FAULT-PRONE FILTERING:

DETECTION OF FAULT-PRONE MODULES USING SPAM FILTERING TECHNIQUE

Osamu Mizuno,
Shiro Ikami, Shuya Nakaichi, and Tohru Kikuno
Graduate School of Information Science and Technology
Osaka University, JAPAN

ESEM 2007 presentation

Osaka Univ.

THIS WORK: AT A GLANCE

- Objective: Fault-prone filtering
  - Development of simple and easy approach to detect fault-prone modules using generic text discriminator.
- Experiment
  - SPAM filter: CRM114 (generic text discriminator)
  - Data of fault-proneness from an OSS project (Eclipse)
  - 10-fold cross validation
- Result
  - Achieved good accuracy

OVERVIEW

- **Preliminary**
- Fault-Prone Filtering
- Experiments
  - 10-fold cross validation
  - Results
- Conclusions

PRELIMINARY:
FAULT-PRONE MODULES

- Fault-prone modules are:
  - Software modules (a certain unit of source code) which may include faults.
- In this study:
  - A software module:
    - Source code of a Java method
  - Fault-prone module:
    - Source code of a Java method which seems to include faults from the information of a bug tracking system.

PRELIMINARY:
SPAM E-MAIL FILTERING (1)

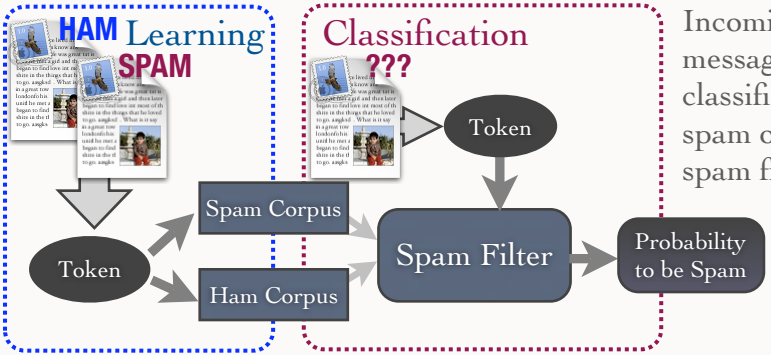
- Spam e-mail increases year by year.
  - About 94% of entire e-mail messages are Spam.

- Various spam filters have been developed.
  - Pattern matching based approach causes a rat race between spammers and developers.
  - Bayesian classification based approach has been recognized effective[1].

[1] P. Graham, Hackers and Painters: Big Ideas from the Computer Age, chapter 8, pp. 121-129, 2004.

PRELIMINARY:
SPAM E-MAIL FILTERING (2)

- All e-mail messages can be classified into
  - Spam: undesired e-mail
  - Ham: desired e-mail
- Tokenize and learn both spam and ham e-mail messages as text data and construct corpuses.



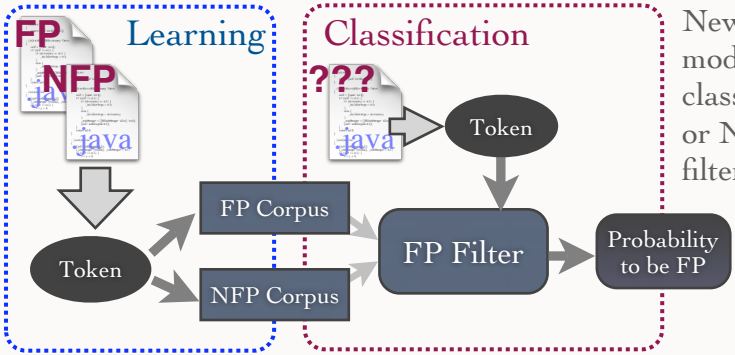
Incoming e-mail messages are classified into spam or ham by spam filter.

