機械学習を用いた
メソッド抽出リファクタリングの推薦手法

後藤 祥(阪大), 吉田 則裕(名大),
藤原 賢二(NAIST), 崔 恩瀞, 井上 克郎(阪大)

メソッド抽出リファクタリング

- メソッドの一部を新たなメソッドとして抽出するリファクタリングパターン
  - 頻繁に行われるリファクタリングパターンの1つ
- 主な用途
  - 行数の長いメソッドを短いメソッドに分割する
  - 複数の機能が実装されたメソッドを機能単位に分割する

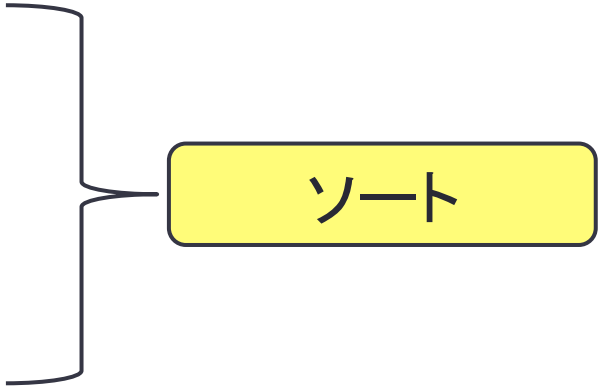
メソッド抽出の例(1/2)

- 1-10までの成績を並び替えて表示するプログラム

```
int[] score = {9,4,8,2,3,10,7,1,6,5};


for(int i=0;i<score.length-1;i++){
 for(int j=0;j<score.length-i-1;j++){
  if(score[j] < score[j+1]){
   int t = score[j];
   score[j] = score[j+1];
   score[j+1] = t;
  }
 }
}

for(int i=0;i<score.length;i++){
 System.out.println(score[i]);
}
```



メソッド抽出の例(2/2)

- 1-10までの成績を並び替えて表示するプログラム

ソート部分をメソッド抽出

```
int[] score = {9,4,8,2,3,10,7,1,6,5};

sort(score);


for(int i=0;i<score.length;i++){
 System.out.println(score[i]);
}
```

```
public void sort(int[] score){
 for(int i=0;i<score.length-1;i++){
  for(int j=0;j<score.length-i-1;j++){
   if(score[j] < score[j+1]){
    int t = score[j];
    score[j] = score[j+1];
    score[j+1] = t;
   }
  }
 }
}
```

メソッド名から処理内容を
把握することができる.

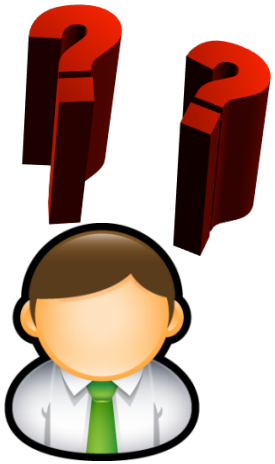
ソート処理を行っている
部分が特定できる.

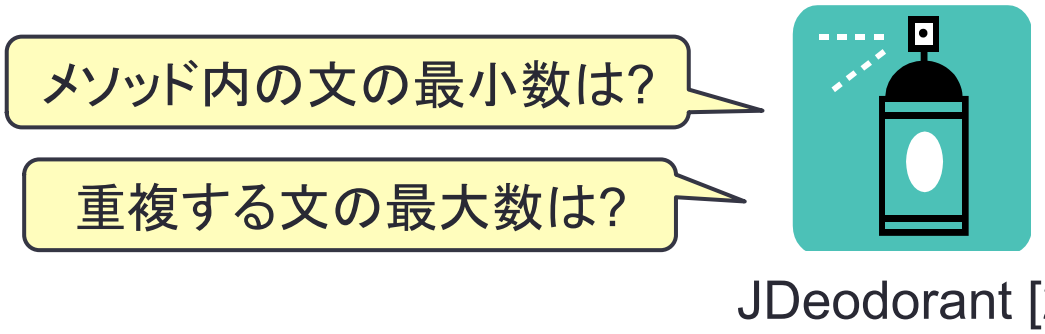
リファクタリング対象の推薦

- 対象を自動で特定して開発者に推薦する
- Bad Smell(リファクタリングが推奨されるコード)
  - Long Method, Duplicated Code

