DSCI-644:
Project Group 6
Duplicate Bug Report Detection Using Dual-Channel Convolutional Neural Networks

Team Members : Mayuresh Nene, Prasad Chavan, Ryan Chui

1. Introduction:

Modern day software systems are increasing in size and complexity rapidly. Along with this, they also update frequently under the agile development approach. As the software grows in complexities, there are time constraints to be kept in mind while releasing the updates as well. During the development of such software , 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. Proposed Solution:

Developers rely on bug reports to fix bugs. Our approach is to implement a Convolutional Neural Network 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 Multi Channel Convolutional Neural Networks (MC-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 multi-channel matrix to represent a bug report pair.

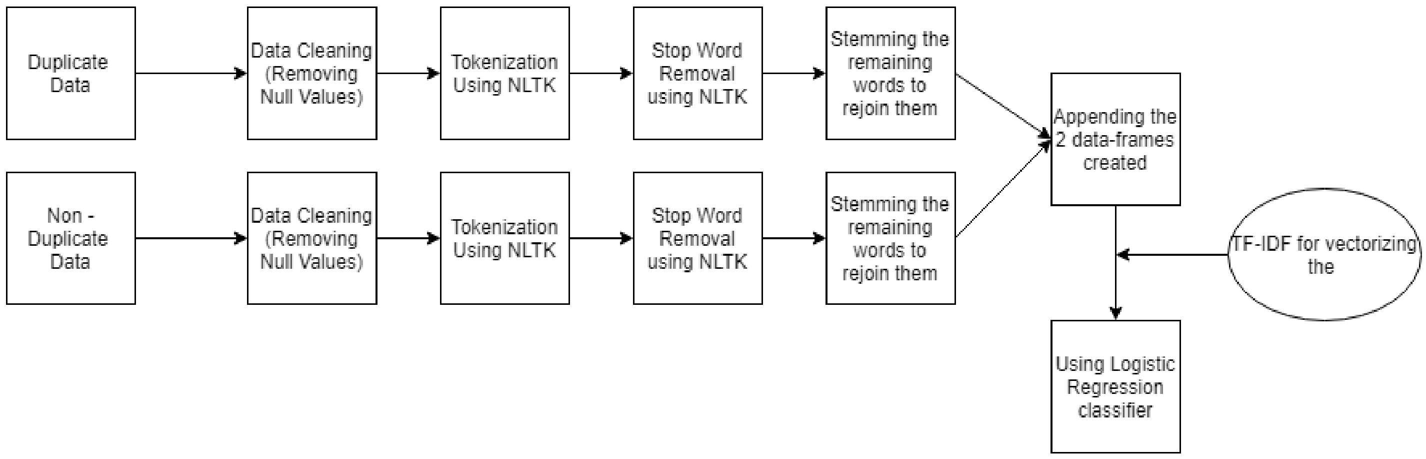
Algorithms and libraries that will be used:

- Pre-processing (Tokenization, lemmatization, Stop word removal, Stemming methods)
- NLTK (Natural Language Toolkit)
- CountVectorizer
- CNN - Use Keras for the framework of the convolutional neural network having the layers:
    - Input Layer
    - Embedding Layer
    - Convolution Layer (Conv1D)
    - Dropout Layer
    - MaxPooling Layer
    - Flattening Layer
- Evaluation metrics : AUC Curve, Confusion matrix, F1 score etc..

3. Implementation:

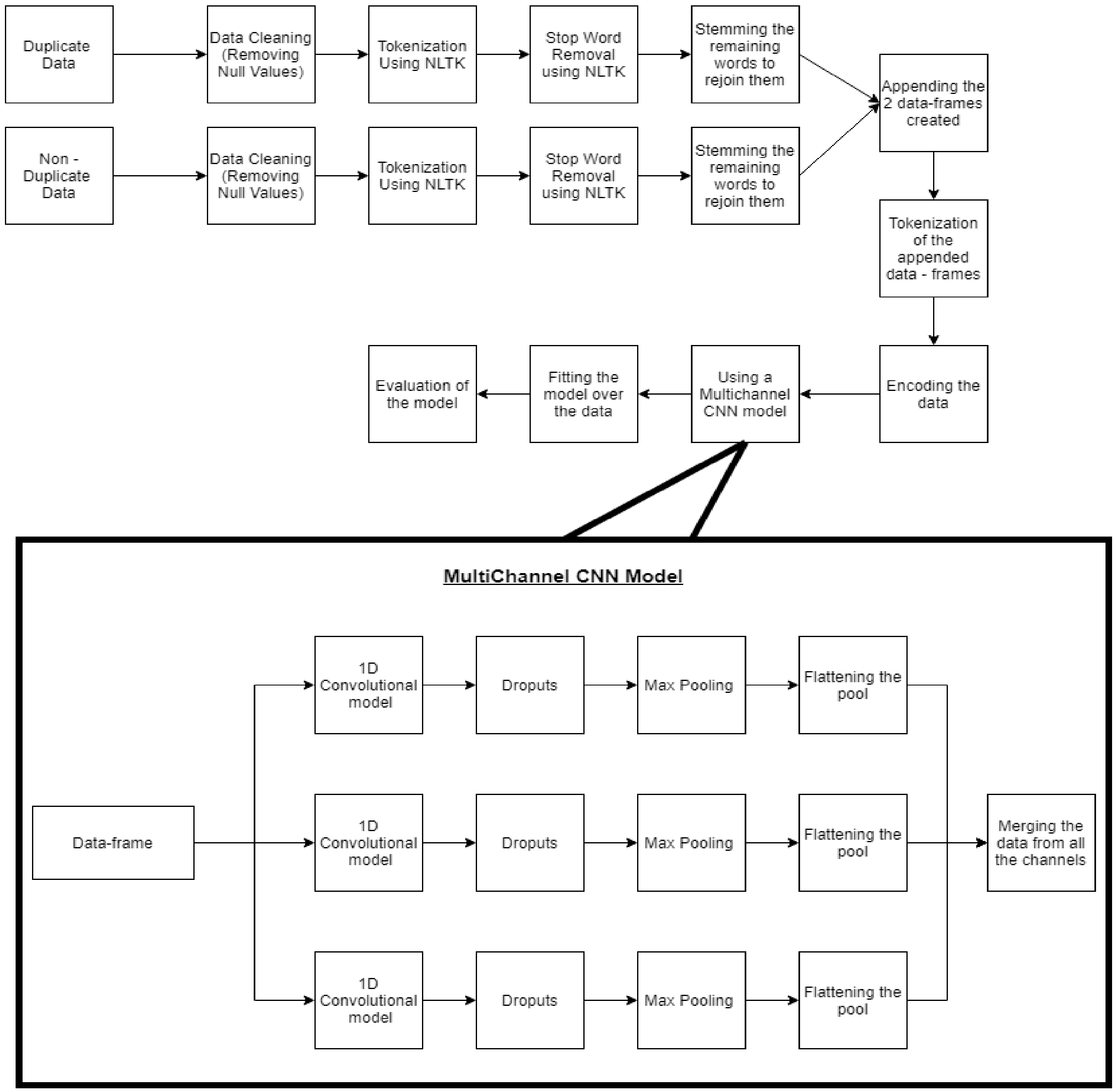
Using Python we extract duplicate and non duplicate data under the ‘Eclipse’ folder to perform Exploratory Data Analysis, and chart analysis. We will have to do pre preprocessing such as stop word removal, case change and lemmatization on the textual data to gain knowledge from the data as well as use the Countvectorizer method 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 Multi Channel Convolutional Neural Network model using Keras framework, and evaluate a fit model on unseen bug reports data.

4. Study Design:


Duplicate Data
Data Cleaning (Removing Null Values)
Tokenization Using NLTK
Stop Word Removal using NLTK
Stemming the remaining words to rejoin them
Non - Duplicate Data
Data Cleaning (Removing Null Values)
Tokenization Using NLTK
Stop Word Removal using NLTK
Stemming the remaining words to rejoin them
Appending the 2 data-frames created
TF-IDF for vectorizing the
Using Logistic Regression classifier

