Classification Algorithms for Detecting Duplicate Bug Reports in Large Open Source Repositories

Sarah E. Ritchey

Youngstown State University

April 28, 2014

Motivation

- Complex software can have more defects

Motivation

- Complex software can have more defects
- Bug repositories to keep track of reported defects

Motivation

- Complex software can have more defects
- Bug repositories to keep track of reported defects
- Multiple reports on same bug (duplicates)

Motivation

- Complex software can have more defects
- Bug repositories to keep track of reported defects
- Multiple reports on same bug (duplicates)
- Triage assigns bugs to developer or marks as duplicate

Motivation

- Complex software can have more defects
- Bug repositories to keep track of reported defects
- Multiple reports on same bug (duplicates)
- Triage assigns bugs to developer or marks as duplicate
- Tremendous amount of work for the triage

Motivation

- Complex software can have more defects
- Bug repositories to keep track of reported defects
- Multiple reports on same bug (duplicates)
- Triage assigns bugs to developer or marks as duplicate
- Tremendous amount of work for the triage
- Enormous waste of time and resources

Motivation

- Complex software can have more defects
- Bug repositories to keep track of reported defects
- Multiple reports on same bug (duplicates)
- Triage assigns bugs to developer or marks as duplicate
- Tremendous amount of work for the triage
- Enormous waste of time and resources
- Larger software projects over 100 bug reports in a single day

Motivation

- Complex software can have more defects
- Bug repositories to keep track of reported defects
- Multiple reports on same bug (duplicates)
- Triage assigns bugs to developer or marks as duplicate
- Tremendous amount of work for the triage
- Enormous waste of time and resources
- Larger software projects over 100 bug reports in a single day
- Up to 30% of all reports in their systems are duplicates

Motivation

- Complex software can have more defects
- Bug repositories to keep track of reported defects
- Multiple reports on same bug (duplicates)
- Triage assigns bugs to developer or marks as duplicate
- Tremendous amount of work for the triage
- Enormous waste of time and resources
- Larger software projects over 100 bug reports in a single day
- Up to 30% of all reports in their systems are duplicates

Essential to detect duplicates automatically to keep software development fiscally feasible!

Two Methods

Top *k*-list

- reduces the work required by triage
- still susceptible to human error

Two Methods

Top *k*-list

- reduces the work required by triage
- still susceptible to human error

Automatic classification

- Automatically determines status of report
- If labeled as a duplicate, requires no work by the triage
- Non duplicate report could potentially be mislabeled

Sample Duplicate Defect Report

bug_{id}	214068
$description$	Failed to preview Chart Viewer ...
$short_{desc}$	Failed to preview ...
$priority$	P3
$bug_{severity}$	critical
$product$	BIRT
$version$	2.3.0
$component$	Build
$creation_{ts}$	1/2/08 1:35
$delta_{ts}$	1/2/08 4:32
$resolution$	DUPLICATE
dup_{id}	214069
bug_{status}	CLOSED

Table: Sample Duplicate Defect Report

Generating Datasets

Table: Details of Bug Datasets

Dataset	From	To	$Bugs_i$	$Dupl_i$	$Bugs_f$	$Dupl_f$	$DuplPairs$
Eclipse	1/2008	12/2008	45,746	4,386	39,020	2,897	6,024
Open Office	1/2008	12/2010	31,333	4,549	23,108	2,861	6,945
Mozilla	1/2010	12/2010	78,236	10,777	65,941	6,534	16,631

Generating Datasets

Table: Details of Bug Datasets

Dataset	From	To	$Bugs_i$	$Dupl_i$	$Bugs_f$	$Dupl_f$	$DuplPairs$
Eclipse	1/2008	12/2008	45,746	4,386	39,020	2,897	6,024
Open Office	1/2008	12/2010	31,333	4,549	23,108	2,861	6,945
Mozilla	1/2010	12/2010	78,236	10,777	65,941	6,534	16,631

Reports were removed with:

- OPEN resolution reports
- DUPLICATE resolution bugs without masters in the dataset

Duplicate Groups and Pairs

- Groups of duplicate reports be greater than 2
- Group of duplicates needed to be calculated

Duplicate Groups and Pairs

- Groups of duplicate reports be greater than 2
- Group of duplicates needed to be calculated
- This was a tedious task!
- Only DUPLICATE resolution bugs contain information
- Impossible to tell if master report has a duplicate

Duplicate Groups and Pairs

- Groups of duplicate reports be greater than 2
- Group of duplicates needed to be calculated
- This was a tedious task!
- Only DUPLICATE resolution bugs contain information
- Impossible to tell if master report has a duplicate
- Long chains of duplicate bugs that branch in both directions
- Algorithm finds duplicate group and duplicate pairs.

