Detecting Duplicate Bug Report Using Character N-Gram-Based Features

Ashish Sureka
Indraprastha Institute of Information Technology (IIIT)
New Delhi, India
ashish@iiitd.ac.in

Pankaj Jalote
Indraprastha Institute of Information Technology (IIIT)
New Delhi, India
jalote@iiitd.ac.in

***Abstract*—We present an approach to identify duplicate bug reports expressed in free-form text. Duplicate reports needs to be identified to avoid a situation where duplicate reports get assigned to multiple developers. Also, duplicate reports can contain complementary information which can be useful for bug fixing. Automatic identification of duplicate reports (from thousands of existing reports in a bug repository) can increase the productivity of a Triager by reducing the amount of time a Triager spends in searching for duplicate bug reports of any incoming report. The proposed method uses character N-gram-based model for the task of duplicate bug report detection. Previous approaches are word-based whereas this study investigates the usefulness of low-level features based on characters which have certain inherent advantages (such as natural-language independence, robustness towards noisy data and effective handling of domain specific term variations) over word-based features for the problem of duplicate bug report detection. The proposed solution is evaluated on a publicly-available dataset consisting of more than 200 thousand bug reports from the open-source Eclipse project. The dataset consists of ground-truth (pre-annotated dataset having bug reports tagged as duplicate by the Triager). Empirical results and evaluation metrics quantifying retrieval performance indicate that the approach is effective.**

***Keywords*-Bug Report Analysis, Duplicate Bug Detection, Text Classification, Software Engineering Task Automation, Software Testing and Maintenance**

I. Introduction

Defect (also referred to as Issue or Bug) reporting is an integral part of a software development, testing and maintenance process. Typically, bugs are reported to an issue tracking system which is analyzed by a Triager (who has the knowledge of the system, project and developers) for performing activities like: quality check to ensure if the report contains all the useful and required information, duplicate detection, routing it to the appropriate expert for correction and editing various project-specific metadata and properties associated with the report (such as current status, assigned developer, severity level and expected time to closure). It has been observed that often a bug report submitted by a tester is a duplicate (two bug reports are said to be duplicates if they describe the same issue or problem and thereby have the same solution to fix the issue) of an existing bug report. Studies show that the percentage of duplicate bug reports can be up-to 25-30% [1][5][7]. This can significantly hamper the bug fixing process and product release [7]. Following are some facts on the volume of bug reports submitted to defect tracking systems:

- 13,016 bug reports were filed from Jun-04 to Jun-05 between the release of Eclipse Platform version 3.0 and 3.1: averaging 37 reports per day, with a maximum of 220 reports in a single day [1].
- Number of bugs reported for Mozilla project = 51,154 during the period 2002-2006 [3].
- In 2005, for Mozilla project, almost 300 bugs filed every day needed triaging [3].

Filtering duplicates in complex and large project settings requires considerable amount of Triager's (a trained and knowledgeable analyst) manual effort and knowledge of the system. Solutions for automating the process of duplicate detection can result in increased productivity of the Triager, speed-up the process of defect management and thus result in reduced software maintenance cost. Bug reports are primarily expressed in natural language text (free-form text consisting of human language used for everyday communication). As a result of which recently there has been a considerable research interest in the application of natural language processing and text analytics techniques for the purpose of duplicate bug report detection [5][7][9][14][16][17]. Automatic detection of duplicate defect reports and linking similar defect reports is a technically challenging problem due to the following reasons:

- Bug reports are expressed in natural language text. Natural language is vast and ambiguous.
- The quantity of problem reports in large and complex software setting is huge.
- Presence of poorly expressed bug reports (missing information, noisy text) poses additional technical challenges.

The aim of the research study presented in this paper is to investigate novel text mining based approaches to analyze bug databases to uncover interesting patterns and knowledge which can be used to support developers and triagers in identification of duplicate bug reports. In the proposed approach we compute the semantic and lexical similarity (in the context of software defect management) of a given (incoming or query) bug-report with existing reports in the defect tracking system to predict the likelihood of it

BUG ID	TITLE
211356	Clipboard example crashes
210894	Examples that use StyledText crash on startup
212404	[persistence] ArrayIndexOfBoundsException in SystemHostPool#getHosts()
206742	[persistence] NPE in SystemRegistry.getHostsBySystemType()
212148	[Markers] NPE in getMarkerItem
212002	[Markers] NPE in CachedMarkerBuilder.getMarkerItem
211897	User libraries often disappear
183117	User Library Lost
211868	Eclipse WTP crashes due to MyLyn Class not laoding
188524	warnings on startup in Europa
211649	[CDateTime] No Border in Win32
182797	[CDateTime] CDT.BORDER does not make effect for Win32
211281	[KeyBindings] CTRL+ALT+Q makes entering of the @-sign impossible on german keyboards
208707	(Console) Remove binding for show console view with CTRL+ALT+Q
294093	Out of memory error opening large files in Eclipse Text Editor
75086	Request for a memory efficient way of loading large documents (in the mega bytes range)
23193	Improve performance editing large .java files
295158	Canceling unresponsive jobs
123075	Allow unconditionally killing the job that refuses to cancel
208750	Background Jobs should have a kill option
272481	Ability to kill tasks

