Vol.2010-SE-167 No.8
2010/3/18

コーディングスタイルの特徴量とソースコード盗用との関係の分析

武田隆之[†1] 牛窓朋義[†1] 山内寛己[†1]
門田暁人[†1] 松本健一[†1]

本稿では, 学生の演習課題のような小規模なソースコードを対象とした盗用の検出を目的とする. インデント, 演算子などのコーディングスタイルに着目し, 59 項目の特徴量として抽出し, 盗用の発見に用いる. 盗用関係にあるソースコード間において 59 項目の特徴量の差分を測定し, 盗用関係にないソースコード間における特徴量の差分と比較したところ, 28 項目の特徴量が盗用検出に有効であること, 28 項目のうち 8 項目の特徴量はプログラムの内容によらず盗用検出に有効であること, 8 項目のうち 3 項目の特徴量はソースコード整形ツールによるインデント整形に対して耐性を持つことが分かった.

An Analysis of Relationship between Coding Style Metrics and Source Code Plagiarism

Takayuki Takeda ,[†1] Tomoyoshi Ushimado ,[†1]
Hiroki Yamauchi ,[†1] Akito Monden [†1]
and Ken-ichi Matsumoto[†1]

The goal of this paper is to detect software plagiarism in small-size source code like exercise assignments at school. This paper focused on coding style elements, such as indents and operators, and computed 59 quantitative measures from these elements. To evalute the usefulness of measures for plagiarism detection, we compared measures of suspected pairs (of plagiarism) and non-suspected pairs. As a result, we found that 28 measures were effective to detect plagiarisms. Especially, 8 of 28 measures were effective for different program specifications, and 3 out of 8 measures were effective even after source code indentation tools were applied.

1. はじめに

大学などの教育機関でのプログラミング技術を習得する科目において, 受講している学生が演習課題に対しコピーアンドペーストなどを用いたソースコード盗用 (以下, 盗用) を行い, 安易に単位を習得しようとする問題がある.

このような盗用により, 成績評価が正しくできないだけでなく, 受講者の習熟度を教員が正しく把握できないといった問題が発生する. したがって, ソースコード間での盗用検出を行う必要がある.

本稿では, 学生のプログラミング演習課題における数百行以下の小規模なソースコード間でのコピーアンドペーストを主とした盗用を検出することを目的とし, 個人のソースコードの書き方による表層的な特徴 (コーディングスタイル) から抽出される特徴量に関して仮説の検証および分析を行う. 具体的な仮説としては「仮説 1:コーディングスタイルから抽出される特定の特徴量は, 盗用関係にあるソースコード間と盗用関係にないソースコード間で差がある.」「仮説 2:プログラム内容が異なっても, 特定の特徴量は盗用関係にあるソースコード間と盗用関係にないソースコード間で差がある.」「仮説 3:ソースコード整形ツールで改ざんしても, 特定の特徴量は盗用関係にあるソースコード間と盗用関係にないソースコード間で差がある.」の 3 つである. これらの仮説を検証し, コーディングスタイルを用いた盗用検出の有用性を確かめるために, 実際にプログラミング演習課題において学生から提出されたソースコードから, 盗用関係にあると思われる組み合わせと盗用関係にないと思われる組み合わせを用意し, それらのコーディングスタイルの特徴量の差分を比較する実験を行った. その結果, 盗用関係にあるソースコード間と盗用関係にないソースコード間に比べて複数のコーディングスタイルの特徴量の差分に差があること, プログラムの内容が変わっても一部の特徴量には同様の差があること, ツールを用いた改ざんがなされても一部の特徴量には同様の差があることがわかった. また, コーディングスタイルの特徴量の差分を用いた判別分析を行った結果, コーディングスタイルの特徴量の差分が盗用検出にある程度有効であることがわかった.

以降, 2 章では従来研究について述べ, 3 章ではコーディングスタイルとソースコード盗用に関する仮説を述べ, 4 章では今回抽出する特徴量について述べる. 5 章では行った実験と

†1 奈良先端科学技術大学院大学
Nara Institute of Science and Technology

その結果について述べ, 6 章では得られた結果について考察を述べ, 7 章ではまとめと今後の課題について述べる.

2. 従来研究

2.1 盗用検出に関する研究

これまでにいくつかの盗用検出手法が提案されている.

2.1.1 ソフトウェアバースマークを用いた手法

ソフトウェアバースマークとは, プログラムを抽象化した特徴量であり, この特徴量を比較することで盗用の有無を判断する. Tamada らは変数の初期値や継承関係などを静的バースマークとし, 類似度の算出方法を提案している[1). 岡本らは Windows の API 呼び出しの順序や頻度を動的バースマークとし, 実行系列をもとにした類似度と実行頻度をもとにした類似度の算出方法を提案している[2) .

