Automated Bug Triaging to the Developers
A Machine Learning Approach

Saagarikha S - 312211104086

Susindaran E - 312211104111

Venugopal C G - 312211104121

Supervisor: Milton R S, Professor

SSN College Of Engineering, Chennai

March 23, 2015

Problem Statement

- Input: A bug report in natural language text submitted by the reporter briefing the problem.

Problem Statement

- Input: A bug report in natural language text submitted by the reporter briefing the problem.
- Output: The component in which the bug may potentially be, and the developer or list of developers to whom it can be be assigned to.

Problem Statement

- Input: A bug report in natural language text submitted by the reporter briefing the problem.
- Output: The component in which the bug may potentially be, and the developer or list of developers to whom it can be be assigned to.
- The probability of the bug getting reassigned must be minimized.

Problem Statement

- Input: A bug report in natural language text submitted by the reporter briefing the problem.
- Output: The component in which the bug may potentially be, and the developer or list of developers to whom it can be be assigned to.
- The probability of the bug getting reassigned must be minimized.
- After fixing the bug, the bug report is annotated/labeled with the developer and the component.

Problem Statement

- Input: A bug report in natural language text submitted by the reporter briefing the problem.
- Output: The component in which the bug may potentially be, and the developer or list of developers to whom it can be be assigned to.
- The probability of the bug getting reassigned must be minimized.
- After fixing the bug, the bug report is annotated/labeled with the developer and the component.
- A dependency structure is formed over time for supervised learning from the fixed bugs.

Literature Survey

- The existing system uses a classifier and tossing graphs to assign bug automatically to developers.

Literature Survey

- The existing system uses a classifier and tossing graphs to assign bug automatically to developers.
- Initially a training data set of fixed bugs that contains information regarding the developers to whom it was assigned and the reassignment to other developers.

Literature Survey

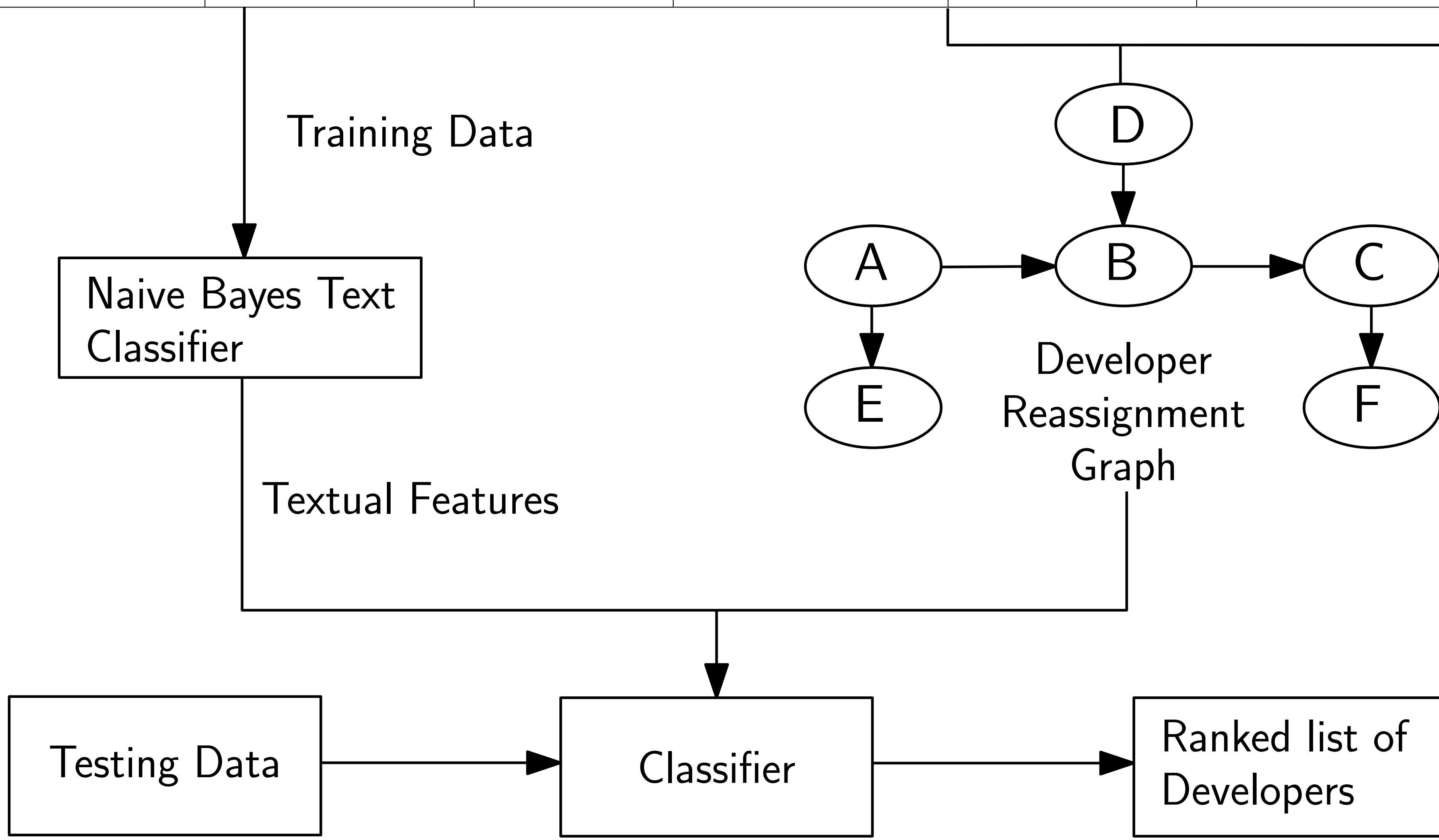
- The existing system uses a classifier and tossing graphs to assign bug automatically to developers.
- Initially a training data set of fixed bugs that contains information regarding the developers to whom it was assigned and the reassignment to other developers.
- The system uses Naive Bayes classifier to classify the new bugs reported, and to calculate the probability of it belonging to a class.