Table I
ILLUSTRATIVE EXAMPLES OF DUPLICATE BUG REPORT TITLES FROM THE JAVA ECLIPSE IDE OPEN SOURCE PROJECT

being a duplicate. Our motivation is to augment existing bug tracking systems with a duplicate bug report detection module where an analyst is presented with a list of Top-N bug reports (along with a numerical value representing a relatedness score) which are similar in terms of problem or fault description to the incoming bug report. The search result consisting of Top-N bug reports and numerical score can be visually inspected and compared with the query report by the Triager to categorize the incoming report as duplicate or unique. Table I lists illustrative examples of duplicate bug report titles from the Java Eclipse IDE open source project.

The approach presented in this paper departs from previous approaches for bug report detection as we make use of low-level character-level representation whereas all the previous approaches are based on word-level analysis. We hypothesize that the advantages of character-level analysis (discussed in detail in the next Section) is suitable to the task of automating duplicate bug report detection as some of the key linguistic features of a bug report can be better captured using character n-grams in contrast to word-level analysis. The performance of character-level representation for the task of bug report classification is an open question. Our research objective is to perform an empirical analysis on a publicly-available large dataset for answering the stated research question. The problem of duplicate bug detection has received considerable interests from researchers and is not a fully solved problem. The study presented in this paper attempts to advance the state-of-the-art on duplicate bug report detection and in context to the related work makes the following unique contributions:

1) A novel method for identifying duplicate bug reports (based on the information contained in bug title and description expressed in free-form text) using a character-level n-gram model for text similarity matching. While the method presented in this paper is not the first approach on duplicate bug report detection, this study is the first to investigate the performance of character-level n-gram language models for the task of duplicate bug report detection. There are two key advantages of character-level models: language independence useful for language portability and capturing of sub-word features useful for analysis of noisy text. All previous approaches are based on word-level analysis as a result of which they are inherently language dependent and rely on language specific pre-processing such as tokenization, stop-word removal, spelling correction and word-stemming. Language dependent text classification suffers from a drawback in terms of portability across languages or performing cross-lingual analysis (for example segmenting words from character sequences is a complex task in several Asian languages such as Chinese or Japanese) [13]. Character-level analysis is language independent as it does not require extensive language specific pre-processing. The paper by Peng et al. lists the advantages of character-level n-gram language models for language independent text categorization tasks [13]. Smart feature engineering is critical to text classification tasks as a document can be represented by a variety of low-level and high-level syntactic and semantic features [13][15]. This study is the first to investigates sub-word features (slices of *n* characters within a word) and a bag-of-characters document

representation model (rather than the bag-of-words model in previous approaches) for the task of duplicate bug report detection.

2) An empirical evaluation of the proposed approach on real-world and publicly available dataset from a popular open-source project. While some of the previous approaches for duplicate bug report detection have been evaluated on bug reports from Eclipse project, the size of the evaluation dataset from Eclipse project for the experiments performed in this study is the largest as compared to previous researches. Hiew et al. examine the performance of their duplicate bug report detection system on dataset of varying size. They notice that the duplicate detection accuracy decreases gracefully as more bug reports are added to the repository [5]. Naturally, the size of the bug repository will have an impact on the accuracy of a duplicate detection algorithm. The size of the bug repository used in previous researches on duplicate detection are: Hiew et al.: 21915 (Firefox), Hiew et al.: 37716 (Eclipse), Hiew et al.: 1782 (Apache 2.0), Hiew at al.: 22076 (Fedora Core), Jalbert et al.: 29000 (Mozilla), Kemp et al.: 14000 (Eclipse), Sun et al.: 12732 (OpenOffice), Sun et al.: 44652 (Eclipse), Sun et al.: 47704 (FireFox), Wang et al. 1749 (Firefox), Wang et al.: (small subset of Eclipse bug repository but exact number not clear from the paper), Runeson et al. (thousands of bug reports belonging to Sony Ericsson Mobile Communication Products but the exact number is not clear from the paper). The size of the bug repository in all the previous experiments is less than 50K whereas the size of the bug repository for the experiments in this study is more than 200K.