しかし, プログラムの規模が非常に小さく, 同一の要求仕様のもとに作られた状況においては, 得られる情報にほとんど差がないため, 盗用と非盗用の判別は困難な場合がある.

2.1.2 コードクローンを用いた手法

コードクローンとは, ソースコード中の類似または一致するコード片をいい, コピーアンドペーストや連続した定型処理, もしくは偶然によって発生する[3). コピーアンドペーストのコード片の検出に注目し, 盗用検出を行ったのが JPlag[4) である. JPlag はソースコード間のコードクローンの含有率を用いた類似度の算出を行う. 岡原らは, オープンソースソフトウェアを対象とした実験から, ソフトウェア間で検出される最大コードクローン長と最大コードクローン長から算出される部分類似度がソースコード流用もしくは盗用の検出に有効としている[5).

しかし, 行単位や文単位で順序を入れ換えることでコードクローン長が短くなることもあり, ソースコードが比較的小規模である場合, 盗用と判断できるコードクローン長をもつコードクローンが存在しない, または存在し得ないことが多く, さらに同一の要求仕様のもとに作られた状況においては, 偶然によるコードクローンが多数発生し, 多数のコードクローンの中に盗用により発生したコードクローンが紛れるため, 盗用と非盗用の判別が困難な場合がある.

2.1.3 local alignment を用いた手法

Ji らは, 2 つの DNA の配列から類似部分を発見するための伝統的な類似度算出手法である local alignment を応用し, 他のソースコードに対する関連数と影響度を基にした距離尺度を定義し, 距離尺度からソースコードの開発過程を示す系統樹を作成するアルゴリズムを提案しており, 同一の問題に対する複数回の解答系列を比較する実験から, 平均数十行のソースコードに対しても盗用の検出が可能であることを示している[6).

しかし, プログラミング演習課題において, 多くの場合, 1 つの課題に対してソースコードは 1 回しか提出されないため, 演習課題に対してどの程度有効であるかは明らかでない.

2.2 コーディングスタイルと著者の関係に関する研究

コーディングスタイルとソースコードの著者の関係に関しては研究が行われており, 一定以上の精度で著者の推定も可能としている.

永井らは, 機械学習を用いることで著者やプログラムの用途の推定が高い精度で行えるとしている[7). 岩間らは, オープンソースプロジェクトを対象にした実験において, コーディングスタイルの違いが検出可能であり, 各プロジェクトで最も多くの部分を書いた開発者以外によって書かれた部分をある程度検出可能だとしている[8).

そこで, 本稿では永井らによって提案されたコーディングスタイルの特徴量に注目し, 比較的小規模なソースコードを対象とした盗用の検出を行う. 以降, その方法について述べる.

3. コーディングスタイルとソースコード盗用の関係

本章では, 想定する盗用者およびソースコード盗用の定義を述べたのち, 盗用関係にあるソースコード間におけるコーディングスタイルの共通点の有無を明らかにするための仮説を立てる. そして, プログラムの内容の違いやツールを用いた改ざんによる影響の有無を明らかにするための仮説を立てる.

3.1 盗用者およびソースコード盗用の定義

本稿で想定するソースコード盗用者および盗用手法は以下のとおりである.

- 盗用者は, 実習科目を受講する学生であり, 同じ科目の受講者からソースコードを盗用する.
- 盗用者は, ソースコードをコピーアンドペーストで盗用する.
- 盗用者は, 盗用を発見されないことを目的としたソースコード編集を行う.

以上を想定した上で, 次の 3 つの仮説を立てる.

3.2 盗用関係における共通性に関する仮説

仮説 1:コーディングスタイルから抽出される特定の特徴量は, 盗用関係にあるソースコード間と盗用関係にないソースコード間で差がある.