Literature Survey

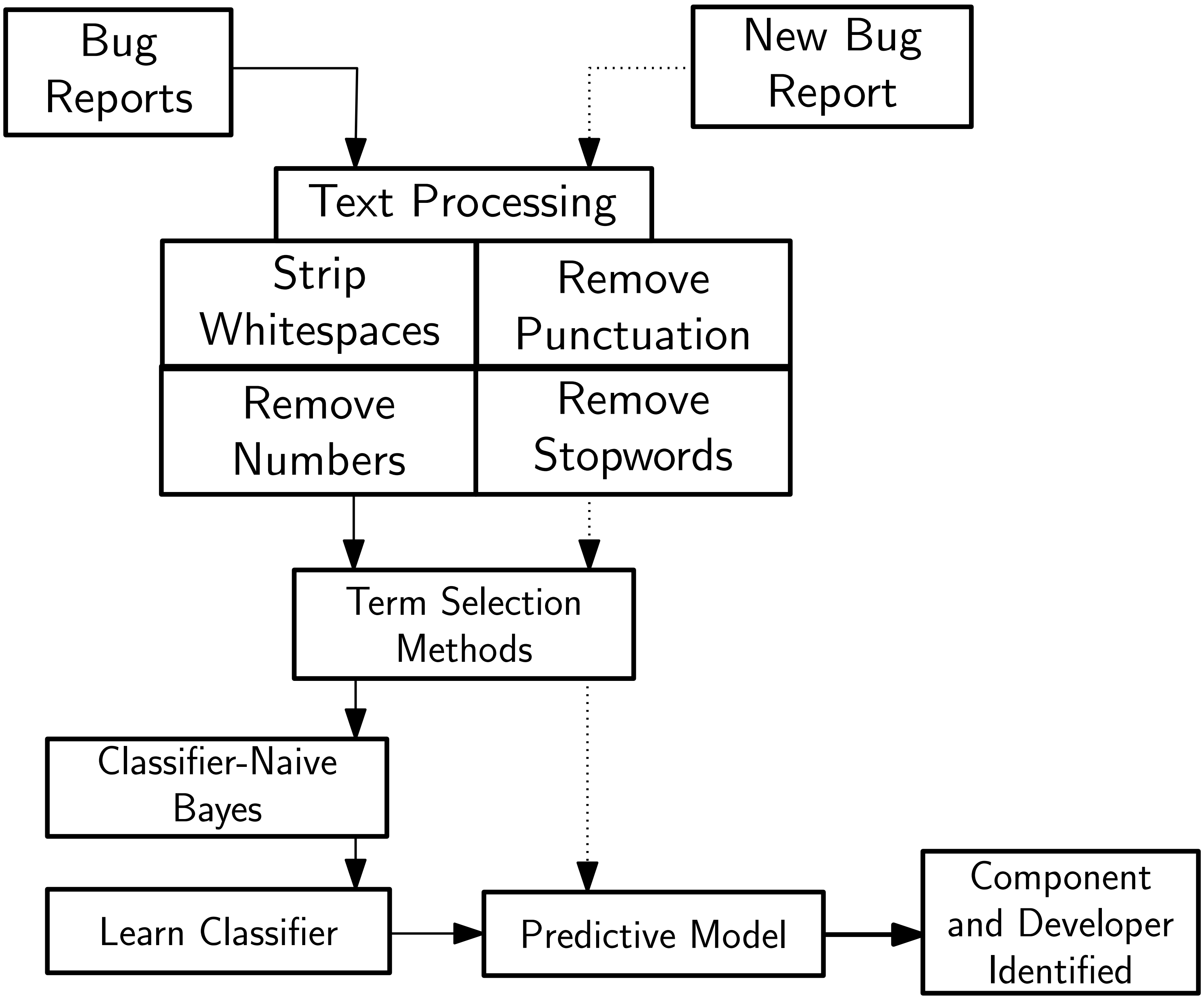
- The existing system uses a classifier and tossing graphs to assign bug automatically to developers.
- Initially a training data set of fixed bugs that contains information regarding the developers to whom it was assigned and the reassignment to other developers.
- The system uses Naive Bayes classifier to classify the new bugs reported, and to calculate the probability of it belonging to a class.
- Graphs are used to predict the probability of reassignment of a bug to another developer.

Architecture

Report ID	Description	Product	Component	Initial Dev	Reassigned
12784	. . .	JDT	Core	A	E



Modules


Bug Reports
New Bug Report
Text Processing
Strip Whitespaces
Remove Punctuation
Remove Numbers
Remove Stopwords
Term Selection Methods
Classifier-Naive Bayes
Learn Classifier
Predictive Model
Component and Developer Identified

Multinomial Naive Bayes Classifier

Feature Vector

$$W = (w_1,\ w_2,\ w_3,\ \ldots,\ w_n)$$

Multinomial Naive Bayes Classifier

Feature Vector

$$W = (w_1,\ w_2,\ w_3,\ ...,\ w_n)$$

$$\log\ p(D_k|W) = \log\ p(D_k) + \sum_{i=1}^{N} w_i\ .\ \log\ p(w_i|D_k) \tag{1}$$

Multinomial Naive Bayes Classifier

Feature Vector

$$W = (w_1,\ w_2,\ w_3,\ \ldots,\ w_n)$$

$$\log\ p(D_k|W) = \log\ p(D_k) + \sum_{i=1}^{N} w_i\ .\ \log\ p(w_i|D_k) \tag{1}$$

$\frac{|N_c|}{|N|}$ Prior

Multinomial Naive Bayes Classifier

Feature Vector

$$W = (w_1,\ w_2,\ w_3,\ ...,\ w_n)$$

$$\log\ p(D_k|W) = \log\ p(D_k) + \sum_{i=1}^{N} w_i\ .\ \log\ p(w_i|D_k) \quad (1)$$

