PREDICTION OF WHO SHOULD REFACTOR THE CODE

S. BENDAPUDI, R. KIPP, T. OLATUNBOSUN, and M. SZCZEPANIAK

TABLE OF CONTENTS

OUR PROBLEM

Established Project

Multiple developers work on and maintain a codebase

Refactoring Request

Something is noticed that needs to be rewritten in order to improve the readability, usability, and functionality of the codebase

Refactoring Triage

The request is assigned to a developer skilled in the type of refactoring needed and the area of the project in which the refactoring is needed

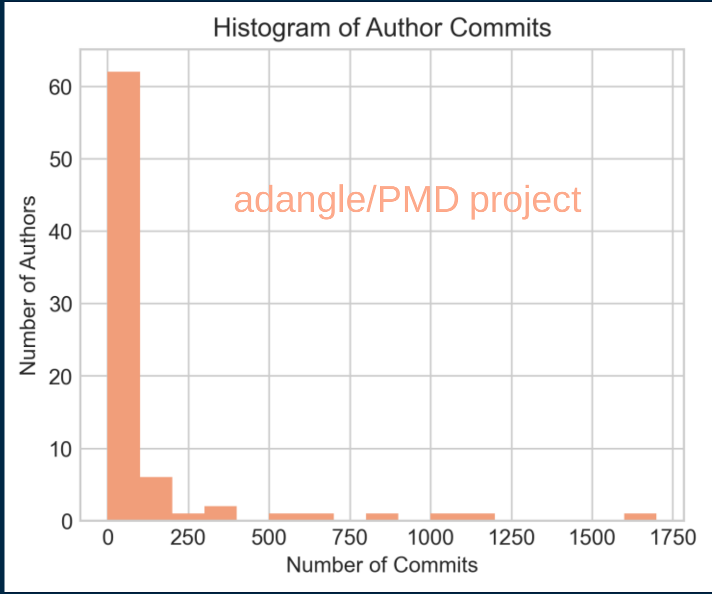
Refactoring Complete

The refactoring is completed by the developer, reviewed and then it is either accepted or reworked

UNDERSTANDING THE PROBLEM

Multiple Developers

There are a few developers with a large number of commits, and a large number of developers with a few commits. This large inequality creates imbalanced groups and complicates our problem



UNDERSTANDING THE PROBLEM

Variety of Refactoring Types

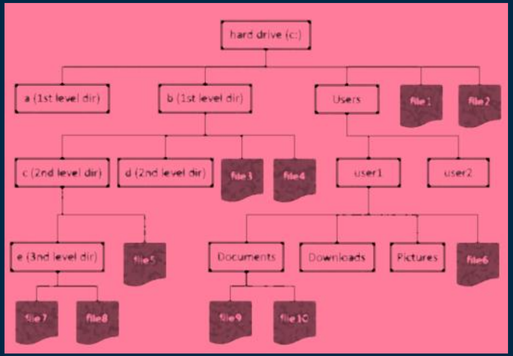
There are a 27 different refactoring types. The expectation would be that certain developers are better at some than others and visa versa. The type of refactoring that is needed is determined in the refactoring request, allowing us to use this information as part of our feature set.

Move Class, Rename Attribute, Rename Method, Extract Method, Inline Method, Rename Parameter, Extract Variable, Rename Variable, Extract Superclass, Move Method, Extract And Move Method, Parameterize Variable, Extract Subclass, Push Down Attribute, Push Down Method, Extract Class, Move Attribute, Move And Rename Class, Inline Variable, Rename Class, Pull Up Method, Replace Variable With Attribute, Move Source Folder, Pull Up Attribute, Extract Interface, Move And Rename Attribute, Change Package

UNDERSTANDING THE PROBLEM

Large File Structure

The file structure of any codebase is complex. Different areas usually correspond to different features or different parts of the program (i.e. frontend, backend, tests, etc). The assumption that different developers will be more familiar with different aspects of the project allows us to use the file structure as an input as well.



OUR RESEARCH QUESTIONS

RQ1

Which of the three classification model tends to give the lowest classification error?

