Java システムにおけるパッケージ誤りのニューラルネットワークを用いた検出手法

依田 和樹　中丸 智貴　穐山 空道　山崎 徹郎　千葉 滋

本論文では，Java システムにおいて本来あるべきと異なるパッケージに配置されてしまったメソッドをニューラルネットワークを用いて検出する手法を提案する．パッケージの誤りは開発現場においてしばしば見られ，開発者の生産性に悪影響を与えることが知られている．しかし，それらを一般に検出する手法は未だ存在していない．我々は開発プロジェクト内のメソッドをその所属するパッケージに分類するという学習を通して，アーキテクチャデザインの観点からあるべきパッケージを予測するモデルを得た．この際，検出対象のプロジェクトのみでは訓練データが足りないといった問題を解決するため，Few-shot 学習の手法を用いた．実際にモデルを実在のプロジェクトに適用し，誤りの実例を検出した．また，Few-shot 学習を活用し，パッケージの配置を考える上で重要なソースコードの特徴はプロジェクトを超えて共通であることを示した．なお，本研究における評価実験は予備的なものであり，本論文の内容は途中経過の報告であることを述べておく．

1　はじめに

ソフトウェアアーキテクチャの重要性は多くの研究で言及され，開発現場においても広く認識されている [22]．Java プロジェクトの開発においては，アーキテクチャはパッケージ間の関係として表現される．つまり，各パッケージは他のパッケージに向けてそれぞれのサービスを公開するインターフェースを持ち，互いに疎結合なパッケージが情報のやり取りをすることでアーキテクチャを表現する．このように，パッケージの境界が明確である状態を保ちつつ開発を進めることが生産性を高めることに寄与するとされている [1].

本研究では誤ったパッケージに配置されたメソッドを一般に検出するツールを提案する．例えば，Model-View-Controller (MVC) アーキテクチャを採用する Web システムにおいてデータベースへのアクセスは Model から行われるべきであるが，仮に Controller 内からデータベースを操作するようなメソッドが存在すれば，それはパッケージを誤っていると言える．このような実装はパッケージの境界を曖昧にし，開発者の生産性を下げることにつながる点で問題である [10] [1]．対象を MVC アーキテクチャに限定してこのような違反を検出する研究は存在するが [11]，MVC 以外にも一般にパッケージの誤り検出を行うタスクは本研究で初めて取り組むものである．

提案手法ではニューラルネットワークを用い，対象プロジェクトに最適化するように訓練した検出器を用いて検出を行う．これは，プロジェクトごとのデザインの考え方の違いを検出に反映するためである．しかし，検出対象のプロジェクトに含まれるデータの量は通常そこまで多くなく，ニューラルネットワークの学習を行うには十分でない．そこで，本研究では Few-shot 学習の手法を応用することで，単一プロジェクトのみのデータでも十分に学習ができるようにモデルを構築した．Few-shot 学習の手法をソースコードを処理するタスクに応用するのも本研究が初めての試みである．

Detecting Misplaced Methods in Wrong Java Packages Using Neural Models.
This is an unrefereed paper. Copyrights belong to the Authors.

Kazuki Yoda, Tomoki Nakamaru, Soramichi Akiyama, Tetsuro Yamazaki, Shigeru Chiba, 東京大学, The University of Tokyo.

```java
// in the controller package
@GetMapping("/{userId}")
public User getUserById(
  @PathVariable("userId")
  final Long userId) {
  return userRepository.findById(userId)
    .orElseThrow(() ->
      new IllegalArgumentException(
        "The requested userId ["
        + userId + "]does not exist."));
}
```

リスト 1　controller から service を経由せずに直接 repository にアクセスしている

本研究の評価実験は依然予備段階ではあるものの，以下の 2 点を貢献として考える．

1. 誤ったパッケージに配置されたメソッドを検出するツールを開発した
2. Few-shot 学習を用い，パッケージの配置を考える上で重要なソースコードの特徴はプロジェクトを超えて共通であることを示した

2　研究の動機

ソフトウェア開発において，コードを分割して整理された状態を保つことの重要性は多くの研究で指摘され，一般の開発者にも浸透している．Java で記述されたソフトウェアにおいては，パッケージがモジュールの役割を果たし，それぞれのパッケージが特定の責務や関心を実装するクラス群を内包する [1] [7]．パッケージの境界は情報の隠蔽 [18] や高凝集・低結合 [21] などの様々な原則に従って規定され，このことが開発者の生産性に寄与するとされている．

