

コーディングスタイルの違いを利用 したソフトウェア再利用の検出

B4 : 西山 知弥

背景

- OSSの種類が多岐にわたり, 開発者が身近にOSSに接するようになった現在, OSSが安易に再利用されがち
→ 意図しない混入でライセンス違反を起こす事例が発生
- 適切な管理を行わずにOSSを再利用すると, ソフトウェアの保守性やセキュリティが損なわれる可能性がある[1]
- ソースコード混入を目で見てチェックして防ぐには限界がある

先行研究

- コードクローン[2] . . . ソースコード中での類似または一致した部分
- ソフトウェア構成分析(SCA) [3]

[2] S. Kim and H. Lee, “Software systems at risk: An empirical study of cloned vulnerabilities in practice,” *Computers & Security*, vol. 77, pp. 720–736, 2018.

[3] Software Composition Analysis Explained, WhiteSource, 2019,
<https://resources.whitesourcesoftware.com/blog-whitesource/software-composition-security-analysis>.

研究の動機

- **なぜコーディングスタイル？**

コピペを検出するのではなく、書き方が違うことを検出できれば良い

→比較対象がいらない

- 事前準備のや計算のコストが小さい？

目的とアプローチ

目的

ソフトウェア(ソースコード)の再利用を検出する

アプローチ

コーディングスタイルの違いを利用する

コーディングスタイル

- プログラミング言語の文法とは異なる
- プログラムを読みやすくするために、字下げや空白のあけ方, 変数名の付け方などに定める一定のルール
- コーディングスタイルに則ったコードは可読性を向上させる[4]

Checkstyle

- Checkstyleは、プログラマーがコーディングスタイルに則ったJavaコードを書くのを補助する開発ツール
- インデント, サイズや変数名の付け方, スペースの入れ方など, コーディングスタイルに関するチェックを行うことができる
- Eclipseやビルドツールに組み込んだりして使用できる



Checkstyleの構造

```
<?xml version="1.0"?>
<!DOCTYPE module PUBLIC
    "-//Puppy Crawl//DTD Check Configuration 1.3//EN"
    "http://www.puppcrawl.com/dtds/configuration_1_3.dtd">

<module name="Checker">
    <module name="FileLength">
        <property name="max" value="5"/>
    </module>
</module>
```

- Checkstyle の設定は xml で記述する.
- Checkstyle のルールは**モジュール**と呼ばれる単位で提供されている.
- 設定ファイルは, このモジュールを指定することで記述する.
- 設定ファイルのルートは, name に Checker を指定した <module> タグにする.
- この下に, 適用したいモジュールを <module> タグで記述していく.
- 上記例では, FileLength というモジュールの使用を宣言している.
- <module> にパラメータを渡す必要がある場合は, <property> タグを使用する.

予備調査

- GitHubからスター数で降順に1000件のJavaプロジェクトを選定
- checkstyle.xml(及びxmlで書かれた設定ファイル)を含むものを100件選定
- Pythonで*xmlを読み込んで,使われているルールを検出