II. Solution Approach

Feature extraction and similarity computation (semantic and lexical) between two bug reports is central to the problem of duplicate detection. Extracting discriminatory features and important indicators from bug description is key to the performance of any duplicate bug report detection system. In this work, we experiment with low-level character n-gram based features which can complement features proposed in previous approaches. In the following sub-section, we describe the character n-gram model and argue in support of our hypothesis (using illustrative examples from Eclipse bug database). We argue that the proposed character n-gram model can capture important linguistic characteristics (discriminatory features) of bug reports which can play an important role for the task of duplicate identification.

A. Character-Level N-Gram Model

N-gram means a subsequence of N contiguous items within a sequence of items. Word n-grams represent a subsequence of words and character n-grams represent a subsequence of characters. For example, various word-level bi-grams (N=2) and tri-grams (N=3) in the phrase *Software Engineering Data Mining* are: *Software Engineering*, *Engineering Data*, *Data Mining*, *Software Engineering Data* and *Engineering Data Mining*. Similarly, various character-level bi-grams and tri-grams in the word *Software* are: *So*, *of*, *ft*, *tw*, *wa*, *ar*, *re*, *Sof*, *oft*, *ftw*, *twa*, *war* and *are* respectively. In the field of Information Retrieval and Natural Language Processing, representing a text document as a bag of word-level and character-level n-grams is a well-known technique used as a pre-processing step for applications like information retrieval, text classification and clustering [2][6][10][11]. In contrast to word-level n-gram representation, character-level n-gram representation has certain distinct advantages which align with the task of duplicate bug report detection. Following are the advantages of character-level representation over word-level representation in context to the problem of duplicate bug detection. We list the advantages by presenting illustrative examples from the evaluation dataset (Eclipse bug reports) used in our study.

1) Ability to Match Concepts from System Messages: Bug reports frequently contain source code segments and system error messages which is not natural language text. For example the term *StringIndexOutOfBoundsException* is a system message and contains important concepts denoted by the terms: *String*, *IndexOutOfBounds* and *Exception*. We notice that the same concept is expressed as *StringIndexOutOfBounds* (the word *Exception* is not mentioned) in a duplicate bug report containing the term *StringIndexOutOfBoundsException*. The two terms *StringIndexOutOfBoundsException* and *StringIndexOutOfBounds* have similar semantic interpretations but are lexically different. The two terms share several character n-grams (as a result of which a character-level matching algorithm will result in a non-zero score) but will not result in a match in a system based on word-level analysis. Another common example of semantically similar interpretation between two expressions having different words: *Out of memory* (three words seperated by space) and *OutOfMemoryError* (one compound term connecting four individual words). In the *Out of memory* Vs. *OutOfMemoryError* example, a character-based analysis will be able to capture and match importance concepts. For a word level analysis system, it is problematic to find the word boundaries of various concepts within a compound concept as the word boundaries can be arbitrary. We observe several such instances in the Eclipse bug reports. Table II lists concrete examples to illustrate the advantages of a character-level model for matching semantically similarity concepts contained in system generated messages and source code. Ability to match concepts from system messages is an important

Concept in one Bug Report	Similar Concept in duplicate Bug Report
viewerContributions	contributions
Add Java exception	AddExceptionDialog
StorageDocumentProvider#setDocumentContent	StorageDocumentProvider
Launch Configuration Type page of Property Sheet	LaunchConfigurationTypePropertyPage
IMethodBinding	method binding
IDebugViewAdapter	AbstractDebugView, IDebugView
ILabelDecorator	label decorators
Out of memory	OutOfMemoryError
TimeOutException	org.eclipse.jdi.TimeOutException

Table II
ILLUSTRATIVE EXAMPLES FROM ECLIPSE BUG DATABASE DEMONSTRATING THE ABILITY OF CHARACTER N-GRAMS TO MATCH SEMANTICALLY SIMILAR CONCEPTS

benefit of character-level model over previous approaches based on word-level analysis.

2) Ability to Extract Super-Word Features: Super-word features are features that spans across the limit of a single word. Character-level analysis can extract super-word features which are suitable for the task of computing semantic and lexical similarity between bug reports. For example, in the two phrases: *switching to* and *changing to*, the common character n-gram that spans across the word boundary is *ing to*. Similarly, the shared character n-gram in *error while* and *error when* is *error wh* and in *create getter* and *generate getter* is *ate getter*. *Switching to* and *changing to* are semantically related and this type of semantic relatedness can be captured using character-level models. Some concrete examples where super-word feature can be useful for solving the problem are: (*Outliner works*, *Reconciler works*), (*for unthrown*, *for unused*), (*does nothing*, *does not*), (*when synchronizing*, *when synch*), (*in TextEditor*, *in TextViewer*), (*Java browser*, *java browsing*).