反対に，パッケージの設計原則に反した実装は開発者の生産性を低下させる原因となる．本研究では誤ったパッケージに配置されたメソッドに着目する．リスト 1 はその一例であり，GitHub 上にソースコードが公開された実在のプロジェクトの実装である．†1 このプロジェクトは Spring Framework†2 を用いた MVC アーキテクチャに基づいて実装されており，このメソッドは controller パッケージの UserController クラスに実装されている．実装としては問題なく動作するものの，この実装は controller パッケージ内から repository パッケージのオブジェクトである userRepository にアクセスしている点で問題である．このプロジェクトの他の実装を参照する限り controller パッケージからは Model を実装する service パッケージのオブジェクトへのアクセスしか許されておらず，service の内部構造である repository への直接的なアクセスはデザインルールに違反している．従って，このメソッドは service の実装となるように移動されるべきであり，controller は単純に service に処理を委譲して結果を返却するのみとされるべきである．この例は Brain Controller と呼ばれるコードスメル の一種としても認識されており [5]，コードの理解容易性，テスト容易性，拡張性，再利用性を下げる原因とされている [10]．

```java
// in the service package
public Author findAuthor(Long id) {
  return this.authorMapperMybatis
             .findAuthor(id);
}
```

リスト 2　本来 service.impl にあるべき実装クラスが service に存在している

パッケージを誤ったメソッドをさらに一例挙げる (リスト 2)．†3 これはリスト 1 とは別のプロジェクトであるが，同様に Spring Framework を用いた MVC アーキテクチャに基づくものである．このメソッドは引数の id を用いてデータベースを検索するものであり，コード中の MyBatis†4 という単語はデータベースアクセスを行うライブラリを指す．このプロジェクトの構造を概観すると，Model の実装はインターフェースとその実装でパッケージが分けられており，インターフェースは service パッケージに，実装クラスは service.impl パッケージに配置されていることがわかる．この点において，リスト 2 の実装は Model の実装クラスであるにも関わらず service

†1　https://github.com/Apress/learn-microservices-w-spring-boot
†2　https://spring.io
†3　https://github.com/zhuzhengping911/MySpringBoot
†4　https://mybatis.org

パッケージに配置されていることが問題であると言える．このような実装もリスト 1 と同様に，プロジェクトの構造理解を妨げ，リファクタリングや自動テストの対象から見落とされてバグの原因となる可能性をはらんでいる．

配置を誤ったメソッドの検出に類する代表的な研究として Feature Envy の検出 [23][8][16] がある．Feature Envy なメソッドとは，現在配置されているクラス内でのメソッド・フィールド呼び出しよりも他のクラスのメソッド・フィールドへの呼び出しが多く，他のクラス側により強い関心を持っているメソッドである [9]．このようなメソッドはより強い関心を持つクラスの下に移動されるべきであると考えられる．つまり Feature Envy の検出は，クラスを誤って配置されたメソッドを検出・移動するタスクであり，本研究で扱うタスクと関連する研究である．

しかし，リスト 1, 2 の例は Feature Envy には該当せず，従来手法を用いて検出することはできない．まず，リスト 2 の例は明らかにメソッド呼び出しに関わる誤りではない．リスト 1 の例についても，メソッド呼び出しの関係からわかることは `controller` から `repository` への呼び出しがあるというだけであり，ここから `service` に移動させるべきという結論は得られない．このように，本研究が対象とする問題は Feature Envy とは異なるものである．

これらの例を誤りとして検出し，正しいパッケージを提示するために考慮すべきことは 2 つあると考える．それは，1) プロジェクトごとのデザインの違いを考慮すること，2) メソッド本文のテキスト情報を用いることである．ここで，プロジェクトのデザインとは，どのパッケージにどのような実装が内包されているかという全体的な傾向を指す．メソッド本文のテキスト情報と併せてリスト 1 の例を考えると，`controller` パッケージのメソッドには `service` という単語が多く出現するが `repository` という単語は出現しないといった傾向がわかる．また，`service` パッケージには他のどのパッケージよりも多く `repository` という単語が出現することから，リスト 1 のメソッドは `controller` から `service` に移動されるべきであるという結論が導ける．同様に，リスト 2 の例も，`service` にはインターフェース宣言ばかりだが，`service.impl` には `Mybatis` という単語を用いる実装が多いため `service.impl` に移動させるべきという結論を導ける．

以上の観察を基に，本研究ではメソッド本文の関連度による比較を行い，プロジェクトのデザインルールに照らして他のパッケージに移動されるべきメソッドを検出する手法を提案する．