推薦における問題点

- プロジェクトや開発者による基準の違い
- 多くのツールはパラメータ設定によって対応
  - パラメータ設定はあまり変更されない [1]
  - 設定値から結果を想定するのは困難



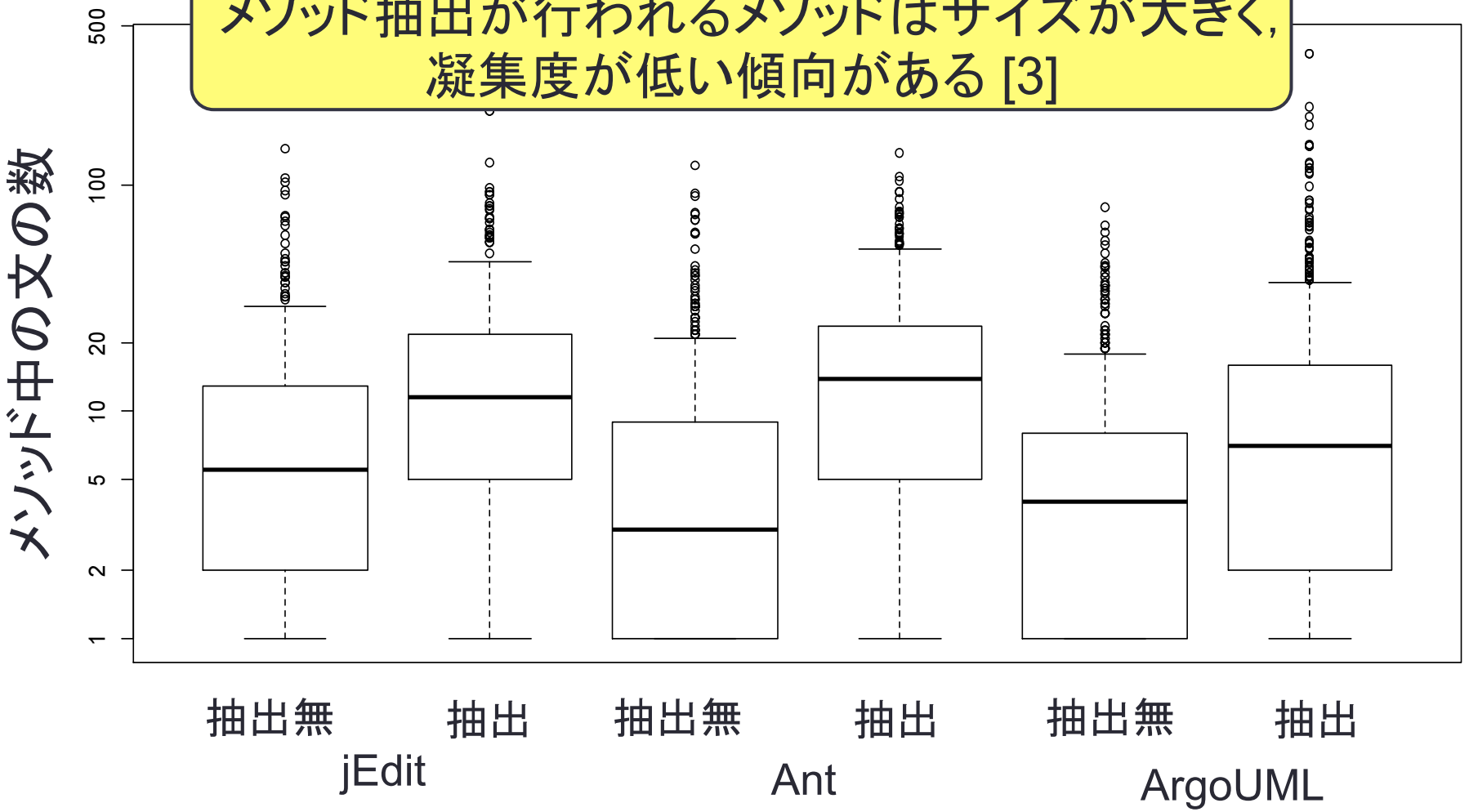


JDeodorant [2]

