Detecting Duplicate Bug Reports using Natural Language Processing

Jared Murphy
Golisano College of Computing and Information Sciences
Rochester Institute of Technology
jmm2188@rit.edu

Rajan Anand
Golisano College of Computing and Information Sciences
Rochester Institute of Technology
ra9303@rit.edu

Sathish Sakthivelan
Golisano College of Computing and Information Sciences
Rochester Institute of Technology
sathish@mail.rit.edu

Swarnabh Kashyap
Golisano College of Computing and Information Sciences
Rochester Institute of Technology
sk2052@rit.edu

1. INTRODUCTION

Software maintenance is a significant part of the software development lifecycle [5]. It can often account for a large proportion of a project's total expenses, so most, if not all, businesses have a great interest in ensuring that no time is wasted in the maintenance process. One of the maintenance areas that commonly lacks efficiency is Bug Reporting, where a software bug, which is defined as an error, flaw or fault in a computer program or system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways [4] is reported by the users back to the developers of the software. Usually, these bug reports are collected as logs that contain the details of the scenario in which that particular bug was found and is sent back. Developers generally employ a bug tracking system (e.g., Jira, Bugzilla) to file bugs found by testers and issues encountered by users [3], however, it is often the case that many of these reports describe identical issues or sometimes the same issues but described differently using different titles, or different context or different vocabulary and phrasing. This is usually a problem that snowballs for larger projects. Multiple bug reports detailing the same bug are generated by multiple testers, who are generally unaware of the bugs reported or unable to identify existing logged bug reports as the bugs they are encountering. These duplicate bug reports become noise for the developers managing the projects. Some studies have found that upwards of 20% of bug reports are duplicates of this kind [2]. Bug reports can often be quite cryptic and verbose, thus causing the tester to use a fair amount of time digesting the information. A large amount of duplicate bug reports cause developers and testers to spend much time looking over redundant data, and thus time and money are wasted in the process. With the rapid development expansion of Open-Source Software, this problem seems to be exploding in proportions in future thus creating a need for tools that can automate the process of detection of such duplicate bug reports quickly and accurately so that software maintenance resources can be allocated for future feature development and is not wasted on these monotonous non-value-added activity.

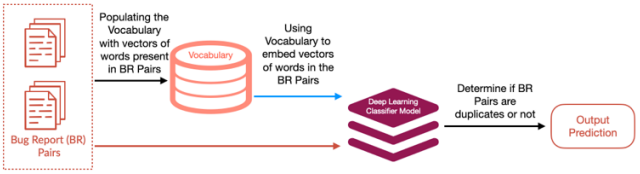
Over the years, many solutions to automatically detect duplicate bug reports have been proposed. Some of the methods employed include the use of methods such as Latent Drinchlett Allocation [6, 7], and Support Vector Machines [9], yet despite the substantial volume of work, the accuracy of existing duplicate detection systems is comparatively low [2]. Most bug reports have the following three features in common: Title, Description and comments. Many easy detection methods like matching the titles, descriptions or keywords in the comments that could eliminate only a specific portion of the duplications. However, these methods do not appear to sufficiently extract and make use of latent features nor adequately capture the semantic relationships between text documents [2], therefore this has left much room for improvement. Some works have been done where the textual similarities and change similarities are studied, and then these similarities are combined to identify duplicate reports [10].

2. DATASET

The bug reports collected from three sources: Mozilla, Eclipse, and Thunderbird have already been aggregated into two groups, duplicate and Non-duplicated. Each line of record has the initial issue's id, title of the initial issue and its description, and the duplicate issue's id, its title, and its description.

3. APPROACH

Most of the existing works have approach very similar to the one shown below. The bug report pair is selected, and the words within the reports are vectorized and is populated in the vocabulary repository and this repository is then used to embed the vector of words in the report pairs which is then sent to the machine learning classifier model to predict if the reports are duplicate or not.