OVERVIEW

- Preliminary
- **Fault-Prone Filtering**
- Experiments
  - 10-fold cross validation
  - Results
- Conclusions

FAULT-PRONE FILTERING

- All software modules can be classified into
  - bug-detected (fault-prone: FP)
  - not-bug-detected (not-fault-prone: NFP)
- Tokenize and learn both FP and NFP modules as text data and construct corpuses



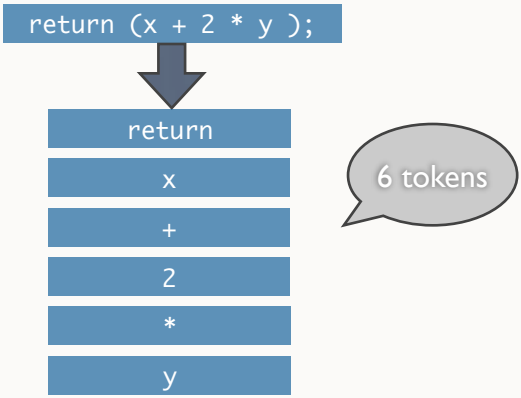
Newly developed modules are classified into FP or NFP by the FP filter.

FAULT-PRONE FILTERING:
SPAM FILTER: CRM114

- Spam filter: CRM114 (http://crm114.sourceforge.net/)
  - Generic text discriminator for various purpose
  - Generation of tokens
    - A token is a **combination of words**
    - 3 tokenization approaches: complex, simple, trivial
  - Classification techniques
    - Bayesian and Hyper-space distance
  - **4 classifiers** with different tokenization and classification
    - SBPH: Complex tokenization, Bayesian
    - OSB: Simple tokenization, Bayesian
    - BAYES: Trivial tokenization, Bayesian
    - HYPER: Simple tokenization, Hyper-space distance

FAULT-PRONE FILTERING:
TOKENIZATION: TRIVIAL

- Trivial tokenization
  - A token is generated from single word.
  - Conventional approach.



FAULT-PRONE FILTERING:
TOKENIZATION: COMPLEX

- Complex tokenization
  - Tokens are generated from combinations of at most 5 words

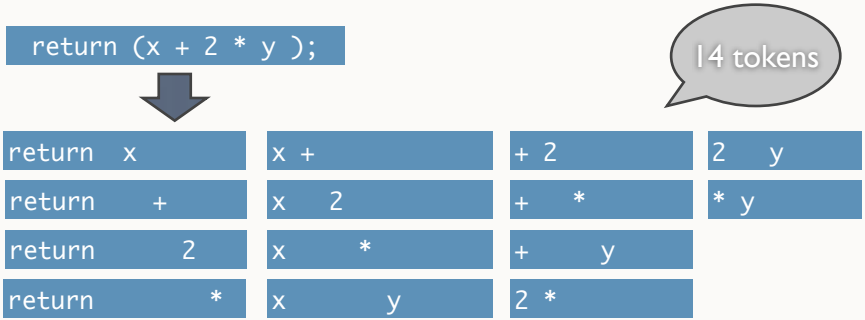
```
return (x + 2 * y);
```

41 tokens

return x	return 2 *	x y	x 2 * y	+ 2 * y
return +	return x + 2	x + 2	x + 2 y	2 *
return 2	return x + *	x + *	x + 2 * y	2 y
return *	return x 2 *	x + y	+ 2	2 * y
return x +	return + 2 *	x 2 *	+ *	* y
return x 2	return x + 2 *	x 2 y	+ y	
return x *	x +	x * y	+ 2 *	
return + 2	x 2	x + 2 *	+ * y	
return + *	x *	x + * y	+ 2 y	

FAULT-PRONE FILTERING:
TOKENIZATION: SIMPLE

- Simple tokenization
  - Tokens consisted of 2 words are selected from combinations of 5 words.



FAULT-PRONE FILTERING: CLASSIFICATION: BAYESIAN

- Calculate probability that a new module(T_{new}) is in FP corpus by Bayesian manner