Initial Attempt at solving the given problem

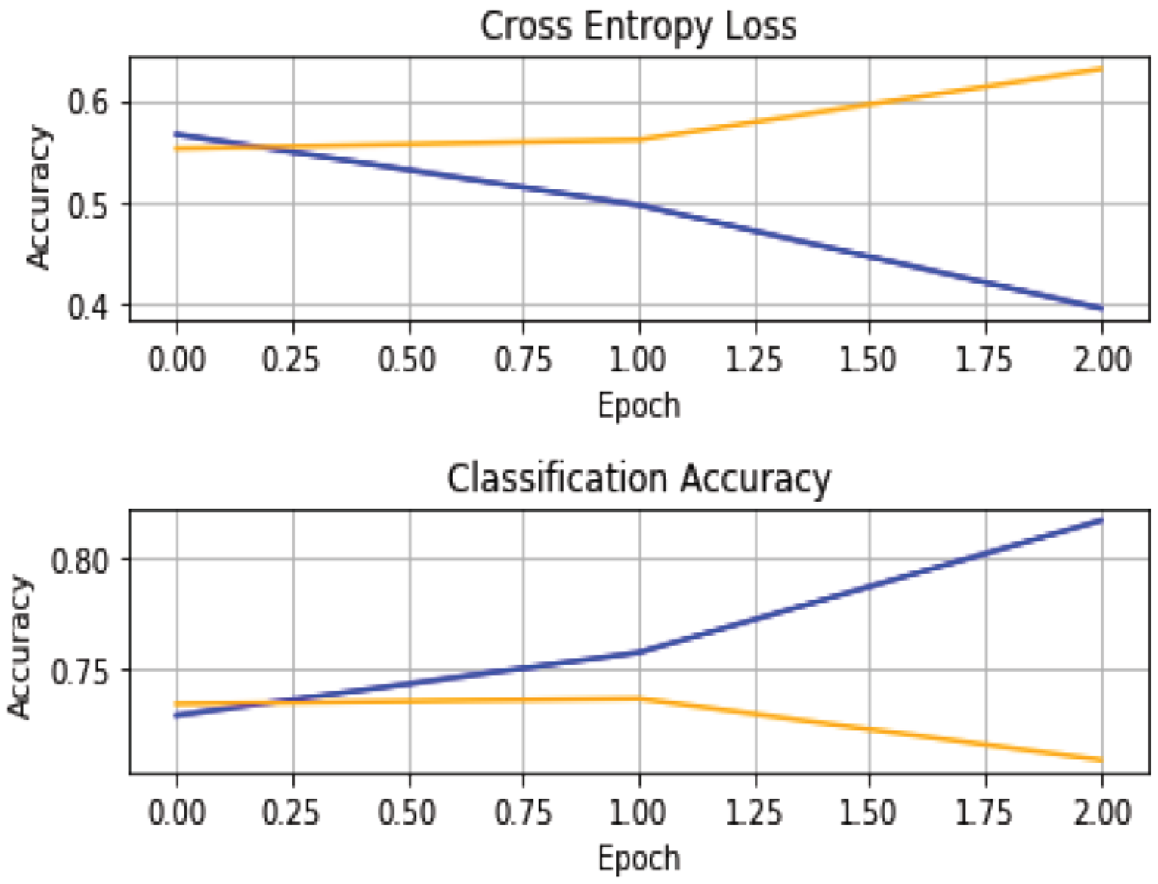
In the first iteration of the solution we see that we receive an f1 score of 0.72..
We cleaned the data using NLTK libraries for cleaning data.
The textual data was converted to lowercase and the stop words were subsequently removed using the NLTK stopwords function.
After the data was cleaned we used TF-IDF vectorizer for vectorizing the appended data-frames
Once tokenized, we used the Logistic Regression classifier as it is one of the best classifiers for the textual weighted data.


Duplicate Data
Data Cleaning (Removing Null Values)
Tokenization Using NLTK
Stop Word Removal using NLTK
Stemming the remaining words to rejoin them
Non - Duplicate Data
Data Cleaning (Removing Null Values)
Tokenization Using NLTK
Stop Word Removal using NLTK
Stemming the remaining words to rejoin them
Appending the 2 data-frames created
Tokenization of the appended data - frames
Encoding the data
Using a Multichannel CNN model
Fitting the model over the data
Evaluation of the model
MultiChannel CNN Model
Data-frame
1D Convolutional model
Droputs
Max Pooling
Flattening the pool
1D Convolutional model
Droputs
Max Pooling
Flattening the pool
1D Convolutional model
Droputs
Max Pooling
Flattening the pool
Merging the data from all the channels

Final Attempt at solving the given problem

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.

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.

5. Experiments:

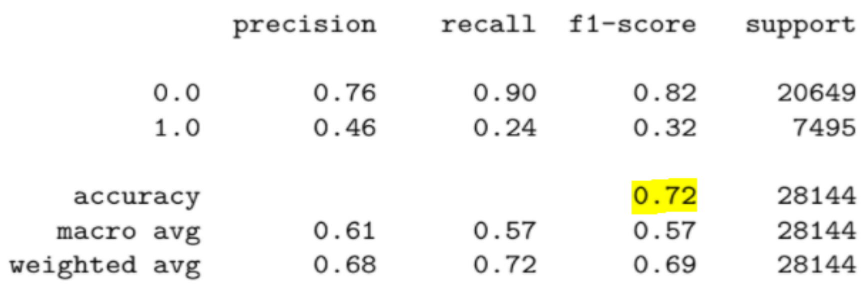
In this section, we focus on answering a few research questions which arose to demonstrate the experimental results and their implications.

(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?

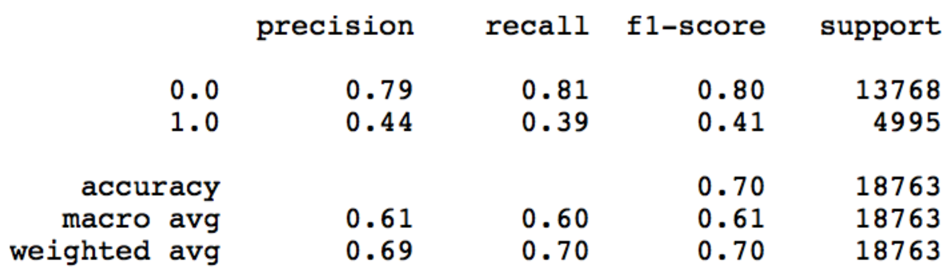
In the initial state, we tried implementing Logistic Regression which was implemented with an f1 score of 72%. Our model using Logistic Regression and Tokenizer gives a good accuracy, but the Multi-CNN model seems to perform better . The improvement in the accuracy for the test set is quite a big step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

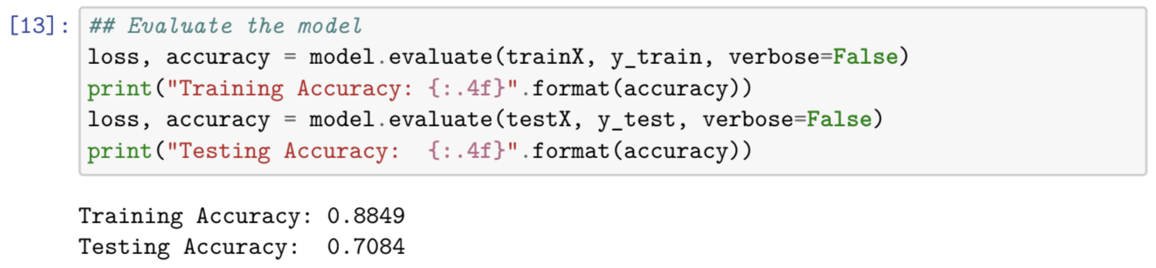


```
              precision    recall  f1-score   support

         0.0       0.79      0.81      0.80     13768
         1.0       0.44      0.39      0.41      4995

    accuracy                           0.70     18763
   macro avg       0.61      0.60      0.61     18763
weighted avg       0.69      0.70      0.70     18763
```

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.

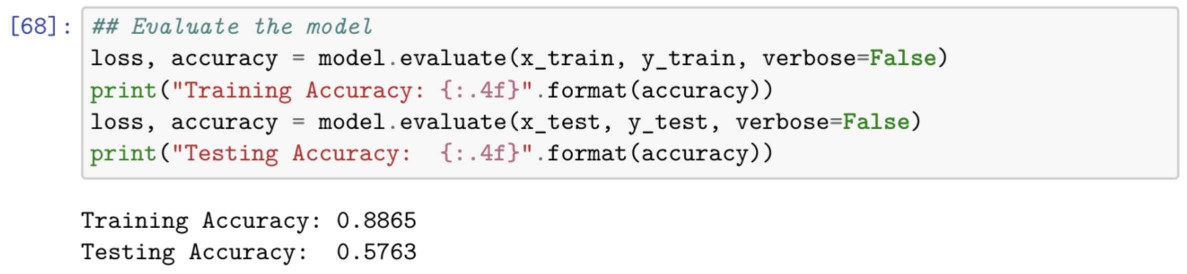
(C). Compared to a generalized CNN model, how does a Multi-Channel CNN model perform?

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.

We see after developing the multi-channel CNN and tuning the hyperparameters, that the accuracy of the model increases while selecting the individual datasets. However while all three data sets are taken into account, we see an increase in the precision and recall of the minority predictions as well.