We will begin with data preparation, extracting useful fields from the bug reports and collating the extracted fields and forming a corpus. We would then leverage the word2vec model, capturing semantic regularities from the collated corpus across all three sources. We create pairs of bug-reports to capture the interaction between two bug reports. We would then train and validate the deep learning model. The trained model would give each bug report pair probability, an independent probability. We would then compare this output with the set threshold to classify a pair of bug reports as duplicate or not.

First we created two bug reports by combining the Title and Description features where Bug Report 1 consists of Title1 and Description1 columns, and Bug Report 2 consists of Title2 and Description2 columns. This is based on the simple assumption that both the Description and Title of the Report talk about the same bug, they are just separated for easy reading of the bug reports.

Issue_id	**10954**
Duplicated_issue	440032
Title1	dialup properties needs to be exposed in prefs
Description1	the dialup properties of the profile should be exposed in the prefs panels so the user has an easy way to modify them. the only other alternative would be to make people go to the profile manager to edit them but we dont have that in place either. lets try the prefs panel approach first.
Title2	firefox fails to come back after restart option used e.g. in addon update
Description2	i updated an addon and selected restart to effect it. the browser shutdown the consoel window it was started from echoed nsheaderinfo registerself called nsheaderinfo registerself called and that was the last we saw of firefox. starting it up again produced the popup warning that it was already running. i had to delete the .parentlock file from my profile to get it restarted . start firefox from a console window . addremoveupdate an addon . click on the restart button beyond the repeated warningerror message nothing. no firefox process was left. firefox should have restarted. when i restarted by hand after deleting the .parentlock the requested change to the addon had occurred.

Table 1: A sample of a bug report

	Bug Report 1	Bug Report 2
IssueID	2943	2007
Title	need more info in compare view gkcsqv	dcr compare should be able to see full path gjyrg
Description	when comparing files, it shows just the file names and not the full path of the resource. this is confusing when comparing files with the same name but in different folders or projects. for instance, i tried to compare them. properties file of different projects but there was no indication about which pane was the contents of which file. aw. duplicate of gjyrg itpjuiwinnt dcr compare should be able to see full pathresolvedduplicate.	when i compare two things files or folders with each other it is hard to tell if i have to copy from left to right or right to left if i can't see the full path. this is especially true if they file, or container have the same name. the full paths could be shown as tooltip. test case compares two files with same name with each other. aw. another one gkcsqv itpuiall need more info in compare viewresolvedduplicate.

Table 2: A Sample of Duplicate Bug Report

For both bug reports, we performed sentence preprocessing steps to remove punctuation, case conversion, tokenization, and stop words. Then, preprocessing was done using a custom transform function and Gensim's simple preprocess library. For stopwords removal, we used gensim.parsing.preprocessing.STOPWORDS method. A vocabulary is created with all words from all the

entries of both Report1 and Report 2. Then both the bug reports are encoded into new columns as a zero padded index of words in the vocabulary.

The zero padded words representing each Report1 and Report2 (capped at a max size of 300 words) are fed into two separate input layers where the layer references the word vector (word vector size = 20) for each word index inputted into the model (1 row at a time) thereby creating a matrix representing the bug report where each vector is the embedding of its corresponding word.

These two input layers are then fed into LSTM cells, which are then connected to the layer calculating the Manhattan Distance (similarity function between sentences) between the two reports (forming two parallel Siamese networks). The output obtained is the Manhattan distance which is then used along with an optimal cut-off point to determine if the reports are similar or not.

The signal from the embedding layer is then passed through the LSTM hidden state layers (which is essentially one LSTM layer since they share the weights between the two subnets). The LSTM hidden state is a recurrent architecture that allows the semantic meaning of the words in the sentence to be preserved and compressed into a vector of fixed length which is then used as input for the Manhattan distance layer. The semantic meaning implies the meaning of the words but in the context of the sentence ie- the sequence of words. LSTMs being simple yet highly effective memory cells were used for our project.