$\frac{|N_c|}{|N|}$ Prior

Conditional Probability $\frac{\text{count}(w, D_k)+1}{\text{count}(D_k)+|V|}$

Multinomial Naive Bayes Algorithm

Algorithm 1: TRAIN MULTINOMIALNB - Text classification

Input: R: Training Corpus(List of bug reports)
D: List of Developers
Output: V-vocabulary,prior and condprob
$V \leftarrow$ extract Vocabulary
$N \leftarrow$ Count Bug Reports
foreach *Developer d in D* **do**
  $N_c \leftarrow$ Count bug reports of developer d from R
  prior(d) $\leftarrow \frac{|N_c|}{|N|}$
  $words_d \leftarrow$
  Collect all words from all bug reports of developer d
  foreach *word w in V* **do**
    $T_c \leftarrow$ Count occurences of w in $words_d$
    $T'_c \leftarrow$ Count words(d)
    condprob(w|d) $\leftarrow \frac{|T_c+1|}{T'_c+1}$
  end
end
return $V,\ prior\ and\ condprob$

Multinomial Naive Bayes Algorithm

Algorithm 2: APPLY MULTINOMIALNB - Text classification

```
Input: D: List of Developers
       V: Vocabulary
       prior
       condprob
       R: Bug Report
Output: d: The Developer with the highest probability to whom
           the bug will be assigned
W ← Extract words from Report R
foreach developer d in D do
    P(d|R) ← log(prior(d))
    foreach word w in W do
        freq = count(w,R)
        P(d|R) += freq * log(condprob(w|d))
    end
end
return argmax_d P(d|R)
```

TF-IDF Weight Calculation

$$\mathsf{tf(t,doc)} = 0.5 + \frac{0.5 \times f(t,doc)}{\max\{f(w,doc) : w \in doc\}}$$

TF-IDF Weight Calculation

$$\text{tf(t,doc)} = 0.5 + \frac{0.5 \times f(t,doc)}{\max\{f(w,doc) : w \in doc\}}$$

$$\text{idf(t, DOC)} = \log \frac{N}{1 + |\{doc \in DOC : t \in doc\}|}$$

TF-IDF Weight Calculation

$$\text{tf(t,doc)} = 0.5 + \frac{0.5 \times f(t,doc)}{\max\{f(w,doc) : w \in doc\}}$$

$$\text{idf(t, DOC)} = \log \frac{N}{1 + |\{doc \in DOC : t \in doc\}|}$$

$$\text{tfidf(t,doc,DOC)} = tf(t, doc) \times idf(t, DOC)$$

Support Vector Machine

- An Binary SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible.

Support Vector Machine

- An Binary SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible.

$$D = \{(x_i, y_i) \mid x_i \in R^p,\, y_i \in \{-1, 1\}\}_{i=1}^{n}$$

Support Vector Machine

- An Binary SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible.

$$D = \{(x_i, y_i) \mid x_i \in R^p,\, y_i \in \{-1, 1\}\}_{i=1}^{n}$$

- Multiclass SVM is implemented by reducing the single multiclass problem into multiple binary classification problems.(one-versus-all)

