DSCI-644:
Duplicate Bug Report Detection Using Dual-Channel Convolutional Neural Networks

Team Members : Mayuresh Nene, Prasad Chavan, Ryan Chui

1. **Introduction** :

During software development, there is always an ongoing process of developers, users and other stakeholders, searching for bugs in the software that in some way or the other, hinders the operation and handling of the software. Developers are constantly working on bug fixes and pushing updates to the software to make the software more reliable and as close to being flawless as they can. However, the process is not as simple from the developer's perspective. The process of fixing bugs can take anywhere between just a few hours to a few days. This only scales up as more and more people start reporting bugs in the software.There are situations where multiple people independently report bugs with a title and a description of the bug to the developer. A developer might receive hundreds of bug reports each day, out of which many can be the exact same bug, described using a different set of words. This might not seem like a big problem, but it definitely affects the efficiency of the developers as they spend a lot of time going through the same bugs they are already trying to fix for hours or days. This is not an ideal situation as over a period of time it will decrease the developer's efficiency. However, the developer cannot gauge which bug reports are duplicates unless they spend time going through the details of the incoming bug report. This process takes a lot of time when bug reports are in the hundreds or even thousands at times. This is a problem that can be solved using machine learning.

2. **Open Investigation:**

From a paper using deep neural networks we see that almost all the existing approaches for automatically detecting duplicate bug reports are based on text similarity. Studies have shown that such approaches may become futile in detecting duplicates in bug reports submitted after the just-in-time (JIT) retrieval, which is now a built-in feature of modern BTSs such as Bugzilla
We also see that CNN can be implemented as a better solution for such a problem. Our paper uses Dual-Channel Convolutional Neural Networks (DC-CNN).
Manual Bug Localization is painstaking and very time consuming. Maintenance cost increases and customer satisfaction rate was hampered in the past. Therefore, we want to propose a method that can automatically search for bugs based on similar bugs

that have been fixed or reported before and compare them with the current bug report, in order to avoid duplicates and improve ranking performance.

3. Proposed Solution :

Developers rely on bug reports to fix bugs. Our approach is to implement a CNN model by choosing different kernel sizes to distinguish any duplicate relationships between current bug reports and previous bug reports, and determine the effectiveness of our approach with respect to the model that is chosen. We propose a novel duplicate bug report detection approach by constructing a Dual-Channel Convolutional Neural Networks (DC-CNN) model. In our approach, each bug report is converted to a two-dimensional matrix by employing word embeddings. This matrix is similar to the grayscale image, so we call it a single-channel matrix. Instead of encoding the two reports like it was done in previous studies, we combine the two single-channel matrices of two bug reports into a dual-channel matrix to represent a bug report pair.

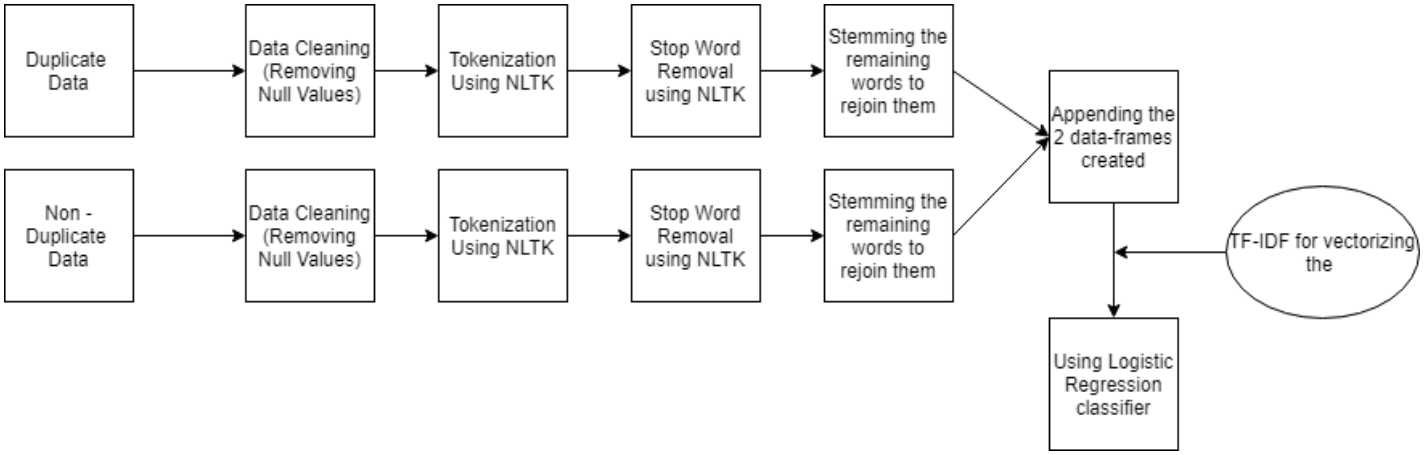
Algorithm and libraries that can be used:

- Pre-processing (Tokenization, lemmatization, Stop word removal, Stemming methods)
- NLTK (Natural Language Toolkit)
- TF-IDFVectorizer (CountVectorizer)
- CNN - Use Tensorflow or Keras for the framework depending on how well we understand
- LSTM, Max-Pooling method.
- Evaluation metrics : AUC Curve, Confusion matrix, F1 score etc..

4. Implementation:

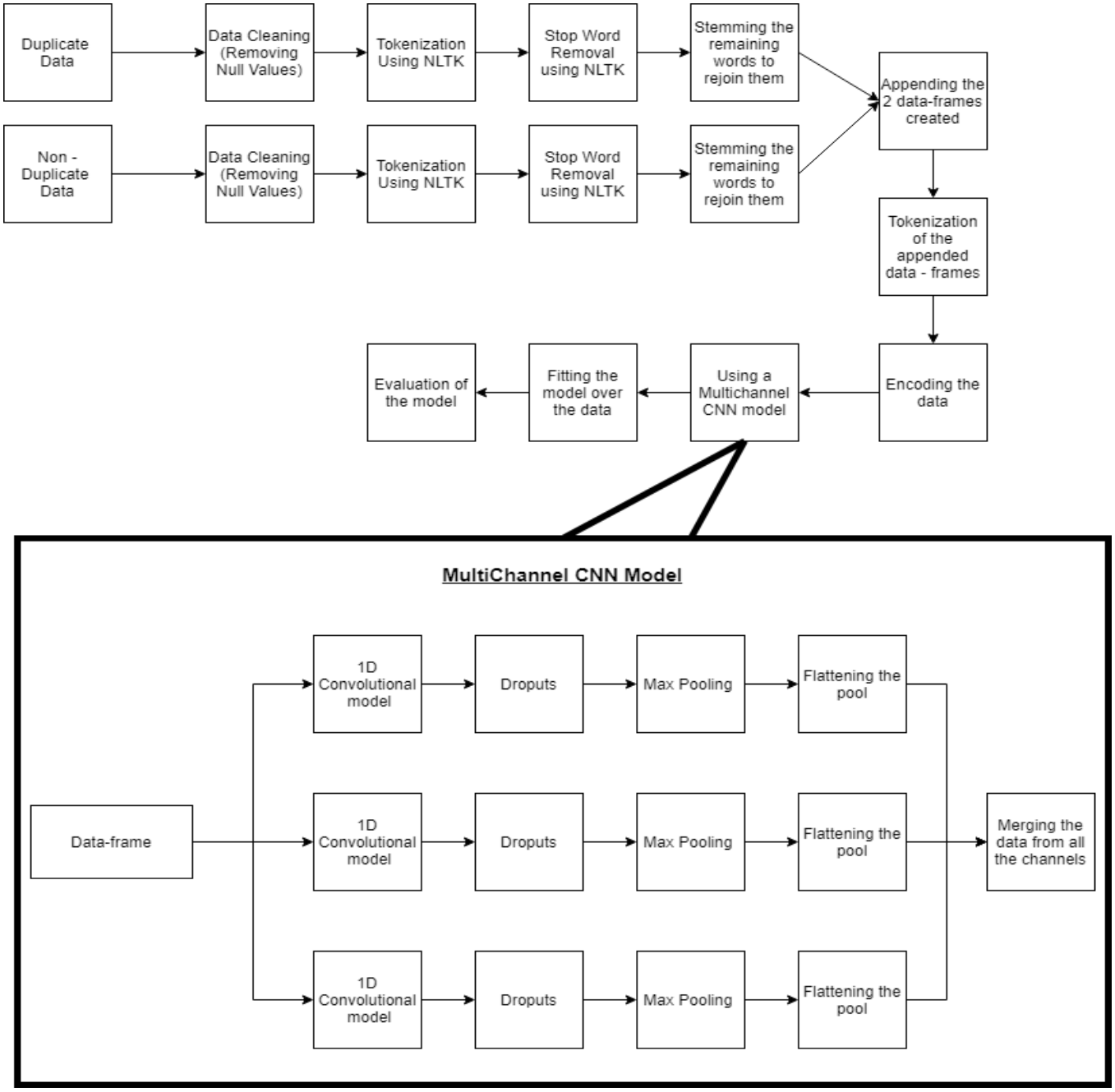
Use Python to extract EU_dup and EU_nondup under the ‘Eclipse’ folder to perform EDA, and chart analysis. Then, we will use TF-IDF, and Countvectorizer methods from the scikit-learn library to transform a given text into a vector to count each word that occurs in the entire text. Next, we will develop a CNN model using Keras, and evaluate a fit model on unseen bug reports data.