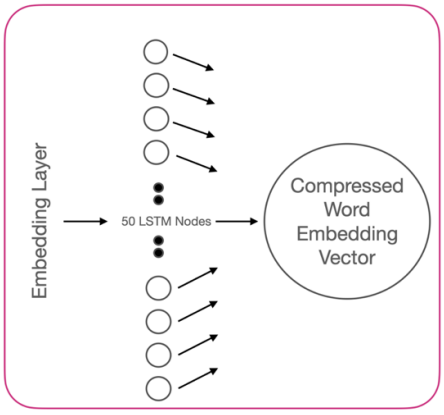


Figure 1: Fully Connected LSTM Layer – Vector Representation

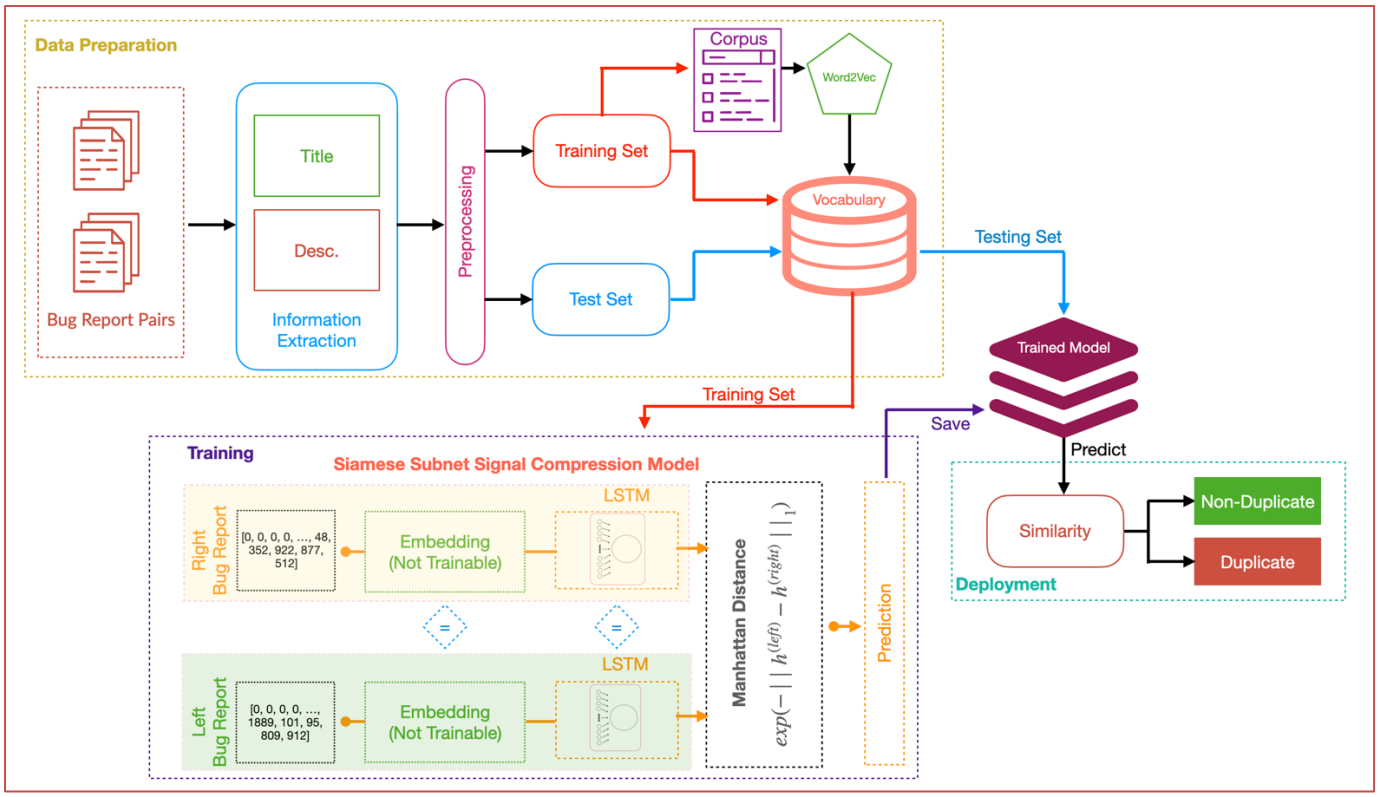


Figure 2: Siamese Subnet Signal Compression Model Approach Overview

We are creating a custom vocabulary of word vectors generated through gensim.models.KeyedVectors, using just the words present in eclipse bug reports. The code goes through all the words in the dataset and builds this vocabulary.