3) Handling Misspelled Words: Online bug reports and the community discussion on a reported bug belongs to the category of user generated content (UGC) which is often not professionally edited or proof-read in contrast to formal documents like a user guide, product manual or a software requirement specification document. It is common to find misspelled words in user comments and unless a spell checker is applied, a word-level similarity algorithm will regard two words as different if they do not match exactly. Also, spelling mistakes in some domain specific terms (such as *Bugzilla* which is the name of a software application) cannot be corrected by standard language specific spell-checkers as they may not be present in the spell-checker lexicon (for example, MS Word identifies *Bugzilla* as misspelled word but does not make any spelling suggestion). Character-level n-gram features are resilient to misspelled words as only a limited portion of the word is affected whereas features based on remaining portion with correct spelling are still retained. Character-level feature extraction eliminates the need of having a language and domain specific spell-checker and is thus advantageous in situations where spelling mistakes are frequent. For example the character quad-grams for the misspelled word *dinamically* are: *dina*, *inam*, *nami*, *amic*, *mica*, *ical*, *call* and *ally*. We notice that except the quad-grams *dina* and *inam*, the rest of the quad-grams are contained in the correctly spelled word *dynamically*.

4) Ability to Match Short-Forms with their Expanded-Form: Usage of abbreviations and short-forms is common in a communication between people having similar background and belonging to the same domain. In a word-level analysis the words *configuration* and *config* will be treated as different. In a character-level analysis, since the n-gram *config* is a substring of the larger string *configuration*, the system returns a non-zero similarity score for the two words. A word-level analysis system needs to have knowledge of the specific domain and language to pre-process config so that it can match its expanded form of configuration. A character-level analysis model is more robust in matching abbreviations and short-forms to their expanded from and does not require a linguistic resource like a short-form resolution module as a pre-preprocessing step. Some of the concrete examples in the evaluation dataset are: (*nav*, *navigator*), (*synch*, *synchronizing*), (*config*, *configuration*), (*temp*, *temporary*), (*multi*, *multiple*), (*anon*, *anonymous*), (*doc*, *documentation*).

5) Ability to Match Term Variations to a Common Root: Character-level analyses are good at handling morphological variations of words without requiring a word stemmer. For example, the words *selection*, *selecting* and *selected* are variations of the word *select* and have similar semantic interpretations. In a word-level processing system, a stemmer (a program to reduce a word to its root-form also called as stem) is required to match variants derived from a common root-word. The words *computing*, *computation*, *computed* shares a common character n-gram *comput* and hence a system based on substring analysis will return a non-zero similarity score between variations which are spelled differently but have similar semantic interpretations. Some of the concrete and common examples in the Eclipse bug reports are: (*class*, *classes*), (*type*, *types*), (*project*, *projects*), (*error*, *errors*), (*dialog*, *dialogs*), (*search*, *searching*), (*link*, *linking*), (*copy*, *copying*), (*enable*, *enabled*).

6) Ability to Match Hyphenated Phrases: Hyphenated phrases consisting of words separated by a hyphen are frequently used to describe compound concepts. For example some of the common phrases in the experimental dataset are: (*drag-and-drop*, *drag-drop*, *drag and drop*), (*plug-in*, *plugin*, *plug in*), (*Ctrl-F7*, *Ctrl F7*), (*read-only*, *read only*), (*out of synch*, *out-of-synch*). *Drag-and-drop* and *drag and drop* share several common character n-grams without requiring a pre-processing step of removing hyphens. Similarly, we notice that in bug reports several terms are enclosed in rectangular brackets such as: *[UI]*, *[Dialogs]*, *[Resources]*. A word-level analysis system needs to have knowledge of this fact and perform word tokenization in such a way that the rectangular brackets are not concatenated with the word or phrase that needs to be extracted.

B. Similarity Score between Two Bug Reports

Figure 1 presents a high-level architecture of the proposed solution. The main components of the solution architecture are: character n-gram feature extractor, similarity (between two bug reports based on comparing title and description) computation module and Top-N similar bug reports retrieval module based on pre-defined threshold learned from experience and historical data. The feature extraction module (FEDR) extracts all the character n-grams of size 4 to 10 from the title of the two bug reports that needs to be compared. Once the character n-grams from the title are extracted, the overall similarity score between two bug reports is calculated as a function of the following three variables:

1) Number of shared character n-grams between the two bug-reports (BR1 and BR2)
2) Number of character n-grams extracted from the title of the first bug report (BR1) present in the description of the second bug report (BR2).
3) Number of character n-grams extracted from the title of the second bug report (BR2) present in the description of the first bug report (BR1).