6. Related Works

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

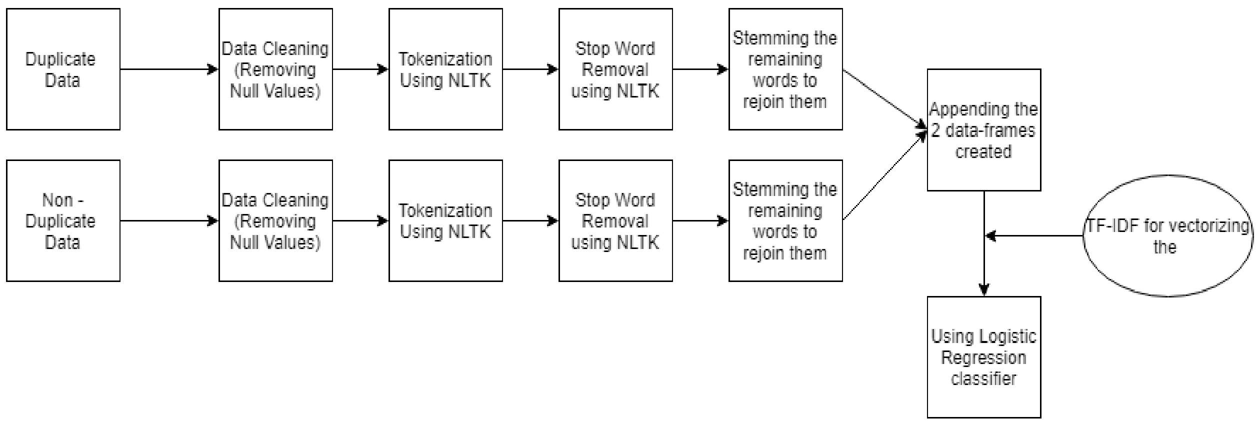
with the current bug report, in order to avoid duplicates and improve ranking performance.

7. Architecture diagrams

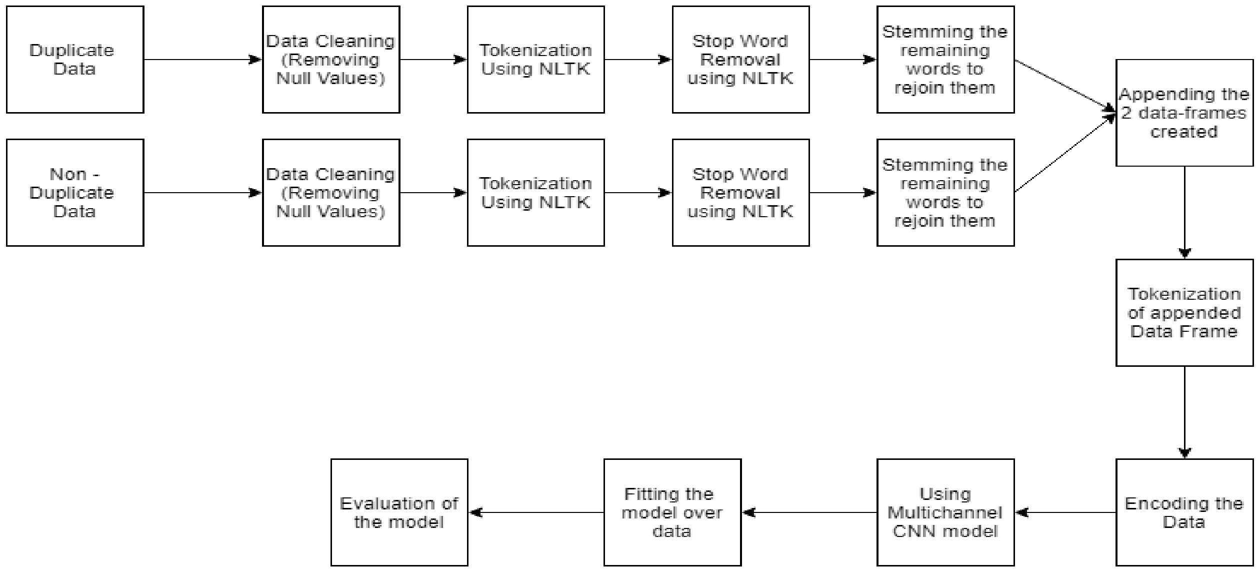
We made two attempts at solving the given question at hand. The first attempt consisted of utilization of the TFIDF vectorizer with the Logistic Regression Classifier. The result was quick to compute but when faced with larger datasets it took more time and the accuracy degraded.
The data cleaning process included the removal of null values. The non null values were then tokenized using NLTK.

For preprocessing the textual data was converted to lowercase. This helps the model in understanding. Once the conversion was completed the stop words were removed from the data. This helps in increasing the accuracy of the model. After the removal of stop words the remaining data went under lemmatization. Lemmatization considers the context of a word which stemming doesn't and hence was implemented in our model

We continued forward with the Multi-Channel CNN model and deployed 3 channels to get the results for our problem. The minority while being detected doesn't display any stellar results. Nonetheless, we can now classify the duplicates with accuracy. This is possible due to the 3 layers deployed in the multi-channel CNN model. The neural network gets better results than a singular CNN and a dual channel CNN as the number of inputs to the model increases which results in better accuracy and precision.


Duplicate Data
Data Cleaning (Removing Null Values)
Tokenization Using NLTK
Stop Word Removal using NLTK
Stemming the remaining words to rejoin them
Non - Duplicate Data
Data Cleaning (Removing Null Values)
Tokenization Using NLTK
Stop Word Removal using NLTK
Stemming the remaining words to rejoin them
Appending the 2 data-frames created
TF-IDF for vectorizing the
Using Logistic Regression classifier

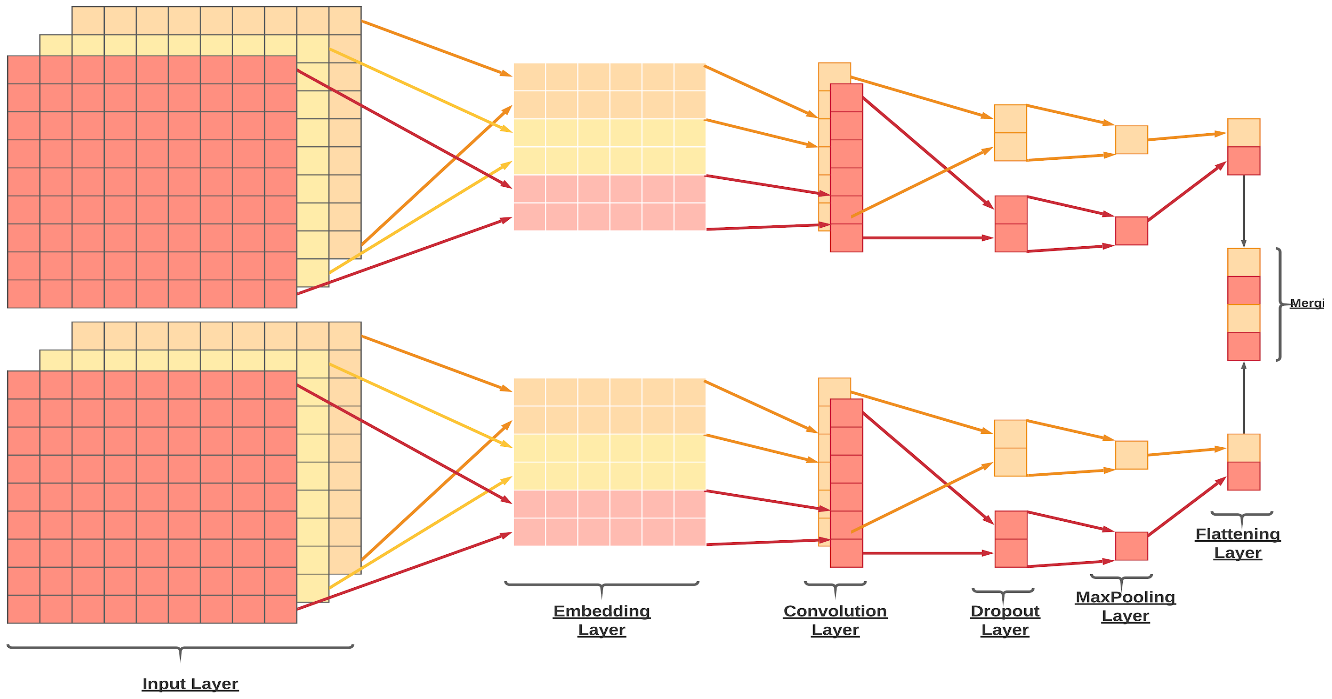
Attempt I



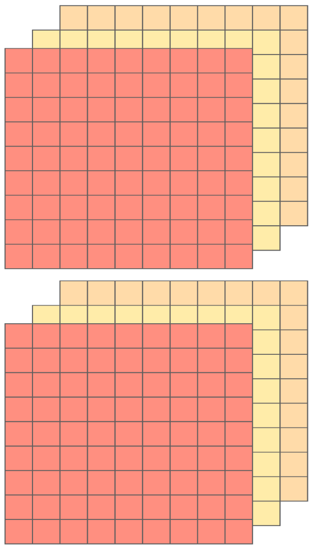
Duplicate Data
Data Cleaning (Removing Null Values)
Tokenization Using NLTK
Stop Word Removal using NLTK
Stemming the remaining words to rejoin them
Non - Duplicate Data
Data Cleaning (Removing Null Values)
Tokenization Using NLTK
Stop Word Removal using NLTK
Stemming the remaining words to rejoin them
Appending the 2 data-frames created
Tokenization of appended Data Frame
Encoding the Data
Using Multichannel CNN model
Fitting the model over data
Evaluation of the model