*<https://docs.python.org/ja/3/library/xml.etree.elementtree.html>

チェックスタイルの違い

- チェックスタイルの違いを調査し対応表を作成

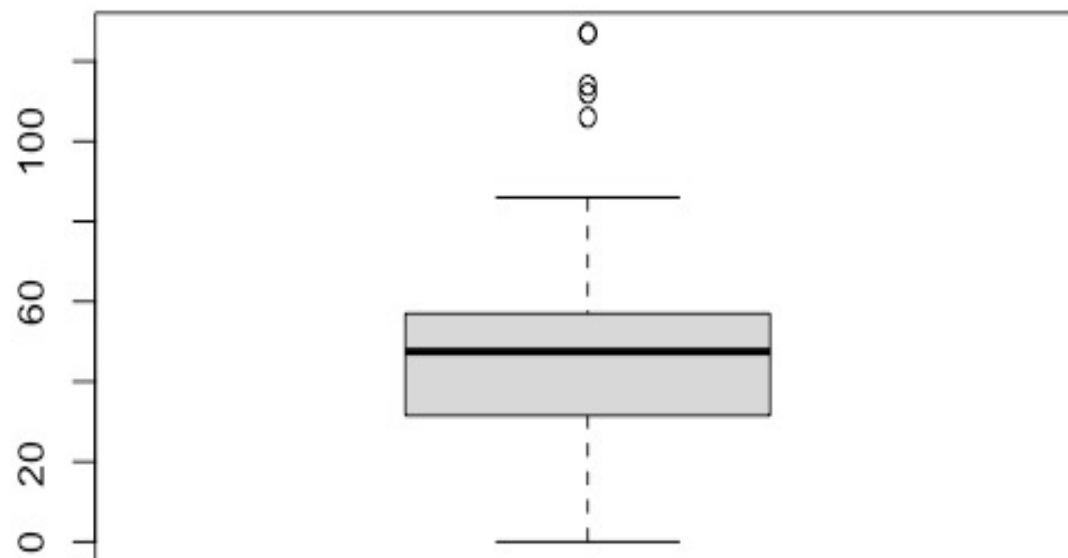
プロジェクト名

ルール名

NaN	Google	Netflix_Hystr	Sun	apache_shar	dromara_she	dromara_hmi	apereo_cas	apache_incu
AnotationLocation	0	0	0	0	0	0	0	0
AnnotationOnSameLine	0	0	0	0	0	0	0	0
AnnotationUseStyle	0	0	0	1	1	1	1	1
MissingDeprecated	0	0	0	1	1	1	1	1
MissingOverride	0	0	0	1	1	1	1	1
PackageAnnotation	0	0	0	0	0	0	0	1
SuppressWarnings	0	0	0	1	1	1	0	0
SuppressWarningsHolder	0	0	0	1	1	1	0	1
AvoidNestedBlocks	0	1	1	1	1	1	1	1
EmptyBlock	1	1	1	1	1	1	1	1
EmptyCatchBlock	1	0	0	1	1	1	1	0
LeftCurly	1	1	1	1	1	1	1	1