FP Corpus (T_{FP})

x ++	* fact
x x	fact ++
* ++	fact x
* x	++ x

NFP Corpus (T_{NFP})

x --	* fact
x x	fact --
* --	fact x
* x	-- x

Tokens (T_{new})

x ++	+ sigma
x x	sigma ++
+ ++	sigma x
+ x	++ x

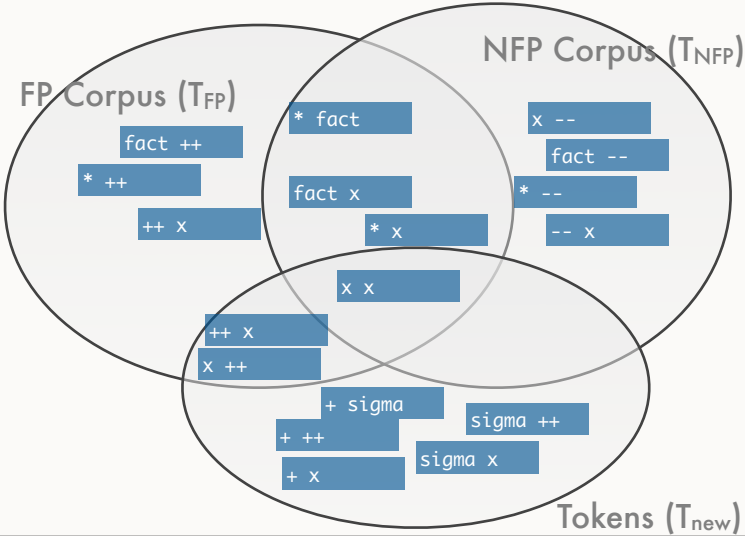
$$P(T_{FP}|T_{new}) = \frac{P(T_{new}|T_{FP})P(T_{FP})}{P(T_{new}|T_{FP})P(T_{FP}) + P(T_{new}|T_{NFP})P(T_{NFP})}$$

$$= \frac{\frac{3}{8} \times \frac{1}{2}}{\frac{3}{8} \times \frac{1}{2} + \frac{1}{8} \times \frac{1}{2}}$$

$$= 0.75$$

FAULT-PRONE FILTERING:
CLASSIFICATION: DISTANCE

- Calculate distances to both tokens, and choose nearest one

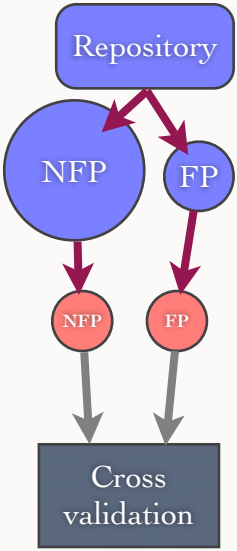


OVERVIEW

- Preliminary
- Fault-Prone Filtering
- **Experiments**
  - **10-fold cross validation**
  - **Results**
- Conclusions

EXPERIMENT: PROCEDURE

- Collect FP and NFP modules from target project.
- Sample randomly FP and NFP modules for 10-fold cross validation from all collected FP and NFP modules.
  - To adjust balance of data between FP and NFP.
- Apply 4 classifiers to sampled modules using 10-fold cross validation.
  - To see which classifier is appropriate for Fault-Prone Filtering.



EXPERIMENT: COLLECTING FP & NFP MODULES

- Track FP modules from CVS log based on an algorithm by Sliwerski, et. al[2].
- [2] J. Sliwerski, et. al., When do changes induce fixes? (on fridays.). In Proc. of MSR2005, pp. 24-28, 2005.
  - Search terms such as
    - “issue”, “problem”, or “#”, and bug id (numbers)
    - “fixed”, “resolved”, or “removed”
    - i. e. “Issue #100 is fixed.”

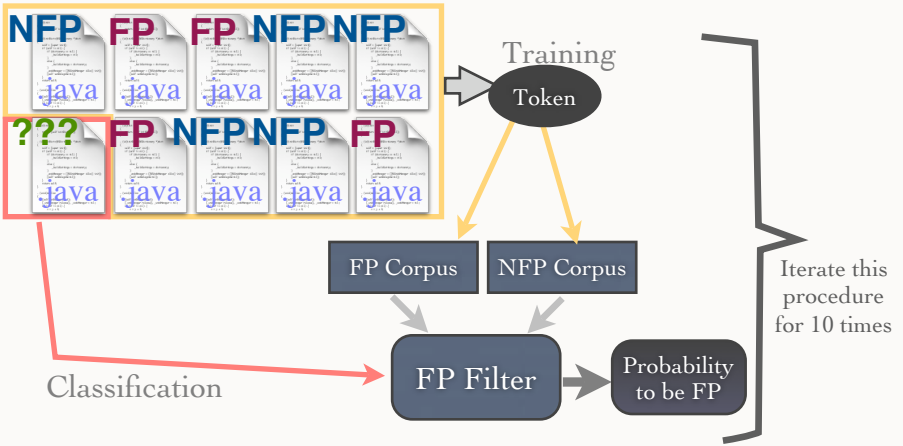
    from CVS log, then identify a revision the bug is removed.
  - Get difference from the previous revision and identify modified modules.
  - Track back repository and identify modules that have not been modified since the bug is reported.
    - They are FP modules.

