再帰型ニューラルネットワークを用いたコーディングスタイルの自動検査手法の提案

小林 佑樹　西田典紀　千葉滋

機械学習を応用したコーディングスタイルの自動検査手法を提案する．近年，Ruby を代表とする文法的に自由な言語が多く利用されるようになり，コーディング スタイルの組織内での統一がより重要視されるようになった．一方で，従来はコーディングスタイルを ルールとして手動で明文化したり，組織が採用する規約を予め明示的に選択して検査する手法しか存在しなかったため，コーディングスタイルを継続的に検査するための運用コストが課題であった．コーディングスタイルをより低コストで統一できる手法として，本手法では自然言語処理の分野で盛んに利用されている再帰型ニューラルネットワークを応用してソースコード内のコーディングスタイルをコーディング規約ではなく統計を基に自動で検査する．今回の提案では文法の自由度が高い Ruby のコーディングスタイルを自動検査できるシステムとして設計を行った．

We propose a method for evaluating the coding style of source code automatically with machine learning. Recently, programming languages which are acceptable to variant grammar especially Ruby are used, and it is important for teams and organizations to consistently use the same coding style, especially in programming languages that allow various coding styles like Ruby. However, the coding style must be clarified and customized manually. In our research, we represent coding style not by conventions but by likelihood and evaluate the coding style of source code automatically. We derive a likelihood function that represents a coding style from a set of programs which is written in the coding style that is considered to be correct by the team. We calculate the likelihood of token sequences with the likelihood function which we derived and detect tokens which have a low likelihood of being the wrong coding style. We built a system using this approach.

1 はじめに

コーディングスタイルとはプログラムの表現作法を指す．C 言語のようなプログラミング言語が利用されていた時から組織などで定められたコーディングスタイルに則ってコードを書くことは重要とされていた．C 言語や Java のような言語は文法の自由度が低く，一般的に利用されるコーディングスタイルも少なかったため，コーディングスタイルに則ってコードを書くことは難しくなかった．またエディタや IDE の formatting 機能を利用して自動でソースコードのコーディングスタイルを修正することも容易であった．しかし近年は Ruby を代表するような文法的に自由度の高い言語が利用されるようになった．文法的に自由度が高い言語は同じ機能を実装する際にも様々な書き方が可能となるため，コーディングスタイルが多様化してしまう．結果的にコーディングスタイルに則ってコードを書くことや自動でコーディングスタイルを修正することが難しくなった．

本論文ではコーディングスタイルを規則ではなく，トークン列の尤度で表現し，コーディングスタイルの適合評価を自動で行う手法の提案を行う．スタイルに合致するプログラム群から導かれる尤度関数をそのスタイルとみなすことでコーディングスタイルの自動抽出を行う．導いた尤度関数を用いて検査対象のプログラム中で尤度を計算し，尤度の低いトークン列をコーディングスタイルに不適合なものとして検出す

Proposal of automatic inspection method of coding style using recursive neural network

Yuki Kobayashi, Shigeru Chiba, Noriki Nishida, 東京大学情報理工学系研究科創造情報学専攻, Graduate School Information and Science Technology, The University of Tokyo.

る．さらに本手法を使ったシステムを実際に Python を用いて構築し，Ruby の静的コード解析ツールの Rubocop と本手法を使ったシステムでコーディングスタイルの判定精度を比較した．

2　コーディングスタイルの統一

複数の開発者で一つのソフトウェアを開発するような組織内ではコーディングスタイルを統一することで長期間開発されるソフトウェアの保守性が向上するなどのメリットが生まれる．近年は Ruby のような文法の自由度が高い言語が登場し，組織で採用されるコーディングスタイルが多様化したため，より組織内で意識的にコーディングスタイルを統一する必要性が出てきた．

組織内でコーディングスタイルを統一する既存手法としては採用しているコーディングスタイルをルールとして明文化し，ドキュメントとして組織内で共有する手法がある．しかしこれはドキュメントの作成や維持に人手が常にかかってしまう．

