nums[val] = -nums[val]
            for i in range(n):
                if nums[i] > 0:
                    res.append(i + 1)
        return res
```

Python:
Problem 448:
Find All Numbers Disappeared in an Array
Input: [4,3,2,7,8,2,3,1]
Output: [5,6]

```java
class Solution {
    public int missingNumber(int[] nums) {
        int res = nums.length;
        for (int i = 0; i < nums.length; i++) {
            res ^= i;
            res ^= nums[i];
        }
        return res; }
}
```

Java:
Problem 268:
Missing Number
Input: [9,6,4,2,3,5,7,0,1]
Output: 8

図 14　サンプル 3

```python
class Solution(object):
    def removeNthFromEnd(self, head, n):
        if head is None:
            return None
        slow = fast = head
        for i in range(n):
            fast = fast.next
        if fast is None:
            head = head.next
            return head
        while fast.next is not None:
            fast = fast.next
            slow = slow.next
        curr = slow.next
        slow.next = curr.next
        return head
```

```java
class Solution {
    public ListNode middleNode(ListNode head) {
        ListNode fast, slow;
        fast = slow = head;
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }
        return slow;
    }
}
```

図 15　サンプル 4

じようなことをやっている。コードクローンであると考えられる．モデルはうまくこれらのコードをコードクローンだと認識している．図 14 サンプル 3 のペアはちょっと興味深い．完全に違うアルゴリズムを使っている。でも、実は目的が似ていて、一つは 1 から n まで全て無くなって数字を探し出しこと、一つは 1 から n まで一つだけ無くなった数字を探し出すことである。アルゴリズムが違うし、そして、関数名も違う言い方や単語をつかっている。なぜか、モデルはコードクローンだと認識している．図 15 サンプル 4 のペアははどっちでも 2 pointer のテクニックを使っている。一つは LinkedList の真ん中のノードを探し

出すこと、一つは後ろから n 番目のノードを見つけて削除する。アルゴリズム的にはかなり近いである。サンプル以外の他のペアもほとんどは違う問題だけど、アルゴリズムあるいは目的が近いコードである．これらのペアは場合によって，価値のあるコードクローンだと考えられる。

5 関連研究

Cross-language コードクローンの検出は最近研究され始めている．今までの研究について簡単に紹介する．

Kraft *et al.*(2008) [14] は cross-language コードクローンの検出に関する最初の研究を行なった．Microsoft .NET フレームワークの CodeDOM ライブラリーを基づいて，C2D2 というツールを開発した．C#と VB.NET. の共通 CodeDOM グラフを生成できる．また，Al-omari *et al.*.(2012) [1] は共通中間言語（Common Intermediate Language (CIL)）を利用して C#，J#と VB.NET. のコードクローンを検出できる．二つとも.NET 言語についての研究である．Java と Python などの違うプラットフォームの言語には使えない．

X. Cheng *et al.*(2017) [4] は CLCMiner を開発した．多言語プロジェクトの Cross-language コードクローンは修正履歴 (diff) も類似するという仮説を基づいて研究を行なった．でも，修正履歴を持たない一般コードには使えない．

T. Vislavski *et al.*(2018) [21] は Budimac *et al.* の SSQSA アーキテクチャー [3] を利用して，コードを enriched Concrete Syntax Tree (eCST) に変換して行列を生成する．それを用いてコードクローンを検出する．しかし，比較できるコードは制限がある．コードの長さ，ステップとフローが同じでなければならない．実用的に使うには難しい．

Perez *et al.*(2019) [17] は AST を利用してトークンの分散表現を生成する．それから，競技プログラミングのデータを教師データとして，教師あり学習でコードクローンを検出する，K.Nafi *et al.*(2019) [16] はあらかじめ決めた文法特徴を使って，コードを分析する．それから，API ドキュメントの類似度を利用したフィルタを使う．最後に，on-the-fly または競技プログラミングのデータを使った教師あり学習を行う．Perez *et al.* と同じく，競技プログラミングの教師データは良いデータではないため，精度は高くない．

6 まとめ

本研究では二つの仮説の元に，教師なし学習での cross-language コードクローン検出を試みた．実験の結果，高い性能を得られた．結果から見ると，二つの仮説が成立すると考えられる．今後，マスク言語モデルについて実験を行う予定である，また，既存手法と同じデータセットで比較する．

参 考 文 献