Attempt II

The MultiChannel Convolution Network is explained further by describing what each layer does in detail.
The diagram below shows a complete model which will be broken down into individual parts while explaining the different layers of the model

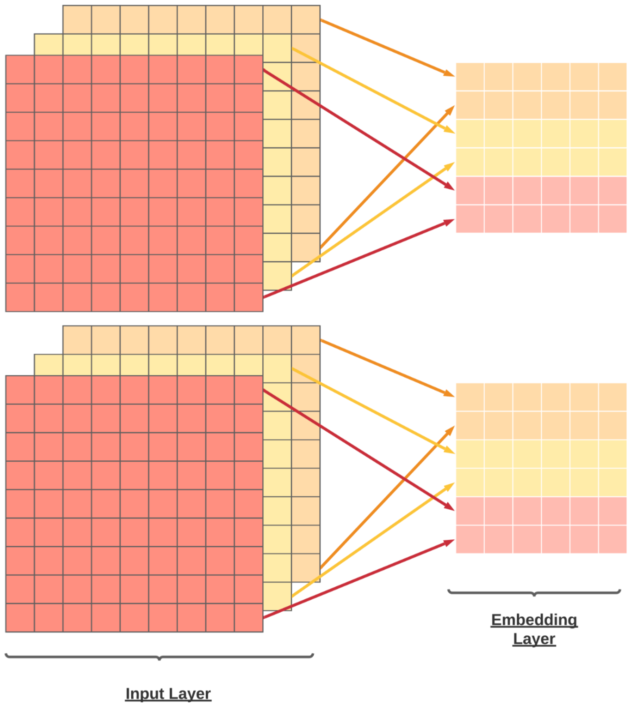


The different layers used in the keras Multi CNN model:



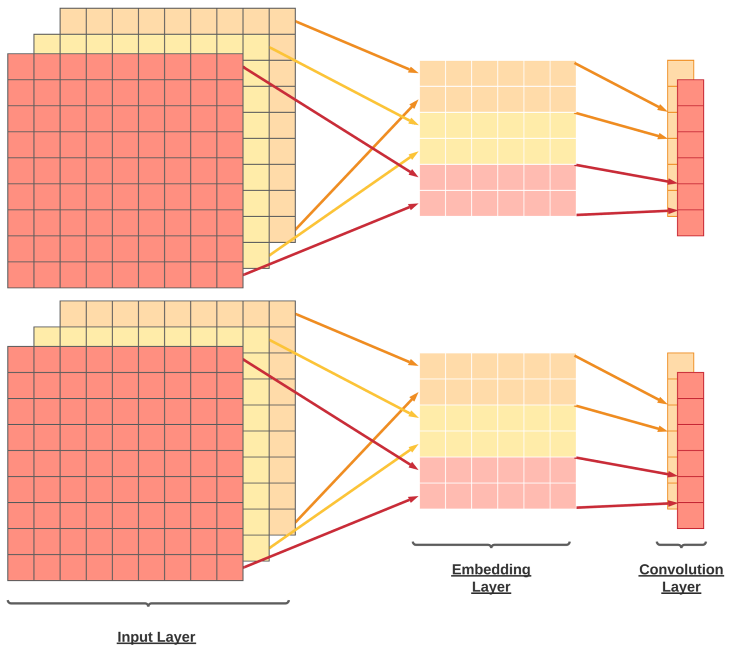
Input layer:

Used to initiate a keras tensor. A tensor has attributes where the input is set as the length of the data. This input layer takes the preprocessed data into account



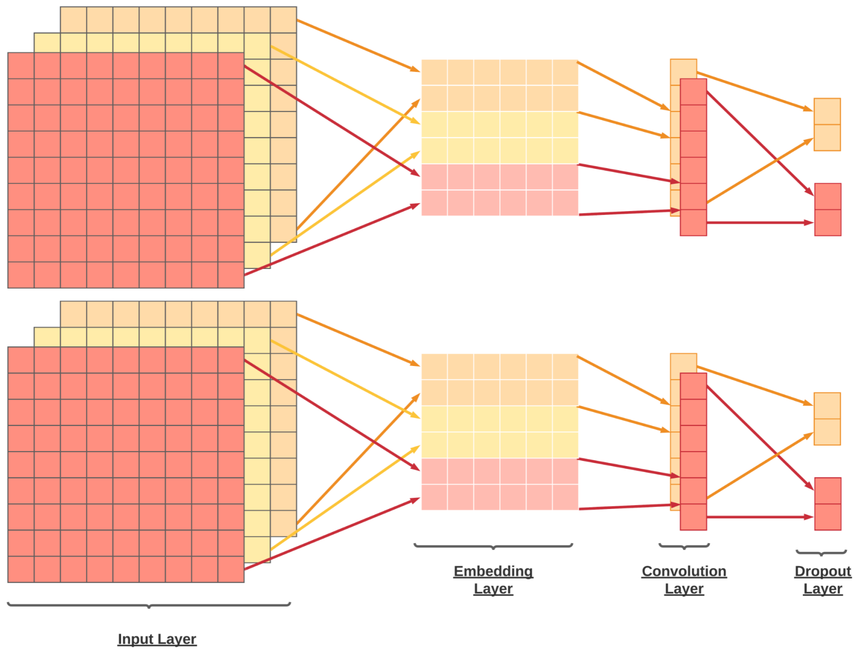
Embedding layer:

Turns positive integers (indexes) into dense vectors of fixed size. Here the input dimensions are given by the size of the vocabulary that was built in the input layer.



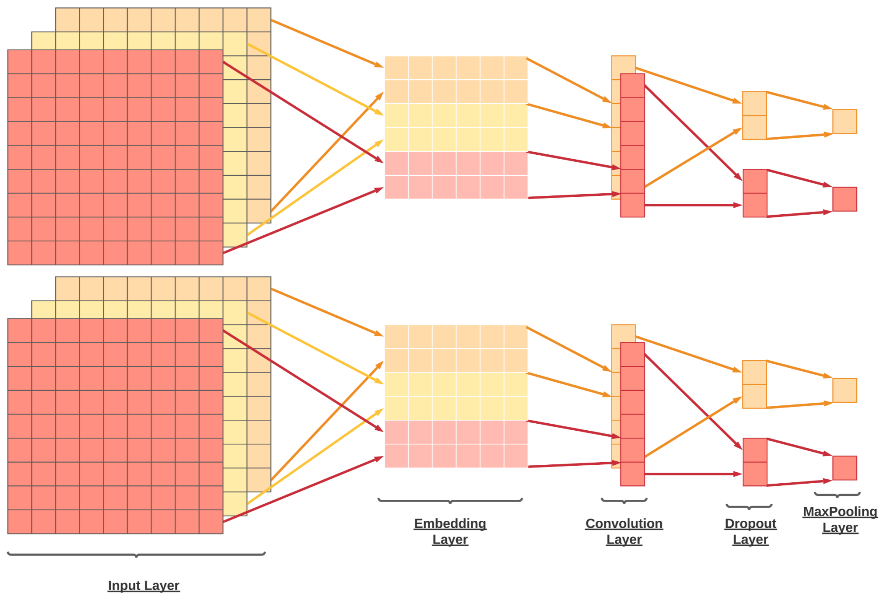
Convolution layer

The CONV1D layer creates the Convolution kernel that is convolved with the input layer. Here the parameters are set for the kernel size and the activation. The activation can be either of the functions; relu, selu or elu cause they are helpful while dealing with linear units.