RQ2

What is the trade-off in overall model performance if improved performance of the low committer group is a priority?

RQ3

Which features are most important in predicting which developer is best suited for a given refactoring request?

- We will explore these questions using data from the adangel/PMD and hibernate open-source projects

INPUTS

This is a random sample of our feature space. Although the dataset includes the commit message, the only way to make use of this information would be to compare it to the text from an actual refactoring request which were not available.

RefactoringType	L1	L2	L3	L4	L5	L6
Move Method	pmd-ui	src	main	java	net	sourceforge/pmd/util/fxdesigner/util/DesignerU...
Rename Variable	pmd	src	net	sourceforge	pmd	util/ASTViewer.java
Extract Superclass	pmd-java	src	test	java	net	sourceforge/pmd/lang/java/oom/metrics/Abstract...
Extract Method	pmd-eclipse	src	net	sourceforge	pmd	eclipse/preferences/PMDPreferencePage.java
Move Class	pmd	src	net	sourceforge	pmd	jsp/ast/ASTJspScriptlet.java

Splitting the path into 6 features allows the patterns in the hierarchy to be included in the model.

MODELS

	DESCRIPTION
NAÏVE BAYES	Assumes that the presence of a particular feature in a class is independent of each other feature
J48 DECISION TREE	Divide the data into ranges based on the attribute values found in the training sample
RANDOM FOREST	A large number of individual uncorrelated decisions trees all vote on the outcome and the probability of the class corresponds to the number of votes each class gets

PREDICTION

We formulated our approach as a classification problem rather than a ranking problem. Our response was a 4-level committer group based on the historical commit frequency defined in the table below:

Adangle/PMD	hibernate	Committer Group
6	3	Extremely High Committers (authors with >400 commits)
9	3	High Committers (authors with 100-400 commits)
8	4	Medium Committers (authors with 50-200 commits)
54	12	Low Committers (authors with <50 commits)

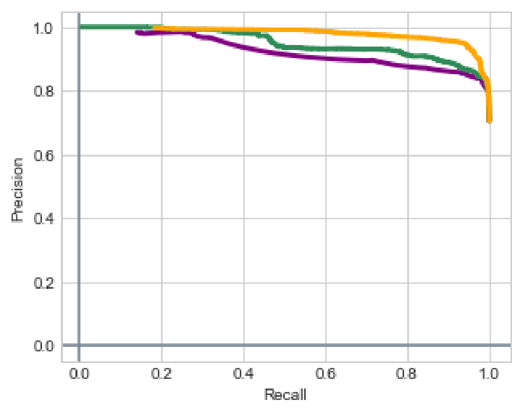
Why Classification?

1. Simple to implement
2. Simple to interpret
3. Ranking assumes a fixed effectiveness hierarchy of “best” to “worst” refactor candidates. Classification uses a looser assumption: groups (not individuals) form an effectiveness hierarchy.
4. Assumption of each committer group having similar effectiveness is likely to be more realistic than underlying ranking assumption.

Best Performing Model

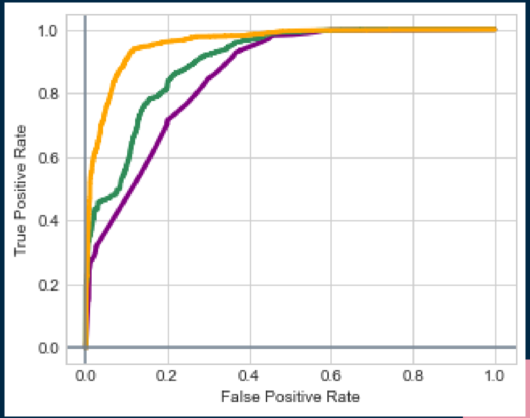
As shown on the next couple slides, Random Forest clearly outperformed the Naïve Bayes and J48 Decision Tree models in both the majority (extremely high) and the minority (low) classes based on the shapes of the ROC and PR curves.