If we are faced with the issue of unseen words present in bug reports, We will use pretrained word embedding vectors which are trained on a Google News dataset (about 100 billion words). This would give us coverage of unseen words which might be visible in bug reports.

4. EVALUATION METRICS

Accuracy indicates the proportion of all correctly predicted bug report pairs to all pairs, and it shows the performance of our model to classify all the bug report pairs correctly.

Recall indicates the proportion of all bug report pairs that are correctly predicted as duplicates to all actual duplicate pairs.

Precision indicates the proportion of all bug report pairs correctly predicted as duplicates to all predicted pairs as duplicate.

F1-Score is a comprehensive evaluation of Recall and Precision, which is the harmonic mean of them. It evaluates if an increase in Precision (or Recall) outweighs a reduction in Recall (or Precision), respectively and provides a balanced view of Precision and Recall.

AUC is the Area Under the Curve of Receiver Operator Characteristic (ROC). ROC can be plotted by computing the True Positive Rates (TPR) and the False Positive Rates (FPR) according to different thresholds. Taking all the FPR values as the horizontal axis and all the TPR values as the vertical axis, the ROC curve can be obtained.

MCC The Matthews correlation coefficient (MCC), is a more reliable statistical rate which produces a high score only if the prediction obtained good results in all of the four confusion matrix categories (true positives, false negatives, true negatives, and false positives), proportionally both to the size of positive elements and the size of negative elements in the dataset.

$$MCC_j = \frac{(TP_j * TN_j) - (FP_j * FN_j)}{\sqrt{(TP_j + FP_j)(FN_j + TN_j)(FP_j + TN_j)(TP_j + FN_j)}}$$

5. EXPERIMENTAL RESULTS

Next we will focuses on answering the following four research questions to demonstrate the experimental results and their implications.

5.1 RQ1: How does the Siamese Subnet Signal Compression (SSSC) Manhattan distance perform overall?

Model	Settings	Level	Precision	Recall	F1-Score	Accuracy	ROC AUC
SSSC Manhattam Distance	Threshold = 0.3	0	0.84	0.96	0.9	0.84	0.744
		1	0.85	0.52	0.65		
	Threshold = 0.5	0	0.81	1	0.89	0.83	0.702
		1	0.99	0.4	0.57		
	Threshold = 0.7	0	0.75	1	0.86	0.76	0.574
		1	1	0.15	0.26		

Table 3: The results of SSSC Manhattan Distance Approach

		Predicted Duplicate Status		
		No	Yes	Total
True Duplicate Status	No	1735	67	1802
	Yes	339	374	713
	Total	2074	441	2515

Table 4: A confusion matrix of SSSC Manhattan Distance Model (Threshold 0.3)

		Predicted Duplicate Status		
		No	Yes	Total
True Duplicate Status	No	1798	4	1802
	Yes	424	289	713
	Total	2222	293	2515

Table 5: A confusion matrix of SSSC Manhattan Distance Model (Threshold 0.5)

		Predicted Duplicate Status		
		No	Yes	Total
True Duplicate Status	No	1802	0	1802
	Yes	607	106	713
	Total	2409	106	2515

Table 6: A confusion matrix of SSSC Manhattan Distance Model (Threshold 0.7)

Table 3-6 shows the results of SSSC Approach. We focus on Accuracy, F1-Score and ROC AUC to evaluate the performance of our approach. We find that the threshold of 0.3 has performed better than the other two thresholds of 0.5 and 0.7. Reviewing the F1-Score, we see that the level of 0 is high at 90% and level of 1 also high compared to other thresholds.