3 提案手法

本研究で扱うタスクは，誤ったパッケージに配置されたメソッドを検出し，本来あるべきパッケージを指摘するものである．入力情報として与えられるのはプロジェクト全体のソースコードのみであり，検出結果としてパッケージを誤ったメソッドとその本来あるべきパッケージを挙げる．例えば，プロジェクト全体からリスト 1 のような例を挙げ，`controller` から `service` に移動させるべきと指摘するタスクとなる．

検出モデルの入力にはメソッドのテキスト情報を与え，ニューラルネットワークを用いてベクトルへの変換を行う．セクション 2 では，このタスクにおいてメソッド本文の情報を活用する必要性を述べた．従来の研究におけるテキスト情報の利用は識別子のラベルのみを一定のルールに基づき数値変換するに留まっていたが [14]，近年のニューラルネットワークを用いた研究では AST (抽象構文木) 上の文構造なども併せてエンコードする手法が頻繁に用いられ，様々なタスクで従来手法を上回る成果を挙げている [3][4][2]．本研究においてもメソッドのエンコードに同様の手法を用いることで性能の向上が期待できると考えられる．

プロジェクトごとに異なるデザインの違いを検出に反映するため，モデル全体の構成には prototypical networks [20] (以下，PN) を用いる．PN はそれぞれのプロジェクトごとに最適化した検出モデルを行うことのできるモデルである．これを用いて Few-shot 学習を行うことで，単一プロジェクトの少ないデータからでも十分な学習が可能となる．セクション 2 で指摘した通り，リスト 1, 2 のような例を誤りと指摘するためには，プロジェクトごとのデザイン指針，つまり，各パッケージにどのようなメソッドが集まって

いるかという情報を抽出する必要がある．PN を用いることでこれらの情報を抽出し，対象プロジェクトに最適化した検出結果が期待できる．

3.1 システムの全体像

パッケージを誤ったメソッドの検出は，入力したメソッドをいずれかのパッケージに分類するような分類器を訓練することによって行う．検出の全体像を図 1 に示す．分類器の出力はプロジェクト内のパッケージごとに所属確率を推論したものであり，最も高い確率が出力されたパッケージが本来あるべきパッケージであると結論する．その後，推論されたパッケージと入力したメソッドの現在存在しているパッケージが異なり，かつ確率が一定の閾値を超えている場合にそのメソッドの配置が誤りであるとして検出する．

検出モデルの学習は二段階あり，1) 対象プロジェクトに適用する前に大量の別プロジェクトによって行う事前学習と，2) 対象プロジェクトのデータを用いてそのプロジェクトでの分類に最適化した分類器を得るための学習に分かれている．具体的な手法はセクション 3.2 で詳述する．

本システムを用いて実際に検出を行う際は，対象プロジェクト内のすべてのメソッドを訓練データとテストデータに分割する必要がある．例えば，プロジェクト全体のメソッドのうち 8 割を訓練データ，2 割をテストデータとなるように分割する．事前学習済みのモデルを用いて訓練データで分類器を訓練することにより，対象とするプロジェクトに最適化した分類が行えるようになる．この方法でテストデータを対象とした検出が行えるようになるものの，これではプロジェクト全体の 2 割のデータしか対象にすることができていない．そのため，実際のユースケースでは訓練データとテストデータを入れ替えながら，すべてのデータが一回ずつテストデータとなるようにして検出を行う．また，開発中のプロジェクトにおいて新規のコード差分に対してのみ検出を行うことも可能である．この場合は，既に存在するコードベースをすべて訓練データに用い，コード差分に含まれるメソッドのみを対象として検出を行う．

3.2 モデルの構成と訓練方法

検出に用いるモデルは大きく 1) メソッドのテキスト情報をベクトルに変換するエンコーダーと 2) エンコーダーの出力結果をいずれかのパッケージに分類する分類器から構成される (図 1)．以下ではそれぞれの詳細な構成と訓練方法について述べる．

エンコーダー エンコーダーはメソッドのテキスト情報をベクトルに変換するニューラルネットワークである．本研究では Code2vec [4] のエンコーダーを改変し，型の情報を活用できるようにしたモデルを用いる．