[1] E. Murphy-Hill et al. “How We Refactor, and How We Know It”, IEEE Trans. on Softw. Eng., 2011.
[2] jDeodorant : http://www.jdeodorant.com/

既存研究:
メソッド抽出が行われるメソッドの特徴

メソッド抽出が行われるメソッドはサイズが大きく,
凝集度が低い傾向がある [3]

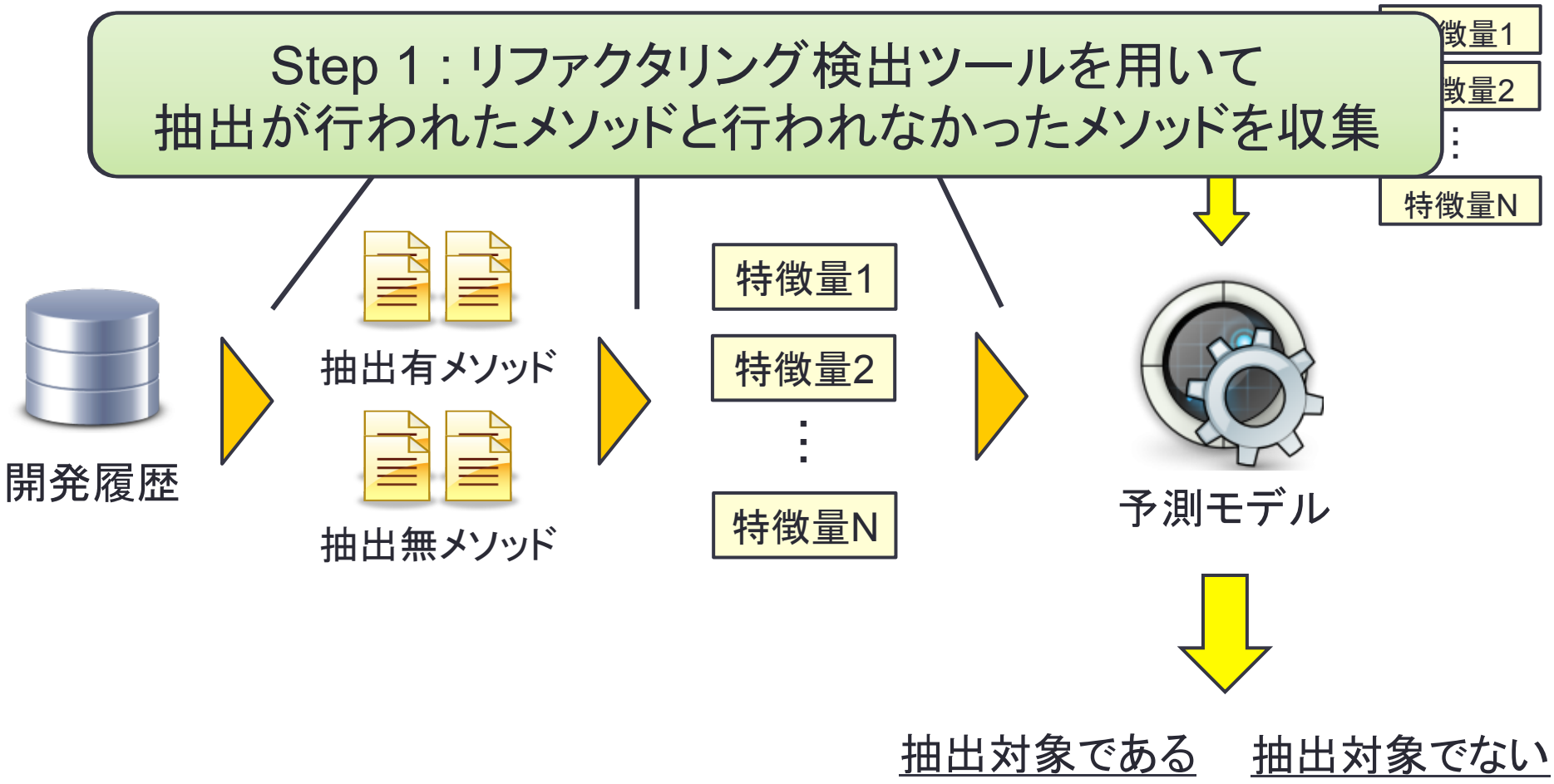


[3]後藤ら: "メソッド抽出リファクタリングが行われるメソッドの特徴調査", コンピュータソフトウェア, 2014

提案手法

- 機械学習を用いたメソッド抽出リファクタリングの推薦
  - 過去にリファクタリングが行われた事例から，ソースコードの特徴量を抽出し，学習を行う
- 機械学習を用いることの利点
  - プロジェクトごとの基準を反映させることができる
  - 他のリファクタリングパターンへの拡張性がある