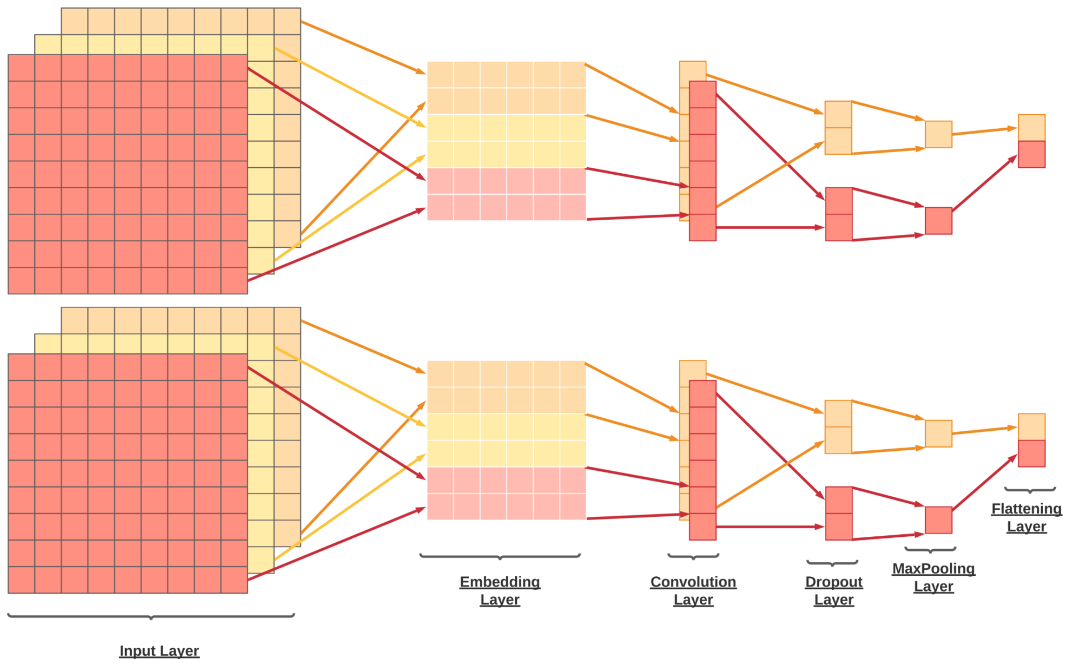
Dropout layer

The Dropout layer randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent overfitting



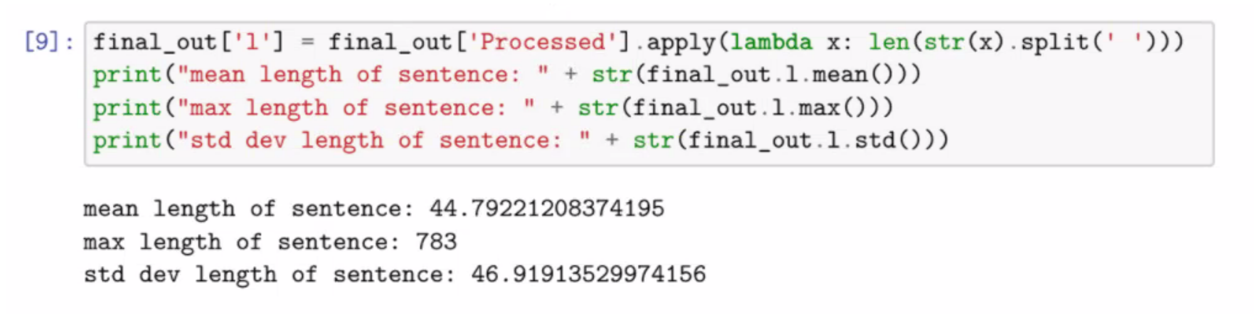
MaxPooling layer

Max pooling layer calculates the largest value in each patch of each feature. The results are down sampled or pooled feature maps that highlight the most present feature in the patch, not the average presence of the feature in the case of average pooling.



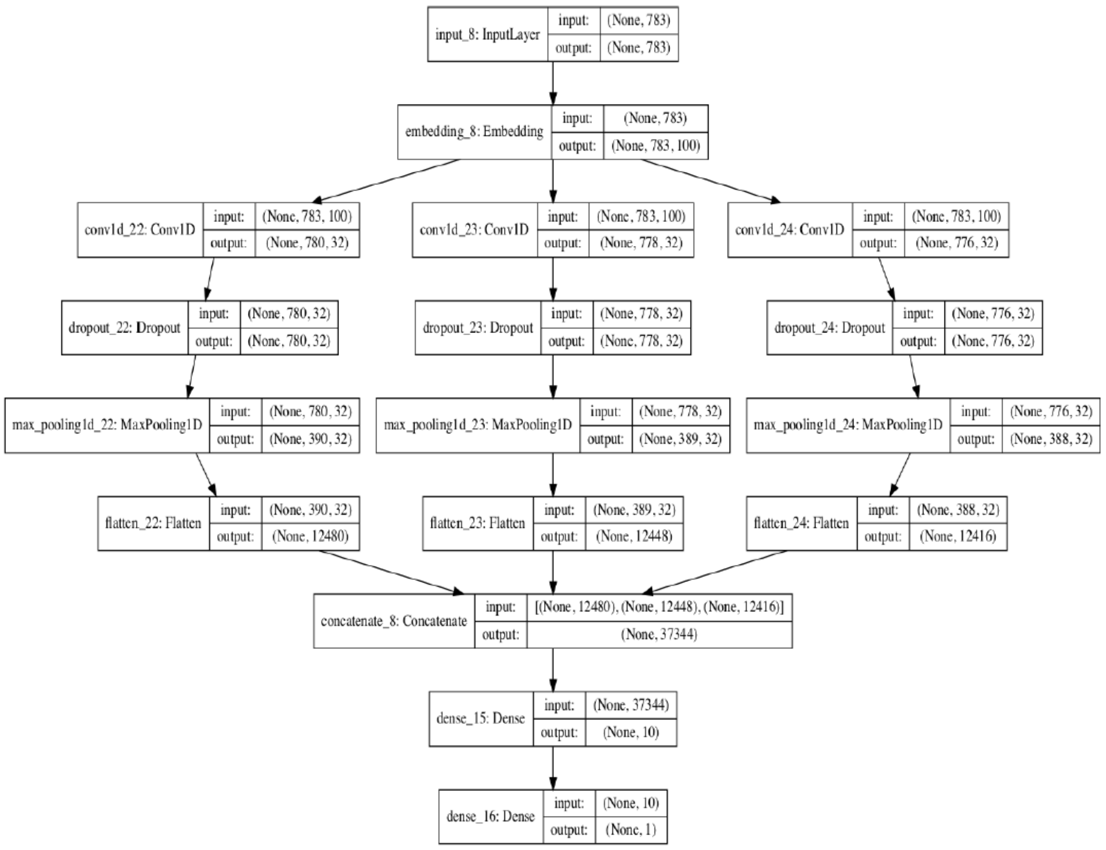
Flatten layer
Flattens the input. Does not affect the batch size. This layer is concerned with the condensation of the data bits received in the max pooling layer.
The model runs the layers using three channels as we have specified it to. Once the channels produce the flattened output, it is merged to predict the label.

Once the model runs we can also plot the linking of the various layers. As seen in the diagram below, we can see how the embedding layer sends the output of the layer to the 3 different channels running the convolution layer.



```
[9]: final_out['l'] = final_out['Processed'].apply(lambda x: len(str(x).split(' ')))
     print("mean length of sentence: " + str(final_out.l.mean()))
     print("max length of sentence: " + str(final_out.l.max()))
     print("std dev length of sentence: " + str(final_out.l.std()))

     mean length of sentence: 44.79221208374195
     max length of sentence: 783
     std dev length of sentence: 46.91913529974156
```

The max length of the sentence is displayed here in the diagram which we found using the code snippet displayed above


input_8: InputLayer | input: (None, 783) | output: (None, 783)
embedding_8: Embedding | input: (None, 783) | output: (None, 783, 100)
conv1d_22: Conv1D | input: (None, 783, 100) | output: (None, 780, 32)
conv1d_23: Conv1D | input: (None, 783, 100) | output: (None, 778, 32)
conv1d_24: Conv1D | input: (None, 783, 100) | output: (None, 776, 32)
dropout_22: Dropout | input: (None, 780, 32) | output: (None, 780, 32)
dropout_23: Dropout | input: (None, 778, 32) | output: (None, 778, 32)
dropout_24: Dropout | input: (None, 776, 32) | output: (None, 776, 32)
max_pooling1d_22: MaxPooling1D | input: (None, 780, 32) | output: (None, 390, 32)
max_pooling1d_23: MaxPooling1D | input: (None, 778, 32) | output: (None, 389, 32)
max_pooling1d_24: MaxPooling1D | input: (None, 776, 32) | output: (None, 388, 32)
flatten_22: Flatten | input: (None, 390, 32) | output: (None, 12480)
flatten_23: Flatten | input: (None, 389, 32) | output: (None, 12448)
flatten_24: Flatten | input: (None, 388, 32) | output: (None, 12416)
concatenate_8: Concatenate | input: [(None, 12480), (None, 12448), (None, 12416)] | output: (None, 37344)
dense_15: Dense | input: (None, 37344) | output: (None, 10)
dense_16: Dense | input: (None, 10) | output: (None, 1)

8. Threats to validity :

In this section, we analyze several potential aspects that may threaten the validity of our approach and experiments.

1. Threats to Internal Validity :

Threats to internal validity refer to errors which can potentially occur in our experimental implementation. We have double checked our code and the results that it has generated. However, there might be specific use-cases where there might be some error, which we did not notice. We have used very commonly implemented libraries (NLTK, Keras, etc) which reduce the chances of critical errors to a minimum. A change in the fields of a typical bug report can also cause errors in the experiment.

2. Threats to External Validity :

Threats to external validity relates to how our experiment will behave in a generalized environment. This refers to the performance of this system in a previously unseen dataset that might have some or the other anomalies or a dataset which is much larger or smaller than the datasets used for the experimentation.