*有効なルール→1, 無効なルール→0

有効なルール数



- 平均・・・約 44
- 30～60のルールを使用しているプロジェクトが多かった

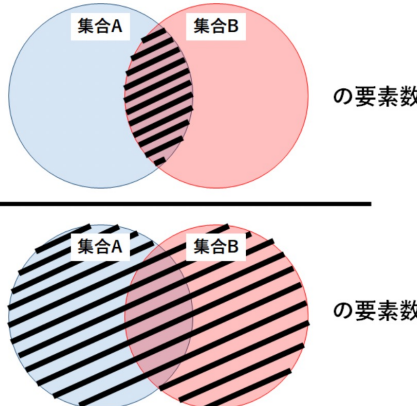
類似度の計算

コサイン類似度・・・

ルールがある→1, ない→0 としてベクトルにして
類似度を計算

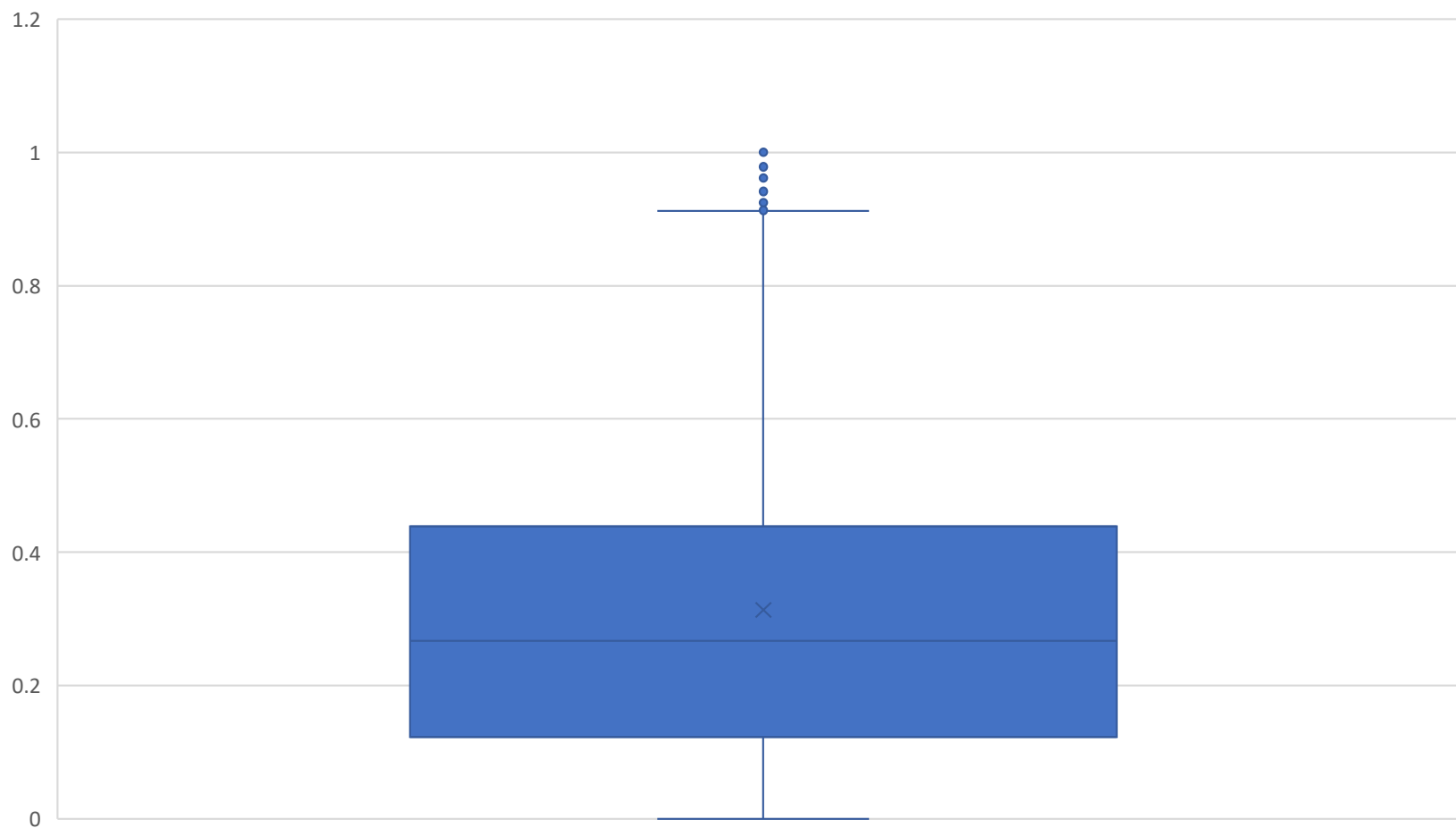
* 次元の大きさの影響を受けて使えなかった

ジャッカード類似度・・・存在しているルール名
の集合を作成し, 集合どうしの類似度を計算


$$J(A,B) = \frac{\text{集合Aと集合Bの共通要素数}}{\text{集合Aと集合Bの要素数の和}}$$

ジャッカード類似度の結果

- 類似度の平均 . . . 0.27
中央値 . . . 0.31



ヒートマップの作成

- ルール数で降順にプロジェクトを並び替えた
→ ルール数が極端に少ないと, 類似しているものが少ない

NaN	apache_sh	apache_sh	dromara_sl	dromara_hi	apereo_ca	apache_inc	androidann	Tencent_ti	scribejava_	powermock	redisson_r	skylot_jadx	halo-dev_hi	alibaba_na	citerus_dd	Netflix_Hys	TeamNewP	Netflix_zuul	Netflix_eur
apache_sh	1	1	0.88189	0.880952	0.593103	0.562963	0.571429	0.587786	0.473282	0.443609	0.44186	0.434109	0.390977	0.423077	0.390977	0.383459	0.383459	0.383459	0.383459
apache_sh	1	1	0.88189	0.880952	0.593103	0.562963	0.571429	0.587786	0.473282	0.443609	0.44186	0.434109	0.390977	0.423077	0.390977	0.383459	0.383459	0.383459	0.383459
dromara_sl	0.88189	0.88189	1	0.964912	0.602941	0.609756	0.580645	0.598361	0.525424	0.443548	0.441667	0.470085	0.433333	0.470085	0.409836	0.413223	0.413223	0.413223	0.413223
dromara_hi	0.880952	0.880952	0.964912	1	0.576642	0.593496	0.577236	0.608333	0.521368	0.45082	0.425	0.465517	0.440678	0.478261	0.404959	0.408333	0.408333	0.408333	0.408333
apereo_ca	0.593103	0.593103	0.602941	0.576642	1	0.532258	0.528455	0.484127	0.482759	0.315385	0.434783	0.366667	0.344262	0.366667	0.477477	0.495413	0.468468	0.495413	0.495413
apache_inc	0.562963	0.562963	0.609756	0.593496	0.532258	1	0.69697	0.477876	0.56701	0.438095	0.510417	0.469388	0.285714	0.345794	0.5	0.537634	0.537634	0.537634	0.537634
androidann	0.571429	0.571429	0.580645	0.577236	0.528455	0.69697	1	0.5	0.530612	0.40566	0.554348	0.526882	0.290909	0.352381	0.494737	0.5	0.516129	0.5	0.5
Tencent_ti	0.587786	0.587786	0.598361	0.608333	0.484127	0.477876	0.5	1	0.460784	0.383178	0.463918	0.453608	0.602273	0.658824	0.424242	0.443299	0.443299	0.443299	0.443299