提案手法の概要


Step 1 : リファクタリング検出ツールを用いて
抽出が行われたメソッドと行われなかったメソッドを収集
徴量1
徴量2
特徴量N
開発履歴
抽出有メソッド
抽出無メソッド
特徴量1
特徴量2
特徴量N
予測モデル
抽出対象である
抽出対象でない

Step 1 : メソッド抽出事例の収集

- リファクタリング検出ツールを使用して収集
  - 開発履歴中から，リファクタリングが適用された箇所を検出するツール
- 藤原らが提案しているリファクタリング検出ツールを使用 [4]
  - 高速かつ高精度で検出が可能

[4] 藤原ら “ソフトウェアリポジトリを対象とした細粒度リファクタリング検出”, 第20回ソフトウェア工学の基礎ワークショップ, 2013.

Step 2 : 特徴量の計測

カテゴリ	特徴量
サイズ	メソッド中の文数
シグネチャ	引数の数，返り値の有無，アクセスレベル
凝集度	Tightness, Coverage, Overlap
複雑度	サイクロマチック数
構文	ループ数，if文数，case文数 ブロック数，ネストの深さの最大値
変数	メソッド内のローカル変数の数
CKメトリクス	WMC, DIT, CBO, NOC, RFC, LCOM
重複コード	メソッド内の重複コードが存在するか

Step 3 : 予測モデルの構築

- データマイニングツール Weka を使用 [5]
  - 多くの機械学習アルゴリズムが実装されている
- モデル構築前に変数選択を行う
  - ラッパー法
- モデルは3種類使用
  - ロジスティック回帰，決定木，ベイジアンネット

[5] weka : http://www.cs.waikato.ac.nz/ml/weka/

実験概要

- 予測モデルの評価と有用な特徴量の調査
  - 5つのソフトウェアに提案手法を適用

ソフトウェア	メソッド抽出事例数	データセット中のメソッド数 (抽出有 : 抽出無)
Ant	766	1532 (766 : 766)
ArgoUML	740	1480 (740 : 740)
jEdit	502	1004 (502 : 502)
jFreeChart	90	180 (90 : 90)
Mylyn	490	980 (490 : 490)