Model Selection RESULTS (Extremely High Group)



Receiver Operating Characteristic

Precision and Recall

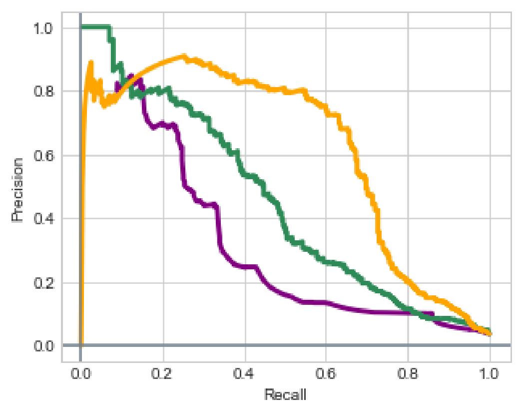


Random Forest

Naïve Bayes

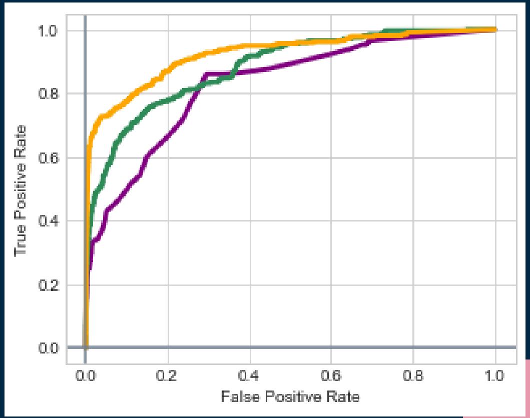
J48 Decision Tree

Model Selection RESULTS (Low Group)



Receiver Operating Characteristic

Precision and Recall



Random Forest
Naïve Bayes
J48 Decision Tree

CONFUSION MATRIX

Assumption:

- Cost (misassigning ex high to low) > Cost (misassigning low to ex high)
- In other words: Redoing Cost > Experience Cost

Redoing Cost – should be done by ex high but was not

Random Forest - adangle

a	b	c	d	<-- classified as
5815	76	19	30	a = extremely high
605	1141	13	13	b = high
68	27	281	12	c = medium
115	9	7	184	d = low

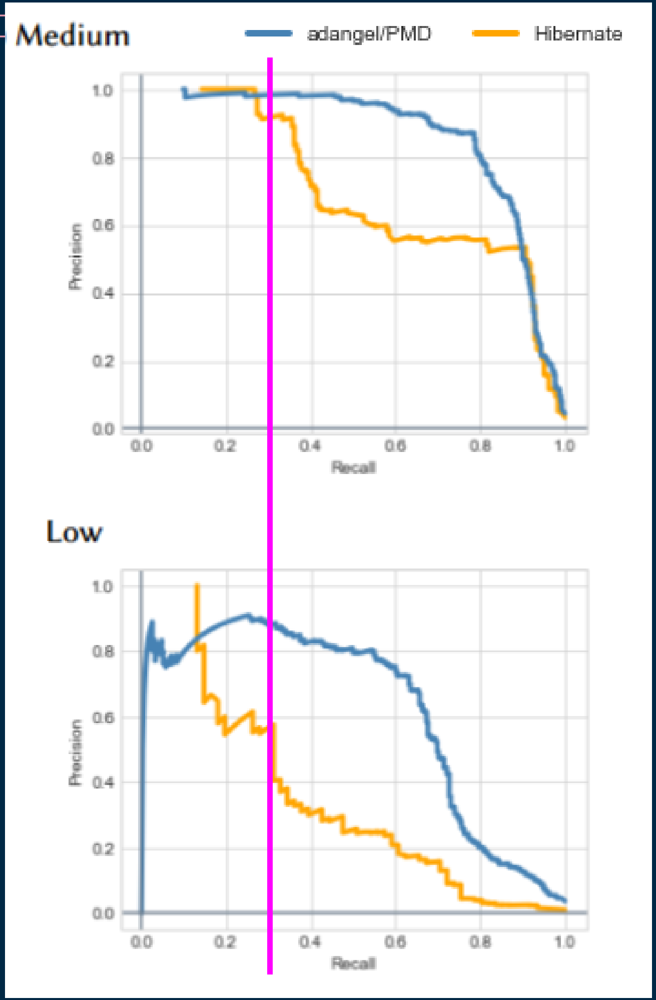
Experience Cost – should not be done by lower groups, but was not

	NB	J48	RF
Accuracy %	82.3	84.3	88.2
RMSE %	26.1	25.2	20.7

Improving Low Group Recall

Why improve the recall of low group?