Code2vec はメソッドから得られる AST をベクトルに埋め込むモデルである．ただし，メソッド名の情報は AST から除かれている．入力形式は木構造そのものではなく，2 つの葉ノードとそれらを結ぶパス (AST context) を構文木上からランダムに n 個サンプルしたものである．個別の AST context は $(w_s, w_t, r_1, \ldots, r_l)$ というタプルとして表現するものとする．w_s, w_t は始点と終点の葉ノードであり，メソッド中のいずれかのトークンに相当する．$r_1, \ldots, r_l$ は 2 つの葉を結ぶ l 個のノードであり，式や文などの AST の構成要素に相当する．エンコーダーの役割はこれら n 個の AST context からそれぞれのベクトル表現 $\mathbf{z}_1, \ldots, \mathbf{z}_n$ を求めて，それらを用いて最終的なメソッドの分散表現 $\mathbf{v} \in \mathbb{R}^d$ を得ることである．

パスのエンコードには Code2seq [3] で採用された Bi-directional LSTM を用いた構成を採用する．これは Code2vec の構成への改善として Code2seq で導入されたものと同様である．

$$h_1, \ldots, h_l = LSTM(E^{nodes}_{r_1}, \ldots, E^{nodes}_{r_l})$$
$$encode\_path(r_1, \ldots, r_l) = [\overrightarrow{h_l}; \overleftarrow{h_1}]$$

ここで，E^{nodes} は AST ノード中に現れる単語に対する埋め込み行列である．

葉ノードのエンコードも概ね Code2seq の構成に倣うが，型情報を入力に含めるように拡張を行った．まず，入力値の w は $split(\cdot)$ 関数を用いてキャメルケースの大文字部分を基準としたサブトークンに分割する．例えば `getCurrentTime` というメソッド名は `get`，`current`，`time` の 3 つに分割される．次に

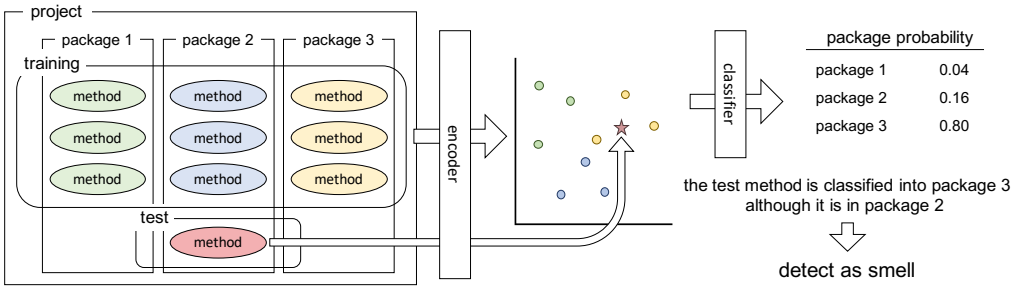

図 1　エンコーダーと分類器を用いた検出の全体像

$types(\cdot)$ 関数を用いて w の型を表す集合を得る．つまり，w がクラス名・インターフェース名または変数名・引数名を表すトークンである場合，$types(\cdot)$ はその型の名前及び直接的に継承するスーパークラスと実装するインターフェースの名前からなる集合を返す．該当しない場合は $types(w) = \emptyset$ である．また，この時すべての型名は上記と同様なサブトークン化が行われる．例えば，w が `ConcreteModel` 型のインスタンスで， `ConcreteModel` が `AbstractModel` を継承し `Serializable` を実装する場合には $types(w) = \{concrete, abstract, model, serializable\}$ となる．これとサブトークンに現れる単語の埋め込み行列である $E^{subtokens}$ を用いて葉ノードのエンコードを以下のように定義する．

$$S = split(w) \cup types(w)$$
$$encode\_token(w) = \sum_{s \in S} E_s^{subtokens}$$

単一の AST context の分散表現 $\mathbf{z}$ を求めるための手順は Code2seq と同様である．

$$\mathbf{z} = tanh(W_{in}[encode\_token(w_s); \\ encode\_path(r_1, \ldots, r_l); \\ encode\_token(w_t)])$$

こうして得られた n 個の分散表現 $\mathbf{z}_1, \ldots, \mathbf{z}_n$ からメソッド全体の分散表現 $\mathbf{v}$ を求める方法はオリジナルの Code2vec と同様である．つまり，soft attention を用いて計算される $\mathbf{z}_1, \ldots, \mathbf{z}_n$ の加重和が $\mathbf{v}$ である．

分類器 分類器は各パッケージの代表点を内部に保持し，入力値がどのパッケージに属するかの判断を行う．これは PN で通常用いられる分類器と同様である．また，分類器はニューラルネットワークではないため事前学習において何らかのパラメータを学習することはない．