In our experiments, we have used three different datasets. However, we cannot say our experiment can perform well on every other dataset which hasn't been tested. To mitigate this issue or to reduce the chances of this issue to a minimum, we combined the three datasets into one dataset which was much larger in size and experimented on it.

3. Threats to Construct Validity :

The datasets being used are labelled in natural language by different people and so, there is a chance that some of the data is mislabeled. This can lead to an incorrect output in some cases, leading to the evaluation scores being slightly misleading. To avoid this issue as much as possible, three different datasets belonging to three different projects have been used. This can avoid the

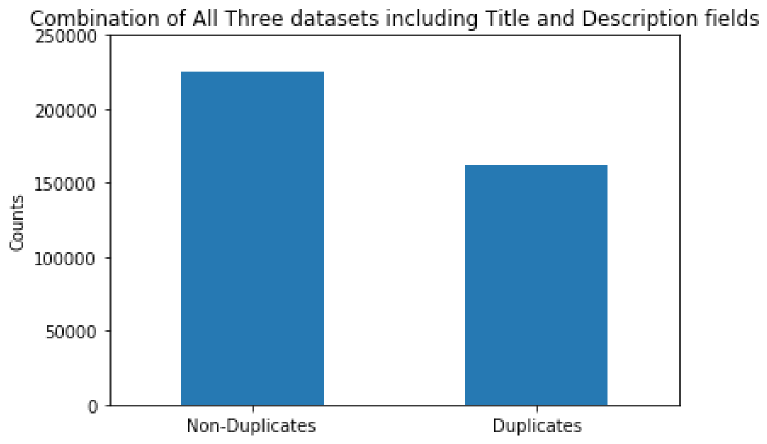
issue because they are written by different people and so, the chances of mislabelling the data decrease. As natural language is used, we can also see some instances where a commit might contain an idiom which would be then taken up as a literal by the model, thereby skewing it's accuracy.
We used Accuracy, Precision, Recall and F1 score, which are commonly used in other studies for binary classification to evaluate our model.

4. **Threats to Conclusion validity :**

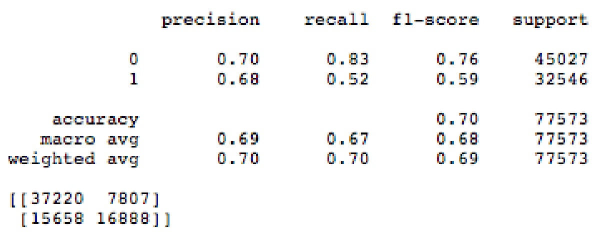
The experiment of selection of classifiers for the first attempt was necessary as there are multiple classifiers to choose from such as Support Vector Machine, Logistic Regression and MultiLayer Perceptron. We saw the results to be in favor of Logistic Regression and continued with it.
The next experiment that we performed was comparison of the Logistic Regression Model. This was done in order to compare the accuracy of the two models. While we saw that the logistic regression model performed better when the CNN model wasn't tuned; after tuning the model, the multi-channel CNN performed better. This was due to the fact that the multi-channel CNN model had 3 layers which were convoluted and the results obtained from therein would yield higher accuracy than the Logistic Regression model.
We ran a normal uni channel CNN to check how it holds up against the multichannel model. To no surprise we see that the multi-channel CNN model performs better than its counterpart.
Next we experimented on the dataset by selecting only description and only title columns. The description column having more number of words made it more significant to the model.

9. Results

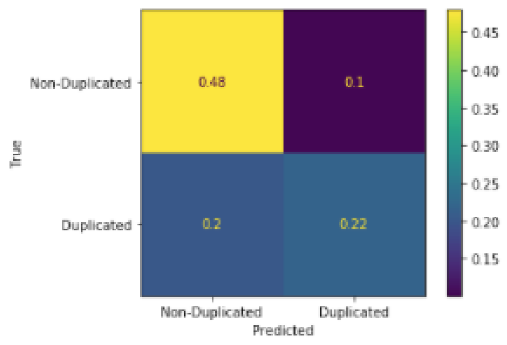
Before we evaluate the performance of the classifiers, let's first look into exploratory of data analysis after aggregating Eclipse, Thunderbird and Mozilla datasets with title and description fields.



Logistic Regression on All three datasets (Title & Description fields)

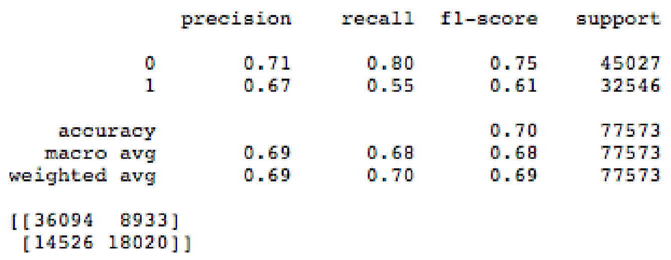


```
              precision    recall  f1-score   support

           0       0.70      0.83      0.76     45027
           1       0.68      0.52      0.59     32546

    accuracy                           0.70     77573
   macro avg       0.69      0.67      0.68     77573
weighted avg       0.70      0.70      0.69     77573

[[37220  7807]
 [15658 16888]]
```

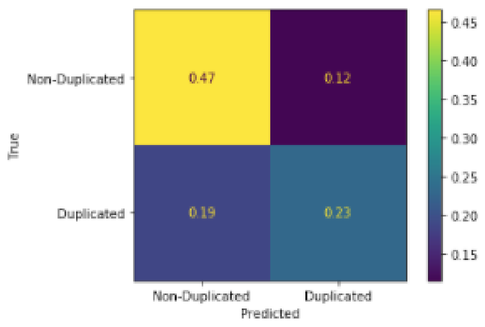


After combining Thunderbird, Eclipse and Mozilla datasets and using 80/20 split rule, the result of the logistic regression above shows that true positive is set to 0.22 suggesting that the algorithm predicts that both fields are correctly labelled as duplicated and they actually are duplicate, while true negative is 0.48 suggesting that the algorithm predicted non-duplicate and they are actually not duplicate. On the contrary, when the Title & Description fields are actually non-duplicated, there is 0.1 probability that it is predicted as duplicated and is an example of false positive, whereas Title & Description fields are duplicated, there is a 0.2 probability that it is predicted as non-duplicate, resulting of False Negative and Type II error. The result of high precision and low recall is suggesting that from this logistic regression, we can trust the predictions for this class, but the model itself is still not too good at detecting it. The ideal solution is achieving high recall and precision from a good classifier. Below is an example of a support vector classifier and its result.

Linear SVC Classifier on All three datasets (Title & Description fields)

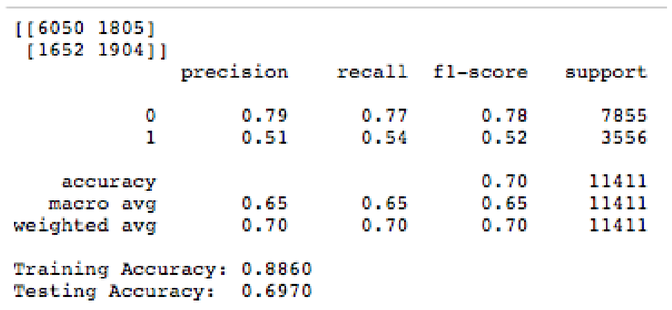


```
              precision    recall  f1-score   support

           0       0.71      0.80      0.75     45027
           1       0.67      0.55      0.61     32546

    accuracy                           0.70     77573
   macro avg       0.69      0.68      0.68     77573
weighted avg       0.69      0.70      0.69     77573

[[36094  8933]
 [14526 18020]]
```

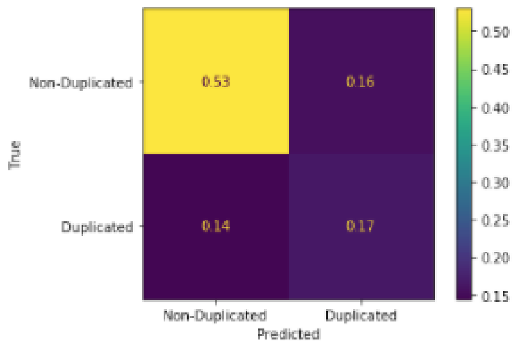


Next, we look into the performance of individual datasets given to us, and below are results of training with the MC-CNN classifier model.