Distinguishing Duplicate and Non-Duplicates

Goal of project is to correctly identify pairs of reports as duplicates or not.

Distinguishing Duplicate and Non-Duplicates

Goal of project is to correctly identify pairs of reports as duplicates or not.

- Additional feature, *decision*, added to each pair

Distinguishing Duplicate and Non-Duplicates

Goal of project is to correctly identify pairs of reports as duplicates or not.

- Additional feature, *decision*, added to each pair
- *decision*=1 for duplicate pairs
- *decision*=-1 for non-duplicate pairs

Distinguishing Duplicate and Non-Duplicates

Goal of project is to correctly identify pairs of reports as duplicates or not.

- Additional feature, *decision*, added to each pair
- *decision*=1 for duplicate pairs
- *decision*=-1 for non-duplicate pairs
- 4:1 ratio of non-duplicate pairs to duplicate pairs

Feature Generation

Assumption: duplicate reports will be similar

Feature Generation

Assumption: duplicate reports will be similar

- What does it mean to be similar?
- Generate 25 different similarity measures
- Each features represents coordinate of point in 25-space.

Feature Generation

Assumption: duplicate reports will be similar

- What does it mean to be similar?
- Generate 25 different similarity measures
- Each features represents coordinate of point in 25-space.
- First 18 measure textual similarity of $short_{desc}$ and *description*.
- Inspired by Saric et al. [5], are generated using TakeLab system

Feature Generation

Assumption: duplicate reports will be similar

- What does it mean to be similar?
- Generate 25 different similarity measures
- Each features represents coordinate of point in 25-space.
- First 18 measure textual similarity of $short_{desc}$ and *description*.
- Inspired by Saric et al. [5], are generated using TakeLab system
- Next 7 features measure categorial similarity and use other report fields

N-gram Overlap

An n-gram is a set of *n* consecutive words

N-gram Overlap

An n-gram is a set of *n* consecutive words

Definition

Let S_1 and S_2 be the sets of consecutive n-grams in the first and the second report, respectively. The ngram overlap is defined as follows:

$$ngo(S_1, S_2) = 2\left(\frac{|S_1|}{|S_1 \cap S_2|} + \frac{|S_2|}{|S_1 \cap S_2|}\right)^{-1} \tag{1}$$

The overlap is computed for unigrams, bigrams, and trigrams.

N-gram Word Overlap After Lemmatization

- Content words are nouns, verbs, adjectives, and adverbs.
- Function words (prepositions, code, conjunctions, articles) carry less semantic information than content words.
- Removing them may eliminate noise
- So n-grams(lemmas) are created using only content words.
- Calculated using eq. (1) for unigrams, bigrams, and trigrams.

WordNet Based Augmented Word Overlap

- High unigram overlap only if exactly the same words appears.
- WordNet is a hierarchical network of words linked by relations.
- WordNet to assign partial scores to words not in both.

Definition

We define the WordNet augmented coverage:

$$PWN(S_1, S_2) = \frac{1}{|S_2|} \sum_{w_1 \in S_1} sim(w_1, S_2)$$

where $sim(w, S)$ is minimum WordNet path length.

Definition

The WordNet-augmented word overlap feature is defined as a harmonic mean of $PWN(S_1, S_2)$ and $PWN(S_2, S_1)$.

Weighted Word Overlap and Normalized Differences

- Longer reports are more likely to be similar to other reports
- Frequently used words can also cloud results
- Several features measure the normalized differences
  - Weighted word overlap
  - Sentence length
  - Aggregate word information content
- This prevents longer reports and frequently used words from having a advantage over shorter reports.

Shallow Named Entity

- Proper nouns convey a huge amount of information.
- SNE measure the similarity of named entities
- Overlap of capitalized words (longer than one character)
- Overlap of stock index symbols (all capital letters)
- Classified into types
  - persons
  - organizations
  - locations
  - dates
  - rudimentary temporal expressions
- Binary overlap of each type are calculated for each pair
- The absence or presence of class in both is indicated

Numbers Overlap

Remove bias against reports containing different sets of numbers.