プロジェクト P における分類器は以下のように作る．分類器の訓練データは $D = \{(x_1, y_1), ..., (x_N, y_N)\}$ として与えられる．$i = 1, 2, \ldots, N$ として，x_i はメソッド m_i のソースコードであり，$y_i \in \{1, \ldots, K\}$ は m_i を内包するパッケージを表すラベルである．パッケージ k に属するすべてのサンプルを D_k として，各パッケージの代表点 $\mathbf{c}_k$ を以下のように求める．

$$\mathbf{c}_k = \frac{1}{|D_k|} \sum_{(x_j, y_j) \in D_k} f_\phi(x_j) \tag{1}$$

ここで $f_\phi(\cdot)$ は先に述べたエンコーダーである．つまり，$\mathbf{v}_i = f_\phi(x_i)$ である．また，ϕ はニューラルネットワークの学習可能パラメーターである．

推論時の出力は K 個のパッケージそれぞれに対する所属確率である．推論対象のメソッドのソースコードを q として，パッケージ $k \in K$ への所属確率は以下の通り求める．

$$p_\phi(y = k \mid q) = \frac{\exp\left(-dist(f_\phi(q), \mathbf{c}_k)\right)}{\sum_{k' \in K} \exp\left(-dist(f_\phi(q), \mathbf{c}_{k'})\right)} \tag{2}$$

ここで，$dist(\cdot)$ は 2 点間のユークリッド距離を求める関数である．

meta-learning meta-learning は単一プロジェクトの少ないデータからでも性能のよい分類器を作るための事前学習の方法である．これは PN の一般的な事前学習手法である．meta-learning においては，大量に用意した事前学習用のプロジェクトからプロジェクトの小さなサブセットを作成し，その少ないデータから分類器を作って分類するという訓練を 1 タスクとして繰り返す．つまり，meta-learning とは，少数のデータから性能のよい分類器をつくるためのパラ

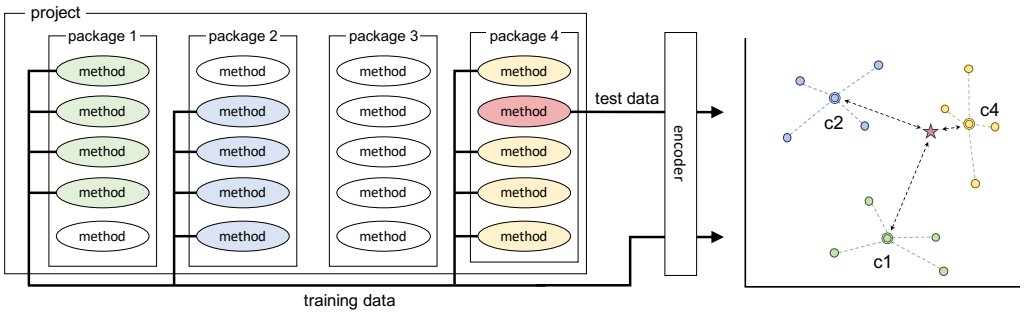

図 2　4-shot 3-way の設定による meta-learning

メーター ϕ をエンコーダーに学習させるためのものである．

meta-learning における 1 回のミニバッチ学習の様子を図 2 に示した．ミニバッチごとに 1 つのプロジェクトを訓練データの中から選ぶ．この際，プロジェクトはそのパッケージ数に応じて確率的に選択される．そして，対象のプロジェクトから，内包するメソッド数に応じて確率的に K パッケージを選び，K パッケージのそれぞれから無作為に N 個ずつのメソッドを選ぶ．こうして得られた NK 個のメソッドは分類器の訓練データとして用いられ，式 (1) によって代表点の計算が行われる．次に，K 個のパッケージから先の訓練データと重複しないようにメソッド q を選択する．式 (2) による推論結果について，$J(\phi) = -\log p_\phi(y = k \mid q)$ を最小化するように勾配降下が行われる．このようなミニバッチ学習を繰り返すことで目的とするエンコーダーが得られる．以上の学習設定を N-shot K-way の meta-learning と呼ぶ．

訓練データに関する議論 事前学習に用いるプロジェクトに含まれるメソッドとパッケージの対応はすべて正しいものであることが期待されるが，実際には少数の誤りが含まれてしまっている．このことは検出結果に悪影響を与えることが予想される．しかし，セクション 4 で述べる通り，事前学習に用いたデータは GitHub 上でスターの多いプロジェクトであるため，これらはよく整理されて比較的誤りの少ない状態に保たれていることが期待できる．また，ニューラルネットワークは過学習を起こさない限り，こうした少数の誤りに過敏に反応して性能を落とすことはない．関連研究にも同様の問題が見られるが，実際に学習データ上の誤りの影響は存在しないまたは軽微であり，十分な性能を挙げている [16] [4].