EXPERIMENT: MODULE COLLECTION

- Extracted bugs from Bugzilla database of Eclipse
  - Conditions:
    - Type of faults: Bugs
    - Status of faults: Resolved, Verified, or Closed
    - Resolution of faults: Fixed
    - Severity: Blocker, Critical, Major, or Normal
  - Total # of faults: *40,627*
- Result of collection
  - # of faults found in CVS log: 21,761 (52% of total)
    - # of fault-prone(FP) modules: 65,782
    - # of not-fault-prone(NFP) modules: 1,113,063
  - For experiments, we randomly chose about 20,000 modules from both FP and NFP modules.

EXPERIMENT:
10-FOLD CROSS VALIDATION

- Divide all modules into 10 subsets randomly
- Each subset classified once.



EXPERIMENT:
EVALUATION MEASUREMENTS

- Accuracy
  - Overall accuracy of prediction
  - (N1+N4) / (N1+N2+N3+N4)
- Recall
  - How much actual FP modules are predicted as FP.
  - N4 / (N3+N4)
- Precision
  - How much predicted FP modules include actual FP modules
  - N4 / (N2+N4)
- Type-I error: N2
- Type-II error: N3 (Should be avoided)

Result of prediction		Predicted	
		NFP	FP
Actual	NFP	N1	N2
	FP	N3	N4

EXPERIMENT:
RESULT OF CROSS VALIDATION

SBPH		Predicted	
		NFP	FP
Actual	NFP	11,073	8,269
	FP	2,979	16,236

Precision: 0.663 Recall: **0.844**
Accuracy: **0.708**

OSB		Predicted	
		NFP	FP
Actual	NFP	12,249	7,093
	FP	2,972	16,243

Precision: 0.696 Recall: **0.845**
Accuracy: **0.739**

BAYES		Predicted	
		NFP	FP
Actual	NFP	10,514	8,828
	FP	1,955	17,260

Precision: 0.662 Recall: **0.898**
Accuracy: **0.720**

HYPER		Predicted	
		NFP	FP
Actual	NFP	14,995	4,347
	FP	6,354	12,861

Precision: **0.747** Recall: 0.669
Accuracy: **0.722**

EXPERIMENT: DISCUSSION

- OSB classifier is appropriate for fault-prone filtering.
  - Both high accuracy and high recall
- Others are also good.
  - Bayesian approaches tend to have high recall.
  - Distance based approach has high precision.
- High recall implies high coverage of faults.
- High precision implies high cost-effectiveness of testing.
  - Balance of recall and precision is required.
  - Changing threshold of determining FP and NFP enables to control cost-effectiveness of testing.

OVERVIEW

- Preliminary
- Fault-Prone Filtering
- Experiments
  - 10-fold cross validation
  - Results
- **Conclusions**

THREATS TO VALIDITY

- Construction validity
  - Collection of fault-prone modules from OSS projects.
  - We could not cover all faults in bugzilla database.
- Internal validity
  - 10-fold cross validation cannot deal with important information on fault-prone detection:
    - Order of creation or modification of modules
    - Application to time series data is effective
- External validity
  - Generalizability of the results

CONCLUSIONS

- Summary
  - We proposed the new approach to detect fault prone modules
    using spam filter.
  - The case study showed that our approach can predict fault prone modules with high accuracy.
- Future works
  - Using differences between revisions as an input of Fault-prone filtering
    - Seems more reasonable...
  - Application to industrial data