また Ruby の Rubocop [1] のような静的解析ツールでコーディングスタイルを自動評価する手法も存在する．例えば Rubocop では YAML 形式の設定ファイルにユーザが設定したいコーディングスタイルに対応するルールをドキュメントから探し出して記述する．Rubocop には Cops というコーディングスタイルに対応したクラスが用意されていて，自分が採用したいコーディングスタイルに対応したクラスを設定ファイルに記述することでソースコード内でそのコーディングスタイルに準拠した書き方になっているかどうかを Rubocop がチェックするようになる．Cops にはパラメータを指定でき，自分たちが採用するコーディングスタイルに対応するパラメータを指定することで自分たちがどのコーディングスタイルを採用しているのかを Rubocop に明示的に指定する．

図 1 に Rubocop の設定ファイルの記述例を示す．図 1 に示すようにユーザーは自分がチェックしたいコーディングスタイルに関するルールを順番に記述する．この例では 3 つのルールが設定されている．1, 2 行目では Linelength という Cops で，一行に許容する文字数を 128 文字に設定している．4, 5 行目では Style というグループの NumericLiterals という Cops をオンにすることで，5 桁を超える数値を表記する際には 1_000_000 のように 3 桁毎にアンダースコアを挿入しているかをチェックするよう設定している．7 から 9 行目では Layout というグループの IndentArray という Cops をオンにして，EnforcedStyle オプションを consistent に設定している．この Cops は Ruby の配列リテラルのインデントのコーディングスタイルをチェックするルールであり，EnforcedStyle オプションに望ましいインデントルールを指定する．これに指定できるオプションは以下の 3 つである．

```
1 LineLength:
2   Max: 128
3
4 Style/NumericLiterals:
5   Enabled: true
6
7 Layout/IndentArray:
8   Enabled: true
9   EnforcedStyle: consistent
```

図 1　Rubocop の設定ファイルの例

(a) special_inside_parentheses
(b) consistent
(c) align_brackets

(a) の special_inside_parentheses を指定すると図 2 に示すように，普通の式中に配列を書く際にはその式の行頭を基準としてインデントを行う一方．メソッド呼び出しの引数として配列を書く場合には開きブラケットの位置を基準にインデントを行わなければならない．(b) の consistent を指定すると，図 3 に示すようにメソッド呼び出しの引数としての配列も行頭を基準にしてインデントを行わなければならない．一方，(c) の align_brackets を指定すると図 4 に示すように普通の式中の配列も，開きブラケットを基準にしてインデントを行う．

しかし Rubocop のようなツールは予めツールにルールを人手で設定しなくてはならない．ルール設定の際には，人手で用意された Rubocop のドキュメントを確認しながら，望む Cops の名前をみつけては設定する．文法の自由度が高い言語ではルールの設定の数が増え，現状のままでは初期コストが増大する．

```
1  # special_inside_parentheses
2  array = [
3    :value
4  ]
5
6  array_in_a_method_call([
7                              :value
8                            ])
```

図 2 special_inside_parentheses **オプション**

```
1  # consistent
2  array = [
3    :value
4  ]
5
6  array_in_a_method_call([
7    :value
8  ])
```

図 3 consistent **オプション**

```
1  # align_brackets
2  array = [
3            :value
4          ]
5
6  array_in_a_method_call([
7                              :value
8                            ])
```

図 4 align_brackets **オプション**

3 機械学習を応用した統計的アプローチ

我々は，コーディングスタイルをプログラム中のトークン列の尤度で表現する方法を提案する．機械学習を利用してこの尤度を計算することで，与えられたプログラムのコーディングスタイルがどれだけ正しそうかを機械的に判定できるようになる．機械学習を用いることでこれまでコーディングスタイルを統一するために組織毎に手動で行われてきた作業が不要になり，より低いコストでコーディングスタイルの統一を行うことが可能となる．

本手法において，正しいコーディングスタイルとは，正しいコーディングスタイルにしたがって書かれたプログラムをコーパスとしたときのトークン列の尤度となる．したがって本方式を利用するには，そのようなコーパスを予め用意できなくてはいけないが，それを用意することは現実的に十分可能である．例えば Git の master ブランチにコミットされたコードレビュー済みのソースコード群や，組織がオープンソース化したプロジェクト内のソースコード群などは，きちんと組織のコーディングスタイルに則って記述されたコードかどうかを組織内で優秀なエンジニアがレビューしていることが多い．そのようなソースコード群は組織が採用するコーディングスタイルに則ったソースコード群であると考えられる．提案する手法では，組織に実在するこのようなソースコードの量から，学習に必要な十分なコーパスを生成できる．