4　評価

提案手法の評価にあたっては以下の二点を明らかにするために二種類の実験を行った．

- **RQ1.** 実際にパッケージを誤ったメソッドを検出できるか
- **RQ2.** 機械学習モデルの設計は適切であったか

一つは実際の OSS (オープンソースソフトウェア) を用いてパッケージを誤ったメソッドの事例を検出するものであり，もう一つは機械学習モデルの一部を改変した結果と比較することで提案手法の設計について議論するものである．なお，この実験結果は予備段階の結果であり，実験の内容・考察ともに今後より精緻化していく必要がある点についてここで述べておく．

データセット データセットには Code2vec [4] の評価に用いられたものと同じデータセット (java-large) を用いる．これは GitHub でスターの多い Java プロジェクトを 9000 以上集めたものである．

検証にあたって，使用するデータセットには適切な前処理を行った．まず，今回の検出に不要と思われる自動テストの実装をすべてデータセットから除いた．同じく，Java の実装に多く見られるボイラープレートについても除いた．ここで，ボイラープレートとは，getter，setter，`Object#equals`，`Object#hashcode`，`Object#toString` などの実装コードを指す．また，ソースコード中のコメントもエンコードには用いな

いため前処理の中で除外した．

モデルの訓練 前処理の段階でモデルの訓練に用いる単語の辞書を作成した．単語の辞書にはメソッドのテキスト情報の単語を集めたものと AST 上のノード名を集めたものの 2 つあり，それぞれセクション 3.2 で導入した $E^{subtokens}$，E^{nodes} に対応する．メソッドのテキスト情報の単語は 1) メソッドに含まれるトークンのテキスト情報をサブワードに分割したもの 2) メソッドに含まれる識別子やクラス名の型情報をサブワードに分割したものから構成される．これらの単語の出現頻度を数え，頻度の低いものは `<UNK>` という特殊な単語に置き換えた．これは自然言語処理のタスクで用いられる手法で，モデルの過学習を防ぐ効果がある．また，モデルの学習率は 0.001 に設定し，PN としての学習設定は 10-shot 20-way とした．

4.1 ケーススタディによる評価

RQ1 に対する回答として，実際の OSS プロジェクトにモデルを適用し，パッケージを誤ったメソッドの実例を検出できるかをケーススタディによって確かめる．実験用のプロジェクトとして，1) GitHub 上でスターが多く，2) パッケージ数 15 以上 50 未満であり，3) java-large に含まれない 1000 個のプロジェクトを収集した．実験には java-large の training データセットで訓練したモデルを用いた．誤りとして検出する条件は，推論されたパッケージと現在のパッケージが異なり，かつ推論先の確率が 0.9 を超える場合となるように設定した．

リスト 1, 2 の事例はこのケーススタディによって検出したものである．いずれもセクション 2 で述べた通り，パッケージを誤っており開発者の生産性に影響を与えかねないものである．これ以外にも，code clone や scattered functionality [10] と思われるいくつかの事例を検出した．このことは，メソッドの本文中にはパッケージの配置を考える上で重要な特徴が含まれており，Few-shot 学習によってそれらの知識をプロジェクトを超えて転移できたことを示している．

現状の結果は，検出事例から true positive と思われる事例を挙げたのみであり，一定量のサンプルをすべて人の目で判断して precision を計測するような作業はできていない．実際の検出事例には false positive と見られるものも多くあり，今後これらに対する考察・改善が必要である．また，検出に用いた閾値も暫定的であるため，プロジェクトごとにどのような閾値を設定する必要があるのかは今後の研究で明らかにする必要がある．

4.2 モデル設計の評価

RQ2 に対する回答として，モデル設計の是非を評価する．セクション 3.2 では単語の分割や型情報を活用する機構を通常の Code2vec モデルに追加したが，これが性能にどのような結果を与えるのかを考察する．また，メソッドをエンコードする手法として AST ベースの Code2vec を用いたが，その他の手法，つまりグラフベースの GGNN [2] およびトークンベースの Bi-directional LSTM + attention [12] との比較を行う．なお，Allamanis らの手法 [2] で用いられた GGNN モデルではグラフノードの分散表現を得るまでしかできないため，ここでは読み出し機構 [15] を追加してグラフ全体，つまりメソッド全体の分散表現を得られるようにしている．