Folding

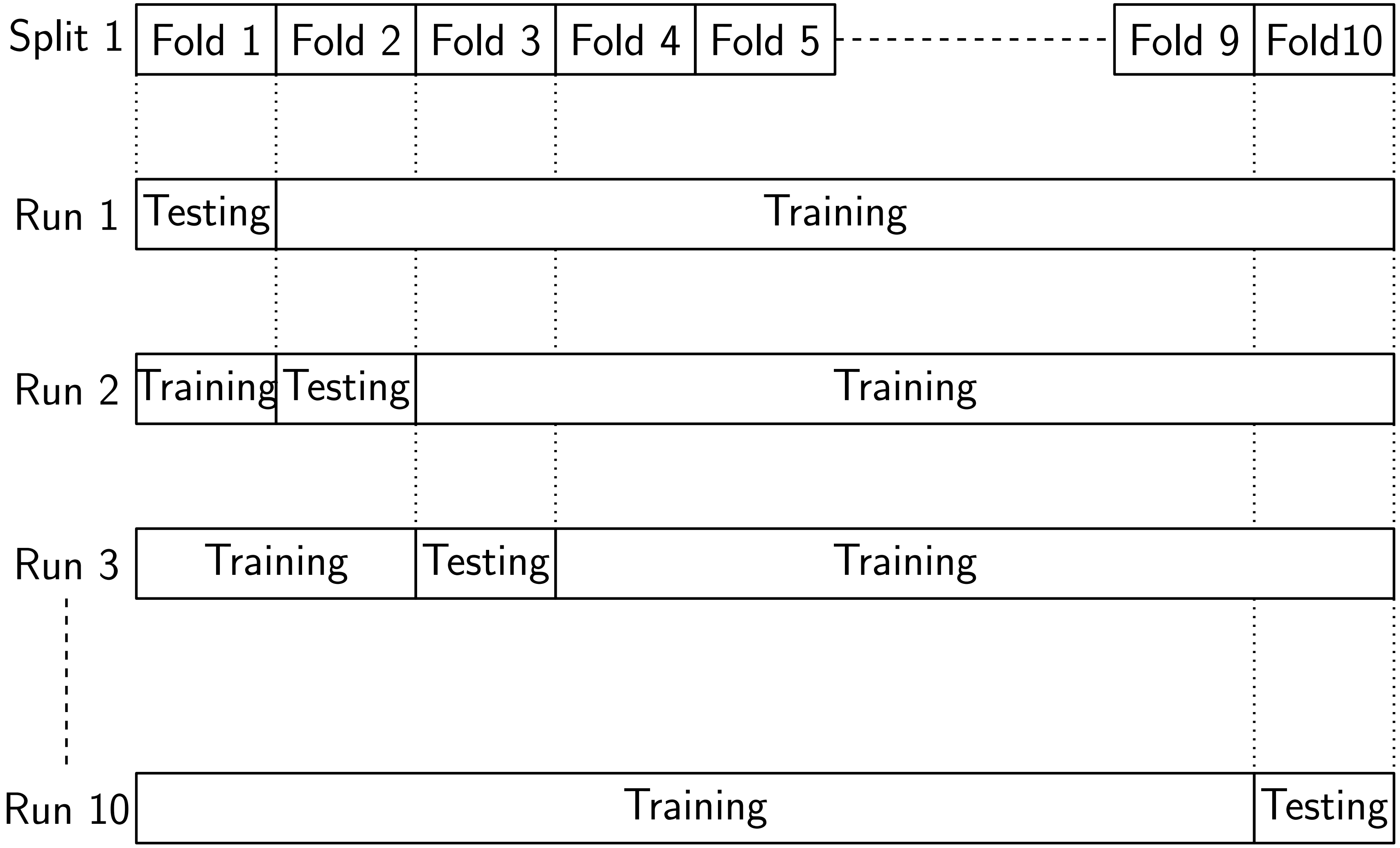
- In folding based training and validation approach, also known as cross validation, the algorithm first collects all bug reports to be used for TDS (Training Data Set) sorts them in chronological order(based on the update time of the bug) and then divides them into n folds.

Folding

- In folding based training and validation approach, also known as cross validation, the algorithm first collects all bug reports to be used for TDS (Training Data Set) sorts them in chronological order(based on the update time of the bug) and then divides them into *n* folds.

- In the n^{th} run, fold *n* is used as testing data and the remaining folds are used for training the classifier.