Product	**Component**	**Num. Records**
Same	Same	19903 (73.62%)
Same	Different	4056 (15.00%)
Different	Same	1209 (4.48%%)
Different	Different	1868 (6.90%)

Table III
PERCENTAGE OF DUPLICATE BUG REPORTS BELONGING TO SAME OR DIFFERENT PRODUCT AND COMPONENT

III. EXPERIMENTAL EVALUATION

A. Evaluation Dataset

Bug reports for several open source projects such as Mozilla Firefox[1] and Eclipse[2] are available in public domain and can be exported in XML format for research and experimental purposes. We downloaded 213000 Eclipse bugs (XML format) from the MSR (Mining Software Repositories) Mining Challenge 2008 website[3]. The data consists of Eclipse bugs numbered 1-213000. We use the Eclipse dataset from MSR track as it serves as a publicly available benchmark set for several bug database mining tasks and thus enables the performance of our approach to be compared with other approaches on the same dataset. Also, the dataset contains the ground-truth (pre-annotated having bug-reports tagged as duplicate by the Triager).

B. Performance Metrics

Due to the nature of the problem, the standard measures of precision (percentage of retrieved results relevant to the query or information need) and recall (percentage of the relevant results in the repository retrieved as search results in response to a query) cannot be directly used and hence the commonly used evaluation metrics used by researchers for the problem of duplicate big detection is *recall rate* [14]. *Recall rate* (RR) measures the accuracy of the duplicate detection system in terms of counting the percentage of duplicates (a query which is a duplicate) for which the master bug-report is found within the top-N search results. [14][17][16]. The *recall rate* measure overcomes the limitation of the standard *recall* measure which will result in a value which is either 0% (not found) or 100% (found). The *recall rate* measure was originally proposed by Johan et al. [12] for a different problem and has been adapted by Runeson et al. and Wang et al. for their experiments. We use the same evaluation metric for the solution proposed as it has been well accepted by previous work on similar problem.

C. Empirical Results

The bug reports in the evaluation dataset are numbered from 1 to 213000. However, there are some bug reports

[1]https://bugzilla.mozilla.org/
[2]https://bugs.eclipse.org/bugs/
[3]http://msr.uwaterloo.ca/msr2008/challenge/

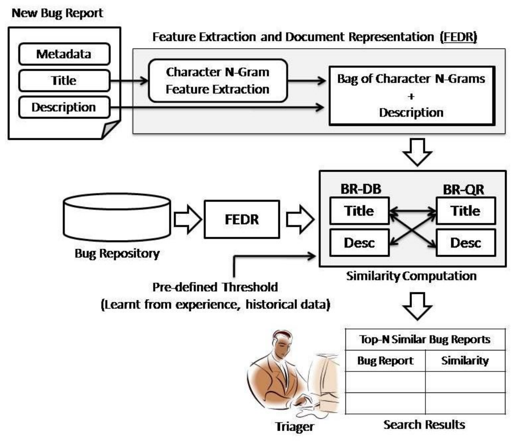
New Bug Report
Metadata
Title
Description
Feature Extraction and Document Representation (FEDR)
Character N-Gram Feature Extraction
Bag of Character N-Grams + Description
Bug Repository
FEDR
BR-DB
BR-QR
Title
Desc
Title
Desc
Similarity Computation
Pre-defined Threshold (Learnt from experience, historical data)
Top-N Similar Bug Reports
Bug Report
Similarity
Triager
Search Results

Figure 1. High-level system architecture of the proposed solution

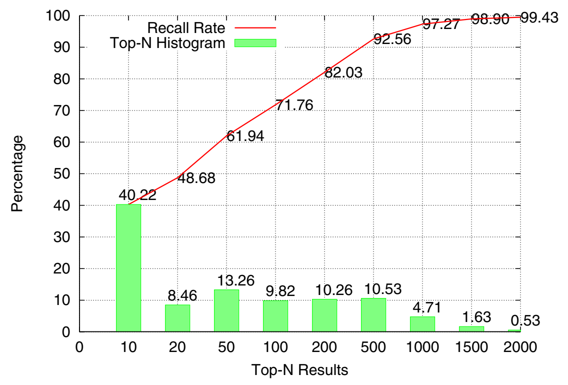

Figure 2. Recall rate for 2270 bug reports with Title-Title score greater than 50 based on Title-Title similarity

B1	B2	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%	85%	90%	95%	100%
T	T	9	12	16	21	28	35	42	50	62	73	87	106	132	176	273	1290
T	D	21	27	32	38	45	51	59	67	76	86	98	113	132	158	209	938
D	T	22	28	34	40	47	54	61	70	79	89	101	116	134	159	206	994
ID DIFF		357	623	983	1475	2197	3138	4326	5983	8236	11151	15335	21513	30275	43487	68624	205949

Table IV
QUANTILES INFORMATION FOR THE FOUR VARIABLES: T-T, T-D, D-T AND ID DIFF

which are not available in the dataset (either the title or description is null or it cannot be retrieved due to formatting problems). We were able to retrieve 205242 bug reports from the input dataset size of 213000. The number of bug reports marked as duplicates in the evaluation dataset are 27036 (27036/205242 = 13.17%). Compared to the previous approaches on duplicate bug detection using Eclipse bug database [5][9][16][17], this paper presents an empirical analysis on the largest evaluation dataset. The size of the bug repository has implications for the purpose of evaluating the retrieval or classification performance of a duplicate bug report detection system. The larger the input dataset, the more difficult (the number of false positives can increase) and time consuming it is for the similarity computation module to filter the true positive from similar (but not duplicate) bug reports.