5.2 RQ2: How does the Siamese Subnet Signal Compression (SSSC) Manhattan distance method compare to a Random Forest Classifier?

We will be using Accuracy, Recall, Precision, F1-Score and AUC of both Siamese Subnet Signal Compression Manhattan distance and Random Forest to see if there is a benefit in performing Siamese Subnet Signal Compression Manhattan distance to regular Random Forest.

Model	Settings	Level	Precision	Recall	F1-Score	Accuracy	ROC AUC
Random Forest Classifier	Default Hyperparameters	0	0.78	0.99	0.87	0.79	0.757
		1	0.88	0.24	0.38		
	n_estimators = 200, min_samples_leaf = 10, n_jobs = -1	0	0.77	0.99	0.87	0.77	0.792
		1	0.9	0.17	0.29		

Table 7: The results of Random Forest Classifier

		Predicted Duplicate Status		
		No	Yes	Total
True Duplicate Status	No	6783	85	6868
	Yes	1914	600	2514
	Total	8697	685	9382

Table 8: A confusion matrix of Random Forest Classifier (Default Hyperparameters)

		Predicted Duplicate Status		
		No	Yes	Total
True Duplicate Status	No	6821	47	6868
	Yes	2079	435	2514
	Total	8900	482	9382

Table 9: A confusion matrix of Random Forest Classifier (Hyperparameters: n_estimators = 200, min_samples_leaf = 10, n_jobs = -1)

Table 7-9 shows the results of Random Forest classifier. Comparing the results of the SSSC Manhattan Distance Approach and Random Forest classifier, We find that the RF classifier at default hyperparameters is in parity with the SSSC Manhattan distance approach at threshold of 0.3. But when we set custom hyperparameters, we see that the RF models performs poorer than the SSSC Manhattan Distance Approach.

5.3 RQ3: How does SSSC Manhattan distance compare with Cosine Similarity Classifier?

We will be using the same metrics of Accuracy, Recall, Precision, F1-Score and AUC of both SSSC Manhattan distance and Cosine Similarity Classifier to see if there is a benefit in performing Siamese LSTM to Cosine Similarity Classifier.

Model	Settings	Level	Precision	Recall	F1-Score	Accuracy	ROC AUC
Cosine Similarity Classifier	Threshold = 0.3	0	0.77	1	0.87	0.78	0.589
		1	0.97	0.18	0.30		
	Threshold = 0.5	0	0.74	1	0.85	0.74	0.517
		1	1.00	0.03	0.07		
	Threshold = 0.7	0	0.74	1	0.86	0.74	0.514
		1	1.00	0.03	0.05		

Table 10: The results of Cosine Similarity Classifier

		Predicted Duplicate Status		
		No	Yes	Total
True Duplicate Status	No	34140	82	34222
	Yes	10404	2281	12685
	Total	44544	2363	46907

Table 11: A confusion matrix of Cosine Similarity Classifier (Threshold 0.3)

		Predicted Duplicate Status		
		No	Yes	Total
True Duplicate Status	No	34222	0	34222
	Yes	12555	430	12685
	Total	46477	430	46907

Table 12: A confusion matrix of Cosine Similarity Classifier Model (Threshold 0.5)

		Predicted Duplicate Status		
		No	Yes	Total
True Duplicate Status	No	34222	0	34222
	Yes	12328	357	46907
	Total	46550	357	46907

Table 13: A confusion matrix of Cosine Similarity Classifier Model (Threshold 0.7)

Table 10-13 shows the results of Cosine Similarity classifier. Comparing the SSSC Manhattan Distance Approach and Cosine Similarity classifier results, We find that the Cosine Similarity classifier performs poorer than the SSSC Manhattan Distance Approach in all levels of all three thresholds.