盗用関係にあるソースコードに見られる共通点を明らかにする. 本稿では, コピーアンド

ペーストを用いたソースコード盗用を想定している．したがって, 盗用によって作成されたソースコードのコーディングスタイルは, 盗用元のソースコードのコーディングスタイルを踏襲する傾向があると考えられる．そのため, 盗用関係にあるソースコード間で抽出された特徴量の差は, 盗用関係にないソースコード間で抽出された特徴量の差に比べて小さくなると考えられる．

仮説 2:プログラム内容が異なっても, 特定の特徴量は盗用関係にあるソースコード間と盗用関係にないソースコード間で差がある．

盗用関係にあるソースコードに見られる共通点へのプログラムの内容による影響を明らかにする．盗用を行う上で, 細工を施しやすいと盗用者が考える部分はプログラムの内容によって変化する．たとえば, ソースコード中に少し複雑な部分があれば, その部分に改変が加わると課せられた要求仕様どおりにプログラムが作動しない可能性があるため, その部分に盗用者は細工を施さないと考えられる．そのため, 盗用者が編集しにくいと考える部分に現れやすい特徴があれば, 特徴量として抽出し比較することで, 盗用関係にあるソースコード間では盗用関係にないソースコード間と比べて一部の特徴量の差が小さくなると考えられる．

3.3 改ざんに対する耐性に関する仮説

仮説 3:ソースコード整形ツールで改ざんしても, 特定の特徴量は盗用関係にあるソースコード間と盗用関係にないソースコード間で差がある．

盗用関係にあるソースコードに見られる共通点への改ざんによる影響を明らかにする．本稿では, コピーアンドペーストを用いて盗用が行われると想定しているが, ソースコードの自動整形ツールを用いることで, 盗用元のコーディングスタイルの特徴を残さずにソースコードをコピーアンドペーストできる可能性がある．しかし, そのような改ざんがなされていても, ソースコード整形の影響が小さい, もしくは影響を受けないために, 盗用関係にあるソースコード間では盗用関係にないソースコード間と比べて一部の特徴量の差が小さくなると考えられる．

4. コーディングスタイルの特徴量

本章では, 抽出するコーディングスタイルの特徴量について述べる．これらの特徴量は永井らによって提案されており, 詳細は永井らの論文を参照されたい[7]．以降, 簡潔に説明する．抽出する特徴量は合計で 59 項目あり, それぞれ「長さとインデント」「括弧前後の書き方」「コメント」「変数や関数の名前」「演算子」の 5 つに分類される．抽出する特徴量の一覧を表 1 - 5 に示す．

4.1 長さ, インデントの特徴量

ファイル全体の見た目を表す特徴量として, 表 1 のような特徴量を抽出する．スコープレベルが増えるごとにインデントの数を増やす表記法は一般的であることから, スコープレベルとインデントの数との比の平均を取ることにより, プログラムを書いた人の特徴を抽出する．また, スコープレベルごとに, スコープレベルとインデントの数との比の平均に対し, インデントの数の差をとり, その差の平均と分散を調べることによって, 一貫したスコープレベルとインデントの数の関係があるのかを特徴量として抽出する．さらに, 盗用者はコメントや空行の追加や削除によって盗用の発見を免れようとする可能性があることから, コメントを削除し連続した改行を全て 1 度の改行に変換した上での特徴量もそれぞれ抽出する．なお, インデントについては空白を使ったインデントとタブを使ったインデントの 2 種類が見られ, タブを使ったインデントはエディタによって表示が異なるが, 今回は全て 8 つの空白に変換してからそれぞれの特徴量を抽出する．

4.2 括弧前後の書き方の特徴量

括弧は, 式やブロックを区別する要素となる．クラスや関数の開始や終了, 条件などを表すのに用いられる．したがって, 前後の書き方に特徴が現れることが多い．また, コーディング規約がある場合, それを順守しているかの目安になる．したがって, 括弧の前後に改行がどのくらいの割合であるのか, 括弧の前後に空白がどれだけ挿入されるのかを特徴量として抽出する．抽出する特徴量を表 2 に示す．

4.3 コメントの特徴量

4.1 節で触れたようにコメントは追加や削除により改ざんされる可能性があるが, 頻度や 1 箇所あたりのコメントの量は個人差が現れやすい要素であるため, 特徴量として抽出する．抽出する特徴量を表 3 に示す．

4.4 変数やクラスの名前の特徴量

ソースコードを作成する者によって, 変数やクラスにその機能を特徴づける名前を使うことがある．そのため, 変数やクラスの名前の長さは作成者の特徴を表す要素となり得る．また, スコープレベルが大きくなるにつれ, 名前の付け方が簡略化されることが考えられる．したがって, 変数やクラスの名前の長さの平均と分散, 大文字, 小文字, 数字, アンダースコアの使用頻度を特徴量として抽出する．抽出する特徴量を表 4 に示す．

4.5 演算子の特徴量

プログラムが対象とする演算や手続きについて, それを実現する方法が複数ある場合, 特定の演算子を多用するなど, 演算子の用法には, アルゴリズムに関する特徴が強く現れると