5. Study Design:

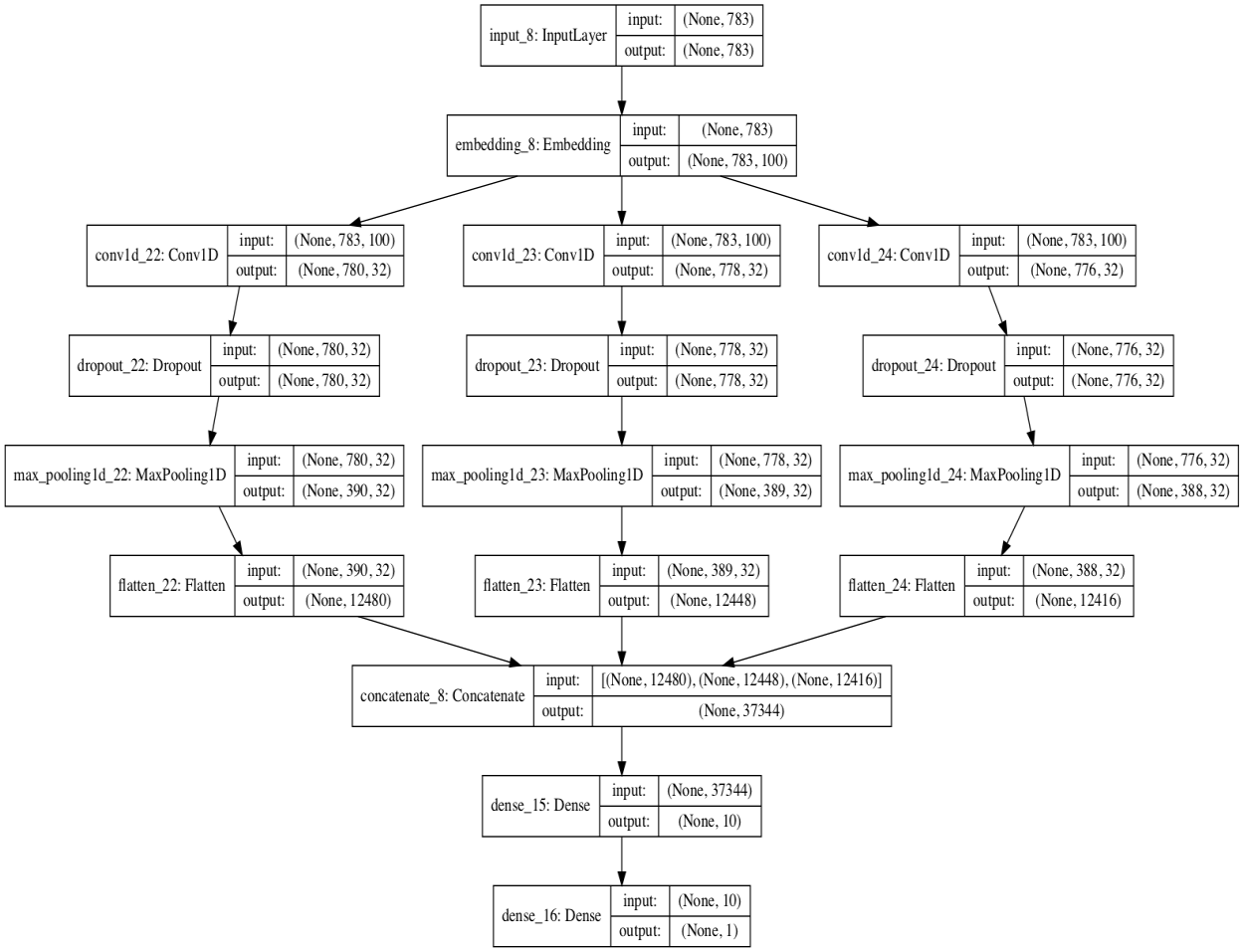


Initial Attempt at solving the given problem

In the first iteration of the solution we see that we receive an f1 score of 0.72.. We cleaned the data using NLTK libraries for cleaning data. We used TF-IDF vectorizer for vectorizing the appended data-frames and used a Logistic Regression classifier.

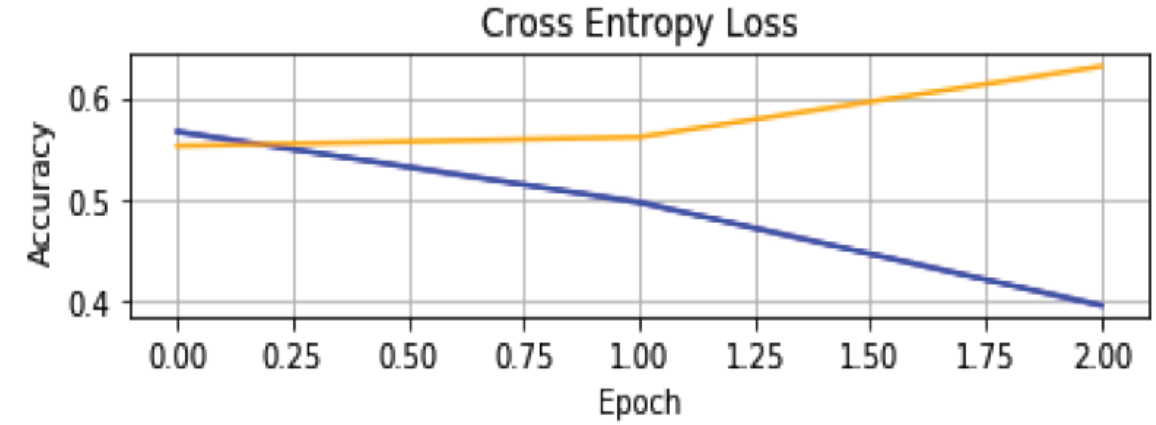


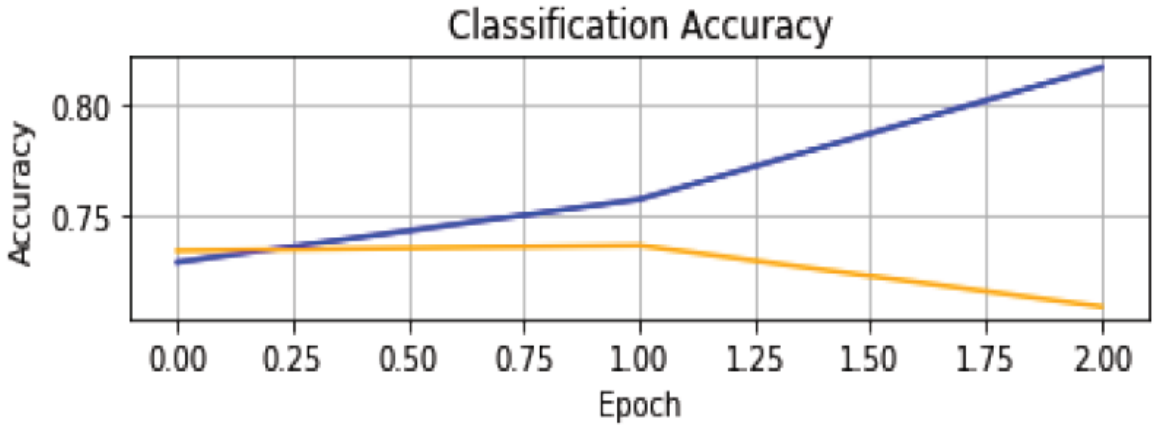
Here we see the pipeline diagram of the solution that we have come up with.
The duplicate and non duplicate data gets pre-processed separately and then is appended after stemming. Once we have the final appended data-frame (final_out) we are able to tokenize it. Once the data is tokenized we pass it through the multi-channel Convolutional Neural Network model. Once the model is fit over the data, we can then evaluate it.