THE END

FAULT-PRONE FILTERING:

DETECTION OF FAULT-PRONE MODULES USING SPAM FILTERING TECHNIQUE

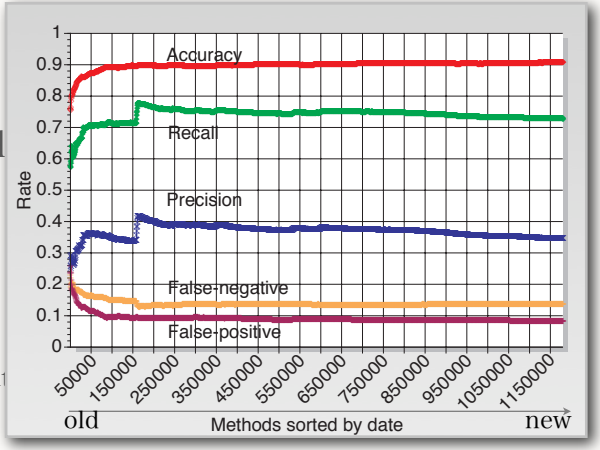
Thank you!

Any questions?

CONTACT: o-mizuno@ist.osaka-u.ac.jp

RESULT OF EXPERIMENT (TRANSITION OF RATES)

- All extracted modules are sorted by date, and applied FP filter one by one from the oldest one.
- Observation
  - The prediction result become stable after 50,000 modules classification.



EXPERIMENT:
CHANGING THRESHOLD

- Threshold of probability (t)
  - A threshold to determine FP or NFP from given probability. Usually set as 0.5.

- Changing threshold contributes accuracy of the model.

OSB t=0.8		Predicted	
		NFP	FP
Actual	NFP	14,436	4,906
	FP	4,802	14,413

Precision: 0.746
Recall: 0.750
Accuracy: 0.748

OSB t=0.5		Predicted	
		NFP	FP
Actual	NFP	12,249	7,093
	FP	2,972	16,243

RELATED WORKS

- Much research has been done so far.
  - Logistic regression
  - CART
  - Bayesian classification
  - and more.
- Most of them use software metrics
  - McCabe, Halstead, Object-oriented, and so on.
- Intuitively speaking, our approach uses a new metric, “frequency of tokens”.

EXPERIMENT:
CHANGING THRESHOLD

HYPER		Predicted	
		NFP	FP
Actual	NFP	732	297
	FP	297	665

Precision: **0.691**
Recall: **0.691**
Accuracy: **0.702**

EXPERIMENT: TARGET PROJECT

- Target: argoUML project
  - Written in Java
    - "Methods" in Java classes are considered as modules
  - Large CVS repository (about 900MB)
  - Faults are recorded precisely

EXPERIMENT: COLLECTING FP & NFP MODULES

- Track FP modules from CVS log based on an algorithm by Sliwerski, et. al[2].
- [2] J. Sliwerski, et. al., When do changes induce fixes? (on fridays.). In Proc. of MSR2005, pp. 24-28, 2005.

  - Search terms such as “issue”, “problem”, “#”, and bug id as well as “fixed”, “resolved”, or “removed” from CVS log, then identify a revision the bug is removed.
  - Get difference from the previous revision and identify modified modules.
  - Track back repository and identify modules that have not been modified since the bug is reported.
    - They are FP modules.

EXPERIMENT: MODULE COLLECTION

- Extracted bugs from bugzilla database of argoUML
  - Conditions:
    - Type of faults: Bugs
    - Status of faults: Resolved, Verified, or Closed
    - Resolution of faults: Fixed
    - Severity: Blocker, Critical, Major, or Normal
  - Total # of faults: 1,058
- Result of collection
  - # of faults found in CVS log: 396 (37% of total)
    - # of fault-prone(FP) modules: 962
    - # of not-fault-prone(NFP) modules: 331,488
      - For experiments, we randomly chose 1,029 modules from 331,488 modules.