5.4 RQ4: How effective is SSSC Manhattan distance in cross project detection setting??

We will use the same metrics of Accuracy, Recall, Precision, F1-Score and AUC to see how effective SSSC Manhattan distance is in cross project detection setting. For RQ4 we are attempting to test the performance of the SSSC model trained on the bug reports of a specific project (Eclipse) on completely different projects (mozilla and thunderbird). This model is the same SSSC model trained in RQ1, however we have expanded the vocabulary using Google's pretrained word embedding vectors (Google News 300). The purpose of expanding the vocabulary is so that unseen words (words not visible in the Eclipse bug reports) may also be embedded if found in Mozilla or thunderbird bug reports. Based on various iterations we have heuristically set the standard threshold cut-off to 0.1. This approach is experimental and various iterations are further required to confirm the model performance.

Model	Level	Precision	Recall	F1-Score	Accuracy	ROC AUC
Eclipse Data trained model tested on Mozilla Data	0	0.61	0.63	0.62	0.53	0.507
	1	0.4	0.38	0.39		
Eclipse Data trained model tested on Thunderbird Data	0	0.71	0.58	0.64	0.54	0.52
	1	0.33	0.46	0.38		

Table 14: The results of model trained on Eclipse data and tested on Mozilla data

Table 14 shows the results of model trained on Eclipse data and tested on Mozilla data and Thunderbird data, We find that model performed poorly on cross trained model, but there is still a lot of room left to improve.

Since we are dealing with imbalanced classes, we have used MCC to check if one model is performing better than the other. MCC factors in the imbalance in the classes. We believe MCC is the best comparison between the models as compared to AUC and precision metrics.

Model	Pr	Re	F1	AUC	MCC
Optimal Threshold (0.3) SSSC model	0.85	0.52	0.65	0.744	0.57
Optimal Threshold (default hyperparameters) Random Forest model	0.88	0.24	0.38	0.757	0.39
Optimal threshold (0.3) Cosine similarity classifier	0.97	0.18	0.30	0.589	0.36

Table 15: Comparison of Evaluation Metrics

True Positives: It is evident to an uninformed reviewer that both these reports on table 16 might be talking about the same bug. The model is able to identify that getting context about the bugs building on the context of words like "Button", "cut off", "right". The model performed as expected for this data point.

Report 1	Report 2
buttons with characters that start with a line are cut off in right to left buttons laid out in rtl that have a character that starts with a striahg line r h k l etc. are cut off.	buttons have first few pixels cut off in righttoleft bidi if you set the orientation of a composite to righttoleft you will get the first few pixels cut off on a button. steps launch eclipse on hebrew select the debug dialog revert and apply are cut off

Table 16: Example of true positive

False Positives: As seen in the table 17, These pairs of bugs are predicted as similar by the model but are not the same. It is due the similar vector sequences generated for words like "background" and "decorator". This is resulting in model identifying the bugs as the same but in reality they are not.

Report 1	Report 2
decorators menu item doesnt update when i have windows open what we will get more with the new perspective behaviour and in one window i switch cvs decorators on or off windowdecoratorscvs then the the state is not updated in the other window	nested tabs look strange. in .m the tab background picks up the correct background however placing a tabfolder in a tab gives a strange looking result. the tab bar doesnt seem to pick up the background of its underlying widget.

Table 17: Example of false positive

True Negatives are all examples that arent the duplicate bug and are not identified as the duplicate bug either. Fairly straight forward interpretation. Example not required.

False Negatives: Great example where our manhattan distance logic breaks. It is evident that both the Reports talk about the same issue. But due to the difference in length of the text, the model is not able to identify both the Reports as identical. The significant difference in length of the bug reports breaks the text similarity logic. This could be fixed by having rules that do not allow a difference in length above a certain threshold between the two bug reports.

Report 1	Report 2
upgrade to ant .. eclipse should upgrade to the released ant ...	move to ant . i think that if ant . is found to be stable enough it should be part of eclipse .. eclipse is currently using ant .. see bug as of october .beta is out. i imagine that the full release will be out well before the scheduled may release of eclipse .. my primary concern at the moment is because . fixes this bug httpissues.apache.orgbugzillashowbug.cgiid. that bug causes the javac task to fail when using fork and a source directory with a space in it. there are also tons of other bug fixes and new features that i am sure will make lots of other people happy. the sooner ant . get in the better.