実験方法

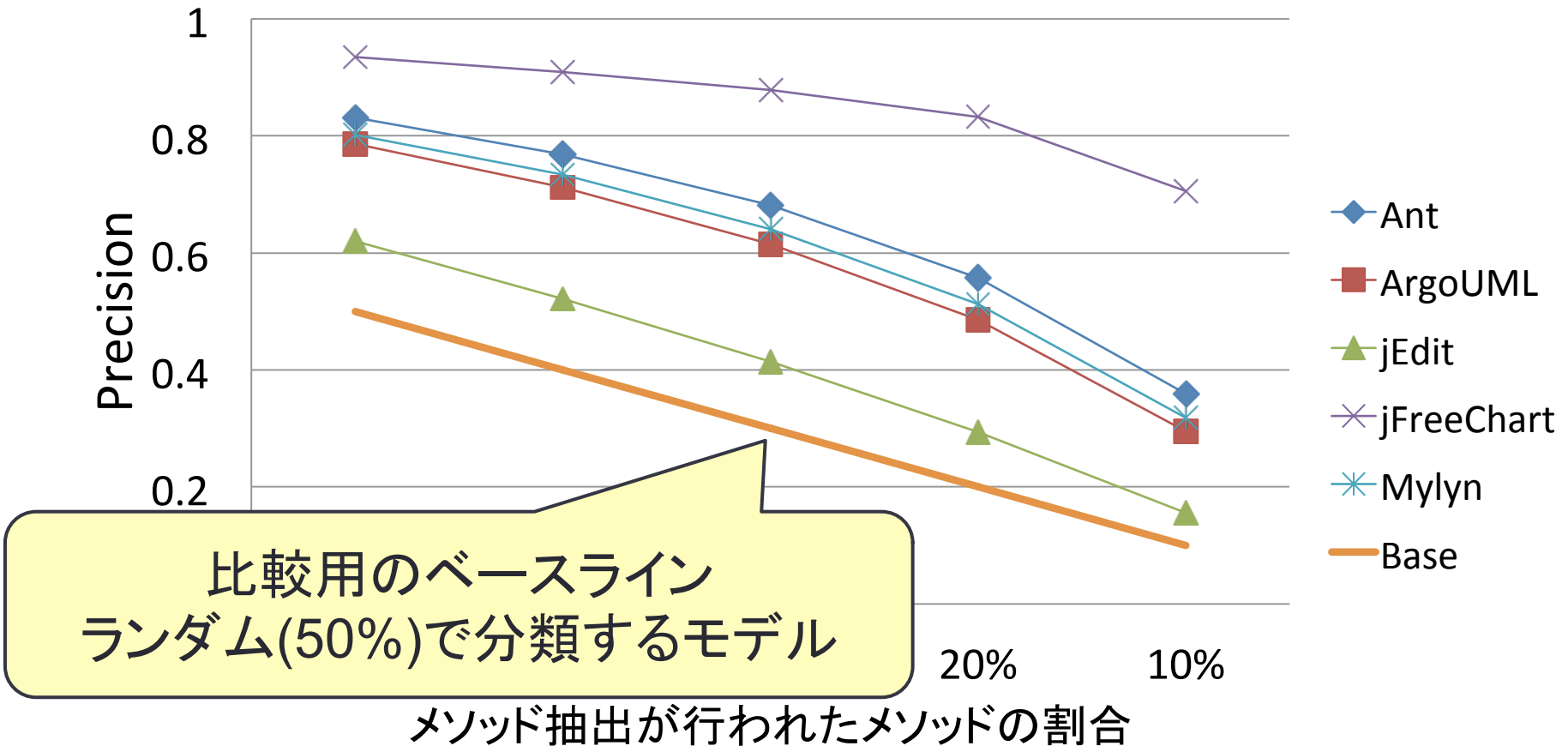
- モデルの構築，評価に交差検定を用いる
  - データセットを学習セットと評価セットに分割
- 評価セットの割合を変化させる