Definition

Let N_1 and N_2 be sets of number in two sentences: Define numbers overlap features to be

$$nof1(N_1, N_2) = \log(1 + |N_1| + |N_2|)$$

$$nof2(N_1, N_2) = \frac{2|N_1 \cap N_2|}{|N_1| + |N_2|}$$

and

$$nof3(N_1, N_2) = \begin{cases} 1, & \text{if } N_1 \subseteq N_2 \text{ or } N_2 \subseteq N_1 \\ 0, & \text{otherwise} \end{cases} .$$

Numbers differing in number of decimal places are “equal”

Categorical Features

Definition

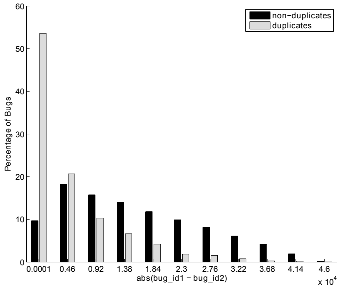
Let b_1 and b_2 be bug reports. Then $b_i.\mathit{property}$ is the field called property of b_i. Let f_n be the nth feature of the model defined by:

$$
\begin{aligned}
f_{19}(b_1, b_2) &= \begin{cases} 1, & \text{if } b_1.\mathit{product} = b_2.\mathit{product} \\ 0, & \text{otherwise} \end{cases} \\
f_{20}(b_1, b_2) &= \begin{cases} 1, & \text{if } b_1.\mathit{component} = b_2.\mathit{component} \\ 0, & \text{otherwise} \end{cases} \\
f_{21}(b_1, b_2) &= \begin{cases} 1, & \text{if } b_1.\mathit{bug}_{\mathit{severity}} = b_2.\mathit{bug}_{\mathit{severity}} \\ 0, & \text{otherwise} \end{cases} \\
f_{22}(b_1, b_2) &= \frac{1}{1 - |b_1.\mathit{priority} - b_2.\mathit{priority}|} \\
f_{23}(b_1, b_2) &= \frac{1}{1 - |b_1.\mathit{version} - b_2.\mathit{version}|}
\end{aligned}
$$

Categorical Features

53% of all duplicates were submitted within 20 days of master.

$$f_{24}(b_1, b_2) = |b_1.creation_{ts} - b_2.creation_{ts}|$$



$$f_{25}(b_1, b_2) = |b_1.bug_{id} - b_2.bug_{id}|$$

Binary Classification

We ran:

- K Nearest Neighbours
- Linear Support Vector Machine
- RBF Support Vector Machine
- Decision Tree
- Random Forest
- Naïve Bayes

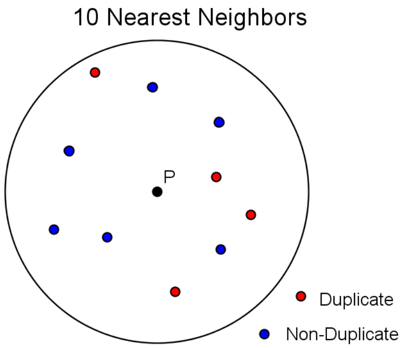
Binary classification models implemented in the Python package, *scikit-learn* [4].

K Nearest Neighbors

- All instances in the training dataset are stored
- When a new report is tested, the closest *k* neighbors are calculated using Euclidean distance.
- If the majority of those neighbors belong to one class, then the new report is predicted to be in that class as well.

K Nearest Neighbors

- All instances in the training dataset are stored
- When a new report is tested, the closest *k* neighbors are calculated using Euclidean distance.
- If the majority of those neighbors belong to one class, then the new report is predicted to be in that class as well.