Each bug report in the Eclipse bug database is assigned a Product, Component and Version in addition to several fields such as severity, time-stamp and priority. The number of different Products across the bug reports in the evaluation dataset are 77 (for example: Platform, JDT, Equinox, Web Tools, EPP and Java Server Faces), number of Components are 482 (Resources, Compare, UI, Build, Doc, Debug, SWT, IDE, CVS and Server) and number of version are 165 (1.1.4, 2.0, 3.3, 1.0.0 Release, Ganymade and 4.0 Beta-1). We compared the Platform and Component of two reports which are duplicates of each-other to investigate if they belong to the same Product and Component. This result of such analysis have potential implications in the retrieval strategy. Table III shows that 73.62% of the duplicate bug reports belong to the same component and product whereas 26.38% of the pair of duplicate bug reports have differences in the Product and/or Component. We note that more than 1/4th of the duplicate bug reports do not belong to the same Product and Component and hence we compare the query bug report with all the bug reports rather than applying a filter based on the Product and Component attribute.

We compute the values of four variables for 27036 duplicate bug reports in the dataset: number of shared character n-grams between the titles of two reports (T-T) , number of character n-grams in the title of the first report present in the description of the second report (T-D) , number of character n-grams in the title of the second report present in the description of the first report (D-T) and the difference between the bug ids of the two reports (ID DIFF). Table IV presents the quantiles information for the four variables: T-T, T-D, D-T and ID DIFF. The letter *T* in the Table IV represents Title, *D* represents Description, *B1* and *B2* represents bug report 1 and bug report 2 respectively. Our objective is to determine (or learn from historical dataset) the values of the four variables that divides the dataset into equal-sized subsets. The data in Table IV is used

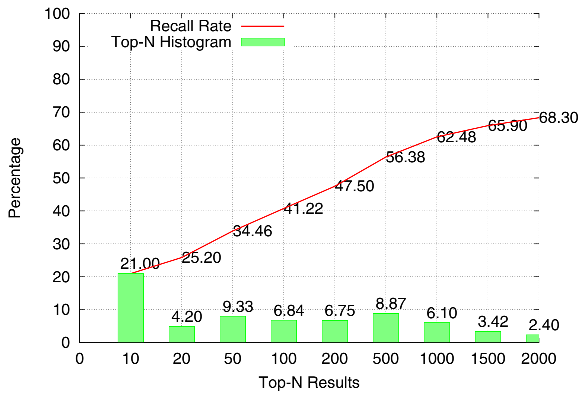

Figure 3. Recall rate for 1100 bug reports based on T-T, T-D and D-T similarity

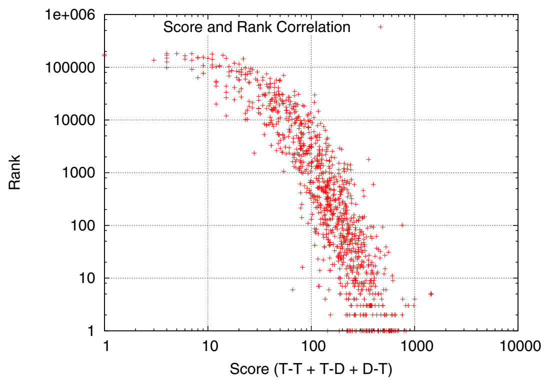

Figure 4. Scatter plot illustrating the correlation between rank and score (T-T + T-D + D-T) for 1100 bug reports

to understand the distribution of the four variables which represent similarity between reports. For example, 40% of the data points has the value of variable *T-T* between 50 and 1290. The quantile information in Table IV reveal interesting properties which can be used to optimize the duplicate bug detection procedure. Based on our analysis of the previous papers on bug report detection, the quantile-based analysis of the dataset presented in this paper offers a fresh perspective as such analysis has not been applied in the past literature on duplicate bug report detection. We notice that the average difference between the bug ids of the master and duplicate report is less than 3138 (median value) for 50% of the bug reports and 21513 for 80% of the bug reports in the dataset. The exact value may differ across the nature of the project and software but such information can be automatically derived from historical data of the specific dataset and can be used to optimize the search algorithm. According to the Table IV, a duplicate bug report for an incoming report has a likelihood of 80% to be within the radius of 21513 bug reports. However, we notice that for 5% of the duplicate bug reports, the distance between the duplicate and master report is more than 68624. We notice that for 35% of the duplicate reports, the shared character level n-grams between titles is less than or equal to 16. This shows that there are several cases where there is a small character n-gram based overlap between titles and in such cases the role of detailed description is important (variables *T-D* and *D-T* in our case). The quantile information for variables *T-D* and *D-T* in Table IV reveals that on an average there is a 50-55 character n-grams common between the title of one duplicate report and the description of its corresponding master report.