評価手法 評価の方法として，複数のモデルに対して 5-fold の交差検証を行い，その性能を比較する．まず，test データセット内の各プロジェクトのデータをパッケージごとに 8 割の訓練データと 2 割のテストデータに分割する．この訓練データとテストデータを入れ替えながら 5 回の性能測定を行い，その平均を検証対象のモデルのスコアとする．

性能のメトリクスには top-1 accuracy と set-base の hierarchical F1 [13] を用いる．評価はテストデータのメソッドをモデルに入力し，あるべきパッケージを推論させることで行う．この時，出力結果の確率が最大であるパッケージが元のパッケージと完全一致していた割合が top-1 accuracy である．仮に結果が誤っていた場合でも，hierarchical F1 を用いることで，パッケージの木構造の上で遠い誤りと近い誤りを区別して評価することができる．

表 1　異なるエンコーダーによる分類の性能比較

Model	top-1 accuracy	F1-score
Code2vec	0.577	0.762
GGNN	0.558	0.749
Bi-LSTM	0.580	0.744
Ours	**0.645**	**0.775**

評価結果 評価実験を行った結果を表 1 に示す．セクション 3.2 で導入した我々のモデルは Code2vec をベースに改善を加えたものであったが，この実験の結果において我々のモデルは Code2vec の結果を accuracy と F1 のいずれにおいても上回った．つまり，サブワード化や型情報の活用がこのタスクにおいては性能に寄与することがわかった．また，AST ベース以外のエンコーダーとして GGNN と Bi-LSTM のモデルとの比較を行ったが，結果として我々の選択したモデルがこのタスクにおいては最も高い性能を得られることがわかった．

5　関連研究

コードスメルの検出やリファクタリングの研究には本研究と関連する内容が多い．しかし，パッケージを誤ったメソッドの検出というタスクに取り組んだ研究はなく，用いている手法も本研究とは大きく異なる．セクション 2 で述べたように，Feature Envy の検出 [23][8][16] は本研究に関連したタスクであるが，メソッド呼び出しやメソッド名を特徴量として検出を行っており，本研究のアプローチとは全く異なる．Hayashi らの研究 [11] では，MVC アーキテクチャを採用するプロジェクトを対象にモジュールを誤って配置された実装を検出するタスクを扱っていたが，その検出手法は MVC アーキテクチャのみに閉じており，一般的なパッケージの誤りに拡張することは難しいものであった．また，パッケージをリファクタリングするツールとして ARIES [17] が挙げられるが，これは凝集性の低いパッケージを解体して他のパッケージに整理するものであり，誤りを一般に検出する本研究の動機とは異なるものである．

また，近年ではニューラルネットワークのソフトウェア工学への応用は盛んであり，コードスメルの検出に用いられた事例も存在する [16][19]．しかし，本研究のように検出アルゴリズムを特定のプロジェクトに特化するように訓練したものはない．本研究で用いた Few-shot 学習の手法は，最初は画像処理に始まり，近年では自然言語処理での応用も多い [6][24] が，ソースコードを処理するタスクに用いた事例は本研究が初めての試みである．

6　まとめと今後の課題

本研究ではパッケージを誤ったメソッドを一般に検出する手法を提案し，手法の有用性を示すいくつかの事例を挙げた．しかし，評価の方法・結果は依然予備的なものであり，今後改善すべき点も多い．まず，ケーススタディとして，機械学習モデルの検出結果を人の目で再判断し，定量的に precision を計測する必要がある．また，モデル選択の評価についても現状では適切なベースラインが設定できていないため，これを与える必要がある．

参 考 文 献

[1] Abdeen, H., Ducasse, S., and Sahraoui, H.: Modularization Metrics: Assessing Package Organization in Legacy Large Object-Oriented Software, *2011 18th Working Conference on Reverse Engineering*, 2011, pp. 394–398.

[2] Allamanis, M., Brockschmidt, M., and Khademi, M.: Learning to Represent Programs with Graphs, *International Conference on Learning Representations*, 2018.

[3] Alon, U., Brody, S., Levy, O., and Yahav, E.: code2seq: Generating Sequences from Structured Representations of Code, *International Conference on Learning Representations*, 2019.

[4] Alon, U., Zilberstein, M., Levy, O., and Yahav, E.: Code2vec: Learning Distributed Representations of Code, *Proc. ACM Program. Lang.*, Vol. 3, No. POPL(2019).