EXPERIMENT:
EVALUATION MEASUREMENTS

Result of prediction		Predicted	
		NFP	FP
Actual	NFP	N1	N2
	FP	N3	N4

- Accuracy
  - Overall accuracy of prediction
  - (N1+N4) / (N1+N2+N3+N4)
- Recall
  - How much actual FP modules are predicted as FP.
  - N3 / (N3+N4)
- Precision
  - How much predicted FP modules include actual FP modules
  - N2 / (N2+N4)

EXPERIMENT:
RESULT OF CROSS VALIDATION

SBPH		Predicted	
		NFP	FP
Actual	NFP	601	428
	FP	189	773

Precision: 0.643 Recall: **0.803**
Accuracy: 0.690

OSB		Predicted	
		NFP	FP
Actual	NFP	546	483
	FP	156	806

Precision: 0.625 Recall: **0.838**
Accuracy: 0.674

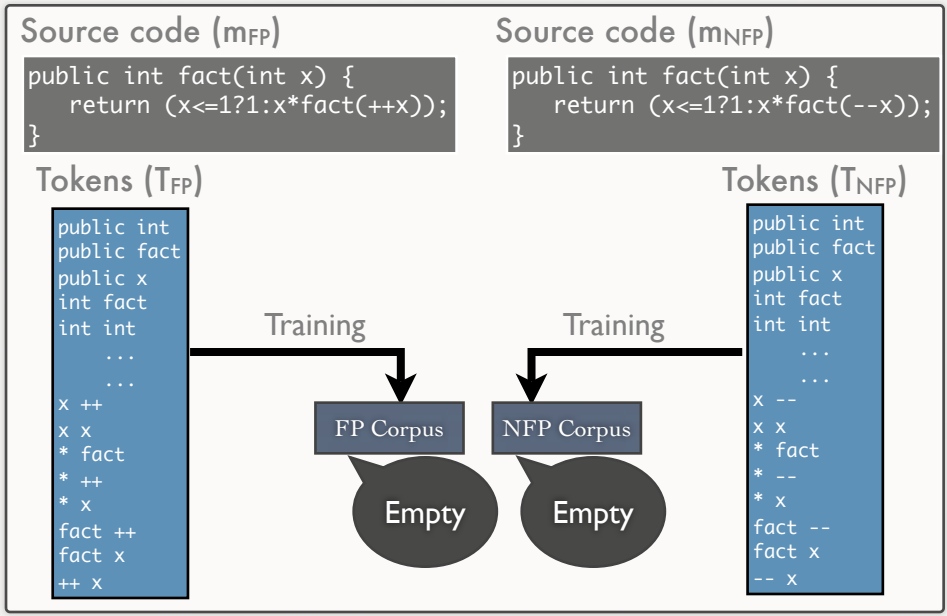
BAYES		Predicted	
		NFP	FP
Actual	NFP	236	793
	FP	29	933

Precision: 0.540 Recall: **0.970**
Accuracy: 0.587

HYPER		Predicted	
		NFP	FP
Actual	NFP	786	243
	FP	352	610

Precision: **0.715** Recall: 0.634
Accuracy: **0.701**

FAULT-PRONE FILTERING:
FAULT-PRONE TRAINING



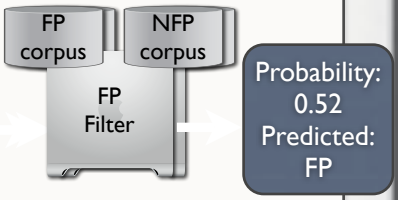
FAULT-PRONE FILTERING:
FAULT-PRONE CALCULATION

Source code (m_{new})

```
public int sigma(int x) {
   return (x<=0?0:x+sigma(++x));
}
```

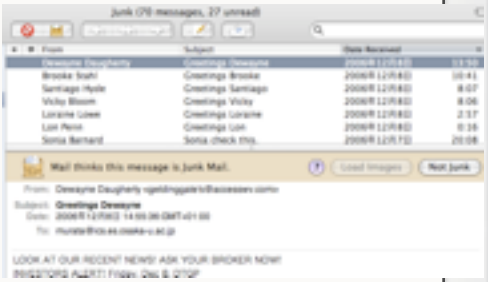
Tokens (T_{FP})	Tokens (T_{new})	Tokens (T_{NFP})
public int	public int	public int
public fact	public sigma	public fact
public x	public x	public x
int fact	int sigma	int fact
int int	int int	int int
...	...	...
...	...	...
x ++	x ++	x --
x x	x x	Prediction
* fact	+ sigma	* fact
* ++	+ ++	* --
* x	+ x	* x
fact ++	sigma ++	fact --
fact x	sigma x	fact x
++ x	++ x	-- x

m_{new} is predicted as FP because T_{FP} has more similarity than T_{NFP}.