正しいコーディングスタイルで書かれたプログラムから実際に学習に用いるコーパスを生成するために，(i) トークン列化と (ii) トークン列の分割をおこなう．(i) トークン列化では，通常のコンパイラで実施するようなトークン化と異なり，コーディングスタイルの表現に適したトークンにプログラムのテキストを変換する．従来のトークン化と異なり，まず，本来であればトークン化されない空白や改行もトークン化する．これは図 5 のように中括弧の前後にスペースがある場合とない場合の異なる 2 つのコーディングスタイルが存在するなど，コード中のどこで改行を行うかがコーディングスタイルを評価する上で重要な情報になるためである．次にインデントは前行と比較した相対的なインデントの深さの数を保持している．例えば図 6 の中の 2 から 4 行目と 7 から 9 行目では内容は同一の記述だが，2 から 4 行目は関数 `func` 内で定義されているため，7 から 9 行目よりもインデントが 1 つ深くなっている．絶対的なインデントの深さを保持した場合には，この二つのコーディングスタイルは異なるものとして学習されてしまうが，インデントを前行との相対的な情報として保持すれば 2 から 4 行目と 7 から 9 行目のコードは同じコーディングスタイルで書かれたコードとして学習することが可能となる．さらに，クラス名や変数名など各ソースコード固有の情報は，同一のトークンに変換し，抽象化している．これはソースコード固有の情報はコーディングスタイルを学習する上では不要な情報であるためである．プログラムの構成要素とトークンの対応表を 表 1 に示す．

```
1 # Space有
2 10.times { puts 'Hello␣world' }
3
4 # Space無
5 10.times{puts 'Hello␣world'}
```

図 5　スペースの有無の違い

```
1 def func
2   puts [1, 2, 3].map {|i|
3     i * i
4   }
5 end
6
7 puts [1, 2, 3].map {|i|
8   i * i
9 }
```

図 6　インデントのレベルが異なる同一コード

表 1　プログラムの構成要素と対応するトークン

プログラムの構成要素	トークン
空白	<SPACE>
改行	<NEWLINE>
インデント	<INDENTX>
変数	<ID>
クラス	<CLASS>
インスタンス変数	<INSTANCE_VAL>
関数	<FUNCTION>
文字列	<STRING>
数値	<NUMBER>
リテラル	<LITERAL>
オペレーター	<OPERATOR>
クラス変数	<CLASS_VAL>
定数	<CONSTANT_ID>
名前空間	<NAMESPACE>
グローバル変数	<GLOBAL_VAL>
コメント	<COMMENT>
解析に失敗したトークン	<ERROR>

(ii) トークン列の分割には sliding window algorithm [2] を用いる．sliding window algorithm は予めユーザが設定したウィンドウサイズ分の行数を 1 つのウィンドウとして，ウィンドウ内のコードを 1 つの学習データとする．このウィンドウを 1 行ずつずらしていき，1 つのソースコードから複数の学習データ群を生成する．これにより少ない量のプログラムから学習に必要な十分な量のコーパスを生成する．プログラムの 1 行を 1 回の学習に用いる 1 つの学習データとして扱ってしまうと行を跨って表現されるコーディングスタイルを学習することができない．一方で 1 つのソースコード全体をそのまま 1 つの学習データとして扱うと，1 回の学習に用いる学習データがが長くなるため，上手く学習が行われない場合がある．そこで sliding window algorithm を用いて適切なウィンドウサイズを設定することで，複数行に跨ったコーディングスタイルを効率よく学習させる．一方で sliding window algorithm はウィンドウサイズをユーザーが指定するため，分割する行数の選び方が学習結果に影響を与える可能性がある．

学習には Bi-drectional LSTM を用いる．Bi-Directional LSTM は 2 つの LSTM を組み合わせたニューラルネットワークで，入力された学習データのシーケンスを前向きと後ろ向きの双方から学習する．学習に用いる情報量が多いため，N-Gram や単一の LSTM では，尤度を高い精度で計算できない．Bi-Directional LSTM は自然言語処理の分野でも文章中のコンテキストを学習する研究 [3] などで利用されている．