Linear Support Vector Machine

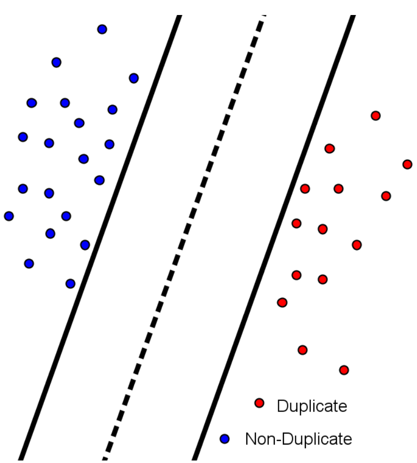
Definition

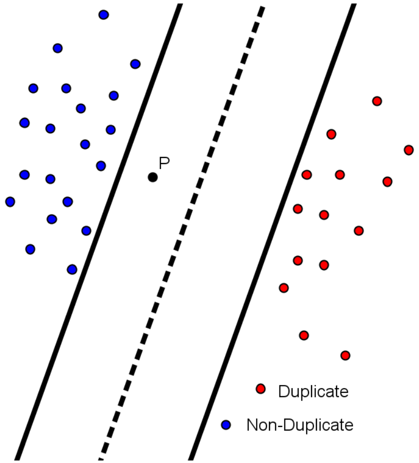
Given some training data $\mathcal{D}$, a set of n points of the form

$$\mathcal{D} = \{(\mathbf{x}_i, y_i) \mid \mathbf{x}_i \in \mathbb{R}^m,\ y_i \in \{-1, 1\}\}_{i=1}^{n}$$

- Two hyperplanes that divide all points with $y_i = 1$ from $y_i = -1$ with no points are between the two planes.
- Largest such planes are called maximum-margin hyperplanes.
- The region bounded by them is called “the margin”.
- A hyperplane then divides the the margin in half.
- Test pairs are then mapped into m-space.
- Status determined by position relative to hyperplane

Linear Support Vector Machine





Radial Basis Function Support Vector Machine

Similar to Linear SVM, except that the dot products used to define the hyperplane are replaced by Gaussian radial basis function.

Definition

Let $r = \|\mathbf{x} - \mathbf{x}_i\|$ and $\epsilon > 0$. Then, define the Gaussian radial basis function to be:

$$\phi(r) = e^{-(\varepsilon r)^2}$$

- Fits maximum-margin hyperplane in a transformed feature space.
- Though the classifier is a hyperplane in the high-dimensional feature space, it may be nonlinear in the original input space.
- Using the Gaussian radial basis function, the corresponding feature space is a Hilbert space of infinite dimension!
- A Hilbert space is an abstract vector space possessing the structure of an inner product that allows length and angle to be measured.

Decision Tree and Random Forest

Decision trees

- leaves represent class labels
- branches represent conjunctions of features
- “learned” by splitting set into subsets based on feature.
- Repeated on each derived subset in a recursive manner.
- Recursion is completed when the subset has the same value
- This top-down induction of decision trees is a greedy algorithm

Decision Tree and Random Forest

Decision trees

- leaves represent class labels
- branches represent conjunctions of features
- “learned” by splitting set into subsets based on feature.
- Repeated on each derived subset in a recursive manner.
- Recursion is completed when the subset has the same value
- This top-down induction of decision trees is a greedy algorithm

Random forest

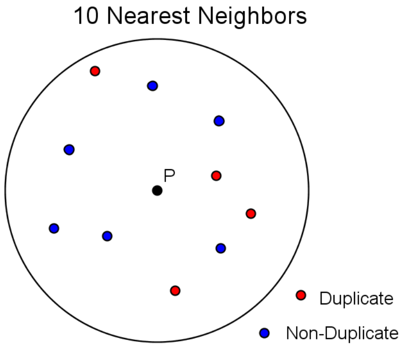
- Classifier uses a number of decision trees
- Improves the classification rate.
- Random sample selected from the training set
- Decision tree created for that set and process is repeated
- Trees are then averaged together to create a random forest.

Naïve Bayes

- Takes probability of class into account
- 1:4 ratio of duplicates to non duplicates in training set.
- For test point, the proportions of neighboring duplicates and non duplicates are calculated.
- These proportions are compared with the expected probabilities.
- The new query is then labeled accordingly

Naïve Bayes

- Takes probability of class into account
- 1:4 ratio of duplicates to non duplicates in training set.
- For test point, the proportions of neighboring duplicates and non duplicates are calculated.
- These proportions are compared with the expected probabilities.
- The new query is then labeled accordingly



Measures

Definition

Let t_p, t_n, f_p, and f_n be the number of true positive, true negatives, false positives, and false negatives, respectively. Then,

$$\text{Accuracy} = \frac{t_p + t_n}{t_p + t_n + f_p + f_n}$$

$$\text{Precision} = \frac{t_p}{t_p + f_p}$$

$$\text{Recall} = \frac{t_p}{t_p + f_n}$$

Area Under the Curve (AUC) is created by plotting the fraction of true positives out of the total actual positives vs. the fraction of false positives out of the total actual negatives, at various threshold settings.