Final attempt at solving the problem using a Multi- Channel CNN

In the final iteration of the solution we see that we receive an f1 score of 0.73





As we can see in the given graphs, for the last solution, after the first epoch the accuracy of the training data starts increasing. We also notice there is a fall in the entropy loss after the first epoch. However we see that for the test data, there is a decline in the accuracy and an increase in the entropy. This needs to be rectified by tuning the parameters appropriately.

6. Experiments:

In this section, we focus on answering a few research questions which arose to demonstrate the experimental results and their implications.

(A). How was the classifier selected?

Compared to the data trees we see that Logistic Regression works better and is more commonly used. We have also noticed that the logistic regression classifier, classifies the data into binary classification (0 or 1), and this gives a high level overview before we compute and evaluate more complex models. This enables us to save time by having an idea whether we are headed in the right direction.

(B). How was the performance in Logistic regression compared to the Multi-Channel CNN model?

In the initial state, we tried implementing Logistic Regression which was implemented with an f1 score of 72%. Our model using Logistic Regression and Tokenizer gives a good accuracy, but the Multi-CNN model seems to perform better . The improvement in the accuracy for the test set is quite a big step up from the test set accuracy for the DC-CNN by about 13%.

Logistic Regression Classification Report Performance

	precision	recall	f1-score	support
0.0	0.76	0.90	0.82	20649
1.0	0.46	0.24	0.32	7495
accuracy			0.72	28144
macro avg	0.61	0.57	0.57	28144
weighted avg	0.68	0.72	0.69	28144

Multi-CNN Classification Report Performance

	precision	recall	f1-score	support
0.0	0.73	1.00	0.85	20649
1.0	0.00	0.00	0.00	7495
accuracy			0.73	28144
macro avg	0.37	0.50	0.42	28144
weighted avg	0.54	0.73	0.62	28144

Above result in multi-cnn model displayed 0's in False Positive and False Negative. F1 score using multi-cnn has slightly outperformed the Logistic regression model. As a rule of thumb, the weighted average of F1 should be used only to compare classifier models, not the global accuracy.

(C). Compared to a generalized CNN model, how does a Multi-Channel CNN model perform?

DC-CNN Model

[13]:
```
## Evaluate the model
loss, accuracy = model.evaluate(trainX, y_train, verbose=False)
print("Training Accuracy: {:.4f}".format(accuracy))
loss, accuracy = model.evaluate(testX, y_test, verbose=False)
print("Testing Accuracy:  {:.4f}".format(accuracy))
```

```
Training Accuracy: 0.8849
Testing Accuracy:  0.7084
```

CNN Model

[68]:
```
## Evaluate the model
loss, accuracy = model.evaluate(x_train, y_train, verbose=False)
print("Training Accuracy: {:.4f}".format(accuracy))
loss, accuracy = model.evaluate(x_test, y_test, verbose=False)
print("Testing Accuracy:  {:.4f}".format(accuracy))
```

```
Training Accuracy: 0.8865
Testing Accuracy:  0.5763
```

The testing accuracy in DC-CNNmodel is higher than the CNN model. This is evidence enough that the DC-CNN has a better feature selection. We also see that if the number of channels increases the entropy in the model increases which leads to a decline in the overall accuracy of the predictions.

(D). Which column, title or description, is more important for the model to predict accurately?

We see after running the model separately for title and description columns, that the description column is more important. The description column contains more keywords which are more weighted which help in improving the accuracy of the predictions. The title contains a few keywords but they cannot be used fully for making firm predictions.

7. References :

1. Bugzilla (https://www.bugzilla.org)
2. Jayati Deshmukh, Sanjay Podder, Shubhashis Sengupta, Neville Dubash, et al. 2017. Towards accurate duplicate bug retrieval using deep learning techniques. In 2017 IEEE International conference on software maintenance and evolution (ICSME). IEEE, 115–124.
3. Guanping Xiao, Xiaoting Du, Yulei Sui, Tao Yue. HINDBR: Heterogeneous Information Network Based Duplicate Bug Report Prediction