We randomly selected 2270 bug reports from the set of pre-annotated duplicate bug reports for which the character n-gram similarity between the titles of the master and duplicate is greater than or equal to 50. Our objective is to understand the performance of the system for cases belonging to 40% of the total bug reports where the variable T-T is greater than a pre-defined threshold of 50 (refer to Table IV). Figure 2 presents the recall rate (based on T-T similarity) for Top-N results. We observe that the system has a recall rate of 40.22% for Top-10 results and 61.94% for Top-50. Figure 3 presents performance results for randomly selected 1100 duplicate bug reports (i.e., a random selection made from the set of all duplicate bug reports) without any predefined threshold. Figure 3 presents the recall rate (based on T-T + T-D + D-T similarity) and histogram for Top-N results. Figure 4 presents a scatter plot illustrating the correlation between the similarity score (T-T + T-D + D-T) and the rank. The scatter plot reveals that a similarity score of 100 to 1000 results in Top 100 rank. We notice that for several pairs of master and duplicate bug reports the similarity score is quite low and hence the character based similarity approach needs to be augmented to correctly classify cases belonging to this set. We observe that the system has a recall rate of 21.00% for Top-10 results and 33.92% for Top-50 (refer to Figure 3).

IV. Related Work

Mining software repositories is an emerging field that has received significant research interest in recent times. Several tools and techniques based on data mining approaches have been developed to assist a practitioner in decision making and automating software engineering tasks [4][8]. Following are the papers applying data and text mining for the task of duplicate bug report detection which is the focus of this paper.

Hiew et al. presents an approach based on grouping similar reports in the repository and deriving a centroid of the clustered reports [5]. An incoming report (represented as a document vector) is then compared to the centroids of bug report groups to compute a similarity score for detecting duplicates based on a predefined threshold value. One of the unique features of the approach presented by Hiew et al. is pre-processing of bug reports to create a model (model updated as new bug reports arrived) consisting bug-report groups and their respective centroid which is then used for similarity computations. Hiew et al. perform experiments on bug report data from open-source projects such as Eclipse and Firefox project [5].

Runeson et al. apply the standard vector-space model approach discussed in Information Retrieval literature. The approach consists of representing a bug report as a weighted vector of words (each index word in the document set represents a dimension in the vector space) and then computing the distance (similarity measures such as Cosine, Dice and Jaccard) between multi-dimensional vectors for determining similar and duplicate bug reports [14]. Runeson et al. also create a domain specific thesaurus (list of semantically related terms), apply stop-word removal, spell checking and word stemming. Runeson et al. perform experiments on thousands of reports belonging to the mobile phone domain (Sony Ericsson Mobile Communications) [14].

Jalbert et al. present a classifier for categorizing a bug report as duplicate based on surface level textual features, textual similarity metrics and graph clustering algorithms [7]. Jalbert et al. perform experiments on 29000 bug reports from the Mozilla project and identify 25.9% of the reports at duplicates. The basis of the classifier implemented by Jalbert et al. is a linear regression over properties of bug reports. Jalbert at al. also use a bag of word approach for creating a multi-dimensional word vector representation of the text document and use cosine based similarity measure for computing the distance between query report and the reports in the defect management system [7].

Wang et al. hypothesize that execution information (execution trace and list of executed methods) of bug-revealing runs convey useful information and can be leveraged in conjunction with natural language information for the purpose of identifying duplicate defect reports [17]. Wang et al. perform experiments on a subset of the Eclipse and Firefox bug repository and conclude that exploiting execution information as additional data results in accuracy improvements of duplicate bug detection [17].

Kemp et al. approach is based on natural language vector-cosine techniques for duplicate text identification, using machine learning and neural networks to build a predictive model for categorizing a bug report as duplicate or unique [9]. The experimental dataset for the work presented by Kemp et al. consists of about 14000 bug reports from the Eclipse Project.

Sun et al. [16] apply a discriminative model based approach (SVM learning algorithm) for duplicate bug report detection. Their approach consists of three main steps: preprocessing (tokenization, stemming and stop words removal), training a discriminative model (Support Vector Machine based Model) and retrieving duplicate bug reports. Sun et al. evaluate their approach on bug report datasets from three popular open-source projects: Firefox (web browser), Eclipse (integrated development environment) and OpenOffice (rich text editor).