考えられる．そこで, 行あたりの演算子の出現回数, 演算子の種類ごとに演算子の全て演算子に対する利用の割合と, 演算子の前後の空白の平均を種類ごとに特徴量として抽出する．抽出する特徴量を表 5 に示す．

表 1 長さ・インデントに関する特徴量

長さ・インデントに関する特徴量 (14 項目)
プログラムの行の長さの平均
プログラムの行の長さの分散
インデントの平均
インデントの分散
インデントのスコープレベルとの比の平均
スコープレベルごとの平均とインデントの差の平均
スコープレベルごとの平均とインデントの差の分散
コメント削除&空行マージ済み プログラムの行の長さの平均
コメント削除&空行マージ済み プログラムの行の長さの分散
コメント削除&空行マージ済み インデントの平均
コメント削除&空行マージ済み インデントの分散
コメント削除&空行マージ済み インデントとスコープレベルの比の平均
コメント削除&空行マージ済み スコープレベルごとの平均とインデントの差の平均
コメント削除&空行マージ済み スコープレベルごとの平均とインデントの差の分散

表 2 括弧の前後の書き方に関する特徴量

括弧の前後の書き方に関する特徴量 (11 項目)
{ の前に改行の来る割合, 後に改行の来る割合
} の前に改行の来る割合, 後に改行の来る割合
(の前の空白の平均, 後の空白の平均
) の前の空白の平均, 後の空白の平均
予約語に続く (の前の空白の平均
演算子に続く (の前の空白の平均
その他に続く (の前の空白の平均

表 3 コメントに関する特徴量

コメントに関する特徴量 (4 項目)
コメントのバイト数の平均
コメントの行数の平均
バイト数についてのファイルに対する割合
行数についてのファイルに対する割合

表 4 変数やクラスの名前に関する特徴量

変数やクラスの名前に関する特徴量 (8 項目)
名前の長さの平均
名前の長さの分散
行あたりの名前の種類数
名前が大文字のみである割合
名前が小文字のみである割合
名前に大文字と小文字が混在する割合
名前に数字が含まれる割合
名前にアンダースコアが含まれる割合

表 5 演算子に関する特徴量

演算子に関する特徴量 (22 項目)
全演算子に対する単項算術演算子の出現割合
全演算子に対する単項論理演算子の出現割合
全演算子に対する二項算術演算子の出現割合
全演算子に対する二項論理演算子の出現割合
全演算子に対する二項比較演算子の出現割合
全演算子に対する二項代入演算子の出現割合
全演算子に対する三項条件演算子の出現割合
単項算術演算子の前の空白平均, 後の空白平均
単項論理演算子の前の空白平均, 後の空白平均
二項算術演算子の前の空白平均, 後の空白平均
二項論理演算子の前の空白平均, 後の空白平均
二項比較演算子の前の空白平均, 後の空白平均
二項代入演算子の前の空白平均, 後の空白平均
三項条件演算子の前の空白平均, 後の空白平均
演算子の行あたりの出現回数

5. 実　　験

本章では, 3 章で挙げた仮説を検証するための実験手法とその結果について述べる．仮説 1 および 2 を 5.1 節の手法で検証し, 仮説 3 を 5.2 節の手法で検証する．なお, 実験対象には神戸大学で連続して課された Java プログラミング演習課題（それぞれ課題 1, 2 とする）に対し, 学生から提出されたソースコードを用いた．実験対象の課題の提出数, 平均行数および盗用と思われる組み合わせ数を表 6 に示す．

5.1 盗用関係における共通性に関する仮説の検証実験

仮説 1, 2 を検証するために以下の手順で実験を行う．

まず, 提出されたソースコードに対し, 明らかに盗用関係があると思われるソースコードの組み合わせを目視で確認し, 盗用関係の集合を作る．さらに, 他のどのソースコードとも

表 6 実験対象となる課題

	提出数	平均行数	盗用と思われる組数
課題 1	73	45.5	9
課題 2	89	68.3	12

盗用関係にないと思われるソースコードを 10 件確認し, 盗用関係のない集合を作る.

次に, 4 章で挙げた特徴量を抽出し, 盗用関係の集合においてはそれぞれ盗用関係間の特徴量の差分をとり, 盗用関係のない集合においては総当たりで各ソースコード間で特徴量の差分をとる.

次に, それぞれの特徴量に対し, 盗用関係の集合の差分と盗用関係のない集合の差分の 2 群を対象とし, 以下の仮説で F 検定を有意水準 0.05 で行う.

- 帰無仮説 H_0 :盗用関係の集合の差分の群と盗用関係のない集合の差分の群には分散に差がない.
- 対立仮説 H_1 :盗用関係の集合の差分の群と盗用関係のない集合の差分の群には分散に差がある.