TRAINING ONLY ERRORS PROCEDURE

- In Spam filtering:
  - Apply e-mail messages to spam filter in order of arrival.
  - Only misclassified e-mail messages are trained in corpuses.
  - You may do this procedure in daily e-mail assorting.



- In Fault-prone filtering:
  - Apply software modules to fault-prone filter in order of construction and modification.
  - Only misclassified modules are trained in corpuses.

RESULT OF CROSS VALIDATION

- Result for Eclipse BIRT plugin
- 10-fold cross validation

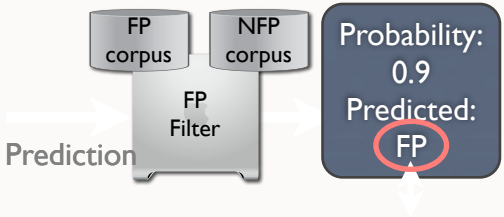
Cross Validation OSB		Predicted	
		NFP	FP
Actual	NFP	70,369	16,011
	FP	2,039	7,501

- Precision: 0.319
- Recall: 0.786
- Accuracy: 0.811

- Recall is important for quality assurance.
- Precision implies the cost for finding FP modules.

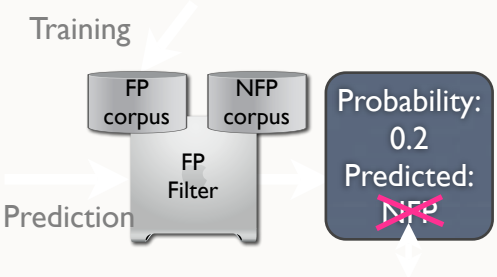
Recall is rather high, and precision is rather low.

TRAINING ONLY ERRORS PROCEDURE



FP

TRAINING ONLY ERRORS PROCEDURE

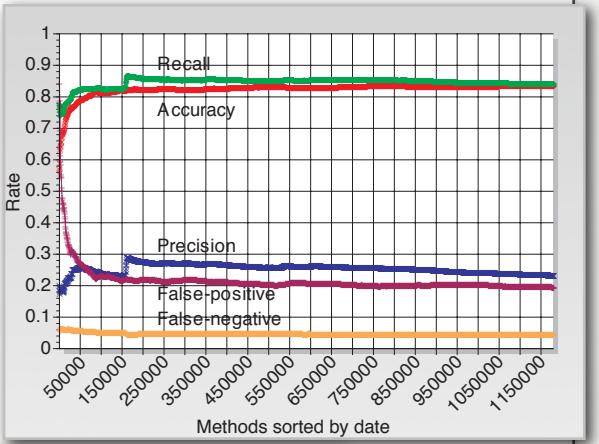


PROCEDURE OF EXPERIMENT

- Two experiments with different thresholds of probability (t_{FP}) to determine FP and NFP.
  - Changing t_{FP} may achieve higher recall
- Experiment 1:
  - TOE with OSB classifier, t_{FP}=0.5
- Experiment 2:
  - TOE with OSB classifier, t_{FP}=0.25
  - Predict more modules as FP than Experiment 1

RESULT OF EXPERIMENT (OSB, T_{FP} = 0.25)

- Comparison with threshold = 0.50
  - Precision becomes lower.
    - Only 1/4 of FP predicted modules hits actual faulty modules.
  - Recall becomes much higher.
    - 83% of actual faulty modules can be detected.



TOE - final OSB		Predicted	
		NFP	FP
Actual	NFP	930,218	182,845
	FP	10,592	55,190

- Precision: 0.232
- Recall: 0.839
- Accuracy: 0.835