学習セット	評価セットの割合 (抽出有：抽出無)
50:50	50:50
	40:60
	30:70
	20:80
	10:90

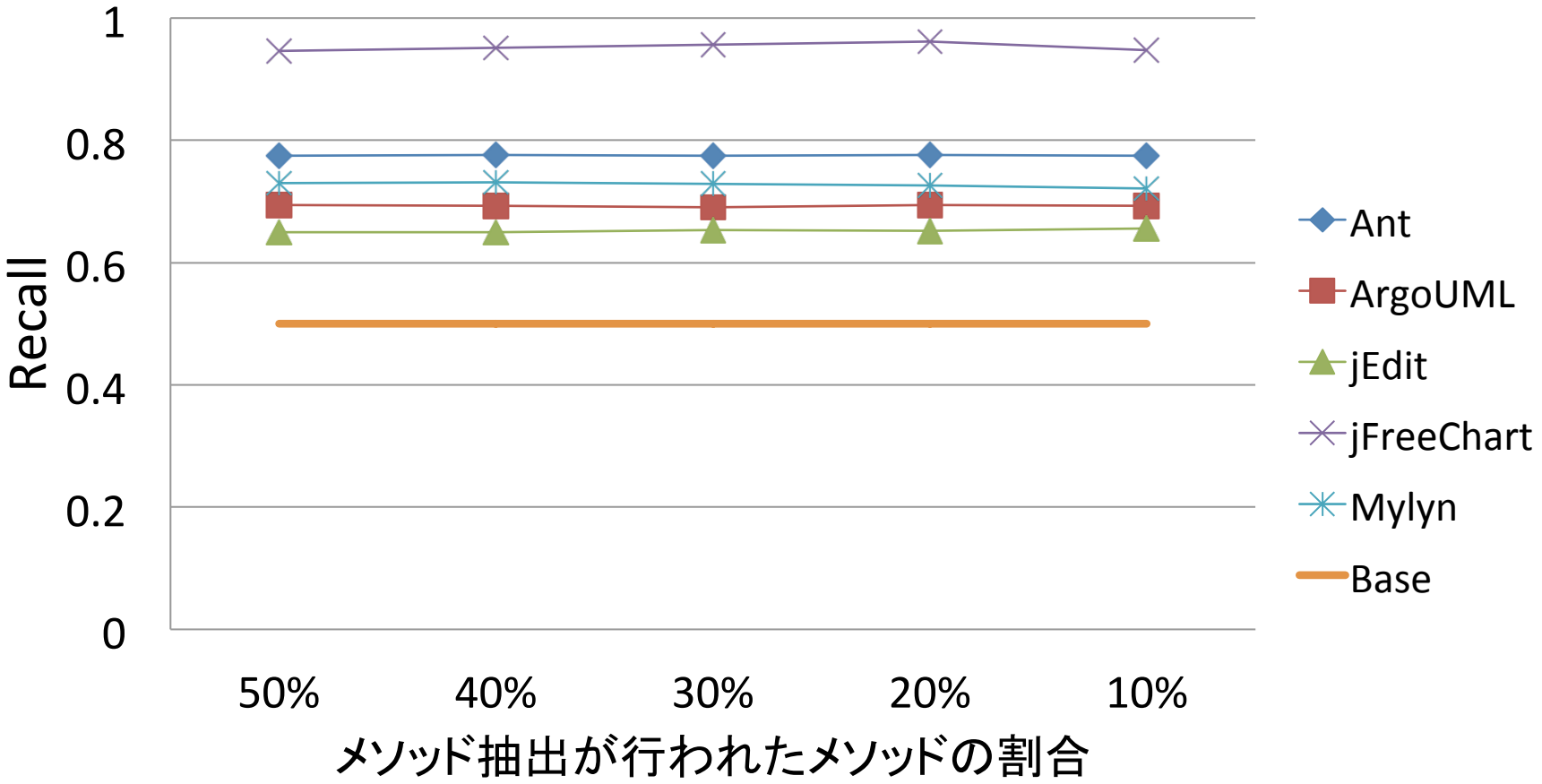
評価尺度

- Precision
  - 推薦されたメソッドのうち，抽出が行われたメソッドの割合
- Recall
  - データセット中の抽出が行われたメソッドのうち，モデルによって推薦されたメソッドの割合

実験結果：Precision



実験結果：Recall


1
0.8
0.6
0.4
0.2
0
Recall
50%
40%
30%
20%
10%
メソッド抽出が行われたメソッドの割合
Ant
ArgoUML
jEdit
jFreeChart
Mylyn
Base

実験結果：特徴量

- 変数選択によって選択された回数をもとに，有用な特徴量を調査

特徴量	選択された回数(最大15回)
メソッドの文数	13
凝集度メトリクス Coverage	13
メソッドの引数の数	12
⋮	
サイクロマチック複雑度	7
凝集度メトリクス Overlap	6
case 文数	6

実験結果に関する議論

- 全ての場合でベースラインより良い結果
  - 特徴量による学習の効果があった

- Recallに変化はないがPrecisionは減少する
  - 査読者様「必要条件の一部は判定できているが，十分条件はあまり判定できていないのでは？」

実験の設計に関する議論

- メトリクス計測ツールの出力以外の特徴量を検討すべき
  - 既存のリファクタリング候補推薦手法の出力
- 実際のリファクタリング事例を正解にすべきか
  - 優れたOSSを対象としたため，正確にリファクタリングが実施されていると仮定した
  - 査読者様「リファクタリングが行われる以前からメソッドとされているコード断片を考慮すべきでは？」
    - 既存のメソッドのインライン化をして正解とする方法が考えられる.

まとめと今後の課題

- 機械学習を用いたメソッド抽出リファクタリングの推薦手法を提案
- オープンソースのソフトウェアを対象に実験
  - メソッド抽出が行われたメソッドの6割から9割は推薦することができた
  - 実用のためにはさらなる精度の改善が必要
- 今後の課題
  - 追加すべき特徴量の検討
    - 既存のリファクタリング候補推薦手法の出力
  - 機械学習を用いない場合との比較