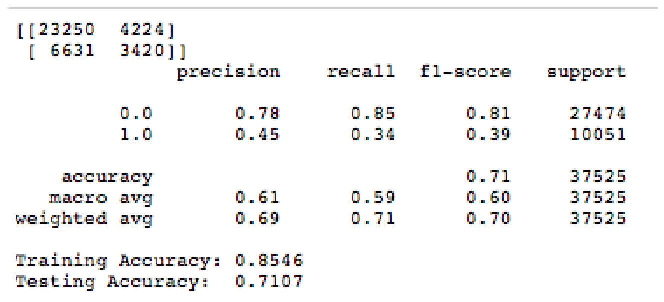
MC-CNN on Thunderbird (Title & Description fields)



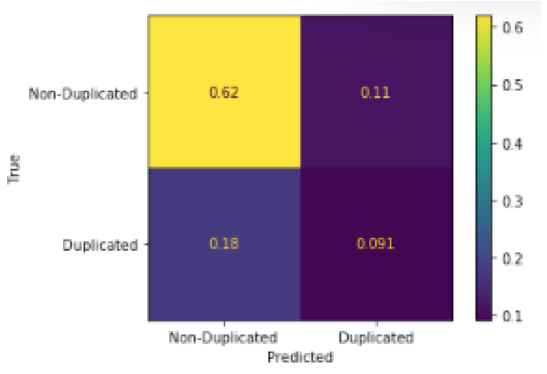
```
[[6050 1805]
 [1652 1904]]
              precision    recall  f1-score   support

           0       0.79      0.77      0.78      7855
           1       0.51      0.54      0.52      3556

    accuracy                           0.70     11411
   macro avg       0.65      0.65      0.65     11411
weighted avg       0.70      0.70      0.70     11411

Training Accuracy: 0.8860
Testing Accuracy:  0.6970
```



MC - CNN on Eclipse (Title & Description fields)

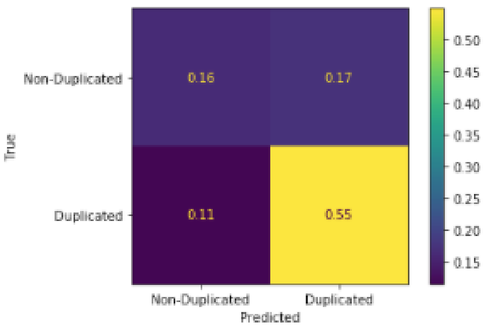


```
[[23250  4224]
 [ 6631  3420]]
              precision    recall  f1-score   support

         0.0       0.78      0.85      0.81     27474
         1.0       0.45      0.34      0.39     10051

    accuracy                           0.71     37525
   macro avg       0.61      0.59      0.60     37525
weighted avg       0.69      0.71      0.70     37525

Training Accuracy: 0.8546
Testing Accuracy:  0.7107
```



MC - CNN on Mozilla (Title & Description fields)

```
[[ 4707  4957]
 [ 3259 15715]]
              precision    recall  f1-score   support

           0       0.59      0.49      0.53      9664
           1       0.76      0.83      0.79     18974

    accuracy                           0.71     28638
   macro avg       0.68      0.66      0.66     28638
weighted avg       0.70      0.71      0.71     28638
```



In order to evaluate an imbalanced classification problem, we can look at the precision and recall score to fully evaluate the overall effectiveness of a model. For example, the model on Thunderbird and Mozilla datasets have high recall and low precision and return many positive results, but most of its predicted labels are incorrect when compared to the ground truth. On the other hand, the model on the Eclipse dataset having high precision but low recall score returns very few results, but most of its predicted labels are correct when compared to the ground-truth. Again, an ideal scenario would be a model with high precision and high recall, meaning it will return many results, with all results labeled correctly. However, in actuality, improving precision typically may reduce recall and vice versa.

MC - CNN on Mozilla Dataset only

Description field

```
[[2671 2252]
 [2302 7094]]
              precision    recall  f1-score   support

           0       0.54      0.54      0.54      4923
           1       0.76      0.76      0.76      9396

    accuracy                           0.68     14319
   macro avg       0.65      0.65      0.65     14319
weighted avg       0.68      0.68      0.68     14319

Training Accuracy: 0.9515
Testing Accuracy:  0.6820
```

Title field

```
[[1990 2933]
 [1564 7832]]
              precision    recall  f1-score   support

           0       0.56      0.40      0.47      4923
           1       0.73      0.83      0.78      9396

    accuracy                           0.69     14319
   macro avg       0.64      0.62      0.62     14319
weighted avg       0.67      0.69      0.67     14319

Training Accuracy: 0.8877
Testing Accuracy:  0.6859
```

In particular, we reach 95% accuracy in training and 68% in testing for the description field. Therefore, while training the data in the description field on duplicates and non-duplicates, we find that high training accuracy and low testing accuracy on the description field result in the model being overfitted to the data. The testing accuracies between description and title fields are nearly identical at 68%. Overfitting could be an issue where a model learns the patterns of a training dataset too well, perfectly explaining the training data set but failing to generalize its predictive power to other sets of data.

Here are the examples of what is being classified as False Positive, False Negative, True Negative, and True Positive among the three datasets.

False Positive:
Issue Number: 421292
we should refer to the icons from org.eclipse.ui.images in our platform ui bundles. see bug
After Stop word removal :
org.eclipse.ui, images, bug platform

Issue number: 422040
i open this bug for the work to use the new images provided by org.eclipse.ui.images.
After Stop word removal :
Bug org.eclipse.ui, images

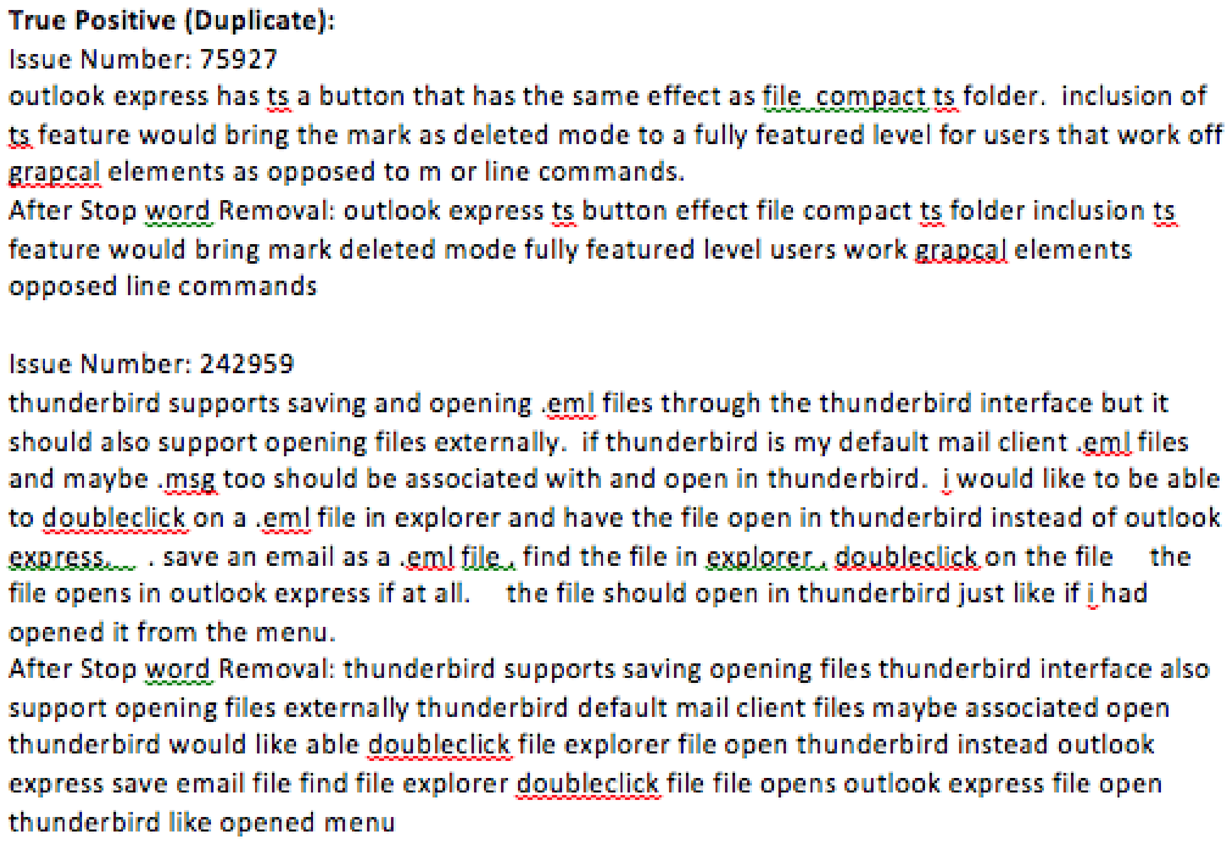
False Positive is the proportion of non-duplicate cases that were incorrectly classified as duplicate. The Neural Network while being smart cannot sometimes differentiate between two columns if they have the same keywords leftover after the stopword removal. In the two github issues mentioned above, we see that after removal of stopwords and some basic data cleaning, the two issue descriptions have a lot of words in common. However, this does not mean that the two issues were talking about the same bug. The model mislabels these descriptions as duplicates due to this occurrence of common words.

False Negative:

Issue number: 344692
firefox ... is not closing down correctly. when i use the windows x to exit fx it exits the fx window but leaves fx running in the background. when i use the file menu and choose exit. i get the same result i am forced to ctrlaltdel to end the process. so i can restart fx. . open fx . use the windows x to close fx . fx window but leaves fx running in the background. . open fx . use the file menu and choose exit. . same as step . closes the fx window but leaves fx running in the background fx should exit leaving no process running in background. extensions talkback ... ie tab .. clicktab .. themes firefox default .