- recall = proportion of actual positive correctly classified
  = (true classified lows) / (actual lows)
  = (true classified lows) / (true classified low + lows falsely classified as another class)
  = TP / (TP + FN) where TP="True Positives", FN="False Negatives"
- Assuming (Redoing Cost > Experience Cost) is correct, we want to drive up proportion of actual lows while keeping proportion of predicted lows as high as possible (precision).
- Possible solutions: over-sample minority class, under-sample majority class, combination of over and under-sampling, <u>S</u>ynthetic <u>M</u>inority <u>O</u>ver-sampling <u>Te</u>chnique (SMOTE), threshold adjustment.
- Base on van den Goorbergh et. al. (1), threshold adjustment avoids issues that arise when using over or under-sampling techniques.



Variable Importance

The top 2 most important variables in the model match our intuition:

- The last part of the file path (L6) most unique part of the file being committed.
- L1 is the first part of the file descriptor which implies the model is making the most use of the first and last parts of the file path
- Other elements of the file path (L2 – L5) repeat far more often and contain less information about the file.
- The RefactoringType provide more information to the model in the hibernate project than the adangle.

```
=== Attribute Selection on all input data ===

Search Method:
        Attribute ranking.

Attribute Evaluator (supervised, Class (nominal): 8 AuthorGroup):
        Information Gain Ranking Filter

Ranked attributes:
 0.9627    7 L6
 0.3198    2 L1
 0.1009    4 L3
 0.0982    6 L5
 0.0782    5 L4
 0.078     1 RefactoringType
 0.0552    3 L2
```

adangle model results

```
=== Attribute Selection on all input data ===

Search Method:
    Attribute ranking.

Attribute Evaluator (supervised, Class (nominal): 8 AuthorGroup):
    Information Gain Ranking Filter

Ranked attributes:
 0.54682    7 L6
 0.0647     2 L1
 0.05525    1 RefactoringType
 0.00968    6 L5
 0.00752    4 L3
 0.0069     3 L2
 0.00581    5 L4
```