この仮説検定において, 特徴量として抽出できないなどの理由で F 検定が不可能な特徴量は除外し, F 検定で有意差があったもの, つまり H_1 を採択した特徴量には不等分散仮定で, F 検定で有意差が無かったもの, つまり H_0 が棄却されない特徴量には等分散仮定で, 次の仮説で t 検定を有意水準 0.05 で行う.

- 帰無仮説 H_0 :盗用関係の集合の差分の群と盗用関係のない集合の差分の群には平均値に差がない.
- 対立仮説 H_1 :盗用関係の集合の差分の群と盗用関係のない集合の差分の群には平均値に差がある.

この試行を課題 1, 2 に対してともに行い, 2 つのうちどちらかにおける t 検定に有意差がある, つまり課題 1, 2 のどちらかの平均値に関する仮説検定で H_1 を採択する特徴量があれば, 仮説 1 を支持する特徴量とする. また, 課題 1, 2 双方において平均値に関する仮説検定で H_1 を採択する特徴量があれば, 仮説 2 を支持する特徴量とする.

最後に, 仮説 2 を支持する特徴量の差分を用いた線形判別分析を課題 1, 2 に行う. 課題 1 から得られた判別関数を用いて課題 2 に対しての線形判別分析を行い, 課題 2 から得られた判別関数を用いて課題 1 に対しての線形判別分析を行い, それぞれの正判別率を算出し, 仮説 2 を支持する特徴量の差分がどの程度盗用検出に有効かを調査する.

5.2 改ざんに対する耐性に関する仮説の検証実験

整形ツールによる改ざんに対する耐性を確認するべく, 盗用関係の集合と盗用関係のない集合に含まれるソースコードに対し, ソースコードの整形を行った上での特徴量の比較を行う. ソースコードの整形には Meadow3.0[9] を用い, 全てのソースコードに対し indent-region 機能を用いたインデント整形を行う. 整形を行ったのち, 5.1 節と同様の手順で特徴量の差分の比較を行う. どちらの整形においても 5.1 節の実験と共通して, 平均値に関する仮説検定で t 検定に有意差があり, H_1 を採択する特徴量があれば仮説 3 を支持する特徴量とする.

さらに, 仮説 3 を支持する特徴量の差分を用いて 5.1 節と同様の手順で判別分析を行い, 仮説 3 を支持する特徴量の差分がどの程度盗用検出に有効かを調査する.

5.3 盗用関係における共通性に関する仮説の検証実験の結果

特徴量の差分の仮説検定を行ったところ, 課題 1 において 18 項目, 課題 2 において 18 項目の特徴量で差分の平均値に有意差があった. また, 8 項目の特徴量において, 2 つの課題に共通して差分の平均値に有意差があった. 結果を表 7 にまとめる.

この結果を受け, 課題 1, 課題 2 に共通して差分の平均値に有意差があった特徴量を用いて判別分析を行った. 課題 1, 課題 2 の判別分析結果の詳細を表 8 および表 9 に示す. 課題 1 から得られた判別関数を用いた課題 2 の判別結果の詳細を表 10, 課題 2 から得られた判別関数を用いた課題 1 の判別結果の詳細を表 11 に示す.

5.4 改ざんに対する耐性に関する仮説の検証実験の結果

課題 1 においては 12 項目の, 課題 2 においては 5 項目の特徴量で差分の平均値に有意差があり, また, 課題 2 において差分の平均値に有意差があった特徴量 5 項目全てが課題 1 と共通して有意差があった項目となった. 差分の平均値に有意差があった項目の一覧を表 12 に示す.

この結果を受け, 課題 1, 課題 2 に共通して差分の平均値に有意差があった特徴量を用いて判別分析を行った. 課題 1, 課題 2 の判別分析結果の詳細を表 13 および表 14 に示す. 課題 1 から得られた判別関数を用いた課題 2 の判別結果の詳細を表 15, 課題 2 から得られた判別関数を用いた課題 1 の判別結果の詳細を表 16 に示す.