scribejava_	0.473282	0.473282	0.525424	0.521368	0.482759	0.56701	0.530612	0.460784	1	0.343434	0.443182	0.5	0.340426	0.4	0.448276	0.470588	0.470588	0.470588	0.470588
powermock	0.443609	0.443609	0.443548	0.45082	0.315385	0.438095	0.40566	0.383178	0.343434	1	0.340426	0.373626	0.25	0.302083	0.25	0.265306	0.278351	0.265306	0.265306
redisson_r	0.44186	0.44186	0.441667	0.425	0.434783	0.510417	0.554348	0.463918	0.443182	0.340426	1	0.4	0.239583	0.307692	0.525641	0.532468	0.552632	0.532468	0.532468
skylot_jadx	0.434109	0.434109	0.470085	0.465517	0.366667	0.469388	0.526882	0.453608	0.5	0.373626	0.4	1	0.388235	0.475	0.325843	0.344828	0.376471	0.344828	0.344828
halo-dev_hi	0.390977	0.390977	0.433333	0.440678	0.344262	0.285714	0.290909	0.602273	0.340426	0.25	0.239583	0.388235	1	0.84375	0.255319	0.271739	0.285714	0.271739	0.271739
alibaba_na	0.423077	0.423077	0.470085	0.478261	0.366667	0.345794	0.352381	0.658824	0.4	0.302083	0.307692	0.475	0.84375	1	0.311111	0.329545	0.360465	0.329545	0.329545
citerus_dd	0.390977	0.390977	0.409836	0.404959	0.477477	0.5	0.494737	0.424242	0.448276	0.25	0.525641	0.325843	0.255319	0.311111	1	0.918033	0.8	0.918033	0.918033
Netflix_Hys	0.383459	0.383459	0.413223	0.408333	0.495413	0.537634	0.5	0.443299	0.470588	0.265306	0.532468	0.344828	0.271739	0.329545	0.918033	1	0.84127	1	1
TeamNewP	0.383459	0.383459	0.413223	0.408333	0.468468	0.537634	0.516129	0.443299	0.470588	0.278351	0.552632	0.376471	0.285714	0.360465	0.8	0.84127	1	0.84127	0.84127
Netflix_zuul	0.383459	0.383459	0.413223	0.408333	0.495413	0.537634	0.5	0.443299	0.470588	0.265306	0.532468	0.344828	0.271739	0.329545	0.918033	1	0.84127	1	1
Netflix_eur	0.383459	0.383459	0.413223	0.408333	0.495413	0.537634	0.5	0.443299	0.470588	0.265306	0.532468	0.344828	0.271739	0.329545	0.918033	1	0.84127	1	1
OpenFeign	0.383459	0.383459	0.413223	0.408333	0.495413	0.537634	0.5	0.443299	0.470588	0.265306	0.532468	0.344828	0.271739	0.329545	0.918033	1	0.84127	1	1