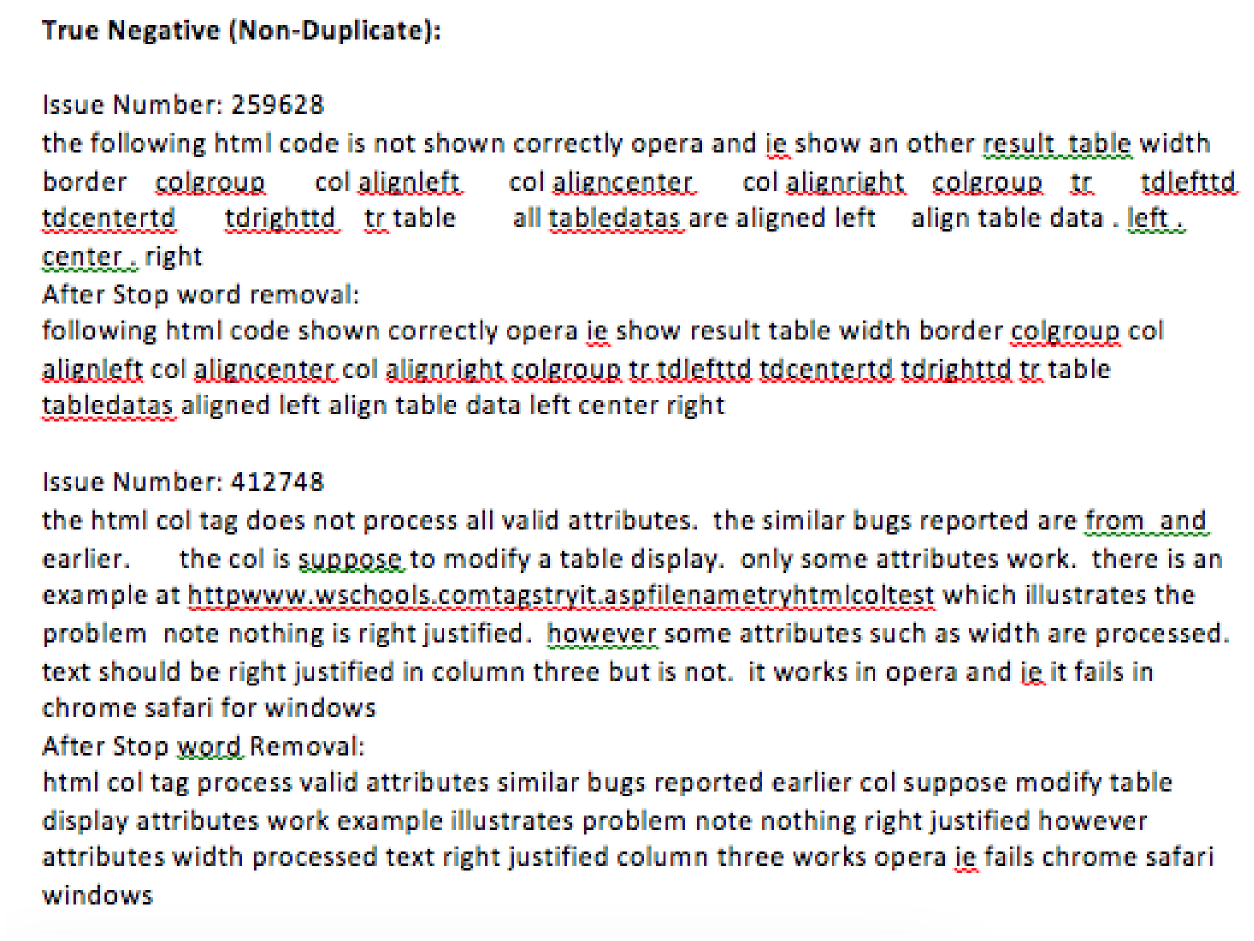
Issue Number: 246942
downloaded the browser rebooted and the program will not launch. no error messages upon installation. i had turned off my antivirus software while installing. after trying to open i can see the process started however the application never launches. .click on desktop icon .start program from task manager .open firefox.exe from program file i get an hour glass for a couple of seconds then absolutely nothing launched the program

False Negative is defined as the proportion of duplicate cases that were incorrectly classified as non-duplicate. Here we see that the mentioned issues were marked as a non-duplicate by the classifier which when seen with the context of the sentences are actually duplicates. This results in the issue being marked as a false negative and is a result of Type II Error. Thus we see that the neural network does make mistakes at times but this can be avoided with the addition of more supporting words in the future extensions.


True Positive (Duplicate):
Issue Number: 75927
outlook express has ts a button that has the same effect as file compact ts folder. inclusion of ts feature would bring the mark as deleted mode to a fully featured level for users that work off grapcal elements as opposed to m or line commands.
After Stop word Removal: outlook express ts button effect file compact ts folder inclusion ts feature would bring mark deleted mode fully featured level users work grapcal elements opposed line commands

Issue Number: 242959
thunderbird supports saving and opening .eml files through the thunderbird interface but it should also support opening files externally. if thunderbird is my default mail client .eml files and maybe .msg too should be associated with and open in thunderbird. i would like to be able to doubleclick on a .eml file in explorer and have the file open in thunderbird instead of outlook express. . save an email as a .eml file . find the file in explorer . doubleclick on the file the file opens in outlook express if at all. the file should open in thunderbird just like if i had opened it from the menu.
After Stop word Removal: thunderbird supports saving opening files thunderbird interface also support opening files externally thunderbird default mail client files maybe associated open thunderbird would like able doubleclick file explorer file open thunderbird instead outlook express save email file find file explorer doubleclick file file opens outlook express file open thunderbird like opened menu

A true positive is an outcome where the model correctly predicts the positive class. Here the sentence is labeled as ‘1’ by the developers and the model correctly predicted the positive class which is a duplicate.



True Negative (Non-Duplicate):

Issue Number: 259628
the following html code is not shown correctly opera and ie show an other result table width border colgroup col alignleft col aligncenter col alignright colgroup tr tdlefttd tdcentertd tdrighttd tr table all tabledatas are aligned left align table data . left . center . right
After Stop word removal:
following html code shown correctly opera ie show result table width border colgroup col alignleft col aligncenter col alignright colgroup tr tdlefttd tdcentertd tdrighttd tr table tabledatas aligned left align table data left center right

Issue Number: 412748
the html col tag does not process all valid attributes. the similar bugs reported are from and earlier. the col is suppose to modify a table display. only some attributes work. there is an example at httpwww.wschools.comtagstryit.aspfilenametryhtmlcoltest which illustrates the problem note nothing is right justified. however some attributes such as width are processed. text should be right justified in column three but is not. it works in opera and ie it fails in chrome safari for windows
After Stop word Removal:
html col tag process valid attributes similar bugs reported earlier col suppose modify table display attributes work example illustrates problem note nothing right justified however attributes width processed text right justified column three works opera ie fails chrome safari windows

A true negative is an outcome where the model correctly predicts the negative class. The sentence here is labeled as ‘0’ by the developers and the model correctly predicted the negative class which is a non-duplicate.

10. Limitations:

1. The model can prove to be computationally expensive as it takes a long time to run. This is partially unavoidable as there is a lot of preprocessing that takes place on the data which leads to good results.

2. Smaller sentences can be removed as they don’t add much meaning to the

bug report.

3. Feature selection can prove to be a bottleneck in cases where performance does not increase beyond a certain threshold. This will indicate that the features that we have aren't enough to correctly classify a majority of the data that we have and some additional features might be required.

4. Class imbalance problems can lead to a bit of a biased result. In this case, we can undersample the class with the majority of the instances or we can oversample the minority class instances. This can potentially lead to better results.

5. If the performance still does not improve much, we can try to get additional features in the bug reports, which can potentially result in higher scores.

11. Extension of experiments

- The number of channels in the multi-channel model can be increased.
- The data sets can be cleaned to avoid the smaller sentences with no value.

12. Conclusion

In this paper we experiment with a Multi-Channel Convolutional Neural Network to predict whether a bug report is duplicate or not. We feed the two bug reports as different channels to the CNN model, which then extracts the hidden features and patterns between the two. We have experimented with three different datasets - Eclipse, Mozilla and Thunderbird and a combination of all three datasets with an approximate total of 122,000 instances of duplicate and non-duplicate data points. We conclude the following from our experiments -

1. Our DC-CNN performs better compared to Logistic Regression or state-of-the-art deep learning models.

2. The combined data set gives us a slightly higher result in terms of precision and F1 score than the individual datasets.

Future work will include cross-project experiments, meaning we will train the model on one dataset and test it on another dataset. In this way we can use knowledge gained from one dataset into another. We can also have more empirical studies that can be used to validate our models by making use of other open-source or industrial projects.

13. 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