本方式では，この学習結果を用いて，与えられたプログラムが正しいコーディングスタイルに沿っているかを判定する．各トークンの尤度を計算し，尤度が低いと見積もられたトークンを正しいコーディングスタイルに沿わないトークンと判断する．判定結果はヒートマップを用いて可視化する．ソースコードのどの部分が不適切なコーディングスタイルか認識できるように，プログラム全体を表示し，尤度がより低いと判定されたトークンの背景をより濃い赤で表示する．

4　実装と実験

Python を使って本手法を利用したシステムを実装した．Bi-Directional LSTM の実装には Chainer を利用し，その他の処理については独自に実装した．ソースコードをトークン化する前処理では字句解析器を作成する必要があるが，Python の Pygments と

表2　学習に用いたパラメータ

埋め込み層	隠れ層	ウィンドウサイズ	閾値
150	256	15	0.0005

いうモジュールが提供する字句解析器を使って実現した．また本実装は Ruby を対象言語として実装している．

実装したシステムを使ったコーディングスタイルの自動評価の精度を測定するために Rubocop との性能比較実験を行った．実験では Rubocop の判定結果を正解として，提案手法を用いて実装されたシステムがどれだけの精度を出せるのかを precision，recall，F 値の 3 つの値を使って計測した．比較するための実験データはクックパッド株式会社の協力を得て，同社が github 上で公開しているオープンソースプロジェクトを学習データに，社内に保有する内部データを検査対象のデータにした．またニューラルネットワークの次元数，ウィンドウサイズ，システムが誤りと判定する尤度の閾値の 3 つのパラメータを判定精度が上がるように調整した．学習に利用したソースコードはファイル数で 507 ファイル，行数で 21439 行であった．

実験の結果， 表 2 のパラメータのとき，最も精度がよくなり， 表 3 の結果が得られた．実験では，ウィンドウサイズの値が精度に大きく影響を与えていた．図 7 は閾値を 0.005，埋め込み層次元数を 150，隠れ層次元数を 256 に固定し，ウィンドウサイズを変化させた際の指標値の変化を表す．図 7 が示すようにウィンドウサイズを大きくすると指標値が改善され，ウィンドウサイズが 15 の時に F 値が最も大きな値をとる．また全実験を通して precision と recall の相関は正の相関関係が観察された．この結果から我々が実装したシステムにはコーディングスタイルの検査を得意とするソースコードとそうでないソースコードがあると考えられる．

5　まとめ

ニューラルネットワークを用いた尤度計算によって組織のコーディングスタイルに則ったコードかどうかを自動で判定する手法を提案した．Python で本提案

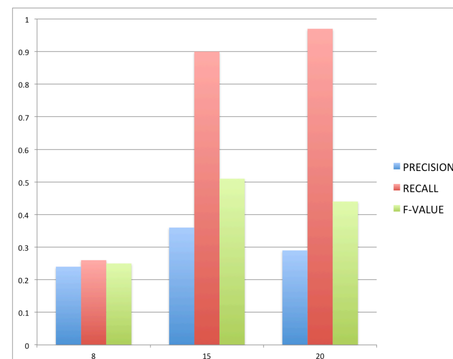

図7　ウィンドウサイズを変化させた際の指標値の変化

表3　提案手法の判定精度

precision	recall	F 値
0.40	0.72	0.51

手法を基にしたシステムを構築し，システムの実用性を検証するためにクックパッド株式会社から提供された実際のソースコードで実験を行い，精度を測定した．ウィンドウサイズによって学習の精度に大きな差がでてしまうため，プログラムの構造などを解析して自動的に適切なウィンドウサイズを設定すること等が今後の課題である．

謝辞

実験に協力いただいたクックパッド株式会社に感謝いたします．

参考文献

[1] `https://github.com/bbatsov/rubocop`.
[2] Koç, C. K.: Analysis of sliding window techniques for exponentiation, *Computers & Mathematics with Applications*, Vol. 30, No. 10(1995), pp. 17–24.
[3] Melamud, O., Goldberger, J., and Dagan, I.: context2vec: Learning Generic Context Embedding with Bidirectional LSTM., *CoNLL*, 2016, pp. 51–61.