Netflix_riib	0.383459	0.383459	0.413223	0.408333	0.495413	0.537634	0.5	0.443299	0.470588	0.265306	0.532468	0.344828	0.271739	0.329545	0.918033	1	0.84127	1	1
apache_hiv	0.415385	0.415385	0.436975	0.444444	0.468468	0.554348	0.532609	0.414141	0.488095	0.305263	0.573333	0.360465	0.231579	0.3	0.746269	0.784615	0.731343	0.784615	0.784615
Netflix_Simi	0.37594	0.37594	0.404959	0.4	0.486239	0.526882	0.489362	0.43299	0.458824	0.268041	0.539474	0.333333	0.274725	0.333333	0.901639	0.982759	0.825397	0.982759	0.982759
real-logic_a	0.389313	0.389313	0.420168	0.42735	0.364407	0.438776	0.43299	0.393939	0.397727	0.340659	0.45	0.402439	0.263736	0.337209	0.455696	0.443038	0.480519	0.443038	0.443038
apache_ca	0.389313	0.389313	0.396694	0.391667	0.364407	0.41	0.40404	0.352941	0.413793	0.370787	0.468354	0.292135	0.173469	0.223404	0.369048	0.373494	0.390244	0.373494	0.373494
Genymobile	0.389313	0.389313	0.420168	0.415254	0.463636	0.516129	0.463158	0.4375	0.481928	0.284211	0.567568	0.369048	0.277778	0.337209	0.825397	0.9	0.809524	0.9	0.9
oracle_gra	0.343284	0.343284	0.380165	0.386555	0.419643	0.544444	0.457447	0.373737	0.406977	0.290323	0.443038	0.361446	0.269663	0.313953	0.486842	0.513514	0.555556	0.513514	0.513514
apache_stu	0.409449	0.409449	0.45614	0.464286	0.316667	0.301887	0.307692	0.607143	0.37931	0.27957	0.269663	0.4	0.836066	0.806452	0.258427	0.275862	0.290698	0.275862	0.275862
square_ott	0.376923	0.376923	0.40678	0.401709	0.449541	0.5	0.494505	0.451613	0.518987	0.252632	0.547945	0.365854	0.258427	0.317647	0.806452	0.85	0.761905	0.85	0.85
Alluxio_allux	0.376923	0.376923	0.418803	0.413793	0.385965	0.468085	0.416667	0.534091	0.445783	0.265957	0.313953	0.454545	0.555556	0.6	0.417722	0.441558	0.441558	0.441558	0.441558
sshahine_J	0.401575	0.401575	0.447368	0.455357	0.319328	0.292453	0.298077	0.614458	0.352273	0.268817	0.244444	0.3875	0.881356	0.819672	0.261364	0.27907	0.27907	0.27907	0.27907