Table 18: Example of false negative

6. THREAT TO VALIDITY

This section analyzes several potential aspects that may threaten the validity of our approach and experiments.

Construct Validity: Ideally the data used for modeling would have been labeled by many different groups of developers so that errors in labeling could be distributed approximately uniformly. We make experiments with three datasets from three different projects labelled by different groups of people. Training our model on all available data before deployment on a new project in the real world would hopefully eliminate and systemic labeling errors.

Internal Validity: Relates to potential errors in our experimentation. We have meticulously reviewed our code and are using widely accepted python packages such as Gensim, Word2Vec, Scikit-Learn and Keras.

External Validity: This threat is related to the generalizability of our experiment results. RQ4 tries to accomplish this. If our model can work across projects, then it is more applicable to the real world. It is very unlikely that a growing project will have a large, curated dataset of labeled bug report pairs to train models on. It would be best if we could train on what's available and deploy on any other project large or small.

7. RELATED WORKS:

There has been extensive research in the field of detection of duplicate bug reports. We list a few below.

Hiew et al. [11] use a text similarity approach for detecting duplicate bug reports. Their approach relies on building a model of reports, which groups similar reports into centroids. When a new report arrives, the report is compared to each centroid, looking for occurrences of high similarity. They use the Term Frequency-Inverse Document Frequency (TF-IDF) term weighting scheme to represent the reports as vectors. The detection approach achieves 29% precision and 50% recall.

Weimar et al. [12] built a classifier system using SVM, for incoming bug reports by combining the surface features of the report, textual similarity metrics, and graph clustering algorithms to identify duplicates. They were able to build a classifier system which could reduce development cost by filtering out 8% of duplicate bug reports. The system also allowed at least one report for each real defect to reach developers and spent only 20 seconds per incoming bug report to make a classification.

Few approaches, however, can derive the semantic relationship between bug reports. Deep-learning techniques excel at NLP tasks and can be used as a potential solution. Xu et al. [13] proposed a duplicate bug report detection approach by constructing a Dual-Channel Convolutional Neural Networks (DC-CNN) model. Each bug report is converted to a two-dimensional matrix by employing word embeddings in their approach. They combined the two single-channel matrices of two bug reports into a dual-channel

matrix to represent a bug report pair. CNN can handle multichannel input hence; they use CNN to construct a classifier to capture the relationship among the contextual words by using different sizes of kernels.

The most closest works on this area of study is done by Nguyen et al. [14], Aggarwal et al. [15] and Deshmukh et al. [4]. As we compare our results to those works, we find that our model's performance is in par with the others or better. Table 21 shows the comparison of Accuracy of our model to as that of the related works.

Dataset	Nguyen et al.(2012)	Aggarwal et al.(2017)	Deshmukh et al.(2017)	Proposed System
Eclipse Accuracy	82%	93.18%	72.68%	84%

Table 19: Related Works Accuracy Comparison

8. CONCLUSION

When there are duplicate reports in an error reporting system, It warrents for a system that will be effective in detecting duplicate reports. We proposed a Siamese Subnet Signal Compression Model that uses two connected LSTM layers to predict the duplicates. We tested three threshold of 0.3, 0.5 and 0.7 to find the best performing setup. The threshold of 0.3 has performed better than the other two thresholds of 0.5 and 0.7. Reviewing the F1-Score, we see that the level of 0 is high at 90% and level of 1 also high compared to other thresholds.

Our investigation and preliminary prototype of a SSSC classifier has shown some promising results over baseline techniques like cosine similarity and random forest. We are shocked at how well the Random Forest Classifier did in comparison to our SSSC Manhattan Distance Classifier. The SSSC Manhattan Distance Classifier did outperform the Random Forest with regards to F1 score, but Random Forest outperformed on precision, which is our primary metric for performance evaluation. But, The RF classifier at default hyperparameters is in parity with the SSSC Manhattan distance approach at threshold of 0.3. Comparing to Cosine Similarity Classifier, SSSC Manhattan Distance Approach outperformed Cosine Similarity Classifier, in both levels and in all three thresholds.