[5] Aniche, M., Bavota, G., Treude, C., Gerosa, M. A., and Deursen, A.: Code smells for Model-View-Controller architectures, *Empirical Software Engineering*, Vol. 23(2018).

[6] Bao, Y., Wu, M., Chang, S., and Barzilay, R.: Few-shot Text Classification with Distributional Signatures, *International Conference on Learning Representations*, 2020.

[7] Beck, F. and Diehl, S.: On the Congruence of Modularity and Code Coupling, *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Soft-*

ware Engineering, ESEC/FSE '11, New York, NY, USA, Association for Computing Machinery, 2011, pp. 354–364.

[8] Fokaefs, M., Tsantalis, N., Stroulia, E., and Chatzigeorgiou, A.: JDeodorant: identification and application of extract class refactorings, *2011 33rd International Conference on Software Engineering (ICSE)*, 2011, pp. 1037–1039.

[9] Fowler, M.: *Refactoring: Improving the Design of Existing Code*, Addison-Wesley Longman Publishing Co., Inc., USA, 1999.

[10] Garcia, J., Popescu, D., Edwards, G., and Medvidovic, N.: Identifying Architectural Bad Smells, *2009 13th European Conference on Software Maintenance and Reengineering*, 2009, pp. 255–258.

[11] Hayashi., S., Minami., F., and Saeki., M.: Inference-based Detection of Architectural Violations in MVC2, *Proceedings of the 12th International Conference on Software Technologies - Volume 1: ICSOFT,*, INSTICC, SciTePress, 2017, pp. 394–401.

[12] Iyer, S., Konstas, I., Cheung, A., and Zettlemoyer, L.: Summarizing Source Code using a Neural Attention Model, *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Berlin, Germany, Association for Computational Linguistics, August 2016, pp. 2073–2083.

[13] Kosmopoulos, A., Partalas, I., Gaussier, E., Paliouras, G., and Androutsopoulos, I.: Evaluation Measures for Hierarchical Classification: A Unified View and Novel Approaches, *Data Min. Knowl. Discov.*, Vol. 29, No. 3(2015), pp. 820–865.

[14] Kuhn, A., Ducasse, S., and GÃ®rba, T.: Semantic clustering: Identifying topics in source code, *Information and Software Technology*, Vol. 49, No. 3(2007), pp. 230–243. 12th Working Conference on Reverse Engineering.

[15] Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. S.: Gated Graph Sequence Neural Networks, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Bengio, Y. and LeCun, Y.(eds.), 2016.

[16] Liu, H., Xu, Z., and Zou, Y.: Deep Learning Based Feature Envy Detection, *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2018, pp. 385–396.

[17] Palomba, F., Tufano, M., Bavota, G., Oliveto, R., Marcus, A., Poshyvanyk, D., and De Lucia, A.: Extract Package Refactoring in ARIES, *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 2, 2015, pp. 669–672.

[18] Parnas, D. L.: On the Criteria to Be Used in Decomposing Systems into Modules, *Commun. ACM*, Vol. 15, No. 12(1972), pp. 1053–1058.

[19] Pigazzini, I.: Automatic Detection of Architectural Bad Smells through Semantic Representation of Code, *Proceedings of the 13th European Conference on Software Architecture - Volume 2*, ECSA '19, New York, NY, USA, Association for Computing Machinery, 2019, pp. 59–62.

[20] Snell, J., Swersky, K., and Zemel, R.: Prototypical Networks for Few-shot Learning, *Advances in Neural Information Processing Systems*, Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R.(eds.), Vol. 30, Curran Associates, Inc., 2017, pp. 4077–4087.

[21] Stevens, W. P., Myers, G. J., and Constantine, L. L.: Structured Design, *IBM Syst. J.*, Vol. 13, No. 2(1974), pp. 115–139.

[22] Taylor, R. N., Medvidovic, N., and Dashofy, E. M.: *Software Architecture: Foundations, Theory, and Practice*, Wiley Publishing, 2009.

[23] Terra, R., Valente, M. T., Miranda, S., and Sales, V.: JMove: A novel heuristic and tool to detect move method refactoring opportunities, *Journal of Systems and Software*, Vol. 138(2018), pp. 19–36.

[24] Yu, M., Guo, X., Yi, J., Chang, S., Potdar, S., Cheng, Y., Tesauro, G., Wang, H., and Zhou, B.: Diverse Few-Shot Text Classification with Multiple Metrics, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, New Orleans, Louisiana, Association for Computational Linguistics, June 2018, pp. 1206–1215.