Folding



Split 1
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Fold 9
Fold10
Run 1
Testing
Training
Run 2
Training
Testing
Training
Run 3
Training
Testing
Training
Run 10
Training
Testing

Tossing Graph

$$P(D- > D_j) = \frac{\#(D- > D_j)}{\sum_{i=1}^{n}(D- > D_i)}$$

Tossing Graph

$$P(D- > D_j) = \frac{\#(D- > D_j)}{\sum_{i=1}^{n} (D- > D_i)}$$

- Tossing graphs are weighted directed graphs such that each node represents a developer, and each directed edge from D 1 to D 2 represents the fact that a bug assigned to developer D 1 was tossed and eventually fixed by developer D 2 .

Tossing Graph

$$P(D- > D_j) = \frac{\#(D- > D_j)}{\sum_{i=1}^{n} (D- > D_i)}$$

- Tossing graphs are weighted directed graphs such that each node represents a developer, and each directed edge from D 1 to D 2 represents the fact that a bug assigned to developer D 1 was tossed and eventually fixed by developer D 2 .
- The weight of an edge between two developers is the probability of a toss between them, based on bug tossing history.

Tossing Graph [1]

Tossing paths
A→B→C→D
A→E→D→C
A→B→E→D
C→E→A→D
B→E→D→F

Developer who tossed the bug	Total tosses	Developer who fixed the bug					
		C		D		F	
		#	pr	#	pr	#	pr
A	4	1	0.25	3	0.75	0	0.00
B	3	0	0.5	2	0.67	1	0.33
C	2			2	1.00	0	0.00
D	2	1	0.50			1	0.50
E	4	1	0.25	2	0.50	1	0.25

Tossing Graph [2]

- In the above table we provide sample tossing paths and show how toss probabilities are computed.

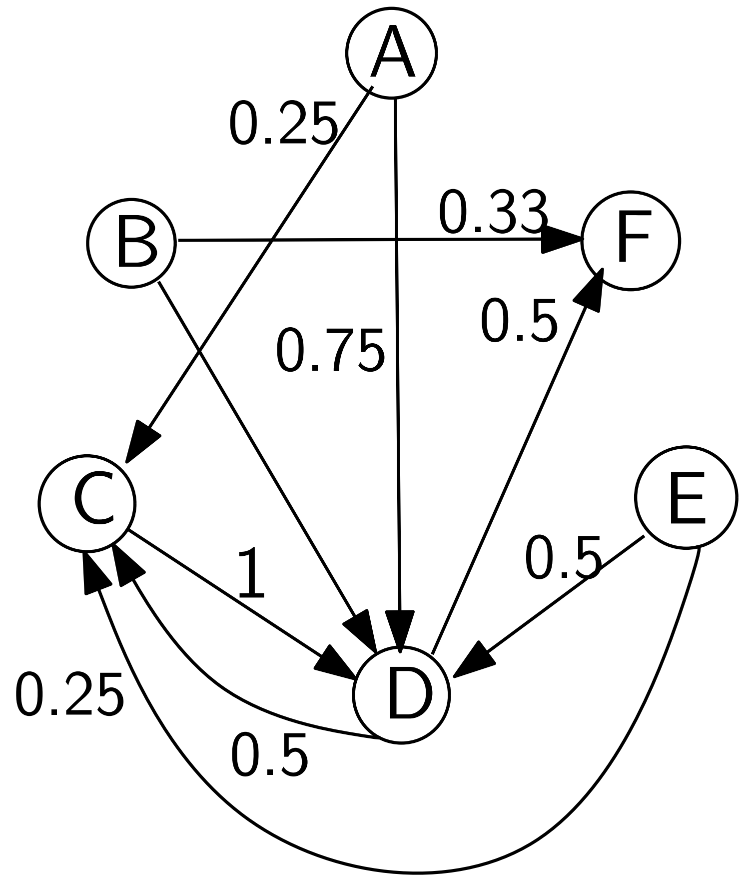
Tossing Graph [2]

- In the above table we provide sample tossing paths and show how toss probabilities are computed.
- For example, developer A has tossed four bugs in all, three that were fixed by D and one that was fixed by C, hence $P\ r(A \to D) = 0.75$, $P\ r(A \to C) = 0.25$, and $P\ r(A \to F\ ) = 0$.