Our approach also probes into the feasibility and relevance of using the classification models in detecting cross-project bug classification. We found that our model did not perform well in a cross project detection setting, but definitely very promising. While currently inconclusive, we remain optimistic that better results can be achieved through a much more rigorous study. Cross project functionality is extremely important when it comes to the SSSC Manhattan Distance Classifier's real world application, so it opens pathway for improvement in the future studies.

9. REFERENCES

[1] Jianjun He, Ling Xu, Meng Yan, Xin Xia, Yan Lei. 2020. Duplicate Bug Report Detection Using Dual-Channel Convolutional Neural Networks. ICPC '20, October 5–6, 2020.

[2] Ashima Kukkar, Rajni Mohana, Yugal Kumar, Anand Nayyar, Muhammad Bilal, Kyung-Sup Kwak. 2020. Duplicate Bug Report Detection and Classification System Based on Deep Learning Technique. IEEE Access V.8, 2020.

[3] Qi Xie∗ , Zhiyuan Wen§ , Jieming Zhu‡ , Cuiyun Gao¶ , Zibin Zheng. 2018. Detecting Duplicate Bug Reports with Convolutional Neural Networks. 25th Asia-Pacific Software Engineering Conference (APSEC).

[4] Jayati Deshmukh, Annervaz K M, Sanjay Podder, Shubhashis Sengupta, Neville Dubash. 2017. Towards Accurate Duplicate Bug Retrieval using Deep Learning Techniques. IEEE International Conference on Software Maintenance and Evolution, 2017.

[5] https://en.wikipedia.org/wiki/Software_bug

[6] DavidMBlei, AndrewYNg, MichaelIJordan. 2003. Latentdirichletallocation. Journal of machine Learning research 3, Jan (2003), 993–1022.

[7] YingFu, MengYan, XiaohongZhang, LingXu, DanYang, and JeffreyDKymer. 2015. Automated classification of software change messages by semi-supervised Latent Dirichlet Allocation. Information and Software Technology 57 (2015), 369–377.

[8] Tie-Yan Liu et al. 2009. Learning to rank for information retrieval. Foundations and Trends® in Information Retrieval 3, 3 (2009), 225–331.

[9] WenZhang,TaketoshiYoshida,andXijinTang.2008.Textclassification basedon multi-word with support vector machine. Knowledge-Based Systems 21, 8 (2008), 879–886.

[10] Li ZX, Yu Y, Wang T et al. Detecting duplicate contributions in pull-based model combining textual and change similarities. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 36(1): 191–206 Jan. 2021. DOI 10.1007/s11390-020-9935-1

[11] Lyndon Hiew. 2006. Assisted detection of duplicate bug reports. Ph.D. Dissertation. University of British Columbia.

[12] Nicholas Jalbert and Westley Weimer. 2008. Automated duplicate detection for bug tracking systems. In 2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN). IEEE, 52–61.

[13] Ling Xu, Jianjun he, Meng Yan, Xin Xia and Yan Lei. 2020. Duplicate Bug Report Detection Using Dual-Channel Convolutional Neural Networks.

[14] A.T.Nguyen, T.T.Nguyen, T.N.Nguyen, D.Lo, and C.Sun, "Duplicate bug report detection with a combination of information retrieval and topic modeling," in Proc. 27th IEEE/ACM Int. Conf. Autom. Softw. Eng., 2012, pp. 70–79

[15] K.Aggarwal, F.Timbers, T.Rutgers, A.Hindle, E.Stroulia, and R.Greiner, "Detecting duplicate bug reports with software engineering domain knowl- edge," J. Softw., Evol. Process, vol. 29, no. 3, p. e1821, Mar. 2017.