表 7 盗用関係における共通性に関する仮説の検証実験において差分の平均値に有意差があった特徴量

特徴量	課題 1 で有意差	課題 2 で有意差
プログラムの行の長さの平均		あり
インデントの平均	あり	あり
インデントのスコープレベルとの比	あり	
スコープレベルごとの平均とインデントの差の平均	あり	あり
スコープレベルごとの平均とインデントの差の分散	あり	あり
コメント削除&空行マージ済み プログラムの行の長さの平均	あり	
コメント削除&空行マージ済み プログラムの行の長さの分散	あり	あり
コメント削除&空行マージ済み インデントの平均	あり	あり
コメント削除&空行マージ済み インデントの分散		あり
コメント削除&空行マージ済み インデントとスコープレベルとの比		あり
コメント削除&空行マージ済み スコープレベルごとの平均とインデントの差の平均		あり
コメント削除&空行マージ済み スコープレベルごとの平均とインデントの差の分散		あり
{ の前に改行の来る割合	あり	
{ の後に改行の来る割合	あり	
} の前に改行の来る割合	あり	
) の後の空白の平均	あり	あり
コメントの行数の平均	あり	
バイト数についてのファイルに対する割合	あり	
全演算子に対する単項算術演算子の出現割合		あり
全演算子に対する二項代入演算子の出現割合	あり	あり
二項代入演算子の前の空白平均	あり	
二項代入演算子の後の空白平均		あり
演算子の行あたりの出現回数	あり	あり
変数の行あたりの名前の種類数	あり	
変数が大文字のみである割合	あり	
名前の長さの平均		あり
名前の長さの分散		あり
名前に大文字と小文字が混在する割合		あり

表 8 盗用関係における共通性に関する仮説の検証実験における課題 1 の判別分析結果

	非盗用と判別	盗用と判別	合計
実際は非盗用	38(84.4%)	7(15.6%)	45
実際は盗用	1(11.1%)	8(88.9%)	9

正判別率 85.2% , 再現率 88.9% , 適合率 53.3%

表 9 盗用関係における共通性に関する仮説の検証実験における課題 2 の判別分析結果

	非盗用と判別	盗用と判別	合計
実際は非盗用	37(82.2%)	8(17.8%)	45
実際は盗用	1(8.3%)	11(91.7%)	12

正判別率 84.2% , 再現率 91.7% , 適合率 57.9%

表 10 盗用関係における共通性に関する仮説の検証実験における課題 1 の判別関数を用いた課題 2 の判別結果

	非盗用と判別	盗用と判別	合計
実際は非盗用	31(68.9%)	14(31.1%)	45
実際は盗用	5(41.7%)	7(58.3%)	12

正判別率 66.7% , 再現率 58.3% , 適合率 33.3%

表 11 盗用関係における共通性に関する仮説の検証実験における課題 2 の判別関数を用いた課題 1 の判別結果

	非盗用と判別	盗用と判別	合計
実際は非盗用	40(88.9%)	5(11.1%)	45
実際は盗用	1(11.1%)	8(88.9%)	9

正判別率 84.2% , 再現率 88.9% , 適合率 61.5%

表 12 全てのソースコードに整形を行った場合において差分の平均値に有意差があった特徴量

特徴量	課題 1 で有意差	課題 2 で有意差
インデントの平均	あり	あり
スコープレベルごとの平均とインデントの差の平均	あり	あり
コメント削除&空行マージ済み プログラムの行の長さの平均	あり	あり
コメント削除&空行マージ済み プログラムの行の長さの分散	あり	
{ の前に改行がある割合	あり	
) の後の空白の平均	あり	
コメントのバイト数の平均	あり	
コメント行数の平均	あり	
二項代入演算子の出現割合	あり	あり
二項代入演算子の前の空白の平均	あり	
演算子の行あたりの出現回数	あり	あり
変数の行あたりの名前の数	あり	

表 13 全てのソースコードに整形を行った場合における課題 1 の判別分析結果

	非盗用と判別	盗用と判別	合計
実際は非盗用	38(84.4%)	7(15.6%)	45
実際は盗用	0(0.0%)	9(100.0%)	9

正判別率 87.0% , 再現率 100% , 適合率 56.3%

表 14 全てのソースコードに整形を行った場合における課題 2 の判別分析結果

	非盗用と判別	盗用と判別	合計
実際は非盗用	35(84.4%)	10(15.6%)	45
実際は盗用	0(0.0%)	12(100.0%)	12

正判別率 82.5% , 再現率 100% , 適合率 54.5%

表 15 全てのソースコードに整形を行った場合における課題 1 の判別関数を用いた課題 2 の判別分析結果

	非盗用と判別	盗用と判別	合計
実際は非盗用	34(75.6%)	11(24.4%)	45
実際は盗用	2(16.7%)	10(83.3%)	12

正判別率 77.2% , 再現率 83.3% , 適合率 47.6%

表 16 全てのソースコードに整形を行った場合における課題 2 の判別関数を用いた課題 1 の判別分析結果

	非盗用と判別	盗用と判別	合計
実際は非盗用	39(86.7%)	6(13.3%)	45
実際は盗用	1(11.1%)	8(88.9%)	9

正判別率 87.0% , 再現率 88.9% , 適合率 57.1%

6. 考　　察

本章では, 5 章の結果についての考察を述べる. 特徴量の差分の平均値の検定についての考察をそれぞれ述べたのち, 双方で行った判別分析の結果についての考察を述べる.