100×100で大きいので, 上図は一部

Netflix関連

	Netflix_Hystrix	Netflix_zuul	Netflix_eureka	Netflix_ribbon	Netflix_SimianArmy
Netflix_Hystrix	1	1	1	1	0.982758621
Netflix_zuul	1	1	1	1	0.982758621
Netflix_eureka	1	1	1	1	0.982758621
OpenFeign_feign	1	1	1	1	0.982758621
Netflix_ribbon	1	1	1	1	0.982758621
Netflix_SimianArm	0.982758621	0.98275862	0.982758621	0.982758621	1

- Netflix関連のプロジェクトの類似度が高い
→ 企業でコーディングスタイルが統制されている可能性

	Netflix_Hystrix	Netflix_zuul	Netflix_eureka	Netflix_ribbon	Netflix_SimianArmy
google_goog	0.265957447	0.26595745	0.265957447	0.265957447	0.268817204
Sun_Sun	0.90625	0.90625	0.90625	0.90625	0.890625

- Netflix関連プロジェクトとSunとの類似度が高い
→ Sunのチェックを一部変更して使っているのでは？

SunとNetflixとの違いを調査した結果

	Sun	Netflix
DesignForExtension	○	×
MultipleVariableDeclarations	○	×
InvalidJavadocPosition	○	×
JavadocPackage	○	×
MissingJavadocMethod	○	×
FinalParameters	○	×

- Netflixに存在かつ Sunに非存在のルールがなかった
→ Sunのルールから, 必要ないルールを削除している

異なるプロジェクトで類似している

→ (仮) 元々がGoogleやSunと類似？

目視するとNetflixと類似度が高いものは, Sunとも類似度が高かった

ファイル単位での違反の傾向

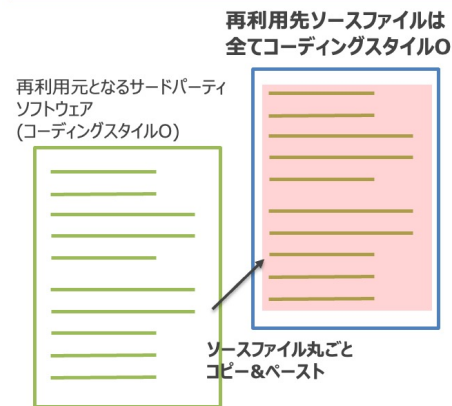
EmptyForInitializerPad	0	0	0	0	0	0
EmptyForIteratorPad	0	0	0	0	0	0
EmptyLineSeparator	249	2	3	33	116	125
FileTabCharacter	0	0	0	206	0	0
GenericWhitespace	0	0	0	0	0	0
MethodParamPad	21	8	0	0	0	4
NoLineWrap	0	0	0	0	0	0
NoWhitespaceAfter	7	0	1	0	1	2
NoWhitespaceBefore	0	0	0	0	0	1
OperatorWrap	18	0	17	0	8	2
ParenPad	9	0	1	0	4	7
SeparatorWrap	0	0	0	0	0	0
SingleSpaceSeparator	0	8	0	18	0	3
TypecastParenPad	0	0	0	0	0	0
WhitespaceAfter	32	1	7	213	8	5
WhitespaceAround	34	0	13	850	32	12
合計違反数	1359	50	448	5916	629	566
ファイル行数	6738	5117	4414	3962	3524	3264
違反数/ファイル行数	0.201691897	0.00977135	0.10149524	1.49318526	0.17849035	0.17340686

今後の流れ

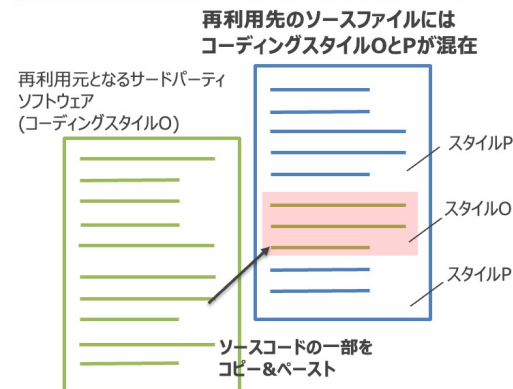
- 検出の手法を考える
- コードクロールン関連の論文を読んで, データセットを探す