V. Conclusions

This paper presents an approach to compute text similarity between two bug reports to assist a Triager in the task of duplicate bug report detection. The central idea behind the proposed approach is the application of character n-grams as low-level features to represent the title and detailed description of a bug report. The advantages of the approach are language independence as it does not require language specific pre-processing and ability to capture sub-word features which is useful in situations requiring comparison of noisy text. The approach is evaluated on a bug database consisting of more than 200 thousand bug reports from open-source Eclipse project. The recall rate for the Top 50 results is 33.92% for 1100 randomly selected test cases and 61.94% for 2270 randomly selected test cases with a title to title similarity (between the master and the duplicate) of more than a pre-defined threshold of 50.

References

[1] John Anvik, Lyndon Hiew, and Gail C. Murphy. Who should fix this bug? In *ICSE '06: Proceedings of the 28th international conference on Software engineering*, pages 361–370, New York, NY, USA, 2006. ACM.

[2] W. B. Cavnar and J. M. Trenkle. 3rd annual symposium on document analysis and information retrieval. In *International Journal on Artificial Intelligence Tools, Volume 16 Number 6*, 1994.

[3] T. Nguyen W. Weimer C. Le Goues Forrest, S. Fixing software bugs in 10 minutes or less using evolutionary computation. In *Human-Computer Competition The 6th Annual (2009) HUMIES AWARDS*, 2009.

[4] Ahmed E. Hassan and Tao Xie. Mining software engineering data. In *Proceedings of the 32nd International Conference on Software Engineering (ICSE 2010), Companion Volume, Tutorial*, Cape Town, South Africa, May 2010.

[5] Lyndon Hiew. Assisted detection of duplicate bug reports. In *MS Computer Science Thesis, Department of Computer Science*, British Columbia, Canada, 2003. The University of British Columbia.

[6] Ioannis Houvardas Efstathios Stamatatos Ioannis Kanaris, Konstantinos Kanaris. Words versus character n-grams for anti-spam filtering. In *International Journal on Artificial Intelligence Tools, Volume 16 Number 6*, pages 1047–1067.

[7] Nicholas Jalbert and Westley Weimer. Automated duplicate detection for bug tracking systems. In *DSN 2008: 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 52–61, Anchorage, Alaska, USA, 2008. IEEE Computer Society.

[8] Huzefa Kagdi, Michael L. Collard, and Jonathan I. Maletic. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. volume 19, pages 77–131, New York, NY, USA, 2007. John Wiley & Sons, Inc.

[9] Trevor C. Kemp. Automated detection of duplicate free-form english bug reports. In *MS Computer Science Thesis, Department of Computer Science*, Morgantown, West Virginia, USA, 2009. West Virginia University.

[10] Paul Mcnamee and James Mayfield. Character n-gram tokenization for european language text retrieval. *Inf. Retr.*, 7(1-2):73–97, 2004.

[11] Yingbo Miao, Vlado Kešelj, and Evangelos Milios. Document clustering using character n-grams: a comparative evaluation with term-based and word-based clustering. In *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 357–358, New York, NY, USA, 2005. ACM.

[12] Johan Natt Och Dag, Thomas Thelin, and Björn Regnell. An experiment on linguistic tool support for consolidation of requirements from multiple sources in market-driven product development. volume 11, pages 303–329, Hingham, MA, USA, 2006. Kluwer Academic Publishers.

[13] Fuchun Peng, Dale Schuurmans, and Shaojun Wang. Language and task independent text categorization with simple language models. In *NAACL '03: Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, pages 110–117, Morristown, NJ, USA, 2003. Association for Computational Linguistics.

[14] Per Runeson, Magnus Alexandersson, and Oskar Nyholm. Detection of duplicate defect reports using natural language processing. In *ICSE '07: Proceedings of the 29th international conference on Software Engineering*, pages 499–510, Washington, DC, USA, 2007. IEEE Computer Society.

[15] Sam Scott and Stan Matwin. Feature engineering for text classification. In *ICML '99: Proceedings of the Sixteenth International Conference on Machine Learning*, pages 379–388, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[16] Chengnian Sun, David Lo, Xiaoyin Wang, Jing Jiang, and Siau-Cheng Khoo. Discriminative model approach towards accurate duplicate bug report retrieval. In *ICSE 2010: Proceedings of the 32nd international conference on Software Engineering*, Cape Town, South Africa, 2010. IEEE Computer Society.

[17] Xiaoyin Wang, Lu Zhang, Tao Xie, John Anvik, and Jiasu Sun. An approach to detecting duplicate bug reports using natural language and execution information. In *ICSE '08: Proceedings of the 30th international conference on Software engineering*, pages 461–470, New York, NY, USA, 2008. ACM.