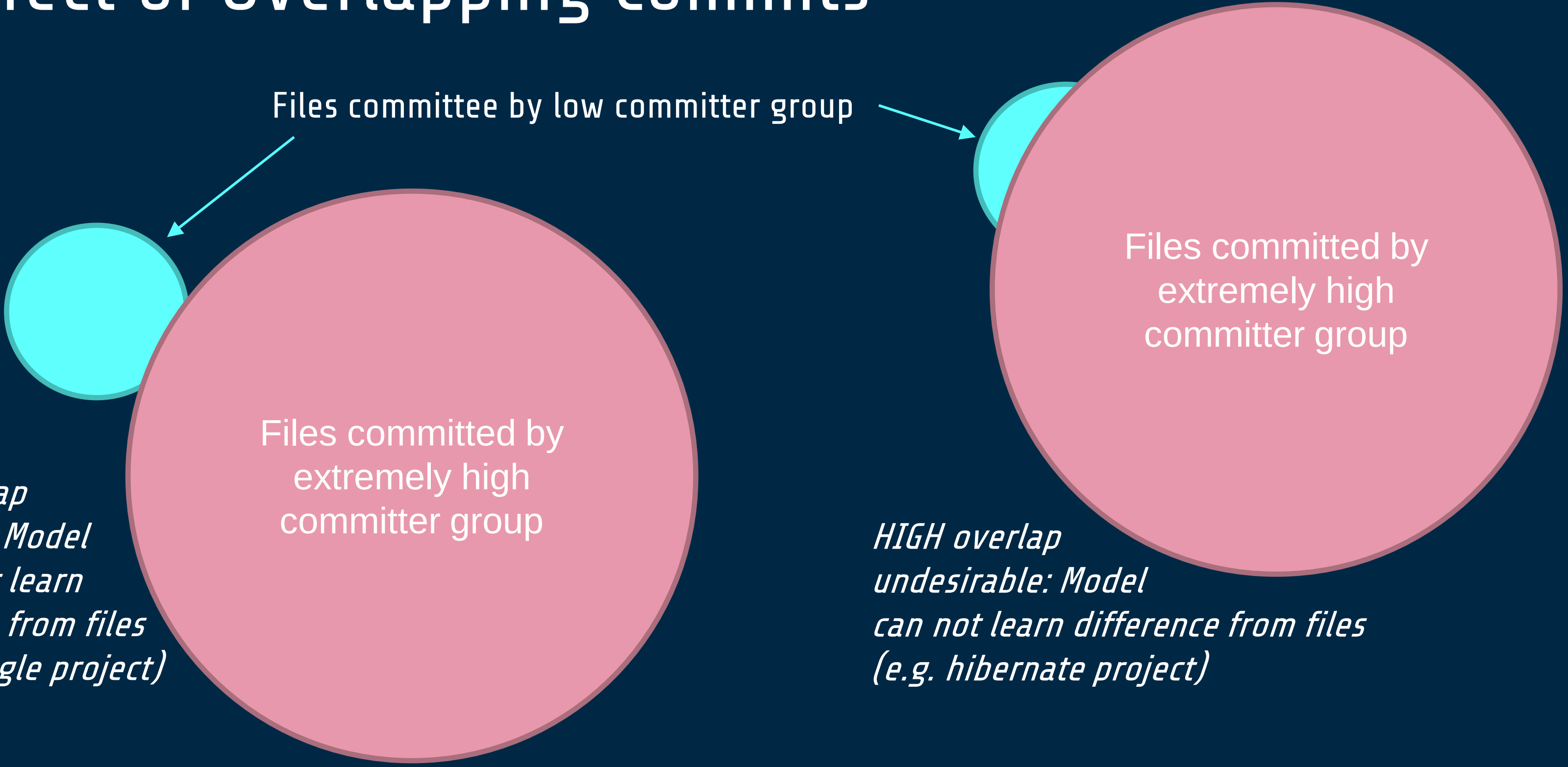
hibernate model results

THREATS TO VALIDITY

- Due to our small feature space, if the low committers and the high committers are working on similar files, the model will have a difficult time distinguishing between the classes.
    - We can at least partially explain the degradation in performance of the hibernate low and medium groups vs. adangle this way.

- Commit frequency may not accurately reflect ability to do a refactor
    - A low committer may have low commit counts for a particular project, but may be more capable because they have experience on other projects that our data is not reflecting.

Effect of Overlapping Commits



LOW overlap desirable: Model can better learn difference from files (e.g. adangle project)

HIGH overlap undesirable: Model can not learn difference from files (e.g. hibernate project)

Testing Commit Files Overlap

The table below shows significantly more overlap in low vs. extremely high and medium vs. extremely high committed files:

Commit Group	low files committed by ex high	Total committed files	proportion of total	P-value, H01: $p_{low, ada} = p_{low, hib}$ H02: $p_{med, ada} = p_{med, hib}$
adangle – low	115	315	0.3651	HA1: $p_{low, ada} < p_{low, hib}$
hibernate – low	43	63	0.6825	3.039e-06
adangle – medium	96	388	0.2474	HA2: $p_{med, ada} < p_{med, hib}$
hibernate – medium	78	183	0.4262	1.146e-05

References

1. van den Goorbergh R, van Smeden M, Timmerman D, Van Calster B The harm of class imbalance corrections for risk prediction models: Illustration and simulation using logistic regression
2. https://machinelearningmastery.com/perform-feature-selection-machine-learning-data-weka/