再利用のパターン

1. ソースファイルを丸ごと再利用



2. コードの一部を再利用



1. 検出された違反の類似度を計算する

→ 他のファイルと違反の類似度が低いものは、別の所からコピーされた可能性が高い

2. 他のOSSが再利用されている部分は、違反の性質(数, 種類)が異なる

コードブロック単位でCheckstyle 違反を集計

- ブロック単位でコードを分割
- コマンドライン上で, Checkstyleを実行
- 結果はテキストファイルとして得られる

例)

```
:1:1: パッケージ名はディレクトリと同じではありません。 [PackageDeclaration]
:3:14: 名前 'hello' はパターン '^([A-Z][a-zA-Z0-9]*)$' に一致しなければなりません。 [TypeName]
:5:1: 行にタブ文字が含まれています。 [FileTabCharacter]
:5:9: 'method def modifier' のインデントレベル 8 は正しくありません。期待されるレベルは 4 です。
[Indentation]
:5:9: コメントではない main メソッドが見つかりました。 [UncommentedMain]
:5:33: パラメータ args は final にしてください。 [FinalParameters]
:6:1: 行にタブ文字が含まれています。 [FileTabCharacter]
```

*サンプル

```
1 package test;
2
3 public class hello {
4
5     public static void main(String[] args) {
6         //自分で書いたコード
7         System.out.println("hello world");
8         int A = 1;
9         int B = 5;
10        int C;
11        C = A+B;
12        //System.out.println(C);
13        //
14        for(int i=1; i<=10; i++) {
15            System.out.println(i);
16        }
17        //ここからコピペ
18        for(int i = 1; i <= 9; i++) {
19            for(int j = 1; j <= 9; j++){
20                System.out.print(" " + i * j);
21            }
22            //改行
23            System.out.println();
24        }
25    }
26 }
```

集計したテキストの分析

1. 同じテキストに何度も出てくる違反

⇒ そのブロックの違反の特徴

2. 様々なテキストに出現する違反

⇒ 違いとは言えない

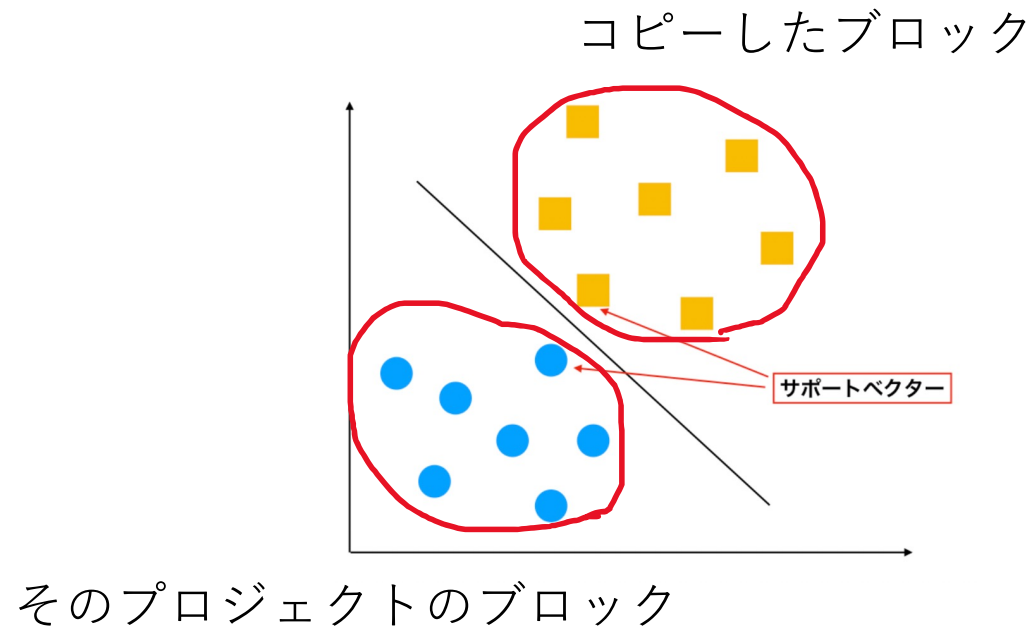
テキストをベクトル化して類似度を計算する

	ルールA	B	C	D	E	...
ファイル名	0	2	1	0	3	...

*違反されたルールをカウント

分類

- SVMなどで分類してみる



<https://qiita.com/3000manJPY/items/a0652d488ce3c956613d>

<https://aiacademy.jp/media/?p=248>