Results

Eclipse	NKN	LSVM	RBF SVM	Tree	Forest	Naïve
Accuracy:	0.99992	0.99996	0.99984	0.99984	0.99996	1
Precision:	0.99960	0.99980	0.99921	0.99921	0.99980	1
Recall:	1	1	1	1	1	1
AUC:	0.99995	0.99998	0.9999	0.9999	0.99998	1

Open Office	NKN	LSVM	RBF SVM	Tree	Forest	Naïve
Accuracy:	0.99993	0.99993	0.99970	0.99980	0.99990	0.99993
Precision:	0.99966	0.99966	0.99847	0.99898	0.99949	0.99966
Recall:	1	1	1	1	1	1
AUC:	0.99996	0.99996	0.99981	0.99987	0.99994	0.99996

Mozilla	NKN	LSVM	RBF SVM	Tree	Forest	Naïve
Accuracy:	0.99995	0.99995	0.99994	0.99935	0.99994	0.99981
Precision:	0.99975	0.99975	0.99968	0.99694	0.99981	1
Recall:	1	1	1	0.99981	0.99988	0.99904
AUC:	0.99997	0.99997	0.99996	0.99952	0.99991	0.99952

Preliminary Results and Observations

- Textual features together with categorical features provide very good classification results!
- Accuracy is over 99% for all combinations of datasets and classification algorithms.
- A recall of 100% was obtained for all datasets (positive instances correctly classified).
- Precision is also high, but not 100%. (A few pairs of non-duplicate bug pairs were classified as duplicates.)
- Only two pairs of bug reports were misclassified.
- Three of the classification algorithms are common, but the additional algorithms we used may work better for larger datasets.

Comparison to Recent Published Works

- The new set of textual features presented in this paper improves the accuracy between 3.25% and 6.32% over the contextual approach proposed by Alipour et al.

	New Features		Alipour	
	Accuracy	AUC	Accuracy	AUC
Eclipse	100.0000	1.0000	96.75	0.9900
Open Office	99.9899	0.9999	93.67	0.9660
Mozilla	99.9930	0.9999	94.78	0.9430

Table: Comparison with Alipour's Results [1]

Sun et al. [6] proposed features 19 - 23, in addition measures based on the *BM25F*. Their results are reported in terms of top-k list. However, no more than 80% of the duplicates were correctly identified.

Conclusions and Future Work

- Improved method based on textual similarity measures
- A total of 25 new textual features are used.
- Binary classification methods categorize bugs into two classes.
- Tested on bug reports from Eclipse, Open Office, and Mozilla.
- Improves duplicate bug report detection by 6.32% .
- These preliminary results are very promising.
- In future work, we plan on using much larger datasets.

Bibliography

A. Alipour, A. Hindle, and E. Stroulia.
A contextual approach towards more accurate duplicate bug report detection.
Proceedings of the Tenth International Workshop on Mining Software Repositories, pages 183–192, 2013.

C.-C. Chang and C.-J. Lin.
LIBSVM: A library for support vector machines.
ACM Transactions on Intelligent Systems and Technology, 2:27:1–27:27, 2011.
Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

A. Lazar, S. Ritchey, and B. Sharif.
Improving the accuracy of duplicate bug report detection using textual similarity measures.
Proceedings of the Eleventh International Workshop on Mining Software Repositories, 2014.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay.
Scikit-learn: Machine learning in Python.
Journal of Machine Learning Research, 12:2825–2830, 2011.

F. Saric, G. Glavas, M. Karan, J. Snajder, and B. Basic.
Takelab: Systems for measuring semantic text similarity.
In *Proceedings of the First Joint Conference on Lexical and Computational Semantics*, pages 441–448, Montreal, Canada, June 2012.

C. Sun, D. Lo, S. Khoo, and J. Jiang.
Towards more accurate retrieval of duplicate bug reports.
Proceedings of the 26th IEEE/ACM Automated Software Engineering, pages 253–262, 2011.

Thanks Dr. Lazar and Dr. Sharif for awesome advising!