[1] Al-omari, F., Keivanloo, I., Roy, C., and Rilling, J.: Detecting Clones Across Microsoft .NET Programming Languages, 10 2012, pp. 405–414.

[2] Baker, B.: A Program for Identifying Duplicated Code, *Proceedings of Computing Science and Statistics: 24th Symposium on the Interface*, Vol. 24(1992).

[3] Budimac, Z., Rakić, G., and Savi'c, M.: SSQSA architecture, 09 2012, pp. 287–290.

[4] CHENG, X., PENG, Z., Jiang, L., Zhong, H., Yu, H., and Zhao, J.: CLCMiner: Detecting Cross-Language Clones without Intermediates, *IEICE Transactions on Information and Systems*, Vol. E100.D(2017), pp. 273–284.

[5] Cordy, J. and Roy, C.: The NiCad clone detector, 06 2011, pp. 219–220.

[6] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K.: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 10 2018.

[7] Firth, J.: A synopsis of linguistic theory 1930-55, *Studies in linguistic analysis. The Philological Society, Oxford*, (1957), pp. 1–32.

[8] Gage, P.: A new algorithm for data compression, *The C Users Journal*, Vol. 12(1994), pp. 23–38.

[9] Harris, Z.: Distributional Structure, *Word*, Vol. 10(1954), pp. 146–162.

[10] Hinton, G. and Salakhutdinov, R.: Reducing the Dimensionality of Data with Neural Networks, *Science (New York, N.Y.)*, Vol. 313(2006), pp. 504–7.

[11] Hochreiter, S. and Schmidhuber, J.: Long Short-term Memory, *Neural computation*, Vol. 9(1997), pp. 1735–80.

[12] Jiang, L., Misherghi, G., Su, Z., and Glondu, S.: DECKARD: scalable and accurate tree-based detection of code clones, 06 2007, pp. 96–105.

[13] Kamiya, T., Kusumoto, S., and Inoue, K.: CCFinder: A multilinguistic token-based code clone detection system for large scale source code, *Software Engineering, IEEE Transactions on*, Vol. 28(2002), pp. 654– 670.

[14] Kraft, N., Bonds, B., and Smith, R.: Cross-language Clone Detection., 01 2008, pp. 54–59.

[15] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J.: Distributed Representations of Words and Phrases and their Compositionality, *Advances in Neural Information Processing Systems*, Vol. 26(2013).

[16] Nafi, K., Kar, T., Roy, B., Roy, C., and Schneider, K.: CLCDSA: Cross Language Code Clone Detection using Syntactical Features and API Documentation, 11 2019, pp. 1026–1037.

[17] Perez, D. and Chiba, S.: Cross-Language Clone Detection by Learning Over Abstract Syntax Trees, 05 2019, pp. 518–528.

[18] Roy, C. and Cordy, J.: A Survey on Software Clone Detection Research, *School of Computing TR 2007-541*, (2007).

[19] Sajnani, H., Saini, V., Svajlenko, J., Roy, C., and Lopes, C.: SourcererCC: scaling code clone detection to big-code, 05 2016, pp. 1157–1168.

[20] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, L., and Polosukhin, I.: Attention Is All You Need, (2017).

[21] Vislavski, T., Rakić, G., Cardozo, N., and Budimac, Z.: LICCA: A tool for cross-language clone detection, 03 2018.

[22] Weaver, W.: Translation, *Machine Translation of Languages. Cambridge, Massachusetts: MIT Press*, (1955), pp. 15–23.