Tossing Graph [2]

- In the above table we provide sample tossing paths and show how toss probabilities are computed.
- For example, developer A has tossed four bugs in all, three that were fixed by D and one that was fixed by C, hence P r(A $\rightarrow$ D) $=$ 0.75, P r(A $\rightarrow$ C) $=$ 0.25, and P r(A $\rightarrow$ F) $=$ 0.
- The developers who did not toss any bug (e.g., F) do not appear in the first column, and developers who did not fix any bugs (e.g., A) do not have a probability column

Tosssing graph built using tossing path



Output

```
Terminal
File  Edit  View  Search  Terminal  Help
sags@sags-HP-630-Notebook-PC ~/Project/Review1 $ python review2.py
Training dataset : 53379
Testing Dataset : 13344
The accuracy for MultinomialNB is :  0.770353717026
[0.78239922074029677, 0.7633673010639892, 0.78629551925670615, 0.778597122302158
27, 0.7732014388489209, 0.76540767386091124, 0.77919664268585132, 0.785941247002
39809, 0.76211031175059951, 0.78219424460431652]
The accuracy for Kfold MultinomialNB Classifier is :  0.786295519257
The accuracy using pipeline for LinearSVM is :  0.800554556355
The predicted developer is : platform-debug-inbox@eclipse.org
Predicted Developer ID is : 9
Tossing possibilities in the decreasing order of probability ---
9 -> 14 Probability : 0.291666666667
9 -> 7 Probability : 0.25
9 -> 4 Probability : 0.125
9 -> 18 Probability : 0.125
9 -> 42 Probability : 0.125
9 -> 9 Probability : 0.0416666666667
9 -> 24 Probability : 0.0208333333333
9 -> 178 Probability : 0.0208333333333
sags@sags-HP-630-Notebook-PC ~/Project/Review1 $
```

Results

- An accuracy of 78.63% is achieved using Multinomial Naive Bayes Classifier for a dataset containing approximately 68000 tagged bug reports.

Results

- An accuracy of 78.63% is achieved using Multinomial Naive Bayes Classifier for a dataset containing approximately 68000 tagged bug reports.
- The same feature vectors are used to train the Multiclass SVM Classifier and an accuracy of 80.05% is achieved.

Results

- An accuracy of 78.63% is achieved using Multinomial Naive Bayes Classifier for a dataset containing approximately 68000 tagged bug reports.
- The same feature vectors are used to train the Multiclass SVM Classifier and an accuracy of 80.05% is achieved.
- The learning time of the classifier is drastically reduced by using sparse representation of the word counts.

Results

- An accuracy of 78.63% is achieved using Multinomial Naive Bayes Classifier for a dataset containing approximately 68000 tagged bug reports.
- The same feature vectors are used to train the Multiclass SVM Classifier and an accuracy of 80.05% is achieved.
- The learning time of the classifier is drastically reduced by using sparse representation of the word counts.
- The efficiency of the classifier was further improved by using tf-idf weights and extracting only important features and using them.

References

1 D. Cubranic, G. C. Murphy, Automatic bug triage using text categorization, in: SEKE, 2004.
2 J. Anvik, L. Hiew, G. C. Murphy, Who should fix this bug?, in: ICSE, 361370, 2006.
3 Z. Lin, F. Shu, Y. Yang, C. Hu, Q. Wang, An empirical study on bug assignment automation using Chinese bug data, in: ESEM, 2009.
4 G. Jeong, S. Kim, T. Zimmermann, Improving Bug Triage with Bug Tossing Graphs, in: FSE, 2009.
5 Pamela Bhattacharyaa, Iulian Neamtiua, Automated, Highly-Accurate, Bug Assignment Using Machine Learning and Tossing Graphs, University of California, Riverside, 2012.
6 $https://github.com/ansymo/msr2013-bug\_dataset$.

Thank You.