6.1 特徴量の差分の平均値の検定についての考察

6.1.1 盗用関係における共通性に関する仮説の検証実験について

盗用関係があるソースコード間と盗用関係がないソースコード間では複数の特徴量で差分の平均値に有意差があった. これにより仮説 1 は支持されたといえる. また, 2 つの課題に共通して差分の平均値に有意差があった項目があったことから, 仮説 2 は支持されたといえる. 以降, 2 つの課題の両方において有意差があった特徴量についてそれぞれ考察を述べる.

- インデントの平均
- コメント削除＆空行マージ済みインデントの平均

4 章で述べたように, インデントの変遷はプログラムの変遷をよく表すため, 盗用関係にあるソースコード間のインデントは類似しやすい. そのためインデントの平均はどのようなプログラムにおいても盗用検出の手がかりになると考えられる.

- コメント削除＆空行マージ済みプログラムの行の長さの分散

コメントおよび空行マージを行うことで, ソースコード中のプログラム動作に関係する部分のみが残り, 内容の差に特徴量の差が対応するようになったと考えられ, 盗用検出の手がかりになると考えられる.

- スコープレベルごとの平均とインデントの差の平均
- スコープレベルごとの平均とインデントの差の分散

スコープレベルとインデントの関係については, スコープレベルの変化に応じてインデント数を変化させるコーディングスタイルが一般的に浸透しているため, インデントの平均と同様の理由で差分が小さくなりやすく, 盗用検出の手がかりになると考えられる.

-) の後の空白の平均

差分の平均値に有意差があった理由として, 編集の煩雑さが考えられる.) がソースコード中に出現する状況を考えると, 変数の型の変換や条件分岐, 演算の途中などプログラムの中で比較的複雑な部分であり, 盗用者にとって細工しにくい部分であるといえる. そのため, 盗用検出の手がかりになると考えられる.

- 全演算子に対する二項代入演算子の出現割合
- 演算子の行あたりの出現回数

二項代入演算子の出現割合に有意差があった理由として代入演算の数が変化させづらいという点があげられる. 差分を大きくする方法としてはダミー演算の挿入が挙げられるが, 演習課題という対象の性質上, 盗用者にとっては, 教員などによる評価が下がる要因となりうるだけでなく, 盗用を発見される手がかりにもなりうるため, ダミー演算を挿入することは難しいと考えられる. 同様に, 盗用関係にあるソースコードは演算数が限られている上に行数も近くなりやすいため演算子の行あたりの出現回数の差分が小さくなったと考えられる. 以上から, 盗用検出の手がかりになると考えられる.

6.1.2 改ざんに対する耐性に関する仮説の検証実験について

ソースコードの整形を行っても, 課題 1 と課題 2 に共通して, 差分の平均値に有意差がある特徴量があったことから, 仮説 3 は支持されたといえる. 以降, 仮説 1, 2 および 3 を全て支持する項目について考察を述べる.

- スコープレベルごとの平均とインデントの差の平均

有意差が現れる主な理由としては盗用関係における共通性に関する仮説の検証実験と同様であるが, インデントの整形による影響があるにもかかわらず, 有意差が現れた理由としては Meadow の indent-region の機能が空行におけるインデント数に影響を及ぼさないことが考えられる.

- 全演算子に対する二項代入演算子の出現割合
- 演算子の行あたりの出現回数

盗用関係における共通性に関する仮説の検証実験と同様の理由であり, また, インデントの整形による影響が全くないため, 同様に有意差があったと考えられる.

6.2 判別分析についての考察

判別分析の結果は軒並み正判別率が 80 % から 87 % の間で大きな変化はなかった. したがって, コーディングスタイルから抽出される特徴量の差分を用いた線形判別関数による判別分析にはある程度の判別精度が期待できると考えられ, 年度や学期などの異なる期間で同じ演習課題を行う場合において, 盗用検出に有効であると考えられる.

一方で, 他の課題での判別関数を用いた判別分析の結果は, もとの判別関数を用いたものより正判別率が上昇するものと下降するものがそれぞれあり, 正判別にややばらつきのある結果となった. 理由としては, 他と比べて低い正判別率が結果として出た盗用関係における共通性に関する仮説の検証実験では「インデントの平均」と「コメント削除&空行マージ済みインデントの平均」という非常に近い意味をもつ特徴量の差分が判別関数の変数としてあり, 変数選択が適切でない可能性があることが考えられる.

また, 全ての判別分析結果の適合率と再現率を比較すると, 再現率が適合率を大きく上回る結果となった. このことから, コーディングスタイルの特徴量を用いた盗用の判別は盗用関係の検出よりも盗用関係を含むと考えられる集合の絞り込みに適していると考えられる.

7. おわりに

本稿では, 学生のプログラミング演習課題における数百行以下の小規模なソースコード間でのコピーアンドペーストを主とした盗用を検出することを目的として, コーディングスタイルから抽出される特徴量に関する仮説について検証および分析を行った. 実際のプログラミング演習課題を対象とした実験の結果から得た知見は以下の通りである.

- 抽出したコーディングスタイルの特徴量 59 項目のうち 28 項目で, 盗用関係にあるソースコード間と盗用関係にないソースコード間で差があった.
  - 8 項目で, プログラム内容が変わっても共通して, 盗用関係にあるソースコード間と盗用関係にないソースコード間で差があった.
  - 以下の 3 項目で, プログラム内容の違いおよびソースコード整形ツールによる改ざんの有無にかかわらず盗用関係にあるソースコード間と盗用関係にないソースコード間で差があった.
    - スコープレベルごとの平均とインデントの差の平均
    - 全演算子に対する二項代入演算子の出現割合
    - 演算子の行あたりの出現回数
- コーディングスタイルから抽出される特徴量を用いた盗用の判別は, 一定以上の精度が期待できる.
  - 線形判別分析の結果から, 異なる年度や学期において同じ課題を出す場合などに有効と考えられる.
  - 適合率よりも再現率が高くなる傾向があるため, 盗用の検出よりも盗用関係を含むと考えられる集合の絞り込みに適していると考えられる.

今後の課題としては, さらなるデータの追加, 新たな特徴量の抽出が挙げられる. 今回用いた 2 つの演習課題において, ソースコード間の差分の平均値に有意差がなくても, 一般的には差分の平均値に有意差をもつ特徴量がある可能性が考えられる. さらに, 今回抽出したものとは異なる特徴量が盗用の判別に有効である可能性も考えられる. これらの可能性を確かめることで判別精度の向上が期待される.

謝辞 神戸大学大学院工学研究科の中村匡秀准教授には貴重な実験データをご提供いただきました. 心より感謝申し上げます.

また, 本研究の一部は, 文部科学省科学研究費補助金 (基盤 C：課題番号 19500056) による助成を受けた.

参 考 文 献

1) Tamada, H., Nakamura, M., Monden, A. and Matsumoto, K.: Java Birthmarks – Detecting the Software Theft, *IEICE Transactions on Information and Systems*, Vol.E88-D, No.9, pp.2148–2158 (2005).
2) 岡本圭司, 玉田春昭, 中村匡秀, 門田暁人, 松本健一：API 呼び出しを用いた動的バースマーク, 電子情報通信学会論文誌 D, Vol.J89-D, No.8, pp.1751–1763 (2006).
3) 肥後芳樹, 楠本真二, 井上克郎：コードクローン検出とその関連技術, 電子情報通信学会論文誌 D, Vol.J91-D, No.6, pp.1465–1481 (2008).
4) Prechelt, L., Malpohl, G. and Phlippsen, M.: JPlag: Finding plagiarisms among a set of programs, Technical report, University of Karlsruhe, Department of Informatics (2000).
5) 岡原 聖, 真鍋雄貴, 山内寛己, 門田暁人, 松本健一, 井上克郎：ソースコード流用のコードクローンメトリクスに基づく検出手法, 電子情報通信学会技術研究報告, Vol.109, No.307, pp.73–78 (2009).
6) Ji, J., Park, S., Woo, G. and Cho, H.: Understanding the Evolution Process of Program Source for Investigating Software Authorship and Plagiarism, *Proceedings of IEEE International Conference on Digital Information Management*, pp.92–97 (2007).
7) 永井洋一, シムウォンボ, 三輪 誠, 近山 隆：機械学習を用いたプログラムの表層的特徴による分類, 第 9 回プログラミングおよびシステムに関するワークショップ 2006 (2006).
8) 岩間 太, 中村大賀：コーディングスタイルに基づくメトリクスを用いたソースコードからの属人性検出, ソフトウェア工学の基礎 15, 近代科学社 (2008).
9) Meadow: http://www.meadowy.org/.